Tensor networks in a nutshell

Raghav G. Jha[a]

Thomas Jefferson National Accelerator Facility, Newport News, VA 23606, USA

E-mail: raghav.govind.jha@gmail.com

Abstract: In these lecture notes, we discuss the technique of studying classical and quantum statistical systems and field theories using tensor networks. In particular, we focus on real-space tensor renormalization group methods based on coarse-graining networks to understand the behaviour of these systems. We demonstrate the algorithm for two-dimensional Ising model which has exact solution. We then study $O(2)$ model and its deformation and locate the phase transitions in these models. We then move to tensor formulation of $SU(2)$ gauge-Higgs model with Wilson action plus a matter term in the unitary gauge. We consider the extension of these algorithms to three-dimensional Euclidean models such as q-state Potts model and $O(2)$ model. The `Python` codes are provided to allow for hands-on experience. We will also use QUTIP for some problems.

Last edited: 2024-08-04 at 17:22:06

Contents

Contents

1 Introduction

The main problem of dealing with quantum many-body systems is the size of the Hilbert space. There is no hope to completely address the entire space even if we enter the NISQ era or even beyond it. Therefore, an understanding of the 'states that matters' becomes very important. In the last decade or so, it has been found that the ground states of local gapped Hamiltonian with short-range interactions are astonisghgly concentrated in a very tiny region of the Hilbert space which we hereafter call 'area-law' states. If one focusses just on this region and forget about the complete Hilbert space, one can still uncover the ground-state properties to reasonable accuracy. One of the guiding lights to identify and isolate 'area-law' states has been the notion of entanglement entropy. It has been shown that these special states follow a different scaling of the entropy than naively expected (volume-scaling) from a random state in the Hilbert space.

This problem is not so easy to deal for critical systems or systems where you have long-range interactions (say the Hamiltonian is next-to-next neighbour and even more non-local) but it seems that the fruit does not fall far from 'area-law' states even in those cases. So, one still has the advantage of not dealing with exponentially large Hilbert space $\mathcal{H}$.

If we pick a random state of the Hilbert space, we get a highly entangled state with a volume law, i.e., $S \propto L^d$ where d is the number of spatial dimensions. The ground states of gapped systems satisfy an area law i.e., $S \propto L^{d-1}$ i.e., S is constant for one-dimensional quantum systems. Hence, we can represent ground states efficiently with MPS of small bond dimension. However, for critical i.e., gapless) systems, the area law is not strictly true, but there are logarithmic corrections. Even then, the entanglement is small compared to random states.

Let us consider the Lagrangian $\mathcal{L} \sim \lambda \partial^{n_1} \phi^{n_2}$, then we have the mass dimension of $[\lambda] = D + \left(1 - \frac{D}{2}\right) n_2 + n_1$ because $S \sim \int d^D x \mathcal{L}$ is dimensionless and $[\phi] = [E]^{\frac{D-2}{2}}$ and $[d^D x] = [E]^{-D}$.

Consider a theory with fixed point $S_\star$ with one relevant perturbation i.e.,

$$S = S_\star + h \int d^D x O[\sigma(x)], \tag{1.1}$$

under the change of scale (RG) $b = b'/a$

$$S' = S_\star + b^y h \int d^D x O[\mu(x)], \tag{1.2}$$

with $y > 0$ for a relevant operator. The fixed point at $h = 0$ is destabilized by relevant operator $O[\sigma(x)]$. This means that the $h = 0$ FP is actually a IR unstable fixed point (UV stable fixed point).

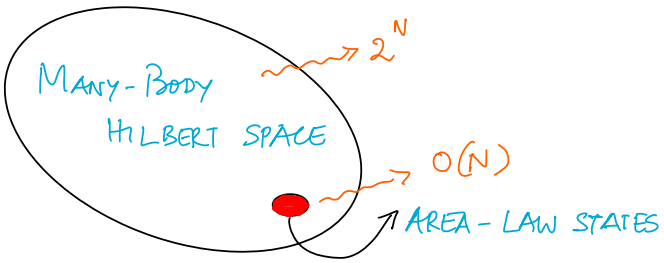

Figure 1. XX

```
from ncon import ncon
import numpy as np
```

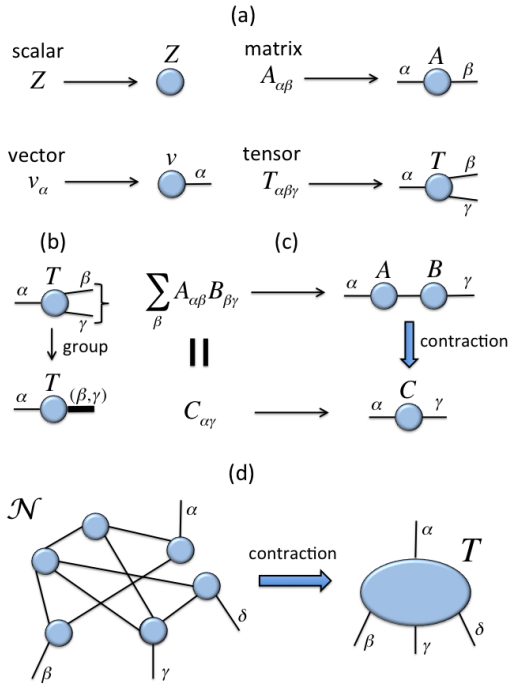

Figure 2. XX

"einsum"(Einstein's summation convention) is a very useful NumPy tool for contracting tensors. For example, matrix multiplication using this is - np.einsum('ij,jk->ik', A, B) which is just $C_{ik} = A_{ij}B_{jk}$. Dot product is - np.einsum('i,i->', A, B). See this.

But "einsum" is slow. In these lecture notes, we will use NCON except Exercise 1 where we will use "einsum" once. Please download/copy NCON from here.

For example, $C_{ip} = A_{ijk}B_{pjk}$ can be implemented in Python using NCON as:
`C = ncon((A, B),([-1,2,1], [-2,2,1]))`
- Same positive integers are contracted indices while the order [-1, ···] stands for the ordering of the indices in the resulting tensor.

Exercise 1: Consider the Hamiltonian of three spins ($N = 3$) quantum Ising model given by:

$$H = \sigma_1^x \otimes \sigma_2^x \otimes \mathbb{I}_2 + \mathbb{I}_2 \otimes \sigma_2^x \otimes \sigma_3^x + \sigma_3^x \otimes \sigma_1^x \otimes \mathbb{I}_2 + h\Big(\sigma_1^z \otimes \mathbb{I}_4 + \mathbb{I}_2 \otimes \sigma_2^z \otimes \mathbb{I}_2 + \mathbb{I}_4 \otimes \sigma_3^z\Big)$$

Since $\dim(\mathcal{H}) = 8$, use exact diagonalization and compute ground state energy for various h.

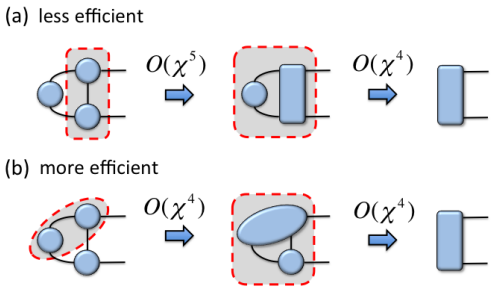

Figure 3. The figures are taken from [1]

Exercise 2: Construct the quantum Ising Hamiltonian for N spins and check that it reproduces the result from previous exercise when $N = 3$. Now use exact diagonalization and compute ground state energy for same values of h with $N = 7$ or $N = 8$. Please do not try to run with $N > 10$ since it might long time.

Exercise 3: Calculate the trace of product of four random 40×40 matrices using einsum and check that result the result agrees with that obtained from np.trace and np.dot. You can construct random matrices using: `A = np.random.rand(3,3)`. Now try to set the flag `optimize=True` in einsum and notice difference in runtime.

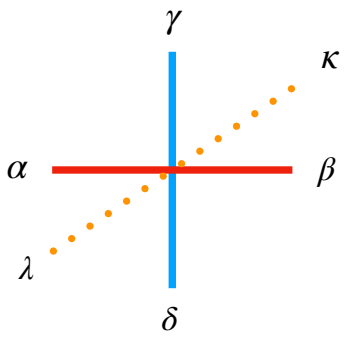

Figure 4. Diagrammatic representation of a rank-six tensor, $T_{\alpha\beta\gamma\delta\kappa\lambda}$ which can serve as a fundamental tensor of some 3d classical statistical system.

Exercise 3: Compute the rank-four tensor A_{rqba} which is equal to $B_{ijkl}C_{jiqr}D_{lkab}$ using NCON where all indices run from $1 \cdots 3$. Draw a tensor diagram of this contraction. You can choose the tensors to be random like before.

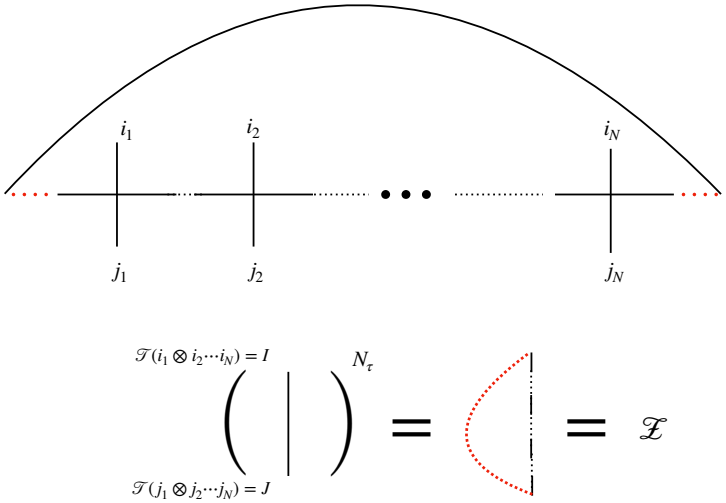

Figure 5. The schematic diagram which shows how we can coarse-grain the tensor along one direction and construct the transfer matrix and partition function.

SCHMIDT DECOMPOSITION THEOREM: Any given state vector $|\psi\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$ can be written as,

$$|\psi\rangle = \sum_{i=1}^{N} \lambda_i |\alpha_i\rangle |\beta_j\rangle \tag{1.3}$$

for positive, real λ_i and orthonormal sets $\alpha_i \in \mathcal{H}_A$ (with dimension N_A) and $\beta_i \in \mathcal{H}_B$ (with dimension N_B). Note that $N \leq \min.(N_A, N_B)$. For a product state we have $N = 1$ and $N > 1$ refers to entangled state. Small N are often referred to as 'slightly entangled'.

1.1 Singular Value Decomposition (SVD) and QR decomposition

The SVD of an $m \times n$ real or complex matrix $\mathbf{M}$ is a factorization of the form $U\Sigma V^\dagger$, where U is $m \times m$ real or complex unitary matrix, Σ is $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V is an $n \times n$ real or complex unitary matrix. The diagonal entries Σ_{ii} of Σ are known as the singular values of $\mathbf{M}$. The number of non-zero singular values is equal to the rank of $\mathbf{M}$.

Mathematical applications of the SVD include computing the pseudoinverse, matrix approximations, and determining the rank, range, and null space of a matrix. The SVD is also extremely useful in all areas of science, engineering, and statistics, such as signal processing, least squares fitting of data, and process control.

SVD is an integral step in all tensor network coarse-graining algorithm. It helps in reducing the ever-growing size of the fundamental tensor such that meaningful computations can be carried on classical computers. There are other alternatives which have been explored - such as randomized SVD. We will only use SVD in these lectures. Howver, there is a cheaper option when we do not explicitly want the singular matrix s. It is called QR decomposition.

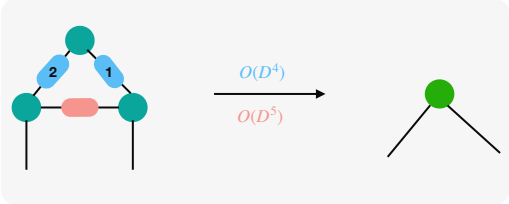

Figure 6. .....

2 Exact diagonalization - an example code

Here we will discuss the method of exactly diagonalizing the quantum hamiltonian for 3 spins and plot the ground state energy as a function of magnetic field. here

```
# Exact Diagonalization!
import numpy as np
from matplotlib import pyplot as plt
from numpy import linalg as LA
import sys

E = np.eye(2)
EE = np.eye(4)
X = [[0.0, 1.0],[1.0, 0]] # \sigma_x
Z = [[1.0, 0.0],[0.0, -1.0]] # \sigma_z
XX = np.kron(X,X)
HXX = np.kron(XX,E) + np.kron(E, XX) + np.kron(X,np.kron(E,X))
# (XX,E) means 1 & 2 are X and third is identity.
HZ = np.kron(Z,EE) + np.kron(E,np.kron(Z,E)) + np.kron(EE,Z)
h = np.arange(0.0, 2.0, 0.2).tolist()
Nsteps = int(np.shape(h)[0])
E0 = np.zeros(Nsteps)

# Now generalize ...
def buildH(N, h):

    if N > 12:
      print ("Lower N to finish quickly")
      sys.exit(1)

    HXX = XX
    HZ = np.kron(Z,E) + np.kron(E,Z)
    for n in range (3, N+1):
        HXX = np.kron(HXX,E)+ np.kron(np.eye(2**(n-2)), XX)
        HZ =  np.kron(HZ,E) + np.kron(np.eye(2**(n-1)), Z)
```

```
    HXX = HXX + np.kron(X, np.kron(np.eye(2**(N-2)), X))
    H = HXX + (h*HZ)
    return H

for i in range (0, Nsteps):
    #H = HXX + (h[i] * HZ)
    H = buildH(10, h[i])
    D, U = LA.eigh(H)
    E0[i] = min(D)

plt.plot(h, E0, marker="*", color = "r")
plt.title('Ground state energy: Quantum Ising model for "N" spins',
    fontsize=20)
plt.xlabel('Magnetic Field, h', fontsize=16)
plt.ylabel('Energy, E', fontsize=16)
plt.show()
```

Before moving to the tensor network computations, we do a quick recap of contracting indices in Python.

```
import numpy as np

# Multiply matrices
A = np.random.rand(10,10)
B = np.random.rand(10,10)
out1 = A@B
out2 = np.einsum('ij, jk',A,B)
print(np.allclose(out1, out2))

# Contract tensors
B = np.random.rand(3,3,3,3)
C = np.random.rand(3,3,3,3)
D = np.random.rand(3,3,3,3)
out2 = np.einsum('ijkl,jiqr,lkab->rqba',B,C,D)
```

3 Quantum Ising Model

The Hamiltonian of the Ising model is given by:

$$\mathcal{H} = -J \sum_{\langle ij \rangle} Z_i Z_j, \tag{3.1}$$

and the Hamiltonian is invariant under flipping of all the spins i.e., $Z \to -Z$. It has two ground states where all spins are oriented in the same direction i.e., $|000\cdots0\rangle$ and

$|111\cdots1\rangle$. They both have energy $-J(N-1)$. Note that the ground state is not invariant under $Z^{\otimes N}$ i.e., $Z^{\otimes N}|0\rangle^{\otimes N} \neq |0\rangle^{\otimes N}$. Therefore, we say that ground state breaks the symmetry spontaneously. This is known as 'spontaneous symmetry breaking' (SSB). The magnetization which is the average spin per lattice site is either $\pm N$ corresponding to $|000\cdots0\rangle = |\uparrow\uparrow\uparrow\cdots\uparrow\rangle$ and $|111\cdots1\rangle = |\downarrow\downarrow\downarrow\cdots\downarrow\rangle$ respectively. To make things more interesting, we add another term to the $\mathcal{H}$ above. This is then known as transverse-field Ising model (TFIM).

$$\mathcal{H} = -J\sum_{\langle ij\rangle} Z_iZ_j - h\sum_i X_i. \tag{3.2}$$

When $h \gg J$, the ground state is an eigenstate of X i.e., $|\rightarrow\rightarrow\rightarrow\cdots\rightarrow\rangle = |+++\cdots+\rangle$ where $|+\rangle$ is the eigenstate of X obtained from $|0\rangle$ by performing $H|0\rangle$ where H is the Hadamard gate. When $J \gg h$, we can ignore this second term and return to two ground states as discussed before. Due to such striking change in the ground state structure, there must be be something interesting going on for intermediate values i.e, $h \sim J$. And indeed, it turns out that this model has a quantum phase transition for $h = 1$. The critical exponents are well-known and given by $\alpha = 0, \beta = 1/8, \gamma = 7/4, \delta = 15, \eta = 1/4, \nu = 1$. The ground state energy for general J and h was given in Ref. [2] and is:

$$\frac{E_0}{N} = -\frac{2}{\pi}(1+h)\ E\left(\frac{\pi}{2}, \sqrt{\frac{4h}{(1+h)^2}}\right) \tag{3.3}$$

We can also insert factors of J to be more precise as shown below in the Mathematica code:

```
E0[J_, h_] := -2 J/\[Pi] (1+h/J)EllipticE[\[Pi]/2, Sqrt[(4 h/J)/(1+h/J)^2]];
```

4 MPS methods for one-dimensional quantum systems

The ground state of a local-Hamiltonian of N spins which lives in 2^N dimensional Hilbert space is written as:

$$|\psi\rangle = \sum_{\sigma_1\cdots\sigma_L} c_{\sigma_1\cdots\sigma_L}|\sigma_1\cdots\sigma_L\rangle \tag{4.1}$$

$$|\psi\rangle = \sum_{\boldsymbol{\sigma}}\sum_{a_1,\cdots,a_L} A^{\sigma_1}_{1,a_1}A^{\sigma_2}_{a_1,a_2}\cdots A^{\sigma_L}_{a_{L-1},a_L}|\boldsymbol{\sigma}\rangle \tag{4.2}$$

The physical dimension (denoted by σ) is d while the a indices are truncated to some χ known as bond dimension. Therefore, each A is a rank-three tensor with $d\chi^2$ elements hence total of $d\chi^2L$ parameters. This is substantial improvement over d^L. With spin-1/2 systems or qubits, we have $d = 2$. For models like AKLT, we have $d = 3$.

4.1 AKLT ground state

One of the famous examples of description of ground state in terms of MPS is for AKLT (Affleck-Kennedy-Lieb-Tasaki) model. The Hamiltonian is given by:

$$\mathcal{H} = \sum_i \vec{S}_i \cdot \vec{S}_{i+1} + \frac{1}{3}(\vec{S}_i \cdot \vec{S}_{i+1})^2 \tag{4.3}$$

The ground state of this H can be defined by MPS with
Make sure it is correct using Mathematica:

```
CT = KroneckerProduct;
Ap = {{0, Sqrt[2/3]}, {0, 0}};
Ao = {{-1/Sqrt[3], 0}, {0, 1/Sqrt[3]}};
Am = {{0, 0}, {-Sqrt[2/3], 0}};
CT[Ap, Ap] + CT[Ao, Ao] + CT[Am, Am] // MatrixForm
```

5 TRG in 2+0-dimensions

The basic algorithm remains the same irrespective of the model. Different models have different initial tensors. Let us look at one step of coarse-graining using TRG based algorithms.

```
def coarse_graining(t):
    Tfour = contract('jabe,iecd,labf,kfcd->ijkl', t, t, t, t)
    U = SVD(Tfour,[0,1],[2,3],D_cut)
    Tx = contract('abi,bjdc,acel,edk->ijkl', U, t, t, U)
    Tfour = contract('aibc,bjde,akfc,flde->ijkl',Tx,Tx,Tx,Tx)
    U = SVD(Tfour,[0,1],[2,3],D_cut)
    Txy = contract('abj,iacd,cbke,del->ijkl', U, Tx, Tx, U)
    norm = np.max(Txy)
    Txy /= norm
    return Txy, norm
```

5.1 Ising Model

We can use QUTIP.

```
# To install run: 'pip install qutip'
from qutip import *
import numpy as np

rho = ket2dm(bell_state(state='00'))
rhosq = rho*rho
tracerhosq = np.trace(rhosq.full())
print("Tr(rho^2)=",tracerhosq)
rhoA=rho.ptrace([1])
print ("von Neumann entropy", entropy_vn(rhoA))

#Create a random density matrix
```

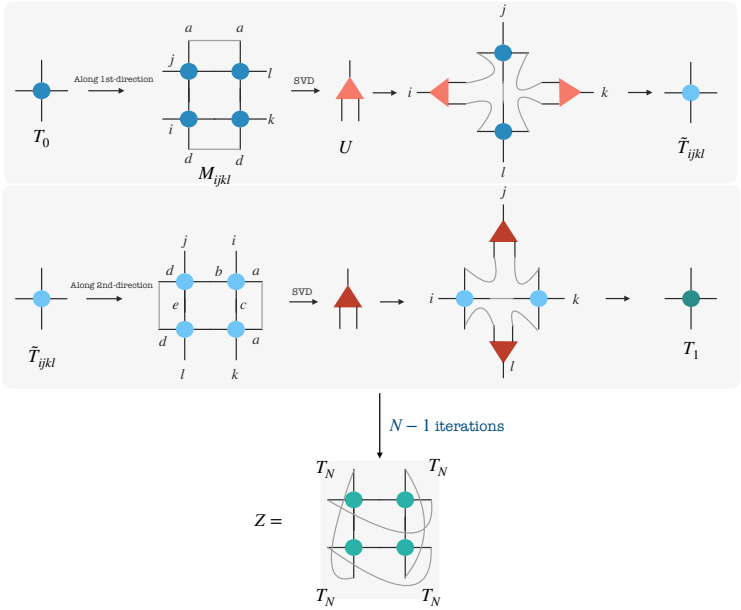

Figure 7.

```
rho_dm = rand_dm(4)

#Trace over one qubit of 3-qubit GHZ state

nqubits = 3
ghz1 = ghz_state(nqubits)
rho1 = ket2dm(ghz1)
rhoAB_GHZ=rho1.ptrace([1,2])
print ("von Neumann entropy", entropy_vn(rhoAB_GHZ))

# Create W-state and do the same
ws = w_state(3)
rho1 = ket2dm(ws)
rhoAB_W=rho1.ptrace([1,2])
print ("von Neumann entropy", entropy_vn(rhoAB_W))
```

The exact result obtained is [4]

Let us define $\kappa = \frac{\sinh(2\beta)}{2\cosh^2(2\beta)}$ and then the free energy density is given by: $f = -\frac{1}{\beta}\left(\log(2\cosh(2\beta)) - \kappa^2\, {}_4F_3\left(1,1,1.5,1.5;2,2,2;16\kappa^2\right)\right)$

The critical temperature is given by:

$T_c = \frac{2}{\log(1+\sqrt{2})} = 2.26918531421$ i.e. $\beta_c \approx 0.44069$

which is obtained by solving

$\sinh(2\beta_c)^2 = 1$.

In this example, we will reproduce Fig.(3) of arXiv version here[5].

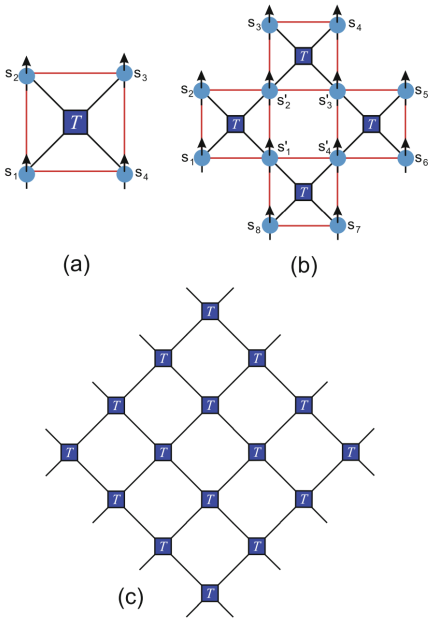

Figure 8. The figure is taken from Ref. [3]

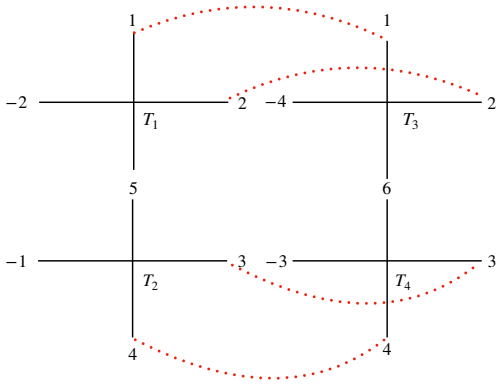

Figure 9. One step of coarse-graining along a specific direction. The `ncon` equivalent of this diagram is `ncon([T1,T2,T3,T4],[[-2,1,2,5],[-1,5,3,4],[-4,1,2,6],[-3,6,3,4]])`

The fundamental tensor T for the 2d classical Ising model can be written as:
$T_{ijkl} = W_{pi}W_{pj}W_{pk}W_{pl}$ `where` W `is given by:`
`W = np.array([[`$\sqrt{\cosh\beta}$`,`$\sqrt{\sinh\beta}$`],[`$\sqrt{\cosh\beta}$`,-`$\sqrt{\cosh\beta}$`]])`

The code is pasted below. You can refer to it if you are stuck. You will also need to use a simple numerical differentiation code to compute $\frac{-\partial \ln Z}{\partial \beta}$ from log of partition function. You can see it below:

```python
# CPU/CUDA version of tensor formulation of 2d classical models.
import numpy as np
import scipy as sp
import sys
import os
import time
from scipy.special import iv
from math import sqrt, log
from numpy import prod
from datetime import datetime
from math import sqrt, log, cos, cosh, sinh, tanh, pi
import scipy.integrate as integrate
from numpy import linalg as LA
from itertools import product

import torch
use_cuda = torch.cuda.is_available()

print("NumPy version", np.__version__)
print("SciPy version", sp.__version__)
print("Torch version", torch.__version__)


# Check if the number of input parameters is correct or not
if len(sys.argv) != 5:
  print("Usage:", str(sys.argv[0]), "<Temperature, Bond dimension, Niter,
      model>")
  sys.exit(1)

# Model Parameters
Temp = float(sys.argv[1])       # Temperature
D_cut = int(sys.argv[2])        # Bond Dimension
Niters = int(sys.argv[3])       # Number of iterations
model = str(sys.argv[4])        # Model to run

beta = float(1.0/Temp)          # Inverse temperature
Ns = int(2**((Niters)))         # Number of lattice sites in each dimension
vol = Ns**2                     # Lattice volume
Dn = int(D_cut/2.0)             # For initial tensor for XY/GXY

models_allowed = ['Ising', 'XY', 'GXY', 'Potts'] # The models that can be
    run using 2dTRG.py code

# Check if the model is an allowed model or not
if model not in models_allowed:
    print ("Model not supported. Exit")
```

```
    sys.exit(1)

# Check whether CUDA available or not and load corresponding library
use_cuda = torch.cuda.is_available()

# Start running the code
print ("STARTED", datetime.now())

# Some more model parameters
if model == 'GXY': delta, mcut = 0.8, 50  # For GXY Model
if model == 'XY': delta, mcut = 1.0, 0    # For XY Model
if model == 'Potts': qstate = 3           # For 3-state Potts model

# Print GPU-related information and load libraries
if use_cuda:
    print ('------------------------------------------------')
    print('__CUDNN VERSION:', torch.backends.cudnn.version())
    print('__Number CUDA Devices:', torch.cuda.device_count())
    print('__CUDA Device Name:',torch.cuda.get_device_name(0))
    print('__CUDA Device Total Memory
        [GB]:',torch.cuda.get_device_properties(0).total_memory/1e9)
    print ('------------------------------------------------')

    import opt_einsum_torch as ee
    #ee = EinsumPlanner(torch.device('cuda:0'), cuda_mem_limit = 0.7)

# Import CPU-based python libraries if CUDA not available
else:
    import psutil
    import platform
    import multiprocessing
    from opt_einsum import contract


def exact_free_energy_Ising(temp):
    beta = 1.0 / temp
    cc, ss = cosh(2.0 * beta), sinh(2.0 * beta)
    k = 2.0 * ss / cc**2

    def integrant(x):
        return log(1.0 + sqrt(abs(1.0 - k * k * cos(x)**2)))

    integral, err = integrate.quad(integrant, 0, 0.5 * pi, epsabs=1e-13,
        epsrel=1e-13)
    result = integral / pi + log(cc) + 0.5 * log(2.0)
    return -result / beta

```

```python

def SVD(t, left_indices, right_indices, D):
    '''
    Perform singular value decomposition for a tensor
    Return U out of U, s, V.
    '''
    T = torch.permute(t, tuple(left_indices + right_indices)) if use_cuda
        else np.transpose(t, left_indices + right_indices)
    left_index_sizes = [T.shape[i] for i in range(len(left_indices))]
    right_index_sizes = [T.shape[i] for i in range(len(left_indices),
        len(left_indices) + len(right_indices))]
    xsize, ysize = np.prod(left_index_sizes), np.prod(right_index_sizes)
    T = torch.reshape(T, (xsize, ysize)) if use_cuda else np.reshape(T,
        (xsize, ysize))
    U, _, _ = torch.linalg.svd(T, full_matrices=False) if use_cuda else
        sp.linalg.svd(T, full_matrices=False)
    size = np.shape(U)[1]
    D = min(size, D)
    U = U[:, :D]
    U = torch.reshape(U, tuple(left_index_sizes + [D])) if use_cuda else
        np.reshape(U, left_index_sizes + [D])
    return U


def coarse_graining(t):

    Tfour = ee.einsum('jabe,iecd,labf,kfcd->ijkl', t, t, t, t) if use_cuda
        else contract('jabe,iecd,labf,kfcd->ijkl', t, t, t, t)
    U = SVD(Tfour,[0,1],[2,3],D_cut)
    Tx = ee.einsum('abi,bjdc,acel,edk->ijkl', U, t, t, U) if use_cuda else
        contract('abi,bjdc,acel,edk->ijkl', U, t, t, U)
    Tfour = ee.einsum('aibc,bjde,akfc,flde->ijkl',Tx,Tx,Tx,Tx) if use_cuda
        else contract('aibc,bjde,akfc,flde->ijkl',Tx,Tx,Tx,Tx)
    U = SVD(Tfour,[0,1],[2,3],D_cut)
    Txy = ee.einsum('abj,iacd,cbke,del->ijkl', U, Tx, Tx, U) if use_cuda else
        contract('abj,iacd,cbke,del->ijkl', U, Tx, Tx, U)
    norm = torch.max(Txy) if use_cuda else np.max(Txy)
    Txy /= norm
    return Txy, norm


def weights(index, beta, delta):
    return sum([iv(index-2.0*j, beta*delta)*iv(j, beta*(1.0-delta)) for j in
        range(-mcut, mcut+1)])


def init_tensors(model):
```

```
127
128     if model == 'GXY' or model == 'XY':
129
130         L = [sqrt(weights(i, beta, delta)) for i in range(-Dn, Dn+1)]
131
132         if use_cuda: t1 = torch.tensor(L)
133         out = ee.einsum('i,j,k,l->ijkl', t1,t1,t1,t1) if use_cuda else
                contract('i,j,k,l->ijkl', L, L, L, L)
134
135         for l,r,u,d in product(range (-Dn,Dn+1), repeat=4):
136
137             index = l+u-r-d
138             if index != 0:
139                 out[l+Dn][r+Dn][u+Dn][d+Dn] = 0.0
140
141         return out
142
143
144     if model == 'Ising':
145
146         tau = 1 # This is np.exp(0.250000*beta*h) for finite 'h'.
147         #tau = np.exp(0.250000*beta*)
148         #a = np.sqrt(np.cosh(beta))
149         #b = np.sqrt(np.sinh(beta))
150         #W = np.array([[a*tau,b*tau],[(a/tau),-(b/tau)]])
151         W =
                np.array([[np.exp(beta),np.exp(-beta)],[np.exp(-beta),np.exp(beta)]])
152         W = LA.cholesky(W)
153
154         if use_cuda:
155             t1 = torch.tensor(W)
156             out = ee.einsum('ia,ib,ic,id->abcd', t1,t1,t1,t1)
157         else:
158             out = contract("ia, ib, ic, id -> abcd", W, W, W, W)
159
160         return out
161
162
163     if model == 'Potts':
164
165         Wnew = np.zeros((qstate, qstate))
166         for i in range (qstate):
167             for j in range (qstate):
168                 if i == j:
169                     Wnew[i][j] = np.exp(beta)
170                 else:
171                     Wnew[i][j] = 1.
```

```
        L = LA.cholesky(Wnew)

        if use_cuda:
            L = torch.tensor(L)
            out = ee.einsum("ia, ib, ic, id -> abcd", L, L, L, L)
        else:
            out = contract("ia, ib, ic, id -> abcd", L, L, L, L)

        return out


if __name__ == "__main__":

    start = time.time()

    T = init_tensors(model)
    norm = torch.max(T) if use_cuda else np.max(T)
    T = T/norm
    C = log(norm)

    for i in range(Niters):

        print ("Iteration #",i+1,"Timestamp:",datetime.now())
        T, norm = coarse_graining(T)
        torch.cuda.empty_cache()
        C += np.log(norm)/4**(i+1)

        if i > Niters-4:
        # Only compute free energy in the last few iterations

            Z1 = ee.einsum('aibj,bkal->ijkl', T, T) if use_cuda else
                contract('aibj,bkal->ijkl', T, T)
            Z = ee.einsum('abcd,badc->''', Z1, Z1) if use_cuda else
                contract('abcd,badc->''', Z1, Z1)
            if Z > 0:
               Free = -Temp * (C + (np.log(Z)/(4**Niters)))
            else:
                print ("WARNING: Z !> 0 ")

    end = time.time()

    if model == 'Ising':
        exact = exact_free_energy_Ising(Temp)
        print ("Exact answer:", exact)
        error_in_f_from_exact = abs((Free-exact)/(exact))
```

```
217
218
219     path = os.path.join('./', str(model) + '_data')
220     if not os.path.exists(path):
221         os.makedirs(path)
222
223
224     fileout = model + str(int(datetime.now().strftime("%Y%m%d%H%M%S"))) +
            '_GPU' + '_N' + str(Niters) + '_D' + str(D_cut) + '.txt' if use_cuda
            else model + str(int(datetime.now().strftime("%Y%m%d%H%M%S"))) +
            '_CPU' + '_N' + str(Niters) + '_D' + str(D_cut) + '.txt'
225     f=open(os.path.join(path, fileout), "a+")
226
227     if model == 'GXY' or model == 'XY':
228         f.write("%4.10f \t %4.14f \t %2.0f \t %2.0f \t %2.4f \t %6.2f \n" %
                (Temp, Free, Niters, D_cut, delta, end-start))
229         f.close()
230     elif model == 'Ising':
231         f.write("%4.10f \t %4.10f \t %2.0f \t %2.0f \t %2.3e \t %6.2f \n" %
                (Temp, Free, Niters, D_cut, error_in_f_from_exact, end-start))
232         f.close()
233     elif model == 'Potts':
234         f.write("%4.10f \t %4.10f \t %2.0f \t %2.0f \t %2.0f \t %6.2f \n" %
                (Temp, Free, Niters, D_cut, qstate, end-start))
235         f.close()
236
237     print ("COMPLETED", datetime.now())
238     print("Run time (in seconds):", round(end-start,2))
```

6 Example II – 2d classical XY model

In this exercise, we will construct tensor formulation of classical XY model in two dimensions with $h = 0$ and calculate the free energy to reproduce the plot given in 10 from [6].

We start with the fundamental tensor (four legs) which sits on each lattice site and tiles the lattice.

$$T_{i,j,k,l} = \sqrt{I_l(\beta)I_r(\beta)I_u(\beta)I_d(\beta)}I_{i+k-j-l}(\beta h), \tag{6.1}$$

where indices (i, j, k, l) denote the four legs of the tensor. The length of each leg, called bond dimension, is infinite in principle from the character expansion formula. The coefficient $I_n(\beta)$ decreases exponentially with increasing n. Thus we can truncate the series and approximate $T_{i,j,k,l}$ by a tensor with finite bond dimension D with high precision. This leads to a finite-dimensional tensor representation for the partition function

$$Z = \mathrm{Tr}\prod_s T_{i_s,j_s,k_s,l_s}. \tag{6.2}$$

A bond links two local tensors. The two bond indices defined from the two end points are implicitly assumed to take the same values. For example, if the bond connecting i and j along the x direction, then $r_i = l_j$. The trace is to sum over all bond indices.

Magnetization is defined as,

$$m = \frac{1}{\beta}\frac{\partial \ln Z}{\partial h} = \frac{I_{l+u-r-d-1}(\beta h) + I_{l+u-r-d+1}(\beta h)}{2I_{l+u-r-d}(\beta h)} \tag{6.3}$$

GPU acceleration [7]

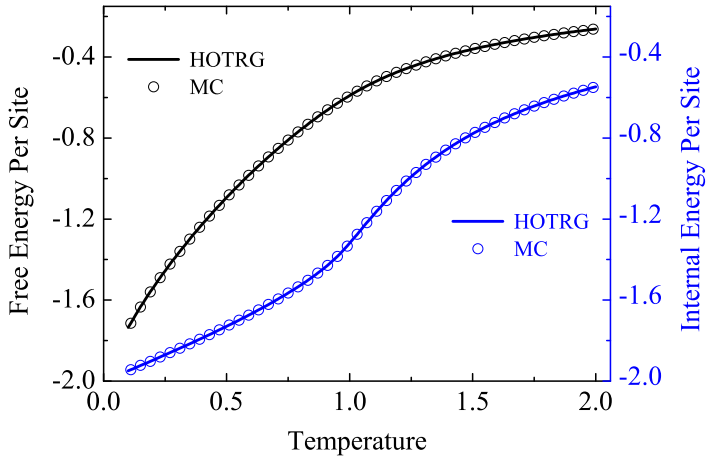

Figure 10. The free and internal energy of the 2d XY model.

7 Example III – Entanglement Entropy

Consider dividing a Hilbert space, $\mathcal{H}$ into a product of two spaces as, $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$ corresponding to sub-systems A and B. Then the subsystem A can be described by,

$$\rho_A = \mathrm{Tr}_B\ \rho = \sum_i \langle \psi_B^i | \rho | \psi_B^i \rangle \tag{7.1}$$

where the Tr is only over the $\mathcal{H}_B$. Then the entanglement entropy (also von Neumann, sometimes also bipartite entanglement entropy) entropy is defined as,

$$S_A = -\mathrm{Tr}_A\Big(\rho_A \ln \rho_A\Big) \tag{7.2}$$

Some nice properties of S_A are mentioned below:

- $S_A(\rho_A)$ is maximum when the state is maximally entangled. In such a case, $S_A(\rho_A) = \ln(\dim \mathcal{H}_A)$
- If ρ_A is a pure state (*i.e.* $\rho^2 = \rho$), then $S_A = 0$

- S_A is constant under change of basis (unitary), *i.e.* $S_A(\rho_A) = S_A(U\rho_A U^\dagger)$

Consider a state like,

$$|\psi\rangle = \cos\theta|\downarrow\uparrow\rangle + \sin\theta|\uparrow\downarrow\rangle \tag{7.3}$$

Then we can write, $\rho_A = \mathrm{Tr}_B\ \rho$ as,

$$\rho_A = \cos^2\theta|\downarrow\rangle\langle\downarrow| + \sin^2\theta|\uparrow\rangle\langle\uparrow| \tag{7.4}$$

which gives entropy as,

$$S_A = -\cos^2\theta \mathrm{log}\cos^2\theta - \sin^2\theta \mathrm{log}\sin^2\theta \tag{7.5}$$

Note that at $\theta = \pi/4$, the entropy is maximum and the corresponding state is maximally entangled.

EE CODE

```
# This code generates a random state of "N" qubits and
# then computes the reduced density matrix of first "p" qubits.
# Then it calculates the entanglement entropy. Note that
# the entropy will be "p".

import sys
import math
from math import *
import numpy as np
from scipy import special
from numpy import linalg as LA
from numpy.linalg import matrix_power
from numpy import ndarray
import time

N=24
p=4   # Becomes expensive with increasing $p$. Don't try p>10

Psi = np.random.randn(2**N)
# 2^N random coefficients
Psi = Psi/LA.norm(Psi)

A = Psi.reshape(2**p, 2**(N-p))
Rho = np.dot(A, np.transpose(A).conj())

def comEE(Rho):
   u,v = LA.eig(Rho)
```

```
    chi = u.shape[0]
    #print ("Shape of u", np.shape(u)) # 2^p
    #print ("Shape of v", np.shape(v)) # 2^p x 2^p
    EE = 0
    for n in range (0 , chi):
       if u[n] > 0:
          EE += -u[n] * math.log(u[n],2)
    return EE

if __name__ == "__main__":

    entropy = comEE(Rho)
    print ("Entanglement Entropy is", entropy)

# S_exact = -rho log2(rho) = -1/d * ln (1/d) summed 'd' times i.e. log2(d) =
    log2(2^p) = p
```

8 Homework problem!

It is also possible to formulate the tensor network for Wilson's action for SU(2) gauge group in two dimensions. This was done in [8]. Try to do this. You can also refer to my GitHub page to see the code if you want here.

9 Details

Fuchs (21.50) reads (denoting $d_\Lambda = \dim(V_\Lambda) = 2\Lambda + 1$ and G is the group manifold,

$$\int_G d\mu_\gamma D_\Lambda^{m_1 n_1}(\gamma) D_{\Lambda'}^{m_2 n_2}(\gamma) = \frac{1}{d_\Lambda} \delta_{\Lambda\Lambda'} \delta_{m_1 m_2} \delta_{n_1 n_2} \tag{9.1}$$

We can re-write two Wigner D-matrices as a single one using,

$$D_\Lambda^{m_1 n_1} D_{\Lambda'}^{m_2 n_2} = \sum_{R=|\Lambda-\Lambda'|}^{\Lambda+\Lambda'} \sum_{m,n} C_{m_1 m_2 m}^{\Lambda\Lambda' R} D_{mn}^R C_{n_1 n_2 n}^{\Lambda\Lambda' R} \tag{9.2}$$

$$D_\sigma^{nn} \tilde{D}_{\frac{1}{2}}^{\alpha\beta} = \sum_{R=|\sigma-\frac{1}{2}|}^{|\sigma+\frac{1}{2}|} \sum_{M,N} C_{n\alpha M}^{\sigma \frac{1}{2} R} D_{MN}^R C_{n\beta N}^{\sigma \frac{1}{2} R} \tag{9.3}$$

Post-multiplying Equation (13) in notes by $\tilde{D}_{\alpha\beta}^{\frac{1}{2}}$

$$\int dU D_{m_{lb} m_{la}}^{r_l} D_{m_{ra} m_{rb}}^{r_r \dagger} D_{nn}^\sigma \tilde{D}_{\alpha\beta}^{\frac{1}{2}} = \sum_{n=-\sigma}^{n=\sigma} \frac{1}{2r_r + 1} C_{r_l m_{lb} \sigma n}^{r_r m_{rb}} C_{r_l m_{la} \sigma n}^{r_r m_{ra}} \sum_{R=|\sigma-\frac{1}{2}|}^{|\sigma+\frac{1}{2}|} \sum_{M,N} C_{n\alpha M}^{\sigma \frac{1}{2} R} D_{MN}^R C_{n\beta N}^{\sigma \frac{1}{2} R} \tag{9.4}$$

We can store this as a four-index object $\tilde{A}$ given by ,

$$\tilde{A}_{(r_l,m_{lb},m_{la})(r_r,m_{ra},m_{rb})\alpha\beta} = \frac{1}{2r_r+1}\sum_{M,N=-R}^{M,N=R} C^{r_r m_{rb}}_{r_l m_{lb} RM} C^{r_r m_{ra}}_{r_l m_{la} RN} \sum_{R=|\sigma-\frac{1}{2}|}^{|\sigma+\frac{1}{2}|} C^{RM}_{\sigma\alpha\frac{1}{2}\beta} D^R_{MN} C^{RN}_{\sigma n\frac{1}{2}n} \tag{9.5}$$

ANOTHER : four-index object $\tilde{A}$ given by ,

$$\tilde{A}_{(r_l,m_{lb},m_{la})(r_r,m_{ra},m_{rb})\alpha\beta} = \frac{1}{2r_r+1}\sum_{M,N=-R}^{M,N=R} C^{r_r m_{rb}}_{r_l m_{lb} RM} C^{r_r m_{ra}}_{r_l m_{la} RN} \sum_{R=|\sigma-\frac{1}{2}|}^{|\sigma+\frac{1}{2}|} C^{\sigma\alpha}_{\frac{1}{2}nRM} D^R_{MN} C^{\sigma\beta}_{RN\frac{1}{2}n} \tag{9.6}$$

Note that the above expression is just the statement that Haar measure projects out the trivial/singlet ....

The quenched BFSS action reads,

$$S = \frac{N}{4\lambda}\sum_t \left[-\mathrm{Tr}\Big(X_i(t)U(t)X_i(t+1)U + X_i^2 \Big) + \sum_{i<j} \mathrm{Tr}\Big(X_i X_j X_i X_j \Big) - \sum_{i<j} \mathrm{Tr}\Big(X_i X_j X_j X_i \Big) \right] \tag{9.7}$$

Haar measure for a $\mathbb{U}(N)$ group is given by,

$$dU = \frac{1}{(2\pi)^N} \prod_{1\le j\le k\le N} \left| e^{i\theta_j} - e^{i\theta_k} \right|^2 \prod_{j=1}^{N} d\theta_j \tag{9.8}$$

with, $-\pi \le \theta_1 < \cdots \le \pi$ and $e^{i\theta_1}, e^{i\theta_2}$... are eigenvalues of $U \in \mathbb{U}(N)$

Some results for group integration,

$$\int dU 1 = \int dU \det(U) \qquad = 1 \tag{9.9}$$

$$\int dU U \qquad = 0 \tag{9.10}$$

$$\int dU U_{ij} U^\dagger_{kl} \qquad = \frac{\delta_{il}\delta_{jk}}{N} \tag{9.11}$$

$$\int dU U_{i_1 j_1} \cdot \cdots \cdot U_{i_N j_N} = \frac{1}{N!} \epsilon_{i_1 \cdots i_N} \epsilon_{j_1 \cdots j_N} \tag{9.12}$$

$$\tag{9.13}$$

Another useful relation is,

$$Z_{\alpha,\beta}(M,N) = \int \mathcal{D}U \Big(\mathrm{Tr}(MU) \Big)^\alpha \Big(\mathrm{Tr}(NU^\dagger) \Big)^\beta \tag{9.14}$$

The case where $\alpha = \beta$ is the 'ordinary case'. Then, $Z_{\alpha,\alpha}$ (M,N) is a well known function of MN also known as 'Weingarten's function'.

The character $\chi_r(U)$ is the trace of U in the irreducible representation indexed by r. We have U, which is unitary and diagonalizable and hence the character (trace) is the sum of its eigenvalues. The dimesionality of the representation is the trace of U in the singlet/trivial representation.

$$e^{-S_\square(U,a)} = \sum_r F_r(a) d_r \chi_r(U) \tag{9.15}$$

where,

$$F_r(a) = \frac{1}{d_r} \int dU e^{-S_\square(U,a)} \chi_r^*(U) \tag{9.16}$$

9.15 is also known as character expansion.

Integral representation of $I_n(\beta)$

$$I_n(\beta) = \frac{1}{\pi}\int_0^\pi dx\ e^{\beta\cos(x)}\cos(nx) \tag{9.17}$$

Recently, tensor network renormalization (TNR) was proposed. It uses the disentanglers to remove short-range correlations along with the usual isometries. These disentanglers appeared earlier in the work of the multi-scale entanglement renormalization ansatz (MERA). For a fixed bond dimension (χ), TNR gives significantly more accurate results compared to TRG, but at the cost of increasing the computational complexity. However, it is not clear how to extend the approach of TNR to systems in higher dimensions, whereas the HOTRG/HOSRG mehtods have been applied to three-dimensional systems.

In arbitrary dimension, the scale invariance (also known as criticality of the statistical model) together with translational and rotational (i.e. Lorentz in SO(d,1)) invariance and locality automatically leads to conformal invariance explaining the very large interest in these theories

The formula for the character of the irreducible representation of SU(3) with highest weight (p,q) is

$$\chi^{p,q}(\theta,\phi) = e^{i\theta(p+2q)}\sum_{k=q}^{p+q}\sum_{l=0}^{q} e^{-3i(k+l)\theta/2}\left(\frac{\sin((k-l+1)\phi/2)}{\sin(\phi/2)}\right) \tag{9.18}$$

$$\chi^{1,0}(\theta,\phi) = e^{i\theta}\sum_{k=0}^{1}\sum_{l=0}^{0} e^{-3i(k+l)\theta/2}\left(\frac{\sin((k-l+1)\phi/2)}{\sin(\phi/2)}\right) \tag{9.19}$$

$$= e^{i\theta} + e^{i(\phi-\theta)/2} + e^{-i(\phi+\theta)/2} \tag{9.20}$$

9.1 GHZ state

```
from qutip import *
import numpy as np

nqubits = 12

ghz = ghz_state(nqubits)
rho = ket2dm(ghz)
print("VN Entropy of pure state:", entropy_vn(rho,2))
print ("Check purity of the state", np.trace(rho), np.trace(rho * rho))
rho1 = rho.ptrace([0,1,2,3]) # Split system into 4 and 8 qubits, trace over one
print ("Purity check", np.trace(rho1), np.trace(rho1 * rho1))
# Purity. Should be 'maximally mixed'.
print("VN Entropy", entropy_vn(rho1,2))
```

```
#vN in base 2. If you want to use natural log, exclude 2.
```

10 Three dimensions

3d O(2) model: [9]

References

[1] R. Orus, "Exploring corner transfer matrices and corner tensors for the classical simulation of quantum lattice systems," *Phys. Rev.* **B85** (2012) 205117, arXiv:1112.4101 [cond-mat.str-el].

[2] P. Pfeuty, "The one-dimensional Ising model with a transverse field," *Annals Phys.* **57** no. 1, (1970) 79–90.

[3] S.-J. Ran, E. Tirrito, C. Peng, X. Chen, L. Tagliacozzo, G. Su, and M. Lewenstein, "Lecture notes of tensor network contractions," arXiv:1708.09213.

[4] G. M. Viswanathan, "The hypergeometric series for the partition function of the 2d ising model," *Journal of Statistical Mechanics: Theory and Experiment* **2015** no. 7, (Jul, 2015) P07004. http://dx.doi.org/10.1088/1742-5468/2015/07/P07004.

[5] H.-J. Liao, J.-G. Liu, L. Wang, and T. Xiang, "Differentiable programming tensor networks," *Phys. Rev. X* **9** (Sep, 2019) 031041. https://link.aps.org/doi/10.1103/PhysRevX.9.031041.

[6] J. F. Yu, Z. Y. Xie, Y. Meurice, Y. Liu, A. Denbleyker, H. Zou, M. P. Qin, and J. Chen, "Tensor Renormalization Group Study of Classical XY Model on the Square Lattice," *Phys. Rev.* **E89** no. 1, (2014) 013308, arXiv:1309.4963 [cond-mat.stat-mech].

[7] R. G. Jha and A. Samlodia, "GPU-acceleration of tensor renormalization with PyTorch using CUDA," *Comput. Phys. Commun.* **294** (2024) 108941, arXiv:2306.00358 [hep-lat].

[8] A. Bazavov, S. Catterall, R. G. Jha, and J. Unmuth-Yockey, "Tensor renormalization group study of the non-Abelian Higgs model in two dimensions," *Phys. Rev.* **D99** no. 11, (2019) 114507, arXiv:1901.11443 [hep-lat].

[9] J. Bloch, R. G. Jha, R. Lohmayer, and M. Meister, "Tensor renormalization group study of the three-dimensional O(2) model," *Phys. Rev. D* **104** no. 9, (2021) 094517, arXiv:2105.08066 [hep-lat].