Setting up a Hamiltonian

In this basic tutorial we will address how to compute the band structure of a one dimensional tight binding model.

The Hamiltonian of a one dimensional tight binding chain takes the form

$$H = \sum_n c_n^\dagger c_{n+1} + h.c.$$

This model can be diagonalized analytically, giving rise to a diagonal Hamiltonian of the form

$$H = \sum_k \epsilon_k \Psi_k^\dagger \Psi_k$$

where energy momentum dispersion takes the form

$$\epsilon_k = 2\cos k$$

With the pyqula library, the previous band structure can be computed as

```
from pyqula import geometry
g = geometry.chain() # geometry of the 1D chain
h = g.get_hamiltonian() # generate the Hamiltonian
(k,e) = h.get_bands() # compute band structure
```

Including second and third neighbor hopping

By default, the Hamiltonian generated includes only first neighbor hopping $t_1 = 1$. However, we may want to consider a generalized Hamiltonian of the form

$$H = \sum_n c_n^\dagger c_{n+1} + t_2 \sum_n c_n^\dagger c_{n+2} + t_3 \sum_n c_n^\dagger c_{n+3} + h.c.$$

To compute the eigenvalues in this generalized model taking $t_2 = 0.2$ and $t_3 = 0.3$, we write

```
from pyqula import geometry
g = geometry.chain() # geometry of the 1D chain
h = g.get_hamiltonian() # generate the Hamiltonian
(k,e) = h.get_bands(tij=[1.0,0.2,0.3]) # compute band structure
```

Including an onsite energy

The Hamiltonian can have an onsite energy term, that is equivalent to a chemical potential that takes the form

$$H = \mu \sum_n c_n^\dagger c_n$$

This can be added to the Hamiltonian as

```
from pyqula import geometry
g = geometry.chain() # geometry of the 1D chain
h = g.get_hamiltonian() # generate the Hamiltonian
mu = 0.3 # value of the onsite
h.add_onsite(mu) # add onsite energy
```

Possible inputs

- Float: the same onsite energy is added to all the sites
- Iterable (list or array): adds a different onsite energy to each site in teh geometry
- Callable (function): adds a different onsite energy to each site according to its location **r**

Including an external Zeeman field

In the following we will consider that we want to add an external Zeeman field to the electronic system. We now include the existence of a spin degree of freedom, considering the Hamiltonian

$$H = H_0 + H_Z$$

where H_0 is the original tight binding Hamiltonian

$$H_0 = \sum_{n,s} c_{n,s}^\dagger c_{n+1,s} + h.c.$$

and

$$H_Z = \sum_{n,s,s'} \vec{B} \cdot \vec{\sigma}^{s,s'} c_{n,s}^\dagger c_{n,s'}$$

with n running over the sites and s, s' running over the spin degree of freedom. The magnetic field takes the form $\vec{B} = (B_x, B_y, B_z)$, and σ_α are the spin Pauli matrices. To add a magnetic field of the form $\vec{B} = (0.1, 0.2, 0.3)$ to our chain we write

```
from pyqula import geometry
g = geometry.chain() # geometry of the 1D chain
h = g.get_hamiltonian() # generate the Hamiltonian
h = g.add_zeeman([0.1,0.2,0.3]) # add the Zeeman field
(k,e) = h.get_bands() # compute band structure
```

Including an external orbital field

An external magnetic field can be included using the Peierls substitution

$$t_{\alpha\beta} \to t_{\alpha\beta} e^{i \int_{r_\alpha}^{r_\beta} \vec{A}\cdot d\vec{l}}$$

where $\vec{A}$ is the magnetic potential so that $\vec{B} = \nabla \times \vec{A}$. It can be used as shown in the example below

```
from pyqula import geometry
N = 20 # number of unit cells as the width
g = geometry.square_ribbon(N) # ribbon
B = 0.02 # magnetic field in quantum flux unit
h.add_orbital_magnetic_field(B) # add an out-of plane magnetic field
```

Setting a fiilling

If you want to enforce a certaing filling ν in a Hamiltonian, so that

$$\langle c_n^\dagger c_n \rangle = \nu$$

use

```
from pyqula import geometry
g = geometry.chain() # chain
h = g.get_hamiltonian()
h.set_filling(0.7) # enforce a filling
```

Possible inputs

- float: enforce the filling on average
- array: enforce that each site has a specific filling

Observables

Electronic band structures

For any system that is periodic in space, can compute the electronic band structure as given by

$$H = \sum_{k,\alpha} \epsilon_{k,\alpha} \Psi^{\dagger}_{k,\alpha} \Psi_{k,\alpha}$$

where α is the band index.

The previous calculation can be performed as

```
from pyqula import geometry
g = geometry.honeycomb_lattice() # geometry of the 2D model
h = g.get_hamiltonian() # generate the Hamiltonian
(k,e) = h.get_bands() # compute band structure
```

Density of states

The density of states counts how many states are in a certian energy window. It is defined as

$$D(\omega) = \int \delta(\omega - \epsilon_k) dk$$

where ϵ_k are the eigenenergies of the Hamiltonian. It can be used as shown below

```
from pyqula import geometry
g = geometry.triangular_lattice() # get the geometry
h = g.get_hamiltonian()  # get the Hamiltonian
(es,ds) = h.get_dos()
```

Optional arguments - energies: array with the energies for which the DOS is computed - delta: smearing of the DOS - operator: operator to which the DOS is projected

Local density of states

The density of states counts how many states are in a certian energy window. It is defined as

$$D(\omega, n) = \int \delta(\omega - \epsilon_k) |\langle \Psi_k | n \rangle|^2 dk$$

where ϵ_k are the eigenenergies of the Hamiltonian. It can be used as shown below

```
from pyqula import geometry
g = geometry.hoenycomb_zigzag_ribbon() # get the geometry
h = g.get_hamiltonian()  # get the Hamiltonian
(x,y,d) = h.get_ldos()
```

Optional arguments - energies: array with the energies for which the DOS is computed - delta: smearing of the DOS - operator: operator to which the DOS is projected

Momentum resolved spectral functions

Apart from the band structure, in certain cases it is interesting to compute the momentum resolved spectral function, that takes the form

$$A(k,\omega) = \delta(\omega - \epsilon_k)|\langle \Psi_k | A | \Psi_k \rangle|^2$$

where A is a certain operator. The previous quantity allows define a heatmap of the momentum-resolved spectral function. For the example, in a superconducting state, if operator is chosen to be projection onto the electron-sector, the previous quantity shows the electronic spectral fucntion

```
from pyqula import geometry
g = geometry.triangular_lattice() # get the geometry
h = g.get_hamiltonian()  # get the Hamiltonian
h.get_kdos_bands()
```

Optional arguments - energies: array with the energies for which the DOS is computed - delta: smearing of the DOS - operator: operator to which the DOS is projected

Operators

Both when computing band structures, density of states and expectation values we could define operators to filter the results. In this section we elaborate on some important operators that are available, and we comment on their physical meaning.

Operators in pyqula have some important properties. First, for periodic Hamiltonian they can have an intrinsic momentum dependence. Second, pyqula allows for native algebra between them, namely they can be summed or multiplied, automatically accounting for intrinsic momentum depences. Third, they can be non-linear, providing a generalization of matrix operators.

Spin operators

The simplest operators are the spin operators

$$S_\alpha = \sum_n \sigma_\alpha^{\mu\nu} c^\dagger_{n,\mu} c_{n,\nu}$$

with σ_α the Pauli matrices, that can be obtained as

```
from pyqula import geometry
g = geometry.triangular_lattice() # get the geometry
h = g.get_hamiltonian()  # get the Hamiltonian
sx = h.get_operator("sx") # Spin x component
sy = h.get_operator("sy") # Spin y component
sz = h.get_operator("sz") # Spin z component
```

Location operator

To understand the spatial location of the states we can use the spatial operators, that denote where wavefucntions are located in real space

$$R_\alpha = \sum_{r,s} r_\alpha c^\dagger_{r,s} c_{r,s}$$

with r_α is the component of the position of site r

```
from pyqula import geometry
g = geometry.triangular_lattice() # get the geometry
h = g.get_hamiltonian()  # get the Hamiltonian
x = h.get_operator("xposition") # x component
y = h.get_operator("yposition") # y component
z = h.get_operator("zposition") # z component
```

Bulk-edge operator

In order to know if a state is located at the edge or in the bulk of the system you can use the bulk-edge location operators. The edge operator takes value 1 for sites on the edge, and 0 for sites in the bulk.

$$hatE = \sum_{r\in \text{Edge},s} c^\dagger_{r,s} c_{r,s}$$

The bulk operator takes value 1 for sites on the bulk, and 0 for sites on the edge.

$$hatB = \sum_{r\in \text{Bulk},s} c^\dagger_{r,s} c_{r,s}$$

```
from pyqula import geometry
g = geometry.honeycomb_zigzag_ribbon() # get the geometry
h = g.get_hamiltonian()  # get the Hamiltonian
b = h.get_operator("bulk") # bulk operator
e = h.get_operator("edge") # edge operator
```

Valley operator

For hoenycomb like systems, including aligned and twisted multilayers, an operator that allows to extract the valley degree of freedom can be extracted. This operator takes the form

$$V = i \sum_{\langle\langle ij \rangle\rangle, s} \nu_{ij} \sigma_{ij} c^\dagger_{r_i,s} c_{r_j,s}$$

where $\nu = \pm 1$ and $\sigma = \pm 1$ for clockwise/anticlockwise, sublattice A/B. This the so-called anti-Haldane hopping, and takes opposite values in opposite valleys. It can can be obtained for honeycomb systems as

```
from pyqula import geometry
g = geometry.honeycomb_zigzag_ribbon() # get the geometry
h = g.get_hamiltonian()  # get the Hamiltonian
vall = h.get_operator("valley") # valley operator
```

Nambu operators

In the presence of superconductivity, you can project onto the electron or hole component of the Nambu spinor using the electron-hole operators

```
from pyqula import geometry
g = geometry.triangular_lattice() # get the geometry
h = g.get_hamiltonian()  # get the Hamiltonian
e = h.get_operator("electron") # electron component
h = h.get_operator("hole") # hole component
```

Berry curvature operator

The Berry curvature operator is a first example of an operator that is intrinsically momentum dependent. The Berry curvature operator is defined as

$$O|\Psi_k\rangle = \Omega(k, \epsilon_k)|\Psi_k\rangle$$

where $\Omega(k, \omega)$ is the Berry curvature evaluated at the momentum k and energy ω of the eigenstate $|\Psi\rangle$. In particular, this operator allows to directly see the contribution to the Berry curvature of different states in the band structure.

Inverse participation ratio operator

So far we have considered operators that are linear, namely that fufill the condition

$$A(|\Psi_1\rangle + |\Psi_2\rangle) = A|\Psi_1\rangle + A|\Psi_2\rangle$$

There is however one operator that it is interesting to consider that does not fufill such condition. The operator is the so-called inverse participation ration, which we define as

$$O|\Psi\rangle = \sum_i |\langle i|\Psi\rangle|^4 |\Psi\rangle$$

In particular, the previous operator allows to identify states that are highly localized in a few lattice sites, becoming useful to highlight impurity states and localized modes.

```
from pyqula import geometry
g = geometry.honeycomb_zigzag_ribbon() # get the geometry
h = g.get_hamiltonian()  # get the Hamiltonian
h.add_onsite(0.3) # add a sublattice imbalance
ipr = h.get_operator("IPR") # IPR operator
```

Superconductivity

Up to now we have focused on Hamiltonians that contain only normal terms, namely that the full Hamiltonian can be written as

$$H_0 = \sum_{ijss'} t_{ijss'} c^\dagger_{i,s} c_{j,s'}$$

where ij runs onver sites and ss' over spins.

In the presence of superconductivity, an anomalous term appears in the Hamiltonian taking the form

$$H_{SC} = \sum_{ijss'} \Delta^{ss'}_{ij} c_{i,s} c_{j,s'} + h.c.$$

To solve the Hamiltonian

$$H = H_0 + H_{SC}$$

we define a Nambu spinor that takes the form

$$\Psi_n = \begin{pmatrix} c_{n,\uparrow} \\ c_{n,\downarrow} \\ c^\dagger_{n,\downarrow} \\ -c^\dagger_{n,\uparrow} \end{pmatrix}$$

and rewrite the Hamiltonian as

$$H = \Psi^\dagger \mathcal{H} \Psi$$

where $\mathcal{H}$ is the nambu Hamiltonian. In this new basis, the Hamiltonian can be written in a diagonal form as

$$H = \sum_\alpha \epsilon_\alpha \Psi^\dagger_\alpha \Psi_\alpha$$

where ϵ_α are the Nambu eigenvalues.

s-wave superconductivity

The simplest form of superconductivity is spin-singlet s-wave superconductivity. A minimal superconducting term of this form can be written as

$$H_{SC} = \Delta_0 \sum_n c_{n,\uparrow} c_{n,\downarrow} + h.c.$$

In the following, we address the electronic structure of a triangular lattice with s-wave superconductivity, whose Hamiltonian takes the form

$$H = H_0 + H_{SC}$$

with

$$H_0 = \sum_{\langle ij \rangle} c_i^\dagger c_j + h.c.$$

The previous Hamiltonian can be computed for $\Delta_0 = 0.2$ as

```
from pyqula import geometry
g = geometry.triangular_lattice() # geometry of the 2D model
h = g.get_hamiltonian() # generate the Hamiltonian
h.add_swave(0.2) # add s-wave superconductivity
(k,e) = h.get_bands() # compute band structure
```

Note that due to the BdG nature of the Hamiltonian, the bandstructure shows both the electron and hole states

Interactions at the mean-field level

In this section we address how interactions can be treated at the mean-field level.

The collinear Hubbard model

We will start with the simplest interaction term, a local repulsive interaction in a spinful system. Our full Hamiltonian takes the form

$$H = \sum_{\langle ij \rangle} c_i^\dagger c_j + h.c. + U \sum_i c_{i,\uparrow}^\dagger c_{i,\uparrow} c_{i,\downarrow}^\dagger c_{i,\downarrow}$$

The interaction term $U \sum_i c_{i,\uparrow}^\dagger c_{i,\uparrow} c_{i,\downarrow}^\dagger c_{i,\downarrow}$ is solved at the mean-field level. The mean-field approximation consists on replacing the previous four fermion operator, by all the terms that arise by taking expectation value in two of the fermions. In particular, in its simplest collinear form, the mean-field term takes the form

$$H_U^{MF} = U \sum_i \langle c_{i,\uparrow}^\dagger c_{i,\uparrow} \rangle c_{i,\downarrow}^\dagger c_{i,\downarrow} + c_{i,\uparrow}^\dagger c_{i,\uparrow} \langle c_{i,\downarrow}^\dagger c_{i,\downarrow} \rangle$$

where $\langle \rangle$ denotes the ground state expectation value of those operators. The full Hamiltonian thus takes the form

$$H^{MF} = \sum_{\langle ij \rangle} c_i^\dagger c_j + h.c. U \sum_i \langle c_{i,\uparrow}^\dagger c_{i,\uparrow} \rangle c_{i,\downarrow}^\dagger c_{i,\downarrow} + c_{i,\uparrow}^\dagger c_{i,\uparrow} \langle c_{i,\downarrow}^\dagger c_{i,\downarrow} \rangle$$

As a result, the mean-field Hamiltonian depends on the specific ground state of the system, and the ground state depends of course on the specific mean-field Hamiltonian. The previous circular dependence between the ground state and the mean-field Hamiltonian gives rise to a selfconsistent problem.

This selfconsistent condition is solved as follows. We start with an initial guess for the full many-body ground state, that we call $|GS_0\rangle$. With this initial state, we compute the mean-field Hamiltonian H_0^{MF}. This mean-field Hamiltonian allows to compute a new many-body ground state $|GS_1\rangle$, which in turn allows to compute a new mean-field Hamiltonian H_1^{MF}. The previous algorithm is represented as

$$|GS_0\rangle \rightarrow H_1^{MF} \rightarrow |GS_1\rangle \rightarrow H_1^{MF} \rightarrow |GS_2\rangle \rightarrow H_2^{MF} \rightarrow ...$$

This iterative calculation is performed until $H_n^{MF} = H_{n+1}^{MF}$, at which point the algorithm has converged.

Two important notes can we taken from the previous approach. First, the final solution may be sensitive to the initial guess for the ground state. This guess corresponds to the initialization of the Hamiltonian, and it can be important for system whose eenrgy landscape has several local minima. A second point is that the update procedure from one iteration to the next can be done adiabatically, or very suddenly. This corresponds to the mixing between solutions, and for systems close to the critical point can lead to tricky convergence.

Let now show an example of a mean-field calculation. We will take now a square lattice, make a 2x2 supercell and include local repulsive interactions at half filling. The obtained ground state is an antiferromagnetic Neel state that opens a gap at half filling

```
from pyqula import geometry
g = geometry.square_lattice() # geometry of a square lattice
g = g.get_supercell([2,2]) # generate a 2x2 supercell
h = g.get_hamiltonian() # create hamiltonian of the system
h = h.get_mean_field_hamiltonian(U=2.0,filling=0.5,
                                 mf="random") # perform SCF
(k,e) = h.get_bands() # calculate band structure
m = h.get_magnetization() # get the magnetization
```

Non-collinear Hubbard model

In the mean-field ansatz considered above, only a single term in the Wick contraction was considered. This term is the collinear term in the z-direction, and allows accounting for solutions that have magnetization in the z-direction. However, in the presence of frustration, external magnetic field or spin-orbit coupling, the magnetization of a system may be non-collinear and pointing in an arbitrary direction. To account for that phenomenology, the mean-field Hamiltonian must include the non-collinear term that takes the form

$$H_U^{ncMF} = -U \sum_i \langle c_{i,\downarrow}^\dagger c_{i,\uparrow} \rangle c_{i,\uparrow}^\dagger c_{i,\downarrow} + h.c.$$

When including this additional term, the full mean-field Hubbard Hamiltonian is rotationally invariant, meaning that it respects SO(3) spin rotational symmetry. This rotationally symmetric form is the default form implemented in the library.

With the previous point in mind, we now solve a system that develops a non-collinear magnetic state. We take the square lattice considered in the section above, and we add an external magnetic field. The competition between Zeeman energy and antiferromagnetic correlations gives rise to a canted magnetic state

```
from pyqula import geometry
g = geometry.square_lattice() # geometry of a square lattice
g = g.get_supercell([2,2]) # generate a 2x2 supercell
h = g.get_hamiltonian() # create hamiltonian of the system
h.add_zeeman([0.,0.,0.1]) # add out-of-plane Zeeman field
h = h.get_mean_field_hamiltonian(U=2.0,filling=0.5,
                                 mf="random") # perform SCF
(k,e,c) = h.get_bands(operator="sz") # calculate band structure
m = h.get_magnetization() # get the magnetization
```

Superconducting mean-field

In the cases above we focused on local repulsive interactions that promote collinear or non-collinear magnetism. However, local interactions can promote a different type of symmetry breaking, in particular gauge symmetry breaking associated to superconductivity. The emergence of superconductivity is associated to one of the Wick contractions of the mean-field, the anomalous term, that takes the form

$$H_U^{aMF} = U \sum_i \langle c_{i,\uparrow} c_{i,\downarrow} \rangle c_{i,\downarrow}^\dagger c_{i,\uparrow}^\dagger + h.c.$$

The previous term in the mean-field Hamiltonian can become non-zero for $U < 0$, and yields an interaction induced superconducting state. This term in the mean-field Hamiltonian is automatically accounted for in Hamiltonian with Nambu degree of freedom, of course apart from the collinear and non-collinear terms in the mean-field. We show below how an interaction induced superconducting state can be computed with pyqula

```
from pyqula import geometry
import numpy as np
g = geometry.triangular_lattice() # geometry of a triangular lattice
h = g.get_hamiltonian()  # get the Hamiltonian
h.setup_nambu_spinor() # setup the Nambu form of the Hamiltonian
h = h.get_mean_field_hamiltonian(U=-1.0,filling=
                    0.15,mf="swave") # perform SCF
# electron spectral-function
h.get_kdos_bands(operator="electron",nk=400,
                    energies=np.linspace(-1.0,1.0,100))
```

Long range interactions

Up to now we have considered interacting Hamiltonians that only have local (attractive or repulsive) Hubbard interactions. In the following we are going to consider systems that have many-body interactions also to a certain number of neighbors. Long range interactions are crucial to stabilize specific symmetry broken states, and in particular charge density waves, Peierls instabilities and unconventional superconductivity. The full Hamiltonian we will consider takes the form

$$H = \sum_{\langle ij \rangle} c_i^\dagger c_j + h.c. + U \sum_i c_{i,\uparrow}^\dagger c_{i,\uparrow} c_{i,\downarrow}^\dagger c_{i,\downarrow} + V_1 \sum_{\langle ij \rangle, s, s'} c_{i,s}^\dagger c_{i,s} c_{j,s'}^\dagger c_{j,s'}$$

where U parametrizes onsite interactions and V_1 interactions between first neighbors. The previous Hamiltonian gives rise to a variety of terms when performing a mean-field decoupling. By default, pyqula includes all the Wick

contractions of the mean-field, and in the presence of Nambu spinors it includes all the anomalous contractions. Let us now briefly elaborate on some of the additional terms that arise due to the first neighbor interaction V_1.

The first term is the charge order term, that takes the form

$$H^{MF} \sim \langle c^\dagger_{i,s} c_{i,s} \rangle c^\dagger_{j,s'} c_{j,s'}$$

this term can give rise to a different charge imbalance between different sites, and it leads to charge density wave states.

The second term we consider is the bond order, that takes the form

$$H^{MF} \sim \langle c^\dagger_{i,s} c_{j,s} \rangle c^\dagger_{j,s} c_{i,s} + h.c.$$

which leads to an interaction-enhanced hopping. If this happens in a non-uniform way in the system, the resulting state has a Peierls distortion.

Among the anomalous terms, the mean-field Hamiltonian can generate

$$H^{MF} \sim \langle c^\dagger_{i,\uparrow} c^\dagger_{j,\uparrow} \rangle c_{j,\downarrow} c_{i,\downarrow} + h.c.$$

$$H^{MF} \sim \langle c^\dagger_{i,\downarrow} c^\dagger_{j,\downarrow} \rangle c_{j,\uparrow} c_{i,\uparrow} + h.c.$$

$$H^{MF} \sim \langle c^\dagger_{i,\uparrow} c^\dagger_{j,\downarrow} \rangle c_{j,\downarrow} c_{i,\uparrow} + h.c.$$

where the first two-terms corresponds to the odd superconducting order, and the third term account both for even and odd orders.

Below, we show an example in which an interaction-induced spin-triplet term is generated. By considering a electronic structure with a large Zeeman splitting and attractive first neighbor interactions, a state with non-zero $\Delta_{\uparrow\uparrow}$ and $\Delta_{\downarrow\downarrow}$ emerges.

```
import numpy as np
from pyqula import geometry
g = geometry.triangular_lattice() # generate the geometry
h = g.get_hamiltonian() # create Hamiltonian of the system
h.add_exchange([0.,0.,1.]) # add exchange field
h.setup_nambu_spinor() # initialize the Nambu basis
# perform a superconducting non-collinear mean-field calculation
h = h.get_mean_field_hamiltonian(V1=-1.0,
                      filling=0.3,mf="random")
# electron spectral-function
h.get_kdos_bands(operator="electron",nk=400,
                    energies=np.linspace(-2.0,2.0,400))
```

Spatially resolved density of states

```
from pyqula import islands
g = islands.get_geometry(name="honeycomb",n=3,nedges=3) # get an island
h = g.get_hamiltonian() # get the Hamiltonian
h.get_multildos(projection="atomic") # get the LDOS
```

Electronic structure folding and unfolding

In this section we will discuss how we can compute folded band structures in supercells, and most importantly how band structures in a supercell can be unfolded.

Surface spectral functions

In this section we address how we can compute surface spectral function of semi-infinite systems.

Topological insulators

Here we provide a discussion of observables related with topological insulators

Topological invariants

Chern number

The Chern number characterizes two-dimensional topological insulators with broken time reversal symmetry. It is defined as

$$C = \frac{1}{2\pi}\int \Omega(\mathbf{k}) d^2\mathbf{k}$$

where Ω is the Berry curvature. The Chern number can be computed with the following code

```
from pyqula import geometry
from pyqula import kdos
g = geometry.honeycomb_lattice() # create honeycomb lattice
h = g.get_hamiltonian() # create hamiltonian of the system
h.add_haldane(0.05) # Add Haldane coupling
C = h.get_chern() # Chern number
```

Z2 invariant

The Chern number characterizes two-dimensional topological insulators with time reversal symmetry. It can be computed with the following code

```
from pyqula import geometry
from pyqula import kdos
g = geometry.honeycomb_lattice() # create honeycomb lattice
h = g.get_hamiltonian() # create hamiltonian of the system
h.add_soc(0.05) # Add spin-orbit coupling
from pyqula import topology
z2 = topology.z2_invariant(h) # Z2 invariant
```

Berry curvature density in frequency space

The berry curvature in frequency space is defined as

$$\Omega(\mathbf{k}) = \int_{-\infty}^{\epsilon_F} \Xi(\mathbf{k},\omega) d\omega$$

where Ω is the Berry curvature of the occupiad bands and $\Xi(\mathbf{k},\omega)$ is the energy-resolved Berry curvature

Berry curvature density in real-space

The berry curvature in real-space is defined as

$$\Omega(\mathbf{k}) = \int \Gamma(\mathbf{k},\mathbf{r}) d^2\mathbf{r}$$

where Ω is the Berry curvature of the occupiad bands and $\Gamma(\mathbf{k},\mathbf{r})$ is the spatially-resolved Berry curvature. Note that this object is meaningful for periodic systems with very large unit cells.

Chern number in real-space

The berry curvature in real-space is defined as

$$C = \int F(\mathbf{r}) d^2\mathbf{r}$$

where C is the total Chern number of the occupiad bands and $F(\mathbf{r})$ is the spatially-resolved Chern number. Note that this object is meaningful for periodic systems with very large unit cells.

Topological surface states

```
from pyqula import geometry
from pyqula import kdos
g = geometry.honeycomb_lattice() # create honeycomb lattice
h = g.get_hamiltonian() # create hamiltonian of the system
```

```
h.add_haldane(0.05) # Add Haldane coupling
kdos.surface(h) # surface spectral function
```

Response functions

Here we discuss how response functions can be computed

Charge-charge response function

The charge-charge response function for a spinless system is computed as

$$\chi(\omega, i, j) = \sum_{n,m} f(\epsilon_n)(1 - f(\epsilon_m)) \frac{\Psi_n(i)\Psi_m(j)\Psi_m^*(i)\Psi_m^*(j)}{\epsilon_n - \epsilon_m - \omega + i\delta}$$

where $f(\epsilon)$ is the Fermi-Dirac distribution

```
from pyqula import geometry
g = geometry.chain() # create honeycomb lattice
h = g.get_hamiltonian() # create hamiltonian of the system
(es,chis) = h.get_chi(q=[0.,0.,0.]) # get response function
```

RKKY response function

Quantum transport

In this section we discuss how we can perform quantum transport calculations with pyqula.

Magnetoresistence in metal-metal transport

As specific example, here we will address how we can compute magnetoresistence in transport between two magnetic metals

Superconductor-metal transport

Here we address how transport between a superconducting lead and a metallic lead can be computed. As paradigmatic example, we will focus on the Andreev reflection regime and the tunneling regime

```
from pyqula import geometry
from pyqula import heterostructures
import numpy as np
g = geometry.chain() # create the geometry
h = g.get_hamiltonian() # create the Hamiltonian
h1 = h.copy() # first lead
h2 = h.copy() # second lead
```

```
h2.add_swave(.01) # the second lead is superconducting
es = np.linspace(-.03,.03,100) # set of energies for dIdV
for T in np.linspace(1e-3,1.0,6): # loop over transparencies
    HT = heterostructures.build(h1,h2) # create the junction
    HT.set_coupling(T) # set the coupling between the leads
    Gs = [HT.didv(energy=e) for e in es] # calculate conductance
```

Single defects in infinite systems

Main functions and methods

Geometry functions and methods

g.get_hamiltonian()

Generate the Hamiltonian from a geometry.

Optional arguments

- tij = [1.0,.0,0.]: List with 1st, 2nd, 3rd nearest neighbor hopping

Returns the Hamiltonian

g.get_supercell()

Generate a supercell

Arguments

- N: size of the supercell to create, number or tuple

Returns a new geometry

Hamiltonian functions and methods

h.get_bands()

Compute band structure

Optional arguments:

- nk = 20: number of k-points

Returns kpoint index and energies

h.get_dos()

Compute the density of states.

Optional arguments:

- energies: array with frequencies of the DOS

- delta=0.01: broadening of the DOS

Return energies and DOS

h.add_soc()

Add Kane-Mele intrinsic spin-orbit coupling

Arguments:

- value: value of the SOC

h.add_zeeman()

Add a Zeeman field to the Hamiltonian

Arguments:

- value: value of the Zeeman, as a number (assumes [0,0,Bz]), array or callable function

h.add_rashba()

Add Rashba spin-orbit coupling

Arguments:

- value: value of the Rashba SOC

h.add_onsite()

Add a local onsite energy

Arguments:

- value: value of the oniste energy

h.get_ldos()

Compute the local density of states.

Optional arguments:

- e: energy of the LDOS
- delta=0.01: broadening of the LDOS

Return x, position, y position and LDOS

h.get_chern()

Return Chern number of the Hamiltonian.

Optional arguments: - nk=20: number of kpoints