

# Coding Session: Numerical Simulations of Quantum Many-Body Systems

A ~~Brief~~ Introduction to MPS and DMRG  
full-speed

Johannes Hauschild

Technical University Munich

05.10.2022

# Outline

- |                             |          |
|-----------------------------|----------|
| ➊ Introduction              | (10 min) |
| ➋ Exercise 1: Tensor basics | (30 min) |
| ➌ Explanation MPS           | (20 min) |
| ➍ Exercise 2: MPS basics    | (10 min) |
| ➎ Explanation DMRG          | (20 min) |
| ➏ Exercise 3: DMRG          | (20 min) |
| ➐ Wrapup                    | (10 min) |

# Simulation of quantum many-body systems

## Goal

numerically solve

$$\hat{H} |\psi_0\rangle = E_0 |\psi_0\rangle$$

$$i\hbar\partial_t |\psi(t)\rangle = \hat{H} |\psi(t)\rangle$$

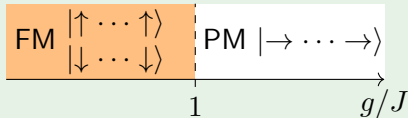
on classical computers

$$|\psi\rangle \in \mathcal{H} = \bigotimes_{i=0}^N \mathbb{C}^d$$

$\Rightarrow$  challenge: exponentially big  $\dim \mathcal{H} = d^N$

## Example (Transverse Field Ising Model)

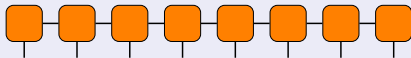
$$\hat{H} = -J \sum_{i=1}^{N-1} \hat{Z}_i \hat{Z}_{i+1} - g \sum_{i=1}^N \hat{X}_i$$
$$\hat{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \hat{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$



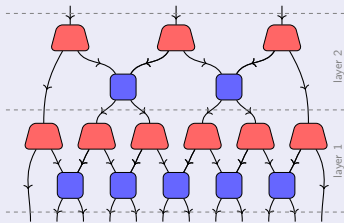
# Tensor Networks

variational ansatz  $|\psi\rangle = \sum_{j_1 \dots j_N} \boxed{\psi}_{j_1 j_2 \dots j_N} |j_1, \dots, j_N\rangle$  with  $\psi$  tensor network

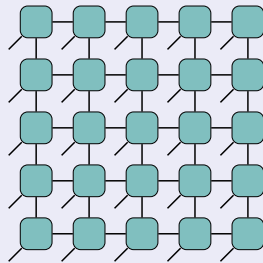
MPS (here)



MERA



PEPS



and infinite versions  
and more ...

# Graphical Notation

$$\left. \begin{array}{l} v_a = a - \boxed{v} \\ M_{ab} = a - \boxed{M} - b \end{array} \right\} \Rightarrow M \cdot v = \sum_b M_{ab} v_b = a - \boxed{M} - b \boxed{v}$$

## Example (overlap)

$$|\psi\rangle = \sum_{j_1 \dots j_N} \boxed{\psi}_{j_1 j_2 \dots j_N} |j_1, \dots, j_N\rangle$$

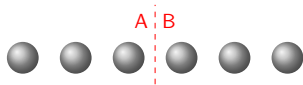
$$\Rightarrow \langle \phi | \psi \rangle = \begin{array}{c} \boxed{\psi} \\ \hline \boxed{\bar{\phi}} \end{array}$$

## Example (reshape)

$$h := \hat{Z}_1 \hat{Z}_2 = \boxed{h} \boxed{Z}_{j_1} \boxed{Z}_{j_2}$$

$$\stackrel{\text{reshape}}{\simeq} \boxed{h}_{j_1 j_2 \dots k = d j_1 + j_2}$$

# Schmidt decomposition



$$|\psi\rangle = \sum_{k_A, k_B} \boxed{\psi}_{k_A, k_B} |k_A\rangle |k_B\rangle$$

$$\stackrel{\text{SVD}}{=} \sum_{k_A, k_B} \boxed{A}_{k_A} \overset{\alpha}{\Lambda} \overset{\alpha'}{\boxed{B}}_{k_B} |k_A\rangle |k_B\rangle$$

$$= \boxed{|\alpha\rangle_A}_{|j_1\rangle|j_2\rangle|j_3\rangle} \overset{\alpha}{\Lambda} \overset{\alpha'}{\boxed{|\alpha'\rangle_B}}_{|j_4\rangle|j_5\rangle|j_6\rangle}$$

with diagonal  $\Lambda_{\alpha, \alpha'} = \underbrace{\lambda_\alpha}_{\geq 0} \delta_{\alpha, \alpha'}$

# SVD Singular Value Decomposition

matrix decomposition

$$M = USV^\dagger$$

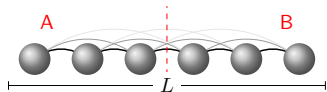
with diagonal  $S \geq 0$  and isometries

$$U^\dagger U = \mathbb{1} \quad V^\dagger V = \mathbb{1}$$

$${}_A \langle \bar{\alpha} | \alpha \rangle_A = \begin{array}{c} \boxed{A}^{\alpha} \\ \hline \boxed{\bar{A}}_{\bar{\alpha}} \end{array} = \begin{pmatrix} \alpha \\ \bar{\alpha} \end{pmatrix} = \delta_{\alpha, \bar{\alpha}}$$

$$\delta_{\alpha', \bar{\alpha}'} = \begin{pmatrix} \alpha \\ \bar{\alpha} \end{pmatrix} = \begin{array}{c} \boxed{B}^{\alpha} \\ \hline \boxed{\bar{B}}_{\bar{\alpha}} \end{array} = {}_B \langle \bar{\alpha}' | \alpha' \rangle_B$$

# Area Law for Entanglement Entropy



## Entanglement entropy

$$S = -\text{Tr} \varrho_A \log(\varrho_A) = -\sum_{\alpha} \lambda_{\alpha}^2 \log \lambda_{\alpha}^2$$

$$\text{where } \varrho_A = \text{Tr}_B |\psi\rangle \langle\psi| = A \Lambda^2 A^{\dagger}$$

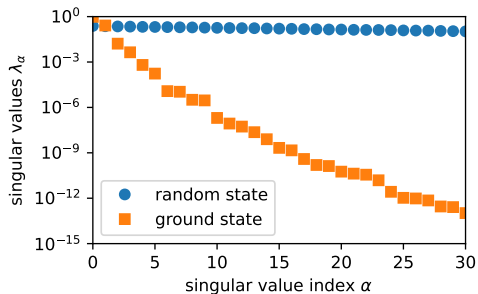
## Area law

for ground states of gapped Hamiltonians

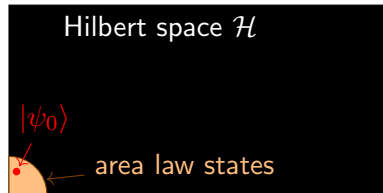
$$S \propto \text{area of cut}_{\text{in 1D}} = \mathcal{O}(L^0)$$



Hastings (J.Stat.Mech. 2007)



⇒ can truncate: only keep  $\chi$  largest  $\lambda_{\alpha}$



# Exercise 1: Tensor Basics

<https://quanthub.pks.mpg.de>, Login and Password same as for WiFi  
also [https://github.com/mgbukov/quant22\\_notebooks](https://github.com/mgbukov/quant22_notebooks)

File Edit View Run Kernel Git Tabs Settings Help

Filter files by name

/ tensor\_networks /

| Name                        | Last Modified |
|-----------------------------|---------------|
| toycode                     | 3 minutes ago |
| exercise_1_basics.ipynb     | seconds ago   |
| exercise_2_mps.ipynb        | 3 minutes ago |
| exercise_3_dmrg.ipynb       | 3 minutes ago |
| solution_1_basics.ipynb     | 3 minutes ago |
| solution_2_mps.ipynb        | 3 minutes ago |
| solution_3_dmrg.ipynb       | 3 minutes ago |
| toycode_example_calls.py... | seconds ago   |

## Tensor basics

Here, we cover some very basic concepts of tensor networks:

- how to represent tensors numerically, and
- what the basic operations are that we need for tensors

For these (toy code) exercises will use the python `numpy` library for simplicity. For production MPS code, you should eventually switch to a more generic library, e.g., `TeNPy`, which provide a more powerful class for representing tensors - this allows to make use of block-sparseness implied by symmetries and helps with index bookkeeping.

```
[1]: import numpy as np
```

For later use, let us also import matplotlib and set some cosmetic display options.

```
[2]: import matplotlib.pyplot as plt
np.set_printoptions(precision=5, suppress=True, linewidth=100, threshold=50)
plt.rcParams['figure.dpi'] = 150
```

## Initializing tensors in numpy

From a mathematical point of view, tensors are multi-linear function between vector spaces. From a numerical point of view, however, tensors are just collections of numbers arranged in a multi-dimensional grid, with element-wise addition and tensor-dot contractions over one or multiple legs.

In this set of toycodes, we represent tensors directly as numpy arrays. The following examples demonstrate different ways to initialize tensors, here for the usual  $2 \times 2$  Pauli matrices:

```
[3]: # np.eye = identity matrix of given dimension 2
Id = np.eye(2)
Id
```

```
[4]: array([[1., 0.],
          [0., 1.]])
```

Simple 1 1 main No Kernel | Unknown Mode: Command Ln 1, Col 1 exercise\_1\_basics.ipynb



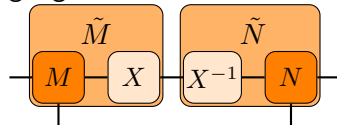
# Matrix Product States (MPS)

## Definition (MPS)

Variational Ansatz

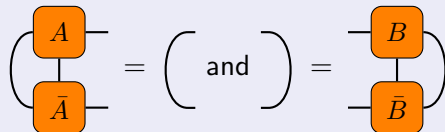
$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{---} \underset{j_1}{\square} \text{---} \square \text{---} \square \text{---} \square \text{---} \underset{j_N}{\square} \text{---} |j_1, \dots, j_N\rangle$$

gauge freedom on contracted legs:

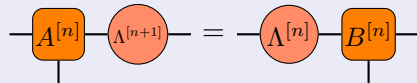


## Canonical Form

ONB of Schmidt states



in canonical form with



# Matrix Product States (MPS)

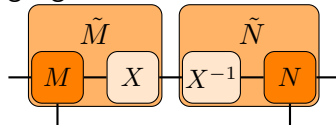
## Definition (MPS)

### Variational Ansatz

$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$

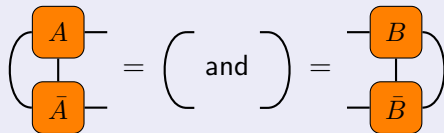
free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$

gauge freedom on contracted legs:

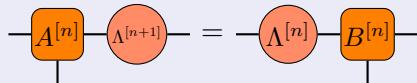


## Canonical Form

ONB of Schmidt states



in canonical form with



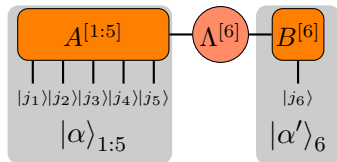
# Matrix Product States (MPS)

## Definition (MPS)

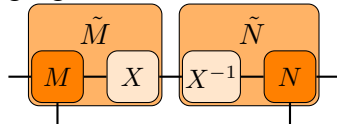
### Variational Ansatz

$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$

free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$

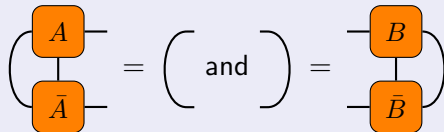


gauge freedom on contracted legs:

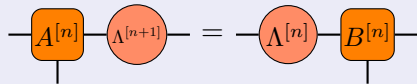


## Canonical Form

ONB of Schmidt states



in canonical form with



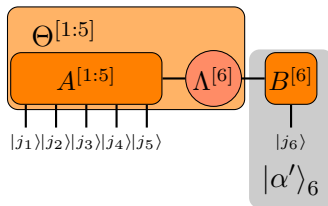
# Matrix Product States (MPS)

## Definition (MPS)

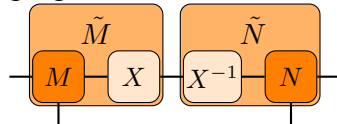
### Variational Ansatz

$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$

free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$

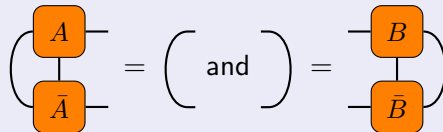


gauge freedom on contracted legs:

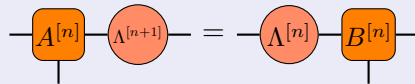


## Canonical Form

ONB of Schmidt states



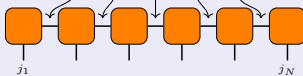
in canonical form with



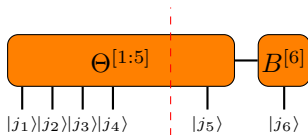
# Matrix Product States (MPS)

## Definition (MPS)

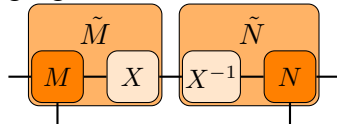
### Variational Ansatz

$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$


free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$

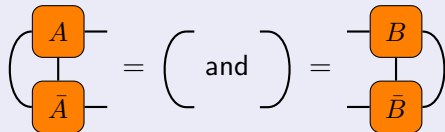


gauge freedom on contracted legs:

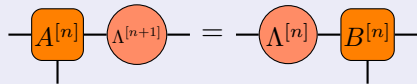


## Canonical Form

ONB of Schmidt states



in canonical form with



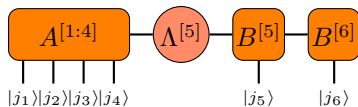
# Matrix Product States (MPS)

## Definition (MPS)

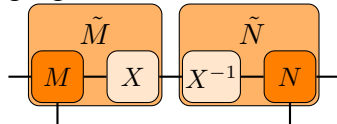
### Variational Ansatz

$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$

free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$

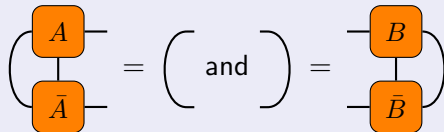


gauge freedom on contracted legs:

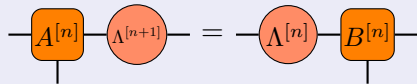


## Canonical Form

ONB of Schmidt states



in canonical form with



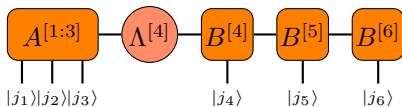
# Matrix Product States (MPS)

## Definition (MPS)

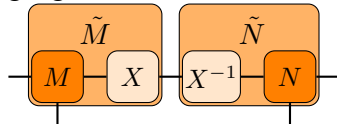
### Variational Ansatz

$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$

free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$

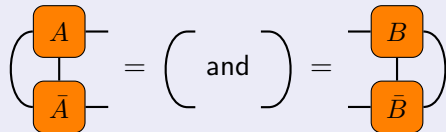


gauge freedom on contracted legs:

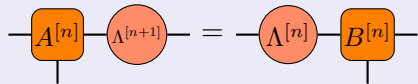


## Canonical Form

ONB of Schmidt states



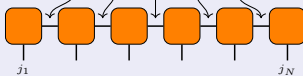
in canonical form with



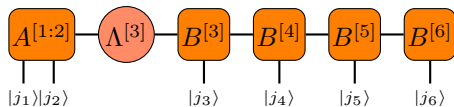
# Matrix Product States (MPS)

## Definition (MPS)

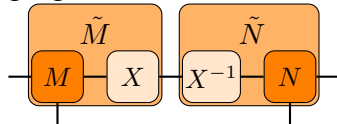
### Variational Ansatz

$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$


free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$

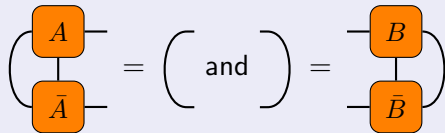


gauge freedom on contracted legs:

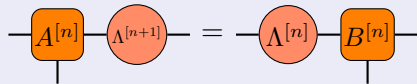


## Canonical Form

ONB of Schmidt states



in canonical form with

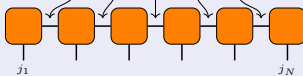




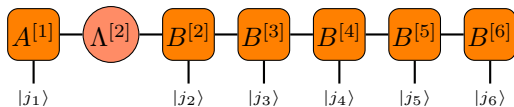
# Matrix Product States (MPS)

## Definition (MPS)

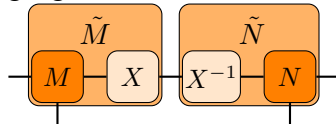
### Variational Ansatz

$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$


free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$

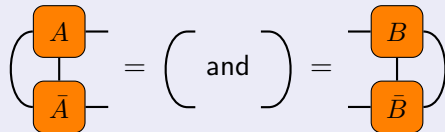


gauge freedom on contracted legs:

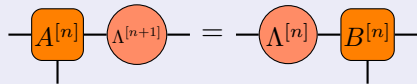


## Canonical Form

ONB of Schmidt states



in canonical form with



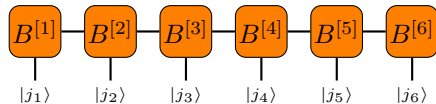
# Matrix Product States (MPS)

## Definition (MPS)

### Variational Ansatz

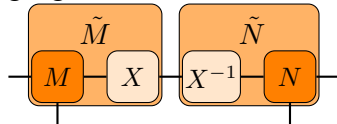
$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$

free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$



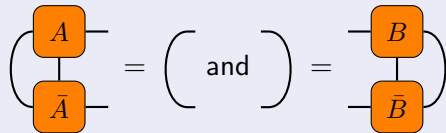
→ in *right* canonical form

gauge freedom on contracted legs:

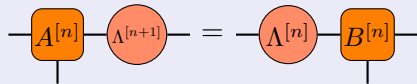


## Canonical Form

ONB of Schmidt states



in canonical form with



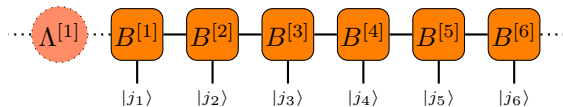
# Matrix Product States (MPS)

## Definition (MPS)

### Variational Ansatz

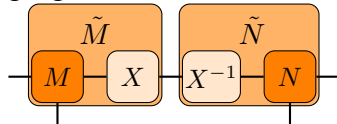
$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$

free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$



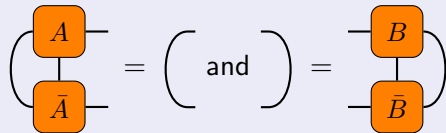
→ in *right* canonical form

gauge freedom on contracted legs:

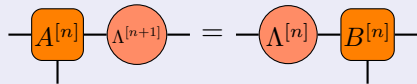


## Canonical Form

ONB of Schmidt states



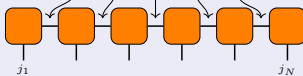
in canonical form with



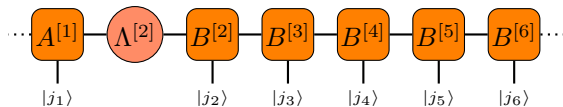
# Matrix Product States (MPS)

## Definition (MPS)

### Variational Ansatz

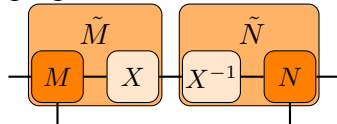
$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$


free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$



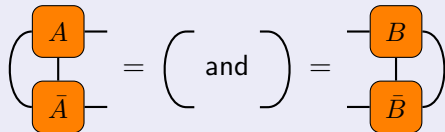
→ in *mixed* canonical form

gauge freedom on contracted legs:

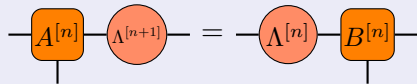


## Canonical Form

ONB of Schmidt states



in canonical form with



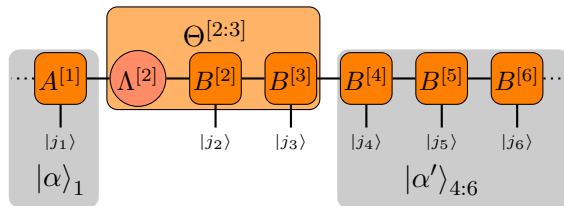
# Matrix Product States (MPS)

## Definition (MPS)

### Variational Ansatz

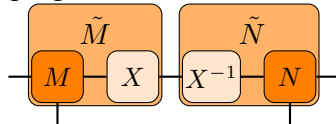
$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$

free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$



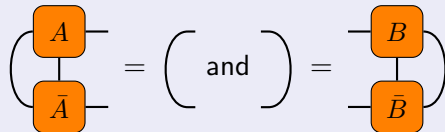
→ in *mixed* canonical form

gauge freedom on contracted legs:

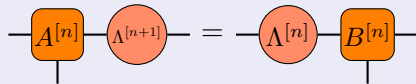


## Canonical Form

ONB of Schmidt states



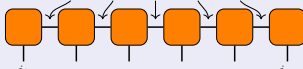
in canonical form with



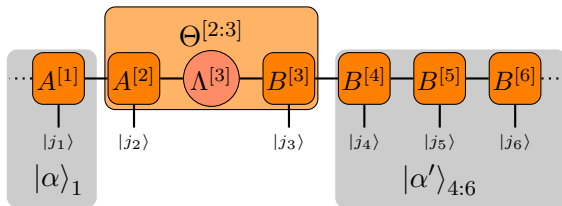
# Matrix Product States (MPS)

## Definition (MPS)

### Variational Ansatz

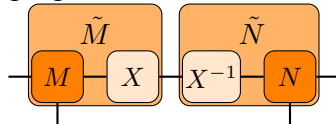
$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$


free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$



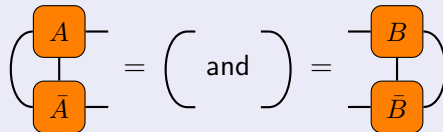
→ in *mixed* canonical form

gauge freedom on contracted legs:

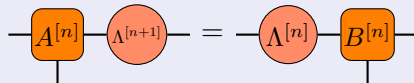


## Canonical Form

ONB of Schmidt states



in canonical form with



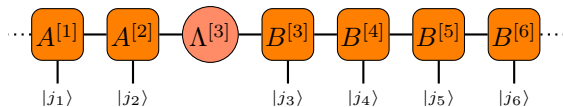
# Matrix Product States (MPS)

## Definition (MPS)

### Variational Ansatz

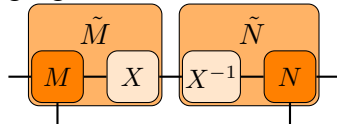
$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$

free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$



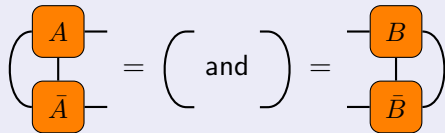
→ in *mixed* canonical form

gauge freedom on contracted legs:

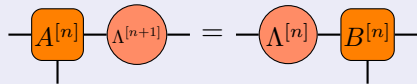


## Canonical Form

ONB of Schmidt states



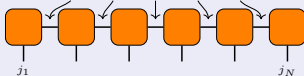
in canonical form with



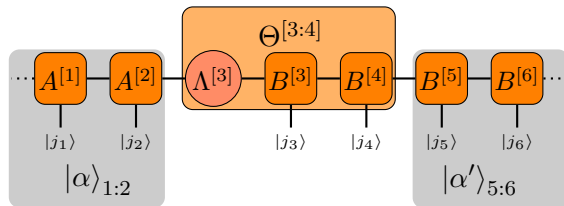
# Matrix Product States (MPS)

## Definition (MPS)

### Variational Ansatz

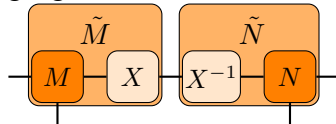
$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$


free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$



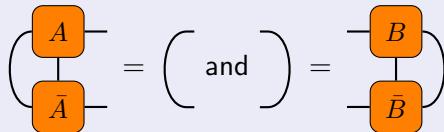
→ in *mixed* canonical form

gauge freedom on contracted legs:

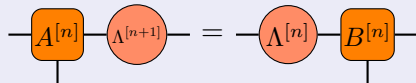


## Canonical Form

ONB of Schmidt states



in canonical form with





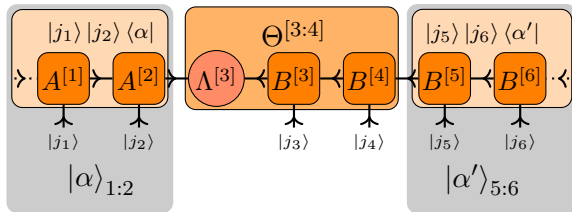
# Matrix Product States (MPS)

## Definition (MPS)

### Variational Ansatz

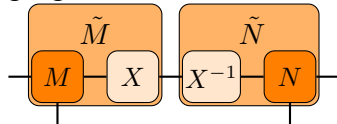
$$|\psi\rangle = \sum_{j_1 \dots j_N} \text{max. bond dimension } \chi \approx 1000 \quad |j_1, \dots, j_N\rangle$$

free parameters  $\mathcal{O}(d^N) \rightarrow \mathcal{O}(N\chi^2)$



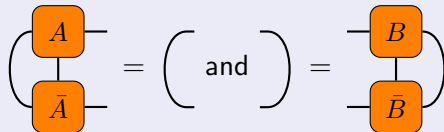
→ in *mixed* canonical form

gauge freedom on contracted legs:

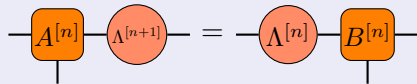


## Canonical Form

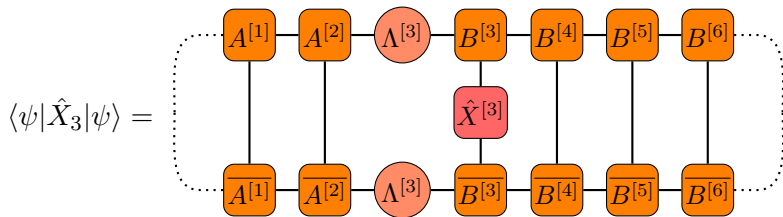
ONB of Schmidt states



in canonical form with



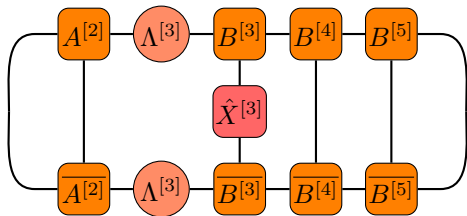
## MPS Expectation values



$\Rightarrow$  always contract *locally*, never top to bottom!

## MPS Expectation values

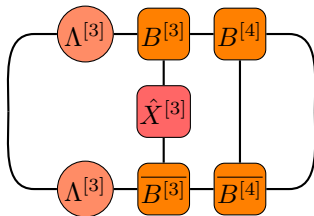
$$\langle \psi | \hat{X}_3 | \psi \rangle =$$



$\Rightarrow$  always contract *locally*, never top to bottom!

## MPS Expectation values

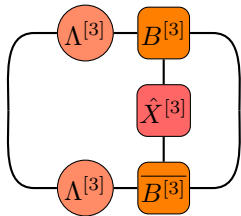
$$\langle \psi | \hat{X}_3 | \psi \rangle =$$



$\Rightarrow$  always contract *locally*, never top to bottom!

## MPS Expectation values

$$\langle \psi | \hat{X}_3 | \psi \rangle =$$

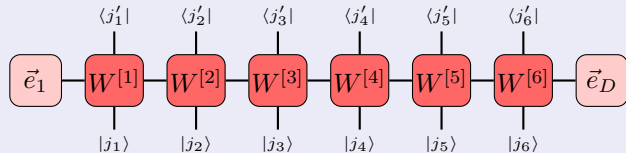


$\Rightarrow$  always contract *locally*, never top to bottom!

# Matrix Product Operator (MPO)

## Definition (MPO)

write Hamiltonian  $\hat{H}$  as

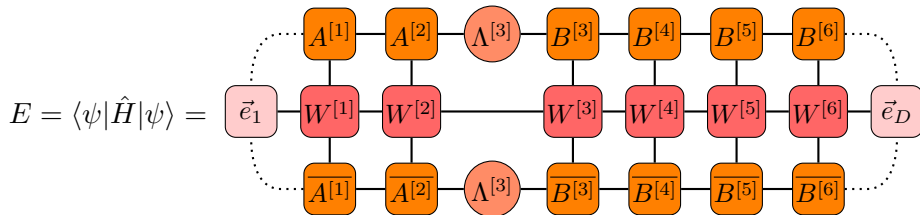


bond dimension  $D$  depends on  $\hat{H}$

## Example (Transverse Field Ising)

$$\hat{H} = -J \sum_i \hat{Z}_i \hat{Z}_{i+1} - g \sum_i \hat{X}_i$$

$$W_{\alpha, \alpha'}^{[n]} = \begin{pmatrix} \mathbb{1} & \hat{Z}_n & -g \hat{X}_n \\ & & J \hat{Z}_n \\ & & \mathbb{1} \end{pmatrix}$$



## Exercise 2: MPS and Model

<https://quanthub.pks.mpg.de>, Login and Passwort same as for WiFi  
also [https://github.com/mgbukov/quant22\\_notebooks](https://github.com/mgbukov/quant22_notebooks)

Launcher

exercise\_1\_basics.ipynb exercise\_2\_mps.ipynb

Markdown Python 3 (ipykernel)

### MPS and model basics

In this notebook, we introduce the `SimpleMPS` class from `toycodes/a_mps.py` and the model class from `toycodes/b_model.py`.

```
[1]: # standard imports and cosmetics
import numpy as np
import matplotlib.pyplot as plt

np.set_printoptions(precision=5, suppress=True, linewidth=100, threshold=50)
plt.rcParams['figure.dpi'] = 150
```

### SimpleMPS class from `toycodes/a_mps.py`

The file `toycodes/a_mps.py` defines a `SimpleMPS` class, that provides methods for expectation values and the entanglement entropy.

You can initialize an initial product state MPS with the provided functions

- `init_FM_MPS` to initialize the state  $|↑↑ \dots ↑↑\rangle$ , and
- `init_Neel_MPS` to initialize the Neel state  $|↑↓ \dots ↓↑\rangle$

```
[2]: import toycodes.a_mps
from toycodes.a_mps import SimpleMPS, init_FM_MPS, init_Neel_MPS

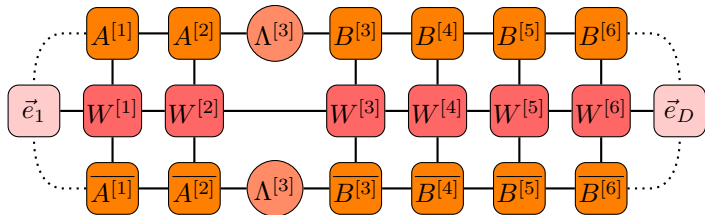
[3]: psi_FM = init_FM_MPS(L=10)
print(psi_FM)
<toycodes.a_mps.SimpleMPS object at 0x7fbf8a0efd00>

[4]: Z = np.diag([1., -1.])
print(psi_FM.site_expectation_value(Z))
[1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

Simple 1 2 main Python 3 (ipykernel) | Idle Mode: Command Ln 1, Col 1 exercise\_2\_mps.ipynb

# Density matrix renormalization group (DMRG) algorithm

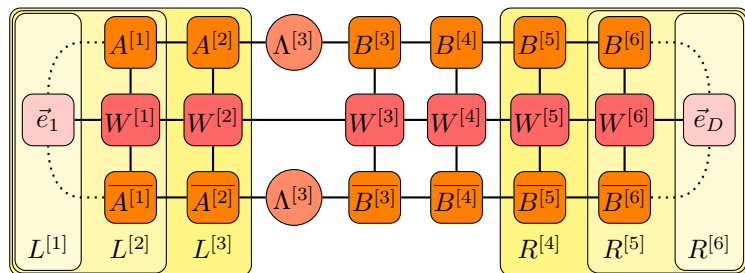
$$E = \langle \psi | \hat{H} | \psi \rangle =$$





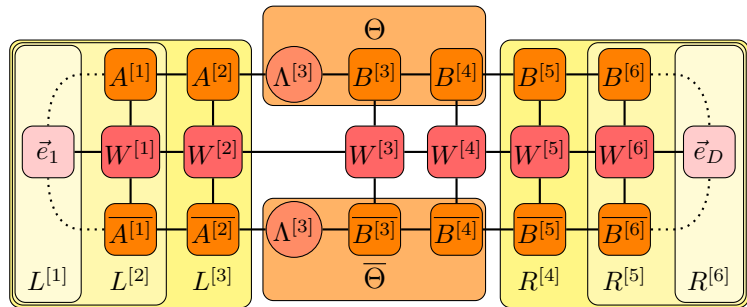
# Density matrix renormalization group (DMRG) algorithm

$$E = \langle \psi | \hat{H} | \psi \rangle =$$

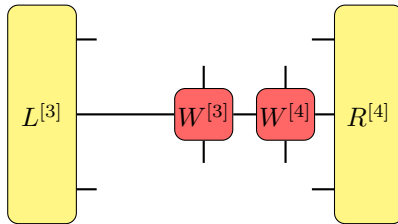


# Density matrix renormalization group (DMRG) algorithm

$$E = \langle \Theta | \hat{H}_{\text{eff}} | \Theta \rangle =$$

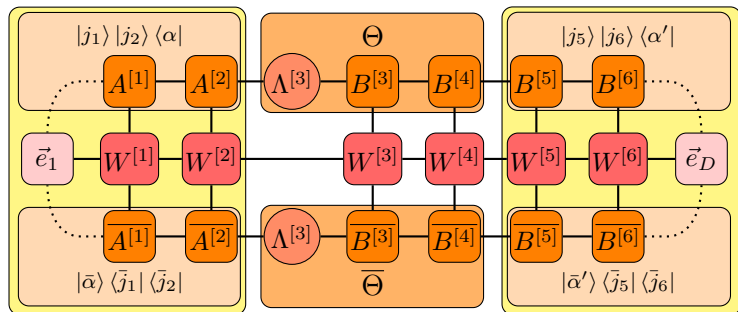


$$\hat{H}_{\text{eff}} =$$



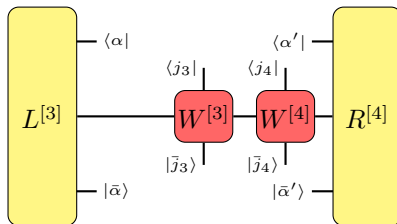
# Density matrix renormalization group (DMRG) algorithm

$$E = \langle \Theta | \hat{H}_{\text{eff}} | \Theta \rangle =$$

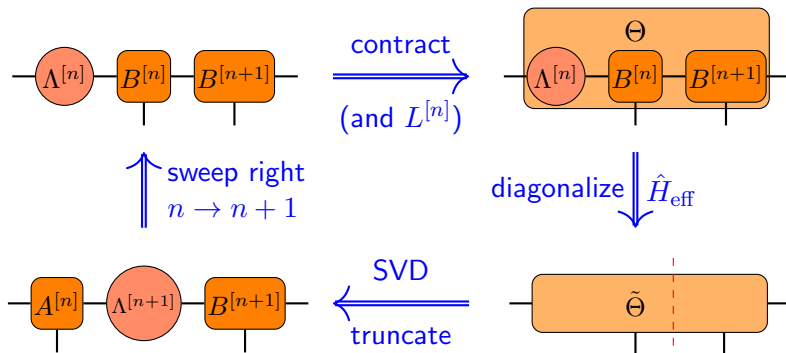


$\Rightarrow$  diagonalize  $\hat{H}_{\text{eff}}$   
with Lanczos  
to update  $\Theta \rightarrow \tilde{\Theta}$

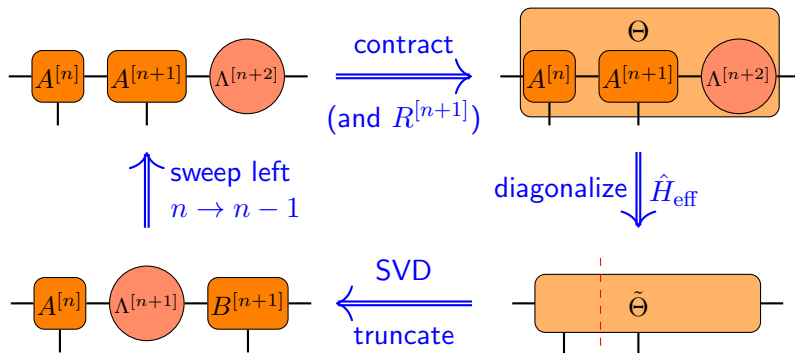
$$\hat{H}_{\text{eff}} =$$



# DMRG update loop

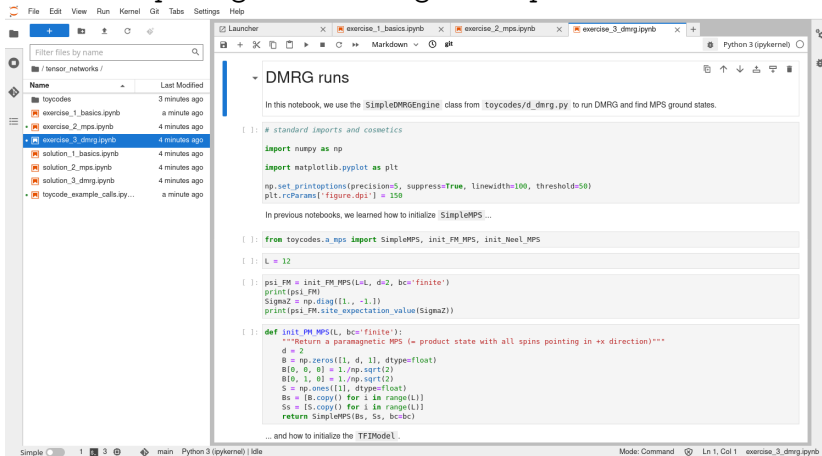


# DMRG update loop



# Exercise 3: DMRG

<https://quanthub.pks.mpg.de>, Login and Passwort same as for WiFi  
also [https://github.com/mgbukov/quant22\\_notebooks](https://github.com/mgbukov/quant22_notebooks)



The screenshot displays the Quanthub JupyterLab environment. On the left, a file browser shows the directory structure under `/tensor_networks/`, with `exercise_3_dmrg.ipynb` selected. The main area shows the notebook content, titled "DMRG runs". The text explains that the `SimpleDMRGEngine` class from `toycode/d_dmrg.py` is used to run DMRG and find MPS ground states. The notebook includes several code cells:

- Standard imports and cosmetics:

```
[ ]: # standard imports and cosmetics
import numpy as np
import matplotlib.pyplot as plt
np.set_printoptions(precision=5, suppress=True, linewidth=100, threshold=50)
plt.rcParams['figure.dpi'] = 150
```
- Initialization of SimpleMPS (text description):

```
[ ]: from toycode.a_mps import SimpleMPS, init_FM_MPS, init_Neel_MPS
```
- Setting system size L:

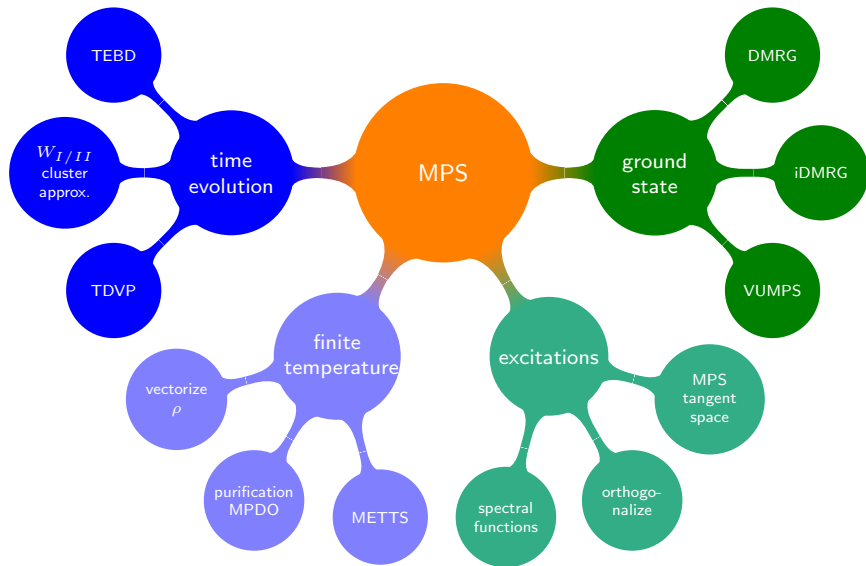
```
[ ]: L = 12
```
- Initializing the MPS object:

```
[ ]: psi_FM = init_FM_MPS(L=L, d=2, bc='finite')
print(psi_FM)
SigmaZ = np.diag([1., -1.])
print(psi_FM.site_expectation_value(SigmaZ))
```
- Defining the SimpleMPS class:

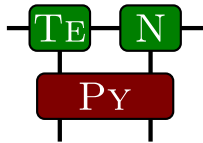
```
[ ]: def init_FM_MPS(L, bc='finite'):
    """Return a paramagnetic MPS (= product state with all spins pointing in +x direction)"""
    d = 2
    B = np.zeros([1, d, 1], dtype=float)
    B[0, 0, 0] = 1./np.sqrt(2)
    B[0, 1, 0] = 1./np.sqrt(2)
    S = np.ones([1], dtype=float)
    Bs = [B.copy() for i in range(L)]
    Ss = [S.copy() for i in range(L)]
    return SimpleMPS(Bs, Ss, bc=bc)
```

The notebook concludes with the text "... and how to initialize the TFIModel".






# Outlook: other MPS-based algorithms



## (Some) References



[github.com/tenpy/tenpy](https://github.com/tenpy/tenpy)

-  White, Phys Rev L **69** 2863 (1992)  
*"Density matrix formulation for quantum renormalization groups"*
-  JH, Pollmann, SciPost Phys. Lect. Notes 5 (2018), arXiv:1805.00055  
*"Efficient numerical simulations with Tensor Networks: Tensor Network Python (TeNPy)"*
-  Schollwoeck, Annals of Physics 326, 96 (2011), arXiv:1008.3477  
*"The density-matrix renormalization group in the age of matrix product states"*
-  Paeckel *et al.*, Annals of Physics 411, 167998 (2019), arXiv:1901.05824  
*"Time-evolution methods for matrix-product states"*
-  Vanderstraten, Haegemann, Verstraete, SciPost Phys. Lect. Notes 7 (2019), arXiv:1810.07006  
*"Tangent-space methods for uniform matrix product states"*
- Yearly schools: European tensor network [quantumtensor.eu](https://quantumtensor.eu)



# Appendix

1 Introduction

2 MPS

- MPS Expectation Values
- MPO

3 DMRG

4 References

5 Appendix

## More MPO examples

### Translation invariant MPO

$$\hat{H} = \sum_{i < k} C_i \left( \prod_{j=i+1}^{k-1} A_j \right) B_k + \sum_i D_i$$

$$W_{\alpha, \alpha'}^{[n]} = \left( \begin{array}{c|c|c} \mathbb{1} & C & D \\ \hline & A & B \\ \hline & & \mathbb{1} \end{array} \right)$$