Fast Approximate Solution of the Non-Symmetric Generalized Eigenvalue Problem on Multicore Architectures

Martin Köhler
joint work with
Peter Benner and Jens Saak

Computational Methods in Systems and Control Theory Max Planck Institute for Dynamics of Complex Technical Systems
Motivation

Spectral Division and the Sign Function

Divide, Shift and Conquer Algorithm

Numerical Results

Conclusions

Outline

1. Motivation

2. Spectral Division and the Sign Function

3. The Divide, Shift and Conquer Algorithm

4. Numerical Results

5. Conclusions
We consider the non-symmetric generalized eigenvalue problem:

\[ Ax = \lambda Bx, \]

where \( A \in \mathbb{R}^{n \times n} \) and \( B \in \mathbb{R}^{n \times n} \) are non-singular matrices and \( \lambda \in \mathbb{C} \) is an eigenvalue with its eigenvector \( x \in \mathbb{R}^n \).
Motivation

Non-Symmetric Generalized Eigenvalue Problem

We consider the non-symmetric generalized eigenvalue problem:

\[ Ax = \lambda Bx, \]

where \( A \in \mathbb{R}^{n \times n} \) and \( B \in \mathbb{R}^{n \times n} \) are non-singular matrices and \( \lambda \in \mathbb{C} \) is an eigenvalue with its eigenvector \( x \in \mathbb{R}^n \).

Key idea behind the solution:

Compute the generalized Schur decomposition:

\[ Q^H AZ y = \lambda Q^H BZ y, \]

where \( S \in \mathbb{C}^{n \times n} \) and \( T \in \mathbb{C}^{n \times n} \) are upper triangular and \( Q \in \mathbb{C}^{n \times n} \) and \( Z \in \mathbb{C}^{n \times n} \) are unitary matrices.
Non-Symmetric Generalized Eigenvalue Problem

We consider the non-symmetric generalized eigenvalue problem:

\[ Ax = \lambda Bx, \]

where \( A \in \mathbb{R}^{n \times n} \) and \( B \in \mathbb{R}^{n \times n} \) are non-singular matrices and \( \lambda \in \mathbb{C} \) is an eigenvalue with its eigenvector \( x \in \mathbb{R}^n \).

Key idea behind the solution:

Compute the generalized Schur decomposition:

\[ Q^T A Z y = \lambda Q^T B Z y, \]

where \( S \in \mathbb{R}^{n \times n} \) and \( T \in \mathbb{R}^{n \times n} \) are quasi upper triangular and \( Q \in \mathbb{R}^{n \times n} \) and \( Z \in \mathbb{R}^{n \times n} \) are orthogonal matrices.
Motivation

Non-Symmetric Generalized Eigenvalue Problem

We consider the non-symmetric generalized eigenvalue problem:

\[ Ax = \lambda Bx, \]

where \( A \in \mathbb{R}^{n \times n} \) and \( B \in \mathbb{R}^{n \times n} \) are non-singular matrices and \( \lambda \in \mathbb{C} \) is an eigenvalue with its eigenvector \( x \in \mathbb{R}^n \).

Key idea behind the solution:

Compute the generalized Schur decomposition:

\[ QT S y = \lambda QT T y, \]

where \( S \in \mathbb{R}^{n \times n} \) and \( T \in \mathbb{R}^{n \times n} \) are quasi upper triangular and \( Q \in \mathbb{R}^{n \times n} \) and \( Z \in \mathbb{R}^{n \times n} \) are orthogonal matrices.

Applications:

- Direct solution of generalized Lyapunov equation:
  \[ AXE^T + EXA^T + M = 0, \]
- Direct solution of generalized Sylvester equations:
  \[ AXB + CXD + M = 0 \]
  or
  \[ AR - LB = C \]
  \[ DR - LE = F, \]

Various analysis methods for dynamical systems, ...

Max Planck Institute Magdeburg

Martin Köhler, Fast Approximate Solution of the NGEP
Motivation

QZ Algorithm

Common way to compute the generalized Schur decomposition:

\[ \text{Common way to compute the generalized Schur decomposition:} \]
## Motivation

### QZ Algorithm

Common way to compute the generalized Schur decomposition:

<table>
<thead>
<tr>
<th>QZ Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Compute $\tilde{B} = QB$ using the QR decomposition and transform $A$ into $\tilde{A} = Q^H A$.</td>
</tr>
<tr>
<td>2. Reduce the pair $(\tilde{A}, \tilde{B})$ to Hessenberg-Triangular form using Givens-Rotations.</td>
</tr>
<tr>
<td>3. Apply QZ steps to $(\tilde{A}, \tilde{B})$ until the matrix $\tilde{A}$ has reduced Hessenberg form. → generalized Schur form.</td>
</tr>
</tbody>
</table>

[Moler, Stewart ’73]
Common way to compute the generalized Schur decomposition:

**QZ Algorithm**

1. Compute $\tilde{B} = QB$ using the QR decomposition and transform $A$ into $\tilde{A} = Q^H A$.
2. Reduce the pair $(\tilde{A}, \tilde{B})$ to Hessenberg-Triangular form using Givens-Rotations.
3. Apply QZ steps to $(\tilde{A}, \tilde{B})$ until the matrix $\tilde{A}$ has reduced Hessenberg form. → **generalized Schur form**.

---

**DGEQRF** provides a level-3 BLAS implementation. 😊
Common way to compute the generalized Schur decomposition:

QZ Algorithm

1. Compute $\tilde{B} = QB$ using the QR decomposition and transform $A$ into $\tilde{A} = Q^H A$.
2. Reduce the pair $(\tilde{A}, \tilde{B})$ to Hessenberg-Triangular form using Givens-Rotations.
3. Apply QZ steps to $(\tilde{A}, \tilde{B})$ until the matrix $\tilde{A}$ has reduced Hessenberg form. → generalized Schur form.

Givens-Rotations are only level-1 BLAS operations. 😞
Common way to compute the generalized Schur decomposition:

<table>
<thead>
<tr>
<th>QZ Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Compute $\tilde{B} = QB$ using the QR decomposition and transform $A$ into $\tilde{A} = Q^H A Q$.</td>
</tr>
<tr>
<td>2. Reduce the pair $(\tilde{A}, \tilde{B})$ to Hessenberg-Triangular form using Givens-Rotations.</td>
</tr>
<tr>
<td>3. Apply QZ steps to $(\tilde{A}, \tilde{B})$ until the matrix $\tilde{A}$ has reduced Hessenberg form. $\rightarrow$ generalized Schur form.</td>
</tr>
</tbody>
</table>

Sequences of Givens-Rotations

[Moler, Stewart ’73]
Common way to compute the generalized Schur decomposition:

**QZ Algorithm**

1. Compute $\tilde{B} = QB$ using the QR decomposition and transform $A$ into $\tilde{A} = Q^H A$.
2. Reduce the pair $(\tilde{A}, \tilde{B})$ to Hessenberg-Triangular form using Givens-Rotations.
3. Apply QZ steps to $(\tilde{A}, \tilde{B})$ until the matrix $\tilde{A}$ has reduced Hessenberg form. $\rightarrow$ generalized Schur form.

$\rightarrow$ Implemented in LAPACK as DGGES or built using DGEQRF, DGGHRD, and DHGEQZ,
$\rightarrow$ Need $\approx 66n^3$ Flops,
$\rightarrow$ No parallel version in ScaLAPACK available.
**Motivation**

**QZ on Multicore Architectures**

**Example:** Runtime to compute the generalized Schur form on a dual 8-core Intel® Xeon® E5-2690:

<table>
<thead>
<tr>
<th>Matrix</th>
<th>dim.</th>
<th>Intel® MKL 11.0</th>
<th></th>
<th></th>
<th>OpenBLAS 0.2.8</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 Th.</td>
<td>8 Th.</td>
<td>16 Th.</td>
<td>1 Th.</td>
<td>8 Th.</td>
</tr>
<tr>
<td>rbs480</td>
<td>480</td>
<td>1.23s</td>
<td>1.10s</td>
<td>1.23s</td>
<td>1.38s</td>
<td>2.07s</td>
</tr>
<tr>
<td>bsst09</td>
<td>1083</td>
<td>16.28s</td>
<td>16.29s</td>
<td>16.46s</td>
<td>16.90s</td>
<td>16.89s</td>
</tr>
<tr>
<td>peec</td>
<td>1434</td>
<td>40.36s</td>
<td>39.90s</td>
<td>40.01s</td>
<td>41.07s</td>
<td>41.08s</td>
</tr>
<tr>
<td>bsst11</td>
<td>1473</td>
<td>48.07s</td>
<td>47.49s</td>
<td>47.48s</td>
<td>48.82s</td>
<td>48.17s</td>
</tr>
</tbody>
</table>
Motivation
QZ on Multicore Architectures

Example: Runtime to compute the generalized Schur form on a dual 8-core Intel® Xeon® E5-2690:

<table>
<thead>
<tr>
<th>Matrix</th>
<th>dim.</th>
<th>Intel® MKL 11.0</th>
<th>OpenBLAS 0.2.8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 Th.</td>
<td>8 Th.</td>
</tr>
<tr>
<td>rbs480</td>
<td>480</td>
<td>1.00</td>
<td>1.12</td>
</tr>
<tr>
<td>bsst09</td>
<td>1083</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>peec</td>
<td>1434</td>
<td>1.00</td>
<td>1.01</td>
</tr>
<tr>
<td>bsst11</td>
<td>1473</td>
<td>1.00</td>
<td>1.01</td>
</tr>
</tbody>
</table>

→ No acceleration using parallel BLAS at all.
→ We need a new and faster way to approximate the generalized Schur decomposition on current hardware.
Example: Runtime to compute the generalized Schur form on a dual 8-core Intel® Xeon® E5-2690:

<table>
<thead>
<tr>
<th>Matrix</th>
<th>dim.</th>
<th>Intel® MKL 11.0</th>
<th>OpenBLAS 0.2.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbs480</td>
<td>480</td>
<td>1.00 1.12 1.00</td>
<td>1.00 0.66 0.57</td>
</tr>
<tr>
<td>bsst09</td>
<td>1083</td>
<td>1.00 1.00 0.99</td>
<td>1.00 1.00 0.99</td>
</tr>
<tr>
<td>peec</td>
<td>1434</td>
<td>1.00 1.01 1.01</td>
<td>1.00 1.00 0.92</td>
</tr>
<tr>
<td>bsst11</td>
<td>1473</td>
<td>1.00 1.01 1.01</td>
<td>1.00 1.01 0.91</td>
</tr>
</tbody>
</table>

→ No acceleration using parallel BLAS at all.
## Motivation

**QZ on Multicore Architectures**

**Example:** Runtime to compute the generalized Schur form on a dual 8-core Intel® Xeon® E5-2690:

<table>
<thead>
<tr>
<th>Matrix</th>
<th>dim.</th>
<th>Intel® MKL 11.0</th>
<th>OpenBLAS 0.2.8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 Th.</td>
<td>8 Th.</td>
</tr>
<tr>
<td>rbs480</td>
<td>480</td>
<td>1.00</td>
<td>1.12</td>
</tr>
<tr>
<td>bsst09</td>
<td>1083</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>peec</td>
<td>1434</td>
<td>1.00</td>
<td>1.01</td>
</tr>
<tr>
<td>bsst11</td>
<td>1473</td>
<td>1.00</td>
<td>1.01</td>
</tr>
</tbody>
</table>

→ No acceleration using parallel BLAS at all.

→ We need a new and faster way to approximate the generalized Schur decomposition on current hardware.
Spectral Division and the Sign Function
From the block generalized Schur form:

\[
\begin{pmatrix}
Q_1^T \\
Q_2^T
\end{pmatrix} A \begin{pmatrix} Z_1 & Z_2 \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\
0 & A_{22} \end{pmatrix}
\]

and

\[
\begin{pmatrix}
Q_1^T \\
Q_2^T
\end{pmatrix} B \begin{pmatrix} Z_1 & Z_2 \end{pmatrix} = \begin{pmatrix} B_{11} & B_{12} \\
0 & B_{22} \end{pmatrix},
\]

we get two independent eigenvalue problems \((A_{11}, B_{11})\) and \((A_{22}, B_{22})\).
Spectral Division and the Sign Function

Spectral Division

From the block generalized Schur form:

\[
\begin{pmatrix}
Q_1^T \\
Q_2^T
\end{pmatrix}
A
\begin{pmatrix}
Z_1 \\
Z_2
\end{pmatrix} =
\begin{pmatrix}
A_{11} & A_{12} \\
0 & A_{22}
\end{pmatrix}
\]

and

\[
\begin{pmatrix}
Q_1^T \\
Q_2^T
\end{pmatrix}
B
\begin{pmatrix}
Z_1 \\
Z_2
\end{pmatrix} =
\begin{pmatrix}
B_{11} & B_{12} \\
0 & B_{22}
\end{pmatrix}
\]

we get two independent eigenvalue problems \((A_{11}, B_{11})\) and \((A_{22}, B_{22})\).

Our Aim: Split \((A, B)\) such that \(\Lambda(A_{11}, B_{11}) \subset \mathbb{C}_-\) and \(\Lambda(A_{22}, B_{22}) \subset \mathbb{C}_+\).
Spectral Division and the Sign Function

Spectral Division

From the block generalized Schur form:

\[
\begin{pmatrix}
Q_1^T \\
Q_2^T \\
Q^T
\end{pmatrix}
A
\begin{pmatrix}
Z_1 \\
Z_2
\end{pmatrix}
= \begin{pmatrix}
A_{11} & A_{12} \\
0 & 0
\end{pmatrix}
\]

and

\[
\begin{pmatrix}
Q_1^T \\
Q_2^T \\
Q^T
\end{pmatrix}
B
\begin{pmatrix}
Z_1 \\
Z_2
\end{pmatrix}
= \begin{pmatrix}
B_{11} & B_{12} \\
0 & B_{22}
\end{pmatrix}
\]

we get two independent eigenvalue problems \((A_{11}, B_{11})\) and \((A_{22}, B_{22})\).

Our Aim: Split \((A, B)\) such that \(\Lambda(A_{11}, B_{11}) \subset \mathbb{C}^-\) and \(\Lambda(A_{22}, B_{22}) \subset \mathbb{C}^+\).
Matrix Sign Function

Let $Y \diag(J_1, J_2) Y^{-1} = A$ be the Jordan canonical form of a matrix $A \in \mathbb{R}^{n \times n}$ with $\Lambda(J_1) \subset \mathbb{C}_-$ and $\Lambda(J_2) \subset \mathbb{C}_+$. Then

$$\sign(A) := Y \begin{pmatrix} -I_1 & 0 \\ 0 & I_2 \end{pmatrix} Y^{-1}$$

is the sign of the matrix $A$, where $\dim(I_i) = \dim(J_i)$, $i = 1, 2$. 
Spectral Division and the Sign Function

(Generalized) Sign Function

Matrix Sign Function

Let \( Y \text{ diag}(J_1, J_2) Y^{-1} = A \) be the Jordan canonical form of a matrix \( A \in \mathbb{R}^{n \times n} \) with \( \Lambda(J_1) \subset \mathbb{C}_- \) and \( \Lambda(J_2) \subset \mathbb{C}_+ \). Then

\[
\text{sign}(A) := Y \begin{pmatrix} -I_1 & 0 \\ 0 & I_2 \end{pmatrix} Y^{-1}
\]

is the sign of the matrix \( A \), where \( \text{dim}(I_i) = \text{dim}(J_i) \), \( i = 1, 2 \).

Some properties:

- \( \text{Range}(I + \text{sign}(A)) \) is the subspace corresponding to all eigenvalues with positive real part.
- \( \text{sign}(A)^2 = I \)
From \( (A)^2 = I \) follows the Newton scheme:

\[
A_0 \leftarrow A, \quad A_{k+1} \leftarrow \frac{1}{2} \left( A_k + A_k^{-1} \right), \quad k = 0, 1, 2, \ldots
\]

to compute the sign of a matrix.
The Generalized Sign function iteration:

\[ A_0 \leftarrow A, \quad A_{k+1} \leftarrow \frac{1}{2} \left( A_k + BA_k^{-1}B \right), \quad k = 0, 1, 2, \ldots \]
The Generalized Sign function iteration:

\[
A_0 \leftarrow A, \quad A_{k+1} \leftarrow \frac{1}{2c_k} \left( A_k + c_k^2 BA_k^{-1}B \right), \quad k = 0, 1, 2, \ldots
\]

where \( c_k \) is a additional scaling factor. Typical: \( c_k = \left( \frac{|\det(A_k)|}{|\det(B)|} \right)^{\frac{1}{n}} \).
The Generalized Sign function iteration:

\[ A_0 \leftarrow A, \quad A_{k+1} \leftarrow \frac{1}{2c_k} \left( A_k + c_k^2 BA_k^{-1}B \right), \quad k = 0, 1, 2, \ldots \]

where \( c_k \) is a additional scaling factor. Typical: \( c_k = \left( \frac{|\text{det}(A_k)|}{|\text{det}(B)|} \right)^{\frac{1}{n}} \).

Properties change to:

- Range \( (B + \text{sign}(A, B)) \) is the right deflating subspace corresponding to all eigenvalues with positive real part.
- Range \( (B - \text{sign}(A, B)) \) is the right deflating subspace corresponding to all eigenvalues with negative real part.
The generalized sign function iteration:

\[ A_0 \leftarrow \frac{1}{2} (A_k + c_k^2 B A_k^{-1} B), \quad k = 0, 1, 2, \ldots \]

where \( c_k \) is an additional scaling factor. Typical: \( c_k = \left( \frac{\left| \text{det}(A_k) \right|}{\left| \text{det}(B) \right|} \right)^{\frac{1}{n}} \).

Properties:

- The generalized sign function iteration employs only level-3 routines: DGETRF, DGETRS, and DGEMM.
- The matrix \( Z = [Z_1, Z_2] \) can be constructed using the range properties.
- \( \text{Range} (B + \text{sign} (A, B)) \) is the right deflating subspace corresponding to all eigenvalues with positive real part.
- \( \text{Range} (B - \text{sign} (A, B)) \) is the right deflating subspace corresponding to all eigenvalues with negative real part.
Spectral Division and the Sign Function
Spectral Division using the Sign Function

Questions:

1. How to construct $Z$ using level-3 operations in a robust way?
2. How to compute the corresponding $Q$?
Spectral Division and the Sign Function

Spectral Division using the Sign Function

[Sun, Quintana-Ortí ’04]

Questions:

1. How to construct $Z$ using level-3 operations in a robust way?
2. How to compute the corresponding $Q$?

Computation of $Z$: From the range properties follows:

$$(B + \text{sign}(A, B))Z_1 = 0 \quad \text{and} \quad (B + \text{sign}(A, B))Z_2 = K$$
Spectral Division and the Sign Function

Spectral Division using the Sign Function

[Sun, Quintana-Ortí ’04]

Questions:

1. How to construct $Z$ using level-3 operations in a robust way?
2. How to compute the corresponding $Q$?

Computation of $Z$: From the range properties follows:

$$(B + \text{sign}(A, B))[Z_1, Z_2] = [0, K]$$
Spectral Division and the Sign Function

Questions:
1. How to construct $Z$ using level-3 operations in a robust way?
2. How to compute the corresponding $Q$?

Computation of $Z$: From the range properties follows:

$$(B + \text{sign}(A, B))^T = [Z_1, Z_2] \begin{pmatrix} 0 \\ K \end{pmatrix}$$
Questions:

1. How to construct $Z$ using level-3 operations in a robust way?
2. How to compute the corresponding $Q$?

Computation of $Z$: From the range properties follows:

$$(B + \text{sign}(A, B))^T = [Z_2, Z_1] \begin{pmatrix} K \\ 0 \end{pmatrix}$$
Spectral Division and the Sign Function

Questions:
1. How to construct $Z$ using level-3 operations in a robust way?
2. How to compute the corresponding $Q$?

Computation of $Z$: From the range properties follows:

$$(B + \text{sign}(A, B))^T \Pi_Z = [Z_2, Z_1] \begin{pmatrix} K \\ 0 \end{pmatrix}$$

→ use a Rank Revealing QR Decomposition (RRQR)
Questions:

1. How to construct $Z$ using level-3 operations in a robust way?
2. How to compute the corresponding $Q$?

Computation of $Q$:

- $Q_1$ lies in the range of $AZ_1 + BZ_1$,
- $Q_2$ is complementary orthogonal to $AZ_1 + BZ_1$. 
Questions:

1. How to contract $Z$ using level-3 operations in a robust way?
2. How to compute the corresponding $Q$?

Computation of $Q$:

\[
\begin{pmatrix}
Q_1^H \\
Q_2^H
\end{pmatrix}
[AZ_1, BZ_1] = 
\begin{pmatrix}
M \\
0
\end{pmatrix}
\]
Spectral Division and the Sign Function

Spectral Division using the Sign Function

[Sun, Quintana-Ortí ’04]

Questions:

1. How to construct $Z$ using level-3 operations in a robust way?
2. How to compute the corresponding $Q$?

Computation of $Q$:

$$[AZ_1, BZ_1] = [Q_1, Q_2] \begin{pmatrix} \mathcal{M} \\ 0 \end{pmatrix}$$
Spectral Division and the Sign Function

Questions:
1. How to construct $Z$ using level-3 operations in a robust way?
2. How to compute the corresponding $Q$?

Computation of $Q$:

$$[AZ_1, BZ_1] \Pi_Q = [Q_1, Q_2] \begin{pmatrix} M \\ 0 \end{pmatrix}$$

→ use a RRQR procedure again.
Questions:

1. How to construct $Z$ using level-3 operations in a robust way?
2. How to compute the corresponding $Q$?

Computation of $Q$:

$$[AZ_1, BZ_1] \prod Q = [Q_1, Q_2] \begin{pmatrix} M \\ 0 \end{pmatrix}$$

$\rightarrow$ use a RRQR procedure again.

We can compute $Q$ and $Z$ from $\text{sign}(A, B)$ using two RRQR procedures.

$\rightarrow$ use level-3 subroutine $\text{DGEQP3}$ from LAPACK.
Spectral Division and the Sign Function

Spectral Division using the Sign Function

**Algorithm 1** Spectral Division using the Generalized Sign function

**Input:** $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times n}$ non-singular, $\Lambda(A, B) \cap i\mathbb{R} = \{\}$.  

**Output:** $Q \in \mathbb{R}^{n \times n}$ and $Z \in \mathbb{R}^{n \times n}$ orthogonal, such that the spectrum is split at $i\mathbb{R}$.

1. Compute $S = \text{sign} (A, B)$ using the Newton iteration
2. Compute $Z = [Z_1, Z_2]$ using a RRQR procedure:
   \[
   (B + S)^T \Pi Z = [Z_2, Z_1] \begin{pmatrix} K \\ 0 \end{pmatrix}
   \]
3. Compute $Q = [Q_1, Q_2]$ using a RRQR procedure:
   \[
   [AZ_1, BZ_1] \Pi Q = [Q_1, Q_2] \begin{pmatrix} M \\ 0 \end{pmatrix}
   \]
Spectral Division and the Sign Function

Spectral Division using the Sign Function

Algorithm 1

Input: $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times n}$ non-singular, $\Lambda(A, B) \cap \mathbb{R} = \emptyset$,

Output: $Q \in \mathbb{R}^{n \times n}$ and $Z \in \mathbb{R}^{n \times n}$ orthogonal, such that the spectrum is split at $i\mathbb{R}$.

1: Compute $S = \text{sign}(A, B)$ using the Newton iteration
2: Compute $Z = [Z_1, Z_2]$ using a RRQR procedure:
   \[(B + S)^T \Pi_Z = [Z_2, Z_1] \begin{pmatrix} K & \cdot \\ \cdot & 0 \end{pmatrix}\]
3: Compute $Q = [Q_1, Q_2]$ using a RRQR procedure:
   \[[AZ_1, BZ_1] \Pi_Q = [Q_1, Q_2] \begin{pmatrix} M & \cdot \\ \cdot & 0 \end{pmatrix}\]

Computational Costs:

- Generalized Sign Function: $\approx 70n^3$ Flops
- RRQR using DGEQP3 for $Z$: $\frac{8}{3}n^3$ Flops
- RRQR using DGEQP3 for $Q$: $0$ Flops
- $(B + S)^T \Pi_Z$ (minimum): $8n^3$ Flops
- $(B + S)^T \Pi_Z$ (maximum): $8n^3$ Flops
- Transform $A$ and $B$: $8n^3$ Flops

→ more than $QZ$
→ but only level-3 enabled operations.
The Divide, Shift and Conquer Algorithm
The Divide, Shift and Conquer Algorithm

Recursive Spectral Division

We got two independent eigenvalue problems for \((A_{11}, B_{11})\) and \((A_{22}, B_{22})\) from the spectral division.
The Divide, Shift and Conquer Algorithm

Recursive Spectral Division

We got two independent eigenvalue problems for \((A_{11}, B_{11})\) and \((A_{22}, B_{22})\) from the spectral division.

**Problem:** Applying the spectral division again will not give smaller subproblems again.

- \(\Lambda(A_{11}, B_{11})\) lies completely in \(\mathbb{C}_-\),
- \(\Lambda(A_{22}, B_{22})\) lies completely in \(\mathbb{C}_+\),

\(\rightarrow\) No recursive scheme possible.
The Divide, Shift and Conquer Algorithm

We got two independent eigenvalue problems for \((A_{11}, B_{11})\) and \((A_{22}, B_{22})\) from the spectral division.

Problem: Applying the spectral division again will not give smaller subproblems again.

\[ \Lambda(A_{11}, B_{11}) \text{ lies completely in } \mathbb{C}^{-}, \]
\[ \Lambda(A_{22}, B_{22}) \text{ lies completely in } \mathbb{C}^{+}, \]
→ No recursive scheme possible.

Original Spectrum:

Idea:
Shift the spectrum of \((A_{11}, B_{11})\) to the right and \((A_{22}, B_{22})\) to the left to get two new spectra which enclose the imaginary axis.
The Divide, Shift and Conquer Algorithm
Recursive Spectral Division

We got **two independent** eigenvalue problems for \((A_{11}, B_{11})\) and \((A_{22}, B_{22})\) from the spectral division.

**Problem:** Applying the spectral division again will not give smaller subproblems again.

- \(\Lambda(A_{11}, B_{11})\) lies completely in \(\mathbb{C}_-\),
- \(\Lambda(A_{22}, B_{22})\) lies completely in \(\mathbb{C}_+\),

\(\rightarrow\) No recursive scheme possible.

**Idea**

Shift the spectrum of \((A_{11}, B_{11})\) to the right and \((A_{22}, B_{22})\) to the left to get two new spectra which enclose the imaginary axis.
The Divide, Shift and Conquer Algorithm

We got two independent eigenvalue problems for \((A_{11}, B_{11})\) and \((A_{22}, B_{22})\) from the spectral division. Problem: Applying the spectral division again will not give smaller subproblems again.

\[ \Lambda(A_{11}, B_{11}) \text{ lies completely in } C^{-}, \quad \Lambda(A_{22}, B_{22}) \text{ lies completely in } C^{+}, \]

\[ \rightarrow \text{No recursive scheme possible.} \]

Idea:
Shift the spectrum of \((A_{11}, B_{11})\) to the right and \((A_{22}, B_{22})\) to the left to get two new spectra which enclose the imaginary axis.

Shifted Spectrum:
The Divide, Shift and Conquer Algorithm

Recursive Spectral Division

We want to have two new eigenvalue problems:

\[(\tilde{A}_{11}, B_{11}) := (A_{11} - \theta - B_{11}, B_{11})\]

and

\[(\tilde{A}_{22}, B_{22}) := (A_{22} - \theta + B_{22}, B_{22})\]

such that we can apply the division algorithm again.
The Divide, Shift and Conquer Algorithm

Recursive Spectral Division

We want to have two new eigenvalue problems:

\[
(\tilde{A}_{11}, B_{11}) := (A_{11} - \theta_- B_{11}, B_{11})
\]

and

\[
(\tilde{A}_{22}, B_{22}) := (A_{22} - \theta_+ B_{22}, B_{22})
\]

such that we can apply the division algorithm again.

**Optimal Choice of \( \theta_* \):** Chose \( \theta_- \) or respectively \( \theta_+ \) such that the problems emerging out of \((\tilde{A}_{11}, B_{11})\) and \((\tilde{A}_{22}, B_{22})\) after the spectral division are equally sized.
The Divide, Shift and Conquer Algorithm

Recursive Spectral Division

We want to have two new eigenvalue problems:

\[(\tilde{A}_{11}, B_{11}) := (A_{11} - \theta - B_{11}, B_{11})\]

and

\[(\tilde{A}_{22}, B_{22}) := (A_{22} - \theta + B_{22}, B_{22})\]

such that we can apply the division algorithm again.

**Optimal Choice of \(\theta_*\):** Chose \(\theta_-\) or respectively \(\theta_+\) such that the problems emerging out of \((\tilde{A}_{11}, B_{11})\) and \((\tilde{A}_{22}, B_{22})\) after the spectral division are equally sized.

**Problem:** Determining the optimal parameters \(\theta_*\) requires the knowledge of all eigenvalues.
w.l.o.g.: We restrict to $(A_{11}, B_{11})$ and the left half-plane.

If the real parts of the eigenvalues are equally distributed, the optimal $\theta_-$ is obviously given by

$$\theta_- := \frac{1}{2} \Re(\lambda_{\text{left}})$$

where $\lambda_{\text{left}}$ is the left-most eigenvalue of $(A_{11}, B_{11})$. 
The Divide, Shift and Conquer Algorithm

Optimal Shift Parameter Approximation

w.l.o.g.: We restrict to \((A_{11}, B_{11})\) and the left half-plane.

If the real parts of the eigenvalues are equally distributed, the optimal \(\theta_-\) is obviously given by

\[
\theta_- := \frac{1}{2} \Re(\lambda_{\text{left}})
\]

where \(\lambda_{\text{left}}\) is the left-most eigenvalue of \((A_{11}, B_{11})\).

**Cheap approximation of** \(\Re(\lambda_{\text{left}})\):

\[-\Re(\lambda_{\text{left}}) \leq \rho(A_{11}, B_{11})\]

where \(\rho(A_{11}, B_{11})\) is the spectral radius of \((A_{11}, B_{11})\).
The Divide, Shift and Conquer Algorithm

Optimal Shift Parameter Approximation

w.l.o.g.: We restrict to \((A_{11}, B_{11})\) and the left half-plane.

If the real parts of the eigenvalues are equally distributed, the optimal \(\theta_-\) is obviously given by

\[
\theta_- := \frac{1}{2} \Re(\lambda_{\text{left}})
\]

where \(\lambda_{\text{left}}\) is the left-most eigenvalue of \((A_{11}, B_{11})\).

**Cheap approximation of \(\Re(\lambda_{\text{left}})\):**

\[-\Re(\lambda_{\text{left}}) \leq \rho(A_{11}, B_{11}) \leq \|B_{11}^{-1}A_{11}\|_2\]

where \(\rho(A_{11}, B_{11})\) is the spectral radius of \((A_{11}, B_{11})\).
w.l.o.g.: We restrict to \((A_{11}, B_{11})\) and the left half-plane.

If the real parts of the eigenvalues are equally distributed, the optimal \(\theta_{-}\) is obviously given by

\[
\theta_{-} := \frac{1}{2} \Re(\lambda_{\text{left}})
\]

where \(\lambda_{\text{left}}\) is the left-most eigenvalue of \((A_{11}, B_{11})\).

**Cheap approximation of \(\Re(\lambda_{\text{left}})\):**

\[
-\Re(\lambda_{\text{left}}) \leq \rho(A_{11}, B_{11}) \leq \|B_{11}^{-1}A_{11}\|_2 \leq \|B_{11}^{-1}A_{11}\|_F
\]

where \(\rho(A_{11}, B_{11})\) is the spectral radius of \((A_{11}, B_{11})\).
**The Divide, Shift and Conquer Algorithm**

**The Algorithm**

Combining the spectral division and the shift parameter computation gives the following recursive scheme:
The Divide, Shift and Conquer Algorithm

The Algorithm

Combining the spectral division and the shift parameter computation gives the following recursive scheme:

\textbf{Algorithm 2} \quad [Q, Z] = dscqz(A, B)

\textbf{Input:} \quad A \in \mathbb{R}^{n \times n} \text{ and } B \in \mathbb{R}^{n \times n} \text{ non-singular, } \Lambda(A, B) \cap i\mathbb{R} = \{\}

\textbf{Output:} \quad (Q^T A Z, Q^T B Z) \text{ in real Schur form.}

1: \quad \textbf{if} \; (A, B) \text{ is trivial to solve } \textbf{then}

2: \quad \text{Compute } Q, Z \text{ directly and return them.}

3: \quad \textbf{end if}

4: \quad \text{Compute } Q \text{ and } Z \text{ using Algorithm 1 and transform } (A, B).

5: \quad \text{Set } \theta_- = -\frac{1}{2} \|B_{11}^{-1} A_{11}\|_F \text{ and } \theta_+ = \frac{1}{2} \|B_{22}^{-1} A_{22}\|_F.

6: \quad [\tilde{Q}_1, \tilde{Z}_1] = \text{dscqz}(A_{11} - \theta_- B_{11}, B_{11}).

7: \quad [\tilde{Q}_2, \tilde{Z}_2] = \text{dscqz}(A_{22} - \theta_+ B_{22}, B_{22}).

8: \quad \text{Update } Q := Q \begin{pmatrix} \tilde{Q}_1 & 0 \\ 0 & \tilde{Q}_2 \end{pmatrix} \text{ and } Z := Z \begin{pmatrix} \tilde{Z}_1 & 0 \\ 0 & \tilde{Z}_2 \end{pmatrix}.

9: \quad \textbf{return} \; [Q, Z]
The Divide, Shift and Conquer Algorithm

The Algorithm

Combining the spectral division and the shift parameter computation gives the following recursive scheme:

\[ (Q, Z) = dscqz(A, B) \]

**Algorithm 2**

**Input:** \( A \in \mathbb{R}^{n \times n} \) and \( B \in \mathbb{R}^{n \times n} \) non-singular, \( \Lambda(A, B) \cap i\mathbb{R} = \{\} \)

**Output:** \( (Q^T A Z, Q^T B Z) \) in real Schur form.

1. **if** \((A, B)\) is **trivial to solve** **then**
2. **Compute** \( Q, Z \) directly and return
3. **end if**
4. **Compute** \( Q \) and \( Z \) using Algorithm 1 and transform \((A, B)\).
5. Set \( \theta_- = -\frac{1}{2} \| B_{11}^{-1} A_{11} \|_F \) and \( \theta_+ = \frac{1}{2} \| B_{22}^{-1} A_{22} \|_F \).
6. \( [\tilde{Q}_1, \tilde{Z}_1] = dscqz(A_{11} - \theta_- B_{11}, B_{11}) \).
7. \( [\tilde{Q}_2, \tilde{Z}_2] = dscqz(A_{22} - \theta_+ B_{22}, B_{22}) \).
8. Update \( Q := Q \left( \begin{array}{cc} \tilde{Q}_1 & 0 \\ 0 & \tilde{Q}_2 \end{array} \right) \) and \( Z := Z \left( \begin{array}{cc} \tilde{Z}_1 & 0 \\ 0 & \tilde{Z}_2 \end{array} \right) \).
9. **return** \([Q, Z]\)

**Trivial:** The Schur form can be computed directly, i.e. the problem is of size \( 1 \times 1 \) or \( 2 \times 2 \).
The evaluation of $\theta_- = -\frac{1}{2} \| B_{11}^{-1} A_{11} \|_F$ and $\theta_+ = \frac{1}{2} \| B_{22}^{-1} A_{22} \|_F$ is only necessary after the first step.

The spectral radius can not increase during the recursion. → We pass $|\theta_-|$ and $|\theta_+|$ as spectral radius $\theta$ to the to the next step and use

$$\theta_- := -\frac{1}{2} \theta \quad \text{and} \quad \theta_+ := \frac{1}{2} \theta$$

as new parameters in the next step.
The Divide, Shift and Conquer Algorithm

Implementation Details

- The evaluation of $\theta_- = -\frac{1}{2} \| B_{11}^{-1} A_{11} \|_F$ and $\theta_+ = \frac{1}{2} \| B_{22}^{-1} A_{22} \|_F$ is only necessary after the first step.

The spectral radius can not increase during the recursion.

→ We pass $|\theta_-|$ and $|\theta_+|$ as spectral radius $\theta$ to the next step and use

$$\theta_- := -\frac{1}{2} \theta \quad \text{and} \quad \theta_+ := \frac{1}{2} \theta$$

as new parameters in the next step.

→ We can guarantee $\theta_* \to 0$ during the recursion.
The evaluation of $\theta_- = -\frac{1}{2} \| B_{11}^{-1} A_{11} \|_F$ and $\theta_+ = \frac{1}{2} \| B_{22}^{-1} A_{22} \|_F$ is only necessary after the first step.

Reformulate the recursion as an iterative scheme.

→ Done using a queue.
→ Restrict the additional memory to $4n^2 + 2n$.
→ Allows further rearrangements of the algorithm.
The Divide, Shift and Conquer Algorithm

Implementation Details

- The evaluation of $\theta_- = -\frac{1}{2}\|B_{11}^{-1}A_{11}\|_F$ and $\theta_+ = \frac{1}{2}\|B_{22}^{-1}A_{22}\|_F$ is only necessary after the first step.
- Reformulate the recursion as an iterative scheme.
- New definition of “trivial to solve”: The can be solved inside the cache of a single CPU-core by DGGES.

The trivial size $n_{\text{triv}}$ given by:

$$n_{\text{triv}} \leq -\frac{11}{8} + \sqrt{-\frac{135}{64} + \frac{C}{4}} \approx \sqrt{\frac{C}{4}}$$

where $C$ is the cache size counted in floating point numbers of the desired precision.
The Divide, Shift and Conquer Algorithm

Parallelization

We split the iterative formulation into 3 phases:

1. Perform the whole spectral division and the divide and conquer procedure of Algorithm 2 **without** solving the trivial problems.

   → only level-3 operations, use a threaded BLAS library
   → requires the whole memory bandwidth
We split the iterative formulation into 3 phases:

1. Perform the whole spectral division and the divide and conquer procedure of Algorithm 2 without solving the trivial problems.

2. Solve the remaining trivial problems in parallel. Each problem is solved by one CPU-core in single-threaded mode.

   → OpenMP, PThreads,...
   → $n_{\text{triv}}$ is hardware dependent.
   → reduce the transfers between cache and main memory.
We split the iterative formulation into 3 phases:

1. Perform the whole spectral division and the divide and conquer procedure of Algorithm 2 \textbf{without} solving the trivial problems.

2. Solve the remaining trivial problems in parallel. Each problem is solved by one CPU-core in single-threaded mode.

3. Update \( Q := Q \text{ diag}(Q_1, Q_2, \ldots) \) and \( Z := \text{diag}(Z_1, Z_2, \ldots) \) with \( Q_* \) and \( Z_* \) from the trivial problems.

\( \rightarrow \) Involves only matrix-matrix products, use a threaded BLAS library.
Numerical Results
Numerical Results

Test hardware:

<table>
<thead>
<tr>
<th></th>
<th>Compue-Server Xeon E5-2690</th>
<th>Workstation Xeon E3-1245</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU:</strong></td>
<td>Dual Xeon E5-2690 @ 2.9 GHz</td>
<td>Xeon E3-1245 @ 3.3GHz</td>
</tr>
<tr>
<td><strong>Cores:</strong></td>
<td>16 (2×8)</td>
<td>4</td>
</tr>
<tr>
<td><strong>L2 Cache:</strong></td>
<td>256KiB</td>
<td>256KiB</td>
</tr>
<tr>
<td><strong>$n_{\text{triv}}$</strong></td>
<td>90</td>
<td>90</td>
</tr>
<tr>
<td><strong>RAM:</strong></td>
<td>32 GiB DDR3</td>
<td>8 GiB DDR3</td>
</tr>
<tr>
<td><strong>OS:</strong></td>
<td>Ubuntu 12.04</td>
<td>Ubuntu 12.04</td>
</tr>
<tr>
<td><strong>Compiler:</strong></td>
<td>GCC 4.6.3</td>
<td>GCC 4.6.3</td>
</tr>
<tr>
<td><strong>BLAS:</strong></td>
<td>Intel MKL 10.2</td>
<td>Intel MKL 10.2</td>
</tr>
</tbody>
</table>

Test matrices from MatrixMarket and the Oberwolfach Collection:

<table>
<thead>
<tr>
<th></th>
<th>Name</th>
<th>Dimension</th>
<th></th>
<th>Name</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>rbs480</td>
<td>480</td>
<td>(b)</td>
<td>bsst09</td>
<td>1 083</td>
</tr>
<tr>
<td>(c)</td>
<td>spiral inductor</td>
<td>1434</td>
<td>(d)</td>
<td>bcsst11</td>
<td>1 473</td>
</tr>
<tr>
<td>(e)</td>
<td>filter2D</td>
<td>1 668</td>
<td>(f)</td>
<td>bcsst21</td>
<td>3 600</td>
</tr>
<tr>
<td>(g)</td>
<td>steel profile</td>
<td>5 177</td>
<td>(h)</td>
<td>steel profile</td>
<td>20 209</td>
</tr>
</tbody>
</table>
## Numerical Results

### Runtime and Speedup

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Xeon E3-1245</th>
<th></th>
<th></th>
<th>Dual Xeon E5-2690 - MKL 10.2</th>
<th></th>
<th></th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QZ</td>
<td>4 Thr.</td>
<td>QZ</td>
<td>1 Thr.</td>
<td>16 Thr.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a)</td>
<td>1.31s</td>
<td>0.59s</td>
<td>1.75s</td>
<td>1.16s</td>
<td>0.51s</td>
<td>3.57</td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>17.27s</td>
<td>10.48s</td>
<td>18.99s</td>
<td>22.68s</td>
<td>6.29s</td>
<td>3.02</td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>40.16s</td>
<td>15.05s</td>
<td>39.86s</td>
<td>32.47s</td>
<td>8.16s</td>
<td>4.88</td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>46.77s</td>
<td>43.09s</td>
<td>64.38s</td>
<td>86.90s</td>
<td>25.69s</td>
<td>2.51</td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td>77.35s</td>
<td>28.38s</td>
<td>80.40s</td>
<td>67.40s</td>
<td>14.41s</td>
<td>4.68</td>
<td></td>
</tr>
<tr>
<td>(f)</td>
<td>616.05s</td>
<td>526.22s</td>
<td>740.78s</td>
<td>1189.69s</td>
<td>383.08s</td>
<td>1.93</td>
<td></td>
</tr>
<tr>
<td>(g)</td>
<td>3046.40s</td>
<td>1006.25s</td>
<td>3286.61s</td>
<td>2684.74s</td>
<td>598.35s</td>
<td>5.49</td>
<td></td>
</tr>
<tr>
<td>(h)</td>
<td>out of memory</td>
<td></td>
<td>255057s</td>
<td>207198s</td>
<td>38200s</td>
<td>6.68</td>
<td></td>
</tr>
</tbody>
</table>
Numerical Results

Runtime and Speedup

<table>
<thead>
<tr>
<th></th>
<th>Xeon E3-1245</th>
<th>Dual Xeon E5-2690 - MKL 10.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Thr.</td>
<td>1.31s</td>
<td>0.59s</td>
</tr>
<tr>
<td>4 Thr.</td>
<td>1.75s</td>
<td>1.16s</td>
</tr>
<tr>
<td>16 Thr.</td>
<td>3.57s</td>
<td>0.51s</td>
</tr>
<tr>
<td>1.31s</td>
<td>0.59s</td>
<td>1.75s</td>
</tr>
<tr>
<td>16 Thr.</td>
<td>3.57s</td>
<td>0.51s</td>
</tr>
</tbody>
</table>

- (a) 1.31s
- (b) 17.27s
- (c) 40.16s
- (d) 46.77s
- (e) 77.35s
- (f) 616.05s
- (g) 3046.40s
- (h) out of memory

- Max Planck Institute Magdeburg
- Martin Kähler, Fast Approximate Solution of the NGEP
### Numerical Results

#### Runtime and Speedup

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Xeon E3-1245</th>
<th></th>
<th>Dual Xeon E5-2690 - MKL 10.2</th>
<th></th>
<th></th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QZ</td>
<td>4 Thr.</td>
<td>QZ</td>
<td>1 Thr.</td>
<td>16 Thr.</td>
<td></td>
</tr>
<tr>
<td>(a)</td>
<td>1.31s</td>
<td>0.59s</td>
<td>1.75s</td>
<td>1.16s</td>
<td>0.51s</td>
<td>3.57</td>
</tr>
<tr>
<td>(b)</td>
<td>17.27s</td>
<td>10.48s</td>
<td>18.99s</td>
<td>22.68s</td>
<td>6.29s</td>
<td>3.02</td>
</tr>
<tr>
<td>(c)</td>
<td>40.16s</td>
<td>15.05s</td>
<td>39.86s</td>
<td>32.47s</td>
<td>8.16s</td>
<td>4.88</td>
</tr>
<tr>
<td>(d)</td>
<td>46.77s</td>
<td>43.09s</td>
<td>64.38s</td>
<td>86.90s</td>
<td>25.69s</td>
<td>2.51</td>
</tr>
<tr>
<td>(e)</td>
<td>77.35s</td>
<td>28.38s</td>
<td>80.40s</td>
<td>67.40s</td>
<td>14.41s</td>
<td>4.68</td>
</tr>
<tr>
<td>(f)</td>
<td>616.05s</td>
<td>526.22s</td>
<td>740.78s</td>
<td>1189.69s</td>
<td>383.08s</td>
<td>1.93</td>
</tr>
<tr>
<td>(g)</td>
<td>3046.40s</td>
<td>1006.25s</td>
<td>3286.61s</td>
<td>2684.74s</td>
<td>598.35s</td>
<td>5.49</td>
</tr>
<tr>
<td>(h)</td>
<td>out of memory</td>
<td></td>
<td>255057s</td>
<td>207198s</td>
<td>38200s</td>
<td>6.68</td>
</tr>
</tbody>
</table>

→ our algorithm uses all available cores,
→ works even on “desktop” computers,
→ significantly faster, even though already the first step of DSCQZ is theoretically more expensive than the entire QZ algorithm only counting the floating point operations involved.
## Numerical Results

### Runtime and Speedup

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Xeon E3-1245</th>
<th>Dual Xeon E5-2690 - MKL 10.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>1.31s</td>
<td>0.59s</td>
</tr>
<tr>
<td>(b)</td>
<td>17.27s</td>
<td>1.75s</td>
</tr>
<tr>
<td>(c)</td>
<td>40.16s</td>
<td>1.16s</td>
</tr>
<tr>
<td>(d)</td>
<td>46.77s</td>
<td>0.51s</td>
</tr>
<tr>
<td>(e)</td>
<td>77.35s</td>
<td>16.20KWh (≈ 2.95d · 225W)</td>
</tr>
<tr>
<td>(f)</td>
<td>616.05s</td>
<td>4.24KWh (≈ 10.6h · 400W)</td>
</tr>
<tr>
<td>(g)</td>
<td>3046.40s</td>
<td>526.22s</td>
</tr>
<tr>
<td>(h)</td>
<td>out of memory</td>
<td>740.78s</td>
</tr>
</tbody>
</table>

Reduce the runtime from \(\approx 3\) days to \(\approx 10.6\) hours.

**Power Consumption:**
- QZ: 16.20KWh (≈ 2.95d · 225W)
- DSCQZ: 4.24KWh (≈ 10.6h · 400W)

→ save 74% energy!

→ our algorithm uses all available cores,
→ works even on “desktop” computers,
→ significantly faster, even though already the first step of DSCQZ is theoretically more expensive than the entire QZ algorithm only counting the floating point operations involved.
We assume that QZ gives the correct result and define a global error:

\[
err_{\text{global}}(A, B) := \frac{\|\Lambda^{QZ}(A, B) - \Lambda^{DSCQZ}(A, B)\|_2}{\|\Lambda^{QZ}(A, B)\|_2}
\]

and local error

\[
err_{\text{local}}(A, B) := \max_{i=1,\ldots,n} \frac{|\lambda_i^{QZ}(A, B) - \lambda_i^{DSCQZ}(A, B)|}{|\lambda_i^{QZ}(A, B)|}
\]

for the eigenvalues of \((A, B)\).
### Numerical Results

#### Accuracy

<table>
<thead>
<tr>
<th>Matrix</th>
<th>$\text{err}_{\text{global}}(A, B)$</th>
<th>$\text{err}_{\text{local}}(A, B)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>3.10 e-10</td>
<td>3.15 e-10</td>
</tr>
<tr>
<td>(b)</td>
<td>4.63 e-13</td>
<td>4.40 e-11</td>
</tr>
<tr>
<td>(c)</td>
<td>1.39 e-14</td>
<td>3.77 e-12</td>
</tr>
<tr>
<td>(d)</td>
<td>4.62 e-15</td>
<td>9.44 e-09</td>
</tr>
<tr>
<td>(e)</td>
<td>7.60 e-15</td>
<td>5.32 e-11</td>
</tr>
<tr>
<td>(f)</td>
<td>6.17 e-15</td>
<td>1.72 e-10</td>
</tr>
<tr>
<td>(g)</td>
<td>1.71 e-14</td>
<td>1.06 e-10</td>
</tr>
<tr>
<td>(h)</td>
<td>5.21 e-14</td>
<td>1.02 e-09</td>
</tr>
</tbody>
</table>

Inaccuracy is caused by the iterative nature of the Newton iteration, but still acceptable for many applications. Increase accuracy for single eigenvalues using the inverse iteration.
## Numerical Results

### Accuracy

<table>
<thead>
<tr>
<th>Matrix</th>
<th>$err_{global}(A, B)$</th>
<th>$err_{local}(A, B)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>3.10 e-10</td>
<td>3.15 e-10</td>
</tr>
<tr>
<td>(b)</td>
<td>4.63 e-13</td>
<td>4.40 e-11</td>
</tr>
<tr>
<td>(c)</td>
<td>1.39 e-14</td>
<td>3.77 e-12</td>
</tr>
<tr>
<td>(d)</td>
<td>4.62 e-15</td>
<td>9.44 e-09</td>
</tr>
<tr>
<td>(e)</td>
<td>7.60 e-15</td>
<td>5.32 e-11</td>
</tr>
<tr>
<td>(f)</td>
<td>6.17 e-15</td>
<td>1.72 e-10</td>
</tr>
<tr>
<td>(g)</td>
<td>1.71 e-14</td>
<td>1.06 e-10</td>
</tr>
<tr>
<td>(h)</td>
<td>5.21 e-14</td>
<td>1.02 e-09</td>
</tr>
</tbody>
</table>

→ Inaccuracy is caused by the iterative nature of the Newton iteration,
→ But still acceptable for many applications.
→ Increase accuracy for single eigenvalues using the inverse iteration.
Conclusions
We have seen that:

- We can formulate a level-3 BLAS based solver for the NGEP,
- The new solver scales on multicore architectures,
- The level-3 BLAS operations make extensive use of the vector registers, (→ see 1 thread results)
- We get a acceptable approximation of the NGEP in drastically reduced time.

Further Research:
- Include more parallelism from the recursive structure
- Use properties of NUMA architectures to share the work
- Develop a hybrid CPU-Accelerator implementation
- Improve robustness (→ develop fall back situations if the DSCQZ algorithm fails.)

Thank you for your attention!

Questions?
Conclusions

We have seen that:

- We can formulate a level-3 BLAS based solver for the NGEP,
- The new solver scales on multicore architectures,
- The level-3 BLAS operations make extensive use of the vector registers, (→ see 1 thread results)
- We get a acceptable approximation of the NGEP in drastically reduced time.

Further Research:

- Include more parallelism from the recursive structure
  → use properties of NUMA architectures to share the work,
- Develop a hybrid CPU-Accelerator implementation,
- Improve robustness
  → develop fall back situations if the DSCQZ algorithm fails.
## Conclusions

We have seen that:

- We can formulate a level-3 BLAS based solver for the NGEP,
- The new solver scales on multicore architectures,
- The level-3 BLAS operations make extensive use of the vector registers, (→ see 1 thread results)
- We get an acceptable approximation of the NGEP in drastically reduced time.

Further Research:

- Include more parallelism from the recursive structure (→ use properties of NUMA architectures to share the work),
- Develop a hybrid CPU-Accelerator implementation,
- Improve robustness (→ develop fall back situations if the DSCQZ algorithm fails).

Thank you for your attention!

Questions?