JASS 2008
St. Petersburg, March 2008

A quantum control algorithm: numerical aspects

Philipp Klenze
Technische Universität München, Germany

This presentation contains slides adapted from the presentation “Numerical Linear Algebra Tasks in a Quantum Control Problem” by Konrad Waldherr.
Gradient Flow Algorithm

One iteration step in the Gradient Flow Algorithm

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

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

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

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

- Calculate the update

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

- Computation of the matrix exponentials
- Computation of all intermediate products

\[ U_0 \]
\[ U_0 \cdot U_1 \]
\[ U_0 \cdot U_1 \cdot U_2 \]
\[ \vdots \]
\[ U_0 \cdot U_1 \cdot U_2 \cdots U_M \]
Properties of $H$

- $H$ is *sparse*, most entries are zero
- $H$ is *hermitian*, $H^\dagger = H$
- $H$ is *persymmetric*, symmetric with respect to the north-west-south-east diagonal, $HJ = JH^\dagger$
- $H$ has the following sparsity pattern
Simplifying the problem

- \( H \) can be transformed into a real matrix
- Then, \( H \) can be transformed to two real blocks of half size:

\[
\begin{pmatrix}
I & J \\
I & -J
\end{pmatrix} \cdot H \cdot 
\begin{pmatrix}
I & I \\
J & -J
\end{pmatrix} = 
\begin{pmatrix}
A_1 & 0 \\
0 & A_2
\end{pmatrix}
\]
Ideas to calculate the matrix exponential

- The problem: compute $e^A := \sum_{k=0}^{\infty} \frac{A^k}{k!}$
Ideas to calculate the matrix exponential

- The problem: compute $e^A := \sum_{k=0}^{\infty} \frac{A^k}{k!}$
- Diagonalise $A$ (eigendecomposition)
Ideas to calculate the matrix exponential

- The problem: compute $e^A := \sum_{k=0}^{\infty} \frac{A^k}{k!}$
- Diagonalise $A$ (eigendecomposition)
- Approximate the exponential function
Ideas to calculate the matrix exponential

• The problem: compute $e^A := \sum_{k=0}^{\infty} \frac{A^k}{k!}$

• Diagonalise $A$ (eigendecomposition)

• Approximate the exponential function
  • with polynomials
Idea to calculate the matrix exponential

- The problem: compute $e^A := \sum_{k=0}^{\infty} \frac{A^k}{k!}$
- Diagonalise $A$ (eigendecomposition)
- Approximate the exponential function
  - with polynomials
    - Taylor series
Ideas to calculate the matrix exponential

- The problem: compute $e^A := \sum_{k=0}^{\infty} \frac{A^k}{k!}$
- Diagonalise $A$ (eigendecomposition)
- Approximate the exponential function
  - with polynomials
    - TAYLOR series
    - CHEBYSHEV series expansion
Ideas to calculate the matrix exponential

- The problem: compute $e^A := \sum_{k=0}^{\infty} \frac{A^k}{k!}$

- Diagonalise $A$ (eigendecomposition)

- Approximate the exponential function
  - with polynomials
    - TAYLOR series
    - CHEBYSHEV series expansion
  - with rational functions
    - PADÉ approximation
Eigendecomposition

- In the case of a diagonal matrix

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

it holds

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

- 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
Scaling and Squaring

- Some approximations work much better if the norm of the matrix $A$ is not too big

$$\left(\frac{e^A}{m}\right)^m = e^A$$
Scaling and Squaring

- Some approximations work much better if the norm of the matrix $A$ is not too big
- $\left(\frac{e^A}{m}\right)^m = e^A$

Scaling & Squaring

$$e^A = \left(\frac{e^A}{2^k}\right)^{2k}$$

- We scale our matrix by a factor of $\frac{1}{2^k}$
- Then, we compute the approximation
- In the end, we square the approximation $k$ times
- This is not very expensive
- Additional error
Taylor series

- Idea: use a partial sum of the Taylor series

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

- Idea: use a partial sum of the Taylor series

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

- The error estimate depends on the norm of \( A \)
  \[ \text{Scaling & Squaring} \]
Taylor series

• Idea: use a partial sum of the Taylor series

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

• The error estimate depends on the norm of \( A \)
  → Scaling & Squaring

• Convergence is slow
Taylor series

• Idea: use a partial sum of the Taylor series

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

• The error estimate depends on the norm of \( A \)
  → Scaling & Squaring

• Convergence is slow

• Not numerically stable
Chebyshev series expansion

- A well-behaved function $f : [-1, 1] \rightarrow \mathbb{C}$ can be approximated by Chebychev polynomials $T_k(x)$:

$$f(x) \approx a_0 \frac{2}{2} + \sum_{k=1}^{m} a_k T_k(x)$$

with

$$a_k := \frac{2}{\pi} \int_{-1}^{1} f(x) T_k(x) \frac{dx}{\sqrt{1-x^2}}$$
Chebyshev series expansion

- A well-behaved function $f : [−1,1] \rightarrow \mathbb{C}$ can be approximated by Chebychev polynomials $T_k(x)$:

$$f(x) \approx \frac{a_0}{2} + \sum_{k=1}^{m} a_k T_k(x)$$

with

$$a_k := \frac{2}{\pi} \int_{-1}^{1} f(x) T_k(x) \frac{dx}{\sqrt{1 - x^2}}$$

- For the exponential function, $a_k$ decreases as $\frac{1}{2^k k!}$
Chebyshev series expansion

- A well-behaved function $f : [-1, 1] \rightarrow \mathbb{C}$ can be approximated by Chebychev polynomials $T_k(x)$:

$$f(x) \approx \frac{a_0}{2} + \sum_{k=1}^{m} a_k T_k(x)$$

with

$$a_k := \frac{2}{\pi} \int_{-1}^{1} f(x) T_k(x) \frac{dx}{\sqrt{1-x^2}}$$

- For the exponential function, $a_k$ decreases as $\frac{1}{2^k k!}$
- This works also for matrices if the norm is smaller than one
Chebyshev series expansion

• A well-behaved function $f : [-1, 1] \rightarrow \mathbb{C}$ can be approximated by Chebychev polynomials $T_k(x)$:

$$f(x) \approx \frac{a_0}{2} + \sum_{k=1}^{m} a_k T_k(x)$$

with

$$a_k := \frac{2}{\pi} \int_{-1}^{1} f(x) T_k(x) \frac{dx}{\sqrt{1 - x^2}}$$

• For the exponential function, $a_k$ decreases as $\frac{1}{2^k k!}$

• This works also for matrices if the norm is smaller than one

• Arbitrary norm
  → Scaling & Squaring
Padé approximation

- Padé approximation works like Taylor, but using a rational function instead of a polynomial

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

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

with $p_m(x) = \sum_{j=0}^{m} \frac{(2m-j)!m!}{(2m)!((m-j))!j!} x^j$, $q_m(x) = \sum_{j=0}^{m} \frac{(2m-j)!m!(-1)^j}{(2m)!((m-j))!j!} x^j$
Padé approximation

- Padé approximation works like Taylor, but using a rational function instead of a polynomial
- For $x \in \mathbb{C}$ the Padé approximation $r_m(x)$ of $e^x$ is given by
  
  $$r_m(x) = \frac{p_m(x)}{q_m(x)}$$

  with
  
  $$p_m(x) = \sum_{j=0}^{m} \frac{(2m-j)!m!}{(2m)!(m-j)!j!} x^j, \quad q_m(x) = \sum_{j=0}^{m} \frac{(2m-j)!m!(-1)^j}{(2m)!(m-j)!j!} x^j$$

- Generalization to matrices:
  
  $$e^A \approx r_m(A) = \left(q_m(A)\right)^{-1} p_m(A)$$

- Approximation is only good around 0
  \[\rightarrow\] Scaling & Squaring
Padé approximation

- Padé approximation works like Taylor, but using a rational function instead of a polynomial.
- For $x \in \mathbb{C}$ the Padé approximation $r_m(x)$ of $e^x$ is given by

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

with

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

$$q_m(x) = \sum_{j=0}^{m} \frac{(2m-j)!m!(-1)^j}{(2m)!(m-j)!j!} x^j$$

- Generalization to matrices:

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

- Approximation is only good around 0
  → Scaling & Squaring

- Expensive part: Computation of the matrix inverse
Comparison of the methods: Computation time

![Graph showing comparison of computation time for different methods.](image-url)
Comparison of the methods: accuracy

![Graph comparing the accuracy of Chebyshev, Padé, and Eigendecomposition methods](image)
Advantages of the Chebyshev series method

- Only the evaluation of a matrix polynomial required: \( \Rightarrow \) BLAS-Routines
- Only products of the form dense \(*\) sparse appear
- Good convergence properties
- Matrix polynomials of order \( k \) can be evaluated with only \( O(\sqrt{k}) \) matrix-matrix-products
- Theoretically nice approach
Parallel matrix-matrix-multiplication

- Numerical task: Compute all intermediate products

\[
\begin{align*}
U_0 \\
U_0 \cdot U_1 \\
U_0 \cdot U_1 \cdot U_2 \\
\vdots \\
U_0 \cdot U_1 \cdot U_2 \cdots U_M
\end{align*}
\]

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

Part 3

CPU0 CPU1 CPU2 CPU3

M1 := ?
M2 := ?
M1 := ?

U0 := ?
U1 := ?

M1 ... M2

U0

U1

U2

t
The slice-wise approach: conclusions

- Broadcast of all matrices $U_k$ to all processors
- Each processor is responsible for "its" rows
- No communication during the algorithm required
- Optimal in terms of scalar multiplications
- Broadcasting costs most of the time
- Memory becomes an issue
The tree-like approach

Please look at the whiteboard.
The tree-like approach: conclusions

- “Expanded” binary tree
- Complicated algorithm
- No broadcasting required
- Still much communication
- More multiplications than strictly needed
- “Super Nodes” do quasi-broadcast
a new approach

Please look at the whiteboard.
The best of both worlds: a new approach

- Pipeline
- Little communication required
- Simple algorithm
- Optimal in terms of total scalar multiplications
- Possibility to parallelize the entire GRAPE algorithm
- Some idle time until the pipeline is filled