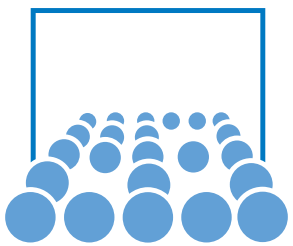
Numerical Linear and Multilinear Algebra in Quantum Tensor Networks

Konrad Waldherr

October 20, 2013

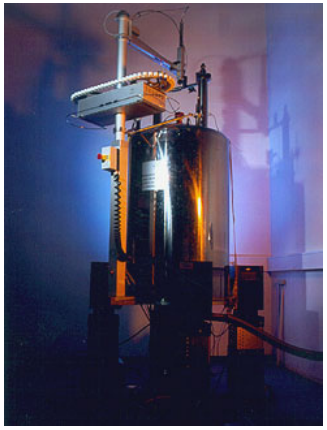


Joint work with Thomas Huckle

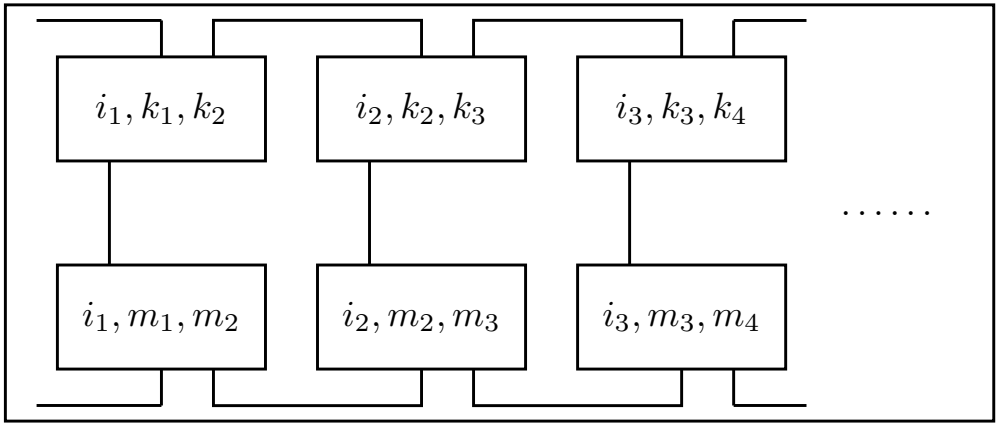
Outline

Numerical (Multi-) Linear Algebraic Approaches to...

- ...Quantum Control



- ...Quantum Tensor Networks



NMR Spectroscopy as Optimal Control Problem

- Goal: find optimal pulse amplitudes $\boldsymbol{u}(t)$ to steer nuclear spin states $\boldsymbol{x}(t)$ over the time interval $[0, T]$
- Optimality can be described by a scalar function f:

$$f(\boldsymbol{x}, \boldsymbol{u}) \;=\; \underbrace{f_T(\boldsymbol{x}(T))}_{\text{quality}}$$

- Application: Synthesization of a unitary target operator $\boldsymbol{U}_G$:

$$f(\boldsymbol{U}(T), \boldsymbol{u}) \;=\; \|\boldsymbol{U}(T) - \boldsymbol{U_G}\|_F^2$$

- The dynamics are governed by the equation of motion:

$$\dot{\boldsymbol{x}}(t) \;=\; -\mathrm{i} \underbrace{\left[\boldsymbol{H_0} + \sum_{m=1}^{M} \boldsymbol{u_m}(t) \boldsymbol{H_m}\right]}_{=\boldsymbol{H}(t)} \boldsymbol{x}(t)$$

Gradient Ascent Pulse Engineering Algorithm:

1: Set initial control amplitudes $u_m^{\{0\}}(t_k)$ for all times t_k and $\boldsymbol{U_0}$

2: **for** $\ell = 0, 1, 2, \ldots$ **until convergence**

a: Calculate **exponentials**

$$\boldsymbol{U_k} = e^{-\mathrm{i}\boldsymbol{H}(t_k)} \quad \text{for all} \quad k = 1, \ldots, K$$

b: Compute the **forward propagation**

$$\boldsymbol{U_{k:0}} = \boldsymbol{U_k U_{k-1}} \cdots \boldsymbol{U_1 U_0} \quad \text{for all} \quad k = 1, \ldots, K$$

c: Compute the **backward propagation**

$$\boldsymbol{W_{K+1:k+1}} = \boldsymbol{U_{K+1} U_K} \cdots \boldsymbol{U_{k+1} U_k} \quad \text{for all} \quad k = K, \ldots, 1$$

d: Calculate the **gradient**

$$\frac{\partial f}{\partial u_m(t_k)} = \mathrm{Re}\left[\mathrm{trace}\left(\boldsymbol{W}^{\mathbf{H}}_{\boldsymbol{K+1:k+1}}(-i\boldsymbol{H_m})\boldsymbol{U_{k:0}}\right)\right]$$

e: **Update** the controls $u_m(t_k)$

$$u_m^{\{\ell+1\}}(t_k) \;=\; u_m^{\{\ell\}}(t_k) + \epsilon \cdot \frac{\partial f}{\partial u_m(t_k)}$$

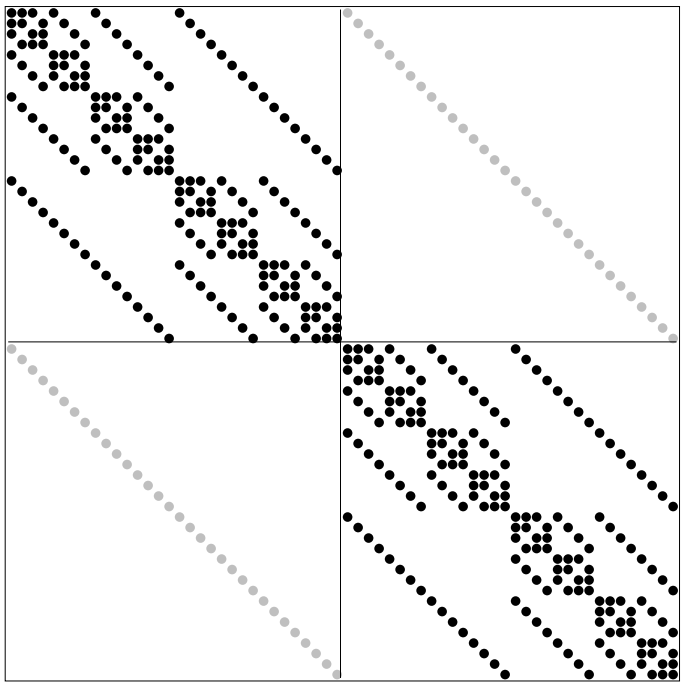
GRAPE: Computational Challenges

1. Computation of the matrix exponentials

$$\boldsymbol{U_k} = e^{-\mathrm{i}\left[\boldsymbol{H_0}+\sum_m u_m(t_k)\boldsymbol{H_m}\right]}$$

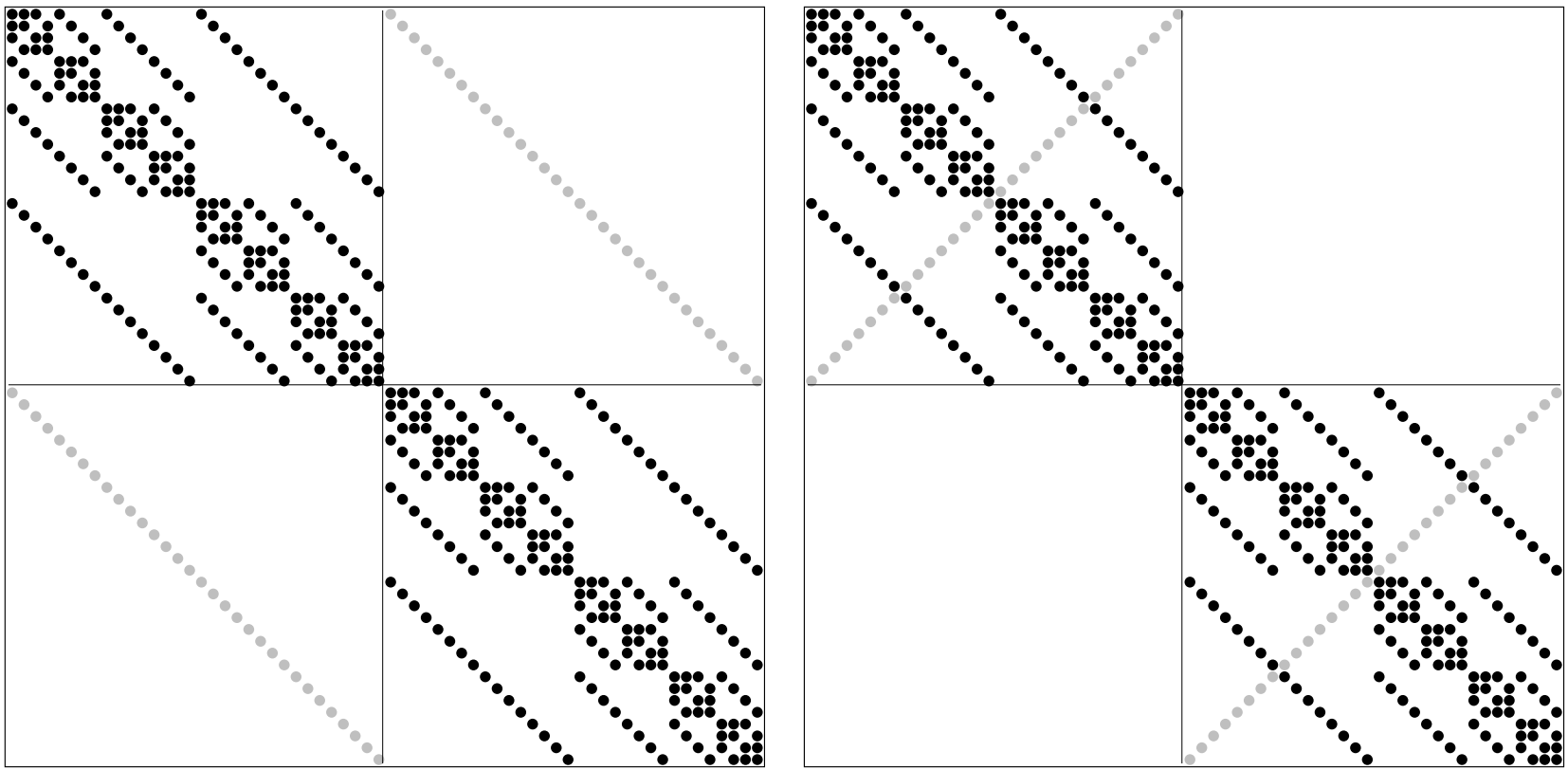
Challenges:

- Exponential of matrices are required in explicit form
- Operand matrices have certain properties (e.g., sparsity structure)



Matrix Exponentials

- **NMR setting:** Problem can be reduced to two problems of half size:
  cn^3 reduces to $2c\left(\frac{n}{2}\right)^3 = c\frac{n^3}{4}$
  $\Longrightarrow$ saves factor 4



Matrix Exponentials

- Old version: Eigendecomposition method
- New version: Chebyshev method

$$\boldsymbol{H} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{H}} \quad \Longrightarrow$$

$$\mathrm{e}^{-\mathrm{i}\Delta t\boldsymbol{H}} = \boldsymbol{V}\mathrm{e}^{-\mathrm{i}\Delta t\boldsymbol{\Lambda}}\boldsymbol{V}^{\mathrm{H}} = \boldsymbol{V}\,\mathrm{diag}\left(\mathrm{e}^{-\mathrm{i}\Delta t\lambda_1}, \ldots, \mathrm{e}^{-\mathrm{i}\Delta t\lambda_n}\right)\boldsymbol{V}^{\mathrm{H}}.$$

Requires eigendecomposition of a typically sparse matrix.

Matrix Exponentials

- Old version: Eigendecomposition method
- New version: Chebyshev method

$$\mathrm{e}^{-\mathrm{i}\Delta t\boldsymbol{H}} = a_0\boldsymbol{I} + 2\sum_{k=1}^{m} a_k C_k\left(\frac{\boldsymbol{H}}{\|\boldsymbol{H}\|}\right),\ a_k = \mathrm{i}^k J_k\left(-\Delta t\|\boldsymbol{H}\|\right)\ .$$

Chebyshev recurrence translates into

for $k = m$ **downto** 0 **do**
$$a_k = \mathrm{i}^k J_k\left(-\Delta t\|\boldsymbol{H}\|\right)$$
$$\boldsymbol{D_k} = a_k\boldsymbol{I} + \tfrac{2}{\|\boldsymbol{H}\|}\boldsymbol{H}\boldsymbol{D_{k+1}} - \boldsymbol{D_{k+2}}$$
$$\mathrm{e}^{-\mathrm{i}\Delta t\boldsymbol{H}} = \boldsymbol{D_0} - \boldsymbol{D_2}$$

Computation relies solely on products of the form `sparse` $\times$ `dense`.
$\Longrightarrow$ No expensive operations such as inversion/eigendecomposition required

GRAPE: Computational Challenges

2. Computation of the intermediate matrix products

- Compute the **forward propagation**

$$\boldsymbol{U_{k:0}} = \boldsymbol{U_k U_{k-1}} \cdots \boldsymbol{U_1 U_0} \quad \text{for all} \quad k = 1, \ldots, K$$

- Compute the **backward propagation**

$$\boldsymbol{W_{K+1:k+1}} = \boldsymbol{U_{K+1} U_K} \cdots \boldsymbol{U_{k+1} U_k} \quad \text{for all} \quad k = K, \ldots, 1$$

Instances of the matrix-multiplication prefix problem:

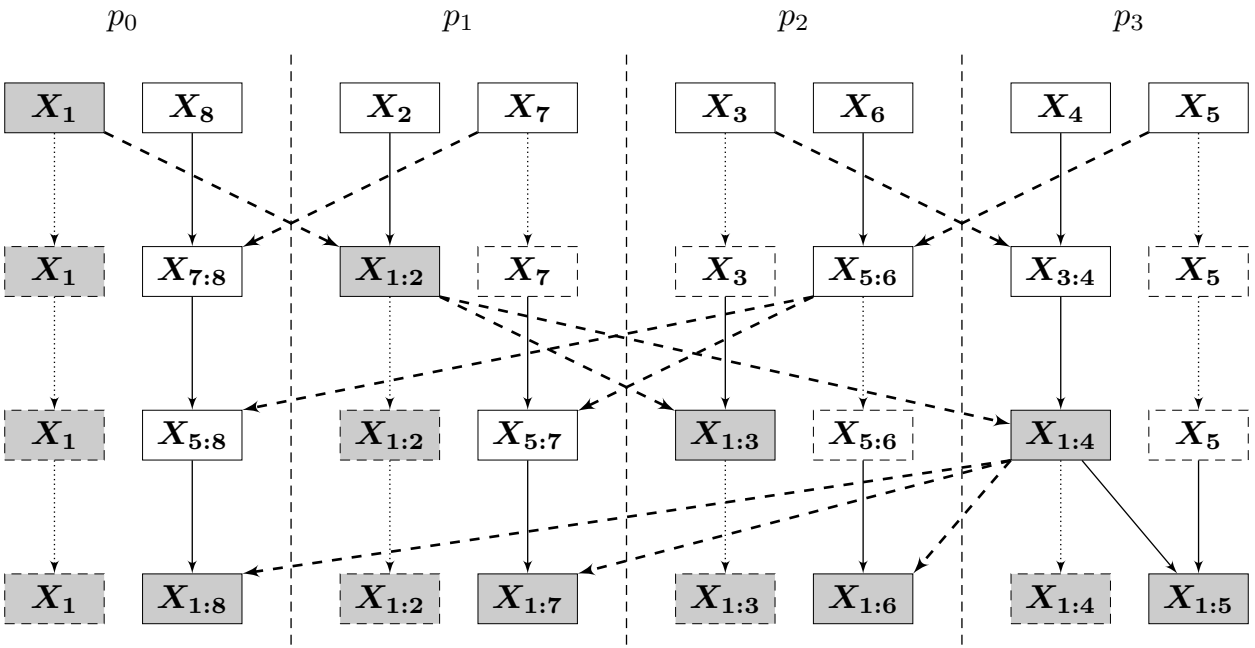
$$\forall k = 1 : K : \quad \left(\boldsymbol{X_1} \cdots \boldsymbol{X_{k-1}} \right) \boldsymbol{X_k}$$

Challenges:

- Parallelization
- Large number of matrices to be multiplied (e.g., $K = 1024$)
- Matrices comparably small (e.g., 1024×1024)

Computation of the Matrix Multiplications

- Old version: coarse-grain parallel prefix tree
- New version: fine-grain $3D$ parallelization:



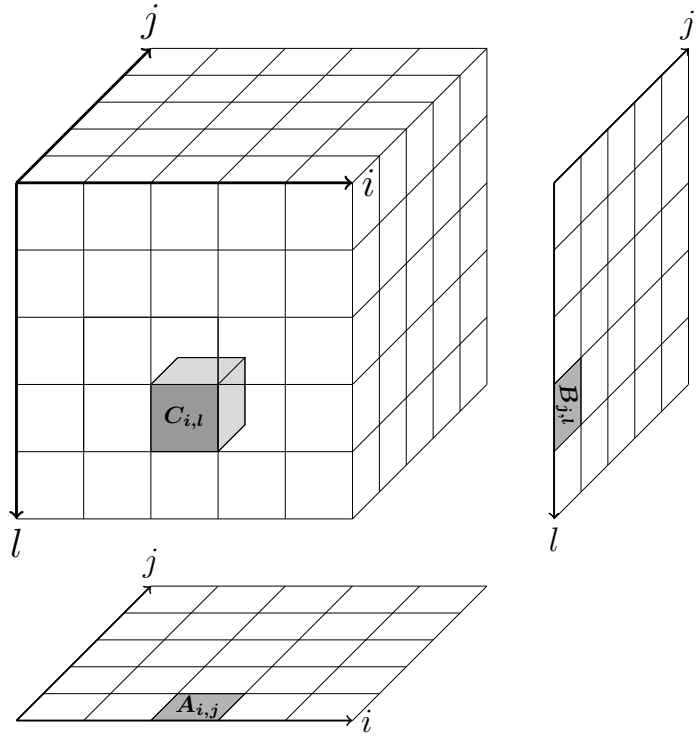
$\Longrightarrow$ individual matrix multiplications are computed **sequentially**

Computation of the Matrix Multiplications

- Old version: coarse-grain parallel prefix tree
- New version: fine-grain $3D$ parallelization:

$\Longrightarrow$ compute loop over k sequentially and
$\Longrightarrow$ **parallelize individual** matrix multiplications

- Total amount of work corresponds to a 3D cube.
- Each subblock corresponds to a block operation $C_{i,l} := C_{i,l} + A_{i,j}B_{j,l}$
- Find an appropriate "domain decomposition".
- 3D parallelization features lowest communication overhead



Numerical Results (10 spins)

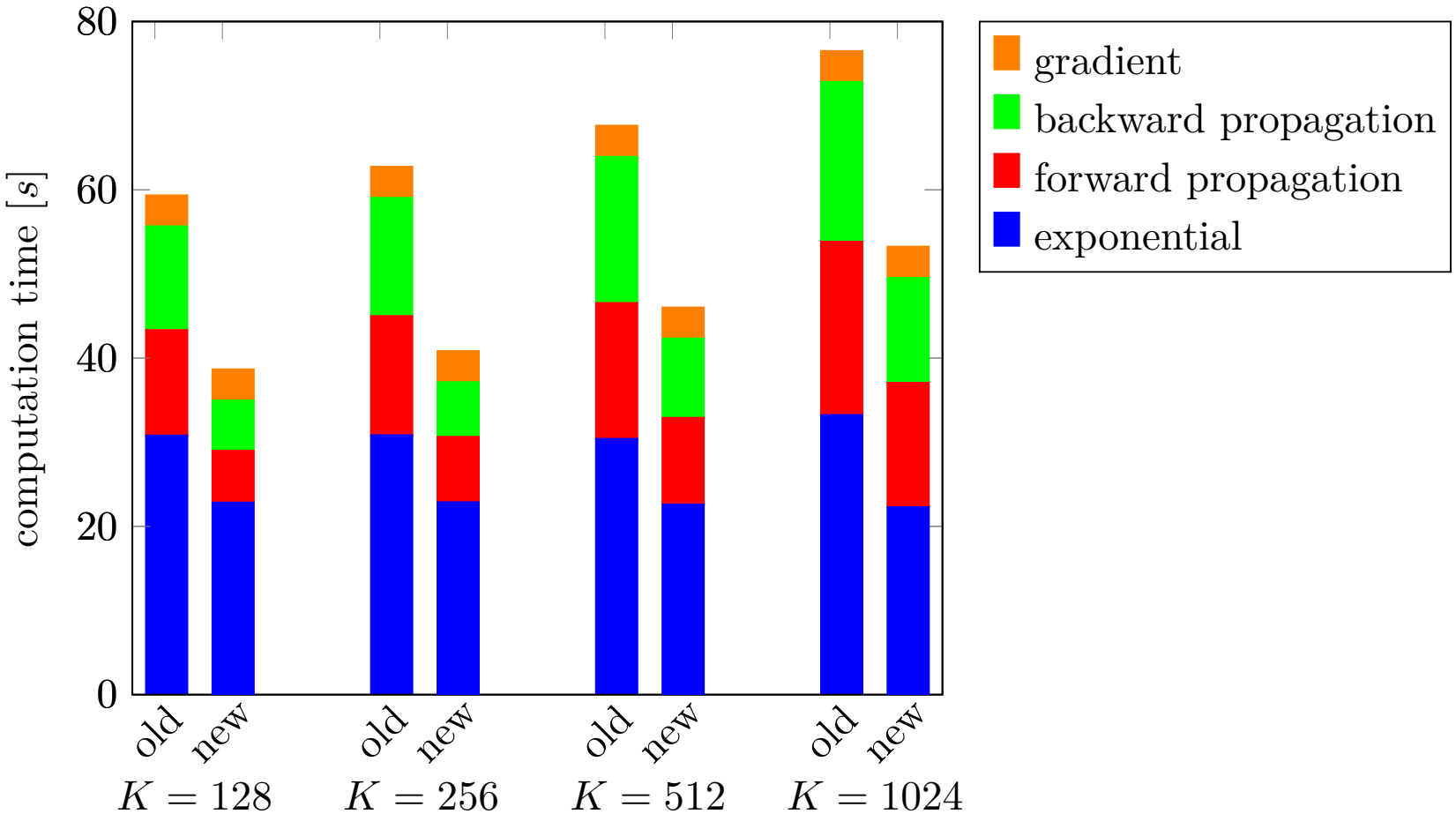


Figure : Runtime of one GRAPE iteration on the HLRB2 using $\frac{K}{2}$ processors.

Numerical Results (11 spins)

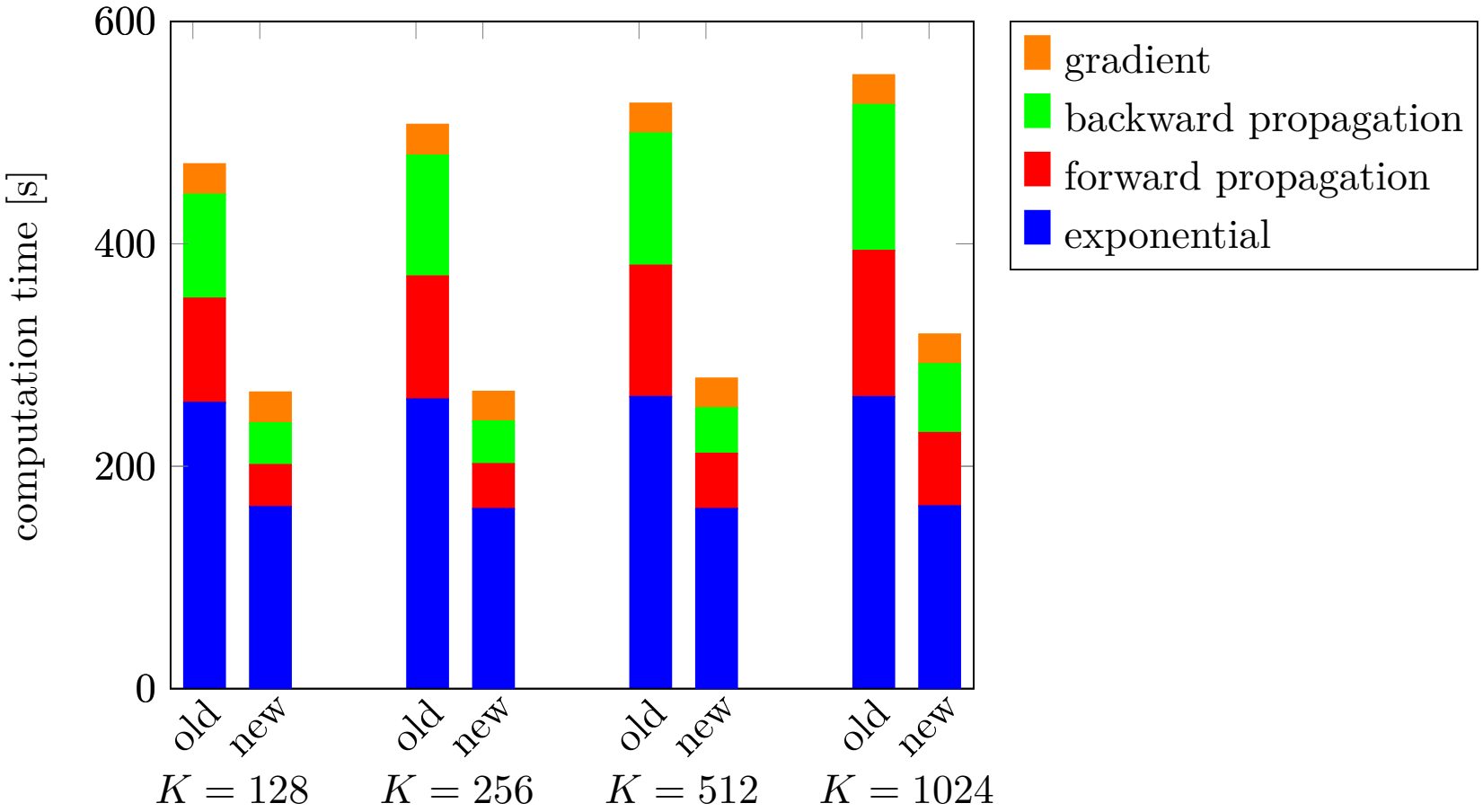


Figure : Runtime of one GRAPE iteration on the HLRB2 using $\frac{K}{2}$ processors.

Ground State of Quantum Systems

- Ground state: Compute lowest eigenvalue of the Hamiltonian

$$\boldsymbol{H} = \sum_{k=1}^{M} \alpha_k \boldsymbol{Q}_{\boldsymbol{1}}^{(\boldsymbol{k})} \otimes \cdots \otimes \boldsymbol{Q}_{\boldsymbol{N}}^{(\boldsymbol{k})} \in \mathbb{C}^{2^N \times 2^N}$$

- Challenges:
  - the vector space $\mathcal{V} = \mathbb{C}^{2^N}$ grows **exponentially** in N
  - for large N (e.g., $N = 100$) traditional matrix methods fail
- **Solution: data-sparse representation format** that allows
  - to overcome the curse of dimensionality,
  - to describe the right “corner” of the Hilbert space,
  - for efficient computations and algorithms.

Ground State of Quantum Systems

- Ground state: Compute lowest eigenvalue of the Hamiltonian

$$\boldsymbol{H} = \sum_{k=1}^{M} \alpha_k \boldsymbol{Q}_{\boldsymbol{1}}^{(\boldsymbol{k})} \otimes \cdots \otimes \boldsymbol{Q}_{\boldsymbol{N}}^{(\boldsymbol{k})} \in \mathbb{C}^{2^N \times 2^N}$$

- Challenges:
  - the vector space $\mathcal{V} = \mathbb{C}^{2^N}$ grows **exponentially** in N
  - for large N (e.g., $N = 100$) traditional matrix methods fail
- **Solution: data-sparse representation format** that allows
  - to overcome the curse of dimensionality,
  - to describe the right “corner” of the Hilbert space,
  - for efficient computations and algorithms.
- Two approaches:
  - subspace method in the MPS format,
  - symmetry-adapted variational ground-state search.

Matrix Product States

- In physical simulations, **Matrix Product States** (MPS) are in use:

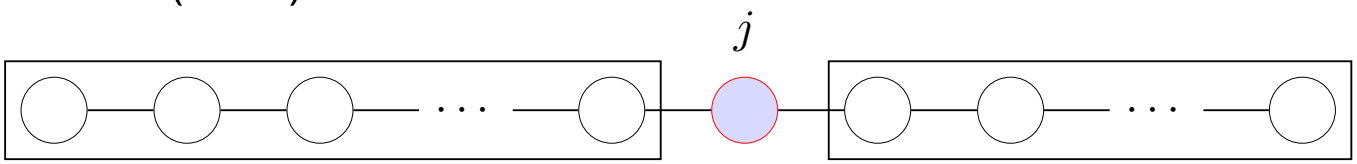
$$x = \sum_{i_1,\ldots,i_N} \operatorname{tr}\left(\boldsymbol{A}_{\mathbf{1}}^{(i_1)} \cdot \boldsymbol{A}_{\mathbf{2}}^{(i_2)} \cdots \boldsymbol{A}_{\boldsymbol{N}}^{(i_N)}\right) \boldsymbol{e}_{i_1 i_2 \ldots i_N} \cdot$$

  with $D_j \times D_{j+1}$ matrices $\boldsymbol{A}_{\boldsymbol{j}}^{(i_j)}$.

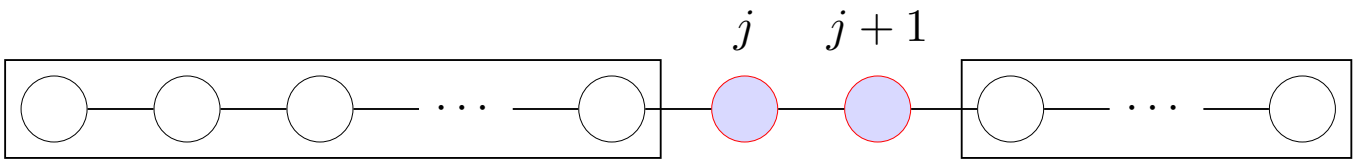
- MPS representation requires $2ND^2$ instead of 2^N entries.

Minimization in terms of MPS

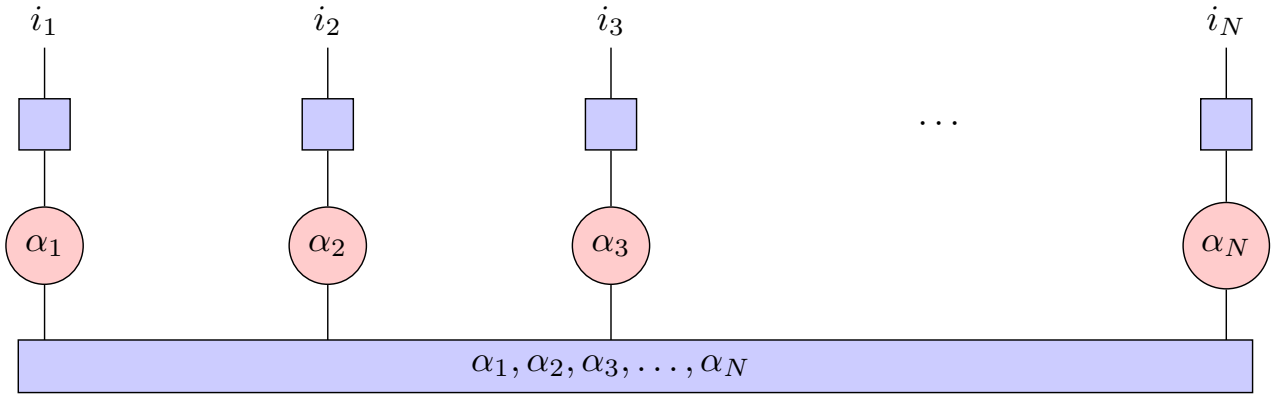
- vMPS (ALS):



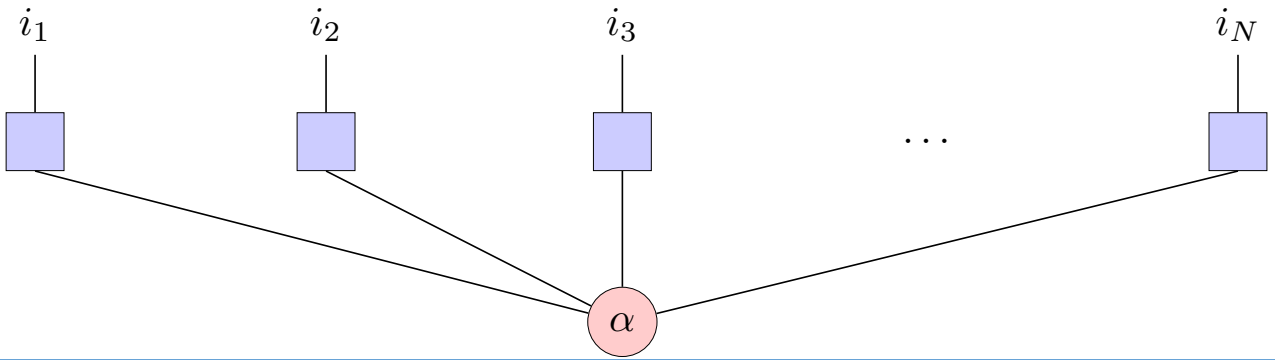
- DMRG (MALS):



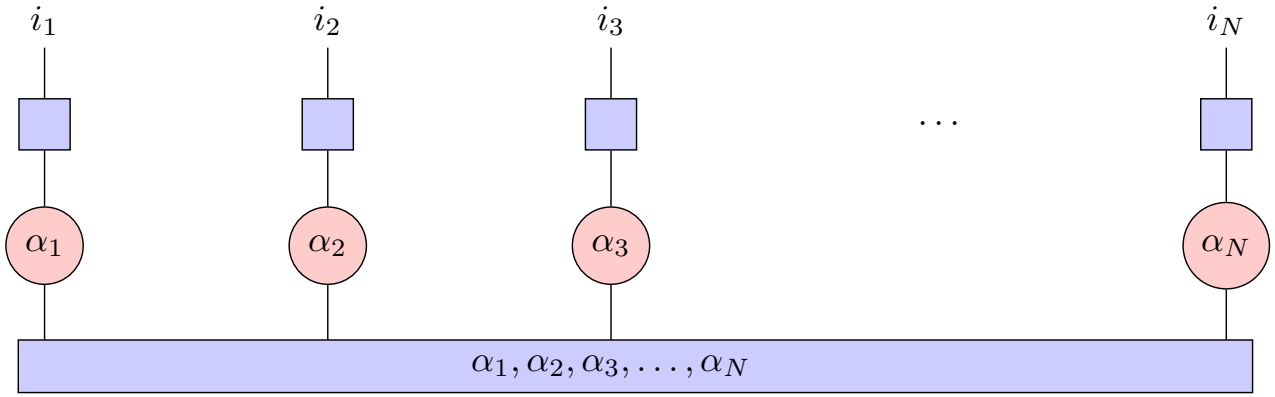
Tucker:


i_1
i_2
i_3
i_N
$\cdots$
α_1
α_2
α_3
α_N
$\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_N$

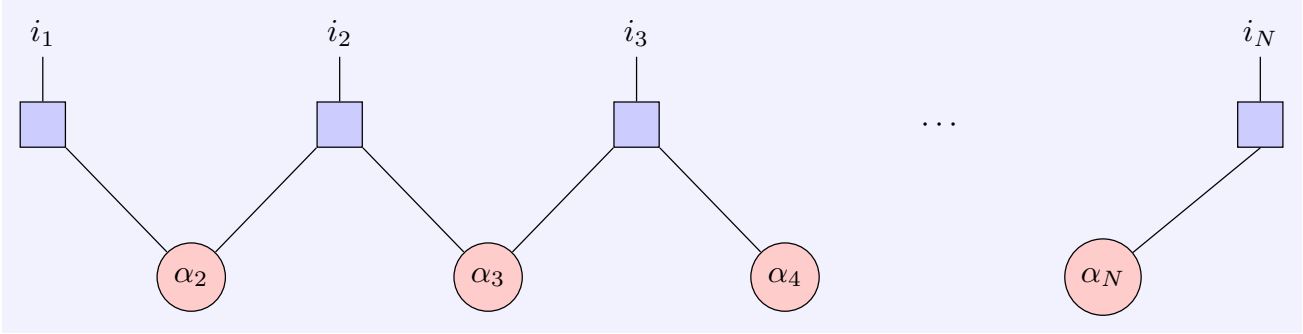
CP:


i_1
i_2
i_3
i_N
$\cdots$
α

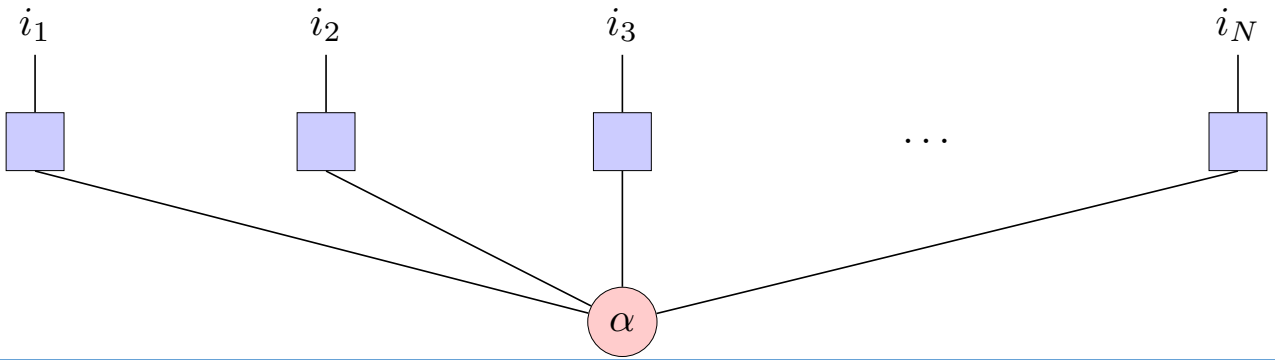
Tucker:



TT/MPS:



CP:



The Lanczos Algorithm

Algorithm: Lanczos iteration for Hermitian matrix $\boldsymbol{H}$

Choose initial guess $\boldsymbol{r_0} \neq \boldsymbol{0}$
Define $\beta_0 = \|\boldsymbol{r_0}\|$ and $\boldsymbol{q_0} = \boldsymbol{0}$
for $j = 1, 2, \ldots, m$ **do**

$$\boldsymbol{q_j} = \boldsymbol{r_{j-1}} / \beta_{j-1}$$
$$\boldsymbol{u_j} = \boldsymbol{H q_j}$$
$$\alpha_j = \boldsymbol{q_j^H u_j}$$
$$\boldsymbol{r_j} = \boldsymbol{u_j} - \alpha_j \boldsymbol{q_j} - \beta_{j-1} \boldsymbol{q_{j-1}}$$
$$\beta_j = \|\boldsymbol{r_j}\|$$

$$\boldsymbol{T_m} = \mathrm{tridiag}(\boldsymbol{\beta}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$
$$(\lambda, \boldsymbol{y}) = \mathrm{eig}(\boldsymbol{T_m})$$
$$\boldsymbol{v} = \boldsymbol{Q y} = (\boldsymbol{q_1}, \ldots, \boldsymbol{q_m}) \boldsymbol{y}$$

The Lanczos Algorithm

Algorithm: Lanczos iteration for Hermitian matrix $\boldsymbol{H}$

Choose initial guess $\boldsymbol{r_0} \neq \boldsymbol{0}$
Define $\beta_0 = \|\boldsymbol{r_0}\|$ and $\boldsymbol{q_0} = \boldsymbol{0}$
for $j = 1, 2, \ldots, m$ **do**
$\boldsymbol{q_j} = \boldsymbol{r_{j-1}} / \beta_{j-1}$
$\boldsymbol{u_j} = \boldsymbol{H q_j}$
$\alpha_j = \boldsymbol{q_j^H u_j}$
$\boldsymbol{r_j} = \boldsymbol{u_j} - \alpha_j \boldsymbol{q_j} - \beta_{j-1} \boldsymbol{q_{j-1}}$
$\beta_j = \|\boldsymbol{r_j}\|$
$\boldsymbol{T_m} = \mathrm{tridiag}(\boldsymbol{\beta}, \boldsymbol{\alpha}, \boldsymbol{\beta})$
$(\lambda, \boldsymbol{y}) = \mathrm{eig}(\boldsymbol{T_m})$
$\boldsymbol{v} = \boldsymbol{Q y} = (\boldsymbol{q_1}, \ldots, \boldsymbol{q_m}) \boldsymbol{y}$

Computational tasks:

The Lanczos Algorithm

Algorithm: Lanczos iteration for Hermitian matrix $\boldsymbol{H}$

Choose initial guess $\boldsymbol{r_0} \neq \boldsymbol{0}$
Define $\beta_0 = \|\boldsymbol{r_0}\|$ and $\boldsymbol{q_0} = \boldsymbol{0}$
for $j = 1, 2, \ldots, m$ **do**

$$\boldsymbol{q_j} = \boldsymbol{r_{j-1}} / \beta_{j-1}$$
$$\boldsymbol{u_j} = \boldsymbol{H q_j}$$
$$\alpha_j = \boldsymbol{q_j^H u_j}$$
$$\boldsymbol{r_j} = \boldsymbol{u_j} - \alpha_j \boldsymbol{q_j} - \beta_{j-1} \boldsymbol{q_{j-1}}$$
$$\beta_j = \|\boldsymbol{r_j}\|$$

$$\boldsymbol{T_m} = \mathrm{tridiag}(\boldsymbol{\beta}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$
$$(\lambda, \boldsymbol{y}) = \mathrm{eig}(\boldsymbol{T_m})$$
$$\boldsymbol{v} = \boldsymbol{Q y} = (\boldsymbol{q_1}, \ldots, \boldsymbol{q_m}) \boldsymbol{y}$$

Computational tasks:

1. Application of the matrix to a vector

The Lanczos Algorithm

Algorithm: Lanczos iteration for Hermitian matrix H

Choose initial guess $r_0 \neq 0$
Define $\beta_0 = \|r_0\|$ and $q_0 = 0$
for $j = 1, 2, \ldots, m$ **do**

$$q_j = r_{j-1}/\beta_{j-1}$$
$$u_j = H q_j$$
$$\alpha_j = q_j^H u_j$$
$$r_j = u_j - \alpha_j q_j - \beta_{j-1} q_{j-1}$$
$$\beta_j = \|r_j\|$$

$$T_m = \text{tridiag}(\beta, \alpha, \beta)$$
$$(\lambda, y) = \text{eig}(T_m)$$
$$v = Qy = (q_1, \ldots, q_m)y$$

Computational tasks:

1. Application of the matrix to a vector
2. Computation of inner products

The Lanczos Algorithm

Algorithm: Lanczos iteration for Hermitian matrix H

Choose initial guess $r_0 \neq 0$
Define $\beta_0 = \|r_0\|$ and $q_0 = 0$
for $j = 1, 2, \ldots, m$ **do**

$$q_j = r_{j-1}/\beta_{j-1}$$
$$u_j = H q_j$$
$$\alpha_j = q_j^H u_j$$
$$r_j = u_j - \alpha_j q_j - \beta_{j-1} q_{j-1}$$
$$\beta_j = \|r_j\|$$

$$T_m = \text{tridiag}(\beta, \alpha, \beta)$$
$$(\lambda_{\min}, y) = \text{eig}(T_m)$$
$$v = Qy = (q_1, \ldots, q_m)y$$

Computational tasks:

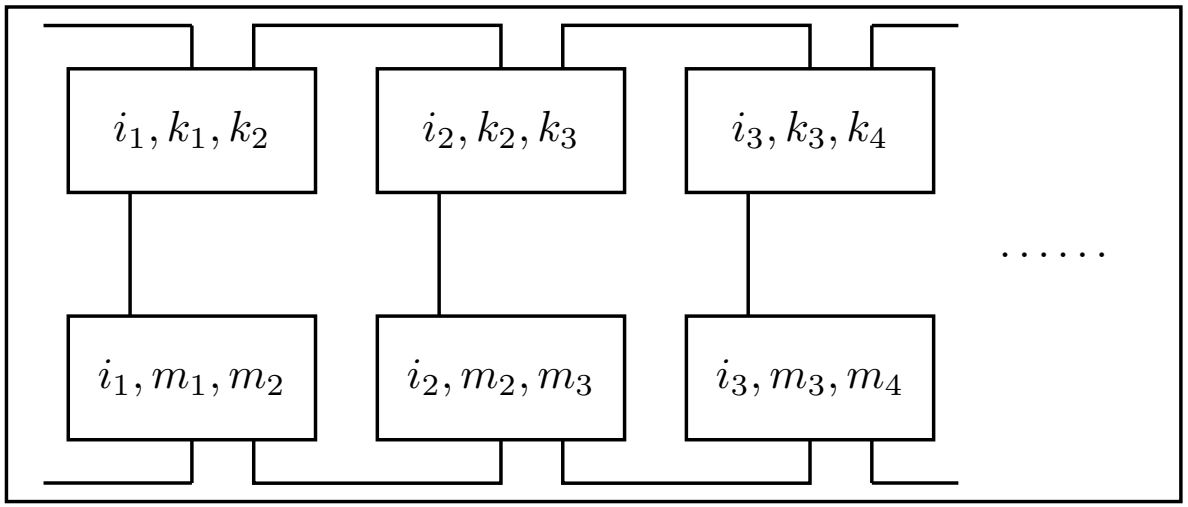
1. Application of the matrix to a vector
2. Computation of inner products
3. Sums and linear combinations of vectors

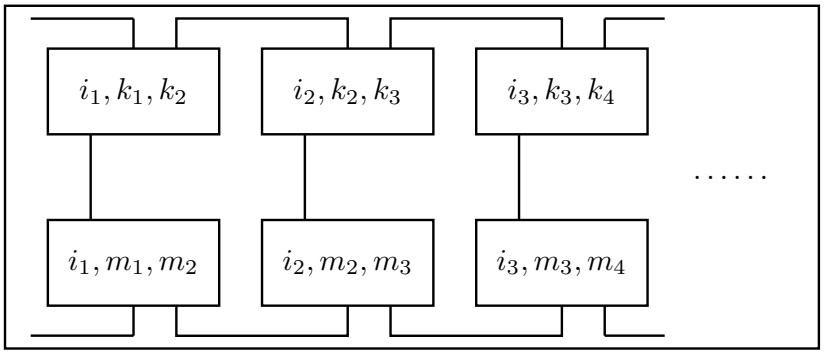
Computation of Inner Products

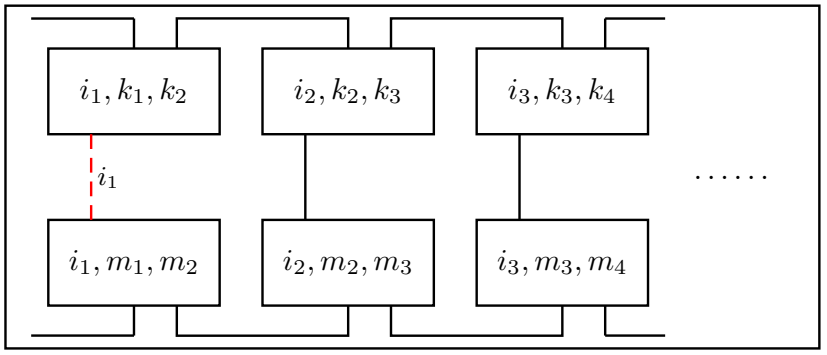
$$\boldsymbol{x}^{\mathrm{H}}\boldsymbol{y} = \sum_{i_1,\ldots,i_N} \left(\operatorname{tr}\left(\bar{\boldsymbol{A}}_{\boldsymbol{1}}^{(\boldsymbol{i_1})} \cdots \bar{\boldsymbol{A}}_{\boldsymbol{N}}^{(\boldsymbol{i_N})}\right)\right) \cdot \left(\operatorname{tr}\left(\boldsymbol{B}_{\boldsymbol{1}}^{(\boldsymbol{i_1})} \cdots \boldsymbol{B}_{\boldsymbol{N}}^{(\boldsymbol{i_N})}\right)\right)$$

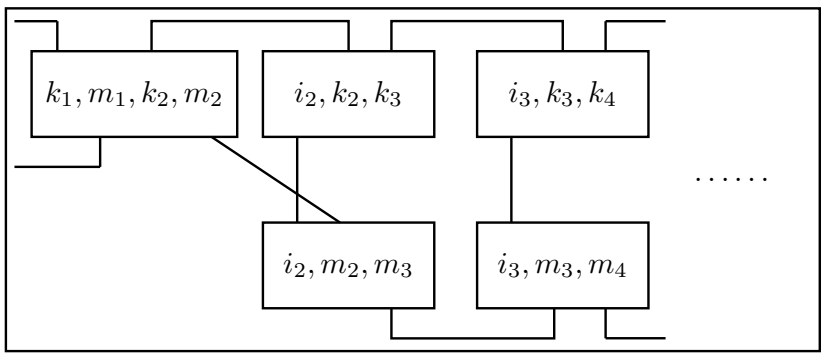
$$= \sum_{i_j} \sum_{k_j} \overline{a_{1;k_1,k_2}^{(i_1)}} \cdots \overline{a_{N;k_N,k_1}^{(i_N)}} \sum_{m_j} b_{1;m_1,m_2}^{(i_1)} \cdots b_{N;m_N,m_1}^{(i_N)}$$

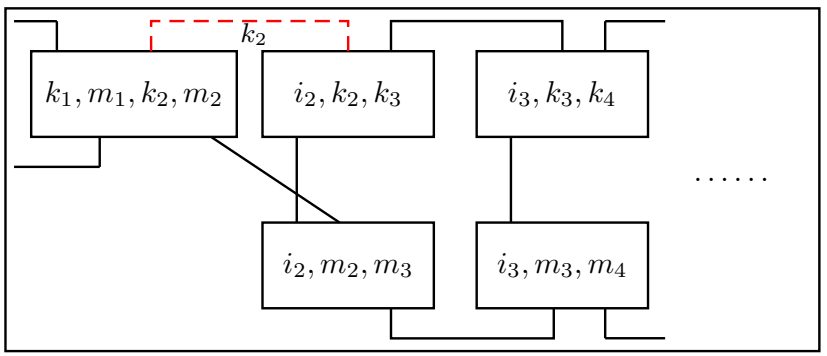
$\Longrightarrow$ Find an efficient ordering of the summations m_j, k_j, i_j.
Consider the tensor network

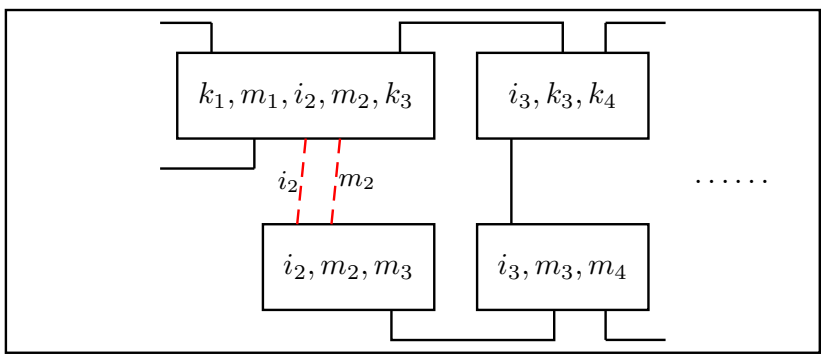


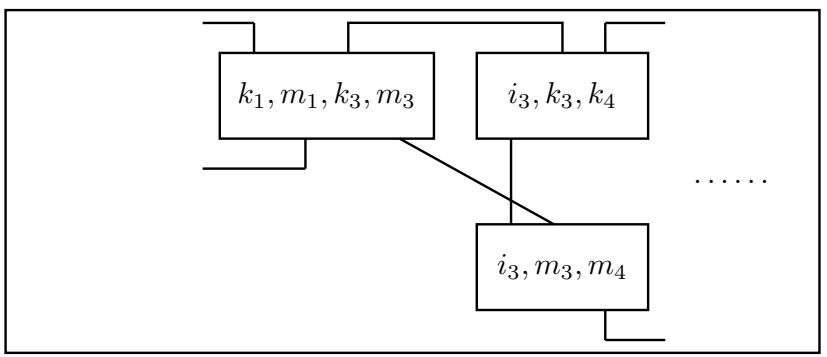

i_1, k_1, k_2
i_2, k_2, k_3
i_3, k_3, k_4
i_1, m_1, m_2
i_2, m_2, m_3
i_3, m_3, m_4


i_1, k_1, k_2
i_2, k_2, k_3
i_3, k_3, k_4
i_1
i_1, m_1, m_2
i_2, m_2, m_3
i_3, m_3, m_4


k_1, m_1, k_2, m_2
i_2, k_2, k_3
i_3, k_3, k_4
i_2, m_2, m_3
i_3, m_3, m_4


k_2
k_1, m_1, k_2, m_2
i_2, k_2, k_3
i_3, k_3, k_4
i_2, m_2, m_3
i_3, m_3, m_4


k_1, m_1, i_2, m_2, k_3
i_3, k_3, k_4
i_2
m_2
i_2, m_2, m_3
i_3, m_3, m_4


k_1, m_1, k_3, m_3
i_3, k_3, k_4
i_3, m_3, m_4

Computation of the Sum of Two MPS

- Sum of two MPS vectors

$$\boldsymbol{x} = \sum_{i_1,\ldots,i_N} \operatorname{tr}\left(\boldsymbol{A}_1^{(i_1)} \boldsymbol{A}_2^{(i_2)} \ldots \boldsymbol{A}_N^{(i_N)}\right) \boldsymbol{e}_{i_1 \ldots i_N}$$

$$\boldsymbol{y} = \sum_{i_1,\ldots,i_N} \operatorname{tr}\left(\boldsymbol{B}_1^{(i_1)} \boldsymbol{B}_2^{(i_2)} \ldots \boldsymbol{B}_N^{(i_N)}\right) \boldsymbol{e}_{i_1 \ldots i_N}$$

$$(\boldsymbol{x}+\boldsymbol{y})_{i_1,\ldots,i_N} =$$

$$\operatorname{tr}\left[\begin{pmatrix} \boldsymbol{A}_1^{(i_1)} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B}_1^{(i_1)} \end{pmatrix} \begin{pmatrix} \boldsymbol{A}_2^{(i_2)} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B}_2^{(i_2)} \end{pmatrix} \cdots \begin{pmatrix} \boldsymbol{A}_N^{(i_N)} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B}_N^{(i_N)} \end{pmatrix}\right]$$

- Adding MPS leads to an **increase** of the bond dimensions:

$$\boldsymbol{x}_{\mathrm{MPS}_D} + \boldsymbol{y}_{\mathrm{MPS}_D} = \boldsymbol{z}_{\mathrm{MPS}_{2D}}$$

Matrix Product Operators (MPO)

The MPS concept

$$x_{i_1,i_2,\ldots,i_N} = \mathrm{tr}\left(\boldsymbol{A}_{\boldsymbol{1}}^{(\boldsymbol{i_1})}\boldsymbol{A}_{\boldsymbol{2}}^{(\boldsymbol{i_2})}\cdots\boldsymbol{A}_{\boldsymbol{N}}^{(\boldsymbol{i_N})}\right)$$

can be generalized to operators:

$$O_{\substack{i_1,\ldots,i_N\\ j_1,\ldots,j_N}} = \mathrm{tr}\left(\boldsymbol{W}_{\boldsymbol{1}}^{(\boldsymbol{i_1},\boldsymbol{j_1})}\boldsymbol{W}_{\boldsymbol{2}}^{(\boldsymbol{i_2},\boldsymbol{j_2})}\cdots\boldsymbol{W}_{\boldsymbol{N}}^{(\boldsymbol{i_N},\boldsymbol{j_N})}\right)$$

Application of such an MPO to an MPS leads to

$$\mathrm{tr}\left[\left(\sum_{j_1}\boldsymbol{W}_{\boldsymbol{1}}^{(\boldsymbol{i_1},\boldsymbol{j_1})}\otimes\boldsymbol{A}_{\boldsymbol{1}}^{(\boldsymbol{j_1})}\right)\cdots\left(\sum_{j_N}\boldsymbol{W}_{\boldsymbol{N}}^{(\boldsymbol{i_N},\boldsymbol{j_N})}\otimes\boldsymbol{A}_{\boldsymbol{N}}^{(\boldsymbol{j_N})}\right)\right],$$

i.e., an MPS of larger size $D_{\mathrm{MPS}}D_{\mathrm{MPO}}$.

- Usually exact representation with small ranks
- Not dependent on N

Augmentation of ranks

- Increase of the bond dimension during the Lanczos algorithm

Lanczos algorithm	Bond dimension
for $j = 1, 2, \ldots, m$	
$\boldsymbol{q_j} = \boldsymbol{r_{j-1}} / \beta_{j-1}$	$D^{(j)}$
$\boldsymbol{u_j} = \boldsymbol{H}\boldsymbol{q_j}$	$D_O D^{(j)}$
$\alpha_j = \boldsymbol{q_j^{\mathrm{H}}} \boldsymbol{u_j}$	
$\boldsymbol{r_j} = \boldsymbol{H}\boldsymbol{q_j} - \alpha_j \boldsymbol{q_j} - \beta_{j-1}\boldsymbol{q_{j-1}}$	$D_O D^{(j)} + D^{(j)} + D^{(j-1)} =$
$\beta_j = \lVert \boldsymbol{r_j} \rVert$	$= D^{(j+1)}$
$\boldsymbol{v} = \boldsymbol{Q}\boldsymbol{y} = (\boldsymbol{q_1}, \ldots, \boldsymbol{q_m})\boldsymbol{y}$	$D^{(1)} + D^{(2)} + \cdots + D^{(m)}$

Augmentation of ranks

- Increase of the bond dimension during the Lanczos algorithm

Lanczos algorithm	Bond dimension
for $j = 1, 2, \ldots, m$	
$\boldsymbol{q_j} = \boldsymbol{r_{j-1}} / \beta_{j-1}$	$D^{(j)}$
$\boldsymbol{u_j} = \boldsymbol{H}\boldsymbol{q_j}$	$D_O D^{(j)}$
$\alpha_j = \boldsymbol{q_j^{\mathrm{H}}} \boldsymbol{u_j}$	
$\boldsymbol{r_j} = \boldsymbol{H}\boldsymbol{q_j} - \alpha_j \boldsymbol{q_j} - \beta_{j-1}\boldsymbol{q_{j-1}}$	$D_O D^{(j)} + D^{(j)} + D^{(j-1)} =$
$\beta_j = \lVert \boldsymbol{r_j} \rVert$	$= D^{(j+1)}$
$\boldsymbol{v} = \boldsymbol{Q}\boldsymbol{y} = (\boldsymbol{q_1}, \ldots, \boldsymbol{q_m})\boldsymbol{y}$	$D^{(1)} + D^{(2)} + \cdots + D^{(m)}$

- Solution: keep the ranks $D^{(j)}$ bounded
  $\Longrightarrow$ Projection required

Compression/Projection of MPS

$$\mathcal{P}_D : \boldsymbol{y}_{MPS_{D'}} \mapsto \underset{\boldsymbol{x}_{MPS_D}}{\arg\min} \left\| \boldsymbol{y}_{MPS_{D'}} - \boldsymbol{x}_{MPS_D} \right\|_2^2 =$$

$$\underset{\boldsymbol{A}}{\arg\min} \left\| \left[\operatorname{tr}\left(\boldsymbol{B}_{\boldsymbol{1}}^{(i_1)} \cdots \boldsymbol{B}_{\boldsymbol{N}}^{(i_N)} \right) - \operatorname{tr}\left(\boldsymbol{A}_{\boldsymbol{1}}^{(i_1)} \cdots \boldsymbol{A}_{\boldsymbol{N}}^{(i_N)} \right) \right]_{i_1,\ldots,i_N} \right\|_2^2$$

SVD-based truncation:

- Suppose $\boldsymbol{A_1}, \ldots, \boldsymbol{A_{r-1}}$ are already constructed. $\boldsymbol{B_r}$ is $D \times D'$

$$\begin{pmatrix} \boldsymbol{B}_{\boldsymbol{r}}^{(0)} \\ \boldsymbol{B}_{\boldsymbol{r}}^{(1)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{U}_{\boldsymbol{r}}^{(0)} \\ \boldsymbol{U}_{\boldsymbol{r}}^{(1)} \end{pmatrix} \boldsymbol{\Sigma_r} \boldsymbol{V_r^H} \overset{trunc}{\approx} \begin{pmatrix} \tilde{\boldsymbol{U}}_{\boldsymbol{r}}^{(0)} \\ \tilde{\boldsymbol{U}}_{\boldsymbol{r}}^{(1)} \end{pmatrix} \tilde{\boldsymbol{\Sigma}}_{\boldsymbol{r}} \tilde{\boldsymbol{V}}_{\boldsymbol{r}}^{\mathbf{H}}$$

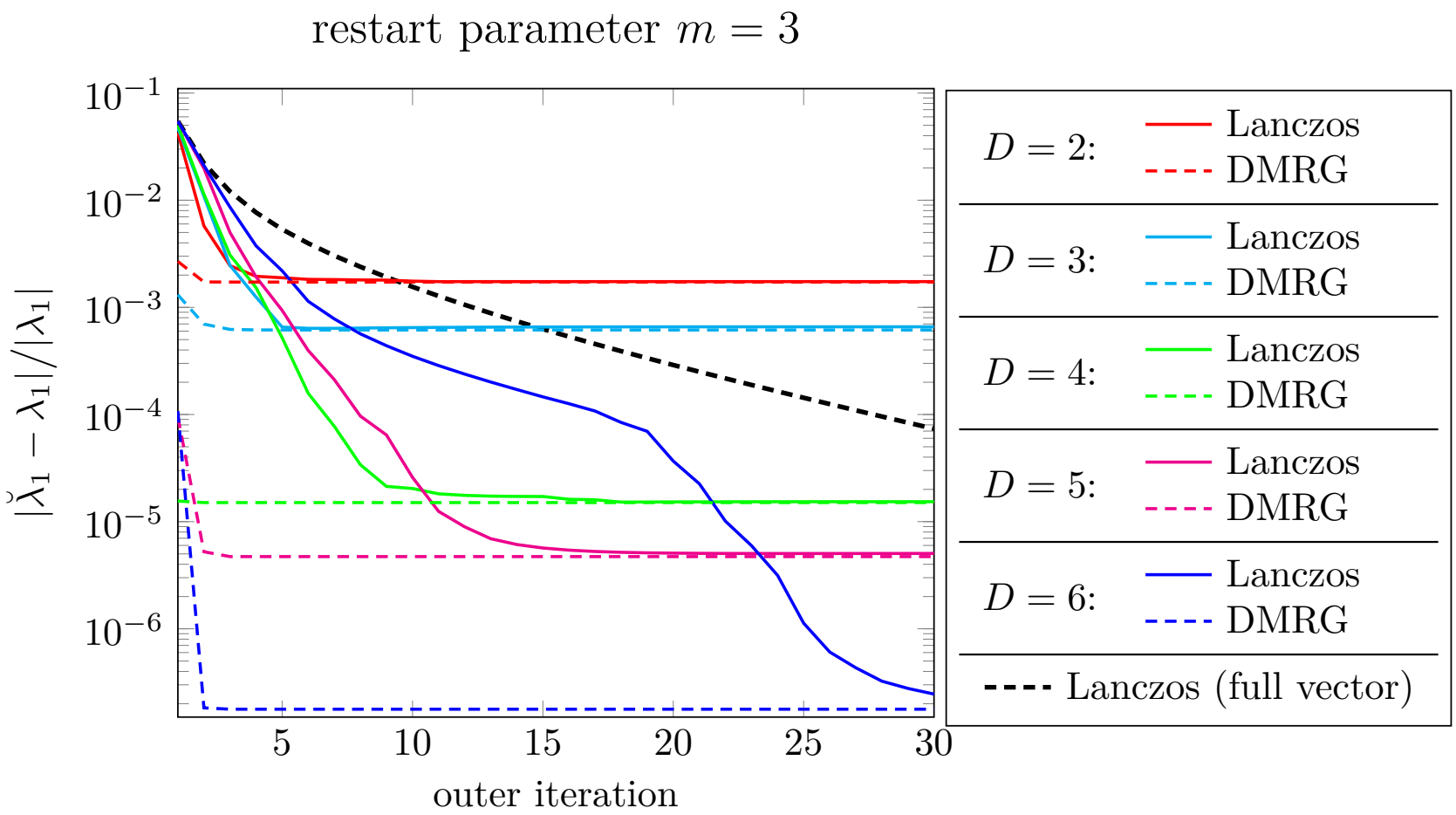
- Define $\boldsymbol{A}_{\boldsymbol{r}}^{(i_r)} = \tilde{\boldsymbol{U}}_{\boldsymbol{r}}^{(i_r)}$, a $D \times D$ matrix
- proceed with the $D \times D'$ matrix $\tilde{\boldsymbol{\Sigma}}_{\boldsymbol{r}} \tilde{\boldsymbol{V}}_{\boldsymbol{r}}^{\mathbf{H}} \boldsymbol{B}_{\boldsymbol{r+1}}^{(i_{r+1})}$

Restarted Lanczos

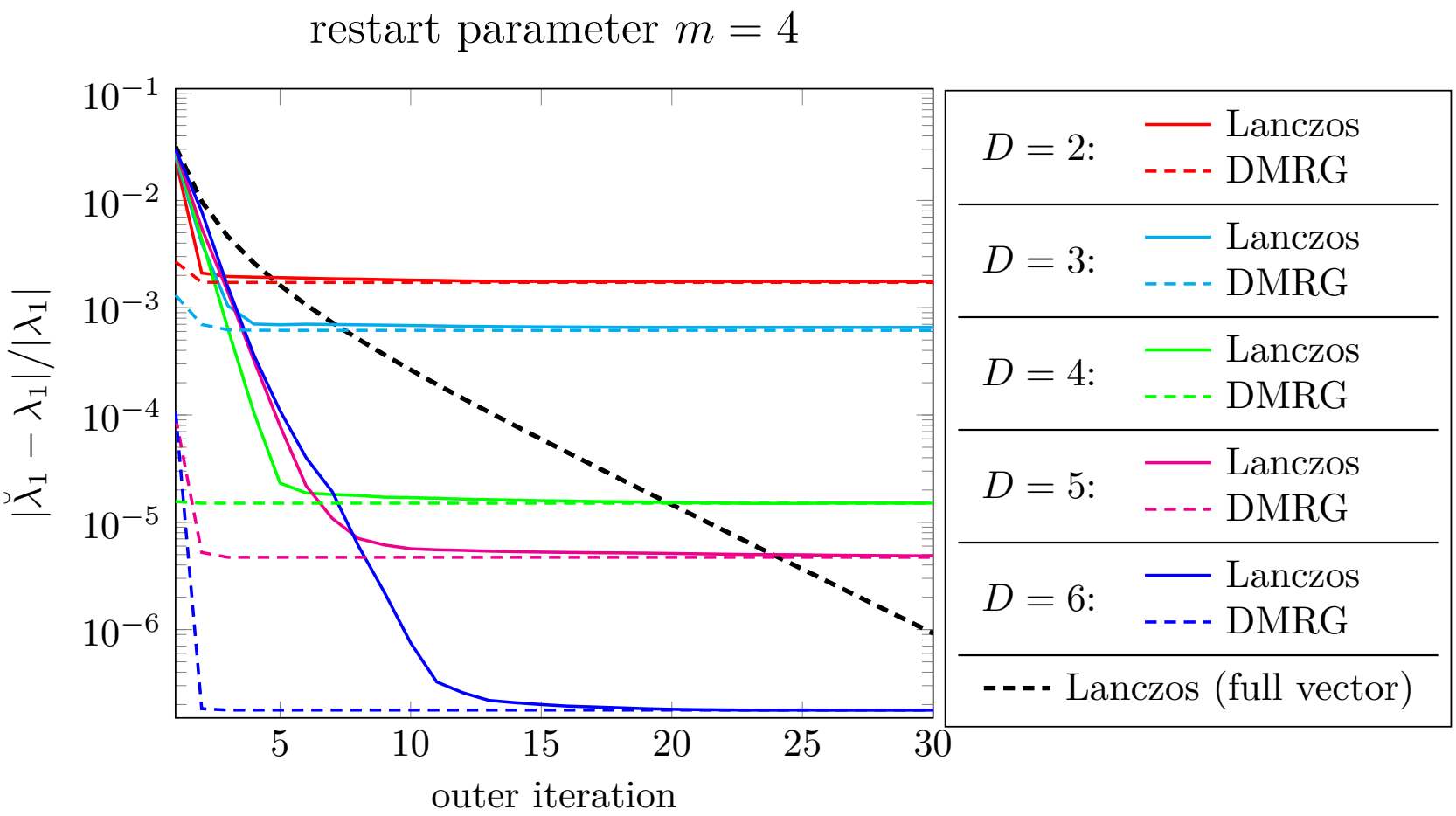
Algorithm: Restarted Lanczos iteration

Choose initial guess $\boldsymbol{r_0} \neq \boldsymbol{0}$;
for $iter = 1, 2, \ldots$ **do**
  Define $\beta_0 = \|\boldsymbol{r_0}\|$ and $\boldsymbol{q_0} = \boldsymbol{0}$;
  for $j = 1, 2, \ldots, m$ **do**
    $\boldsymbol{q_j} = \boldsymbol{r_{j-1}} / \beta_{j-1}$;
    $\boldsymbol{u_j} = \boldsymbol{H q_j}$;
    $\alpha_j = \boldsymbol{q_j}^{\mathrm{H}} \boldsymbol{u_j}$;
    $\boldsymbol{r_j} = \boldsymbol{u_j} - \alpha_j \boldsymbol{q_j} - \beta_{j-1} \boldsymbol{q_{j-1}}$;
    $\beta_j = \|\boldsymbol{r_j}\|$;
  $\boldsymbol{T_m} = \mathrm{tridiag}(\boldsymbol{\beta}, \boldsymbol{\alpha}, \boldsymbol{\beta})$;
  $(\lambda_{\min}, \boldsymbol{y}) = \mathrm{eig}(\boldsymbol{T_m})$;
  $\boldsymbol{v} = \boldsymbol{Q y} = (\boldsymbol{q_1}, \ldots, \boldsymbol{q_m}) \boldsymbol{y}$;
  $\boldsymbol{r_0} = \mathcal{P}_D(\boldsymbol{v})$

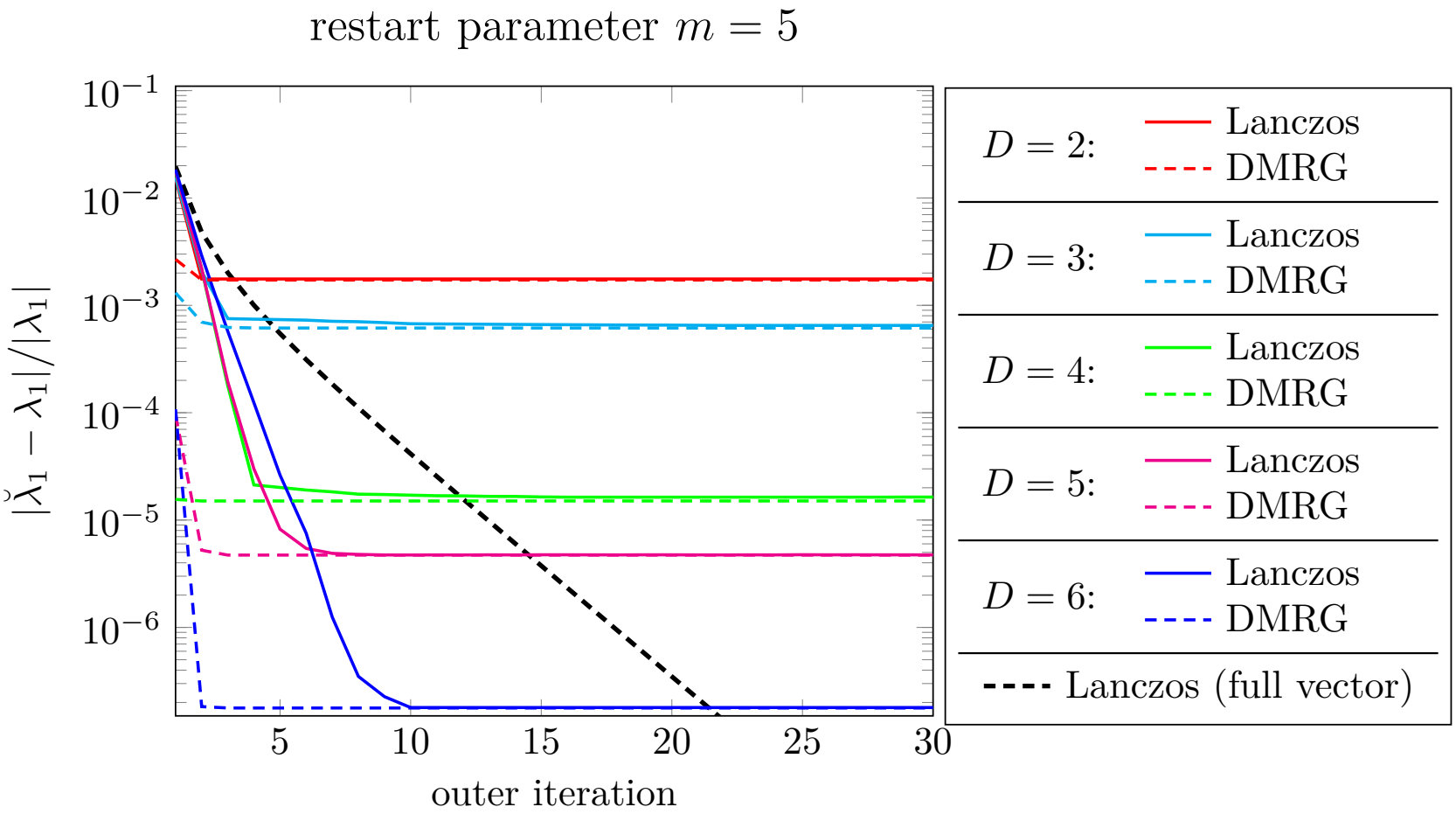
Numerical Results (Ising Hamiltonian $N = 25$)



Numerical Results (Ising Hamiltonian $N = 25$)



Numerical Results (Ising Hamiltonian $N = 25$)



Matrix and Vector Symmetries

- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is symmetric, if $a_{i,j} = a_{j,i}$
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is skew-symmetric, if $a_{i,j} = -a_{j,i}$
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is persymmetric, if $a_{i,j} = a_{n+1-j,n+1-i}$
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is skew-persymmetric, if $a_{i,j} = -a_{n+1-j,n+1-i}$
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is symmetric persymmetric, if $a_{i,j} = a_{j,i} = a_{n+1-i,n+1-j}$
- A vector $\boldsymbol{v} \in \mathbb{R}^n$ is called symmetric, if $v_i = v_{n+1-i}$.
- A vector $\boldsymbol{v} \in \mathbb{R}^n$ is called skew-symmetric, if $v_i = -v_{n+1-i}$.

$$\boldsymbol{A} = \begin{pmatrix} a & b & c & d \\ b & e & f & g \\ c & f & h & i \\ d & g & i & j \end{pmatrix}$$

Matrix and Vector Symmetries

- $A \in \mathbb{R}^{n \times n}$ is symmetric, if $a_{i,j} = a_{j,i}$
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is skew-symmetric, if $a_{i,j} = -a_{j,i}$
- $A \in \mathbb{R}^{n \times n}$ is persymmetric, if $a_{i,j} = a_{n+1-j,n+1-i}$
- $A \in \mathbb{R}^{n \times n}$ is skew-persymmetric, if $a_{i,j} = -a_{n+1-j,n+1-i}$
- $A \in \mathbb{R}^{n \times n}$ is symmetric persymmetric, if $a_{i,j} = a_{j,i} = a_{n+1-i,n+1-j}$
- A vector $v \in \mathbb{R}^n$ is called symmetric, if $v_i = v_{n+1-i}$.
- A vector $v \in \mathbb{R}^n$ is called skew-symmetric, if $v_i = -v_{n+1-i}$.

$$\boldsymbol{A} = \begin{pmatrix} 0 & b & c & d \\ -b & 0 & f & g \\ -c & -f & 0 & i \\ -d & -g & -i & 0 \end{pmatrix}$$

Matrix and Vector Symmetries

- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is symmetric, if $a_{i,j} = a_{j,i}$
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is skew-symmetric, if $a_{i,j} = -a_{j,i}$
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is persymmetric, if $a_{i,j} = a_{n+1-j,n+1-i}$
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is skew-persymmetric, if $a_{i,j} = -a_{n+1-j,n+1-i}$
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is symmetric persymmetric, if $a_{i,j} = a_{j,i} = a_{n+1-i,n+1-j}$
- A vector $\boldsymbol{v} \in \mathbb{R}^n$ is called symmetric, if $v_i = v_{n+1-i}$.
- A vector $\boldsymbol{v} \in \mathbb{R}^n$ is called skew-symmetric, if $v_i = -v_{n+1-i}$.

$$\boldsymbol{A} = \begin{pmatrix} a & b & c & d \\ e & f & g & c \\ h & i & f & b \\ j & h & e & a \end{pmatrix}$$

Matrix and Vector Symmetries

- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is symmetric, if $a_{i,j} = a_{j,i}$
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is skew-symmetric, if $a_{i,j} = -a_{j,i}$
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is persymmetric, if $a_{i,j} = a_{n+1-j,n+1-i}$
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is skew-persymmetric, if $a_{i,j} = -a_{n+1-j,n+1-i}$
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is symmetric persymmetric, if $a_{i,j} = a_{j,i} = a_{n+1-i,n+1-j}$
- A vector $\boldsymbol{v} \in \mathbb{R}^n$ is called symmetric, if $v_i = v_{n+1-i}$.
- A vector $\boldsymbol{v} \in \mathbb{R}^n$ is called skew-symmetric, if $v_i = -v_{n+1-i}$.

$$\boldsymbol{A} = \begin{pmatrix} a & b & c & 0 \\ e & f & 0 & -c \\ h & 0 & -f & -b \\ 0 & -h & -e & -a \end{pmatrix}$$

Matrix and Vector Symmetries

- $A \in \mathbb{R}^{n \times n}$ is symmetric, if $a_{i,j} = a_{j,i}$
- $A \in \mathbb{R}^{n \times n}$ is skew-symmetric, if $a_{i,j} = -a_{j,i}$
- $A \in \mathbb{R}^{n \times n}$ is persymmetric, if $a_{i,j} = a_{n+1-j,n+1-i}$
- $A \in \mathbb{R}^{n \times n}$ is skew-persymmetric, if $a_{i,j} = -a_{n+1-j,n+1-i}$
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is symmetric persymmetric, if
  $a_{i,j} = a_{j,i} = a_{n+1-i,n+1-j}$
- A vector $v \in \mathbb{R}^n$ is called symmetric, if $v_i = v_{n+1-i}$.
- A vector $v \in \mathbb{R}^n$ is called skew-symmetric, if $v_i = -v_{n+1-i}$.

$$\boldsymbol{A} = \begin{pmatrix} a & b & c & d \\ b & e & f & c \\ c & f & e & b \\ d & c & b & a \end{pmatrix}$$

Matrix and Vector Symmetries

- $A \in \mathbb{R}^{n \times n}$ is symmetric, if $a_{i,j} = a_{j,i}$
- $A \in \mathbb{R}^{n \times n}$ is skew-symmetric, if $a_{i,j} = -a_{j,i}$
- $A \in \mathbb{R}^{n \times n}$ is persymmetric, if $a_{i,j} = a_{n+1-j,n+1-i}$
- $A \in \mathbb{R}^{n \times n}$ is skew-persymmetric, if $a_{i,j} = -a_{n+1-j,n+1-i}$
- $A \in \mathbb{R}^{n \times n}$ is symmetric persymmetric, if $a_{i,j} = a_{j,i} = a_{n+1-i,n+1-j}$
- A vector $v \in \mathbb{R}^n$ is called symmetric, if $v_i = v_{n+1-i}$.
- A vector $v \in \mathbb{R}^n$ is called skew-symmetric, if $v_i = -v_{n+1-i}$.

$$\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{pmatrix} = \begin{pmatrix} v_n \\ v_{n-1} \\ \vdots \\ v_2 \\ v_1 \end{pmatrix} = \underbrace{\begin{pmatrix} & & & & 1 \\ & & & 1 & \\ & & \iddots & & \\ & 1 & & & \\ 1 & & & & \end{pmatrix}}_{=:J} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{pmatrix}$$

Matrix and Vector Symmetries

- $A \in \mathbb{R}^{n \times n}$ is symmetric, if $a_{i,j} = a_{j,i}$
- $A \in \mathbb{R}^{n \times n}$ is skew-symmetric, if $a_{i,j} = -a_{j,i}$
- $A \in \mathbb{R}^{n \times n}$ is persymmetric, if $a_{i,j} = a_{n+1-j,n+1-i}$
- $A \in \mathbb{R}^{n \times n}$ is skew-persymmetric, if $a_{i,j} = -a_{n+1-j,n+1-i}$
- $A \in \mathbb{R}^{n \times n}$ is symmetric persymmetric, if $a_{i,j} = a_{j,i} = a_{n+1-i,n+1-j}$
- A vector $v \in \mathbb{R}^n$ is called symmetric, if $v_i = v_{n+1-i}$.
- A vector $v \in \mathbb{R}^n$ is called skew-symmetric, if $v_i = -v_{n+1-i}$.

$$\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{pmatrix} = \begin{pmatrix} -v_n \\ -v_{n-1} \\ \vdots \\ -v_2 \\ -v_1 \end{pmatrix} = - \underbrace{\begin{pmatrix} & & & & 1 \\ & & & 1 & \\ & & \iddots & & \\ & 1 & & & \\ 1 & & & & \end{pmatrix}}_{=:J} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{pmatrix}$$

Symmetries of Spin-$\frac{1}{2}$ Hamiltonians

$$\boldsymbol{H} = \sum_{k=1}^{M} \alpha_k \boldsymbol{Q_1^{(k)}} \otimes \cdots \otimes \boldsymbol{Q_N^{(k)}} \ , \quad \boldsymbol{Q_i^{(k)}} \in \{\boldsymbol{I}, \boldsymbol{\sigma_x}, \boldsymbol{\sigma_y}, \boldsymbol{\sigma_z}\} \subset \mathbb{C}^{2\times 2}$$

$$\boldsymbol{\sigma_x} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \ \boldsymbol{\sigma_y} = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix}, \ \boldsymbol{\sigma_z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \ \boldsymbol{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} .$$

Symmetries of Spin-$\frac{1}{2}$ Hamiltonians

$$\boldsymbol{H} = \sum_{k=1}^{M} \alpha_k \boldsymbol{Q}_{\boldsymbol{1}}^{(\boldsymbol{k})} \otimes \cdots \otimes \boldsymbol{Q}_{\boldsymbol{N}}^{(\boldsymbol{k})} \ , \quad \boldsymbol{Q}_{\boldsymbol{i}}^{(\boldsymbol{k})} \in \{\boldsymbol{I}, \boldsymbol{\sigma_x}, \boldsymbol{\sigma_y}, \boldsymbol{\sigma_z}\} \subset \mathbb{C}^{2\times 2}$$

$$\boldsymbol{\sigma_x} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \ \boldsymbol{\sigma_y} = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix}, \ \boldsymbol{\sigma_z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \ \boldsymbol{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} .$$

- $\boldsymbol{\sigma_x}$, $\boldsymbol{\sigma_z}$ and $\boldsymbol{I}$ are symmetric, $\boldsymbol{\sigma_y}/\mathrm{i}$ is skew-symmetric,

Symmetries of Spin-$\frac{1}{2}$ Hamiltonians

$$\boldsymbol{H} = \sum_{k=1}^{M} \alpha_k \boldsymbol{Q}_{\boldsymbol{1}}^{(\boldsymbol{k})} \otimes \cdots \otimes \boldsymbol{Q}_{\boldsymbol{N}}^{(\boldsymbol{k})} \ , \quad \boldsymbol{Q}_{\boldsymbol{i}}^{(\boldsymbol{k})} \in \{\boldsymbol{I}, \boldsymbol{\sigma_x}, \boldsymbol{\sigma_y}, \boldsymbol{\sigma_z}\} \subset \mathbb{C}^{2\times 2}$$

$$\boldsymbol{\sigma_x} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \ \boldsymbol{\sigma_y} = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix}, \ \boldsymbol{\sigma_z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \ \boldsymbol{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} .$$

- $\boldsymbol{\sigma_x}$, $\boldsymbol{\sigma_z}$ and $\boldsymbol{I}$ are symmetric, $\boldsymbol{\sigma_y}/\mathrm{i}$ is skew-symmetric,
- $\boldsymbol{\sigma_x}$, $\boldsymbol{\sigma_y}/\mathrm{i}$ and $\boldsymbol{I}$ are persymmetric, $\boldsymbol{\sigma_z}$ is skew-persymmetric,

Symmetries of Spin-$\frac{1}{2}$ Hamiltonians

$$\boldsymbol{H} = \sum_{k=1}^{M} \alpha_k \boldsymbol{Q}_{\boldsymbol{1}}^{(\boldsymbol{k})} \otimes \cdots \otimes \boldsymbol{Q}_{\boldsymbol{N}}^{(\boldsymbol{k})} \ , \quad \boldsymbol{Q}_{\boldsymbol{i}}^{(\boldsymbol{k})} \in \{\boldsymbol{I}, \boldsymbol{\sigma_x}, \boldsymbol{\sigma_y}, \boldsymbol{\sigma_z}\} \subset \mathbb{C}^{2\times 2}$$

$$\boldsymbol{\sigma_x} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \ \boldsymbol{\sigma_y} = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix}, \ \boldsymbol{\sigma_z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \ \boldsymbol{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} .$$

- $\boldsymbol{\sigma_x}$, $\boldsymbol{\sigma_z}$ and $\boldsymbol{I}$ are symmetric, $\boldsymbol{\sigma_y}/\mathrm{i}$ is skew-symmetric,
- $\boldsymbol{\sigma_x}$, $\boldsymbol{\sigma_y}/\mathrm{i}$ and $\boldsymbol{I}$ are persymmetric, $\boldsymbol{\sigma_z}$ is skew-persymmetric,
- Kronecker product of two skew-symmetric matrices is symmetric,
- Kronecker product of two skew-persymmetric matrices is persymmetric.

Symmetries of Spin-$\frac{1}{2}$ Hamiltonians

$$\boldsymbol{H} = \sum_{k=1}^{M} \alpha_k \boldsymbol{Q}_1^{(k)} \otimes \cdots \otimes \boldsymbol{Q}_N^{(k)} \ , \quad \boldsymbol{Q}_i^{(k)} \in \{\boldsymbol{I}, \boldsymbol{\sigma_x}, \boldsymbol{\sigma_y}, \boldsymbol{\sigma_z}\} \subset \mathbb{C}^{2\times 2}$$

$$\boldsymbol{\sigma_x} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \ \boldsymbol{\sigma_y} = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix}, \ \boldsymbol{\sigma_z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \ \boldsymbol{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} .$$

- $\boldsymbol{\sigma_x}$, $\boldsymbol{\sigma_z}$ and $\boldsymbol{I}$ are symmetric, $\boldsymbol{\sigma_y}/\mathrm{i}$ is skew-symmetric,
- $\boldsymbol{\sigma_x}$, $\boldsymbol{\sigma_y}/\mathrm{i}$ and $\boldsymbol{I}$ are persymmetric, $\boldsymbol{\sigma_z}$ is skew-persymmetric,
- Kronecker product of two skew-symmetric matrices is symmetric,
- Kronecker product of two skew-persymmetric matrices is persymmetric.

Symmetry of Spin Hamiltonians

Hamiltonians of typical physical models (Ising-ZZ, various Heisenberg models) are real-valued and symmetric persymmetric.

Matrix Symmetries and Eigenvectors

Eigenvectors of Symmetric Persymmetric Matrices ([CB76])

A symmetric persymmetric matrix $\boldsymbol{A}$ can be orthogonally transformed to a block diagonal matrix:

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{C}^{\mathrm{T}} & \boldsymbol{JBJ} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} \boldsymbol{I} & \boldsymbol{I} \\ \boldsymbol{J} & -\boldsymbol{J} \end{pmatrix} \begin{pmatrix} \boldsymbol{B}+\boldsymbol{JC} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B}-\boldsymbol{JC} \end{pmatrix} \begin{pmatrix} \boldsymbol{I} & \boldsymbol{J} \\ \boldsymbol{I} & -\boldsymbol{J} \end{pmatrix} .$$

Hence, the eigenvectors of $\boldsymbol{A}$ are of the form

$$\begin{pmatrix} \boldsymbol{v_i} \\ \boldsymbol{Jv_i} \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} \boldsymbol{v_i} \\ -\boldsymbol{Jv_i} \end{pmatrix} .$$

Matrix Symmetries and Eigenvectors

Eigenvectors of Symmetric Persymmetric Matrices ([CB76])

A symmetric persymmetric matrix A can be orthogonally transformed to a block diagonal matrix:

$$A = \begin{pmatrix} B & C \\ C^{\mathrm{T}} & JBJ \end{pmatrix} = \frac{1}{2} \begin{pmatrix} I & I \\ J & -J \end{pmatrix} \begin{pmatrix} B + JC & 0 \\ 0 & B - JC \end{pmatrix} \begin{pmatrix} I & J \\ I & -J \end{pmatrix} .$$

Hence, the eigenvectors of A are of the form

$$\begin{pmatrix} v_i \\ Jv_i \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} v_i \\ -Jv_i \end{pmatrix} .$$

Ground States of Spin Hamiltonians

Ground states of typical physical models (Ising-ZZ, various Heisenberg models) are either symmetric or skew-symmetric.

MPS Representation of Symmetric Vectors

$$x_{i_1,i_2,i_3,\ldots,i_{N-1},i_N} = x_{\bar{i}_1,\bar{i}_2,\bar{i}_3,\ldots,\bar{i}_{N-1},\bar{i}_N} \iff \boldsymbol{J}\boldsymbol{x} = \boldsymbol{x} \qquad \text{(BF)}$$

Flip-operator $\bar{i} := 1 - i$.

Example (N=3): $\boldsymbol{x} = \begin{pmatrix} a & b & c & d & d & c & b & a \end{pmatrix}^{\mathrm{T}}$

Theorem: MPS Representation of the Vector Symmetry ([Huckle13])

1. If all MPS matrix pairs $(\boldsymbol{A}_j^{(0)}, \boldsymbol{A}_j^{(1)})$ are connected via

$$\boldsymbol{A}_j^{(i_j)} = \boldsymbol{U}_j \boldsymbol{A}_j^{(\bar{i}_j)} \boldsymbol{U}_{j+1} \ , \ j = 1, \ldots, N \ , \qquad (1)$$

   with involutions $\boldsymbol{U}_j$ ($\boldsymbol{U}_j^2 = \boldsymbol{I}$), the represented vector is symmetric (BF).
2. Any symmetric vector (BF) can be represented by an MPS fulfilling relations (1).

- Reduction factor: ≈ 2

MPS Representation of Symmetric Vectors

Theorem: Uniqueness Result for Symmetric Vectors ([Huckle13])

Assume that the MPS matrices (over $\mathbb{K}$) are related by

$$\boldsymbol{A_j^{(1)}} = \boldsymbol{U_{j-1}} \boldsymbol{A_j^{(0)}} \boldsymbol{V_j} \ ,$$

with square matrices $\boldsymbol{V_j}$ and $\boldsymbol{U_j}$ of appropriate size ($j = 1, \ldots, N$). If any choice of matrices $\boldsymbol{A_j^{(0)}}$ results in the symmetry of the represented vector $\boldsymbol{x}$, i.e.,

$$\boldsymbol{Jx} \ = \ \boldsymbol{x} \ ,$$

then $\boldsymbol{U_j}$ and $\boldsymbol{V_j}$ are—up to a scalar factor—involutions for all j: $\boldsymbol{U_j^2} = u_j \boldsymbol{I}$, $\boldsymbol{V_j^2} = v_j \boldsymbol{I}$. Furthermore, $\boldsymbol{U_j} = c_j \cdot \boldsymbol{V_j}$, $j = 1, ..., N$ with constants c_j.

- This theorem gives an argument to use involutions as heuristic.
- An involution has as eigenvalues $\pm 1 \Longrightarrow$ choose $\boldsymbol{U_j} = \boldsymbol{J}$.

Exploiting Symmetries in Computations

- Impose symmetry conditions into variational ground-state search
- Suppose that the desired ground-state vector x is symmetric, i.e., $Jx = x$

$$0 \leq x^{\mathrm{H}} (I - J) x = x^{\mathrm{H}} \underbrace{(x - Jx)}_{=0} = 0 .$$

- Therefore,

$$\lambda_{\min} \leq \frac{y^{\mathrm{H}} H y}{y^{\mathrm{H}} y} \leq \frac{y^{\mathrm{H}} H y}{y^{\mathrm{H}} y} + \rho \frac{y^{\mathrm{H}} (I - J) y}{y^{\mathrm{H}} y} = \frac{y^{\mathrm{H}} (H + \rho(I - J)) y}{y^{\mathrm{H}} y} .$$

- Apply ground-state algorithm (e.g., DMRG) to modified Hamiltonian

$$H(\rho) := H + \rho(I - J) \quad \text{for some} \quad \rho > 0 .$$

Numerical Results: Number of Iterations

Table : Heisenberg-XXX model of $N = 100$ spins: Number of DMRG sweeps until convergence.

	D								
ρ	2	3	4	5	6	7	8	9	10
0	66	71	28	13	16	35	13	11	22
0.1	48	24	22	8	12	43	9	8	27
0.5	13	10	14	6	9	31	9	8	17
1	8	10	15	5	8	20	9	8	15
2	6	14	16	5	8	20	9	8	14
10	4	6	46	5	10	9	9	8	9
100	5	5	29	10	9	9	9	9	10

Numerical Studies: Convergence

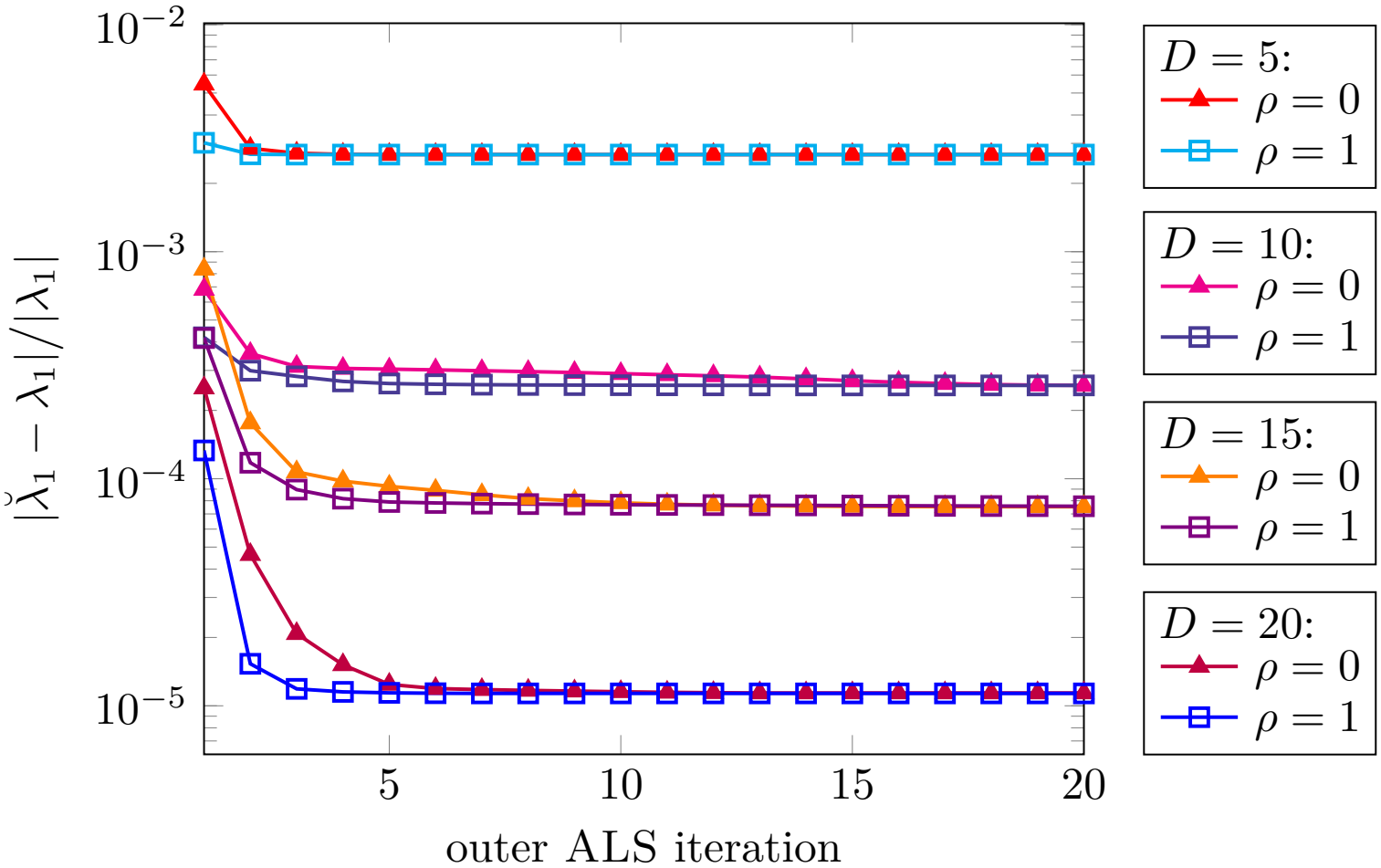


Figure : Heisenberg-XXX model of $N = 100$ spins: Approximation error.

Conclusions

- Simulation of quantum systems relies on standard problems of numerical linear algebra
- Hilbert space grows exponentially in the physical system size

Conclusions

- Simulation of quantum systems relies on standard problems of numerical linear algebra
- Hilbert space grows exponentially in the physical system size
- “Non-generic” behavior of quantum systems
- Nature of the underlying system translates into structured objects of linear and multilinear algebra (e.g., symmetries)
- Numerical treatment requires efficient algorithms acting on problem-adapted data structures
  - NMR computation: problem reduction and algorithmic improvement
  - Projection on MPS manifold may improve the solution
  - Symmetry-adapted ground-state search improves variational MPS methods

Conclusions

- Simulation of quantum systems relies on standard problems of numerical linear algebra
- Hilbert space grows exponentially in the physical system size
- “Non-generic” behavior of quantum systems
- Nature of the underlying system translates into structured objects of linear and multilinear algebra (e.g., symmetries)
- Numerical treatment requires efficient algorithms acting on problem-adapted data structures
  - NMR computation: problem reduction and algorithmic improvement
  - Projection on MPS manifold may improve the solution
  - Symmetry-adapted ground-state search improves variational MPS methods

Conclusions

- Simulation of quantum systems relies on standard problems of numerical linear algebra
- Hilbert space grows exponentially in the physical system size
- “Non-generic” behavior of quantum systems
- Nature of the underlying system translates into structured objects of linear and multilinear algebra (e.g., symmetries)
- Numerical treatment requires efficient algorithms acting on problem-adapted data structures
  - NMR computation: problem reduction and algorithmic improvement
  - Projection on MPS manifold may improve the solution
  - Symmetry-adapted ground-state search improves variational MPS methods

Thank you for your attention.


T. Auckenthaler, M. Bader, T. Huckle, A. Spörl, and K. Waldherr
Matrix exponentials and parallel prefix computation in a quantum control problem.
Parallel Comput., 2010.


T. Huckle, and K. Waldherr
Subspace iteration methods in terms of matrix product states.
Proc. Appl. Math. Mech., 2012.


T. Huckle, K. Waldherr, and T. Schulte-Herbrüggen
Computations in quantum tensor networks.
Lin. Alg. Appl., 2013.


T. Huckle, K. Waldherr, and T. Schulte-Herbrüggen
Exploiting matrix symmetries and physical symmetries in matrix product states and tensor trains.
Linear Multilinear A., 2013.