Uni10-Tutorials Documentation

Release 1.0

Ying-Jer Kao

Nov 04, 2019

Contents

Here you can find

- **pyUni10**: Python wrappers for Uni10.
- **Lectures**: iPython notebooks for lectures.
- Reference Manual for pyUni10.

Uni10 homepage

CHAPTER 1

Setting Up pyUni10

1.1 Preparing Your System

You need to have Python 2 installed on your computer.

- If you are using Linux or MacOS X, Python 2 comes with the system.
- For the lectures, you also need to install iPython.
- As an integrated distribution, we recommend the Anaconda Scientific Python Distribution.

1.2 Library Dependencies

pyUni10 requires BLAS and LAPACK libraries to run.

- For Linux users, you need to install BLAS and LAPACK.
- For Mac OS X users, BLAS and LAPACK come with the OS, so you don't have to do anything.
- For Windows users, required libraries are supplied in the pyUni10 package.

1.3 Obtaining pyUni10

Please download pyUni10 for your architecture:

- Windows : `pyUni10-1.0.0.Win.zip`
- Mac OS X: `pyUni10-1.0.0.OSX.tar.gz`
- Linux : `pyUni10-1.0.0.Linux.tar.gz`

Open a shell terminal, move to the pyUni10 directory, and issue the command::

```
python setup.py install
```

On Windows, open a command prompt window and issue the command::

```
python.exe setup.py install
```

Note: Depending on the location of your Python installation, you might need the administrator privilege to install.

1.4 Runnning Tests

Download the test package `pyUni10test.tgz` or `pyUni10test.zip`

In the `test` folder, you should see the following files

- egB1.py
- egM1.py
- egM2.py
- egQ1.py
- egU1.py
- egU2.py
- egU3.py
- egN1.py
- egN1_network
- TestUni10.ipynb

Manually run the Python codes in the sequence listed above, or use the `TestUni10.ipynb`.

If you don't see any error messages, **congratulations**, you are ready to go!!!

CHAPTER 2

Tutorial 1

2.1 Basic pyUni10

The goal of this part of the tutorial is to familarize the user with basic functionality of pyUni10. The purpose is just to get you familiar with how things are done with pyUni10 in the code before moving to more complicated algorithms.

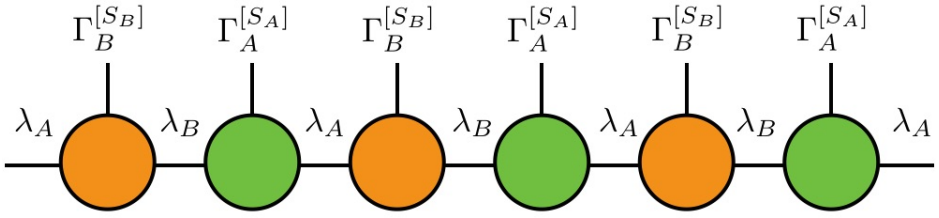
You can download the ipython notebook for this part of the tutorial: `Tutorial1-1.ipynb`.

2.2 iTEBD

We will implement the infinite time evolution bond decimation (iTEBD) method for the 1D transverse Ising model

$$H = \sum_{\langle ij \rangle} \sigma_i^z \sigma_j^z + h \sum_i \sigma_i^x$$

We will use the infinite Matrix Product State, represented in the conanical form as :



where $\Gamma^{[A]s_A}$ and $\Gamma^{[A]s_B}$ are $\chi \times \chi$ matrices on the A and B sublattices, and λ^A and λ^B are Schmidt values.

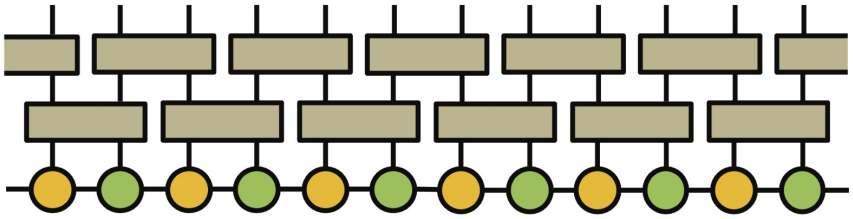
To update the matrices, we use the imaginary time evolution

$$|\Psi_\tau\rangle = \frac{\exp(-H\tau)|\Psi_0\rangle}{\|\exp(-H\tau)|\Psi_0\rangle\|}.$$

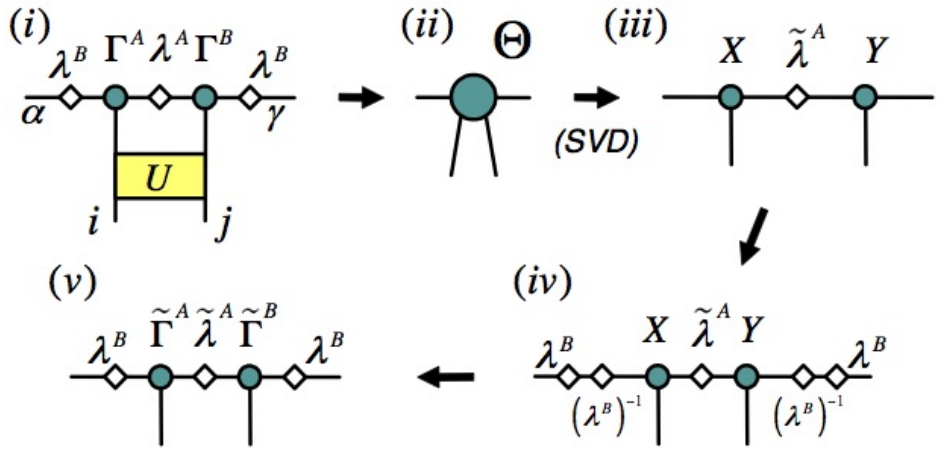
Using Suzuki-Trotter decomposition, the two-site imaginary time evolution operator is given by

$$U^{[r,r+1]} \equiv \exp(-ih^{[r,r+1]}\delta t), \quad \delta t \ll 1,$$

To perform updates, we separate the Hamiltonian into h_{AB} and h_{BA} and apply U_{AB} and U_{BA} alternatively.



The update process is carried out by contracting U_{AB}, and perform SVD of the resulting tensor Θ to obtain new Γ_A, Γ_B, and λ_A



and also for U_{BA}

We repeat the process until the state is converged.

You can download the ipython notebook for this part of the tutorial: `Tutorial1-2.ipynb`.

2.3 References

2.3.1 Martix Procut States

General references on matrix product states, tensor network states

1. Ulrich Schollwoeck, The density-matrix renormalization group in the age of matrix product states, Annals of Physics 326, 96 (2011) , http://arxiv.org/abs/1008.3477
2. R. Orús, A Practical Introduction to Tensor Networks: Matrix Product States and Projected Entangled Pair States, Annals of Physics 349 117 (2014), http://arxiv.org/abs/1306.2164

2.3.2 iTEBD

1. G. Vidal, Classical Simulation of Infinite-Size Quantum Lattice Systems in One Spatial Dimension, Phys. Rev. Lett. 98, 070201 (2007) , http://arxiv.org/abs/cond-mat/0605597
2. R. Orús and G. Vidal, Infinite time-evolving block decimation algorithm beyond unitary evolution, Phys. Rev. B 78, 155117 (2008) , http://arxiv.org/abs/0711.3960

CHAPTER 3

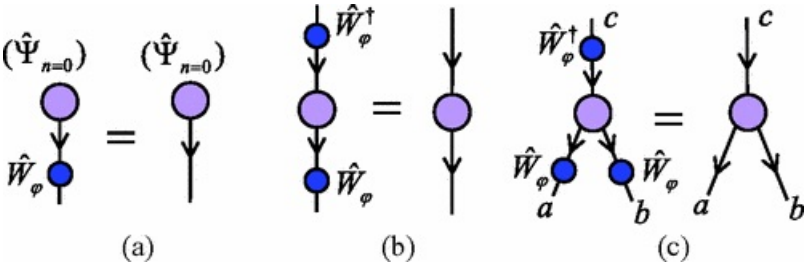
Tutorial 2

In this lecture, we will use pyUni10 to implement the iTEBD algorithm with **U(1)** symmety, and use it to study the Heisenberg model.

3.1 Symmetric Tensors

We will extend our discussion on pyUni10 to construct tensors with **U(1)** symmetry.

A symmetric tensor can be regarded as a tensor with bonds carrying quantum numbers.

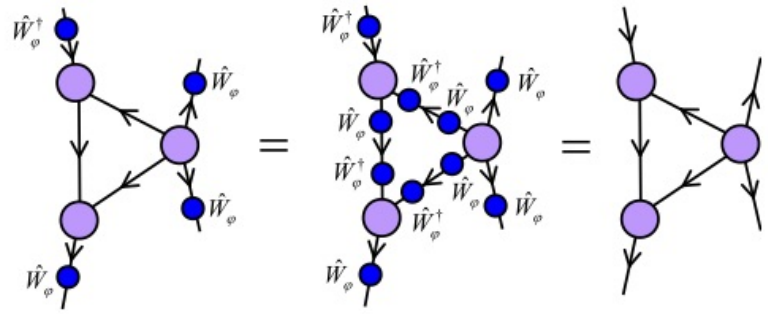


For U(1) invariant tensors, the total U(1) quantum number of the tensor is zero. That is, the U(1) quantum numbers corresponding to incoming and outgoing indices,

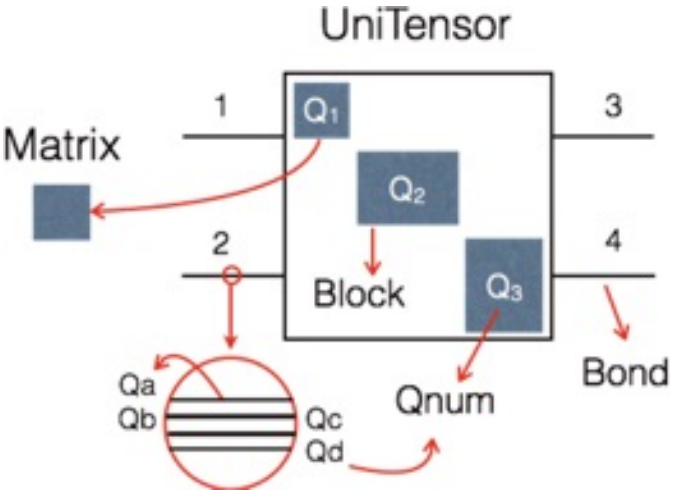
$$N_{\text{in}} = \sum_{n_l \in I} n_l, \quad N_{\text{out}} = \sum_{n_l \in O} n_l,$$

should be equal.

One can form a U(1) invariant tensor network made of U(1) invariant tensors



In pyUni10, a U(1) symmetric tensor is defined as a UniTensor object, which consists of bonds carrying quantum numbers.



You can download the ipython notebook for this part of the tutorial: `Tutorial2-1.ipynb`.

3.2 iTEBD with U(1) Symmetry

We will use iTEBD with U(1) symmetry to study the 1D Heisenberg model,

$$H = \sum_{\langle ij \rangle} \mathbf{S}_i \cdot \mathbf{S}_j$$

You can download the ipython notebook for this part of the tutorial: `Tutorial2-2.ipynb`.

3.3 References

3.3.1 Tensor Network with U(1) Symmetry

1. Sukhwinder Singh, Robert N. C. Pfeifer, and Guifre Vidal, Tensor network states and algorithms in the presence of a global U(1) symmetry, Phys. Rev. B 83 115125 (2011), http://arxiv.org/abs/1008.4774

CHAPTER 4

Tutorial 3

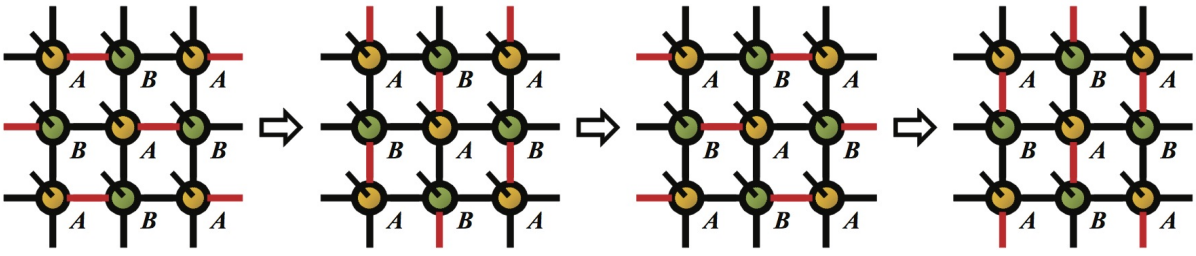
In this part of the tutorial, we will extend the ideas in one dimension to two dimensions.

4.1 2D iTEBD

We will construct a 2D version of imaginary-time iTEBD and apply it to obtain the ground state wave functions in the 2D transverse field Ising model (TFIM).

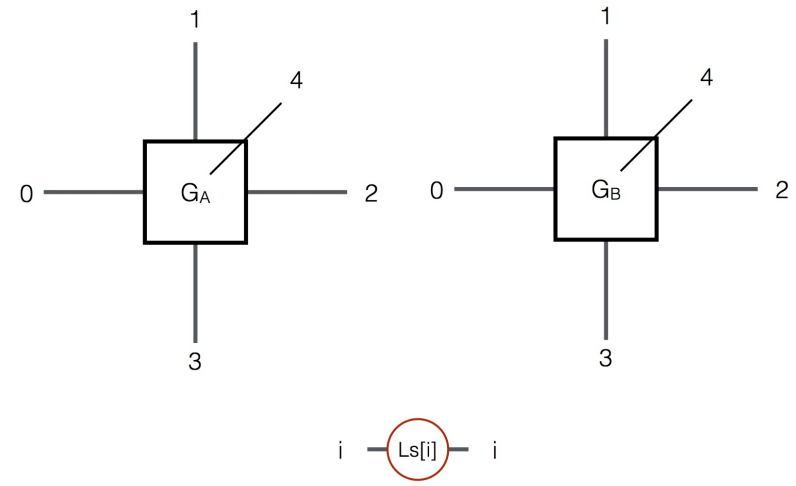
$$H = \sum_{\langle ij \rangle} \sigma_i^z \sigma_j^z + h \sum_i \sigma_i^x$$

We will extend the 1D iTEBD method to 2D. The sites are updated in pairs and the bonds are updated in the sequence of the four bonds shown in red in the following figure.

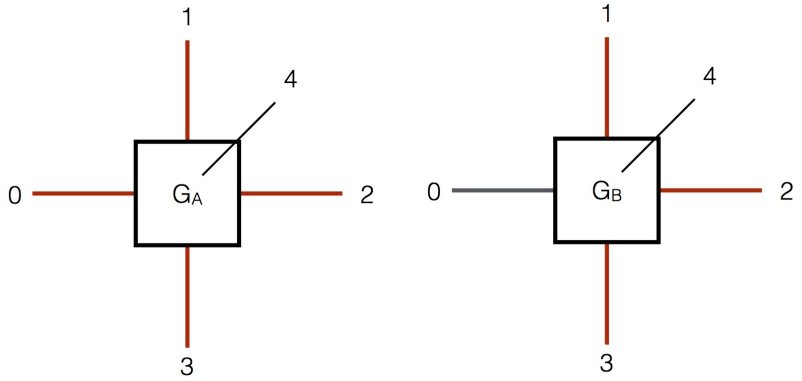


4.1.1 Example

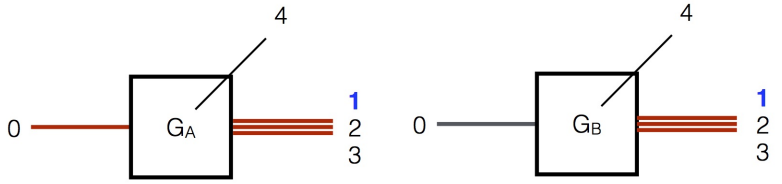
In the following, we give an example of the coordination number $z = 4$ for the virtual bonds of each tensor. We start with two tensors **GA** and **GB**, and diagonal matrices **Ls** on each bond:



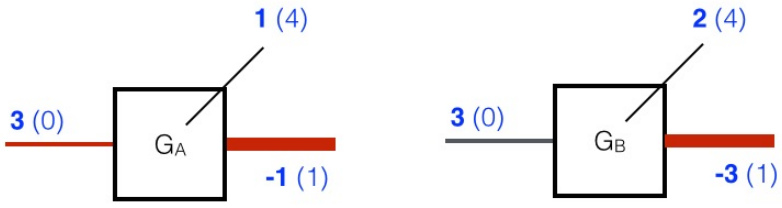
We want to update first bond 0 with **Ls[0]**. We first absorb **Ls** into the tensors (indicated as the red bonds.)



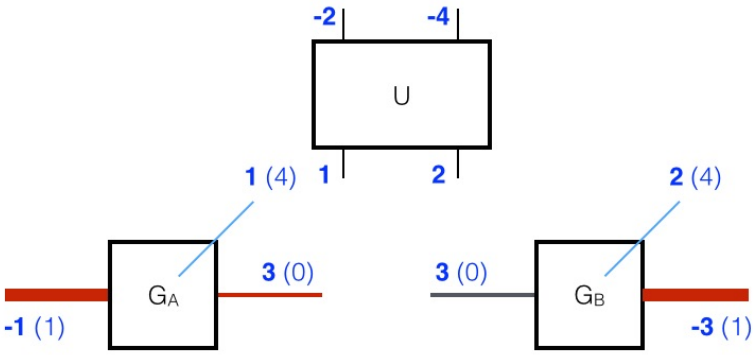
We merge the bonds not being updated to a huge bond with bond dimensions χ^3



Relabeling the tensors so that pyUni10 knows how to contract them:

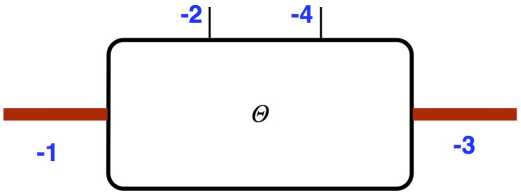


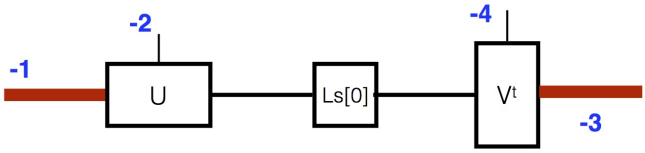
Finally, we obtain a Θ tensor of dimensions $d\chi^3 \times d\chi^3$ by contracting **GA**, **GB** and **U**



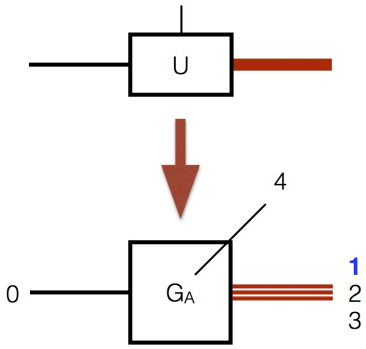
Performing SVD on Θ and truncate the bond, we have three new tensors

- **U** ($d\chi^3 \times \chi$)
- **Ls[0]** ($\chi \times \chi$), and
- **V** ^{t} ($\chi \times d\chi^3$)

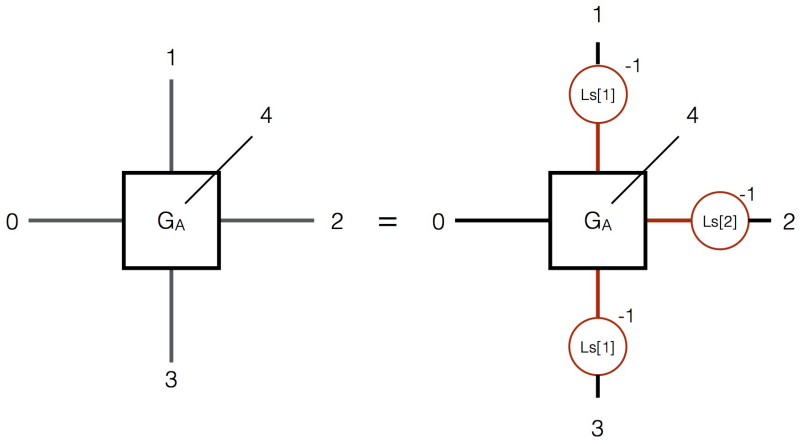




Thus we have new **GA** and **GB** merged with **Ls[1]**, **Ls[2]** and **Ls[3]**, and a new updated diagonal matrice **Ls[0]**.



To obtain **GA**, **GB**, we need to multiply the bonds with **Ls[i]** $^{-1}$



Repeat the same processes for the other bonds 1,2,and 3 to complete a single update.

You can download the ipython notebook `Tutorial3-1.ipynb`.

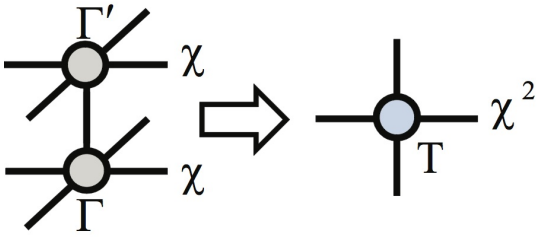
4.2 Tensor Renormalization Group

We will implement the tensor renormalization group (TRG) algorithm to compute the energy and magnetization in TFIM.

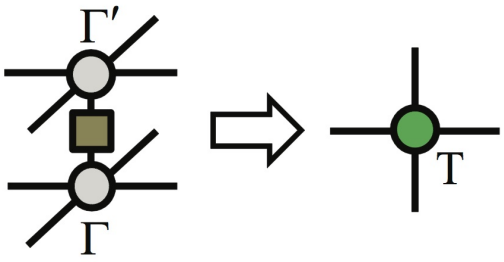
To compute the ground state expectation value for an operator $\hat{O}$

$$\langle \hat{O} \rangle = \frac{\langle \psi_0 | \hat{O} | \psi_0 \rangle}{\langle \psi_0 | \psi_0 \rangle}$$

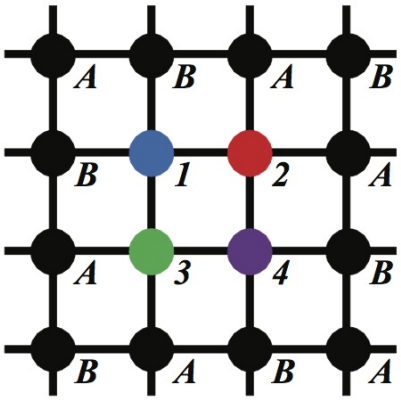
To calculate the norm $\langle \psi_0 | \psi_0 \rangle$ in the denominator, we contract first the physical indices of the bra and ket tensors to get a double tensor **T** :



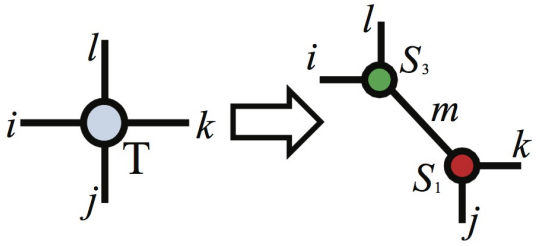
For the numerator, we construct an *impurity* tensor **T'** :

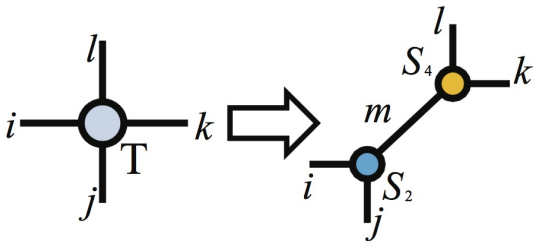


The numerator corresponds to a tensor network like this:

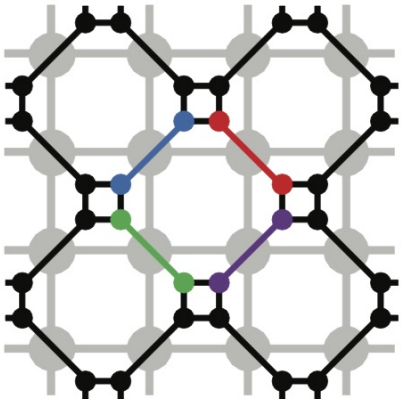


To contract the tensor network using TRG, first decompose the A-site and B-site tensors into two rank-3 tensors using SVD, and perform truncation:

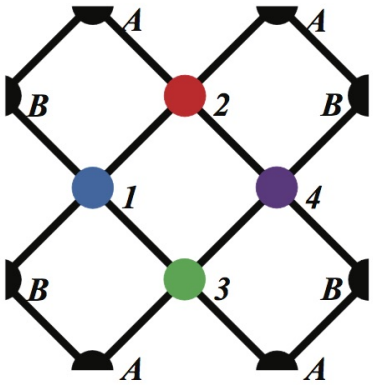




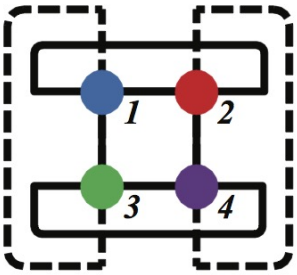
and we obtain,



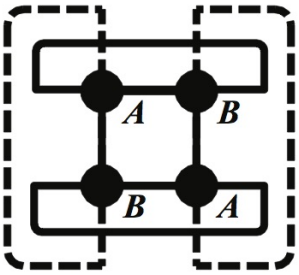
Contracting the tensors in the squares, we obtain a new tensor network with half of the size



Repeat the process until only four tensors remain, and contract them to obtain $\langle\psi_0|\hat{O}|\psi_0\rangle$



Similarly, for $\langle\psi_0|\psi_0\rangle$



You can download the ipython notebook `Tutorial3-2.ipynb`.

You also need these network files: `TRG.net` and `expectation.net`.

The labels for the tensors in the helper functions can be found `here`.

The full python code for the 2D transverse Ising model can be found `here`.

4.3 Refereces

1. J. Jordan, R. Orús, G. Vidal, F. Verstraete, and J. I. Cirac, Phys. Rev. Lett. 101, 250602 (2008), http://arxiv.org/abs/cond-mat/0703788
2. H. C. Jiang, Z. Y. Weng, and T. Xiang, Accurate Determination of Tensor Network State of Quantum Lattice Models in Two Dimensions, Phys. Rev. Lett. 101, 090603 (2008), http://arxiv.org/abs/0806.3719
3. H. H. Zhao, Z. Y. Xie, Q. N. Chen, Z. C. Wei, J. W. Cai, T. Xiang, Renormalization of Tensor-network States, Phys. Rev. B 81, 174411 (2010), http://arxiv.org/abs/1002.1405

CHAPTER 5

pyUni10

5.1 pyUni10 Constants

The constants defined in pyUni10 are:

5.1.1 pyUni10.BD_IN

pyUni10.BD_IN = 1
The value defines an **in-coming** bond

5.1.2 pyUni10.BD_OUT

pyUni10.BD_OUT = -1
The value defines an **out-going** bond

5.1.3 pyUni10.PRTF_EVEN

pyUni10.PRTF_EVEN = 0
Value defines particle parity **even** in a fermionic system

5.1.4 pyUni10.PRTF_ODD

pyUni10.PRTF_ODD = 1
Value defines particle parity **odd** in a fermionic system

5.1.5 pyUni10.PRT_EVEN

pyUni10.PRT_EVEN = 0
Value defines particle parity **even** in a bosonic system

5.1.6 pyUni10.PRT_ODD

pyUni10.PRT_ODD = 1
Value defines particle parity **odd** in a bosnic system

5.1.7 pyUni10.CTYPE

pyUni10.CTYPE = 2
The value defines a **Complex** datatype

5.1.8 pyUni10.RTYPE

pyUni10.RTYPE = 1
The value defines a **Real** datatype

5.2 Global Functions

5.2.1 pyUni10.combine

pyUni10.**combine**(*bdtype*, *bds*)
Combines bonds

Parameters

- **tp** (*BondType*) – BD_IN/BD_OUT
- **bds** (*array of bonds*) – list of bonds

Returns combined bond

Return type *Bond*

5.2.2 pyUni10.contract

pyUni10.**contract**(*Ta*, *Tb*, *fast=False*)

Contract out the bonds with common labels in *UniTensors* Ta and Tb. Ta and Tb are not copied and are permuted in-place, thus the process uses less memory than (Ta * Tb).

Parameters

- **Ta** (UniTensor) – a *UniTensor*
- **Tb** (UniTensor) – a *UniTensor*
- **fast** (*bool*) – if fast set to True, the tensors Ta and Tb will not be permuted back to the origin labels after contraction.

Returns a new *UniTensor* with remaining bonds

Return type *UniTensor*

5.2.3 pyUni10.otimes

pyUni10.**otimes**(*Ta*, *Tb*)
Tensor product of two matrices/tensors.

Parameters

- **Ta** (*Matrix / UniTensor*) – *Matrix* / *UniTensor*
- **Tb** (*Matrix / UniTensor*) – *Matrix* / *UniTensor*

Returns tensor product of Ta and Tb

Return type Matrix / UniTensor

5.2.4 pyUni10.takeExp

pyUni10.**takeExp**(*a*, *mat*)
Returns $\exp(aM)$

Parameters

- **a** (*float*) – mulitiplication constant
- **mat** (Matrix) – input *Matrix*

Returns a matrix of $\exp(aM)$

Return type *Matrix*

5.3 Classes

5.3.1 pyUni10.Qnum

class pyUni10.**Qnum**
Proxy of C++ uni10::Qnum class

Class for quntum numbers

pyUni10.**Qnum**([*U1=0*, *prt=PRT_EVEN*])

pyUni10.**Qnum**(*q*)
Creates a Qnum object

Parameters

- **U1** (*int*) – U1 quantum number
- **prt** (*parityType*) – parity quantum number
- **q** (Qnum) – Another Qnum object

Returns a Qnum** object

Return type Qnum**

pyUni10.**QnumF**(*prtF*[, *U1=0*, *prt=PRT_EVEN*])
Construct a Qnum object with fermionic parity

Parameters

- **prtF** (*parityFType*) – fermionic parity quantum number
- **U1** (*int*) – U1 quantum number
- **prt** (*parityType*) – parity quantum number

Returns Qnum object

Return type *Qnum*

Methods

Qnum.**U1**()
Returns the U1 quantum number

Returns U1 quantum number

Return type int

Qnum.**assign**([*U1=0*, *prt=PRT_EVEN*])
Assign a quantum number to a Qnum object

Parameters

- **U1** (*int*) – U1 quantum number
- **prt** (*parityType*) – parity quantum number

Returns a Qnum object

Return type *Qnum*

Qnum.**assignF**(*prtF*[, *U1=0*, *prt=PRT_EVEN*])
Assign a fermionic quantum number to a Qnum object

Parameters

- **prtF** (*parityFType*) – fermionic parity quantum number
- **U1** (*int*) – U1 quantum number
- **prt** (*parityType*) – parity quantum number

Returns a Qnum object

Return type *Qnum*

Qnum.**prt**()
Returns the parity quantum number

Returns PRT_EVEN or PRT_ODD

Return type parityType

Qnum.**prtF**()
Returns the fermionic parity quantun number

Returns PRTF_EVEN or PRTF_ODD

Return type parityFType

static Qnum.**isFermionic**()
Tests whether fermionic parity PRTF_ODD is ever defined.

Returns True or False

Return type bool

Attributes

Qnum.U1_LOB = -1000
Minimum allowed U1 quantum number

Qnum.U1_UPB = 1000
Maximum allowed U1 quantum number

Related data types

pyUni10.**parityType**
Type of parity quantum number

{PRT_EVEN, PRT_ODD}

pyUni10.**parityFType**
Type of fermionic parity quantum number

{PRTF_EVEN, PRTF_ODD}

5.3.2 pyUni10.Bond

class pyUni10.**Bond**
Proxy of C++ uni10::Bond class

Class for bonds

pyUni10.**Bond**(*bdtype*, *dim*)

pyUni10.**Bond**(*bdtype*, *qnums*)

pyUni10.**Bond**(*bd*)
Creates a Bond object

Parameters

- **bdtype** (*BondType*) – BD_IN / BD_OUT
- **dim** (*int*) – dimension of the bond (no symmetry)
- **qnums** (*array of Qnums*) – list of quantum numbers
- **bd** (Bond) – a Bond object

Returns a Bond object

Return type *Bond*

Methods

Bond.**Qlist**()
Returns a tuple of quantum numbers associate with *Bond*

Returns a tuple of quantum numbers

Return type tuple of Qnum

Bond.**assign**(*bdtype*, *dim*)

Bond.**assign**(*bdtype*, *qnums*)
Set the dimensions / quantum numbers of a bond.

Parameters

- **bdtype** (*BondType*) – BD_IN / BD_OUT
- **dim** (*int*) – dimension of a bond (no symmetry)
- **qnum** (*arrays of Qnum*) – arrays of quantum numbers

Bond.**change**(*bdtype*)
Changes the type of *Bond*, and changes the Qnums of *Bond* . If the bond is changed from incoming(BD_IN) to out-going(BD_OUT) type or vice versa, the resulting Qnums is -Qnums.

Parameters **bdtype** (*BondType*) – BD_IN / BD_OUT

Returns _Bond_ with type bdtype

Return type *Bond*

Bond.**combine**(*bd*)
Merge bd into *Bond*.

Parameters **bd** (Bond) – bond to be merged

Returns merged bond

Return type *Bond*

Bond.**degeneracy**()
Returns a dictionary of quantum numbers as the key and the correspoding number of degeneracy as the value ({Qnum: int}).

Returns dictionary of quantum number:degeracy pairs

Return type dict ({Qnum: int})

Bond.**dim**()
Returns the total dimension of *Bond*

Returns dimension of *Bond*

Return type int

Bond.**type**()
Returns the type of *Bond*

Returns BD_IN / BD_OUT

Return type BondType

5.3.3 pyUni10.Matrix

class pyUni10.**Matrix**
Proxy of C++ uni10::Matrix class

Class for matrices

pyUni10.**Matrix**(*Rnum*, *Cnum*[, *elem*], *diag=False*)

pyUni10.**Matrix**(*Ma*)
Creates a Real Matrix object

Parameters

- **Rnum** (*int*) – number of Rows
- **Cnum** (*int*) – number of Columns
- **diag** (*bool*) – Set True for a diagonal matrix
- **elem** (*iterable*) – Matrix elements
- **Ma** (Matrix) – another Matrix object

Returns a Matrix object

Return type *Matrix*

pyUni10.**CMatrix**(*Rnum*, *Cnum*[, *elem*], *diag=False*)
Creates a Complex Matrix object

Parameters

- **Rnum** (*int*) – number of Rows
- **Cnum** (*int*) – number of Columns
- **diag** (*bool*) – Set True for a diagonal matrix
- **elem** (*iterable*) – Matrix elements
- **Ma** (Matrix) – another Matrix object

Returns a Complex Matrix object

Return type *Matrix*

Methods

If no datatype flag is specified, the member functions default to *Real*. To explicitly declare the data type, the class provides constructors and member functions with the following syntax:

`func(RTYPE, ...)` for *Real* datatype and `func(CTYPE), ...)` for *Complex* datatype.

Matrix.**col**()
Returns the number of columns in *Matrix*

Returns number of columns in *Matrix*

Return type int

Matrix.**eigh**()

Diagonalizes a symmetric/hermitian *Matrix* and returns a tuple of matrices (D, U)**,** where D is a diagonal matrix of eigenvalues and U is a matrix of row-vectors of eigenvectors.

return

- D : diagonal matrix of eigenvalues
- U : matrix of eigenvectors

rtype tuple of Matrix

Note: This function will not check wether the matrix is symmetric/hermitian.

The operation is a wrapper of Lapack function `dsyev()` for Real matrix `zheev()` for Complex matrix.

`Matrix.`**`eig`**`()`

Diagonalizes a general *Matrix* and returns a tuple of matrices (D, U), where D is a diagonal matrix of eigenvalues and U is a matrix of row-vectors of right eigenvectors.

Returns

- D : diagonal matrix of eigenvalues
- U : matrix of right eigenvectors

Return type tuple of Matrix

Note: Only the right eigenvectors will be given. The operation is a wrapper of Lapack function `Xsyev()`.

`Matrix.`**`elemNum`**`()`

Returns the number of elements in *Matrix*

Returns number of elements in *Matrix*

Return type int

`Matrix.`**`getElem`**`()`

Returns the reference to the elements of *Matrix*

Returns reference to the element of *Matrix*

Return type float *

`Matrix.`**`identity`**`()`

Returns an identity matrix

Returns an identity matrix

Return type *Matrix*

`Matrix.`**`isDiag`**`()`

Checks whether *Matrix* is diagonal

Returns True or False

Return type bool

`Matrix.`**`inverse`**`()`

Returns inverse matrix of *Matrix*

Returns inverse matrix of *Matrix*

Rtpye Matrix

`Matrix.`**`load`**`(`*filename*`)`

Loads *Matrix* from a binary file named *filename*

Parameters **`filename`** (*`str`*) – input filename

Matrix.**norm**()
Returns L^2 norm of *Matrix*

Returns L^2 norm of *Matrix*

Return type float

Matrix.**orthoRand**()
Generates an orthogonal basis with random elements and assigns it to the elements of *Matrix*

Returns Matrix of orthogonal basis

Return type *Matrix*

Matrix.**randomize**()
Assigns random elements to *Matrix*

Returns Matrix of random elements

Return type *Matrix*

Matrix.**resize**(*Rnum*, *Cnum*)
Set the dimensions of *Matrix* to (Rnum, Cnum)

Returns Matrix of size (Rnum, Cnum)

Return type *Matrix*

Matrix.**row**()
Returns the number of rows in *Matrix*

Returns number of rows in *Matrix*

Return type int

Matrix.**save**(*filename*)
Saves *Matrix* to a binary file named *filename*

Parameters **filename** (*str*) – output filename

Matrix.**setElem**(*elem*)
Set elements of *Matrix* to elem

Parameters **elem** (*array of floats*) – data

Matrix.**set_zero**()
Set elements of *Matrix* to zero

Parameters **elem** (*array of floats*) – data

Matrix.**sum**()
Performs the summation of all elements in *Matrix*

Returns sum of all elements in *Matrix*

Return type float

Matrix.**svd**()
Performs SVD of *Matrix*

Factorizes the $m \times n$ matrix A into two unitary matrices U and $V^\dagger$, and a diagonal matrix Σ of singular values (real, non-negative) such that

$$A = U\Sigma V^\dagger$$

Returns

- U : a $m \times n$ row-major matrix
- Σ : a $n \times n$ diagonal matrix
- $V^\dagger$: a $n \times m$ row-major matrix

Return type tuple of Matix

Note: This is a wrapper of the Lapack function `Xgesvd()`

`Matrix.`**`trace`**`()`
Takes the trace of a square matrix *Matrix*

Returns trace of a square matrix *Matrix*

Return type float

Raise RunTimeError if *Matrix* is not a square matrix

`Matrix.`**`transpose`**`()`
Performs in-place transpose of a Real *Matrix*. The number of rows and the number of columns are exchanged.

Returns *Matrix*

Return type *Matrix*

`Matrix.`**`conj`**`()`
Performs in-place complex conjugation of elements in *Matrix*.

Returns complex conjugate of *Matrix*

Return type *Matrix*

`Matrix.`**`cTranspose`**`()`
Performs in-place Hermitian conjugate of a Complex *Matrix*. The number of rows and the number of columns are exchanged.

Returns *Matrix*

Return type *Matrix*

`Matrix.`**`max`**`()`
Returns the maximum matrix element

Returns maximum matrix elements

Return type float

`Matrix.`**`absMax`**`()`
Returns the matrix element with the maximum absolute value

Returns the matrix element with the maximum absolute value

Return type float

Note: This method only works for Real matrix

`Matrix.`**`absMaxNorm`**`()`
Normalizes the Matrix such that the maximum matrix element has absolute value 1

Returns normalized Matrix

Return type *Matrix*

Note: This method only works for Real matrix

Matrix.**qr**()

Matrix.**rq**()

Matrix.**ql**()

Matrix.**lq**()
Performs QR, RQ, QL, LQ decompositions of Matrix.

Returns A tuple of Matrix of upper(R)/lower(L) triangular matrix and a unitary matrix (Q)

Return type tuple of Matrix

Matrix.**typeID**()
Returns datatype of Matrix

Returns CTYPE or RTYPE

Return type int

5.3.4 pyUni10.Network

class pyUni10.**Network**

pyUni10.**Network**(*filename*[, *tensors*])
Loads the network structure from file.

Parameters

- **filename** (*str*) – file where the network structure is stored
- **tensors** (*array of UniTensors*) – list of tensors to be put into *Network*

Returns a Network object

Return type *Network*

Methods

Network.**launch**([*name*])
Performs contraction of the tensors in the network, and returns a UniTensor named name (optional).

Parameters **name** (*str*) – name of the output UniTensor

Returns contracted tensor

Return type *UniTensor*

Network.**profile**()
Prints the memory usage of *Network*

Returns current memory usage of *Network*

Return type str

Network.**putTensor**(*idx*, *T*[, *force=True*])

Network.**putTensor**(*tname*, *T*[, *force=True*])
Replaces the tensor of index idx / name tname in the network file with T. If the force flag is set, the tensor will be put in the network without reconstructing the pair-wise contraction sequence.

Parameters

- **idx** (*int*) – sequential index of a tensor in *Network*
- **name** (*str*) – name of a tensor in *Network*
- **T** (UniTensor) – tensor to be put into *Network*
- **force** (*bool*) – if set True, the contraction sequence is not reconstructed

Network.**putTensorT**(*idx*, *T*[, *force=True*])

Network.**putTensorT**(*tname*, *T*[, *force=True*])
Replaces the tensor of index idx / name tname in the network file with the *transpose* of T. If the force flag is set, the tensor will be put in the network without reconstructing the pair-wise contraction sequence.

Parameters

- **idx** (*int*) – sequential index of a tensor in *Network*
- **name** (*str*) – name of a tensor in *Network*
- **T** (UniTensor) – tensor to be put into *Network*
- **force** (*bool*) – if set True, the contraction sequence is not reconstructed

5.3.5 pyUni10.UniTensor

class pyUni10.**UniTensor**
Proxy of C++ uni10::UniTensor class.

Class for symmetric tensors.

pyUni10.**UniTensor**([*val*])
Construct a rank-0 tensor (scalar)

Parameters **var** (*float*) – initial value

Returns a rank-0 tensor (scalar)

Return type *UniTensor*

pyUni10.**UniTensor**(*bds*, *labels*, *name=""*)
Construct a UniTensor

Parameters

- **bds** (*array of Bond*) – list of bonds
- **labels** (*array of int*) – list of labels
- **name** (*str*) – name of the tensor

Returns a UniTensor object

Return type *UniTensor*

Methods

UniTensor.**assign**(*bds*)
Restructures the *UniTensor* with bonds bds, and clear all the content.

Parameters **bds** (*array of Bonds*) – list of bonds

Returns a UniTensor with bonds bds

Return type *UniTensor*

UniTensor.**blockNum**()
Returns the number of blocks in *UniTensor*

Returns total number of blocks

Return type int

UniTensor.**blockQnum**([*idx*])
Returns the quantum number of the idx-th block.

If no input is given, returns full list of quantum numbers associated with blocks in *UniTensor*.

Parameters **idx** (*int*) – block index

Returns quantum number(s)

Return type (array of) *Qnum*

UniTensor.**bond**([*idx*])
Returns the idx-th bond in *UniTensor*.

If no input is given, returns an array of bonds associated with *UniTensor*.

Returns bond(s)

Return type (array of) *Bond*

UniTensor.**bondNum**()
Returns the number of bonds in *UniTensor*.

Returns number of bonds

Return type int

UniTensor.**combineBond**(*labels*)
Combines bonds with *labels*. The resulting bond has the same label and bondType as the bond with the first label in *labels*.

Parameters **labels** (*list*) – array of labels

Returns combined bond with the same label and bondType as first bond in *labels*

Return type *Bond*

UniTensor.**elemCmp**(*Tb*)
Tests whether the elements of the UniTensor are the same as in Tb.

Parameters **Tb** (UniTensor) –

Returns True if the elements of *UniTensor* is the same as in Tb, False otherwise.

Rtyoe bool

UniTensor.**elemNum**()
Returns the number of elements in *UniTensor*.

Returns number of elements

Return type int

UniTensor.**getBlock**([*qnum*], *diag=false*)

Returns the block elements of quantum number qnum as a Matrix. If the diag flag is set, only the diagonal elements of the block will be picked out to a diagonal Matrix. If qnum is not given, returns the Qnum(0) block.

Parameters

- **qnum** (Qnum) – blcok quantum number
- **diag** (*bool*) – If True, output a diagnoal part only

Returns a Matrix of block of qnum

Return type *Matrix*

UniTensor.**getBlocks**()

Returns the a dictionary {qnum:block} of the mapping from Qnum to Matrix .

Returns mapping from Qnum to Matrix

Return type dict

UniTensor.**getElem**()

Returns the reference to th elements of *UniTensor*

Returns Reference to the elements

Return type float *

UniTensor.**getName**()

Returns the name of *UniTensor*

Returns Name of *UniTensor*

Return type str

UniTensor.**getRawElem**()

Returns the raw elements of *UniTensor* with row(column) basis defined by the incoming (outgoing) bonds.

Returns raw elements of *UniTensor*

Return type *Matrix*

UniTensor.**identity**([*qnum*])

Set the diagonal elements to 1 and the off-diagonal elements to 0 in all blocks. If qnum is given, only set the elements in the block with quantum number equal to qnum.

param Qnum qnum quantum number

UniTensor.**inBondNum**()

Returns the number of incoming bonds in *UniTensor*

Returns number of incoming bonds

Return type int

UniTensor.**label**([*idx*])

Returns the label of the idx-th bond. If no input is given, returns an array of labels.

Parameters **idx** (*int*) – bond index

Returns (array of) label(s)

Return type int

UniTensor.**orthoRand**()
Randomly generates orthogonal bases and assigns to blocks.

UniTensor.**partialTrace**(*la*, *lb*)
Traces out bonds of label la and lb, and returns a reference to resulting tensor.

Returns reference to the partial trace of *UniTensor*

Return type float *

UniTensor.**permute**(*[new_label]*, *inBondNum*)
Permutes the order the bonds according to new_label, and changes the number of incoming bonds to inBondNum.

Returns reference to the permuted *UniTensor*

Return type float *

UniTensor.**printRawElem**()
Prints the raw elements of *UniTensor*

static UniTensor.**profile**()
Prints the memory usage of all the existing *UniTensors* .

UniTensor.**putBlock**(*[qnum]*, *mat*)
Assigns the elements of Matrix mat into the block with Qnum qnum of *UniTensor*. If qnum is not give, assigns to Qnum(0) block.

Parameters

- **qnum** (Qnum) – quantum number of the block being assigned to
- **mat** (Matrix) – matrix to be assigned.

UniTensor.**randomize**()
Assigns random numbers between 0 and 1 to the elements of *UniTensor*.

UniTensor.**save**(*filename*)
Saves the content of *UniTensor* to the binary file *filename*.

UniTensor.**setElem**(*elem*)
Assigns the elements to *UniTensor*, replacing the originals.

Parameters **elem** (*array of float*) – elements

UniTensor.**setLabel**(*new_labels*)
Assigns *new_labels* to the bonds of *UniTensor*, replacing the orinals.

Parameters **new_labels** (*array of int*) – new labels

UniTensor.**setName**(*name*)
Assigns *name* to *UniTensor*

Parameters **name** (*str*) – name to be assigned

UniTensor.**setRawElem**(*rawElem*)
Assigns raw elements (non-block-diagonal) to *UniTensor*.

Parameters **rawElem** (*array of float*) – input elements

UniTensor.**set_zero**()
Sets the elements of *UniTensor* to zero.

UniTensor.**similar**(*Tb*)
Tests whether the *UniTensor* is similar to input tensor Tb. Two tensors are said to be similar if the bonds of the tensors are exactly the same.

Parameters **Tb** (UniTensor) – tensor to be compared to.

Returns True if *UniTensor* and Tb are similar.

Return type bool

UniTensor.**trace**()
Traces out incoming and outgoing bonds, and returns the trace value.

Returns trace of *UniTensor*.

Return type float

UniTensor.**transpose**()
Transposes all blocks associated with quantum numbers. The bonds are changed from incoming to outcoming or vice versa while the quantum numbers remain the same on the bonds.

Returns reference to the transposed tensor.

Return type UniTensor &

UniTensor.**hosvd**(*group_labels*, *groups*, *groupsSize*, *Ls*)
Performs High-order SVD of UniTensor.

Parameters

- **group_labels** (*array of int*) – Ordered labels of the bonds
- **list** (*groups*) – Number of external bonds in each mode
- **groupSize** (*int*) – Number of modes
- **Ls** (*array of Matrix*) – Singular values in each direction

Returns array of unitaries, and the core tensor

UniTensor.**hosvd**(*modeNum*, *fixedNum*, *Ls*)
Performs High-order SVD of UniTensor

Parameters

- **modeNum** (*int*) – Number of output modes
- **fixedNum** (*int*) – Number of bonds to remain unchanged

Returns array of unitaries, and the core tensor

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

P

Q

R

S

T

U