NQFT

Release 0.1.0

Antoine de Lagrave

Dec 08, 2022

CONTENTS:

CHAPTER

ONE

NQFT

1.1 nqft package

1.1.1 Submodules

1.1.2 nqft.functions module

This module contains global functions used in the expression and operation on Hubbard hamiltonian.

It is also used more globally to perform operations on data files containing spectral function.

nqft.functions.**add_column**(*file: str*, *column: array*, *idx: int*) → None

Insert new data to given text file as a column.

Parameters

- **file** (*str, default=None*) – File in which add column.
- **column** (*np.array, default=None*) – Data column to add.
- **idx** (*int, default=0*) – Index of the new column inside current file.

nqft.functions.**delta**(*j: int*, *k: int*) → float

Kronecker delta function.

Parameters

- **j** (*int, default=None*) – First indice.
- **k** (*int, default=None*) – Second indice.

Returns

-

Return type

float (0.0 or 1.0)

Examples

```
>>> delta(1, 1)
1.0
>>> delta(0, 1)
0.0
```

nqft.functions.**find_nearest**(*array*, *value*) → int

Finds index in given array of the closest value of parameter 'value'.

Parameters

- **array** (*array-like, shape=(m, n), default=None*) – Array in which search for the value.
- **value** (*int, float, default=None*) – Value to search for in array.

Returns

idx – Index at which user can find the closest value in array.

Return type

int

Examples

```
>>> a = np.array([0, 1, 2, 3, 4, 5, 6])
>>> find_nearest(a, 3)
3
>>> b = np.array([0.0, 1.111, 2.5, 2.6])
>>> find_nearest(b, 2.0)
2
```

nqft.functions.**flatten_fermi_arc**(*save_path='./nqft/Data/fermi_arc_data_1D'*, *size=36*, *res=200*, *type='text'*) → None

Saves Peter's Fermi arc numpy 2D arrays files as 1D arrays numpy files so they can be read in Rust/C.

Parameters

- **save_path** (*str, default="./nqft/Data/fermi_arc_data_1D/"*) – Path where to save flatten arrays.
- **size** (*int, default=36*) – Number of sites of studied model.
- **res** (*int, default=200*) – Resolution of momentum space.
- **type** (*str, default="text"*) – Format in which save flattened arrays (ex: type="npy" leads to .npy files)

Return type

None

nqft.functions.**make_cmap**(*ramp_colors: list*) → LinearSegmentedColormap

Makes a custom colormap to use in matplotlib 'contourf' or any plot using a colorbar.

Parameters

ramp_colors (*list, default=None*) – Hex code(s) of colors to use when making the colormap.

Returns
color_ramp – Custom colormap

Return type
LinearSegmentedColormap

nqft.functions.**read_fermi_arc**(*path='./nqft/Data/fermi_arc_data'*, *size=36*, *res=200*) → dict

Reads Peter's data on spectral weight at Fermi level for a given number of sites.

Parameters

- **path** (*str, default="./nqft/Data/fermi_arc_data/"*) – Path to data directory.
- **size** (*int, default=36*) – Number of sites of studied model.
- **res** (*int, default=200*) – Resolution of momentum space.

Returns
arcs – A dict containing all spectral functions.

Return type
dict, size=6

nqft.functions.**scalar**(*m: Qobj*, *n=None*) → float

Computes scalar product for Fock space vectors such as

scalar(m, n) = < m | n >.

Parameters

- **m** (*qutip.Qobj, default=None*) – Bra on which perform scalar product.
- **n** (*qutip.Qobj, default=None*) – Ket on which perform scalar product.

Returns
- – Result of scalar product.

Return type
int, float

1.1.3 nqft.hall_effect module

Hall effect in cuprates with an incommensurate collinear spin-density wave.

class nqft.hall_effect.**Model**(*hoppings: tuple[float]*, *broadening: float*, *omega=0.0*, *mus=(-4, 4, 0.02)*, *resolution=600*, *use_peters=(None, 200)*, *use_filter=False*)

Bases: object

Model instance to determine Hall coefficient and density from tight-binding hamiltonian.

hoppings

Hopping amplitude coefficients.

Type
tuple, size=3, default=None

broadening

Lorentzian broadening module.

Type
float, default=None

`omega`

Frequency at which we observe the fermi surface.

Type
float, default=None

`mus`

Chemical potential interval values and the interval between each element of an hypothetical array.

Type
tuple, size=3, default=(-4, 4, 0.02)

`resolution`

Resolution of phase space (k_x, k_y).

Type
int, default=600

`use_peters`

Determines if model's based on Peter R. spectral functions. User must choose between None, 36 and 64 for the first element representing the number of sites and between 200 and 500 for the second element representing the resolution of momentum space.

(Note: User must let first element as 'None' to use non-interacting spectrums normally.)

Type
tuple[int], default=(None, 200)

`use_filter`

Determines if spectral weights will be filtered using diamond shape filter to create artificial Fermi arcs.

Type
bool, default=False

`get_density()` → array

Computes electron density.

Returns
density – Electron density.

Return type
np.array, size=M

`get_hall_nb()` → array

Computes Hall number.

Returns
n_H – Hall number.

Return type
np.array, size=M

`plot_hall()` → Figure

Outputs a plot of the Hall coefficient as a function of doping (1 - density).

Returns
- – Graph of Hall number as a function of hole doping.

Return type
plt.Figure

plot_spectral_weight(*mu: float*, *size=36*, *key=None*) → Figure

Ouputs a matplotlib figure containing 3 subplots. Left one represents the spectral function of non-interacting model. Center one the spectrum comming from Peter's article. Right one the superposition of the firsts.

Parameters

- **mu** (*float, default=None*) – Chemical potential at which we observe spectral function.
- **size** (*int, default=36*) – Size of Peter's model used to compare with non-interacting model.
- **key** (*str, default=None*) – Key of the dictionnary containing Peter's spectrums. For example, the 64 sites model has ('N48', 'N52', 'N56', 'N60' and 'N64').

Returns
- – 2D graphs of spectral weights.

Return type
plt.figure

sigma_ii(*variable: str*) → array

Computing longitudinal conductivity at zero temperature in the zero-frequency limit when interband transitions can be neglected.

Parameters
variable (*str, default=None*) – Axis on which compute conductivity. (ex: 'x' or 'y')

Returns
conductivity

Return type
np.array, size=M

sigma_ij() → array

Computing transversal conductivity at zero temperature in the zero-frequency limit when interband transitions can be neglected.

Returns
conductivity

Return type
np.array, size=M

nqft.hall_effect.**get_energies**(*hops: tuple[float]*, *kx: ndarray*, *ky: ndarray*, *mus: array*) → tuple

Outputs model's energies and it's derivatives.

Parameters

- **hops** (*tuple, default=None*) – Hopping amplitudes coefficients.
- **kx** (*np.ndarray, shape=(N, N), default=None*) – kx space as a 2D array.
- **ky** (*np.ndarray, shape=(N, N), default=None*) – ky space as a 2D array.
- **mus** (*np.array, size=M, default=None*) – Chemical potential values array.

Returns
E, dEs – Energies and it's derivatives in a tuple.

Return type
tuple[np.ndarray, dict], size=2

Examples

```
>>> hops = (1.0, -0.3, 0.2)
>>> ks, mus = np.linspace(-np.pi, np.pi, 2), np.linspace(-4, 4, 4)
>>> kx, ky = np.meshgrid(ks, ks)
>>> get_energies(hops, kx, ky, mus)
(
    array([[[8.4       , 8.4       ],
        [8.4       , 8.4       ]],
       [[5.73333333, 5.73333333],
        [5.73333333, 5.73333333]],
       [[3.06666667, 3.06666667],
        [3.06666667, 3.06666667]],
       [[0.4       , 0.4       ],
        [0.4       , 0.4       ]]]),
    {
        'dE_dx': array([[-1.95943488e-16,  1.95943488e-16],
       [-1.95943488e-16,  1.95943488e-16]]),
        'ddE_dxx': array([[-1.6, -1.6],
       [-1.6, -1.6]]),
        'dE_dy': array([[-1.95943488e-16, -1.95943488e-16],
       [ 1.95943488e-16,  1.95943488e-16]]),
        'ddE_dyy': array([[-1.6, -1.6],
       [-1.6, -1.6]]),
        'ddE_dxdy': array([[-0., -0.],
       [-0., -0.]])
    }
)
```

nqft.hall_effect.**get_spectral_weight**(*omega: float*, *eta: float*, *E: ndarray*, *filter=False*) → tuple[numpy.ndarray]

Ouputs the spectral weight as a 3D numpy array.

Parameters

- **omega** (*float, default=None*) – Frequency at which we observe the fermi surface.
- **eta** (*float default=None*) – Lorentzian broadening module.
- **E** (*np.ndarray. shape=(M, N, N), default=None*) – Eigenenergies of the system as a 3D numpy array.
- **filter** (*bool, default=False*) – Determines if we use diamond filter over spectral weights.

Returns

A, diag_line – Spectral weight and diamond line array to plot over.

Return type

tuple[np.ndarray], size=2

Examples

```
>>> hops = (1.0, -0.3, 0.2)
>>> ks, mus = np.linspace(-np.pi, np.pi, 2), np.linspace(-4, 4, 4)
>>> kx, ky = np.meshgrid(ks, ks)
>>> E, dEs = get_energies(hops, kx, ky, mus)
>>> get_spectral_weight(0.0, 0.05, E)
(
    array([[[0.00022555, 0.00022555],
        [0.00022555, 0.00022555]],
       [[0.00048414, 0.00048414],
        [0.00048414, 0.00048414]],
       [[0.00169189, 0.00169189],
        [0.00169189, 0.00169189]],
       [[0.0979415 , 0.0979415 ],
        [0.0979415 , 0.0979415 ]]]),
    array([[0., 0.],
       [0., 0.]])
)
```

nqft.hall_effect.**timeit**(*func*)

1.1.4 nqft.hamiltonian module

This module tests Qutip python library functions, objects and attributes in the context of Hubbard model using square fermion networks.

class nqft.hamiltonian.**Network**(*sites_nb: int*)

Bases: object

A class representing a fermionic many-body problem Network

sites_nb

Number of sites of the network

Type
int, default=None

vaccum

Vaccum state of 2D Fock space.

Type
Qobj, shape=(2, 1)

creation

Creation operator in second quantization formalism.

Type
Qobj, shape=(2, 2)

anihilation

Anihilation operator in second quantization formalism.

Type
Qobj, shape=(2, 2)

number

Number operator in second quantization formalism.

Type
Qobj, shape=(2, 2)

I

Identity operator.

Type
Qobj, shape=(2, 2)

get_e(*H: Qobj*, *states: list*) → float

Outputs a matrix element (energy) from the hamiltonian using given kets on which perform projection.

Parameters

- **H** (*Qobj, shape=(4^self.sites, 4^self.sites), default=None*) – Fermions network hamiltonian.
- **states** (*array-like, shape=(2, 1), default=None*) – Vectors used to process scalar product on H.

  (ex: states=[bra, ket] as integers to convert from binairy)

Returns
E – Representation of projected vectors on H (energy).

Return type
int

Examples

```
>>> N = Network(sites_nb=2)
>>> Hamiltonian = N.get_hamiltonian(model="Hubbard", U=2, t=1)
>>> N.get_e(H=Hamiltonian, states=[15, 15])
2.0
```

get_hamiltonian(*model: str*, *U: int*, ***kwargs*) → Qobj

Outputs the hamiltonian of fermion network using specified many-body model.

Parameters

- **model** (*str, default=None*) – Fermions network configuration.
- **U** (*int, default=None*) – Module of interaction between fermions.
- ****kwargs** –

  t: int, default=None
  Probability amplitude for fermions to jump.

Returns
H – Tensor object representing hamitonian for given 'model'.

Return type
qutip.Qobj, shape=(4^SITES, 4^SITES)

Examples

```
>>> N = Network(sites_nb=4)
>>> N.get_hamiltonian(model="Hubbard", U=1, t=1)
Quantum object: dims = [[2, 2, 2, 2, 2, 2, 2, 2],
[2, 2, 2, 2, 2, 2, 2, 2]], shape = (256, 256), type = oper,
isherm = True Qobj data =
[[ 0.  0.  0. ...  0.  0.  0.]
 [ 0.  0. -1. ...  0.  0.  0.]
 [ 0. -1.  0. ...  0.  0.  0.]
 ...
 [ 0.  0.  0. ...  3. -1.  0.]
 [ 0.  0.  0. ... -1.  3.  0.]
 [ 0.  0.  0. ...  0.  0.  4.]]
```

get_state(*state: int*, *type='ket'*) → Qobj

Gives array-like representation of given state using Qutip 'tensor' operation.

Parameters

- **state** (*int, default=None*) – Integer representing state number in Fock space.
- **type** (*str, default='ket'*) – Type of vector outputed (bra, ket)

Returns

bra, ket – Full vector representation.

Return type

qutip.Qobj, shape=(4^SITES, 1)

Examples

```
>>> N = Network(sites_nb=1)
>>> self.get_state(state=2)
Quantum object: dims = [[2, 2], [1, 1]], shape = (4, 1), type = ket
Qobj data =
[[0.]
 [0.]
 [1.]
 [0.]]
```

lanczos(*H: Qobj*, *iterations: int*, *init_state=None*) → tuple

Implementation of Lanczos algorithm for Network hamiltonian.

Parameters

- **H** (*Qobj, shape=(4^self.sites, 4^self.sites), default=None*) – Fermions network hamiltonian.
- **iterations** (*int, default=None*) – Number of iterations on which perform the algorithm.
- **init_state** (*QObj, shape=(4^self.sites, 1) default=random*) – Initial quantum state to start the first iteration.

Returns

- – Respectively the maximum eigenvalue and the associated eigenvector.

Return type
tuple, shape=(1, 2)

Examples

```
>>> N = Network(sites_nb=2)
>>> Hamitonian = N.get_hamiltonian(model="Hubbard", U=1, t=1)
>>> N.lanczos(H=Hamiltonian, iterations=10)
(2.561552779602264, Quantum object: dims = [[10], [1]],
shape = (10, 1), type = ket Qobj data =
[[0.43516215]
 [0.78820544]
 [0.43516215]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]])
```

1.1.5 nqft.monte_carlo module

nqft.monte_carlo.**build_h**(*shape: tuple[int]*, *eps: array*, *hops: array*) → ndarray

Docs

1.1.6 nqft.qcm module

This module is dedicated to 'pyqcm' experimentation.

class nqft.qcm.**QcmModel**(*shape: tuple[int]*, *filling: int*, *interaction: float*, *hoppings: tuple[float]*, *broadening: float*, *w: float*, *mu: float*, *resolution: int*, *tiling_shift: bool*, *show_spectrum=False*, *overwrite=False*)

Bases: object

QcmModel instance to make the usage of 'pyqcm' easier.

shape

Shape of the source cluster as: (rows, columns).

Type
tuple[int], size=2, default=None

filling

Number of electrons inside each cluster.

Type
int, default=None

interaction

Coefficient of interation operator.

Type
float, default=None

hoppings

Hopping amplitudes coefficients.

Type
tuple[float], default=None

broadening

Lorentzian broadening module.

Type
float, default=None

w

Frequency at which we observe the fermi surfaces.

Type
float, default=None

mu

Chemical potential.

Type
float, default=None

resolution

Resolution of phase space (k_x, k_y).

Type
int, default=None

tiling_shift

Determines if super-vectors are shifted or exactly orthogonals.

Type
bool, default=None

show_spectrum

Determines if 'pyqcm.spectral.mdc' displays spectral function when it computes it.

Type
bool, default=False

overwrite

Determines if the script reuses an already computed model to do further calcultations or if the script computes it from scratch.

Type
bool, default=False

get_cluster_averages(*operators: list[str]*) → dict

Computes single cluster operator averages.

Parameters
operators (*list, default=None*) – Operators to keep in output dictionnary.

Returns
out_dict – Dictionnary containing operator names as keys and tuples as values representing operator average and it's variance.

Return type
dict

get_lattice_averages(*operators: list[str]*) → dict

Computes lattice operator averages.

Parameters
operators (*list, default=None*) – Operators on which get average.

Returns
_ – Dictionnary containing operator names as keys and average as values.

Return type
dict

plot_spectrums(*peter_key: str*, *type='contourf'*, *save=False*) → Figure

Opens spectrums from (2x2, 3x4, 4x3) models and Peters spectrums array to compare the plot for given parameters.

Parameters

- **peter_key** (*str, default=None*) – Determines which of Peter's array to compare. ('N24', 'N26', 'N28', 'N30', 'N32', 'N34', 'N36')
- **type** (*str, default='contourf'*) – Spectral plot type (contourf, pcolormesh).
- **save** (*bool, default=False*) – Saves of not the output plot.

Returns
–

Return type
matplotlib.pyplot.Figure object

nqft.qcm.**build_matrix**(*shape: tuple*) → list

Gives a coordinates matrix of a cluster having shape[0]*shape[1] sites.

Parameters
shape (*tuple, shape=(2, 1), default=None*) – Shape of sites network.

Returns
array – Nested lists of coordinates.

Return type
list, shape=(*shape)

Examples

```
>>> build_matrix(shape=(2, 2))
[[0, 0, 0], [1, 0, 0], [0, 1, 0], [1, 1, 0]]
```

nqft.qcm.**get_hall_coeff**(*spectral_weight: ndarray*, *hoppings: tuple[float]*, *x_coord=None*, *file='./nqft/Data/hall.txt'*) → float

Computes Hall coefficient for given parameters and writes it to a file using specified x coordinate.

Parameters

- **spectral_weight** (*np.ndarray, shape=(N, M), default=None*) – Spectral function used to compute Hall coefficient.
- **hoppings** (*tuple[float], size=3, default=None*) – Hopping amplitudes corresponding to spectral function.

- **x_coord** (*float/int, default=None*) – X coordinate to use if writting Hall coefficient to a file. (Makes plotting easier and faster)
- **file** (*str, default="./nqft/Data/hall.txt"*) – Path to file in which write Hall coefficient and given x coord.

Returns
n_h – Hall coefficient as a float.

Return type
float

1.1.7 Module contents

1.2 tests package

1.2.1 Submodules

1.2.2 tests.test_nqft module

tests.test_nqft.**test_hamiltonian**()

tests.test_nqft.**test_version**()

1.2.3 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

t

INDEX

Q

R

S

T

U

V

W