QuSpin: a Python Package for Dynamics and Exact Diagonalisation of Quantum Many Body Systems.
Part II: bosons, fermions and higher spins

Phillip Weinberg* and Marin Bukov

Department of Physics, Boston University,
590 Commonwealth Ave., Boston, MA 02215, USA
* weinbe58@bu.edu

May 10, 2017

Abstract

We present a major update to QuSpin, *SciPostPhys.2.1.003*, – an open-source Python package for exact diagonalization and quantum dynamics of boson, fermion and spin many-body systems, supporting the use of various symmetries in 1-dimension and (imaginary) time evolution. We explain how to use the new features of QuSpin using six detailed examples of various complexity: (i)... This easily accessible package can serve various purposes, including educational and cutting-edge experimental and theoretical research.

Contents

How can I make the most out of QuSpin?

- research
- education

How do I use the New Features of QuSpin?

...

In the following, we assume the reader is familiar with examples (i)-(iv) from the original QuSpin paper [1].

The Spectrum of the Transverse Field Ising Model and the Jordan-Wigner Transformation

This example shows how to

- construct fermionic hopping, p-wave pairing and on-site potential terms, and spin$-1/2$ interactions and transverse fields,
- implement periodic and anti-periodic boundary conditions with translation and parity (reflection) symmetries,
- use particle conservation modulo 2, spin inversion, reflection, and translation symmetries,
- handle the default built-in particle conservation and symmetry checks,
- obtain the spectrum of a QuSpin Hamiltonian.

Physics Setup—The transverse field Ising (TFI) chain is paradigmatic in our understanding of quantum phase transitions, since it represents an exactly solvable model[CITE Sachdev]. The Hamiltonian is given my

$$H = \sum_{j=0}^{L-1} -J\sigma_{j+1}^z\sigma_j^z - h\sigma_j^x, \tag{1}$$

where the nearest-neighbour (nn) spin interaction is J, h denotes the transverse field, and σ_j^α are the Pauli spin-1/2 matrices. We use periodic boundary conditions and label the L lattice sites $0,\ldots,L-1$ to conform with Python's convention. This model has gapped, fermionic elementary excitations, and exhibits a phase transition from an antiferromagnet to a paramagnet at $(h/J)_c = 1$ CHECK!. This Hamiltonian possesses the symmetries: magnetisation

conservation, parity (reflection about the centre of the chain), spin inversion, and (many-body) momentum conservation.

In one dimension, the TFI Hamiltonian can be mapped to spinless p-wave superconducting fermions via the Jordan-Wigner (JW) transformation[CITE Sachdev, other paper]:

$$c_i = \prod_{j<i} \sigma_j^z \sigma_i^-, \qquad c_i^\dagger = \prod_{j<i} \sigma_j^z \sigma_i^+, \tag{2}$$

where the fermionic operators satisfy $\{c_i, c_j^\dagger\} = \delta_{ij}$. The Hamiltonian is readily shown to take the form

$$H = \sum_{j=0}^{L-1} J\left(-c_j^\dagger c_{j+1} + c_j c_{j+1}^\dagger\right) + J\left(-c_j^\dagger c_{j+1}^\dagger + c_j c_{j+1}\right) + 2h\left(n_j - \frac{1}{2}\right). \tag{3}$$

In the fermionic representation, the spin zz-interaction maps to nn hopping and a p-wave pairing term with coupling constant J, while the transverse field translates to an on-site potential shift of magnitude h. In view of the QuSpin implementation of the model, we have ordered the terms such that the site index is growing to the right which comes at the cost of a few negative signs due to the fermion statistics. The fermion Hamiltonian posses the symmetries: particle conservation modulo 2, parity and (many-body) "momentum" conservation.

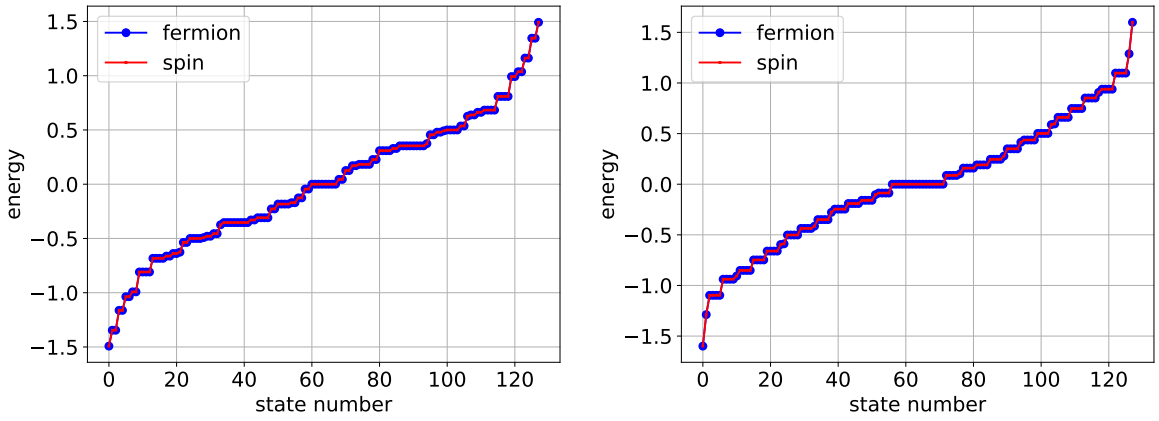
Here, we are interested in studying the spectrum of the TFI model in both the spin and fermion representation. However, if one naively carries out the JW transformation, and computes the spectra of Eqs. (1) and (3), one might be surprised that they do not match exactly. The reason lies in the form boundary condition required to make the JW mapping exact – a subtle issue often left aside in favour of discussing the interesting physics of the TFI model.

Recall that the starting point is the periodic boundary condition imposed on the spin Hamiltonian 1. Due to the symmetries of the spin Hamiltonian (1), we can define the JW transformation on every symmetry sector separately. To make the JW mapping exact, we supplement Eq. (2) with the following boundary conditions: (i) the negative spin-inversion symmetry sector maps to the fermion Hamiltonian (3) with *periodic* boundary conditions (PBC); (ii) the positive spin-inversion symmetry sector maps to the fermion Hamiltonian (3) with *anti-periodic* boundary conditions (APBC). Anti-periodic boundary conditions differ from PBC by a negative sign attached to all coupling constants that cross a single, fixed lattice bond (the bond itself is arbitrary as all bonds are equal for PBC). APBC and PBC are special cases of the more general, twisted boundary conditions, where instead of a negative sign, one attaches a phase factor.

In the following, we show how to compute the spectra of the Hamiltonians in Eqs. (1) and (3) with the correct boundary conditions. Figure 1 shows that they match exactly in both the PBC and APBC cases.

Code Analysis—...

```
from quspin.operators import hamiltonian # Hamiltonians and operators
from quspin.basis import spin_basis_1d, fermion_basis_1d # Hilbert space spin basis
import numpy as np # generic math functions
import matplotlib.pyplot as plt
##### define model parameters #####
L=8 # system size
J=1.0 # spin interaction
```



(a) ETH phase (b) MBL phase

Figure 1: Comparison of the spectra of the spin and fermion representation of the transverse field Ising Hamiltonian in the spin (1) and fermion (3) representations. (a) negative spin inversion/PBC sector and (b) positive spin inversion/APBC sector. The degeneracy is the spectrum is due to the remaining parity and momentum conservations which are not taken into account (see text). The parameters are $J = 1.0$, $h = \sqrt{2}$, and $L = 8$.

```
h=np.sqrt(2) # magnetic field
# loop ober boundary conditions/spin inversion block variable
for PBC in [1,-1]: # periodic or antiperiodic BC
    ##### define spin model
    # site-coupling lists (PBC in both cases)
    J_zz=[[-J,i,(i+1)%L] for i in range(L)] # PBC
    h_field=[[-h,i] for i in range(L)]
    # determine Hilbert space symemtries
    if PBC==1: # include odd spin inversion sector only
        basis_spin = spin_basis_1d(L=L,zblock=-1)
    elif PBC==-1: # include even spin inversion sector only
        basis_spin = spin_basis_1d(L=L,zblock=1)
    # define spin static list
    static_spin =[["zz",J_zz],["x",h_field]]
    # build spin Hamiltonian
    H_spin=hamiltonian(static_spin,[],basis=basis_spin,dtype=np.float64)
    # calculate spin energy levels
    E_spin=H_spin.eigvalsh()
    ##### define fermion model
    # define site-coupling lists for external field
    h_pot=[[2.0*h,i] for i in range(L)]
    if PBC==1: # periodic boundary conditions, include odd particle number subspace only
        # define site-coupling lists (including boudary couplings)
        J_pm=[[-J,i,(i+1)%L] for i in range(L)] # PBC
        J_mp=[[+J,i,(i+1)%L] for i in range(L)] # PBC
        J_pp=[[-J,i,(i+1)%L] for i in range(L)] # PBC
        J_mm=[[+J,i,(i+1)%L] for i in range(L)] # PBC
        # construct fermion basis in the odd particle number subsector
```

```
        basis_fermion = fermion_basis_1d(L=L,Nf=range(1,L+1,2))
    elif PBC==-1: # anti-periodic boundary conditions, include even particle number subspace only
        # define bulk site coupling lists
        J_pm=[[-J,i,i+1] for i in range(L-1)]
        J_mp=[[+J,i,i+1] for i in range(L-1)]
        J_pp=[[-J,i,i+1] for i in range(L-1)]
        J_mm=[[+J,i,i+1] for i in range(L-1)]
        # add boundary coupling between sites (L-1,0)
        J_pm.append([+J,L-1,0]) # APBC
        J_mp.append([-J,L-1,0]) # APBC
        J_pp.append([+J,L-1,0]) # APBC
        J_mm.append([-J,L-1,0]) # APBC
        # construct fermion basis in the even particle number subsector
        basis_fermion = fermion_basis_1d(L=L,Nf=range(0,L+1,2))
    # define fermionic static list
    static_fermion =[["+-",J_pm],["-+",J_mp],["++",J_pp],["--",J_mm],['z',h_pot]]
    # build fermionic Hamiltonian
    H_fermion=hamiltonian(static_fermion,[],basis=basis_fermion,dtype=np.float64,check_pcon=False,check_symm=False)
    # calculate fermionic energy levels
    E_fermion=H_fermion.eigvalsh()
```

The complete code including the lines that produce Fig. ?? is available in Example Code ??.

Integrability Breaking in Higher spin transverse field ising model

This example shows how to:

- construct Hamiltonians for Higher spin operators.
- find ground state of a Hamiltonian
- use obs_vs_time function with costume user defined generator to calculate the expectation value of operators as a function of time.
- use the new functionality of the basis class to calculate the entanglement entropy for higher spin.

Physics Setup In the previous section we introduced the TFIM and showed how one can solve the problem using the Jordan-Wigner transformation. This transformation allows one to solve the Hamiltonian exactly. The fact that this solution exists in deeply connected to the notion of Integrability which has implications of how the system responds to a periodic modulation [CITE]. For non-integrable system when periodic driving, energy is no longer conserved and so generically one would expect that the system will heat up to infinite temperature, while in an integrable system, even though energy is not conserved, there is still an extensive number of other conserved quantities which may be conserved under the drive. If this is the case, then the system will not heat up at long times.

defining two Hamiltonians:

$$H_{zz} = -J_{zz} \sum_{i=0}^{L-1} S_i^z S_{i+1}^z \tag{4}$$

$$H_x = -h_x \sum_{i=0}^{L-1} S_i^x \tag{5}$$

For both $S_i = 1$ and $S_i = 1/2$ operators, We will evolve the ground state of H_{zz} with the following piecewise hamiltonian:

$$H = \{ \tag{6}$$

Repulsively Bound Bosons on Translationally Invariant Ladder

This example shows how to:

- construct Hamiltonians for bosonic systems
- construct ladder Hamiltonians
- using block_tools module to evolve over several symmetry sectors at once using the block_ops class

Fermionic Many-body Localization

This example shows how to:

- construct Hamiltonians for spinful fermions using the tensor_basis class.
- how to use ops_dict class to construct Hamiltonians with varying parameters.
- use new basis functionality to construct simple product states

New Horizons for QuSpin

...

We would much appreciate it if the users could report bugs using the issues forum in the QuSpin online repository.

Acknowledgements

We would like to thank L. Pollet, M. Kolodrubetz, S. Capponi ... for various stimulating discussions and for providing comments on the draft. The authors are pleased to acknowledge that the computational work reported on in this paper was performed on the Shared Computing Cluster which is administered by Boston University's Research Computing Services. The authors also acknowledge the Research Computing Services group for providing consulting support which has contributed to the results reported within this paper. We would also like to thank Github for providing the online resources to help develop and maintain this project.

Funding information This work was supported by ???

Installation Guide in a Few Steps

QuSpin is currently only being supported for Python 2.7 and Python 3.5 and so one must make sure to install this version of Python. We recommend the use of the free package manager Anaconda which installs Python and manages its packages. For a lighter installation, one can use miniconda.

Mac OS X/Linux

To install Anaconda/miniconda all one has to do is execute the installation script with administrative privilege. To do this, open up the terminal and go to the folder containing the downloaded installation file and execute the following command:

```
$ sudo bash <installation_file>
```

You will be prompted to enter your password. Follow the prompts of the installation. We recommend that you allow the installer to prepend the installation directory to your PATH variable which will make sure this installation of Python will be called when executing a Python script in the terminal. If this is not done then you will have to do this manually in your bash profile file:

```
$ export PATH="path_to/anaconda/bin:$PATH"
```

Installing via Anaconda.—Once you have Anaconda/miniconda installed, all you have to do to install QuSpin is to execute the following command into the terminal:

```
$ conda install -c weinbe58 quspin
```

If asked to install new packages just say 'yes'. To keep the code up-to-date, just run this command regularly.

Installing Manually.—Installing the package manually is not recommended unless the above method failed. Note that you must have the Python packages NumPy, SciPy, and Joblib installed before installing QuSpin. Once all the prerequisite packages are installed, one can download the source code from github and then extract the code to whichever directory one desires. Open the terminal and go to the top level directory of the source code and execute:

```
$ python setup.py install --record install_file.txt
```

This will compile the source code and copy it to the installation directory of Python recording the installation location to `install_file.txt`. To update the code, you must first completely remove the current version installed and then install the new code. The `install_file.txt` can be used to remove the package by running:

```
$ cat install_file.txt | xargs rm -rf.
```

Windows

To install Anaconda/miniconda on Windows, download the installer and execute it to install the program. Once Anaconda/miniconda is installed open the conda terminal and do one of the following to install the package:

Installing via Anaconda.—Once you have Anaconda/miniconda installed all you have to do to install QuSpin is to execute the following command into the terminal:

```
> conda install -c weinbe58 quspin
```

If asked to install new packages just say 'yes'. To update the code just run this command regularly.

Installing Manually.—Installing the package manually is not recommended unless the above method failed. Note that you must have NumPy, SciPy, and Joblib installed before installing QuSpin. Once all the prerequisite packages are installed, one can download the source code from github and then extract the code to whichever directory one desires. Open the terminal and go to the top level directory of the source code and then execute:

```
> python setup.py install --record install_file.txt
```

This will compile the source code and copy it to the installation directory of Python and record the installation location to `install_file.txt`. To update the code you must first completely remove the current version installed and then install the new code.

Basic Use of Command Line to Run Python

In this appendix we will review how to use the command line for Windows and OS X/Linux to navigate your computer's folders/directories and run the Python scripts.

Mac OS X/Linux

Some basic commands:

- change directory:

  ```
  $ cd < path_to_directory >
  ```

- list files in current directory:

  ```
  $ ls
  ```

  list files in another directory:

  ```
  $ ls < path_to_directory >
  ```

- make new directory:

```
$ mkdir <path>/< directory_name >
```

- copy file:

```
$ cp < path >/< file_name > < new_path >/< new_file_name >
```

- move file or change file name:

```
$ mv < path >/< file_name > < new_path >/< new_file_name >
```

- remove file:

```
$ rm < path_to_file >/< file_name >
```

Unix also has an auto complete feature if one hits the TAB key. It will complete a word or stop when it matches more than one file/folder name. The current directory is denoted by "." and the directory above is "..". Now, to execute a Python script all one has to do is open your terminal and navigate to the directory which contains the python script. To execute the script just use the following command:

```
$ python script.py
```

It's that simple!

Windows

Some basic commands:

- change directory:

```
> cd < path_to_directory >
```

- list files in current directory:

```
> dir
```

  list files in another directory:

```
> dir < path_to_directory >
```

- make new directory:

```
> mkdir <path>\< directory_name >
```

- copy file:

```
> copy < path >\< file_name > < new_path >\< new_file_name >
```

- move file or change file name:

```
> move < path >\< file_name > < new_path >\< new_file_name >
```

- remove file:

```
> erase < path >\< file_name >
```

Windows also has a auto complete feature using the TAB key but instead of stopping when there multiple files/folders with the same name, it will complete it with the first file alphabetically. The current directory is denoted by "." and the directory above is "..".

Execute Python Script (any operating system)

To execute a Python script all one has to do is open up a terminal and navigate to the directory which contains the Python script. Python can be recognised by the extension `.py`. To execute the script just use the following command:

```
python script.py
```

It's that simple!

Package Documentation

In QuSpin quantum many-body operators are represented as matrices. The computation of these matrices are done through custom code written in Cython. Cython is an optimizing static compiler which takes code written in a syntax similar to Python, and compiles it into a highly efficient C/C++ shared library. These libraries are then easily interfaced with Python, but can run orders of magnitude faster than pure Python code [2]. The matrices are stored in a sparse matrix format using the sparse matrix library of SciPy [3]. This allows QuSpin to easily interface with mature Python packages, such as NumPy, SciPy, any many others. These packages provide reliable state-of-the-art tools for scientific computation as well as support from the Python community to regularly improve and update them [4, 5, 6, 3]. Moreover, we have included specific functionality in QuSpin which uses NumPy and SciPy to do many desired calculations common to ED studies, while making sure the user only has to call a few NumPy or SciPy functions directly. The complete up-to-date documentation for the package is available online under:

https://github.com/weinbe58/QuSpin/#quspin

Complete Example Codes

In this appendix, we give the complete python scripts for the dix examples discussed in Sec. 2. In case the reader has trouble with the TAB spaces when copying from the code environments below, the scripts can be downloaded from github at:

https://github.com/weinbe58/QuSpin/tree/master/examples

References

[1] P. Weinberg and M. Bukov, *QuSpin: a Python Package for Dynamics and Exact Diagonalisation of Quantum Many Body Systems part I: spin chains*, SciPost Phys. **2**, 003 (2017), doi:10.21468/SciPostPhys.2.1.003.

[2] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn and K. Smith, *Cython: The best of both worlds*, Computing in Science & Engineering **13**(2), 31 (2011), doi:http://dx.doi.org/10.1109/MCSE.2010.118.

[3] E. Jones, T. Oliphant, P. Peterson *et al.*, *SciPy: Open source scientific tools for Python* (2001–).

[4] S. v. d. Walt, S. C. Colbert and G. Varoquaux, *The numpy array: A structure for efficient numerical computation*, Computing in Science & Engineering **13**(2), 22 (2011), doi:http://dx.doi.org/10.1109/MCSE.2011.37.

[5] T. E. Oliphant, *Python for scientific computing*, Computing in Science & Engineering **9**(3), 10 (2007), doi:http://dx.doi.org/10.1109/MCSE.2007.58.

[6] K. J. Millman and M. Aivazis, *Python for scientists and engineers*, Computing in Science & Engineering **13**(2), 9 (2011), doi:http://dx.doi.org/10.1109/MCSE.2011.36.