

# 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 3, 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

<b>1</b>	<b>How can I make the most out of QuSpin?</b>	<b>2</b>
<b>2</b>	<b>How do I use the New Features of QuSpin?</b>	<b>2</b>
2.1	The Spectrum of the Transverse Field Ising Model and the Jordan-Wigner Transformation	2
2.2	Free Particle Systems: the SSH Model	5
<b>3</b>	<b>New Horizons for QuSpin</b>	<b>7</b>
<b>A</b>	<b>Installation Guide in a Few Steps</b>	<b>8</b>
A.1	Mac OS X/Linux	8
A.2	Windows	9
<b>B</b>	<b>Basic Use of Command Line to Run Python</b>	<b>9</b>
B.1	Mac OS X/Linux	9
B.2	Windows	10
B.3	Execute Python Script (any operating system)	11
<b>C</b>	<b>Package Documentation</b>	<b>12</b>
<b>D</b>	<b>Complete Example Codes</b>	<b>13</b>

## 1 How can I make the most out of QuSpin?

- research
- education

## 2 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].

### 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 by

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

where the nearest-neighbour (nn) spin interaction is  $J$ ,  $h$  denotes the transverse field, and  $\sigma_j^\alpha$  are the Pauli spin-1/2 matrices. We use periodic boundary conditions and label the  $L$  lattice sites  $0, \dots, L-1$  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]:

$$c_i = \prod_{j<i} \sigma_j^z \sigma_i^-, \quad c_i^\dagger = \prod_{j<i} \sigma_j^z \sigma_i^+, \quad (2)$$

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

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

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*—...

```
1 from quspin.operators import hamiltonian # Hamiltonians and operators
2 from quspin.basis import spin_basis_1d, fermion_basis_1d # Hilbert space spin basis
3 import numpy as np # generic math functions
4 import matplotlib.pyplot as plt
5 ##### define model parameters #####
6 L=8 # system size
7 J=1.0 # spin interaction
8 h=np.sqrt(2) # magnetic field
9 # loop over boundary conditions/spin inversion block variable
```

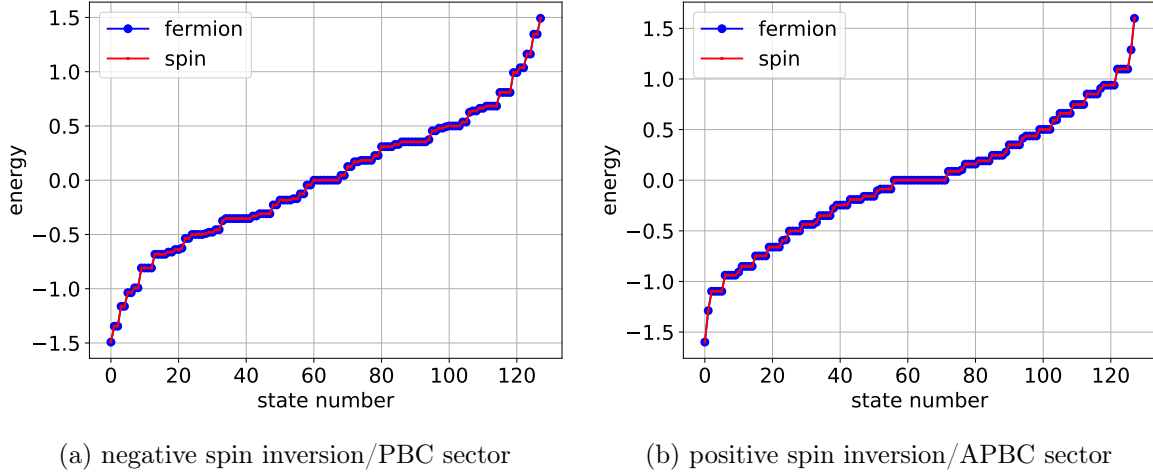


Figure 1: Comparison of the spectra of the spin and fermion representation of the transverse field Ising Hamiltonian in the spin (1) and fermion (3) representations. The degeneracy of the spectrum is due to the remaining parity and momentum conservations which are not taken into account (see text). The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

```

10 for PBC in [1,-1]: # periodic or antiperiodic BC
11     ##### define spin model
12     # site-coupling lists (PBC in both cases)
13     J_zz=[[-J,i,(i+1)%L] for i in range(L)] # PBC
14     h_field=[[-h,i] for i in range(L)]
15     # determine Hilbert space symmetries
16     if PBC==1: # include odd spin inversion sector only
17         basis_spin = spin_basis_1d(L=L,zblock=-1)
18     elif PBC==-1: # include even spin inversion sector only
19         basis_spin = spin_basis_1d(L=L,zblock=1)
20     # define spin static list
21     static_spin=[["zz",J_zz],["x",h_field]]
22     # build spin Hamiltonian
23     H_spin=hamiltonian(static_spin,[],basis=basis_spin,dtype=np.float64)
24     # calculate spin energy levels
25     E_spin=H_spin.eigvalsh()
26     ##### define fermion model
27     # define site-coupling lists for external field
28     h_pot=[[2.0*h,i] for i in range(L)]
29     if PBC==1: # periodic boundary conditions, include odd particle number subspace
        only
30         # define site-coupling lists (including boundary couplings)
31         J_pm=[[-J,i,(i+1)%L] for i in range(L)] # PBC
32         J_mp=[[+J,i,(i+1)%L] for i in range(L)] # PBC
33         J_pp=[[-J,i,(i+1)%L] for i in range(L)] # PBC
34         J_mm=[[+J,i,(i+1)%L] for i in range(L)] # PBC
35         # construct fermion basis in the odd particle number subsector
36         basis_fermion = fermion_basis_1d(L=L,Nf=range(1,L+1,2))
37     elif PBC==-1: # anti-periodic boundary conditions, include even particle number
        subspace only

```

```

38     # define bulk site coupling lists
39     J_pm=[[-J,i,i+1] for i in range(L-1)]
40     J_mp=[[+J,i,i+1] for i in range(L-1)]
41     J_pp=[[-J,i,i+1] for i in range(L-1)]
42     J_mm=[[+J,i,i+1] for i in range(L-1)]
43     # add boundary coupling between sites (L-1,0)
44     J_pm.append([+J,L-1,0]) # APBC
45     J_mp.append([-J,L-1,0]) # APBC
46     J_pp.append([+J,L-1,0]) # APBC
47     J_mm.append([-J,L-1,0]) # APBC
48     # construct fermion basis in the even particle number subsector
49     basis_fermion = fermion_basis_1d(L=L,Nf=range(0,L+1,2))
50     # define fermionic static list
51     static_fermion = [[ "+-",J_pm], [ "-+",J_mp], [ "++",J_pp], [ "--",J_mm], [ 'z',h_pot]]
52     # build fermionic Hamiltonian
53     H_fermion=hamiltonian(static_fermion,[],basis=basis_fermion,dtype=np.float64,
54     check_pcon=False,check_symm=False)
55     # calculate fermionic energy levels
56     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

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

where  $\{c_i, c_j^\dagger\} = \delta_{ij}$  obey fermionic commutation relations. The uniform part of the hopping matrix element is  $J$ ,  $\delta J$  defines the bond dimerisation, and  $\Delta$  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,  $a_k = \sqrt{2/L} \sum_{j \text{ even}}^{L-1} e^{-ikj} c_j$  and  $b_k = \sqrt{2/L} \sum_{j \text{ odd}}^{L-1} e^{-ikj} c_j$ , casts the SSH Hamiltonian in the following form

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

where the reduced Brillouin zone is defined as  $\text{BZ}' = [-\pi/2, \pi/2)$ . We thus see that the Hamiltonian reduces further to a set of independent  $2 \times 2$  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,  $|\text{FS}\rangle$ , defined by filling up the lowest band completely. We are interested in measuring the real-space non-equal time correlation function

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

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—...

```

1 from quspin.operators import hamiltonian # Hamiltonians and operators
2 from quspin.basis import spin_basis_1d, fermion_basis_1d # Hilbert space spin basis
3 import numpy as np # generic math functions
4 import matplotlib.pyplot as plt
5 ##### define model parameters #####
6 L=8 # system size
7 J=1.0 # spin interaction
8 h=np.sqrt(2) # magnetic field
9 # loop over boundary conditions/spin inversion block variable
10 for PBC in [1,-1]: # periodic or antiperiodic BC
11     ##### define spin model
12     # site-coupling lists (PBC in both cases)
13     J_zz=[[-J,i,(i+1)%L] for i in range(L)] # PBC
14     h_field=[[-h,i] for i in range(L)]
15     # determine Hilbert space symmetries
16     if PBC==1: # include odd spin inversion sector only
17         basis_spin = spin_basis_1d(L=L,zblock=-1)
18     elif PBC==-1: # include even spin inversion sector only
19         basis_spin = spin_basis_1d(L=L,zblock=1)
20     # define spin static list
21     static_spin=[["zz",J_zz],["x",h_field]]
22     # build spin Hamiltonian
23     H_spin=hamiltonian(static_spin,[],basis=basis_spin,dtype=np.float64)
24     # calculate spin energy levels
25     E_spin=H_spin.eigvalsh()
26     ##### define fermion model
27     # define site-coupling lists for external field
28     h_pot=[[2.0*h,i] for i in range(L)]
29     if PBC==1: # periodic boundary conditions, include odd particle number subspace
        only

```

```

30     # define site-coupling lists (including boudary couplings)
31     J_pm=[[-J,i,(i+1)%L] for i in range(L)] # PBC
32     J_mp=[[+J,i,(i+1)%L] for i in range(L)] # PBC
33     J_pp=[[-J,i,(i+1)%L] for i in range(L)] # PBC
34     J_mm=[[+J,i,(i+1)%L] for i in range(L)] # PBC
35     # construct fermion basis in the odd particle number subsector
36     basis_fermion = fermion_basis_1d(L=L,Nf=range(1,L+1,2))
37 elif PBC==-1: # anti-periodic boundary conditions, include even particle number
subspace only
38     # define bulk site coupling lists
39     J_pm=[[-J,i,i+1] for i in range(L-1)]
40     J_mp=[[+J,i,i+1] for i in range(L-1)]
41     J_pp=[[-J,i,i+1] for i in range(L-1)]
42     J_mm=[[+J,i,i+1] for i in range(L-1)]
43     # add boundary coupling between sites (L-1,0)
44     J_pm.append([+J,L-1,0]) # APBC
45     J_mp.append([-J,L-1,0]) # APBC
46     J_pp.append([+J,L-1,0]) # APBC
47     J_mm.append([-J,L-1,0]) # APBC
48     # construct fermion basis in the even particle number subsector
49     basis_fermion = fermion_basis_1d(L=L,Nf=range(0,L+1,2))
50     # define fermionic static list
51     static_fermion = [['+-',J_pm],['-+',J_mp],['++',J_pp],['--',J_mm],['z',h_pot]]
52     # build fermionic Hamiltonian
53     H_fermion=hamiltonian(static_fermion,[],basis=basis_fermion,dtype=np.float64,
check_pcon=False,check_symm=False)
54     # calculate fermionic energy levels
55     E_fermion=H_fermion.eigvalsh()

```

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

### 3 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 ???

## 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](https://github.com/weinbe58/QuSpin) 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>

*to report a bug pls visit <https://github.com/weinbe58/QuSpin/issues>*

## 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](https://doi.org/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>.