Quantum information and computing
Final project

Davide Bacilieri 2089214

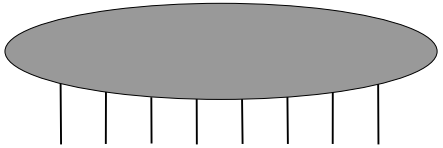
UNIVERSITÀ
DEGLI STUDI
DI PADOVA

What is an MPS?

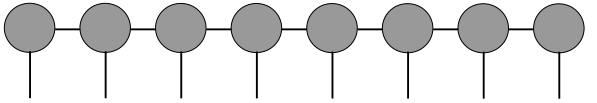
MPS = Matrix Product State, simplest ansatz for the representation of a generic many-body wavefunction.

Complete wavefunction (d^N coefficients):

$$|\psi\rangle = \sum_{n_1,\ldots,n_N=1}^{d} c_{n_1,\ldots,n_N} |n_1\rangle \otimes \ldots \otimes |n_N\rangle$$



VS

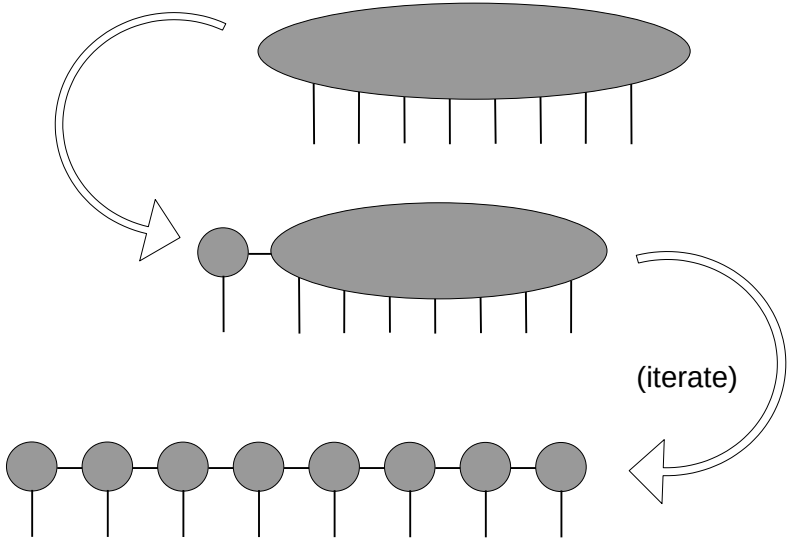


MPS representation ($Nd\chi^2$ coefficients):

$$|\psi\rangle = \sum_{n_1,\ldots,n_N=1}^{d} \sum_{a_1,\ldots,a_{N-1}}^{\chi} A^1_{n_1;a_1} A^2_{n_2;a_1,a_2} \ldots A^N_{n_N;a_{N-1}} |n_1\rangle \otimes \ldots \otimes |n_N\rangle$$

Why does it work?

It can be obtained from the complete representation by iteratively separating physical legs using an SVD, and cutting the result to the χ^{th} singular value.



When is it useful?

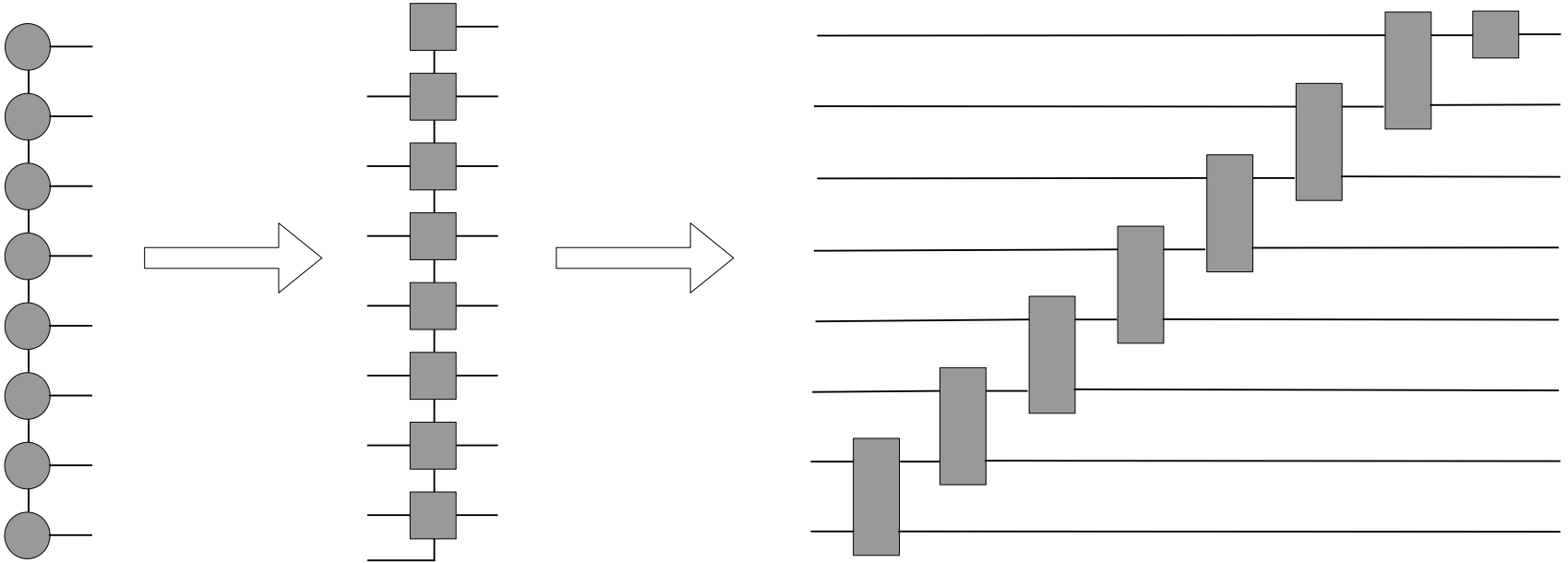
This ansatz can be very useful when searching for solutions to a 1D model with single-site and nearest-neighbor terms, while it struggles with longer-range interactions.
As all tensor network based approaches, it can only represent low-entanglement states, due to the cutting of the singular values.
Both of these drawbacks depend heavily on the maximum bond dimension used, with higher bond dimension allowing for longer range and higher entanglement, at the cost of longer computation times and higher memory requirements.

Qtealeaves & Qmatchatea

- Qtealeaves is the main library used for this project, containing the MPS and Tensor classes
- Qmatchatea builds upon it to allow an easier interface with qiskit and other simulator backends.
- In this project, the evolution of the states is obtained purely with the *apply_n_site_operator* functions of the MPS class.

Exact mapping for $\chi=d$

Each tensor is mapped into an operator; for the last tensor, the corresponding operator has the same values, while for the others the addition of one or two legs makes it necessary to search for additional entries, such that the matrices corresponding to G_{ij} form an orthonormal basis.
The circuit thus obtained is called Matrix Product Disentangler, or MPD.



Iterative mapping for χ>d

For higher bond dimension, a single layer is no longer enough to retrieve the full MPS state.

The simplest idea to get a higher fidelity is to build the MPD iteratively: the MPS is temporarily cut to χ=d, building the layer as shown before; then the process is repeated using the MPS after the existing layers are applied.

Another idea is to optimize the MPD by tuning its values, to achieve a higher fidelity without increasing the circuit depth. This is done by finding for each operator the tensor *F* that would optimize the circuit if it replaced the current operator; this can be calculated by contracting the whole circuit but the operator to be optimized to the given MPS to one side, and to the vacuum state on the other. Then the new operator is found as $U_{new} = U\left(U^{+}F\right)^{l}$, where l is a learning rate.

Optimization modes

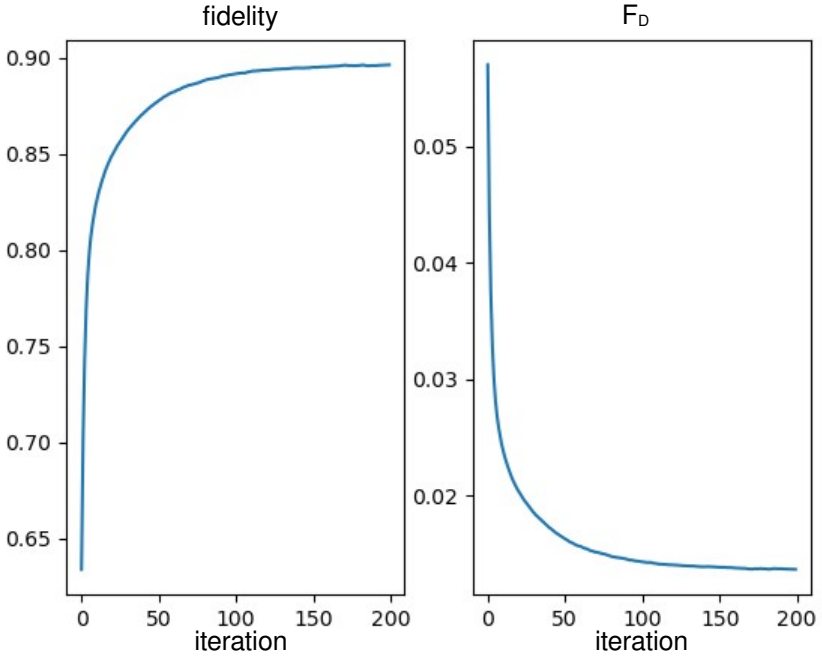
- “analytic”: no optimization, only create layers iteratively
- “optimize”: optimize starting from random tensors
- “final”: optimize all layers of *analytic* mode’s result
- “single”: optimize each layer after it is created analytically
- “batch”: optimize all layers after each is created analytically
- “identity”: optimize all layers after each is created as a set of identity operators
- “continuous”: optimize all layers of *single* mode’s result

All these work by finding the update for each tensor separately and applying them at the same time.

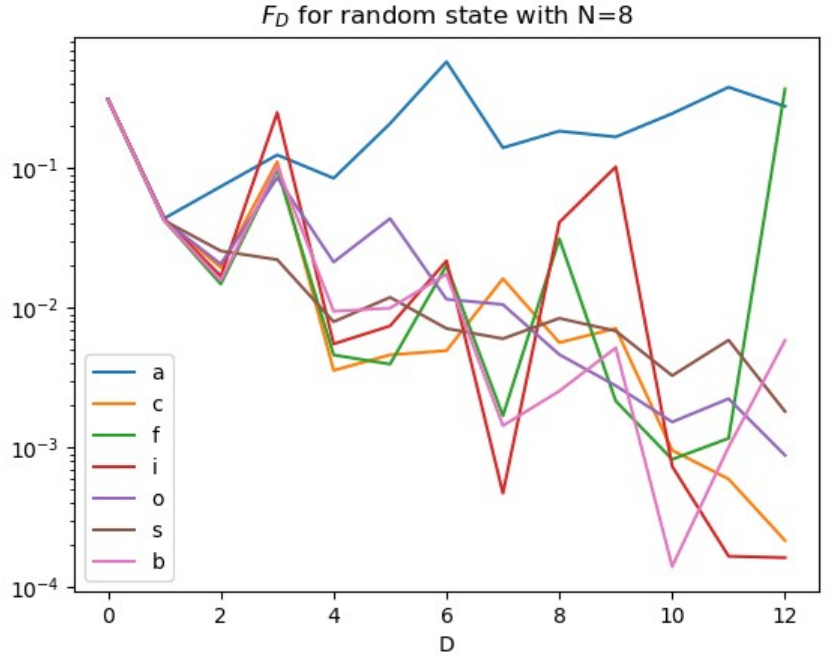
Results

"F_D" referenced in the figures is the negative logarithmic fidelity: $F_D = \frac{-\log|\langle 0|U|\psi\rangle|}{N}$

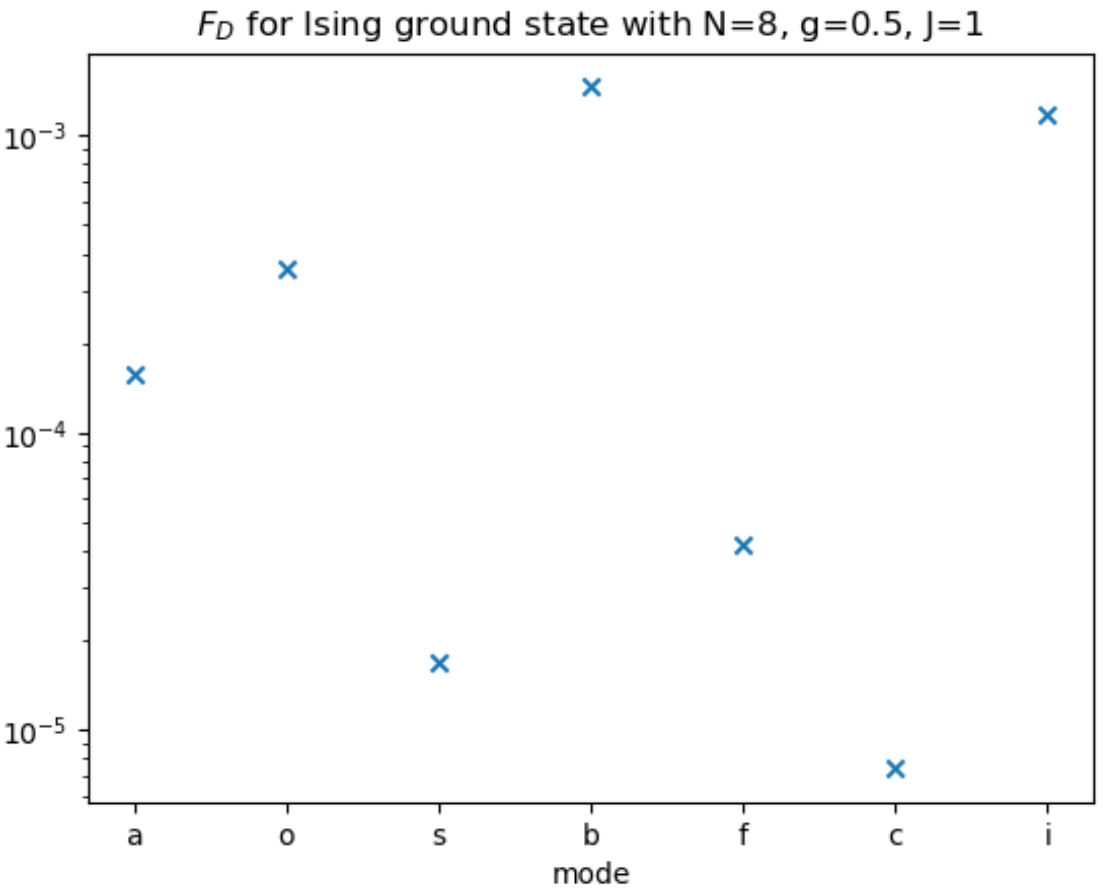
Example optimization run

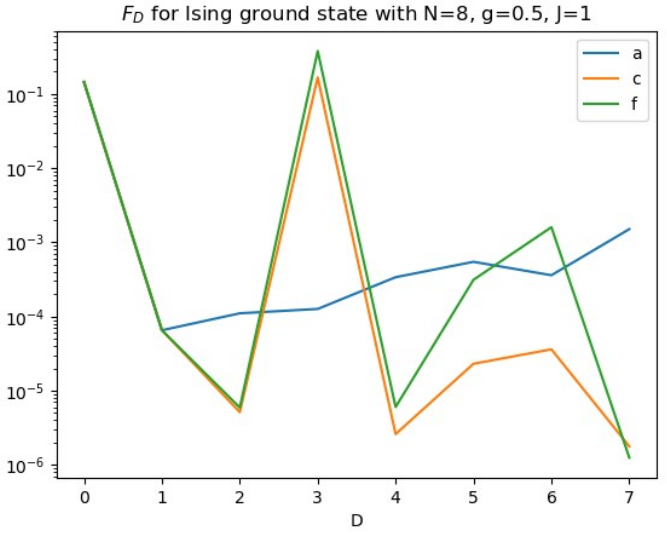


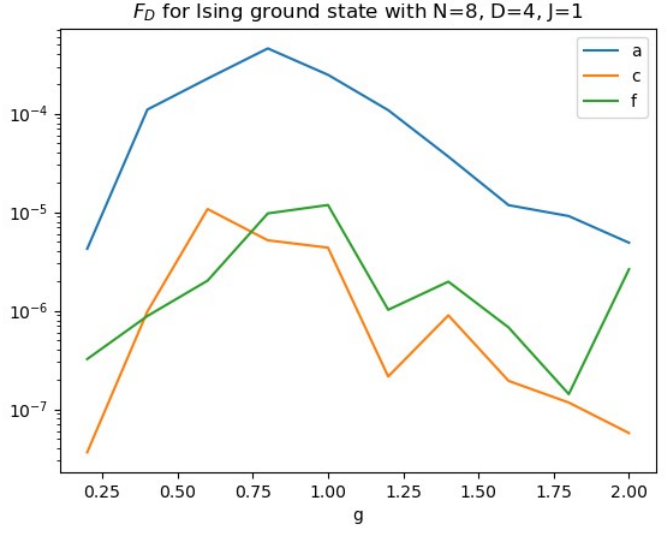
Mode comparison (χ=16)



In this case only the three most performing modes were kept







Next steps

- Create an MPD class with the functions developed until now as methods
- Reuse common quantities during optimization, eg the MPS contracted with the circuit up to a given layer
- Eventually parallelize the optimization process