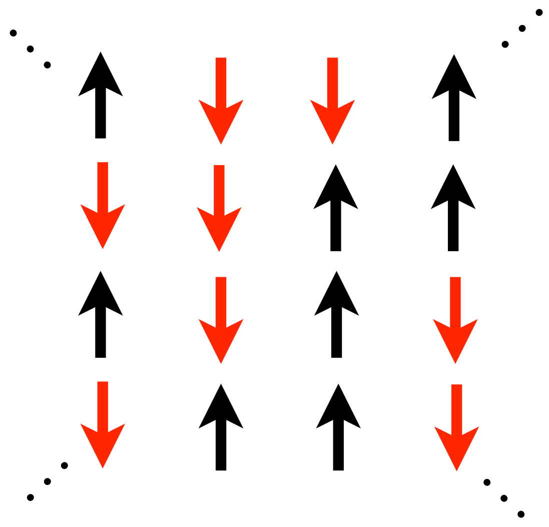
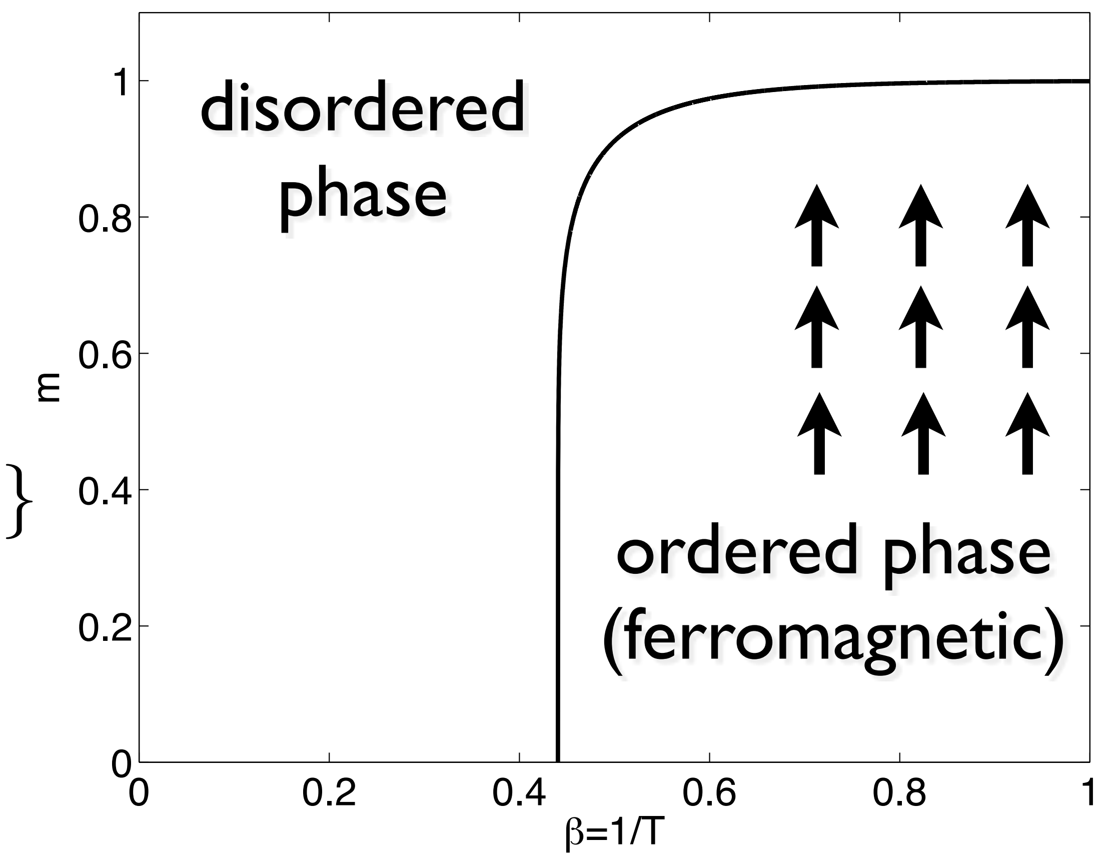
Exercise:
Corner-transfer matrix method for the 2D classical Ising model

slides / codes: tinyurl.com/4an74t8w

CTM method for the classical 2D Ising model



$$s_i \in \{+1, -1\}$$



$$H = \sum_{\langle i,j \rangle} H_b(s_i, s_j) = -\sum_{\langle i,j \rangle} s_i s_j,$$

$$\beta_c = \log(1+\sqrt{2})/2 \approx 0.44069$$

Partition function:

$$Z(\beta) = \sum_{\{c\}} \exp(-\beta H(c)) = \sum_{\{c\}} \prod_{\langle i,j \rangle} \exp(-\beta H_b(s_i, s_j))$$

GOAL: Compute m using tensor network methods

Magnetization per site:

$$m(\beta) = \frac{\sum_{\{c\}} s_r \exp(-\beta H(c))}{Z}$$

Exact solution:

$$= (1 - [\sinh(2\beta)]^{-4})^{1/8}, \quad \text{for } \beta > \beta_c$$

Represent partition function as a 2D TN

$$Q_{s_i s_j} = \exp(-\beta H_b(s_i, s_j)) = \begin{pmatrix} e^{\beta} & e^{-\beta} \\ e^{-\beta} & e^{\beta} \end{pmatrix}$$

$$Z(\beta) = \sum_{\{c\}} \exp(-\beta H(c)) = \sum_{\{c\}} \prod_{\langle i,j \rangle} \exp(-\beta H_b(s_i, s_j))$$

$$a_{ijkl} = \sum_s \left(\sqrt{Q}\right)_{is} \left(\sqrt{Q}\right)_{js} \left(\sqrt{Q}\right)_{ks} \left(\sqrt{Q}\right)_{ls}$$

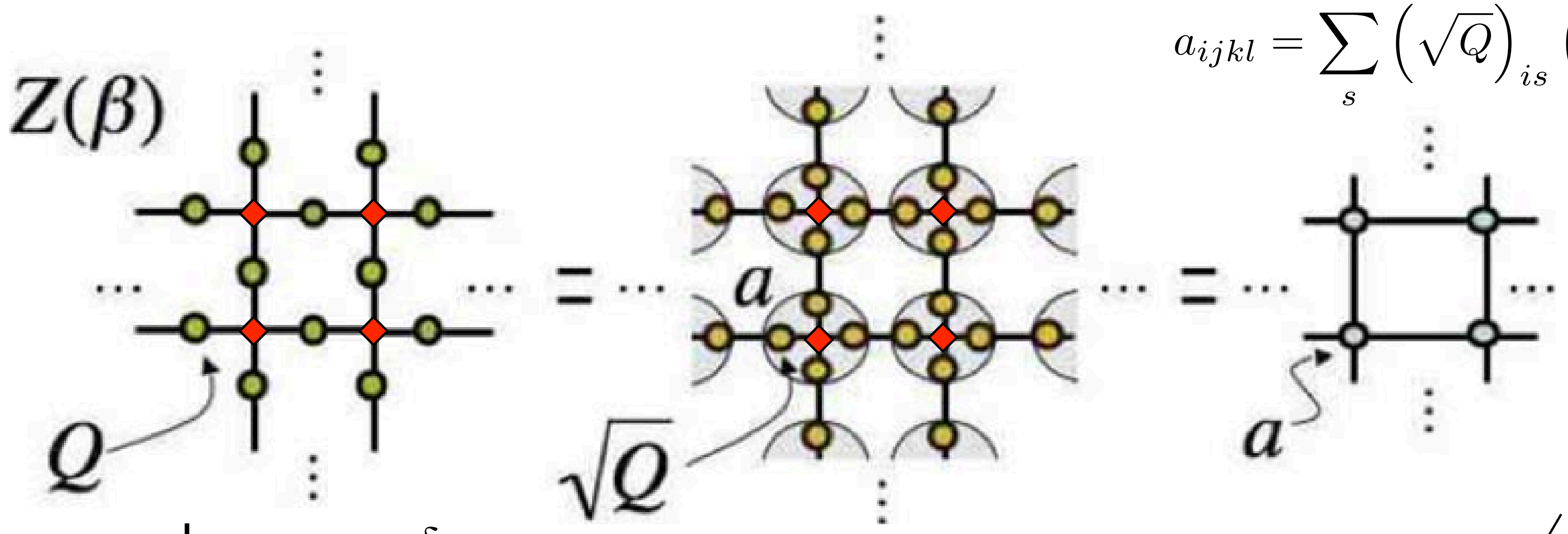
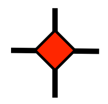


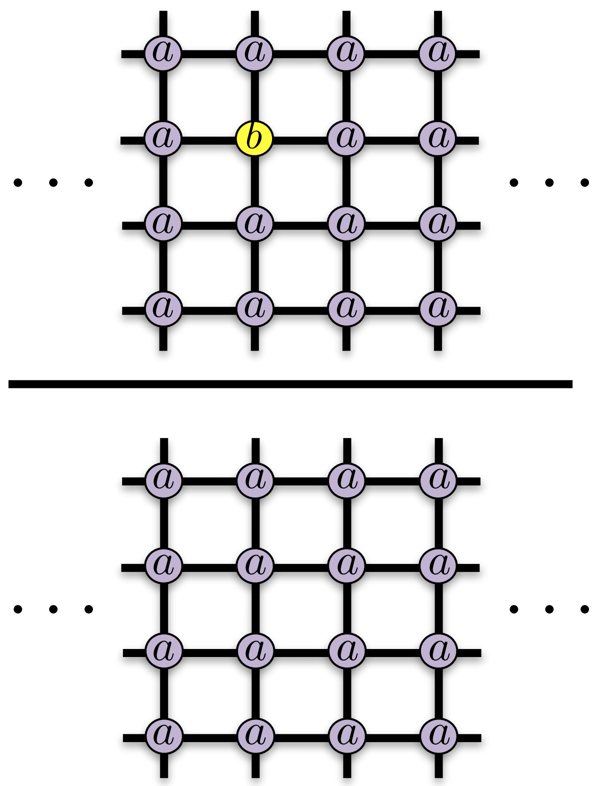
Figure taken from Orús & Vidal, PRB 78, 155117 (2008).

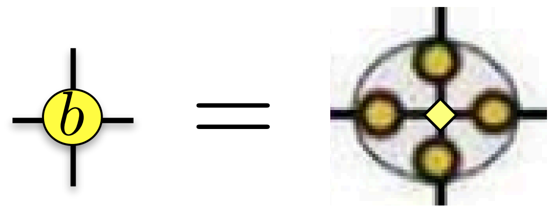


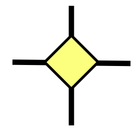
$$g_{ijkl} = \delta_{ijkl}$$

$$b_{ijkl} = \sum_s s \left(\sqrt{Q}\right)_{is} \left(\sqrt{Q}\right)_{js} \left(\sqrt{Q}\right)_{ks} \left(\sqrt{Q}\right)_{ls}$$

$$m(\beta) = \frac{\sum_{\{c\}} s_r \exp(-\beta H(c))}{Z} =$$



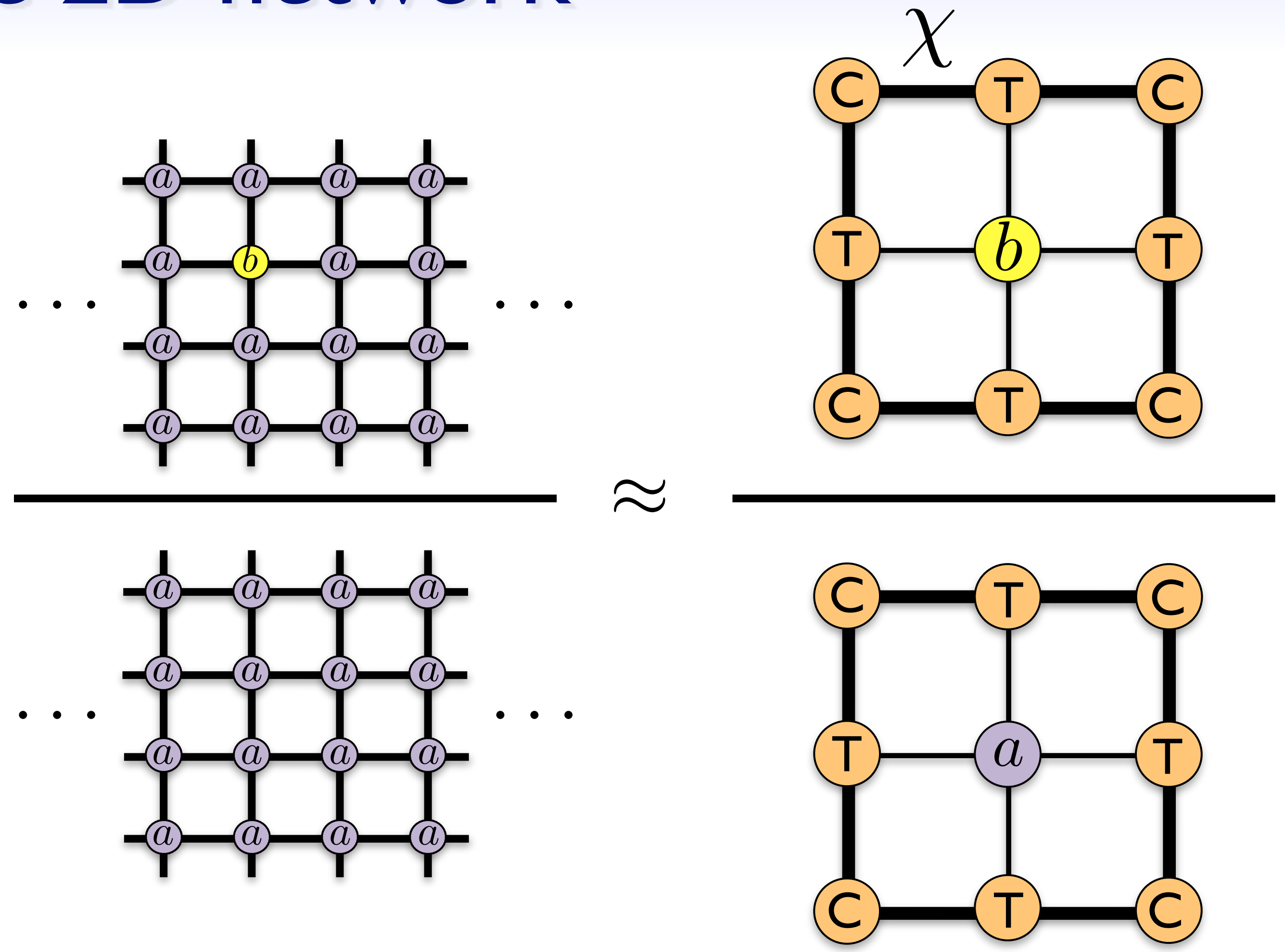




$$g'_{ijkl} = s_i \delta_{ijkl}$$

Use CTM to contract the 2D network

$$m(\beta) = \frac{\sum_{\{c\}} s_r \exp(-\beta H(c))}{Z} =$$



- Compute environment tensors C and T iteratively (CTM)
- Here: symmetric case: all corner/edge tensors the same and

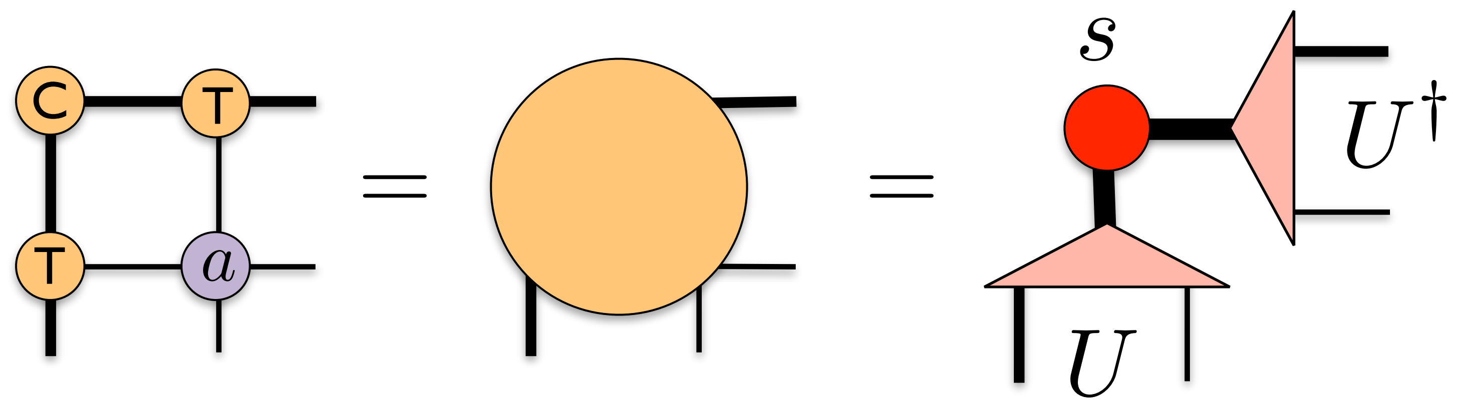
$$C_{ij} = C_{ji} \qquad T^k_{ij} = T^k_{ji}$$

- Start with random (symmetric) C and T, e.g. with $\chi_0 = 2$

CTM for rotational/mirror symmetric tensors

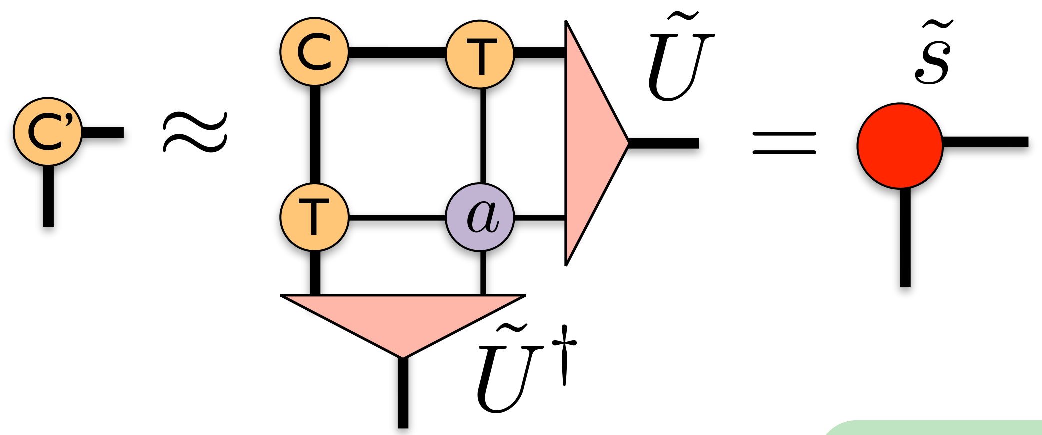
Nishino, Okunishi, JPSJ65 (1996)

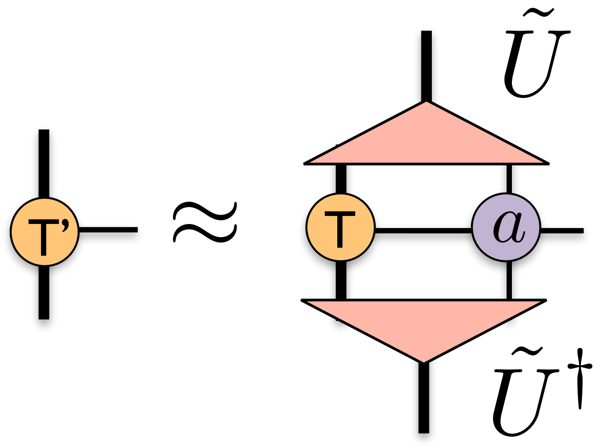
Eigenvalue decomposition (or SVD) of corner



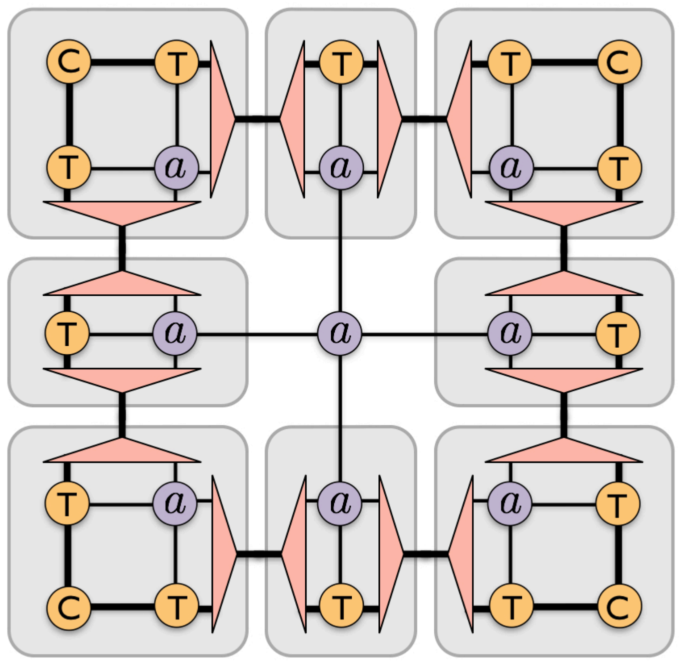
Note: reshape into a matrix before doing the eigenvalue decomposition. Then reshape U back to obtain the 3-legged U

Renormalized tensors: keep only χ states with largest weights





Keep numbers bounded:
e.g. divide each tensor by its largest element

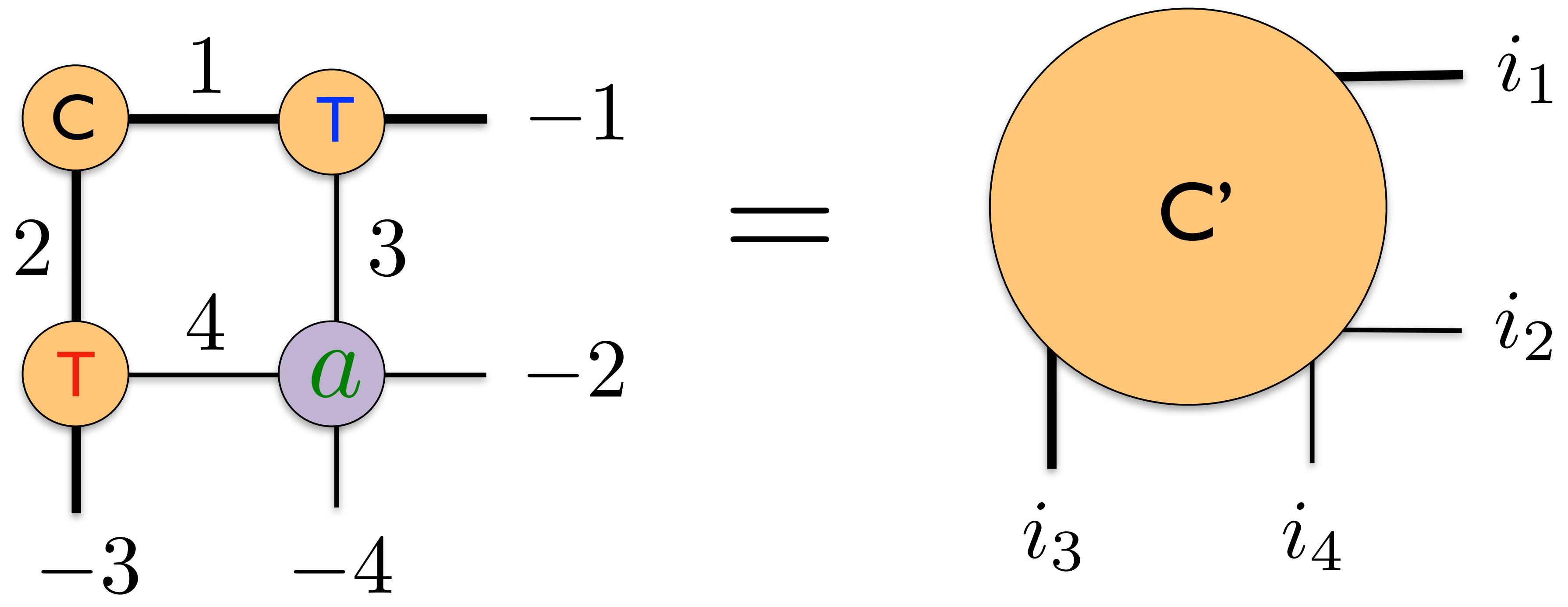


CTM algorithm implementation

- Start with random (symmetric) C and T, e.g. with $\chi_0 = 2$
- Do CTM renormalization steps, keeping (at most) a boundary dimension χ
  - ✦ *The method is converged once the change* $\sum_k |s_k - s'_k| < tol$ *where* s_k *(truncated & normalized) are the singular values of corner C*
  - ✦ *Due to round-off errors the tensors might not be perfectly symmetric anymore. For better numerical stability, symmetrize matrix before doing eig.*
- Once convergence is reached, quantities of interest (e.g. m) can be computed using the converged environment tensors C and T
- Further practical hints:
  - ✦ *use scipy.linalg.sqrtm to compute the square root of the Q matrix when constructing a, b*
  - ✦ *normalize and symmetrize each tensor after each step*
  - ✦ *use scipy.linalg.eigh to perform the eigenvalue decomposition*
  - ✦ *use numpy.transpose and numpy.reshape for permuting / reshaping tensors*
  - ✦ *make use of ncon for the tensor network contractions*

Contracting TNs using NCON

- NCON: Network contractor to conveniently contract TNs
- Written by R. N. C. Pfeifer, G. Evenbly, S. Singh, and G. Vidal, arXiv:1402.0939
- Example:

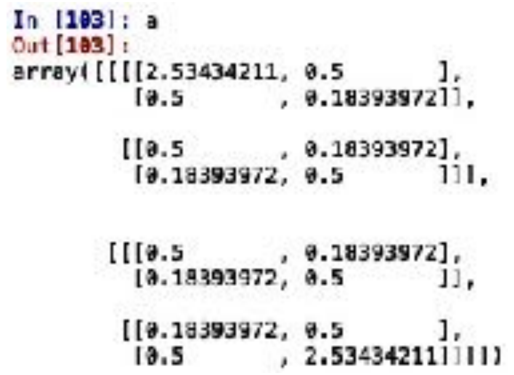


- Matlab code: Cp = ncon({C, T, T, a},{[1 2], [-1 1 3], [2 -3 4], [-2 3 4 -4]});
- Python implementation by Markus Hauru: https://github.com/mhauru/ncon

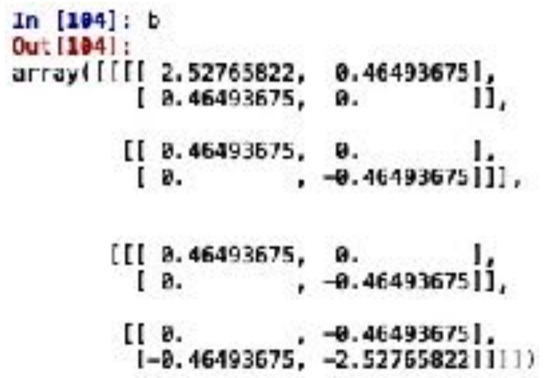
  Cp = ncon([C, T, T, a],([1,2], [-1,1,3], [2,-3,4], [-2,3,4,-4]));
- Complicated networks can be contracted in an easy way **in a single line!**

Implement and test your own code

- Test your code for beta=0.5. You can crosscheck your a, b tensors with these values:



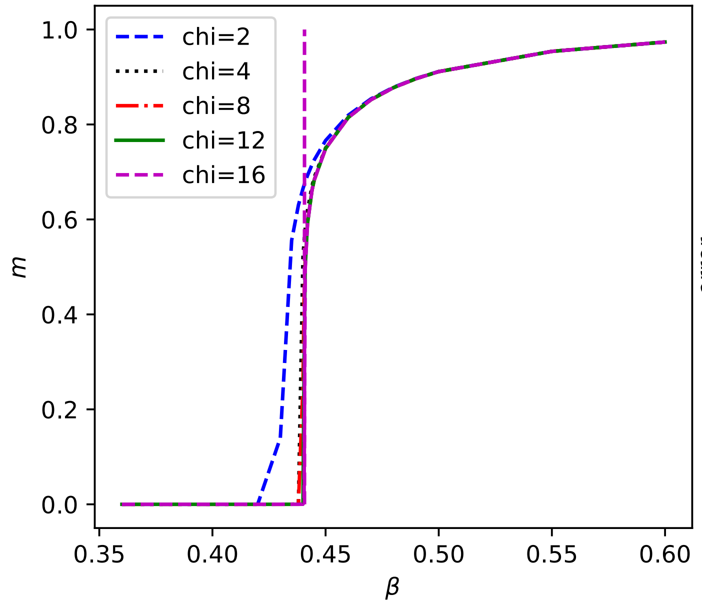
```
In [103]: a
Out[103]:
array([[[[2.53434211, 0.5       ],
         [0.5       , 0.18393972]],

        [[0.5       , 0.18393972],
         [0.18393972, 0.5       ]]],


       [[[0.5       , 0.18393972],
         [0.18393972, 0.5       ]],

        [[0.18393972, 0.5       ],
         [0.5       , 2.53434211]]]])
```

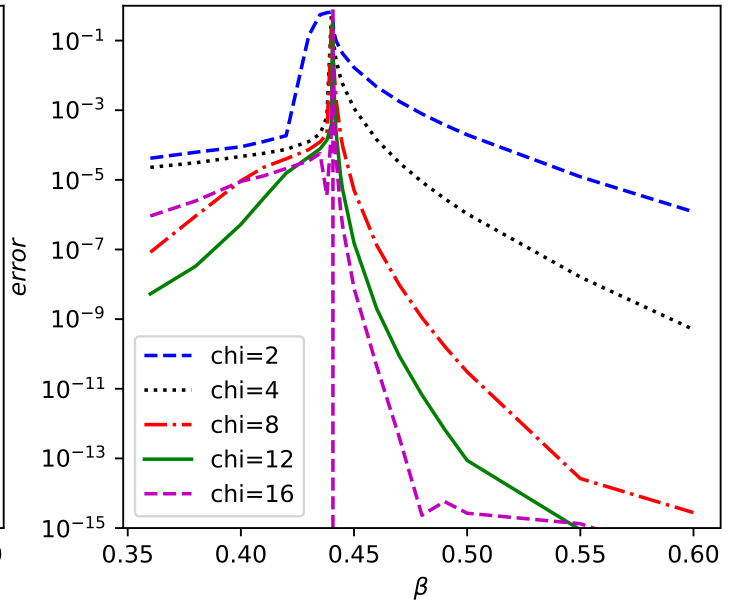


```
In [104]: b
Out[104]:
array([[[[ 2.52765822,  0.46493675],
         [ 0.46493675,  0.        ]],

        [[ 0.46493675,  0.        ],
         [ 0.        , -0.46493675]]],


       [[[ 0.46493675,  0.        ],
         [ 0.        , -0.46493675]],

        [[ 0.        , -0.46493675],
         [-0.46493675, -2.52765822]]]])
```

- Using chi=8 you should obtain for the magnetization: m = 0.911319

- Next, perform simulations for different values of beta and chi and observe the transition and involved finite chi effects (compare with the exact solution)



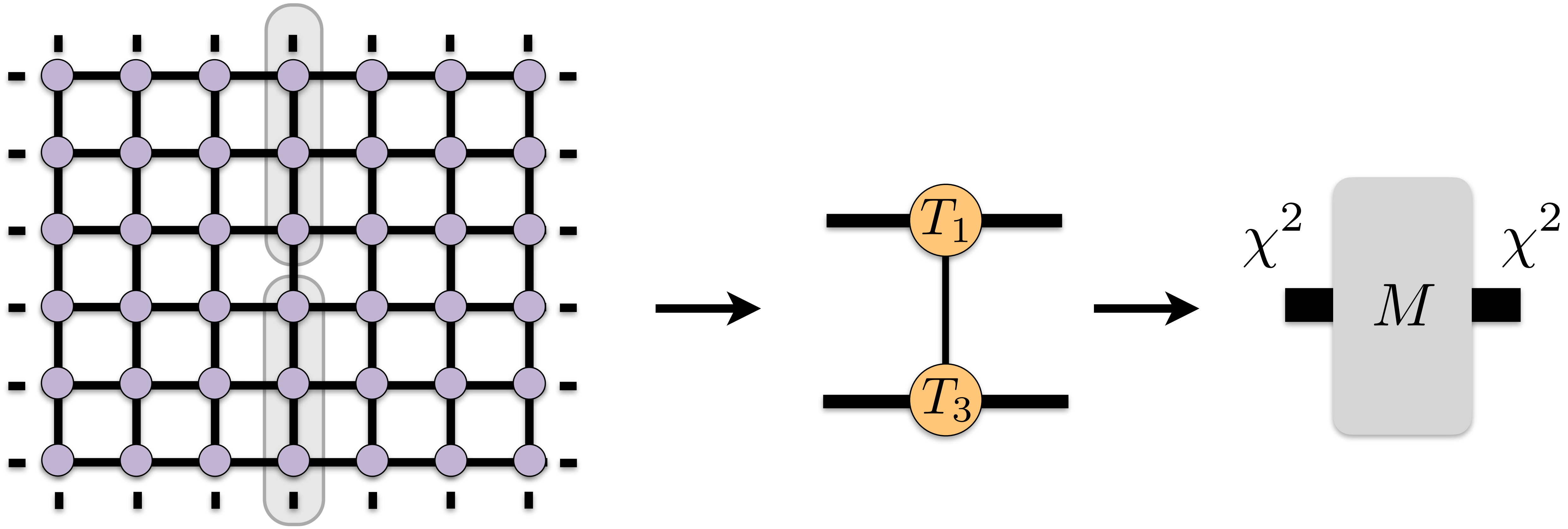


- If you are quick, you can also try to compute the energy (e.g. to get the specific heat) or the correlation length. The simulations can also easily be extended to include an external magnetic field.

- **Try it out! This is the easiest starting point to get into 2D TN**

Computing the correlation length with CTM

Nishino, Okunishi, Kikuchi, Physics Lett. A 213, 69 (1996)



$$\xi(\chi) = \frac{1}{\log(\frac{\lambda_0}{\lambda_1})}$$

1st and 2nd lowest eigenvalue of M