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 7, 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

1 What can QuSpin be Useful for?

Understanding the physics of many-body quantum condensed matter systems often involves a great deal of numerical simulations, be it to gain intuition about the complicated problem of interest, or because the latter does not admit an analytical solution which can be expressed in a closed form. This motivated the development of open-source packages [CITE], the purpose of which is to facilitate the study of condensed matter systems without the need to understand and implement complicated numerical methods which required years to develop. Here, we report a major upgrade to QuSpin [1] – a Python library for exact diagonalisation (ED) and simulation of the dynamics of quantum many-body systems.

Although ED methods are vastly outperformed by more sophisticated numerical techniques in the study of equilibrium systems [CITE], as of present date ED remains essential for most dynamical non-equilibrium problems. The reason for this often times relies on the fact that the underlying physics of these problems cannot be explained without taking into consideration the contribution from high-energy states excited during the evolution. Some prominent examples of such problems include the study of many-body localisation (MBL) [CITE], the Eigenstate Thermalisation hypothesis [CITE], quantum quench dynamics [CITE], periodically-driven systems [CITE], adiabatic and counter-diabatic state preparation, applications of Machine Learning to non-equilibrium physics [CITE], and many more **did I forget smth important?**.

It is, thus, arguably useful to have a toolbox available which allows one to quickly simulate and study these and related nonequilibrium problems. As such, QuSpin offers easy access to performing numerical simulations, which can facilitate the development and inspiration of new ideas and the discovery of novel phenomena, eliminating the cost of spending time to develop a reliable code. Besides theorists, the new version of QuSpin will hopefully even prove valuable to experimentalists working on problems containing dynamical setups, as it can help students and researchers focus on making the experiment run, rather than worrying about writing the supporting simulation code. Last but not least, with the computational processing power growing higher than ever before, the role played by simulations for theoretical research becomes increasingly more important too. It can, therefore, be expected that in the near future quantum simulations become an integral part of the standard physics university curriculum, and having easily accessible toolboxes, such as QuSpin, is one of the required prerequisites.

2 How do I use the New Features of QuSpin?

New in QuSpin 2.0, we have added the following features and toolboxes:

- ...

Installing QuSpin is quick and efficient; just follow the steps outlined in App. A.

Before we carry one, we refer the interested reader to examples (i)-(iv) from the original QuSpin paper [1]. The examples below focus predominantly on the newly introduced features, and are thus to be considered complementary. We emphasize that, since they serve the purpose of explaining how to use QuSpin, for the sake of brevity we shall not discuss the interesting physics related to the interpretation of the results.

2.1 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.

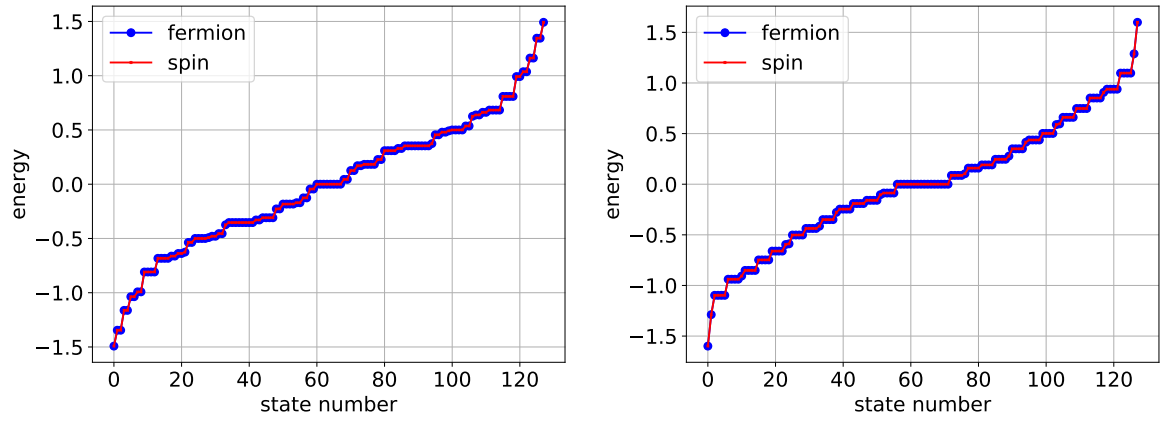
In one dimension, the TFI Hamiltonian can be mapped to spinless p-wave superconducting fermions via the Jordan-Wigner (JW) transformation[CITE Sachdev, other paper]:

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

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



(a) negative spin inversion/PBC sector (b) positive spin inversion/APBC sector

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. 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 $L = 8$.

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
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 **??**.

2.2 Free Particle Systems: the SSH Model

This example shows how to

- construct free-particle Hamiltonians in real space,
- implement translation invariance with a two-site unit cell and construct the single-particle Hamiltonian in momentum space in block-diagonal form,
- compute non-equal time correlation functions,
- ...

Physics Setup—The Su-Schrieffer-Heeger (SSH) model of a free-particle on a dimerised chain is widely used to introduce the concept of edge states, topology, Berry phase, etc., in one spatial dimension. The Hamiltonian is given by

$$H = \sum_{j=0}^{L-1} -(J + (-1)^j \delta J) \left(c_j c_{j+1}^\dagger - c_j^\dagger c_{j+1} \right) + \Delta (-1)^j n_j, \tag{4}$$

where $\{c_i, c_j^\dagger\} = \delta_{ij}$ obey fermionic commutation relations. The uniform part of the hopping matrix element is J, δJ defines the bond dimerisation, and Δ is the staggered potential. We assume periodic boundary conditions.

Below, we show how one can use QuSpin to study the physics of free fermions in the SSH chain. One way of doing this would be to work in the many-body (Fock space) basis, see Sec. ???. However, whenever the particles are non-interacting, the exponential scaling of the Hilbert space dimension with the number of lattice sites imposes an artificial limitation on the system sizes one can do. Luckily, with no interactions present, the many-body wave functions factorise in a product of single-particle states. Hence, it is possible to study the behaviour of many free bosons and fermions by simulating the physics of a single particle.

Making use of translation invariance, a straightforward Fourier transformation to momentum space, $a_k = \sqrt{2/L} \sum_{j \text{ even}}^{L-1} \mathrm{e}^{-ikj} c_j$ and $b_k = \sqrt{2/L} \sum_{j \text{ odd}}^{L-1} \mathrm{e}^{-ikj} c_j$, casts the SSH Hamiltonian in the following form

$$H = \sum_{k \in \mathrm{BZ}'} (a_k^\dagger, b_k^\dagger)^t \begin{pmatrix} \Delta & -(J+\delta J)\mathrm{e}^{-ik} - (J-\delta J)\mathrm{e}^{+ik} \\ -(J+\delta J)\mathrm{e}^{+ik} - (J-\delta J)\mathrm{e}^{-ik} & -\Delta \end{pmatrix} \begin{pmatrix} a_k \\ b_k \end{pmatrix}, \tag{5}$$

where the reduced Brillouin zone is defined as $\mathrm{BZ}' = [-\pi/2, \pi/2)$. We thus see that the Hamiltonian reduces further to a set of independent 2×2 matrices. The spectrum of the SSH model is gapped, see Fig. ???.

Since we are dealing with free fermions, the ground state is the Fermi sea, $|\mathrm{FS}\rangle$, defined by filling up the lowest band completely. We are interested in measuring the real-space non-equal time correlation function

$$C_{ij}(t) = \langle \mathrm{FS}| n_i(t) n_j(0) |\mathrm{FS}\rangle = \langle \mathrm{FS}(t)| n_i(0) \underbrace{U(t,0) n_j(0) |\mathrm{FS}\rangle}_{|\mathrm{nFS}(t)\rangle}. \tag{6}$$

For simplicity, let us focus on a single unit cell. Figure ??? shows the time evolution of $C_{AA}(t)$ and $C_{AB}(t)$.

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
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 **??**.

2.3 The Gross-Pitaevskii Equation and Nonlinear Time Evolution

This example shows how to

- simulate time-dependent nonlinear equations of motion
- use imaginary time dynamics to find a lowest energy configuration
- ...

Physics Setup—The Gross-Pitaevskii wave equation (GPE) has been shown to govern the physics of weakly-interacting bosonic systems. It constitutes the starting point for studying Bose-Einstein condensates, but can also appear in non-linear optics, and represents the natural description of Hamiltonian mechanics in the wave picture. One of its characteristic features is that it exhibits chaotic classical dynamics, a physical manifestation of the presence of a cubic non-linear term.

Here, we study the time-dependent GPE on a one-dimensional lattice:

$$i\partial_t\psi_j(t) = -J\left(\psi_{j-1}(t)+\psi_{j+1}(t)\right) + \frac{1}{2}\omega_{\rm trap}(t)(j-j_0)^2\psi_j(t) + U|\psi_j(t)|^2\psi_j(t), \tag{7}$$

where J is the hopping matrix element, $\omega_{\rm trap}(t) = (\omega_f-\omega_i)t/t_{\rm ramp}+\omega_i$ – the slowly-varying time-dependent harmonic trap frequency over a time scale $t_{\rm ramp}$, and U – the interaction strength. The lattice sites are labelled by $j=0,\ldots,L-1$, and j_0 is the centre of the 1d chain. We set the lattice constant to unity, and use open boundary conditions.

Whenever $U=0$, the system is non-interacting and the GPE reduces to the Heisenberg EOM for the bosonic field operator $\hat{\psi}_j(t)$. Thus, for the purposes of using QuSpin to simulate the GPE, it is instructive to cast Eq. (7) in the following generic form

$$i\partial_t\vec{\psi}(t) = H_{\rm sp}(t)\vec{\psi}(t) + U\vec{\psi}^*(t)\circ\vec{\psi}(t)\circ\vec{\psi}(t), \tag{8}$$

where $[\vec{\psi}(t)]_j=\psi_j(t)$, and $\circ$ represents the element-wise multiplication

$$\vec{\psi}(t)\circ\vec{\phi}(t) = \Big(\psi_0(t)\phi_0(t),\psi_1(t)\phi_1(t),\ldots,\psi_{L-1}(t)\phi_{L-1}(t)\Big)^t.$$

The time-dependent single-particle Hamiltonian in real space is represented as an $L\times L$ matrix, $H_{\rm sp}(t)$, which comprises the hopping term, and the harmonic trap.

We want to initiate the time-evolution of the system at $t = 0$ in its lowest energy state. To this end, we can define a 'ground state' for the GPE equation, in terms of the configuration which minimises the energy of the system:

$$\begin{aligned} \vec{\psi}_{\rm GS} &= \inf_{\vec{\psi}} \left(\vec{\psi}^t H_{\rm sp}(0) \vec{\psi} + \frac{U}{2} \sum_{j=0}^{L-1} |\psi_j|^4 \right), \\ &= \inf_{\psi_j} \left(\sum_{j=0}^{L-1} -J(\psi^*_{j+1}\psi_j + \text{c.c.}) + \frac{1}{2}\omega_{\rm trap}(0)|\psi_j|^2 + \frac{U}{2}|\psi_j|^4 \right). \end{aligned} \tag{9}$$

One way to find the configuration $\vec{\psi}_{\rm GS}$, is to solve the GPE in imaginary time ($it \to \tau$), which induces exponential decay in all modes of the system, except for the lowest-energy state. In doing so, we keep the norm of the solution fixed:

$$\begin{aligned} \partial_\tau \vec{\varphi}(\tau) &= -\Big[H_{\rm sp}(0)\vec{\varphi}(\tau) + U\vec{\varphi}^*(\tau) \circ \vec{\varphi}(\tau) \circ \vec{\varphi}(\tau) \Big], \qquad ||\vec{\varphi}(\tau)|| = \text{const.}, \\ \vec{\psi}_{\rm GS} &= \lim_{\tau\to\infty} \vec{\varphi}(\tau) \end{aligned} \tag{10}$$

Once we have the initial state $\vec{\psi}_{\rm GS}$, we evolve it according to the time-dependent GPE, Eq. (7), and track down the time evolution of the condensate density $\rho_j(t) = |\psi_j(t)|^2$. Fig. ??? shows the result.

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
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 ??.

3 New Horizons for QuSpin

- 2D lattices
- single-particle Hamiltonian class
- Liouville dynamics

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 ???

A 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.

A.1 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.
```

A.2 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.

B 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.

B.1 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!

B.2 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 "..".

B.3 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!

C 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

D 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.