BGCE Research Day

Garching, January 10th, 2008

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 $t_1, t_2, ..., t_k$:

$$\mathbf{U}(t_k) = e^{-i\Delta t \mathbf{H}_k} \cdot e^{-i\Delta t \mathbf{H}_{k-1}} \dots e^{-i\Delta t \mathbf{H}_1}$$

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

$$\mathbf{\Lambda}(t_k) = e^{-i\Delta t \mathbf{H}_k} \cdot e^{-i\Delta t \mathbf{H}_{k+1}} \dots e^{-i\Delta t \mathbf{H}_M}$$

- Calculate the update

$$\frac{\partial h(\mathbf{U}(t_k))}{\partial u_j} = \mathrm{Re}\left\{\mathrm{tr}\left[\mathbf{\Lambda}^\dagger(t_k)(-i\mathbf{H}_j)\mathbf{U}(t_k)\right]\right\}$$

Numerical tasks

- Computation of the matrix exponentials

$$\mathbf{U}_k := e^{-i\Delta t \mathbf{H}_k}$$

- Computation of all intermediate products

$$\begin{array}{l} \mathbf{U}_0 \\ \mathbf{U}_0 \cdot \mathbf{U}_1 \\ \mathbf{U}_0 \cdot \mathbf{U}_1 \cdot \mathbf{U}_2 \\ \vdots \\ \mathbf{U}_0 \cdot \mathbf{U}_1 \cdot \mathbf{U}_2 \cdots \mathbf{U}_M \end{array}$$

Structure of the matrices $\mathbf{H}$

- Pauli matrices

$$\mathbf{P}_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \mathbf{P}_y = \begin{pmatrix} 0 & i \\ -i & 0 \end{pmatrix}, \quad \mathbf{P}_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

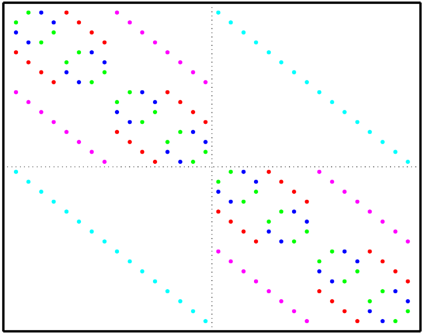
- $\mathbf{H} = \mathbf{H}_d + \mathbf{H}_c$, where

$$\mathbf{H}_d = \sum_{k_1,k_2=0}^{q-1} \gamma_{k_1,k_2} \mathbf{I}_{2^{k_1}} \otimes \mathbf{P}_z \otimes \mathbf{I}_{2^{k_2}} \otimes \mathbf{P}_z \otimes \mathbf{I}_{2^{q-k_1-k_2-2}}$$

$$\mathbf{H}_c = \sum_{k=0}^{q-1} \mathbf{I}_{2^k} \otimes \left(\alpha_k \mathbf{P}_x + \beta_k \mathbf{P}_y\right) \otimes \mathbf{I}_{2^{q-k-1}}$$

Properties of $\mathbf{H}$

- $\mathbf{H}$ is sparse, hermitian and persymmetric
- $\mathbf{H}$ has the following sparsity pattern



- $\mathbf{H}$ can be transformed to two real blocks of half size:

$$\begin{pmatrix} \mathbf{I} & \mathbf{J} \\ \mathbf{I} & -\mathbf{J} \end{pmatrix} \cdot \mathbf{H} \cdot \begin{pmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{J} & -\mathbf{J} \end{pmatrix} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_2 \end{pmatrix}$$

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

$$e^{\mathbf{A}} := \sum_{k=0}^{\infty} \frac{\mathbf{A}^k}{k!}$$

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

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

- **Error Estimate:**

$$\| e^{\mathbf{A}} - S_m(\mathbf{A}) \| \leq \left(\frac{\| \mathbf{A} \|^{m+1}}{(m+1)!} \right) \left(\frac{1}{1 - \| \mathbf{A} \| / (m+2)} \right) \leq \delta$$

Properties of the matrix exponential

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

$$e^{A+B} = e^A \cdot e^B \Longleftrightarrow A \cdot B = B \cdot A$$

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

- **Trotter Product Formula:**

$$\lim_{m\to\infty} \left(e^{\mathbf{A}/m} e^{\mathbf{B}/m}\right)^m = e^{\mathbf{A}+\mathbf{B}}$$

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

- **Scaling & Squaring**

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

Computation of the matrix exponential: eigendecomposition

- In the case of a diagonal matrix

$$\mathbf{A} = \mathrm{diag}(d_1, \ldots, d_n) = \begin{pmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{pmatrix}$$

it holds

$$e^{\mathbf{A}} = \mathrm{diag}(e^{d_1}, \ldots, e^{d_n}) = \begin{pmatrix} e^{d_1} & & \\ & \ddots & \\ & & e^{d_n} \end{pmatrix}$$

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

$$e^{\mathbf{A}} = S\left(\mathrm{diag}(e^{d_1}, \ldots, e^{d_n})\right) S^{-1}$$

- **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:**

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

- 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

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

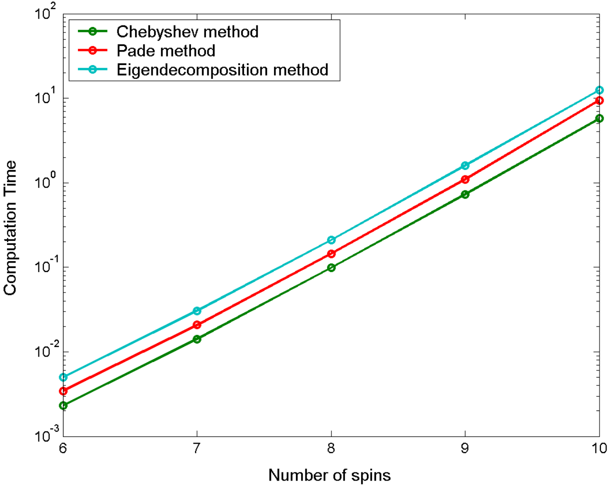
  - J_k: Bessel function
  - T_k: Chebyshev polynomial

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

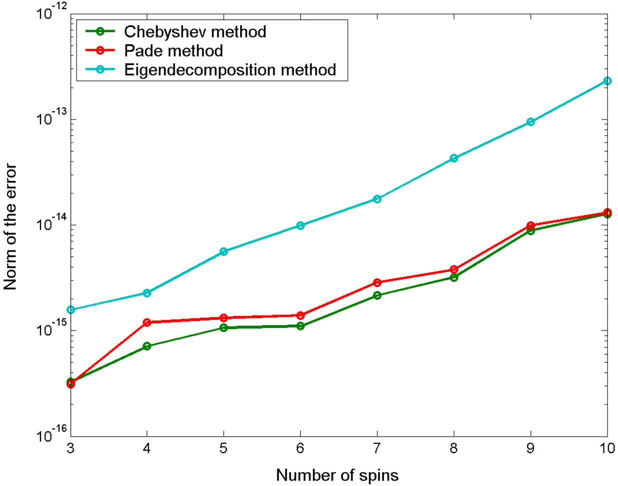
$$e^{\mathbf{A}} = J_0(i) + 2\sum_{k=1}^{\infty} i^k J_k(-i) T_k(\mathbf{A})$$

- In the case of $\mathbf{A}$ with arbitrary norm:
  Scaling & Squaring technique

Comparison of the methods: Computation time



Comparison of the methods: accuracy



Advantages of the Chebyshev series method

- Only the evaluation of a matrix polynomial required:
  $\Longrightarrow$ BLAS-Routines
- In the case of sparse matrices:
  Only products of the form dense * sparse appear
- Good convergence properties
- Matrix polynomials can be evaluated very efficiently:
  $p(\mathbf{A}) = \alpha_6\mathbf{A}^6 + \alpha_5\mathbf{A}^5 + \alpha_4\mathbf{A}^4 + \alpha_3\mathbf{A}^3 + \alpha_2\mathbf{A}^2 + \alpha_1\mathbf{A} + \alpha_0\mathbf{I}$
  - **Horner scheme** requires **5** non-scalar multiplications:

$$(((((\alpha_6\mathbf{A} + \alpha_5\mathbf{I}) \cdot \mathbf{A} + \alpha_4\mathbf{I}) \cdot \mathbf{A} + \alpha_3\mathbf{I}) \cdot \mathbf{A} + \alpha_2\mathbf{I}) \cdot \mathbf{A} + \alpha_1\mathbf{I}) \cdot \mathbf{A} + \alpha_0\mathbf{I}$$

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

$$\mathbf{A}_2 = \mathbf{A} \cdot \mathbf{A}, \quad \mathbf{A}_3 = \mathbf{A}_2 \cdot \mathbf{A}$$
$$p(\mathbf{A}) = \alpha_0\mathbf{I} + \alpha_1\mathbf{A} + \alpha_2\mathbf{A}_2 + \alpha_3\mathbf{A}_3 + \mathbf{A}_3 \cdot (\alpha_4\mathbf{A} + \alpha_5\mathbf{A}_2 + \alpha_6\mathbf{A}_3)$$

Parallel matrix-matrix-multiplication

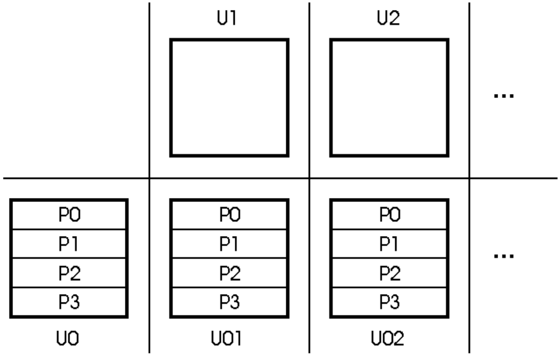
- Numerical task: Compute all intermediate products

$$
\begin{array}{l}
U_0 \\
U_0 \cdot U_1 \\
U_0 \cdot U_1 \cdot U_2 \\
\vdots \\
U_0 \cdot U_1 \cdot U_2 \cdots U_M
\end{array}
$$

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

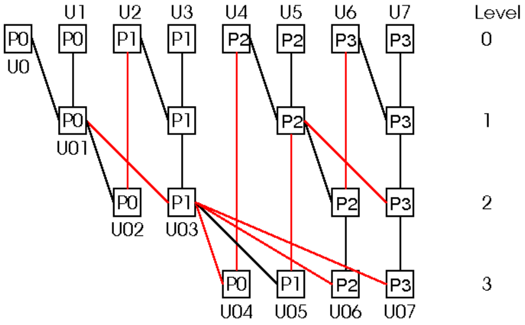
The slice-wise propagation

- Broadcast of all matrices $\mathbf{U}_k$ to all processors
- Each processor is responsible for ”its” rows



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)?
- ...