Final Project: Time-Evolving Block Decimation

SCP8082721 - QUANTUM INFORMATION AND COMPUTING

2022-2023

David Lange

together with James Chryssanthacopoulos

April 8, 2023

Contents

Task

1. Study TEBD as applied to MPS ansatz;
2. Develop your numerical code for the TEBD-MPS and test it for the Ising chain with open boundaries.

Ising Model

- we are dealing with the ising model

$$\hat{H} = \lambda \sum_{i}^{N} \sigma_i^z + J \sum_{i}^{N-1} \sigma_i^x \sigma_{i+1}^x,$$

- with $\sigma^x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ and $\sigma^z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
- both parts can be written explicitly as Tensor products
  1. $\sigma_i^z = \mathbb{1}_1 \otimes \mathbb{1}_2 \otimes ... \otimes \sigma^z \otimes \mathbb{1}_{i+1} \otimes ... \otimes \mathbb{1}_N$
  2. $\sigma_i^x \sigma_{i+1}^x = \mathbb{1}_1 \otimes \mathbb{1}_2 \otimes ... \otimes \sigma^x \otimes \sigma^x \otimes \mathbb{1}_{i+2} \otimes ... \otimes \mathbb{1}_N$
- The general tensor product can be written as (A is a k×l matrix and B a m×n)

$$A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \dots & A_{1l}B \\ \vdots & & & \vdots \\ A_{k1}B & A_{k2}B & \dots & A_{kl}B \end{bmatrix}$$

Matrix product states and SVD

- A general matrix product state is given by

$$|\psi\rangle = A_1^{N+1} A_{N+1,2}^{N+2} \cdots A_{N+J-1,J}^{N+J} \cdots A_{2N-2,N-1}^{2N-1} A_{2N-1,N} \, |1, 2, ..., N\rangle \tag{1}$$

- SVD is given by

$$M = U \Sigma V^{\dagger} \tag{2}$$

where U is m$\times$m unitary matrix, V is n$\times$n unitary matrix and Σ is m$\times$n diagonal matrix. The singular values σ are stored in descending order.

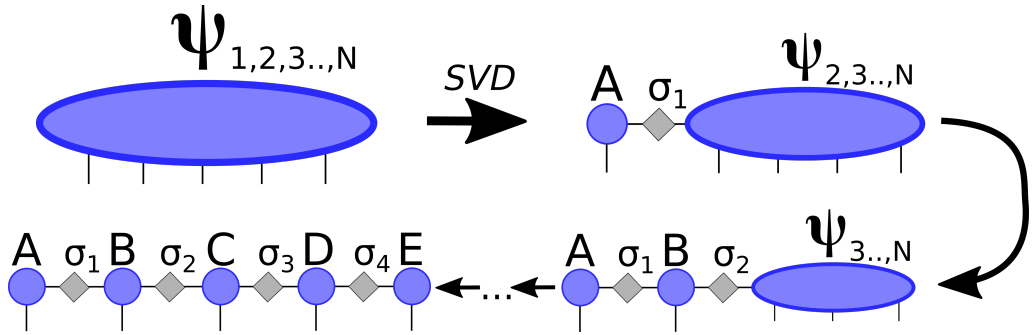


Figure: How to convert general wavefunction into MPS form using successive SVD operations.

- With MPS the computational cost of representing the state is changed from exponentially in N to $\mathcal{O}(Nd\chi^2)$, which can be controlled at the cost of truncation errors, by modifying bond dimension χ.

Suzuki-Trotter decomposition

- In order to time-evolve a quantum state, the time-evolution operator is employed

$$\hat{U} = e^{-i\hat{H}t} = \left(e^{-i\hat{H}\tau}\right)^n, \tag{3}$$

where the last equality, splitting into n infinitesimal segments, is convenient for computation purposes.

- We decompose the Hamiltonian into even and odd parts (even and odd terms commute among themselves since we only consider nearest-neighbor action, an even term and an odd term do not necessarily commute)

$$\hat{H} = \hat{H}_e + \hat{H}_o = \sum_{even i} h_{i,i+1} + \sum_{odd i} h_{i,i+1} \tag{4}$$

- according to Baker-Campbell-Hausdorff formula, one can find a solution to the equation

$$e^A e^B = e^Z, \tag{5}$$

with $Z = A + B + \frac{1}{2}[A, B] + \ldots$

- In the TEBD case, first order is given by (completely ignore the commutators)

$$e^{\tau(A+B)} = e^{\tau A} e^{\tau B} + \mathcal{O}(\tau^2), \tag{6}$$

since our operators A and B scale as τ, and thus the commutator will scale with τ^2 (and equality becomes exact in the limit $\tau \to 0$). This is called first-order Trotterization, because after $N = T/\tau$ time-steps, the error accumulated becomes of the order of τ. (Second-order Trotter has an error of the order of τ^2 etc.)

Time-evolving Block decimation

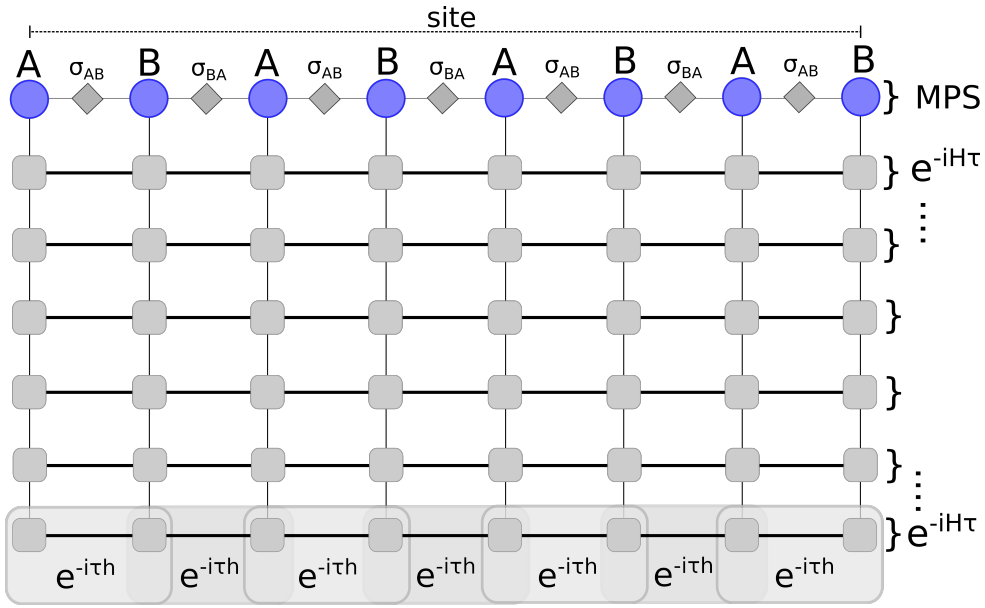


Figure: General TEBD scheme for N = 8 sites.

- The TEBD scheme using first order Trotter expansion is shown
- The two site Hamiltonian is given by
$h = J\left(\sigma^x \otimes \sigma^x\right) + \lambda\left(\sigma^z \otimes \mathbb{1} + \mathbb{1} \otimes \sigma^z\right)$

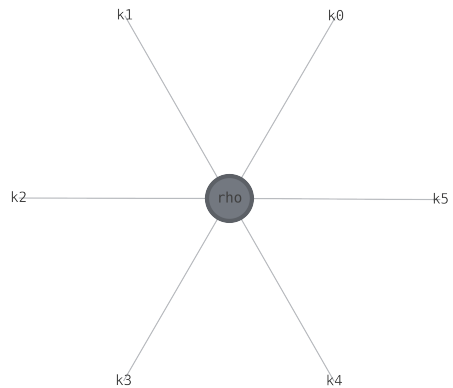
Code development.

QUIMB basics

(a) qtn.Tensor(data=state,inds=('k0', 'k1'), tags=['state'])

(b) network = tensor1 & tensor2

(c) MPS using QUIMB

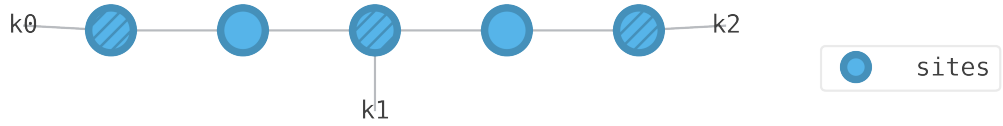


(d) Density matrix

(e) Energy of the Ising chain

$E = Tr(\rho H)$

Matrix product state class

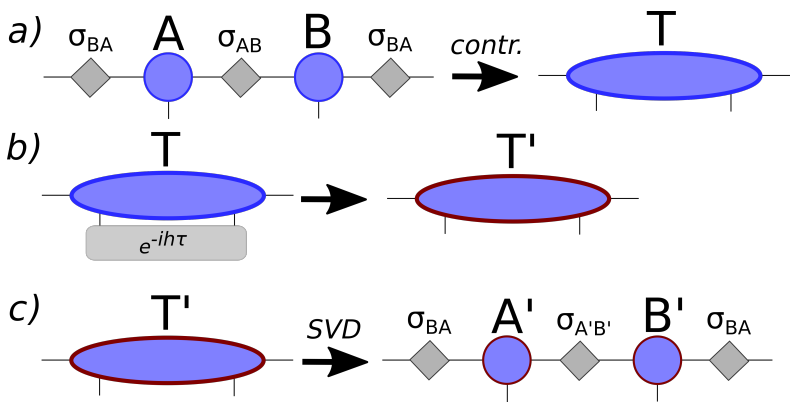


(f) 3 site MPS.

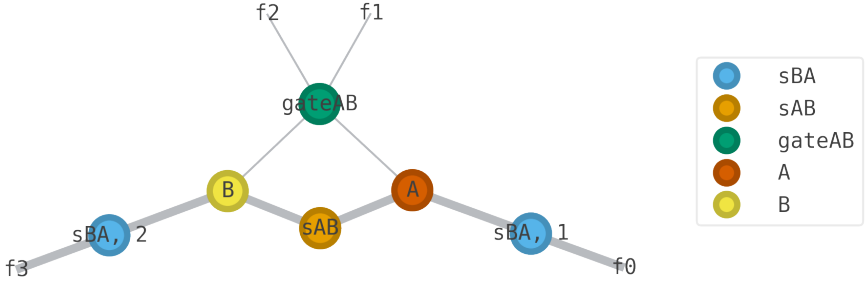
```python
class MatrixProductState:
    """Class representing a matrix product state with given number of states."""

    def __init__(self, d: int, N: int, bond_dim: int, states: Optional[List[np.array]] = None):
        """Initialize the MPS.

        Args:
            d: Dimension of each state
            N: Number of states
            bond_dim: Bond dimension between states
            states: Optional states to initialize with

        """
        self.d = d
        self.N = N
        self.bond_dim = bond_dim

        self.data = []
        self.singular_values = []

        if not states:
            states = []
            states.append(np.random.rand(d, bond_dim))
            for i in range(1, N - 1):
                states.append(np.random.rand(bond_dim, d, bond_dim))
            states.append(np.random.rand(bond_dim, d))

        # create left-most state
        self.data.append(qtn.Tensor(states[0], inds=("k0", "i0"), tags=["state 1"]))

        for i in range(1, N - 1):
            self.singular_values.append(np.eye(bond_dim, bond_dim))

            self.data.append(
                qtn.Tensor(self.singular_values[-1], inds=(f"i{2 * (i - 1)}", f"i{2 * i - 1}"), tags=[f"SV {i}"])
            )

            self.data.append(
                qtn.Tensor(states[i], inds=(f"i{2 * i - 1}", f"k{i}", f"i{2 * i}"), tags=[f"state {i + 1}"]))

        # create right-most state
        self.singular_values.append(np.eye(bond_dim, bond_dim))

        self.data.append(
            qtn.Tensor(self.singular_values[-1], inds=(f"i{2 * (N - 2)}", f"i{2 * N - 3}"), tags=[f"SV {N - 1}"])
        )

        self.data.append(qtn.Tensor(states[N - 1], inds=(f"i{2 * N - 3}", f"k{N - 1}"), tags=[f"state {N}"]))

        self.normalize()
```

(g) Developed code for MPS class.

Time evolution



(h) The general time-evolution scheme is shown, a) contract the two-site network, b) apply time-evoluton operator and c) use SVD (with possible truncation) to get back MPS form.



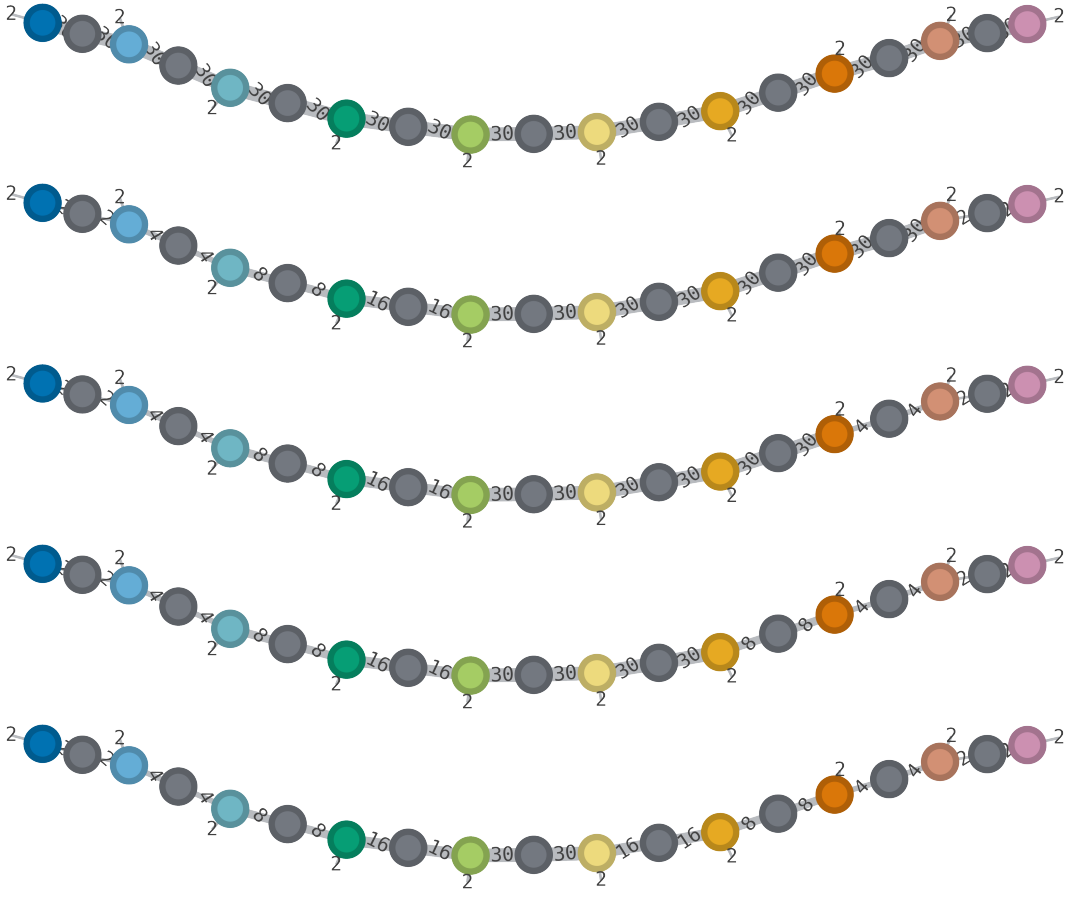
(i) The detailed gate operation on each two-sites in the program.

```python
def _apply_interior_gate(
        self, gate: np.array, left_site: qtn.Tensor, right_site: qtn.Tensor, left_bond: qtn.Tensor,
        central_bond: qtn.Tensor, right_bond: np.array, stol=1e-7
) -> np.array:
    """Apply gate to two interior sites.

    Args:
        left_site: Left-most site
        right_site: Site adjacent to left-most site
        left_bond: Left to the left of left site
        central_bond: Bond between two sites
        right_bond: Bond to the right of right site
        gate: Gate representing time evolution
        stol: Threshold for singular values

    """
    # ensure singular values are above tolerance threshold
    left_bond_data = np.diagonal(left_bond.data)
    left_bond_data = np.diag(left_bond_data * (left_bond_data > stol) + stol * (left_bond_data < stol))

    right_bond_data = np.diagonal(right_bond.data)
    right_bond_data = np.diag(right_bond_data * (right_bond_data > stol) + stol * (right_bond_data < stol))

    left_bond_T = qtn.Tensor(left_bond_data, inds=('f0', 'k1'), tags=['left bond'])
    left_site_T = qtn.Tensor(left_site.data, inds=('k1', 'k2', 'k3'), tags=['left site'])
    central_bond_T = qtn.Tensor(central_bond.data, inds=('k3', 'k4'), tags=['central bond'])
    right_site_T = qtn.Tensor(right_site.data, inds=('k4', 'k5', 'k6'), tags=['right site'])
    right_bond_T = qtn.Tensor(right_bond_data, inds=('k6', 'f3'), tags=['right bond'])
    gate_T = qtn.Tensor(gate, inds=('f1', 'f2', 'k2', 'k5'), tags=['gate'])

    # contract with gate
    TN = left_bond_T & gate_T & left_site_T & central_bond_T & right_site_T & right_bond_T
    TNc = TN ^ ...

    # perform SVD
    nshape = [self.d * left_site.data.shape[0], self.d * right_site.data.shape[2]]
    utemp, stemp, vhtemp = LA.svd(TNc.data.reshape(nshape), full_matrices=False)

    # truncate to reduced dimension
    chitemp = min(self.bond_dim, len(stemp))
    utemp = utemp[:, range(chitemp)].reshape(left_site.data.shape[0], self.d * chitemp)
    vhtemp = vhtemp[range(chitemp), :].reshape(chitemp * self.d, right_site.data.shape[2])

    # remove environment weights to form new MPS tensors A and B
    left_site.modify(data=(LA.inv(left_bond_data) @ utemp).reshape(left_site.data.shape[0], self.d, chitemp))
    right_site.modify(data=(vhtemp @ LA.inv(right_bond_data)).reshape(chitemp, self.d, right_site.data.shape[2]))
    central_bond.modify(data=np.diag(stemp[range(chitemp)] / LA.norm(stemp[range(chitemp)])))
```

```python
class LocalHamiltonian:
    """Base class for representing local Hamiltonians."""

    def __init__(self, d: int, N: int):
        """Initialize local Hamiltonian.

        Args:
            d: Number of dimensions
            N: Number of sites

        """
        self.d = d
        self.N = N
        self.hamiltonians = np.zeros((N - 1, d ** 2, d ** 2))


class LocalIsingHamiltonian(LocalHamiltonian):
    """Create local Hamiltonians for Ising chain."""

    def __init__(self, N: int, J: Optional[float] = 1.0, lmda: Optional[float] = 0.0):
        """Create Hamiltonians for given number of sites.

        Args:
            N: Number of sites
            J: Coupling between neighboring spins
            lmda: Coupling to external magnetic field

        """
        super(LocalIsingHamiltonian, self).__init__(2, N)

        hamiltonian_two_site = (
            J * np.kron(qu.pauli("X"), qu.pauli("X")) +
            lmda * np.kron(qu.pauli("Z"), np.eye(2)) +
            lmda * np.kron(np.eye(2), qu.pauli("Z"))
        )

        self.hamiltonians = np.repeat(hamiltonian_two_site[None, :], N - 1, axis=0)
```

Figure: Upper section is the interior gate algorithm, lower section is the two-sited hamiltonian.

Running the program



```
run_tebd(
model = 'ising',
model_params = {'J': 1.0, 'lmda': 0.0},
N = 7,
bond_dim = 2,
tau = 0.1,
num_iter = 700,
mid_steps = 2,
observables = ["energy"],
print_to_stdout = False,
evol_type = "imag",
st_order = "ST4",
initial_state = '1111111'
)
```

Figure: Function to run the developed TEBD algorithm with main parameters. (other observables may be e.g. entropy, magnetization or wave-function)

Figure: Chain of length N = 10 and bond dimension $\chi = 30$, the initial and the first four iterations of the time-evolution are shown and their impact on the bond dimensions between each site.

Results.

Ground-state energy of the model

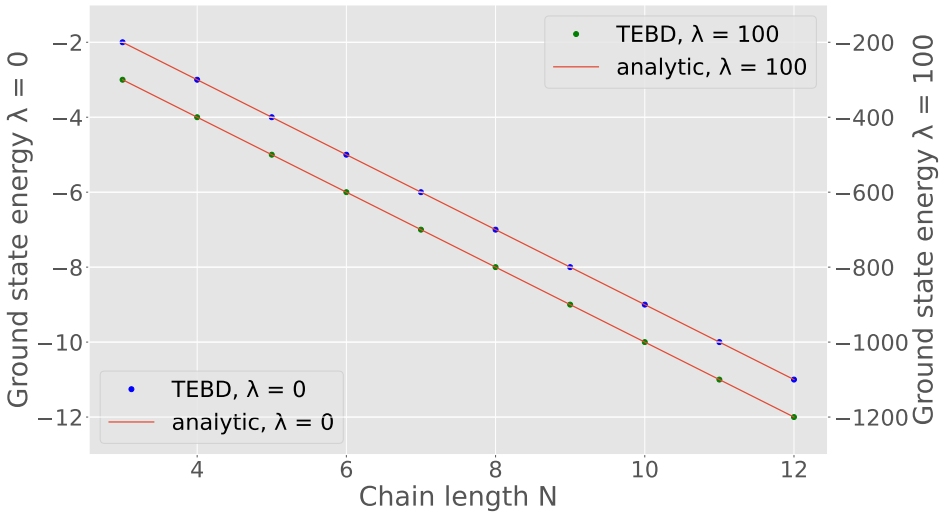


Figure: Ground state energy of the ising Hamiltonian in terms of chain length N for parameters $\tau = 0.1$, 1000 iterations, $\lambda = 0, 100$, $J = 1$ and $\chi = 3$.

- The ground state energy with $\lambda = 0$ is when all spins are aligned so $E/N = -(N-1)/N$
- when $\lambda > J$, energy of lowest state is $E = -\lambda N$

Time-step sizes, Trotter order and convergence

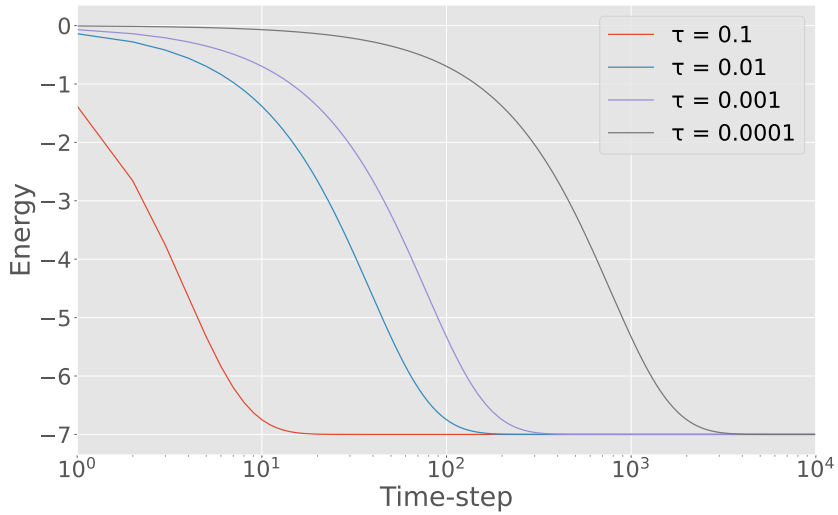


Figure: Ground state energy estimate time evolution of the ising Hamiltonian with different time-step sizes τ. Parameters are $N = 8$, $J = 1.0$, $\lambda = 0.0$ and $\chi = 5$. Note, that for the purpose of choosing τ, one has to compare both convergence speeds (computation times) versus accuracy. Meaning, while lower τ leads to slower convergence in the same number of time-steps, the accuracy after running the algorithm for the same amount of **time**(τ times number of iterations) is enhanced.

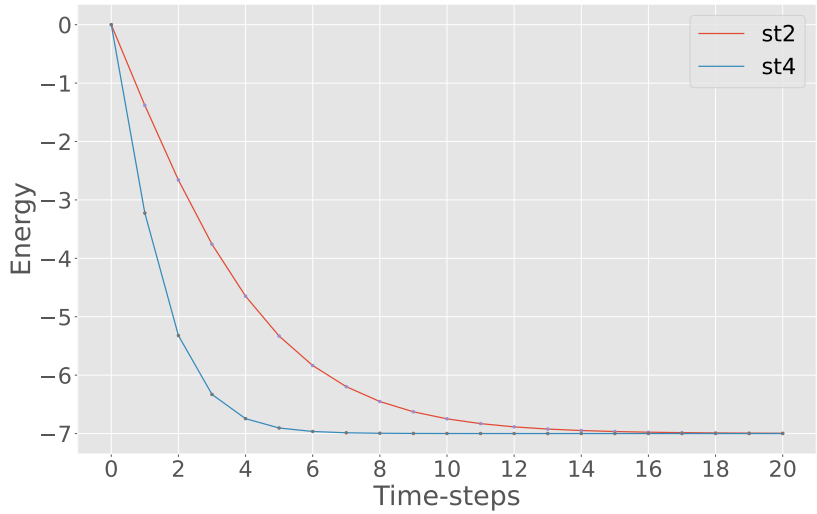


Figure: Comparison in convergence speed st2 versus st4. Parameters are $N = 8$, $J = 1.0$, $\lambda = 0.0$, $\chi = 5$ and $\tau = 0.5$.

Effect of bond dimension for small and large λ

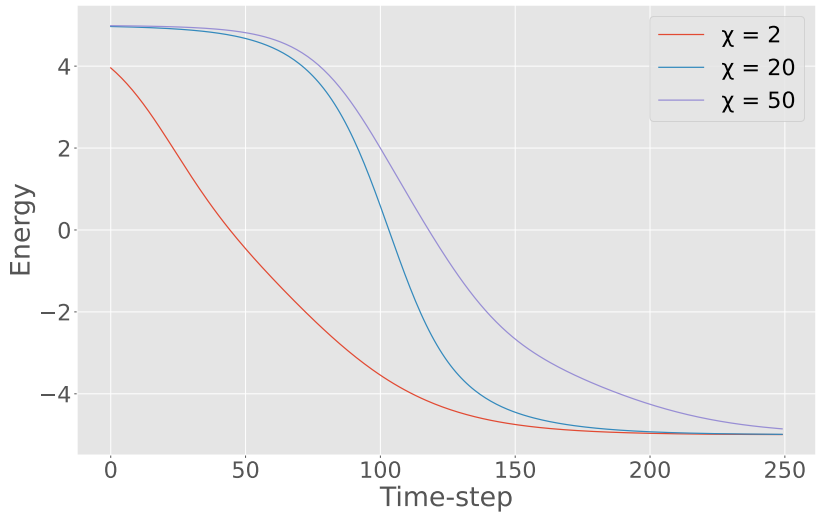


Figure: Relevant parameters are $\tau = 0.01$, $\lambda = 0.001$. Higher bond dimension χ means less entanglement is discarded at each iteration. Since the ground-state is two fold degenerate, this leads to slower convergence.

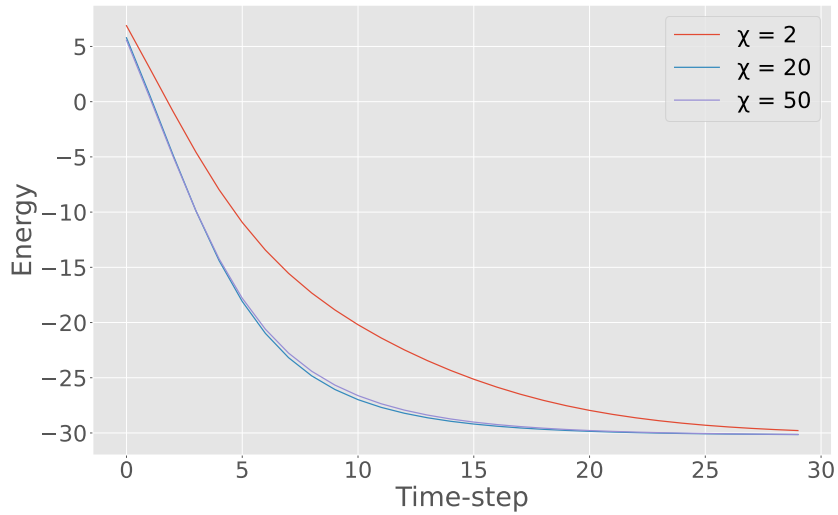


Figure: $\tau = 0.01$, $\lambda = 5$. After surpassing the transition point, the lattice is ferromagnetic, meaning there is a long-range order and thus a higher bond dimension leads to faster convergence.

Magnetization

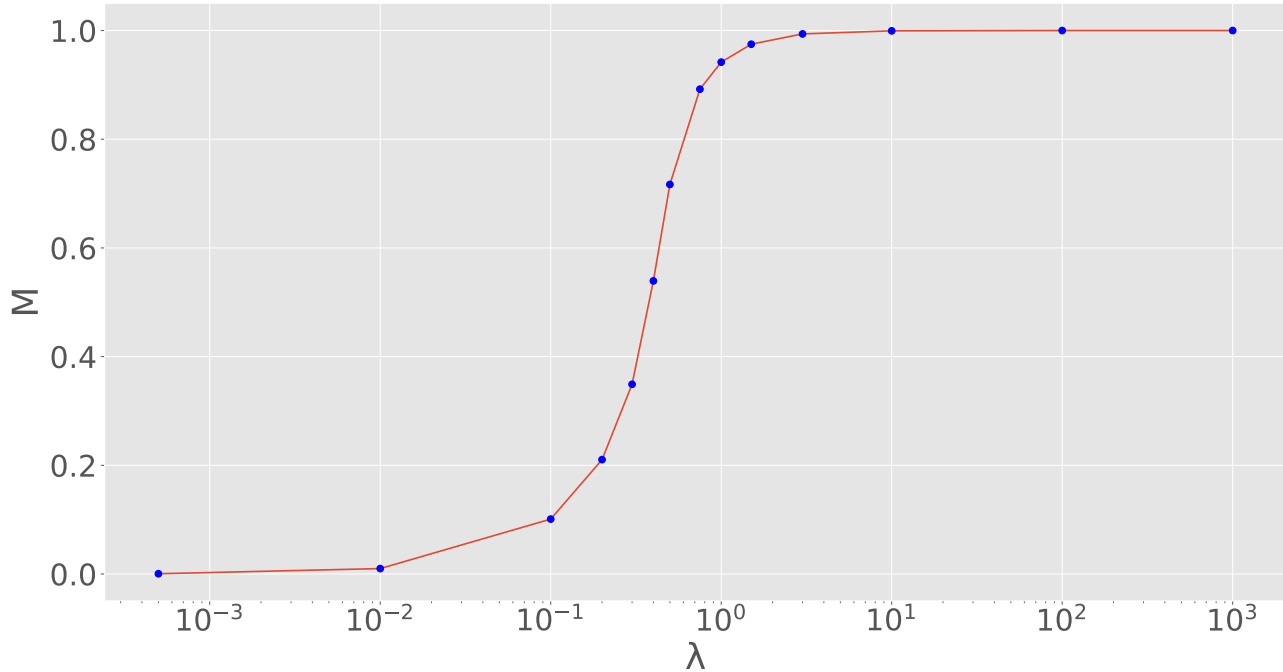
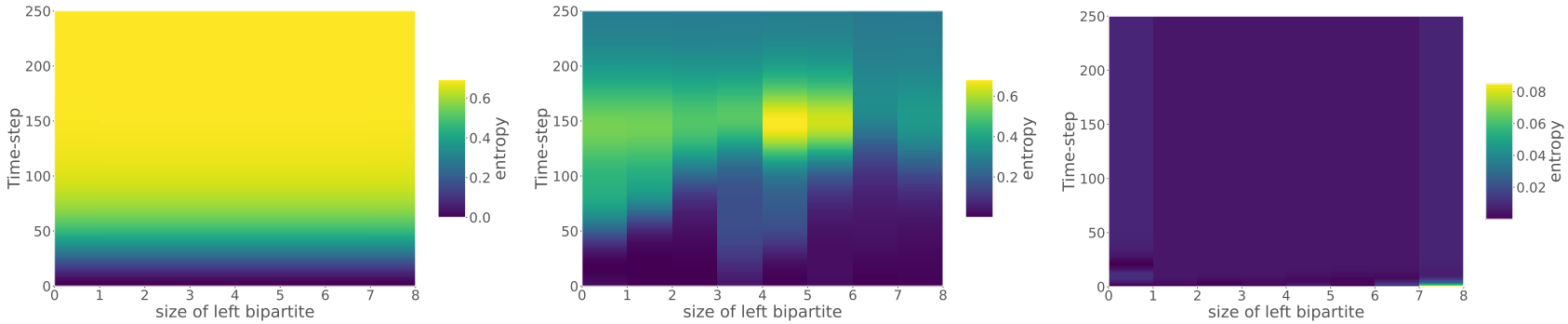


Figure: Calculating the average magnetization of the ground state $\hat{M} = 1/N \sum_{i=1}^{N} \sigma_i^z \rightarrow M = \langle \psi_{gs} | \hat{M} | \psi_{gs} \rangle$ for $N = 7$ again. We can see that the phase transition occurs at around $\lambda = 1$, as expected. In the interaction dominated regime ($\lambda < 1$) the ground state is two fold degenerate (all spins align with the x-axis and z-component of spins is randomly aligned), while in the field dominated regime ($\lambda > 1$) the ground state is not degenerate anymore since the spins have a preferred direction. So we have a quantum phase transition driven by quantum fluctuations, in this case the presence of an external magnetic field.

Entropy considerations and entanglement



(a) Initial state all spins up and small λ. (b) Initial state random, small λ. (c) Initial state random, large λ.

- Von Neumann entropy can be calculated by partitioning the system into two subsystems A and B

$$S = -\operatorname{Tr}\left(\rho_A \log(\rho_A)\right) \tag{7}$$

$$= -\sum_i \lambda_i \log(\lambda_i), \tag{8}$$

  with ρ_A the reduced density matrix of the subsystem and λ_i the eigenvalues.
- for given initial state, the entropy grows smoothly, while for a random state it grows less smooth and converges to uniform value
- For small λ, the entropy gets very high because of the degenerate ground-state, for high λ the entropy stays low because of long-range order.

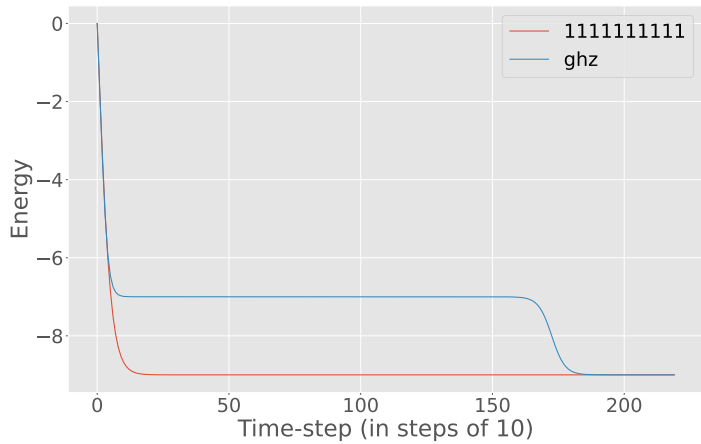
Conclusions and outlook

- good agreement between numerical results and exact solution, combined with the manageable computation times
  $\rightarrow$ TEBD is a great algorithm to compute the ground state energy of Hamiltonians with local character
- Next steps: correct canonicalization (Truncation errors are minimized in this form)
- also other Hamiltonians can be studied e.g. Heisenberg-model

- study more entangled states
- the GHZ state is given by

$$|GHZ\rangle = \frac{1}{\sqrt{2}} \left(|00\ldots0\rangle + |11\ldots1\rangle\right) \tag{9}$$

- and can be represented by a MPS with bond dimension $\chi = 2$.



(d) Initial state unentangled versus highly entangled convergence comparison.

Thank you Fourier attention.

Selection of relevant sources:

1. The relevant package used for Tensor calculations was QUIMB
   Johnnie Gray, QUIMB, (2015-2023), GitHub repository https://github.com/jcmgray/quimb
2. Mostly used was
   Glen Evenbly, *Tensors.net*, URL: https://www.tensors.net/, accessed 07.04.2023.
3. For a lot of the figures I was inspired by
   Frank Pollmann, *Efficient Numerical Simulations Using Matrix-Product States*, LN Max-Planck-Institut Dresden, October 2016.
4. For the theory I used
   - 4.1 S. Montangero, *Introduction to Tensor Network Methods*, Springer, 2018.
   - 4.2 Ulrich Schollwöck et al., *Emergent Phenomena in Correlated Matter Ch. 16/17*, Lecture Notes of the Autumn School Correlated Electrons Jülich, 2013.