NJU_DMRG Documentation

Release 1.0.0

Leo, Yang, Wang and Gu

November 21, 2015

CHAPTER

ONE

HOW TO BUILD LATTICES AND MOMENTUM SPACES

1.1 Variaty Kinds of Lattices

To construct a specific type of lattice, you may use the function

`lattice.latticelib.`**`construct_lattice`**(*N*, *lattice_shape=''*, *a=None*, *catoms=None*, *args={}*)

Uniform construct method for lattice.

N: The size of lattice.

lattice_shape: The shape of lattice.

- '' -> the anonymous lattice.
- 'square' -> square lattice.
- 'honeycomb' -> honeycomb lattice.
- 'triangular' -> triangular lattice.
- 'chain' -> a chain.

a: The unit vector.

catoms: The atoms in a unit cell.

args: Other arguments,

- *form* -> the form used in constructing honeycomb lattice.

It will return a instance of one of the following derivative classes of <Lattice>,

class `lattice.latticelib.`**`Chain`**(*N, a=1.0, catoms=[0.0]*)

Lattice of Chain.

Chain(N,a=(1.),catoms=[(0.,0.)])

`kspace`

The <KSpace> instance correspond to a chain.

class `lattice.latticelib.`**`Square_Lattice`**(*N, catoms=[(0.0, 0.0)]*)

Square Lattice, using C4v Group.

Square_Lattice(N,catoms=[(0.,0.)])

`kspace`

Get the <KSpace> instance.

class `lattice.latticelib.`**`Triangular_Lattice`**(*N, catoms=[(0.0, 0.0)]*)

Triangular Lattice, using C6v Group.

Triangular_Lattice(N,catoms=[(0.,0.)])

kspace
Get the <KSpace> instance.

class lattice.latticelib.**Honeycomb_Lattice**(*N*, *form=1*)
HoneyComb Lattice class.

Honeycomb_Lattice(N,form=1.)

form: The form of lattice. *1* -> traditional one with 0 point at a vertex, using C3v group. *2* -> the C6v form with 0 point at the center of hexagon, using C6v group.

kspace
Get the <KSpace> instance.

The base class <Lattice> is a special kind(derivative) of <Structure>, which is featured with repeated units. To know more about the abilities of our <Lattice> and <Structure> instances,

class lattice.structure.**Structure**(*sites*)
Structure Base class. A collection of sites.

Structure(sites)

sites: Positions of sites.

groups: The translation group and point groups imposed on this lattice.

__kdt__/__kdt_map__: The kd-tree for this structure, and the mapping for kdt index and site index.

b1s
The nearest neighbor bonds.

b2s
The nearest neighbor bonds.

b3s
The nearest neighbor bonds.

findsite(*pos*, *tol=1e-05*)
Find the site at specific position.

pos: The position of the site.

tol: The position tolerence.

***return*:** The site index.

getbonds(*n*)
Get n-th nearest bonds.

n: Specify which set of bonds.

***return*:** A <BondCollection> instance.

initbonds(*nmax=3*, *K=None*, *tol=1e-05*, *leafsize=30*)
Initialize the distance(bond) mesh, and classify it by onsite,1st,2ed,3rd ... nearest neighbours.

nmax: Up to nmax-th neighbor will be considered.

Note: if nmax<0, it will initialize the tree for query but will not initialize bonds.

K: Number of neighbors calculated through cKD Tree, should be >= number of sites up to nmax-th neighbor.

tol: The bond vector tolerence.

leafsize: The leafsize of kdtree.

***return*:** A list of bond vectors, the elements are 0,1,2...-th neightbors.

load_bonds (*filename=None*)
Load bonds.

filename: The target filename.

measure (*i*, *j*, *k=2*)
Measure the 'true' distance between sites at ri and rj. Here, the main problem is the periodic bondary condition.

i/j: index of start/end atom.

k: The maximum times of translation group imposed on r2.

***return*:** (|r|,r), absolute distance and vector distance.

nsite
Number of sites

query (*pos*, *k*, ***kwargs*)
Query the K-nearest sites near specific site.

pos: The position of target site.

k: The number of neighbors.

****kwargs:** Key word arguments for cKDTree.

***return*:** int32 array of length k, the indices of neighbor sites.

save_bonds (*filename=None*, *nmax=3*)
Save bonds.

filename: The target filename.

nmax: Up to nmax-th neighnors are saved.

show_bonds (*nth=(1,)*, *plane=(0, 1)*, *color='r'*)
Plot the structure.

plane: project to the specific plane if it is a 3D structre. Default is (0,1) - 'x-y' plane.

nth: the n-th nearest bonds are plotted. It should be a tuple. Default is (1,) - the nearest neightbor.

c: color, default is 'r' -red.

show_sites (*plane=(0, 1)*, *color='r'*)
Show the sites in this structure.

color: The color, string.

usegroup (*g*)
Apply a group on this lattice.

g: A <Group> instance.

vdim
Dimention of vector space.

class lattice.lattice.**Lattice** (*name, a, N, catoms=[(0.0, 0.0)]*)
Lattice Structure, which contains tranlation of cells.

Lattice(name,a,N,catoms=[(0.,0.)])

name: The name of this latice.

a: Lattice vector

N: Number of cells

catoms: Atoms in one cell.

lmesh: The sites reshaped according to the lattice config (Nx,Ny, ..., ncatom).

cbonds
Get a list of bonds within a unit cell.

dimension
The dimension of lattice.

findsite(*pos*, *tol=1e-05*)
Get the lattice indices from the position.

pos: the position r.

tol: the tolerence of atom position.

***return*:** an array of lattice index.

index2l(*index*)
Get lattice indices (n1,n2,...,atom index in cell) from site index.

index: the site index.

kspace
Get the KSpace instance correspond to this lattice.

l2index(*lindex*)
Get the site index from lattice indices. lindex:

lattice index - (n1,n2,...,atom index in cell)

ncatom
number of atoms within a unit cell.

showcell(*bondindex=(1, 2)*, *plane=(0, 1)*, *color='r'*, *offset=None*)
Plot the cell structure.

bondindex: the bondindex-th nearest bonds are plotted. It should be a tuple. Default is (1,2) - the nearest, and second nearest neightbors.

plane: project to the specific plane if it is a 3D structre. Default is (0,1) - 'x-y' plane.

color: color, default is 'r' -red.

offset: The offset of the sample cell.

siteconfig
The site configuration, taking catoms into account.

CHAPTER

TWO

HOW TO WRITE DOWN HAMILTONIANS

l