ParCo2013 - September 10-13, 2013

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

MAX PLANCK INSTITUTE FOR DYNAMICS OF COMPLEX TECHNICAL SYSTEMS MAGDEBURG

Outline

Motivation

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 $x \in \mathbb{R}^n$.

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*:

$$\underbrace{Q^H AZ}_{S}\, y = \lambda \underbrace{Q^H BZ}_{T}\, y,$$

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.

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*:

$$\underbrace{Q^{T} A Z}_{S} y = \lambda \underbrace{Q^{T} B Z}_{T} y,$$

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.

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*:

$$\underbrace{Q^T AZ}_{S} y = \lambda \underbrace{Q^T BZ}_{T} y,$$

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

$$\begin{aligned} AR - LB &= C \\ DR - LE &= F, \end{aligned}$$

- Various analysis methods for dynamical systems,
- …

Motivation

QZ Algorithm

[Moler, Stewart '73]

Common way to compute the generalized Schur decomposition:

Motivation

QZ Algorithm

[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.

Motivation

QZ Algorithm

[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.

`DGEQRF` provides a level-3 BLAS implementation. ☺

Motivation

QZ Algorithm

[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

Givens-Rotations are only level-1 BLAS operations. ☹

Motivation

QZ Algorithm

[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 $\tilde{A} = Q^T A$.

   Sequences of Givens-Rotations ☹☹

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.

Motivation

QZ Algorithm

[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:

		Intel® MKL 11.0			OpenBLAS 0.2.8		
Matrix	dim.	1 Th.	8 Th.	16 Th.	1 Th.	8 Th.	16 Th.
rbs480	480	1.23s	1.10s	1.23s	1.38s	2.07s	2.41s
bsst09	1083	16.28s	16.29s	16.46s	16.90s	16.89s	17.13s
peec	1434	40.36s	39.90s	40.01s	41.07s	41.08s	44.86s
bsst11	1473	48.07s	47.49s	47.48s	48.82s	48.17s	53.44s

Motivation

QZ on Multicore Architectures

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

Matrix	dim.	Intel® MKL 11.0			OpenBLAS 0.2.8		
		1 Th.	8 Th.	16 Th.	1 Th.	8 Th.	16 Th.
rbs480	480	1.00	1.12	1.00	1.00	**0.66**	**0.57**
bsst09	1083	1.00	1.00	**0.99**	1.00	1.00	**0.99**
peec	1434	1.00	1.01	1.01	1.00	1.00	**0.92**
bsst11	1473	1.00	1.01	1.01	1.00	1.01	**0.91**

Motivation

QZ on Multicore Architectures

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

Matrix	dim.	Intel® MKL 11.0			OpenBLAS 0.2.8		
		1 Th.	8 Th.	16 Th.	1 Th.	8 Th.	16 Th.
rbs480	480	1.00	1.12	1.00	1.00	**0.66**	**0.57**
bsst09	1083	1.00	1.00	**0.99**	1.00	1.00	**0.99**
peec	1434	1.00	1.01	1.01	1.00	1.00	**0.92**
bsst11	1473	1.00	1.01	1.01	1.00	1.01	**0.91**

$\rightarrow$ 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:

Matrix	dim.	Intel® MKL 11.0			OpenBLAS 0.2.8		
		1 Th.	8 Th.	16 Th.	1 Th.	8 Th.	16 Th.
rbs480	480	1.00	1.12	1.00	1.00	**0.66**	**0.57**
bsst09	1083	1.00	1.00	**0.99**	1.00	1.00	**0.99**
peec	1434	1.00	1.01	1.01	1.00	1.00	**0.92**
bsst11	1473	1.00	1.01	1.01	1.00	1.01	**0.91**

$\rightarrow$ No acceleration using parallel BLAS at all.

$\rightarrow$ We need a new and faster way to approximate the generalized Schur decomposition on current hardware.

Spectral Division and the Sign Function

Spectral Division and the Sign Function

Spectral Division

From the block generalized Schur form:

$$\underbrace{\begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix}}_{Q^T} A \underbrace{\begin{pmatrix} Z_1 & Z_2 \end{pmatrix}}_{Z} = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}$$

and

$$\underbrace{\begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix}}_{Q^T} B \underbrace{\begin{pmatrix} Z_1 & Z_2 \end{pmatrix}}_{Z} = \begin{pmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{pmatrix},$$

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:

$$\underbrace{\begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix}}_{Q^T} A \underbrace{\begin{pmatrix} Z_1 & Z_2 \end{pmatrix}}_{Z} = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}$$

and

$$\underbrace{\begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix}}_{Q^T} B \underbrace{\begin{pmatrix} Z_1 & Z_2 \end{pmatrix}}_{Z} = \begin{pmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{pmatrix},$$

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:

$$\underbrace{\begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix}}_{Q^T} A \underbrace{\begin{pmatrix} Z_1 & Z_2 \end{pmatrix}}_{Z} = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}$$

and

$$\underbrace{\begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix}}_{Q^T} B \underbrace{\begin{pmatrix} Z_1 & Z_2 \end{pmatrix}}_{Z} = \begin{pmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{pmatrix},$$

Problem:

Find a way to compute two orthogonal matrices $Q = [Q_1, Q_2]$ and $Z = [Z_1, Z_2]$ by using parallely scaling level-3 BLAS operations.

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

(Generalized) Sign Function

Matrix Sign Function

Let $Y \operatorname{diag}(J_1, J_2) Y^{-1} = A$ 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

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

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 $Y \operatorname{diag}(J_1, J_2) Y^{-1} = A$ 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

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

is the *sign* of the matrix A, where $\dim(I_i) = \dim(J_i)$, $i = 1, 2$.

Some properties:

- $\operatorname{Range}(I + \operatorname{sign}(A))$ is the subspace corresponding to all eigenvalues with positive real part.
- $\operatorname{sign}(A)^2 = I$

Spectral Division and the Sign Function

(Generalized) Sign Function

[Gardiner, Laub'86]

From $\operatorname{sign}(A)^2 = I$ 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.

Spectral Division and the Sign Function

(Generalized) Sign Function

[Gardiner, Laub'86]

The Generalized Sign function iteration:

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

Spectral Division and the Sign Function

(Generalized) Sign Function

[Gardiner, Laub'86]

The Generalized Sign function iteration:

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

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

Spectral Division and the Sign Function

(Generalized) Sign Function

[Gardiner, Laub'86]

The Generalized Sign function iteration:

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

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

Properties change to:

- $\operatorname{Range}\left(B + \operatorname{sign}(A, B)\right)$ is the right deflating subspace corresponding to all eigenvalues with positive real part.
- $\operatorname{Range}\left(B - \operatorname{sign}(A, B)\right)$ is the right deflating subspace corresponding to all eigenvalues with negative real part.

Spectral Division and the Sign Function

(Generalized) Sign Function

[Gardiner, Laub'86]

The Generalized Sign function iteration:

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

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

Observations:

- 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.

Properties change to:

- Range $(B + \operatorname{sign}(A, B))$ is the right deflating subspace corresponding to all eigenvalues with positive real part.
- Range $(B - \operatorname{sign}(A, B))$ 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

[Sun, Quintana-Ortí '04]

Questions:

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

Computation of Z: From the range properties follows:

$$(B + \operatorname{sign}(A, B))Z_1 = 0 \quad \text{and} \quad (B + \operatorname{sign}(A, B))Z_2 = K$$

Spectral Division and the Sign Function

Spectral Division using the Sign Function [Sun, Quintana-Ortí '04]

Questions:

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

Computation of Z: From the range properties follows:

$$(B + \operatorname{sign}(A, B))[Z_1, Z_2] = [0, K]$$

Spectral Division and the Sign Function

Spectral Division using the Sign Function [Sun, Quintana-Ortí '04]

Questions:

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

Computation of Z: From the range properties follows:

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

Spectral Division and the Sign Function

Spectral Division using the Sign Function

[Sun, Quintana-Ortí '04]

Questions:

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

Computation of Z: From the range properties follows:

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

Spectral Division and the Sign Function

Spectral Division using the Sign Function

[Sun, Quintana-Ortí '04]

Questions:

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

Computation of Z: From the range properties follows:

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

$\rightarrow$ use a Rank Revealing QR Decomposition (RRQR)

Spectral Division and the Sign Function

Spectral Division using the Sign Function

[Sun, Quintana-Ortí '04]

Questions:

1. How to contruct 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 $AZ_1 + BZ_1$.

Spectral Division and the Sign Function

Spectral Division using the Sign Function

[Sun, Quintana-Ortí '04]

Questions:

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

Computation of Q:

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

Spectral Division and the Sign Function

Spectral Division using the Sign Function

[Sun, Quintana-Ortí '04]

Questions:

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

Computation of Q:

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

$\rightarrow$ use a RRQR procedure again.

Spectral Division and the Sign Function

Spectral Division using the Sign Function

[Sun, Quintana-Ortí '04]

Questions:

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

Computation of Q:

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

$\rightarrow$ use a RRQR procedure again.

We can compute Q and Z from $\operatorname{sign}(A, B)$ using two RRQR procedures.
$\rightarrow$ use level-3 subroutine `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 $S = \operatorname{sign}(A,B)$ 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: ... 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 $S = \operatorname{sign}(A,B)$ 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:

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

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	
minimum:	0 Flops
maximum:	$8n^3$ Flops
Transform A and B:	$8n^3$ Flops

$\rightarrow$ **more than QZ**

$\rightarrow$ **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

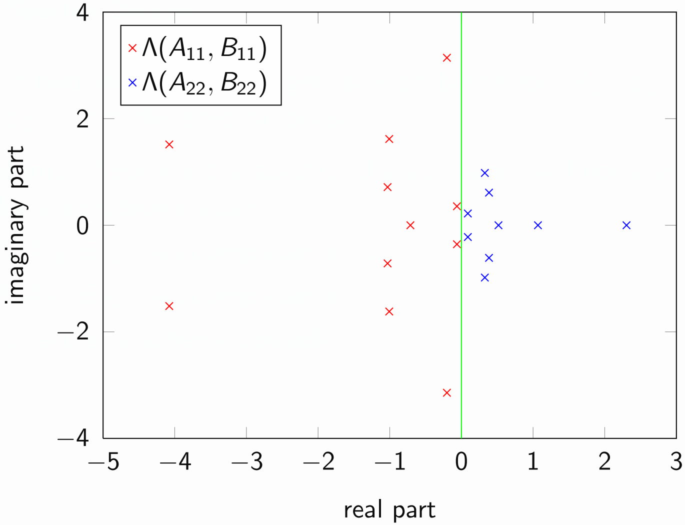
We got **two independent** eigenvalue problems for (A_{11}, B_{11}) and (A_{22}, B_{22}) from the spectral division.

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.

Original Spectrum:



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.

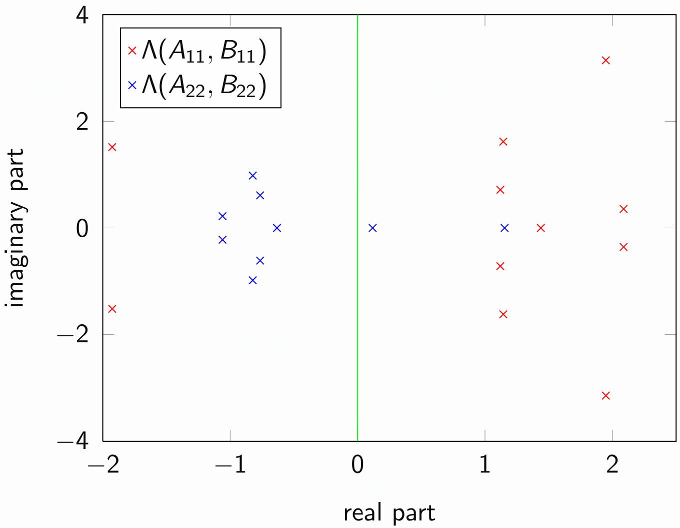
Idea

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

Recursive

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 θ_*: Chose θ_- or respectively θ_+ 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 θ_*: Chose θ_- or respectively θ_+ 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 θ_* requires the knowledge of all eigenvalues.

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 θ_- is obviously given by

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

where λ_{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 θ_- is obviously given by

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

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

Cheap approximation of $\Re(\lambda_{\mathrm{left}})$:

$$-\Re(\lambda_{\mathrm{left}}) \leq \rho(A_{11}, B_{11})$$

where $\rho(A_{11}, B_{11})$ is the spectral radius of 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 θ_- is obviously given by

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

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

Cheap approximation of $\Re(\lambda_{\mathrm{left}})$:

$$-\Re(\lambda_{\mathrm{left}}) \;\leq\; \rho(A_{11}, B_{11}) \leq \; \|B_{11}^{-1}A_{11}\|_2$$

where $\rho(A_{11}, B_{11})$ is the spectral radius of 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 θ_- is obviously given by

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

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

Cheap approximation of $\Re(\lambda_{\mathrm{left}})$:

$$-\Re(\lambda_{\mathrm{left}}) \;\leq\; \rho(A_{11}, B_{11}) \leq \; \|B_{11}^{-1}A_{11}\|_2 \leq \; \|B_{11}^{-1}A_{11}\|_F$$

where $\rho(A_{11}, B_{11})$ is the spectral radius of 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:

Algorithm 2 `[Q,Z] = dscqz(A,B)`

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

Output: $(Q^T AZ, Q^T BZ)$ in real Schur form.

1: **if** (A, B) is trivial to solve **then**
2: Compute Q, Z directly and return them.
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: $[\tilde{Q}_1, \tilde{Z}_1]=\mathtt{dscqz}(A_{11} - \theta_- B_{11}, B_{11})$.
7: $[\tilde{Q}_2, \tilde{Z}_2]=\mathtt{dscqz}(A_{22} - \theta_+ B_{22}, B_{22})$.
8: Update $Q := Q \begin{pmatrix} \tilde{Q}_1 & 0 \\ 0 & \tilde{Q}_2 \end{pmatrix}$ and $Z := Z \begin{pmatrix} \tilde{Z}_1 & 0 \\ 0 & \tilde{Z}_2 \end{pmatrix}$.
9: **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:

Algorithm 2 `[Q,Z] = dscqz(A,B)`

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

Output: $(Q^T AZ, Q^T BZ)$ in real Schur form.

if (A, B) is trivial to solve **then**
    Compute Q, Z directly and return them.
end if
Compute Q and Z using Algorithm 1 and transform (A, B).
Set $\theta_- = -\frac{1}{2}\|B_{11}^{-1}A_{11}\|_F$ and $\theta_+ = \frac{1}{2}\|B_{22}^{-1}A_{22}\|_F$.
`[`$\tilde{Q}_1, \tilde{Z}_1$`]=dscqz(`$A_{11} - \theta_- B_{11}$`,`B_{11}`).`
`[`$\tilde{Q}_2, \tilde{Z}_2$`]=dscqz(`$A_{22} - \theta_+ B_{22}$`,`B_{22}`).`
Update $Q := Q \begin{pmatrix} \tilde{Q}_1 & 0 \\ 0 & \tilde{Q}_2 \end{pmatrix}$ and $Z := Z \begin{pmatrix} \tilde{Z}_1 & 0 \\ 0 & \tilde{Z}_2 \end{pmatrix}$.
return `[`Q`,`Z`]`

Trivial: The Schur form can be computed directly, i.e. the problem is of size 1×1 or 2×2.

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.
$\rightarrow$ We pass $|\theta_-|$ and $|\theta_+|$ as spectral radius θ 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.
$\rightarrow$ We pass $|\theta_-|$ and $|\theta_+|$ as spectral radius θ 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.
$\rightarrow$ We can guarantee $\theta_* \rightarrow 0$ during the recursion.

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.

$\rightarrow$ Done using a queue.
$\rightarrow$ Restrict the additional memory to $4n^2 + 2n$.
$\rightarrow$ 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_{triv} given by:

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

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.

$\rightarrow$ only level-3 operations, use a threaded BLAS library
$\rightarrow$ requires the whole memory bandwidth

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

$\rightarrow$ OpenMP, PThreads,...
$\rightarrow$ n_{triv} is hardware dependent.
$\rightarrow$ reduce the transfers between cache and main memory.

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.
2. Solve the remaining trivial problems in parallel. Each problem is solved by one CPU-core in single-threaded mode.
3. Update $Q := Q \operatorname{diag}(Q_1, Q_2, \ldots)$ and $Z := \operatorname{diag}(Z_1, Z_2, \ldots)$ 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:

	Compue-Server Xeon E5-2690	Workstation Xeon E3-1245
CPU:	Dual Xeon E5-2690 @ 2.9 GHz	Xeon E3-1245 @ 3.3GHz
Cores:	16 (2×8)	4
L2 Cache:	256KiB	256KiB
n_{triv}	90	90
RAM:	32 GiB DDR3	8 GiB DDR3
OS:	Ubuntu 12.04	Ubuntu 12.04
Compiler:	GCC 4.6.3	GCC 4.6.3
BLAS:	Intel MKL 10.2	Intel MKL 10.2

Test matrices from MatrixMarket and the Oberwolfach Collection:

	Name	Dimension		Name	Dimension
(a)	rbs480	480	(b)	bsst09	1 083
(c)	spiral inductor	1434	(d)	bcsst11	1 473
(e)	filter2D	1 668	(f)	bcsst21	3 600
(g)	steel profile	5 177	(h)	steel profile	20 209

Numerical Results

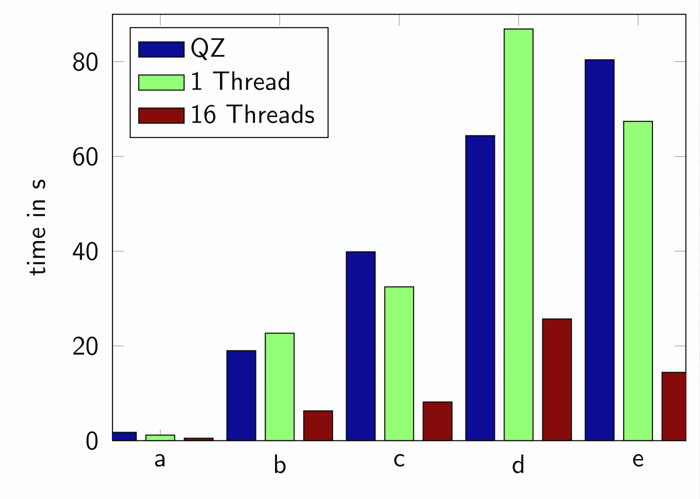
Runtime and Speedup

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

Numerical Results

Runtime and Speedup

Runtime:



Numerical Results

Runtime and Speedup

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

→ 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

Matrix	Xeon E3-1245		Dual Xeon E5-2690 - MKL 10.2			
	QZ	4 Thr	QZ	1 Thr	16 Thr	Speedup
(a)	1.31s	0.59s	1.75s	1.16s	0.51s	3.57
(b)	17.27s	10.48s	18.99s	22.68s	6.29s	3.02
(c)	40.16s	15.05s	39.86s	32.47s	8.16s	4.88
(d)	46.77s	43.09s	64.38s	86.90s	25.69s	2.51
(e)	77.35s	28.58s	80.4[illegible]s	67.40s	14.41s	4.68
(f)	616.05s	526.22s	740.78s	1189.69s	383.08s	1.93
(g)	3046.40s	1006.25s	3286.61s	2684.74s	59[illegible].35s	5.49
(h)	out of	memory	$\boxed{\textbf{255057s}}$	207198s	$\boxed{\textbf{38200s}}$	6.68

Reduce the runtime from ≈ **3 days** to ≈ **10.6 hours.**

Power Consumption:

QZ: 16.20KWh $(= 2.95d \cdot 225W)$

DSCQZ: 4.24KWh $(= 10.6h \cdot 400W)$

→ **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.

Numerical Results

Accuracy

We assume that QZ gives the correct result and define a global error:

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

and local error

$$err_{local}(A, B) := \max_{i=1,\dots,n} \frac{|\lambda_i^{QZ}(A, B) - \lambda_i^{DSCQZ}(A, B)|}{|\lambda_i^{QZ}(A, B)|}$$

for the eigenvalues of (A, B).

Numerical Results

Accuracy

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

Numerical Results

Accuracy

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

$\rightarrow$ Inaccuracy is caused by the iterative nature of the Newton iteration,

$\rightarrow$ But still acceptable for many applications.

$\rightarrow$ Increase accuracy for single eigenvalues using the inverse iteration.

Conclusions

Conclusions

We have see 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, ($\rightarrow$ see 1 thread results)
- We get a acceptable approximation of the NGEP in drastically reduced time.

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, ($\rightarrow$ 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
  $\rightarrow$ use properties of NUMA architectures to share the work,
- Develop a hybrid CPU-Accelerator implementation,
- Improve robustness
  $\rightarrow$ 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 a acceptable approximation of the NGEP in drastically reduced time.

Thank you for your attention!

Questions?

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.