LegBuilder Documentation

Release 1.0.0

Jinguo Leo

September 27, 2015

CONTENTS

contents

CHAPTER

ONE

DISCRETIZATION MODULE

Discretize the continuous hybridization function into a discretized model(DiscModel). checking method is also provided.

class discretization.**DiscHandler**(*token, Lambda, N, D=[-1.0, 1.0], Gap=[0.0, 0.0]*)
Bases: object

Handler for discretization of hybridization function.

token: the token of this handler, which is a string for saving/loading data.

Lambda: scaling factor.

N: the maximum discretization index.

D: the band range, [-1,1] for default.

z: number of z.

Gap: the gap range.

check_mapping_eval(*rhofunc*, *Efunc*, *Tfunc*, *scalefunc*, *sgn*, *Nx=1000*, *Nw=200*, *smearing=0.02*)
check the mapping quality by eigenvalues - the multiple-band Green's function version.

rhofunc: the original hybridization function.

Efunc/Tfunc: the representative energy/hopping term as a function of x.

scalefunc: scale function.

sgn: the branch.

Nx/Nw: number of samples in x(index)- and w(frequency)- space.

smearing: smearing constant.

check_mapping_pauli(*rhofunc*, *Efunc*, *Tfunc*, *scalefunc*, *sgn*, *Nx=1000*, *Nw=200*, *smearing=0.02*)
check the mapping quality by Pauli decomposition - only 2-band Green's function is allowed.

rhofunc: the original hybridization function.

Efunc/Tfunc: the representative energy/hopping term as a function of x.

scalefunc: scale function.

sgn: the branch.

Nx/Nw: number of samples in x(index)- and w(frequency)- space.

smearing: smearing constant.

get_Efunc(*Efuncs*)
Get the Efunc for multi-band system.

Efuncs: energy functions for individual bands.

***return*:** a function of representative energy E(x).

get_Tfunc(*wxfuncs*, *efuncs*, *sgn*)
Get the hopping term for multi-band system.

wxfuncs: functions of weights for individual bands, t=sqrt(w) in 1-band system and this is for multi-band.

efuncs: functions of representative energies e(x) for each band.

***return*:** a function of representative hopping terms T(x).

get_discrete_model(*funcs*, *z=1.0*, *append=False*)
get a discrete set of models for specific zs.

funcs: a super tuple ->

(scalefunc_positve,Efunc_positive,Tfunc_positive),(scalefunc_negative,Efunc_negative,Tfunc_negative)

z: twisting parameters, scalar or 1D array.

append: append model informations instead of generating one.

***return*:** a discrentized model - DiscModel instance.

get_efunc_singleband(*scalefunc*, *sgn*, *bandindex=0*, *Nx=500000*, *rk=False*)
get representative Energy function e(x) for specific band and branch -the python version.

scalefunc: a function of discretization points $epsilon(x)$.

sgn: the positive branch if sgn>0, else the negative one.

bandindex: the band index.

Nx: the number of samples in x-space for integration.

rk: use Ronge-Kutta if True(in this version, it's better to set False).

***return*:** a function of representative energy e(x) for specific band and branch.

get_wxfunc_singleband(*scalefunc*, *sgn*, *bandindex=0*)
Get weight function w(x), which is equal to int^e(x)_e(x+1) d(w,bandindex) dw

scalefunc: the function for scale ticks.

sgn: specify the branch.

bandindex: the bandindex.

***return*:** weight of hybridization function w(x) at interval epsilon(x)~epsilon(x+1)

quick_map(*tick_type='log'*, *Nx=500000*)
Perform quick mapping(All in one suit!) for nband x nband hybridization matrix.

tick_type: the type of tick,

- *adaptive* -> adaptive tick,
- *sclog* -> logarithmic ticks suited for superconductor.
- *log* -> logarithmic tick,

Nx: the number of samples for integration over $rho(epsilon(x))$.

***return*:** a super tuple of functions->

(scalefunc_positve,Efunc_positive,Tfunc_positive),(scalefunc_negative,Efunc_negative,Tfunc_negative)

set_rhofunc(*rhofunc*, *Nw=50001*)

rhofunc: the hybridization function.

Nw: the number of ws for rho(w).

unique_token
return a unique token, for saving/loading datas.

class discretization.**DiscModel**(*Elist*, *Tlist*, *z=1.0*)
Bases: object

discrete model.

Elist/Tlist: a list of on-site energies and hopping terms. The shape is (2N,nz,nband,nband)

z: the twisting parameters.

N
number of particles for each branch(positive or negative).

nband
number of bands.

nz
number of twisting parameters.

discretization.**get_scalefunc_adaptive**(*Lambda*, *wlist*, *Rlist*)
get adaptive scale tick function(Ref: Comp. Phys. Comm. 180.1271), this is a numerical stable version even for **Gapped** systems.

Lambda: scaleing factor.

wlist,Rlist: RL is the integration of rho(w) over wlist.

***return*:** a scale function epsilon(x).

discretization.**get_scalefunc_log**(*Lambda*, *D*, *Gap*)
get logarithmic scale tick function. epsilon(x)=Lambda^(2-x)

Lambda: scaling factor.

D/Gap: the bandwidth/Gap range.

***return*:** a scale function epsilon(x).

discretization.**get_scalefunc_sclog**(*Lambda*, *D*, *Gap*)
get logarithmic scale tick function suited for superconductor(logarithmic for normal part).

Lambda: scaling factor.

D/Gap: the bandwidth/Gap range.

***return*:** a scale function epsilon(x).

CHAPTER

TWO

CHAINMAPPER MODULE

Map a discretized model into a Chain by the method of (block-)lanczos tridiagonalization. checking method is also provided.

class chainmapper.**Chain**(*t0*, *elist*, *tlist*)
Bases: object

NRG chain class.

t0: the coupling term of the first site and the impurity.

elist/tlist: a list of on-site energies and coupling terms.

to_scalar()
transform 1-band non-scalar model to scalar one by remove redundant dimensions.

is_scalar
True if this is a model mapped from scalar hybridization function.

nband
number of bands.

class chainmapper.**ChainMapper**(*prec=3000*)
Bases: object

A Chain Model Mapper for NRG.

prec: the precision in mapping process.

check_spec(*chain*, *dischandler*, *rhofunc*, *mode='eval'*, *Nw=500*, *smearing=1.5*)
check mapping quality.

chain: the chain after mapping.

dischandler: discretization handler.

rhofunc: hybridization function.

mode: *eval* -> check eigenvalues *pauli* -> check pauli components

Nw: number of samples in w-space.

map(*model*)
Map discretized model to a chain model using lanczos method.

model: the discretized model(DiscModel instance).

***return*:** a Chain object

chainmapper.**load_chain**(*token*)
load a Chain instance from files.

token: a string as a prefix to store datas of a chain.

***return*:** a Chain instance.

chainmapper.**save_chain**(*token*, *chain*)
save a Chain instance to files.

token: a string as a prefix to store datas of a chain.

chain: a chain instance.

CHAPTER

THREE

TRIDIAGONALIZE MODULE

Tridiagonalization methods for both scalar(tridiagonalize) and block(tridiagonalize_qr) versions. Some test functions are also included.

tridiagonalize.**check_tridiagonalize**(*H0*, *trid*)

check the quality of tridiagonalization.

H0: the original hamiltonian.

trid: tridiagonalization result, a tuple of (data,offset).

tridiagonalize.**tridiagonalize**(*A*, *q*, *m=None*, *prec=None*, *getbasis=False*)

Use m steps of the lanczos algorithm starting with q to generate eigenvalues for the sparse symmetric matrix A.

A: a sparse symmetric matrix.

q: the starting vector.

m: the steps to run.

getbasis: return basis vectors if True.

***return*:** (data,offset,vectorbase)

Use scipy.sparse.diags(res[0],res[1]) to generate a sparse matrix

tridiagonalize.**tridiagonalize_qr**(*A*, *q*, *m=None*, *prec=None*)

Use m steps of the lanczos algorithm starting with q - the block version with QR decomposition.

Note: we need to specify a two-column starting vectors here (q0,q1) with q0,q1 orthogonal to each other.

A: a sparse symmetric matrix.

q: the starting othogonal vector q=(q0,q1).

m: the steps to run.

***return*:** (data,offset), the trdiagonal matrix can be generated by scipy.sparse.diags(data,offset).

CHAPTER

FOUR

UTILS MODULE

Author: Jinguo Leo Date : 8 September 2014 Description : physics utility library

utils.**H2G**(*h*, *w*, *tp='r'*, *geta=0.01*, *sigma=None*)
Get Green's function g from Hamiltonian h.

h: an array of hamiltonian.

w: the energy(frequency).

tp: the type of Green's function. 'r': retarded Green's function.(default) 'a': advanced Green's function. 'matsu': finite temperature Green's function.

geta: smearing factor. default is 1e-2.

sigma: additional self energy.

***return*:** a Green's function.

utils.**eigh_pauliv_npy**(*a0*, *a1*, *a2*, *a3*)
eigen values for pauli vectors - numpy version.

a0/a1/a2/a3: pauli components.

***return*:** (evals,evecs)

utils.**mpconj**(*A*)
get the conjugate of matrix A(to avoid a bug of gmpy2.mpc.)

A: the input matrix.

***return*:** matrix with the same dimension as A

utils.**ode_ronge_kutta**(*func*, *y0*, *tlist*, ***kwargs*)
Integrate use Ronge Kutta method.

func: the function of (x,y).

y0: the starting y.

tlist: a list of t.

****kwargs:** additional arguments for scipy.ode.set_integrator

***return*:** return integrated array(like cumtrapz).

utils.**qr2**(*A*)
analytically, get the QR decomposition of a matrix.

A: the matrix.

***return*:** (Q,R), where QR=A

utils.**s2vec**(*s*)

Transform a 2 x 2 matrix to a 4 dimensional vector, corresponding to s0,sx,sy,sz component.

s: the matrix.

utils.**vec2s**(*n*)

Transform a vector of length 3 or 4 to a pauli matrix.

n: a 1-D array of length 3 or 4 to specify the *direction* of spin.

***return*:** 2 x 2 matrix.

CHAPTER

FIVE

HYBRI_SC MODULE

Get the hybridization function of a conventional superconductor. wide-band approximation(get_hybri_wideband) version and finite-band width(get_hybri) version are available.

hybri_sc.**get_hybri**(*Gap*, *Gamma*, *D0=1.0*, *mu=0.0*, *eta=1e-10*)

D(w) for superconducting surface.

Gap: the Gap value.

Gamma: the overall strength.

D0: the band width of normal part.

mu: chemical potential.

eta: the smearing factor.

***return*:** hybridization function for superconductor.

hybri_sc.**get_hybri_wideband**(*Gap*, *Gamma*, *D=1.0*, *mu=0.0*, *eta=1e-10*)

D(w) for superconducting surface, taking wide-band approximation.

Gap: the Gap value.

Gamma: the overall strength.

D: the band width.

mu: chemical potential.

eta: the smearing factor.

***return*:** hybridization function for superconductor with wideband approximation.

c

d

h

t

u