Documentation: BerryEasy

Alexander C. Tyner

Last updated: March 2024

Email: alextyner[at]gmail.com

Installation

- Requirements: Python version >= 3.6, numpy, scipy, pybinding, tqdm
- CuPy required for GPU utilization
- Install via pip by running:
  - pip install BerryEasy

Getting started:

- If using CPU version:
  - from BerryEasy import BerryEasyCPU as be
- If using GPU version:
  - from BerryEasy import BerryEasyGPU as be

Citing:

- When using BerryEasy please cite:
  - *Tyner, A.C., BerryEasy: A GPU enabled python package for diagnosis of nth-order and spin-resolved topology in the presence of fields and effects*, arXiv:2312.13051 (2023)

Functionalities: Wannier charge centers

- BerryEasy relies on computation of Wannier charge centers (WCCs) for determining topological properties of the system.
- Wannier charge centers are defined in one-dimension as:

$$\bar{x}_n = \frac{ia}{2\pi} \int_{-\pi/a}^{\pi/a} dk \langle u_{nk} | \partial_k | u_{nk} \rangle = \frac{a}{2\pi} \int_{-\pi/a}^{\pi/a} dk \mathcal{A}_n(k)$$

- In BerryEasy, WCCs are computed discretely as:

$$e^{i2\pi\bar{x}_n} = F_{n,\mathbf{k}+N\Delta k} \ldots F_{n,k+\Delta k} F_{n,k}$$

Where we define:

$$F_{n,k+N\Delta k} = \langle u_{n,k+\Delta k} | u_{n,k} \rangle \quad \text{and} \quad \Delta k = 2\pi/N$$

Functionalities: Wannier charge centers

- In order to compute topological invariants for systems in more than one-dimension, BerryEasy relies on computation of hybrid WCCs.
- Hybrid WCCs are simply defined as WCCs computed along a principal direction of a d-dimensional system as a function of the transverse momenta.
- For example, in a two-dimensional system we compute:

$$\bar{x}_n(k_y) = \frac{ia}{2\pi}\int_{-\pi/a}^{\pi/a} dk \langle u_{nk}|\partial_{k_x}|u_{nk}\rangle = \frac{a}{2\pi}\int_{-\pi/a}^{\pi/a} dk_x \mathcal{A}_n(\mathbf{k})$$

- Using BerryEasy, there are two main functions for computing hybrid WCCs:
  - *WSurf*- To compute two-dimensional invariants efficiently
  - *WLine*- To compute WCCs over a single custom defined, closed curve

Functionalities: *WSurf*

- The *WSurf* function allows for computation of hybrid Wannier center charges as a function of a given transverse momenta.
- Implementation of *WSurf*:

```
import BerryEasy as be
WCCx=be.WSurf(vec,syst,bands,ds,ds2,rvec)
```

→ Returns: Array with WCC values computed at each value of transverse momenta considered

Ex.) vec=lambda t1, t2: [t1, t2, 0]

- t1 : direction of integration in units of the reciprocal lattice vectors.
- t2 : transverse momenta at which to compute WCCs in units of reciprocal lattice vectors.
- In this case: specifies integation along kx as function of ky.
- Note that t1 must be periodic, i.e. the Hamiltonian at [t1=0,t2,0].rvec and [t1=1,t2,0].rvec must be equivalent

- Pybinding instance to be investigated
- List; bands to calculate WCCs. Can be given as a list or range, i.e. bands=[0,1] is equivalent to bands=range(2). Bands are labeled sequentially in terms of increasing energy

- Integer: determines discretization of integration
- Integer: discretization of transverse momentum direcion
- Array: Reciprocal lattice vectors

Functionalities: *WLine*

- The *WLine* function allows for computation of the WCCs along a single closed curve in reciprocal space.
- While less convenient for most situations, it is useful for considering arbitrary closed curves that may not travel along a principal direction.

```
import BerryEasy as be
WCCx=be.WLine(line,syst,bands)
```

→ Returns: Array with WCC value for each band

Array: contains path-ordered list of points discretizing the custom line along which to integrate the WCCs

- Pybinding instance to be investigated

- List; bands to calculate WCCs. Can be given as a list or range, i.e. bands=[0,1] is equivalent to bands=range(2). Bands are labeled sequentially in terms of increasing energy

Functionalities: Nested Wilson Loops (NWLs)

- Nested Wilson loops (NWLs) can be computed for systems in two or more physical dimensions.
- The nested Wilson loop is computed by first computing the Wannier Hamiltonian. For example, we can consider a two-dimensional system and compute the x-direction Wannier Hamiltonian which is a function of the transverse momenta ky, $H_{Wx,n}(k_y)$:

$$e^{iH_{Wx,n}(k_y)} = e^{i2\pi\bar{x}_n(k_y)}$$

- The nested Wilson loop then follows as computation of the WCC for the Wannier Hamiltonian
- BerryEasy offers the *WNWL* function for computing NWLS

Functionalities: *WNWL*

- The WNWL function allows for computation of nested WCCs within BerryEasy

import BerryEasy as be
WCCx=be.WNWL(vec,syst,bnds,bnds2,ds,ds2,rvec)

Returns: Array with WCC values computed at each value of transverse momenta considered

Ex.) vec=lambda t1, t2: [t1, t2, 0]

- t1 : direction of integration for computing Wannier Hamiltonian in units of reciprocal lattice vectors
- t2 : direction of integration for computing WCC of Wannier Hamiltonian in units of reciprocal lattice vectors
- Ex. function shows computation of WCCs for Wannier Hamiltonian $H_{Wx,n}(k_y)$

- Pybinding instance to be investigated
- List: Bands for construction of Wannier Hamiltonian
- List: Bands of Wannier Hamiltonian to consider when computing WCCs

- Integer: determines discretization of integration for constructing Wannietr Hamiltonian
- Integer: discretization of integration for computing WCCs of Wannier Hamiltonian
- Array: Reciprocal lattice vectors

Functionalities: Spin-resolved WCCs

- In spinful systems the up/down spin bands may carry an opposite topological invariant. We must compute the WCCs for the spin up/down sectors independently to define the relative invariant
- The most general way to do this is through construction of the projected spin-operator (PSO)

$$PSO(\mathbf{k}) = \mathbf{P}(\mathbf{k})\hat{s}\mathbf{P}(\mathbf{k})$$

- Where **P**(**k**) is the projector over occupied bands and $\hat{s}$ is the spin-operator for the preferred spin direction. If a spectral gap exists in this operator the negative/positive eigenstates correspond to the spin down/up states.
- WCCs for the PSO can then be computed following the algorithm used previously, substituting the Bloch Wavefunctions of the Hamiltonian with those of the PSO.
- BerryEasy offers two functions for computing spin-resolved WCCs:
  - *WSpinSurf* – To compute spin-resolved two-dimensional invariants efficiently
  - *WSpinLine*- To compute spin-resolved WCCs over a single custom defined, closed curve

Functionalities: *WSpinSurf*

- The *WSpinSurf* function allows for computation of spin-resolved hybrid Wannier center charges as a function of a given transverse momenta.
- Implementation of *WSpinSurf*:

```
import BerryEasy as be
WCCx=be.WSpinSurf(vec,syst,bands,ds,ds2,S,rvec)
```

→ Returns: Array with spin-resolved WCC values computed at each value of transverse momenta considered

Ex.) vec=lambda t1, t2: [t1, t2, 0]

- t1 : direction of integration in units of the reciprocal lattice vectors.
- t2 : transverse momenta at which to compute WCCs in units of reciprocal lattice vectors.
- In this case: specifies integation along kx as function of ky.
- Note that t1 must be periodic, i.e. the Hamiltonian at [t1=0,t2,0].rvec and [t1=1,t2,0].rvec must be equivalent

- Pybinding instance to be investigated
- List; bands to calculate WCCs. Can be given as a list or range, i.e. bands=[0,1] is equivalent to bands=range(2). Bands are labeled sequentially in terms of increasing energy

- Integer: determines discretization of integration
- Integer: discretization of transverse momentum direcion
- Array: Operator for preferred spin direction
- Array: Reciprocal lattice vectors

Functionalities: *WSpinLine*

- The *WSpinLine* function allows for computation of the WCCs along a single closed curve in reciprocal space.
- While less convenient for most situations, it is useful for considering arbitrary closed curves that may not travel along a principal direction.

```
import BerryEasy as be
WCCx=be.WSpinLine(line,syst,bands,S)
```

→ Returns: Array with spin-resolved WCC value for each band

Array: contains path-ordered list of points discretizing the custom line along which to integrate the WCCs

- Pybinding instance to be investigated

- List; bands to calculate WCCs. Can be given as a list or range, i.e. bands=[0,1] is equivalent to bands=range(2). Bands are labeled sequentially in terms of increasing energy

- Array: Operator for preferred spin direction

Functionalities: *spin_spectrum*

- In order to compute spin-resolved topological quantities, we need information about the proper spin-direction for which the projected spin-operator is gapped.
- The spin-spectrum function computes the spectra of the projected spin-operator to aid such searchers

```
import BerryEasy as be
WCCx=be.spin_spetcrum(kx,ky,kz,syst,bands,S)
```

Returns: Array with eigenvalues of PSO at given k-point

- Pybinding instance to be investigated
- Array: Operator for preferred spin direction
- List; bands to calculate WCCs. Can be given as a list or range, i.e. bands=[0,1] is equivalent to bands=range(2). Bands are labeled sequentially in terms of increasing energy

Allowing for disorder and impurities:

- In the presence of disorder/impurities which violate the translational symmetry, we can continue to utilize the methodologies defined above by working with twisted boundary conditions (TBCs).
- Twisted boundary conditions are imposed in one-dimension as:

$$\psi(x) = e^{i\theta_x/L}\psi(x+L)$$

- If we then define:

$$k_x \rightarrow k_x^{'} + \theta_x/L,\ 0 \leq \theta_x \leq 2\pi,\ k_x^{'} = 2n_x\pi/L,\ 0 \leq n_x < 2\pi$$

- The same computations defined previously for determining topology are robust.
- When interfacing with PyBinding we fix $\theta_x = 0$ by setting periodic boundary conditions and vary $k_x^{'}$

BerryEasy takes care of this automatically. In the presence of disorder we must only be sure to properly define:

- The reciprocal lattice vectors of the disordered system we consider, i.e. for a 2D disordered N x M system the reciprocal lattice vectors become $(\frac{2\pi}{M}, \frac{2\pi}{N}, 0)$
- Define the Pybinding instance such that the supercell has periodic boundary conditions with the proper lattice vectors, i.e. Lattice vectors (N, M) for a 2D, disordered N x M

Functionalities: Analyzing Wannier90 tight-binding models

- Wannier90 is an extremely useful package for creating realistic tight-binding models from density functional theory simulations
- BerryEasy offers a built-in function to create a PyBinding instance from the output of Wannier90 for topological analysis.
- The Wannier90 output files required are :
  - Wannier90_hr.dat
  - Wannier90_centres.xyz
- The function to create the PyBinding lattice from Wannier90 is *wan90_lat*

Functionalities: *wan90_lat*

- The *wan90_lat* function allows for creation of a Pybinding lattice from Wannier90 output files

```
import BerryEasy as be
WCCx=be.wan90_lat(ECut,HopCut,seedname)
```

Returns: PyBinding lattice object

Float: hopping strengths below this value will be set to zero

- Integer: Hopping between neighbors further than this value will be set to zero

- String: Name of input files, i.e. seedname_tb.dat and seedname_centres.xyz

Interfacing with Kwant

- Kwant is a widely used package for real-space analysis of tight-binding models (https://kwant-project.org/)
- BerryEasy offers a beta interface with Kwant (Warning that it is still in development)
- Wannier90 compatibility is yet to be implemented
- **In Kwant, the model must be defined as a function of the twisted boundary conditions, i.e. in one-dimension:**

$$H(\theta_x) \text{ where, } \psi(x) = e^{i\theta_x/L}\psi(x+L)$$

- Upon being defined as a function of twisted boundary conditions, use of BerryEasy is identical , except in the function we must define “kwant=True”, for example:

```
import BerryEasy as be
WCCx=be.WSpinLine(line,syst,bands,S,kwant=True)
```

→ Returns: Array with spin-resolved WCC value for each band

- For explicit examples, please see the Kwant-BerryEasy Tutorial on the BerryEasy GitHub page

List of functionalities:

- *WSurf*
- *WLine*
- *WNWL*
- *WSpinSurf*
- *WSpinLine*
- *wan90_lat*

In development: *Spin-resolved density of states and enhanced Kwant Interface*
Source Code is available on PyPi: ***https://pypi.org/project/BerryEasy/***

Email if there are more you would like to see!