Numerical Linear Algebra Tasks in a Quantum Control Problem

Konrad Waldherr
Technische Universität München, Germany
Overview

- Gradient Flow Algorithm
- The structure of the matrices
- Methods of computing the exponential of a matrix
  - Taylor series
  - Padé approximation
  - eigendecomposition
  - Chebyshev series expansion
- Computing the intermediate products
  - slice-wise approach
  - tree-like approach
Gradient Flow Algorithm

One iteration step in the Gradient Flow Algorithm

- Calculate the forward-propagation for all $t_1, t_2, \ldots, t_k$:

  $$U(t_k) = e^{-i\Delta t H_k} \cdot e^{-i\Delta t H_{k-1}} \ldots e^{-i\Delta t H_1}$$

- Compute the backward-propagation for all $t_M, t_{M-1}, \ldots, t_k$

  $$\Lambda(t_k) = e^{-i\Delta t H_k} \cdot e^{-i\Delta t H_{k+1}} \ldots e^{-i\Delta t H_M}$$

- Calculate the update

  $$\frac{\partial h(U(t_k))}{\partial u_j} = \text{Re} \left\{ \text{tr} \left[ \Lambda^\dagger(t_k) (-iH_j) U(t_k) \right] \right\}$$
Numerical tasks

- Computation of the matrix exponentials

\[ U_k := e^{-i\Delta t H_k} \]

- Computation of all intermediate products

\[ U_0 \]
\[ U_0 \cdot U_1 \]
\[ U_0 \cdot U_1 \cdot U_2 \]
\[ \vdots \]
\[ U_0 \cdot U_1 \cdot U_2 \cdots U_M \]
Structure of the matrices $H$

- Pauli matrices

$$P_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad P_y = \begin{pmatrix} 0 & i \\ -i & 0 \end{pmatrix}, \quad P_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

- $H = H_d + H_c$, where

$$H_d = \sum_{k_1, k_2 = 0}^{q-1} \gamma_{k_1, k_2} I_{2^{k_1}} \otimes P_z \otimes I_{2^{k_2}} \otimes P_z \otimes I_{2^{q-k_1-k_2-2}}$$

$$H_c = \sum_{k=0}^{q-1} I_{2^k} \otimes (\alpha_k P_x + \beta_k P_y) \otimes I_{2^{q-k-1}}$$
Properties of $H$

- $H$ is sparse, hermitian and persymmetric
- $H$ has the following sparsity pattern

$$
\begin{pmatrix}
I & J \\
I & -J
\end{pmatrix}
H
\begin{pmatrix}
I & I \\
J & -J
\end{pmatrix}
= 
\begin{pmatrix}
A_1 & 0 \\
0 & A_2
\end{pmatrix}
$$
The exponential of a matrix

- **Definition:** The exponential of a matrix \( A \in \mathbb{C}^{n \times n} \) is defined by the infinite Taylor series

\[
e^A := \sum_{k=0}^{\infty} \frac{A^k}{k!}
\]

- **Method 1:** Computation of the matrix exponential by using a partial sum of the Taylor series

\[
e^A \approx S_m(A) := \sum_{k=0}^{m} \frac{A^k}{k!}
\]

- **Error Estimate:**

\[
\| e^A - S_m(A) \| \leq \left( \frac{\| A \|^{m+1}}{(m+1)!} \right) \left( \frac{1}{1 - \| A \| / (m + 2)} \right) \leq \delta
\]
Properties of the matrix exponential

- The functional equation $e^{x+y} = e^x e^y$ does in general not hold for matrices:
  $$e^{A+B} = e^A \cdot e^B \iff A \cdot B = B \cdot A$$

  "The great matrix exponential tragedy"
  (Moler, van Loan, 1978)

- **Trotter Product Formula:**
  $$\lim_{m \to \infty} \left( e^{A/m} e^{B/m} \right)^m = e^{A+B}$$

- $$\left( e^{A/m} \right)^m = e^A$$

- **Scaling & Squaring**
  $$e^A = \left( e^{A/2^k} \right)^{2^k}$$
Computation of the matrix exponential: eigendecomposition

- In the case of a diagonal matrix

\[ A = \text{diag}(d_1, \ldots, d_n) = \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix} \]

it holds

\[ e^A = \text{diag}(e^{d_1}, \ldots, e^{d_n}) = \begin{pmatrix} e^{d_1} \\ \vdots \\ e^{d_n} \end{pmatrix} \]

- If \( A = SDS^{-1} = S (\text{diag}(d_1, \ldots, d_n)) S^{-1} \) it follows

\[ e^A = S \left( \text{diag}(e^{d_1}, \ldots, e^{d_n}) \right) S^{-1} \]

- Expensive part: Computation of the eigendecomposition
Computation of the matrix exponential: Padé approximation

- For $x \in \mathbb{C}$ the Padé approximation $r_m(x)$ of $e^x$ is given by

$$r_m(x) = \frac{p_m(x)}{q_m(x)}$$

- $p_m(x) = \sum_{j=0}^{m} \frac{(2m-j)!m!x^j}{(2m)!(m-j)!j!}$

- $q_m(x) = \sum_{j=0}^{m} \frac{(2m-j)!m!(-x)^j}{(2m)!(m-j)!j!}$

- Generalization to matrices:

$$e^A \approx r_m(A) = (q_m(A))^{-1} p_m(A)$$

- Combination with Scaling & Squaring

- Expensive part: Computation of the matrix inverse
Computation of the matrix exponential: Chebyshev series expansion

- For $|x| \leq 1$ we have

$$e^x = J_0(i) + 2 \sum_{k=1}^{\infty} i^k J_k(-i) T_k(x)$$

- $J_k$: Bessel function
- $T_k$: Chebyshev polynomial

- **Generalization to matrices:** For $\|A\| \leq 1$

$$e^A = J_0(i) + 2 \sum_{k=1}^{\infty} i^k J_k(-i) T_k(A)$$

- In the case of $A$ with arbitrary norm: Scaling & Squaring technique
Comparison of the methods: Computation time

- Chebyshev method
- Padé method
- Eigendecomposition method

Number of spins

Computation Time
Comparison of the methods: accuracy
Advantages of the Chebyshev series method

- Only the evaluation of a matrix polynomial required: 
  \[ p(A) = \alpha_6 A^6 + \alpha_5 A^5 + \alpha_4 A^4 + \alpha_3 A^3 + \alpha_2 A^2 + \alpha_1 A + \alpha_0 I \]

- In the case of sparse matrices: 
  Only products of the form dense * sparse appear

- Good convergence properties

- Matrix polynomials can be evaluated very efficiently:

  - Horner scheme requires 5 non-scalar multiplications:
    \[ ((((((\alpha_6 A + \alpha_5 I) \cdot A + \alpha_4 I) \cdot A + \alpha_3 I) \cdot A + \alpha_2 I) \cdot A + \alpha_1 I) \cdot A + \alpha_0 I) \]

- Optimal method: (Only 3 matrix-matrix-products required)

  \[ A_2 = A \cdot A, \quad A_3 = A_2 \cdot A \]
  \[ p(A) = \alpha_0 I + \alpha_1 A + \alpha_2 A_2 + \alpha_3 A_3 + A_3 \cdot (\alpha_4 A + \alpha_5 A_2 + \alpha_6 A_3) \]
Parallel matrix-matrix-multiplication

• Numerical task: Compute all intermediate products

\[ U_0 \]
\[ U_0 \cdot U_1 \]
\[ U_0 \cdot U_1 \cdot U_2 \]
\[ \vdots \]
\[ U_0 \cdot U_1 \cdot U_2 \cdots U_M \]

• Two approaches for a parallel algorithm:
  • slice-wise method
  • tree-like method
The slice-wise propagation

- Broadcast of all matrices $U_k$ to all processors
- Each processor is responsible for "its" rows

```
  | U1    | U2    | ...
  | P0    | P0    | P0
  | P1    | P1    | P1
  | P2    | P2    | P2
  | P3    | P3    | P3
  | U0    | U01   | U02
```
The tree-like propagation: Parallel Prefix Algorithm

- The matrices are distributed to the processors
- Communications steps between the levels
Final conclusions

- Chebyshev method very efficient (accuracy and computation time)
- Parallel computation of the matrix exponentials (matrix-multiplications in parallel)?
- Shared memory method to avoid communication overhead?
- Fast matrix-matrix-multiplication (Strassen, Vinograd)?
- ...