PyMPS Documentation

Release 1.0.0

Jinguo Leo

Dec 03, 2016

CONTENTS

1 Authors **1**

2 Contents **2**
- 2.1 Using the code . . . 2
- 2.2 Tutorial . . . 2
- 2.3 API . . . 3

Python Module Index **18**

Index **19**

CHAPTER

ONE

AUTHORS

- JinGuo Leo (NJU)

Source code: https://github.com/GiggleLiu/mpslib

CHAPTER

TWO

CONTENTS

2.1 Using the code

The requirements are:

- Python 2.6 or higher
- numpy and scipy
- matplotlib for plotting
- tba
- blockmatrix

It is recommended to use Anaconda to install these packages(except tba and blockmatrix).

Download the code using the Download ZIP button on github, or run the following command from a terminal:

```
$ wget -O mpslib-master.zip https://github.com/GiggleLiu/mpslib/archive/master.zip
```

Within a terminal, execute the following to unpack the code:

```
$ unzip mpslib-master.zip
$ cd mpslib-master/
$ (sudo) python setup.py install
```

The first program, for instance, can be run by issuing:

```
$ cd mps/
$ python sample_tensor.py
$ python sample_mps.py
$ python sample_bmps.py
```

2.2 Tutorial

2.2.1 Using the tensor

First, I will give a brief introduction to tensors, I will show a rather simple example to explain how it works, it looks like:

Tensors are always labeled, the labels of a tensor could be a str, or it's subclass <BLabel>, labels are the clue for any contraction(einsum). <BLabel> has a member bm(<BlockMarker> instance), when a tensor use <BLabel>s as the labels, we say it is block structured.

The first thing we can do with tensors is to build a MPS. Here I will show you a piece of code to calculate correlation functions $\langle O(i)O(j)\rangle$.

Each cell of MPS is a 3D tensor, it's axes take the meaning of (llink,site,rlink), the center axis is the physical site and llink, rlink are the connector to neighboring cells. The MPS used here is a standard l-canonical one, the l can be changed by canonical move(method <MPS>.canomove). Compressing can be done through consequtive canonical move with tolerence and maximum bond dimension specified. On the other side, a lot of utilities can be used to construct operators, 3 elemental types of operators <OpUnit>,<OpString> and <OpCollection> are able to construct a readable form of any operators. e.g. Sx(0) is an <OpUnit>, Sx(0)*Sx(1)*Sy(3) is an <OpString> and Sx(3)+Sy(2)*Sz(5) is an <OpCollection>, the later the more complex. function get_expect could be used to cope the expectation problem with all these three. Besides, we can use <OpCollection>.toMPO method to construct an <MPO> easily, which will be shown below.

In many cases, the block marker is closely related to good quantum number, in this sample case, it is 'M'. If we define a direction on each labels(axis) to indicate the flow of good quantum number. A tensor is 'balanced' if the 'net flow' of quantum number on each non-zero element is 0. The condition of balance is extremely useful in the definition of MPO and MPS with good quantum number.

In this case, the block markers of cell labels the good quantum number in the left blocks. If we asign a direction to a cell, it will be (1,1,-1), which means (in,in,out). We see that all tensors in a <BMPS> or <BMPO> must be balanced! This is why we are able to set the block markers automatically with only the knowlege about good quantum number and tensor datas.

2.3 API

2.3.1 Tensor

TensorBase and dense Tensor Class.

class `tensor.`**`Tensor`**

Bases: `numpy.ndarray`, `tensor.TensorBase`

Tensor class subclassing ndarray, with each dimension labeled by a string(or BLabel).

Construct:

Tensor(shape,labels,kwargs):** Create a <Tensor> with random data with specified shape.

Tensor(array,labels,kwargs):** Create a <Tensor> converted from array.

Attributes:

labels list, the labels of axes.

refer numpy.ndarray for more details.

`make_copy`(*labels=None*, *copydata=True*)

Make a copy of this tensor.

Parameters:

labels list/None, the new labels, use the old ones if None.

copydata bool, copy the data to the new tensor if True.

Return: <TensorBase>

`take`(*key*, *axis*, *useqn=False*)

Take subspace from this Tensor.

Parameters:

key 0-d/1-d array, the key

axis int, the axis to take.

useqn bool, use label to specify the bond if True.

Return: <TensorBase>

chorder(*order*)
Reorder the axes of this tensor.

Parameters:

order tuple, the new order of the axes.

Return: <TensorBase>

mul_axis(*vec*, *axis*)
Multiply a vector on specific axis.

Parameters:

vec 1d array, the vector.

axis int/str, the axis or label.

Return: <TensorBase>

merge_axes(*sls*, *nlabel=None*, *signs=None*, *bmg=None*)
Merge multiple axes into one.

Parameters:

sls slice, the axes to merge.

nlabel str/None, the new label, addition of old labels if None.

signs 1darray, the signs(flow direction) for each merged block marker.

Return: <TensorBase>

split_axis(*axis*, *dims*, *nlabels*)
Split one axis into multiple.

Parameters:

axis int/str, the axes to merge.

dims tuple, the new dimensions, prod(dims)==self.shape[axis].

nlabels list, the new labels.

Return: <TensorBase>

get_block(*block*, *useqn=False*)
Query data in specific block.

Parameters:

block tuple, the target block.

useqn bool, use label as the block indexer.

Return: ndarray, the data.

b_reorder(*axes=None*, *return_pm=False*)
Reorder rows, columns to make tensor blocked.

Parameters:

axes tuple, the target axes to be reorderd

return_pm bool, return the permutation series.

Return: <Tensor>.

class tensor.**BLabel**

Bases: str

Label string with block marker.

Attibutes:

bm <BlockMarker>,

chbm(*bm*)

Get a new <BLabel> with different block marker.

chstr(*s*)

Get a new <BLabel> with different string.

2.3.2 Tensor Related Functions

tensor.**tdot**(*tensor1*, *tensor2*)

Tensor dot between two tensors, faster than contract in most case?

Parameters:

tensor1,tensor2 <Tensor>, two tensors.

Return: <Tensor>,

tensorlib.**random_tensor**(*shape=None*, *labels=None*)

Generate a random Tensor.

Parameters:

shape the shape of tensor.

labels the labels of axes.

Return: <Tensor>

tensorlib.**random_btensor**(*bms*, *label_strs=None*, *fill_rate=0.2*)

Generate a random Block Dense Tensor.

Parameters:

bms list, the block markers.

label_strs list/None, the labels.

fill_rate propotion of the number of filled blocks.

Return: <Tensor>

tensorlib.**random_bdmatrix**(*bm=None*, *dtype='complex128'*)

Generate a random Block Diagonal 2D Tensor.

Parameters:

bm <BlockMarker>

Return: <Tensor>,

tensorlib.**check_validity_tensor**(*ts*)

Check if it is a valid tensor.

tensorlib.**contract**(**tensors*)
Contract a collection of tensors

Parameters:

tensors <Tensor>s, A list of <Tensor> instances.

Return: <Tensor>

tensorlib.**svdbd**(*A*, *cbond_str='X'*)
Get the svd decomposition for dense tensor with block structure.

Parameters:

A 2D<Tensor>, the input matrix, with <BLabel>s.

cbond_str the labes string for center bond.

Return: (U,S,V) that U*S*V = A

2.3.3 Matrix Product State

Matrix Product State.

class mps.**MPS**(*ML, l, S, is_ket=True, labels=['s', 'a']*)
Bases: mps.MPSBase

Matrix product states.

Attributes:

ML list of 3D array, the sequence of A/B-matrices.

l/S int, 1D array, the division point of left and right scan, and the singular value matrix at the division point. Also, it is the index of the non-unitary M-matrix(for the special case of l==N, S is attached to the right side of N-1-th matrix)

is_ket bool, It is a ket if True else bra.

labels len-2 list of str, the labels for auto-labeling in MPS [site, link].

hndim
The number of state in a single site.

nsite
int, number of sites.

state
1d array, vector representation of this MPS

get(*siteindex*, *attach_S=''*, **args*, ***kwargs*)
Get the tensor element for specific site.

Parameters:

siteindex int, the index of site.

attach_S bool, attach the S-matrix to

- '' -> don't attach to any block.
- 'A' -> right most A block.
- 'B' -> left most B block.

Return: <Tensor>,

set (*siteindex*, *A*, **args*, ***kwargs*)
Get the matrix for specific site.

Parameters:

siteindex int, the index of site.

A <Tensor>, the data

check_link (*l*)
The bond dimension for l-th link.

Parameters:

l int, the bond index.

Return: int, the bond dimension.

get_all (*attach_S=''*)
Get the concatenation of A and B sectors.

Parameters:

attach_S bool, attach the S-matrix to

- '' -> don't attach to any block.
- 'A' -> right most A block.
- 'B' -> left most B block.

Return: list,

canomove (*nstep*, *tol=1e-12*, *maxN=inf*)
Move l-index by one with specific direction.

Parameters:

nstep int, move l nstep towards right.

tol float, the tolerence for compression.

maxN int, the maximum dimension.

Return: float, approximate truncation error.

use_bm (*bmg*, *sharedata=True*)
Use <BlockMarker> to indicate block structure.

Parameters:

bmg <BlockMarkerGenerator>, the generator of block markers.

sharedata bool, the new <BMPS> will share the data with current one if True.

Return: <BMPS>,

toket (*labels=None*)
Get the ket conterpart.

Parameters:

labels list, label strings for site and bond.

Return: <MPS>,

tobra (*labels=None*)
Get the bra conterpart.

Parameters:

labels list, label strings for site and bond.

Return: <MPS>,

tovidal()
Transform to the Vidal form.

chlabel(*labels*)
Change the label of specific axis.

Parametrs:

labels list, the new labels.

query(*serie*)
Query the magnitude of a state.

Parameters:

serie 1d array, sgimas(indices of states on sites).

Return: number, the amplitude.

compress(*tol=1e-08*, *maxN=200*, *niter=3*)
Compress this state.

Parameters:

tol float, the tolerence used for compressing.

maxN int, the maximum retained states.

niter int, number of iterations.

Return: float, approximate truncation error.

recanonicalize(*left=True*, *tol=1e-12*, *maxN=inf*)
Trun this MPS into canonical form.

Parameters:

left bool, use left canonical if True, else right canonical

tol float, the tolerence.

maxN int, the maximum retained states.

llink_axis
int, axis of left link index.

rlink_axis
int, axis of right link index.

site_axis
int, axis of site index.

class mps.**BMPS**(*ML*, *l*, *S*, *bmg*, *bms=None*, ***kwargs*)
Bases: *mps.MPS*

MPS with block structure.

Attributes:

bmg <BlockMarkerGenerator>, the block infomation manager.

see <MPS> for more.

unuse_bm(*sharedata=True*)
Get the non-block version of current ket.

Parameters:

sharedata bool, the new <MPS> will share the data with current one if True.

Return: <MPS>,

2.3.4 MPS Related Functions

mpslib.**state2MPS**(*state*, *sitedim*, *l*, *method='qr'*, *tol=1e-08*, *labels=('s', 'a')*)
Parse a normal state into a Matrix produdct state.

state: The target state, 1D array.

sitedim: The dimension of a single site, integer.

l: The division point of left and right canonical scanning, integer between 0 and number of site.

method: The method to extract A,B matrices. * 'qr' -> get A,B matrices by the method of QR decomposition, faster, rank revealing in a non-straight-forward way. * 'svd' -> get A,B matrices by the method of SVD decomposition, slow, rank revealing.

tol: The tolerence of singular value, float.

labels: (label_site,label_link), The labels for degree of freedom on site and intersite links.

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

mpslib.**mps_sum**(*mpses*, *labels=('s', 'a')*, *maxN=None*)
Summation over <MPS>es.

Parameters:

mpses list of <MPS>, instances to be added.

labels list of str, the new labels for added state.

Return: <MPS>, the added MPS.

mpslib.**random_mps**(*hndim=2*, *nsite=10*, *maxN=50*)
Random <MPS>.

Parameters:

hndim int, the single site Hilbert space dimension.

nsite int, the number of sites.

maxN int, the maximum bond dimension.

Return: <MPS>,

mpslib.**random_bmps**(*bmg*, *nsite*, *maxN=50*)
Random <BMPS>.

Parameters:

bmg <BlockMarkerGenerator>,

nsite int, the number of sites.

maxN int, the maximum bond dimension.

Return: <BMPS>,

mpslib.**product_state**(*config*, *hndim=None*, *bmg=None*)
Generate a product state

Parameters:

config 1Darray/2Darray, for 1D array, it is the occupied single site index, for 2D array, it's the wave functions for each site.

hndim int, the site dimension(needed if config is 1D).

bmg <BlockMarkerGenerator>,

Return: <MPS>, right canonical.

mpslib.**random_product_state**(*nsite*, *hndim*)
Generate a random product state.

Parameters:

nsite/hndim int, the number of site, the dimension of single site.

Return: <MPS>, right canonical.

mpslib.**check_validity_mps**(*mps*)
Check the validity of mps, mainly the bond dimension check.

Parameters: <MPS>,

Return: bool, True if it is a valid <MPS>.

mpslib.**check_flow_mpx**(*mpx*)
Check the quantum number flow of mpx.

Parameters:

mpx <BMPS>/<BMPO>,

Return: bool, true if the flow is quantum number conserving.

mpslib.**check_canonical**(*ket*, *tol=1e-08*)
Check if a MPS meets some canonical condition.

Parameters:

ket <MPSBase>,

tol float, the tolerence.

Return: bool, true if this MPS is canonical.

2.3.5 Operator Representations

Matrix Product State.

class mpo.**OpUnit**(*label*, *data*, *siteindex='-'*, *factor=1.0*, *math_str=None*, *fermionic=False*)
Bases: object

Site-wise Operator, the basic element in constructing Operators.

Attributes:

label string, the label.

data matrix, the data of this Operator unit.

siteindex integer, the site index, or leave '-' to be un specified.

factor number, the factor acting on data.

math_str string/None, the string for mathematical display.

fermionic bool, this is a fermionic operator(sign problem arises) or not.

Readonly Attributes:

hndim int, single site Hilbert space dimenion.

siteindices list, siteindices interface to be compatible with opstring.

opunits list, opunits interface to be compatible with opstring.

hndim
The dimension of Hilbertspace at single site.

maxsite
int, the maximum refered site.

H(*nsite*)
Get the Hamiltonian matrix.

Parameters:

nsite int, number of sites.

Return: matrix, the operator.

as_site(*i*)
Get a copy of this operator at site i.

get_mathstr(*factor=1.0*)
Get the math string for display.

get_data(*dense=True*)
Get the data(taking factor into consideration).

Parameters:

dense bool, dense or not.

Return: matrix, the data.

toMPO(*nsite*)
Turn to MPO format with bond dimension 1.

Parameters:

nsite int, number of sites.

Return: <MPO>,

class mpo.**OpUnitI**(*hndim*, *siteindex='-'*)
Bases: *mpo.OpUnit*

The single site Unitary operator.

class mpo.**OpString**(*opunits=None*)
Multiplication serie of operator units. e.g. Sx(i)*Sx(j).

Attributes:

opunits list, a list of <OpUnit> instances.

Readonly Attributes:

hndim int, single site Hilbert space dimenion.

nunit number of constructing <OpUnit>s

siteindices 1d array, the site indices,

fermionic bool, this is a fermionic operator or not.

siteindices
The site indices.

maxsite
int, the maximum refered site.

H(*nsite*)
Get the Hamiltonian matrix.

Parameters:

nsite int, number of sites.

Return: matrix, the operator.

query(*i*)
Query the specific opunits.

Parameters:

i int, the site index.

Return: list, the <OpUnit>s.

get_mathstr(*factor=1.0*)
Get the math string.

toMPO(*nsite*)
Turn to MPO format with bond dimension 1.

Parameters:

nsite int, number of sites.

Return: <MPO>,

class mpo.**OpCollection**(*ops=None*)
Bases: object

Addition of serie of operator strings/units. e.g. S(i)*S(j)+S(k)*S(l)+S(m)

Attributes:

ops list, a list of <OpString> instances.

Readonly Attributes:

hndim int, single site Hilbert space dimenion.

nop number of constructing <OpUnit>s/<OpUnit>s.

maxsite
int, the maximum refered site.

H(*nsite=None*)
Get the matrix hamiltonian representation.

Parameters:

nsite int, number of sites.

Return: matrix, the operator.

query(**indices*)
Query Operators linking sites i, j...

Parameters:

indices integer, the site indices.

Return: list, a list of operator string.

filter(*func*)
Query Operators meets condition function func.

Parameters:

func function, the condition function on site indices.

Return: list, a list of operator string.

toMPO(*method='direct'*, *nsite=None*, *bmg=None*)
Construct Matrix product Operator corresponding to this <OpCollection> instance.

Parameters:

method str, 'direct' or 'addition'.

- direct: construct directly using complicated manipulation.
- additive: construct by <MPO> addition operations term by term.

nsite int, number of sites.

bmg <BlockMarkerGenerator>,

Return: <MPO>(<BMPO> if bmg is not None).

class mpo.**MPO**(*OL, labels=['m', 's', 'b']*)
Bases: object

Matrix product operator.

Attributes:

labels len-3 list, the labels for legs, in the order site-site-link.

OL list, the Tensor form of MPO datas.

hndim integer, the Hilbert space dimension of single site.

nsite integer, the number of sites.

hndim
The number of state in a single site.

nsite
Number of sites.

H
Get the Hamiltonian.

use_bm(*bmg*)
Use <BlockMarker> to indicate block structure.

Parameters:

bmg <BlockMarkerGenerator>, the generator of block markers.

Return: <BMPS>,

get(*i*, **args*, ***kwargs*)
Get the operator tensor at specific site.

Parameters:

i int, the site index.

Return: 4-leg <Tensor>,

set(*i*, *A*, **args*, ***kwargs*)
Get the matrix for specific site.

Parameters:

i int, the index of site.

A <Tensor>, the data

check_link(*l*)
The bond dimension for l-th link.

Parameters:

l int, the bond index.

Return: int, the bond dimension.

chlabel(*labels*)
Change the label of specific axis.

Parameters:

labels list, the new labels. * 'site1' -> The on-site freedom, the first dimension. * 'site2' -> The on-site freedom, the second dimension. * 'link' -> The label of the links.

compress(*niter=2*, *tol=1e-12*, *maxN=inf*)
Move l-index by one with specific direction.

Parameters:

niter int, number of iteractions.

tol float, the tolerence for compression.

maxN int, the maximum dimension.

Return: float, approximate truncation error.

class mpo.**BMPO**(*OL, bmg, labels=['m', 's', 'b']*)
Bases: *mpo.MPO*

Matrix product operator.

Attributes:

labels len-3 list, the labels for legs, in the order site-site-link.

WL list, the Matrix product operator datas.

OL list, the matrix(Tensor) form of MPO datas.

hndim integer, the Hilbert space dimension of single site.

nsite integer, the number of sites.

unuse_bm()
Get the non-block version <MPO>.

2.3.6 MPO Related Functions

mpo.**WL2MPO**(*WL, labels=['m', 's', 'b'], bmg=None*)
Construct MPO from WL.

mpo.**WL2OPC**(*WL, fix_sites=True*)
Return The serialized form of operator.

mpolib.**opunit_S**(*spaceconfig*, *which*, *siteindex='-'*)
Get S_? operator unit.

Parameters: spaceconfig: <SuperSpaceConfig>/<SpinSpaceConfig>, the space configuration. which: char, specify the *?*.

Return: <OpUnit>

mpolib.**opunit_Sx**(*spaceconfig*)

mpolib.**opunit_Sy**(*spaceconfig*)

mpolib.**opunit_Sz**(*spaceconfig*)

mpolib.**opunit_Sp**(*spaceconfig*)

mpolib.**opunit_Sm**(*spaceconfig*)

mpolib.**opunit_C**(*spaceconfig*, *index*, *dag*, *siteindex='-'*)
Get creation and annilation operator units.

Parameters:

spaceconfig <SuperSpaceConfig>/<SpinSpaceConfig>, the space configuration.

index integer, the flavor of electron.

dag bool, creation or not(annilation).

Return: <OpUnit>

mpolib.**opunit_c**(*spaceconfig*, *index*)

mpolib.**opunit_Z**(*spaceconfig*, *siteindex='-'*)
Get fermionic parity operator units.

Parameters:

spaceconfig <SuperSpaceConfig>, the space configuration.

Return: <OpUnit>, the parity operator.

mpolib.**opunit_cdag**(*spaceconfig*, *index*)

mpolib.**opunit_N**(*spaceconfig*, *index=None*, *siteindex='-'*)
Get pariticle number operator.

Parameters:

spaceconfig <SuperSpaceConfig>, the space configuration.

index int/None, the index of flavor, None for all.

Return: <OpUnit>, the particle number operator.

mpolib.**insert_Zs**(*op*, *spaceconfig*)
Insert fermionic signs between ferminonic operators.

Parameters:

spaceconfig <SpaceConfig>, the configuration of hilbert space.

mpolib.**random_mpo**(*hndim=2*, *nsite=10*, *maxN=6*, *hermitian=True*)
Random <MPO>.

Parameters:

hndim int, the single site hilbert space dimension.

nsite int, number of sites.

maxN int, the maximum bond dimension.

hermitian bool, get a hermitian MPO if True.

Return: <MPO>,

mpolib.**random_bmpo**(*bmg*, *nsite=10*, *maxN=6*)
Random <BMPS>.

Parameters:

nsite int, number of sites.

bmg <BlockMarkerGenerator>,

maxN int, the maximum bond dimension.

Return: <BMPO>,

mpolib.**check_validity_mpo**(*mpo*)
check the validity of mpo.

Parameters:

mpo <MPO>,

Return: bool,

mpolib.**mpo_sum**(*mpos, labels=['m', 's', 'b']*)

Parameters:

mpos list, list of <MPO>s.

labels list, list of string as new labels(site-up,site-down,link).

Return: <MPO>,

2.3.7 Contraction of MPO and MPS

Contraction Handler classed for MPS and MPO.

class contraction.**Contractor**(*mpo*, *ket*, *bra_bond_str='t'*)
Bases: object

Contraction handler.

Attributes:

mpo/ket <MPO>/<MPS>,the operator and ket.

bra_labels list, the labels for bra,

LPART/RPART list of <Tensor>, the result of contraction from left/right.

Readonly Attributes:

bra <MPS>, bra is the hermitian conjugate of ket.

lupdate(*i*)
Update LPARTs.

Parameters:

i int, the target length to update.

rupdate(*i*)
Update RPARTs.

Parameters:

i int, the target length to update.

evaluate()
Get the expectation value of mpo.

show(*space=2*)
Show the contraction Graphically.

contract2l()
Contract all available LPART and RPART.

contraction.**get_expect**(*op*, *ket*, *bra=None*, *sls=slice(None*, *None*, *None)*, *memorial=False*)
Get the expectation value of an operator.

Parameters:

op <OpUnit>/<OpString>/<OpCollection>, the operator.

ket <MPS>, the ket.

bra <MPS>/None, the bra, "same" as ket if is *None*.

sls slice, the interval to perform contraction.

memorial bool, return contraction history if True.

Return: number, the expectation value of operator on this ket.

c

contraction, 16

m

mpo, 10
mps, 6

t

tensor, 3

A

as_site() (mpo.OpUnit method), 11

B

b_reorder() (tensor.Tensor method), 4
BLabel (class in tensor), 5
BMPO (class in mpo), 14
BMPS (class in mps), 8

C

canomove() (mps.MPS method), 7
chbm() (tensor.BLabel method), 5
check_canonical() (in module mpslib), 10
check_flow_mpx() (in module mpslib), 10
check_link() (mpo.MPO method), 14
check_link() (mps.MPS method), 7
check_validity_mpo() (in module mpolib), 16
check_validity_mps() (in module mpslib), 10
check_validity_tensor() (in module tensorlib), 5
chlabel() (mpo.MPO method), 14
chlabel() (mps.MPS method), 8
chorder() (tensor.Tensor method), 4
chstr() (tensor.BLabel method), 5
compress() (mpo.MPO method), 14
compress() (mps.MPS method), 8
contract() (in module tensorlib), 5
contract2l() (contraction.Contractor method), 17
contraction (module), 16
Contractor (class in contraction), 16

E

evaluate() (contraction.Contractor method), 17

F

filter() (mpo.OpCollection method), 13

G

get() (mpo.MPO method), 13
get() (mps.MPS method), 6
get_all() (mps.MPS method), 7
get_block() (tensor.Tensor method), 4
get_data() (mpo.OpUnit method), 11
get_expect() (in module contraction), 17
get_mathstr() (mpo.OpString method), 12
get_mathstr() (mpo.OpUnit method), 11

H

H (mpo.MPO attribute), 13
H() (mpo.OpCollection method), 12
H() (mpo.OpString method), 12
H() (mpo.OpUnit method), 11
hndim (mpo.MPO attribute), 13
hndim (mpo.OpUnit attribute), 11
hndim (mps.MPS attribute), 6

I

insert_Zs() (in module mpolib), 15

L

llink_axis (mps.MPS attribute), 8
lupdate() (contraction.Contractor method), 17

M

make_copy() (tensor.Tensor method), 3
maxsite (mpo.OpCollection attribute), 12
maxsite (mpo.OpString attribute), 12
maxsite (mpo.OpUnit attribute), 11
merge_axes() (tensor.Tensor method), 4
MPO (class in mpo), 13
mpo (module), 10
mpo_sum() (in module mpolib), 16
MPS (class in mps), 6
mps (module), 6
mps_sum() (in module mpslib), 9
mul_axis() (tensor.Tensor method), 4

N

nsite (mpo.MPO attribute), 13
nsite (mps.MPS attribute), 6

O

OpCollection (class in mpo), 12
OpString (class in mpo), 11
OpUnit (class in mpo), 10

opunit_C() (in module mpolib), 15
opunit_c() (in module mpolib), 15
opunit_cdag() (in module mpolib), 15
opunit_N() (in module mpolib), 15
opunit_S() (in module mpolib), 15
opunit_Sm() (in module mpolib), 15
opunit_Sp() (in module mpolib), 15
opunit_Sx() (in module mpolib), 15
opunit_Sy() (in module mpolib), 15
opunit_Sz() (in module mpolib), 15
opunit_Z() (in module mpolib), 15
OpUnitI (class in mpo), 11

P

product_state() (in module mpslib), 9

Q

query() (mpo.OpCollection method), 12
query() (mpo.OpString method), 12
query() (mps.MPS method), 8

R

random_bdmatrix() (in module tensorlib), 5
random_bmpo() (in module mpolib), 16
random_bmps() (in module mpslib), 9
random_btensor() (in module tensorlib), 5
random_mpo() (in module mpolib), 16
random_mps() (in module mpslib), 9
random_product_state() (in module mpslib), 10
random_tensor() (in module tensorlib), 5
recanonicalize() (mps.MPS method), 8
rlink_axis (mps.MPS attribute), 8
rupdate() (contraction.Contractor method), 17

S

set() (mpo.MPO method), 14
set() (mps.MPS method), 6
show() (contraction.Contractor method), 17
site_axis (mps.MPS attribute), 8
siteindices (mpo.OpString attribute), 12
split_axis() (tensor.Tensor method), 4
state (mps.MPS attribute), 6
state2MPS() (in module mpslib), 9
svdbd() (in module tensorlib), 6

T

take() (tensor.Tensor method), 3
tdot() (in module tensor), 5
Tensor (class in tensor), 3
tensor (module), 3
tobra() (mps.MPS method), 7
toket() (mps.MPS method), 7
toMPO() (mpo.OpCollection method), 13
toMPO() (mpo.OpString method), 12
toMPO() (mpo.OpUnit method), 11
tovidal() (mps.MPS method), 8

U

unuse_bm() (mpo.BMPO method), 14
unuse_bm() (mps.BMPS method), 8
use_bm() (mpo.MPO method), 13
use_bm() (mps.MPS method), 7

W

WL2MPO() (in module mpo), 15
WL2OPC() (in module mpo), 15