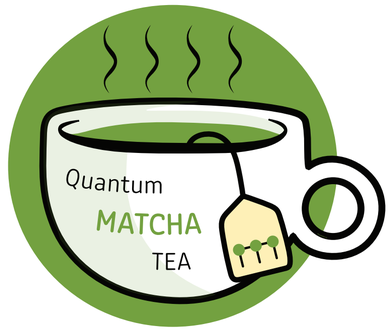
Tensor Network Hackathon guide:

Introduction to Quantum TEA

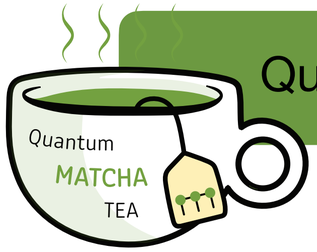
Quantum Tensor Network Emulator
Applications



Quantum TEA Python libraries

Quantum TEA Leaves

- different tensor network ansatzes
- Solves physical problems, such as Schrödinger equation



Quantum Matcha TEA

Uses Quantum TEA Leaves as dependency

- Quantum circuit simulator with MPS

None of the libraries

Projects that need it

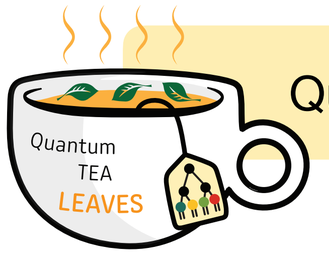
- Quantum machine learning
- Solving MaxCut
- Optimizing camera placement

- Knapsack problem
- Quantum Fourier transform
- Parallel scaling of tensor networks

- Tensor Renormalization Group

Installation of Qtea libraries

Git clone the repository or **Install directly via pip**

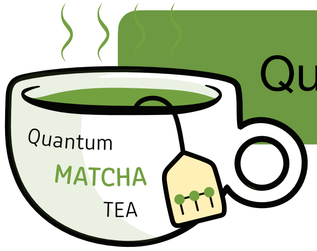


Quantum TEA Leaves

Source code: **here**

```
pip install qtealeaves
```

Documentation: **here**



Quantum Matcha TEA

Source code: **here**

```
pip install qmatchatea
```

Documentation: **here**

Remark:

To use the libraries directly via the source code, you need to copy or simbollically link qtealeaves/ qmatchatea folder to the same folder where the script is

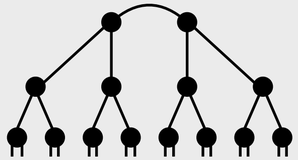
Available tensor network ansatzes

Quantum state representations:

Statevector

Matrix Product State

Tree Tensor Network



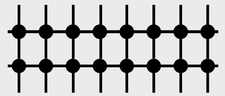
Useful for running tests on small system sizes

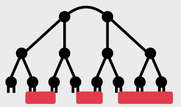
$N \leq 10$

Needed for scaling up towards larger system sizes

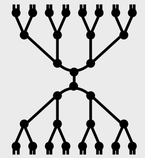
All the rest, not needed for hackathon

LPTN





aTTN



TTO

Main algorithms

Groundstate search

DMRG

TTN ✓ MPS ✓

- 1-tensor update
- Space link expansion

Time evolution

1st & 2nd order TDVP

TTN ✓ MPS ✓

- 1-tensor update
- 2-tensor update
- Space link expansion - not tested properly

Imaginary time evolution

1st order TDVP

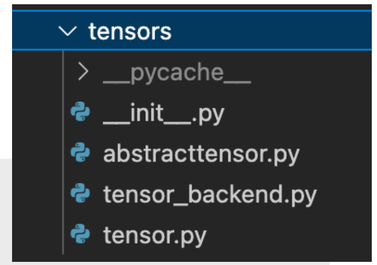
TTN ✓ MPS ✓

- 1-tensor update
- 2-tensor update
- Space link expansion - not tested properly

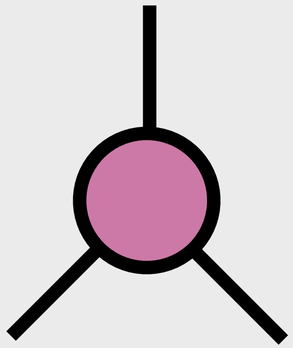
+ Exact diagonalization algorithms

Qtealeaves tensor class- QteaTensor

qtealeaves/tensors/tensor.py



What specifies a tensor in qtealeaves?



- number of links and their dimension
- data type (int, float, complex, etc.)
- device (cpu or gpu)
- elements of tensor

Numpy/Cupy tensor methods available, SVD, QR decomposition functions, …

QteaTensor inherits from _**AbstractQteaTensor** class defined in qtealeaves/tensors/abstracttensor.py

For quick course on QteaTensors, see the Jupyter notebook `general_materials/tensors_guide.ipynb` **in the GitHub repository**

#2 Code structure

Elementary building block:

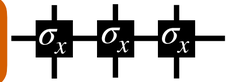
Tensor

$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

Operators

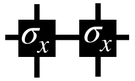
e.g. Pauli operators, bosonic operators

MPOs



Some observables

e.g. local observables, correlators

$< \sigma_x \sigma_x > =$ 

on every site

Hamiltonian modeling

e.g. local term, two-body term

Models

e.g. Bose-Hubbard, Quantum Ising,

Effective operators

MPOs of the TN state

TN Ansatzes

e.g. TTN, MPS

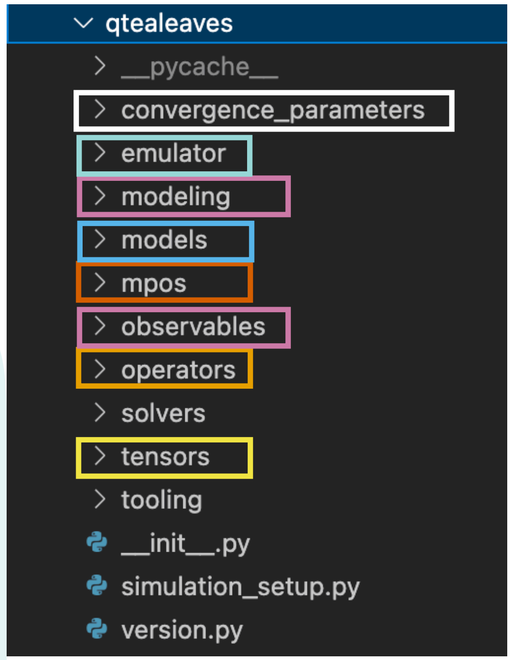
Algorithms

e.g. DMRG, TDVP

Measurements of observables

Energy, local observables (e.g. $< \sigma_x^i >$), correlations, bond entropy, …

Coordinated in qtealeaves/emulator



Important tool:
Convergence parameters

Maximal bond dimension
Cut ratio
Number of iterations
…

Simulation workflow

Simulation class: QuantumGreenTeaSimulation in qtealeaves/simulation_setup.py

User input script:

run simulation

simulation_setup.py

run python simulation

emulator/tn_simulation.py

1. Define an instance of **QuantumGreenTeaSimulation** which contains the input:

- System size
- Model (Hamiltonian)
- Tensor network ansatz
- Which algorithm
- Convergence parameters
- Tensor backend
- Initial state
- Observables to measure
- Input/output folders
- MPO representation
- …

Note: input can be parametrized

1. Creates input/output folders

2. Writes input files

3. Decides whether to:

- call fortran side simulation
- run simulation in python

1. Initializes appropriate TN state

2. Prepares Hamiltonian

3. Runs groundstate and/or time evolution:

Algorithm depends on TN ansatz - therefore corresponding functions inside tn simulator are called

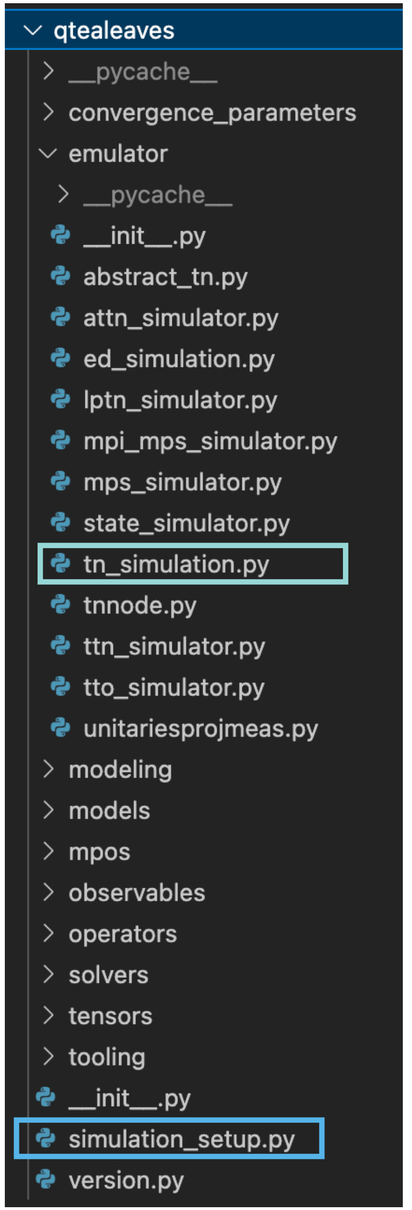
4. Measures the observables

5. Writes the results in output folders

read output

Interface for reading results from output files for every specified observable

Returns results in a dictionary



Example script

See `general_materials/groundstate_search.ipynb` **in GitHub repository**

Thank you for the attention!