

Fast computation of eigenvalues of companion, comrade, and related matrices

David S. Watkins

Department of Mathematics
Washington State University

Providence, June 2013

This is joint work with

- **Raf Vandebril** (KU Leuven)
- **Jared Aurentz** (WSU)

Zeros of a polynomial (monomial basis)

- $p(z) = z^n - c_1z^{n-1} - c_2z^{n-2} - \dots - c_n = 0$
- companion matrix

$$A = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \\ 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 & 0 \end{bmatrix}$$

- Get the zeros of p by computing the eigenvalues of A .
- Example: MATLAB's `roots` command

Zeros of a polynomial (monomial basis)

- $p(z) = z^n - c_1z^{n-1} - c_2z^{n-2} - \dots - c_n = 0$
- companion matrix

$$A = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \\ 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 & 0 \end{bmatrix}$$

- Get the zeros of p by computing the eigenvalues of A .
- Example: MATLAB's `roots` command

Zeros of a polynomial (monomial basis)

- $p(z) = z^n - c_1z^{n-1} - c_2z^{n-2} - \dots - c_n = 0$
- companion matrix

$$A = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \\ 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 & 0 \end{bmatrix}$$

- Get the zeros of p by computing the eigenvalues of A .
- Example: MATLAB's `roots` command

Zeros of a polynomial (monomial basis)

- $p(z) = z^n - c_1z^{n-1} - c_2z^{n-2} - \dots - c_n = 0$
- companion matrix

$$A = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \\ 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 & 0 \end{bmatrix}$$

- Get the zeros of p by computing the eigenvalues of A .
- Example: MATLAB's `roots` command

Zeros of a polynomial (monomial basis)

- $p(z) = z^n - c_1z^{n-1} - c_2z^{n-2} - \dots - c_n = 0$
- companion matrix

$$A = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \\ 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 & 0 \end{bmatrix}$$

- Get the zeros of p by computing the eigenvalues of A .
- Example: MATLAB's `roots` command

Zeros of a polynomial (“arbitrary” basis)

- $p(z) = p_n(z) - c_1 p_{n-1}(z) - c_2 p_{n-2}(z) - \cdots - c_n p_0(z) = 0$
- confederate matrix (Barnett 1983)

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

- $p_k(z)b_{k+1,k} = zp_{k-1}(z) - \sum_{j=1}^k p_{j-1}(z)b_{jk}$
- Short recurrence implies sparse matrix.

Zeros of a polynomial (“arbitrary” basis)

- $p(z) = p_n(z) - c_1 p_{n-1}(z) - c_2 p_{n-2}(z) - \cdots - c_n p_0(z) = 0$
- confederate matrix (Barnett 1983)

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

- $p_k(z)b_{k+1,k} = zp_{k-1}(z) - \sum_{j=1}^k p_{j-1}(z)b_{jk}$
- Short recurrence implies sparse matrix.

Zeros of a polynomial (“arbitrary” basis)

- $p(z) = p_n(z) - c_1 p_{n-1}(z) - c_2 p_{n-2}(z) - \cdots - c_n p_0(z) = 0$
- confederate matrix (Barnett 1983)

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

- $p_k(z)b_{k+1,k} = zp_{k-1}(z) - \sum_{j=1}^k p_{j-1}(z)b_{jk}$
- Short recurrence implies sparse matrix.

Zeros of a polynomial (“arbitrary” basis)

- $p(z) = p_n(z) - c_1 p_{n-1}(z) - c_2 p_{n-2}(z) - \cdots - c_n p_0(z) = 0$
- confederate matrix (Barnett 1983)

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

- $p_k(z)b_{k+1,k} = zp_{k-1}(z) - \sum_{j=1}^k p_{j-1}(z)b_{jk}$
- Short recurrence implies sparse matrix.

Zeros of a polynomial (“arbitrary” basis)

- $p(z) = p_n(z) - c_1 p_{n-1}(z) - c_2 p_{n-2}(z) - \cdots - c_n p_0(z) = 0$
- confederate matrix (Barnett 1983)

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

- $p_k(z)b_{k+1,k} = zp_{k-1}(z) - \sum_{j=1}^k p_{j-1}(z)b_{jk}$
- Short recurrence implies sparse matrix.

Zeros of a polynomial (3-term recurrence)

- $p_k(z)\alpha_k = (z - \beta_k)p_{k-1}(z) - \gamma_k p_{k-2}(z)$
- $p(z) = p_n(z) - c_1 p_{n-1}(z) - c_2 p_{n-2}(z) - \dots - c_n p_0(z) = 0$
- comrade matrix

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \\ & & & & & \times & \times \end{bmatrix}$$

- tridiagonal plus spike
- Examples: Chebyshev, Legendre, Hermite, ...
- Example: Chebfun's roots command

Zeros of a polynomial (3-term recurrence)

- $p_k(z)\alpha_k = (z - \beta_k)p_{k-1}(z) - \gamma_k p_{k-2}(z)$
- $p(z) = p_n(z) - c_1 p_{n-1}(z) - c_2 p_{n-2}(z) - \dots - c_n p_0(z) = 0$
- comrade matrix

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \\ & & & & & \times & \times \end{bmatrix}$$

- tridiagonal plus spike
- Examples: Chebyshev, Legendre, Hermite, ...
- Example: Chebfun's roots command

Zeros of a polynomial (3-term recurrence)

- $p_k(z)\alpha_k = (z - \beta_k)p_{k-1}(z) - \gamma_k p_{k-2}(z)$
- $p(z) = p_n(z) - c_1 p_{n-1}(z) - c_2 p_{n-2}(z) - \dots - c_n p_0(z) = 0$
- comrade matrix

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times & \times \\ & & & & & \times & \times \end{bmatrix}$$

- tridiagonal plus spike
- Examples: Chebyshev, Legendre, Hermite, ...
- Example: Chebfun's `roots` command

Zeros of a polynomial (3-term recurrence)

- $p_k(z)\alpha_k = (z - \beta_k)p_{k-1}(z) - \gamma_k p_{k-2}(z)$
- $p(z) = p_n(z) - c_1 p_{n-1}(z) - c_2 p_{n-2}(z) - \dots - c_n p_0(z) = 0$
- comrade matrix

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \\ & & & & & \times & \times \end{bmatrix}$$

- tridiagonal plus spike
- Examples: Chebyshev, Legendre, Hermite, ...
- Example: Chebfun's roots command

Zeros of a polynomial (2-term recurrence)

- Newton basis
- $p_k(z)\alpha_k = (z - \beta_k)p_{k-1}(z)$
- $p(z) = p_n(z) - c_1p_{n-1}(z) - c_2p_{n-2}(z) - \cdots - c_np_0(z) = 0$

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & & & \\ & & \times & \times & & \\ & & & \times & \times & \\ & & & & \times & \times \\ & & & & & \times & \times \end{bmatrix}$$

- bidiagonal plus spike

Zeros of a polynomial (2-term recurrence)

- Newton basis
- $p_k(z)\alpha_k = (z - \beta_k)p_{k-1}(z)$
- $p(z) = p_n(z) - c_1p_{n-1}(z) - c_2p_{n-2}(z) - \cdots - c_np_0(z) = 0$

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & & & \\ & & \times & \times & & \\ & & & \times & \times & \\ & & & & \times & \times \\ & & & & & \times & \times \end{bmatrix}$$

- bidiagonal plus spike

Zeros of a polynomial (2-term recurrence)

- Newton basis
- $p_k(z)\alpha_k = (z - \beta_k)p_{k-1}(z)$
- $p(z) = p_n(z) - c_1p_{n-1}(z) - c_2p_{n-2}(z) - \cdots - c_np_0(z) = 0$

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & & & \\ & & \times & \times & & \\ & & & \times & \times & \\ & & & & \times & \times \\ & & & & & \times & \times \end{bmatrix}$$

- bidiagonal plus spike

Zeros of a polynomial (1-term recurrence)

- monomial basis
- $p_k(z) = zp_{k-1}(z)$
- $p(z) = p_n(z) - c_1p_{n-1}(z) - c_2p_{n-2}(z) - \cdots - c_np_0(z) = 0$

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & & & & \\ & & \times & & & \\ & & & \times & & \\ & & & & \times & \\ & & & & & \times \end{bmatrix}$$

- one diagonal plus spike
- companion matrix

Zeros of a polynomial (1-term recurrence)

- monomial basis
- $p_k(z) = zp_{k-1}(z)$
- $p(z) = p_n(z) - c_1p_{n-1}(z) - c_2p_{n-2}(z) - \cdots - c_np_0(z) = 0$

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & & & & \\ & & \times & & & \\ & & & \times & & \\ & & & & \times & \\ & & & & & \times \end{bmatrix}$$

- one diagonal plus spike
- companion matrix

How do we compute the eigenvalues?

- narrow band plus spike

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

- Francis's bulge-chasing algorithm ("implicitly-shifted QR ")
- $O(n^3)$ time, $O(n^2)$ space
- Is there a more economical approach?

How do we compute the eigenvalues?

- narrow band plus spike

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

- Francis's bulge-chasing algorithm ("implicitly-shifted QR ")
- $O(n^3)$ time, $O(n^2)$ space
- Is there a more economical approach?

How do we compute the eigenvalues?

- narrow band plus spike

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

- Francis's bulge-chasing algorithm ("implicitly-shifted QR ")
- $O(n^3)$ time, $O(n^2)$ space
- Is there a more economical approach?

How do we compute the eigenvalues?

- narrow band plus spike

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

- Francis's bulge-chasing algorithm ("implicitly-shifted QR ")
- $O(n^3)$ time, $O(n^2)$ space
- Is there a more economical approach?

How do we compute the eigenvalues?

- banded plus spike

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

- Exploit rank structure (generators)
- We pursue a different approach.

How do we compute the eigenvalues?

- banded plus spike

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

- Exploit rank structure (generators)
- We pursue a different approach.

How do we compute the eigenvalues?

- banded plus spike

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

- Exploit rank structure (generators)
- We pursue a different approach.

We have developed great methods that are ...

- lightning fast
- unstable

We have developed great methods that are ...

- lightning fast
- unstable

We have developed great methods that are ...

- lightning fast
- unstable

A useful factorization (by Gaussian elimination)

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & & & & \\ & \times & \times & & & \\ & & \times & \times & & \\ & & & \times & \times & \\ & & & & \times & \times \end{bmatrix}$$

A useful factorization (by Gaussian elimination)

$$\left[\begin{array}{cccccc} \times & \times & \times & \times & \times & \times \\ \times & \times & & & & \\ & \times & \times & & & \\ & & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \end{array} \right] = \left[\begin{array}{cccccc} \times & \times & & & & \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & & & \\ & & \times & \times & & \\ & & & \times & \times & \\ & & & & \times & \times \\ & & & & & \times \end{array} \right]$$

A useful factorization (by Gaussian elimination)

$$\left[\begin{array}{cccccc} \times & \times & \times & \times & \times & \times \\ \times & \times & & & & \\ & \times & \times & & & \\ & & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \end{array} \right] = \left[\begin{array}{cccccc} \times & \times & & & & \\ 0 & \times & \times & \times & \times & \times \\ & \times & \times & & & \\ & & \times & \times & & \\ & & & \times & \times & \\ & & & & \times & \times \end{array} \right]$$

A useful factorization (by Gaussian elimination)

$$\begin{array}{c} \left. \begin{array}{c} \rightarrow \\ \rightarrow \end{array} \right\} \left[\begin{array}{cccccc} \times & \times & \times & \times & \times & \times \\ \times & \times & & & & \\ & \times & \times & & & \\ & & \times & \times & & \\ & & & \times & \times & \\ & & & & \times & \times \end{array} \right] = \left[\begin{array}{cccccc} \times & \times & & & & \\ & \times & \times & & & \\ & & \times & \times & \times & \times \\ & & & \times & \times & \\ & & & & \times & \times \\ & & & & & \times & \times \end{array} \right]$$

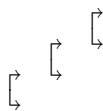
A useful factorization (by Gaussian elimination)

$$\begin{array}{l} \left. \begin{array}{l} \rightarrow \\ \rightarrow \end{array} \right\} \left[\begin{array}{cccccc} \times & \times & \times & \times & \times & \times \\ \times & \times & & & & \\ & \times & \times & & & \\ & & \times & \times & & \\ & & & \times & \times & \\ & & & & \times & \times \end{array} \right] = \left[\begin{array}{cccccc} \times & \times & & & & \\ & \times & \times & & & \\ & & \mathbf{0} & \times & \times & \times \\ & & & \times & \times & \\ & & & & \times & \times \\ & & & & & \times & \times \end{array} \right]$$

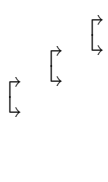
A useful factorization (by Gaussian elimination)

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & & & & \\ & \times & \times & & & \\ & & \times & \times & & \\ & & & \times & \times & \\ & & & & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & & & & \\ & \times & \times & & & \\ & & \times & \times & & \\ & & & \times & \times & \\ & & & & \times & \times \\ & & & & & \times & \times \end{bmatrix}$$

A useful factorization (by Gaussian elimination)


$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & & & & \\ & \times & \times & \times & \times & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & & & & \\ & \times & \times & \times & \times & \\ & & \times & \times & \times & \\ & & & 0 & \times & \times \\ & & & & \times & \times \\ & & & & & \times \end{bmatrix}$$

A useful factorization (by Gaussian elimination)


$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & & & & \\ & \times & \times & & & \\ & & \times & \times & & \\ & & & \times & \times & \\ & & & & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & & & & \\ & \times & \times & & & \\ & & \times & \times & & \\ & & & \times & \times & \\ & & & & \times & \times \\ & & & & & \times & \times \end{bmatrix}$$

- and so on ... to (banded) triangular form.

A useful factorization (by Gaussian elimination)

- We get the factorization

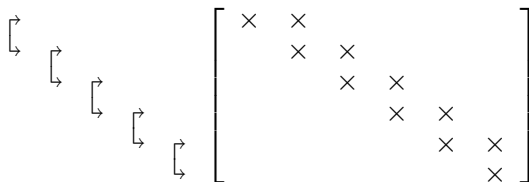
$$A = \begin{matrix} \left[\begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{array} \right] \end{matrix} \begin{bmatrix} \times & \times & & & & & \\ & \times & \times & & & & \\ & & \times & \times & & & \\ & & & \times & \times & & \\ & & & & \times & \times & \\ & & & & & \times & \times \\ & & & & & & \times \end{bmatrix}$$

- core transformations times banded upper triangular
- band width = length of recurrence

What core transformations look like

$$\begin{array}{l} \left[\begin{array}{l} \rightarrow \\ \rightarrow \\ \rightarrow \end{array} \right] \\ \left[\begin{array}{l} \rightarrow \\ \rightarrow \end{array} \right] \\ \left[\begin{array}{l} \rightarrow \\ \rightarrow \end{array} \right] \end{array} = \begin{bmatrix} \times & \times & & \\ \times & \times & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & \times & \times & \\ & \times & \times & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \times & \times \\ & & \times & \times \end{bmatrix}$$

Bulge Chasing Algorithm



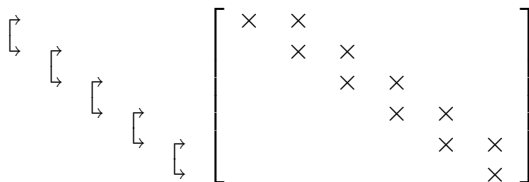
- We will develop a nonunitary bulge-chasing algorithm
- that preserves this factorization.
- Note: Storage is $O(n)$ (about $6n$ for this case).
- We have single-shift and double-shift variants.

Bulge Chasing Algorithm

$$\left[\begin{array}{cccccc} \times & & & & & \\ & \times & \times & & & \\ & & \times & \times & & \\ & & & \times & \times & \\ & & & & \times & \times \\ & & & & & \times & \times \\ & & & & & & \times \end{array} \right]$$

- We will develop a nonunitary bulge-chasing algorithm
- that preserves this factorization.
- Note: Storage is $O(n)$ (about $6n$ for this case).
- We have single-shift and double-shift variants.

Bulge Chasing Algorithm



The diagram illustrates the bulge-chasing algorithm. On the left, a series of five right-facing curly braces are arranged in a descending staircase pattern, representing the movement of a bulge through the matrix. On the right, a square matrix is shown with 'x' marks indicating non-zero entries. The matrix is upper triangular with a single 'x' in the top-left corner. The main diagonal contains 'x' marks at positions (1,1), (2,2), (3,3), (4,4), and (5,5). The sub-diagonal contains 'x' marks at positions (2,1), (3,2), (4,3), and (5,4). The matrix is enclosed in large square brackets.

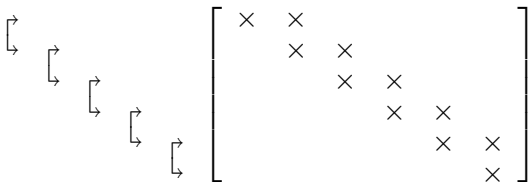
- We will develop a nonunitary bulge-chasing algorithm
- that preserves this factorization.
- Note: Storage is $O(n)$ (about $6n$ for this case).
- We have single-shift and double-shift variants.

Bulge Chasing Algorithm

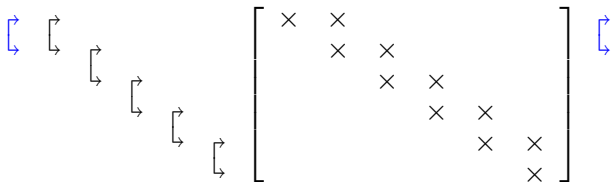
The diagram illustrates a sparse matrix factorization process. On the left, a series of five right-facing curly brackets are arranged in a descending staircase pattern, representing the movement of a bulge through the matrix. To the right of these brackets is a large square matrix enclosed in square brackets. The matrix is sparse, with 'x' marks indicating non-zero entries. The non-zero entries are located at the following positions (row, column): (1,1), (1,2), (2,2), (2,3), (3,3), (3,4), (4,4), (4,5), (5,5), and (5,6). This structure represents a banded matrix with a bulge that is being chased through the matrix.

- We will develop a nonunitary bulge-chasing algorithm
- that preserves this factorization.
- Note: Storage is $O(n)$ (about $6n$ for this case).
- We have single-shift and double-shift variants.

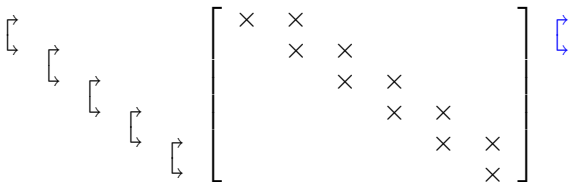
Bulge Chasing Algorithm (single-shift variant)



Bulge Chasing Algorithm (single-shift variant)



Bulge Chasing Algorithm (single-shift variant)



Passing Gauss transform through triangular matrix

- Blue core transformations are Gauss transforms.

$$\begin{bmatrix} \times & \times & \\ & \times & \times \\ & & \times \end{bmatrix} \begin{array}{c} \text{blue} \\ \text{core} \\ \text{transform} \end{array} = \begin{bmatrix} \times & \times & \\ \times & \times & \times \\ & & \times \end{bmatrix} = \begin{array}{c} \text{blue} \\ \text{core} \\ \text{transform} \end{array} \begin{bmatrix} \times & \times & \\ & \times & \times \\ & & \times \end{bmatrix}$$

- We do this at every opportunity.
- From this point on, we suppress the triangular matrix.

Passing Gauss transform through triangular matrix

- Blue core transformations are Gauss transforms.

$$\begin{bmatrix} \times & \times & \\ & \times & \times \\ & & \times \end{bmatrix} \begin{array}{c} \left[\begin{array}{c} \rightarrow \\ \rightarrow \end{array} \right] \\ \\ \end{array} = \begin{bmatrix} \times & \times & \\ \times & \times & \times \\ & & \times \end{bmatrix} = \begin{array}{c} \left[\begin{array}{c} \rightarrow \\ \rightarrow \end{array} \right] \\ \\ \end{array} \begin{bmatrix} \times & \times & \\ & \times & \times \\ & & \times \end{bmatrix}$$

- We do this at every opportunity.
- From this point on, we suppress the triangular matrix.

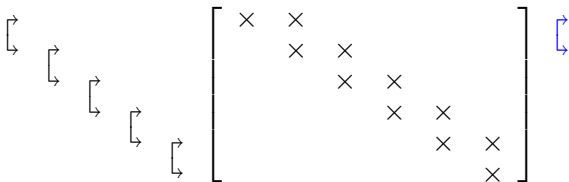
Passing Gauss transform through triangular matrix

- Blue core transformations are Gauss transforms.

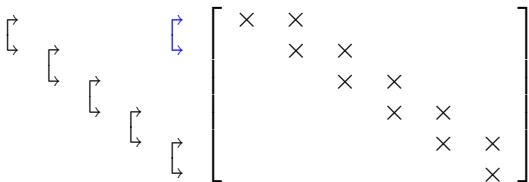
$$\begin{bmatrix} \times & \times & \\ & \times & \times \\ & & \times \end{bmatrix} \begin{array}{c} \left[\right. \\ \left. \right] \end{array} = \begin{bmatrix} \times & \times & \\ \times & \times & \times \\ & & \times \end{bmatrix} = \begin{array}{c} \left[\right. \\ \left. \right] \end{array} \begin{bmatrix} \times & \times & \\ & \times & \times \\ & & \times \end{bmatrix}$$

- We do this at every opportunity.
- From this point on, we suppress the triangular matrix.

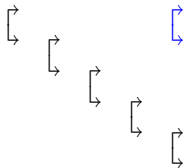
Bulge Chasing Algorithm (single-shift variant)



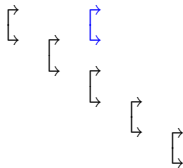
Bulge Chasing Algorithm (single-shift variant)



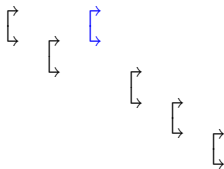
Bulge Chasing Algorithm (single-shift variant)



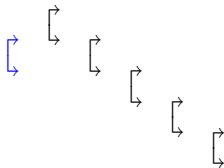
Bulge Chasing Algorithm (single-shift variant)



Bulge Chasing Algorithm (single-shift variant)



Bulge Chasing Algorithm (single-shift variant)



Turnover operation

- Can we do this?

$$\begin{array}{c} \left[\begin{array}{c} \rightarrow \\ \leftarrow \end{array} \right] \quad \left[\begin{array}{c} \rightarrow \\ \leftarrow \end{array} \right] \\ \left[\begin{array}{c} \rightarrow \\ \leftarrow \end{array} \right] \end{array} \quad \Leftrightarrow \quad \left[\begin{array}{c} \rightarrow \\ \leftarrow \end{array} \right] \quad \left[\begin{array}{c} \rightarrow \\ \leftarrow \end{array} \right] \quad \left[\begin{array}{c} \rightarrow \\ \leftarrow \end{array} \right]$$

- crucial question
- Pessimistic answer: No! (not always)
- Optimistic answer: Yes! (almost always)

Turnover operation

- Can we do this?

$$\begin{array}{c} \left[\begin{array}{c} \rightarrow \\ \leftarrow \end{array} \right] \quad \left[\begin{array}{c} \rightarrow \\ \leftarrow \end{array} \right] \\ \left[\begin{array}{c} \rightarrow \\ \leftarrow \end{array} \right] \end{array} \quad \Leftrightarrow \quad \left[\begin{array}{c} \rightarrow \\ \leftarrow \end{array} \right] \quad \left[\begin{array}{c} \rightarrow \\ \leftarrow \end{array} \right] \quad \left[\begin{array}{c} \rightarrow \\ \leftarrow \end{array} \right]$$

- crucial question
- Pessimistic answer: No! (not always)
- Optimistic answer: Yes! (almost always)

Turnover operation

- Can we do this?

$$\begin{bmatrix} \rightarrow \\ \rightarrow \end{bmatrix} \begin{bmatrix} \rightarrow \\ \rightarrow \end{bmatrix} \Leftrightarrow \begin{bmatrix} \rightarrow \\ \rightarrow \end{bmatrix} \begin{bmatrix} \rightarrow \\ \rightarrow \end{bmatrix}$$

- crucial question
- Pessimistic answer: No! (not always)
- Optimistic answer: Yes! (almost always)

Turnover operation

- Can we do this?

$$\begin{bmatrix} \rightarrow \\ \leftarrow \end{bmatrix} \begin{bmatrix} \rightarrow \\ \leftarrow \end{bmatrix} \begin{bmatrix} \rightarrow \\ \leftarrow \end{bmatrix} \Leftrightarrow \begin{bmatrix} \rightarrow \\ \leftarrow \end{bmatrix} \begin{bmatrix} \rightarrow \\ \leftarrow \end{bmatrix} \begin{bmatrix} \rightarrow \\ \leftarrow \end{bmatrix}$$

- crucial question
- Pessimistic answer: No! (not always)
- Optimistic answer: Yes! (almost always)

Turnover operation

$$\begin{array}{c} \left[\begin{array}{c} \rightarrow \\ \rightarrow \end{array} \right] \left[\begin{array}{c} \rightarrow \\ \rightarrow \end{array} \right] \left[\begin{array}{c} \rightarrow \\ \rightarrow \end{array} \right] = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \mathbf{0} & \times & \times \end{bmatrix} \left[\begin{array}{c} \rightarrow \\ \rightarrow \end{array} \right] = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \end{array}$$

- Red submatrix still has rank one.

Turnover operation

$$\begin{array}{c} \left[\begin{array}{c} \rightarrow \\ \rightarrow \end{array} \right] \left[\begin{array}{c} \rightarrow \\ \rightarrow \end{array} \right] = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \mathbf{0} & \times & \times \end{bmatrix} \left[\begin{array}{c} \rightarrow \\ \rightarrow \end{array} \right] = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \end{array}$$

- Red submatrix still has rank one.

Turnover operation

- So ...

$$\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}$$

- Gauss transform does not disturb 2×2 submatrix.

Turnover operation

- So ...

$$\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \end{bmatrix}$$

The diagram shows a sequence of operations on a 3x3 matrix. The first matrix has all entries marked with a red 'x'. An equals sign follows, then a blue double-headed arrow pointing to the second matrix. The second matrix has the bottom-left entry marked with a blue '0', while the other entries are red 'x's. This is followed by another equals sign, then a blue double-headed arrow pointing to a third matrix. The third matrix is identical to the second, but with a grey double-headed arrow above it pointing to the right, and another grey double-headed arrow to its right pointing downwards.

- Gauss transform does not disturb 2×2 submatrix.

Turnover operation

- So ...

$$\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}$$

The diagram shows a sequence of three 3x3 matrices. The first matrix has all entries marked with a red 'x'. The second matrix has the bottom-left entry marked with a blue '0', while the other entries are red 'x'. A blue double-headed arrow is positioned between the first and second matrices. The third matrix has all entries marked with a red 'x'. A blue double-headed arrow is positioned between the second and third matrices. A black double-headed arrow is positioned above the second and third matrices, and another black double-headed arrow is positioned to the right of the second and third matrices.

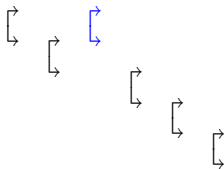
- Gauss transform does not disturb 2×2 submatrix.

Turnover operation

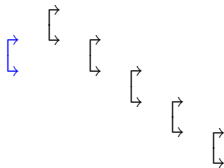
- Complete turnover looks like this:

$$\begin{array}{c} \left[\begin{array}{c} \rightarrow \\ \left[\begin{array}{c} \rightarrow \\ \left[\begin{array}{c} \rightarrow \\ \rightarrow \end{array} \end{array} \end{array} \end{array} \right] \end{array} \right] = \begin{array}{c} \left[\begin{array}{c} \rightarrow \\ \left[\begin{array}{c} \rightarrow \\ \left[\begin{array}{c} \rightarrow \\ \rightarrow \end{array} \end{array} \end{array} \right] \end{array} \right] \end{array}$$

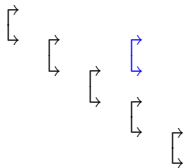
Bulge Chasing Algorithm (single-shift variant)



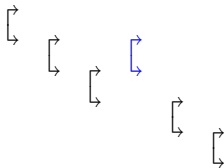
Bulge Chasing Algorithm (single-shift variant)



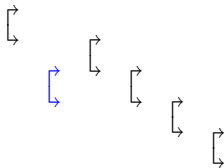
Bulge Chasing Algorithm (single-shift variant)



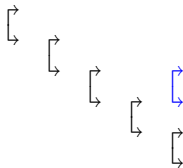
Bulge Chasing Algorithm (single-shift variant)



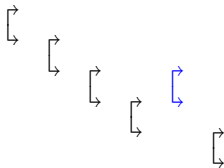
Bulge Chasing Algorithm (single-shift variant)



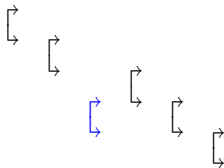
Bulge Chasing Algorithm (single-shift variant)



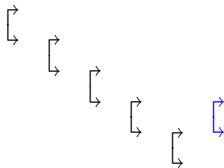
Bulge Chasing Algorithm (single-shift variant)



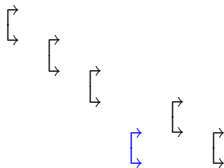
Bulge Chasing Algorithm (single-shift variant)



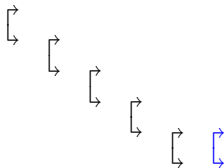
Bulge Chasing Algorithm (single-shift variant)



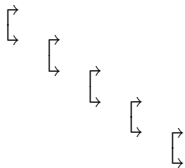
Bulge Chasing Algorithm (single-shift variant)



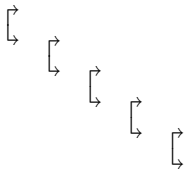
Bulge Chasing Algorithm (single-shift variant)



Bulge Chasing Algorithm (single-shift variant)

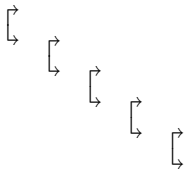


Bulge Chasing Algorithm (single-shift variant)



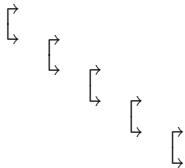
- Done!
- Now repeat again, again, and again

Bulge Chasing Algorithm (single-shift variant)



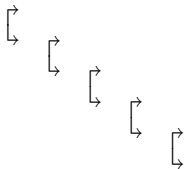
- Done!
- Now repeat again, again, and again

Bulge Chasing Algorithm (single-shift variant)



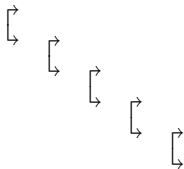
- Done!
- Now repeat again, again, and again

Bulge Chasing Algorithm (single-shift variant)



- Done!
- Now repeat again, again, and again

Bulge Chasing Algorithm (single-shift variant)



- Done!
- Now repeat again, again, and again

Computational complexity

- $O(n)$ storage (everything in cache)
- $O(n)$ flops per iteration
- $O(n)$ iterations
- $O(n^2)$ flops total

Computational complexity

- $O(n)$ storage (everything in cache)
- $O(n)$ flops per iteration
- $O(n)$ iterations
- $O(n^2)$ flops total

Computational complexity

- $O(n)$ storage (everything in cache)
- $O(n)$ flops per iteration
- $O(n)$ iterations
- $O(n^2)$ flops total

Computational complexity

- $O(n)$ storage (everything in cache)
- $O(n)$ flops per iteration
- $O(n)$ iterations
- $O(n^2)$ flops total

Computational complexity

- $O(n)$ storage (everything in cache)
- $O(n)$ flops per iteration
- $O(n)$ iterations
- $O(n^2)$ flops total

Stability issue

- This is a structured, implicitly-shifted LR algorithm.
- Unstable! Can we trust the results? No,
- but we can do *a posteriori* tests.
- $\|(\lambda I - A)v\| = |p(\lambda)|$
- $|\frac{p(\lambda)}{p'(\lambda)}|$ is estimate of error.
- Newton correction
- Cost: $O(n^2)$ flops for n roots.

- This is a structured, implicitly-shifted *LR* algorithm.
- Unstable! Can we trust the results? No,
- but we can do *a posteriori* tests.
- $\|(\lambda I - A)v\| = |p(\lambda)|$
- $|\frac{p(\lambda)}{p'(\lambda)}|$ is estimate of error.
- Newton correction
- Cost: $O(n^2)$ flops for n roots.

- This is a structured, implicitly-shifted *LR* algorithm.
- Unstable! Can we trust the results? No,
- but we can do *a posteriori* tests.
- $\|(\lambda I - A)v\| = |p(\lambda)|$
- $|\frac{p(\lambda)}{p'(\lambda)}|$ is estimate of error.
- Newton correction
- Cost: $O(n^2)$ flops for n roots.

- This is a structured, implicitly-shifted *LR* algorithm.
- Unstable! Can we trust the results? No,
- but we can do *a posteriori* tests.
- $\|(\lambda I - A)v\| = |p(\lambda)|$
- $|\frac{p(\lambda)}{p'(\lambda)}|$ is estimate of error.
- Newton correction
- Cost: $O(n^2)$ flops for n roots.

- This is a structured, implicitly-shifted LR algorithm.
- Unstable! Can we trust the results? No,
- but we can do *a posteriori* tests.
- $\|(\lambda I - A)v\| = |p(\lambda)|$
- $|\frac{p(\lambda)}{p'(\lambda)}|$ is estimate of error.
- Newton correction
- Cost: $O(n^2)$ flops for n roots.

- This is a structured, implicitly-shifted *LR* algorithm.
- Unstable! Can we trust the results? No,
- but we can do *a posteriori* tests.
- $\|(\lambda I - A)v\| = |p(\lambda)|$
- $|\frac{p(\lambda)}{p'(\lambda)}|$ is estimate of error.
- Newton correction
- Cost: $O(n^2)$ flops for n roots.

- This is a structured, implicitly-shifted *LR* algorithm.
- Unstable! Can we trust the results? No,
- but we can do *a posteriori* tests.
- $\|(\lambda I - A)v\| = |p(\lambda)|$
- $|\frac{p(\lambda)}{p'(\lambda)}|$ is estimate of error.
- Newton correction
- Cost: $O(n^2)$ flops for n roots.

- This is a structured, implicitly-shifted LR algorithm.
- Unstable! Can we trust the results? No,
- but we can do *a posteriori* tests.
- $\|(\lambda I - A)v\| = |p(\lambda)|$
- $|\frac{p(\lambda)}{p'(\lambda)}|$ is estimate of error.
- Newton correction
- Cost: $O(n^2)$ flops for n roots.

- This is a structured, implicitly-shifted LR algorithm.
- Unstable! Can we trust the results? No,
- but we can do *a posteriori* tests.
- $\|(\lambda I - A)v\| = |p(\lambda)|$
- $|\frac{p(\lambda)}{p'(\lambda)}|$ is estimate of error.
- Newton correction
- Cost: $O(n^2)$ flops for n roots.

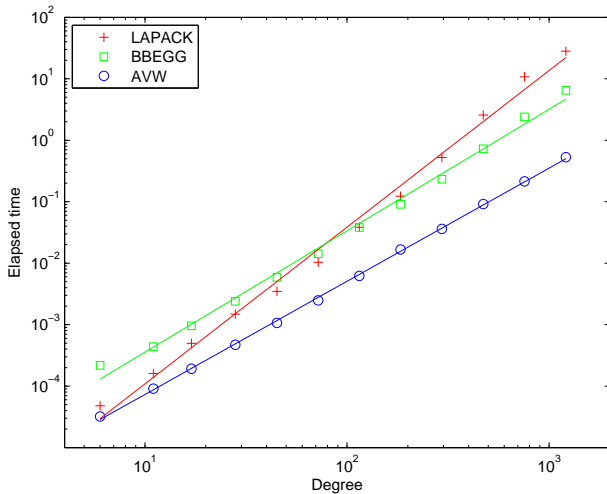
- This is a structured, implicitly-shifted LR algorithm.
- Unstable! Can we trust the results? No,
- but we can do *a posteriori* tests.
- $\|(\lambda I - A)v\| = |p(\lambda)|$
- $|\frac{p(\lambda)}{p'(\lambda)}|$ is estimate of error.
- Newton correction
- Cost: $O(n^2)$ flops for n roots.

First example: companion matrices, complex, single-shift code

Contestants

- LAPACK code ZHSEQR ($O(n^3)$)
- BBEGG (quasiseparable, unitary)
- AVW our code

Speed Comparison, companion case



Speed Comparison, companion case

- AVW times include *a posteriori* test times.
- No crossover point
- Slope = 1.84
- At $n = 1210$:

method	time
LAPACK	28.2
BBEGG	6.5
AVW	0.5

Speed Comparison, companion case

- AVW times include *a posteriori* test times.
- No crossover point
- Slope = 1.84
- At $n = 1210$:

method	time
LAPACK	28.2
BBEGG	6.5
AVW	0.5

Speed Comparison, companion case

- AVW times include *a posteriori* test times.
- No crossover point
- Slope = 1.84

- At $n = 1210$:

method	time
LAPACK	28.2
BBEGG	6.5
AVW	0.5

Accuracy Comparison, companion case

Maximum of residual $\|(\lambda I - A)v\|/\|A\|\|v\|$

degree	17	72	295	1210
LAPACK	1.1×10^{-14}	3.1×10^{-14}	6.8×10^{-14}	2.3×10^{-13}
BBEGG	1.5×10^{-14}	1.1×10^{-13}	4.1×10^{-12}	2.4×10^{-11}
AVW	7.4×10^{-12}	5.0×10^{-11}	3.7×10^{-10}	1.4×10^{-09}

Error estimate $|p(\lambda)/p'(\lambda)|$

degree	17	72	295	1210
LAPACK	4.3×10^{-15}	8.5×10^{-15}	1.4×10^{-14}	2.4×10^{-14}
BBEGG	2.4×10^{-14}	1.8×10^{-13}	1.3×10^{-12}	1.2×10^{-11}
AVW	4.7×10^{-12}	7.5×10^{-11}	5.4×10^{-11}	1.7×10^{-10}

After Newton correction

Maximum of residual $\|(\lambda I - A)v\|/\|A\| \|v\|$

degree	17	72	295	1210
LAPACK	8.1×10^{-16}	1.8×10^{-15}	2.6×10^{-15}	5.2×10^{-15}
BBEGG	7.5×10^{-16}	1.5×10^{-15}	3.5×10^{-15}	5.5×10^{-15}
AVW	8.6×10^{-16}	1.4×10^{-15}	2.6×10^{-15}	5.3×10^{-15}

After Newton correction

Error estimate $|p(\lambda)/p'(\lambda)|$

degree	17	72	295	1210
LAPACK	3.2×10^{-16}	3.4×10^{-16}	3.2×10^{-16}	3.1×10^{-16}
BBEGG	5.2×10^{-16}	3.1×10^{-16}	2.9×10^{-16}	3.8×10^{-16}
AVW	3.3×10^{-16}	3.1×10^{-16}	3.1×10^{-16}	3.1×10^{-16}

Second example: Chebyshev basis, complex, single-shift code

- 3-term recurrence
- comrade matrix, tridiagonal plus spike
- colleague matrix (Good 1961)

Second example: Chebyshev basis, complex, single-shift code

- 3-term recurrence
- comrade matrix, tridiagonal plus spike
- colleague matrix (Good 1961)

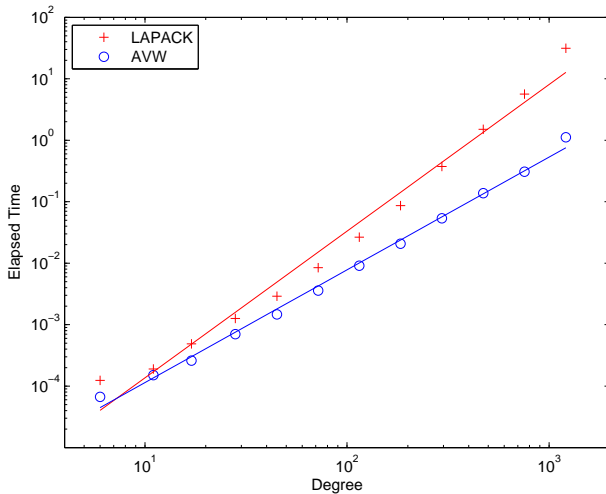
Second example: Chebyshev basis, complex, single-shift code

- 3-term recurrence
- comrade matrix, tridiagonal plus spike
- colleague matrix (Good 1961)

Second example: Chebyshev basis, complex, single-shift code

- 3-term recurrence
- comrade matrix, tridiagonal plus spike
- colleague matrix (Good 1961)

Speed Comparison, colleague case



Speed Comparison, colleague case

- No crossover point
- Slope = 1.76
- At $n = 1210$:

method	time
LAPACK	31.4
AVW	1.1

Speed Comparison, colleague case

- No crossover point
- Slope = 1.76
- At $n = 1210$:

method	time
LAPACK	31.4
AVW	1.1

Speed Comparison, colleague case

- No crossover point
- Slope = 1.76
- At $n = 1210$:

method	time
LAPACK	31.4
AVW	1.1

Accuracy Comparison, colleague case

Maximum of residual $\|(\lambda I - A)v\|/\|A\|\|v\|$

degree	17	72	295	1210
LAPACK	1.7×10^{-13}	1.1×10^{-12}	6.1×10^{-12}	2.8×10^{-11}
AVW	7.7×10^{-12}	2.8×10^{-10}	1.3×10^{-08}	3.3×10^{-07}

Accuracy Comparison, colleague case

Error estimate $|p(\lambda)/p'(\lambda)|$

degree	17	72	295	1210
LAPACK	1.9×10^{-14}	2.7×10^{-14}	3.1×10^{-14}	5.1×10^{-14}
AVW	3.8×10^{-13}	7.9×10^{-12}	3.9×10^{-11}	1.2×10^{-10}

After Newton correction

Maximum of residual $\|(\lambda I - A)v\|/\|A\|\|v\|$

degree	17	72	295	1210
LAPACK	7.7×10^{-15}	2.7×10^{-14}	1.7×10^{-13}	9.0×10^{-13}
AVW	7.4×10^{-15}	2.7×10^{-14}	1.4×10^{-13}	9.9×10^{-13}

After Newton correction

Error estimate $|p(\lambda)/p'(\lambda)|$

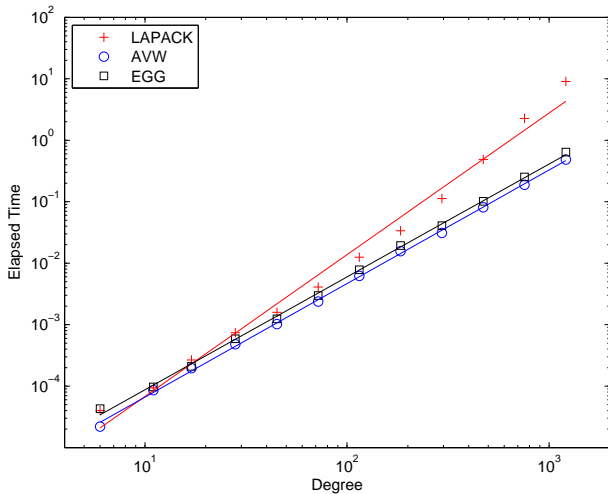
degree	17	72	295	1210
LAPACK	9.8×10^{-16}	6.6×10^{-16}	4.9×10^{-16}	2.1×10^{-16}
AVW	1.7×10^{-15}	3.2×10^{-16}	2.1×10^{-16}	3.4×10^{-16}

Third example: Chebyshev basis, real, double-shift code

Contestants

- LAPACK code DHSEQR ($O(n^3)$)
- EGG (symmetric plus rank-one, backward stable)
- AVW our code (double-shift)

Speed Comparison, real colleague case



Speed Comparison, real colleague case

- We are just a bit faster than EGG
- At $n = 1210$:

method	time
LAPACK	9.00
EGG	0.64
AVW	0.49

Speed Comparison, real colleague case

- We are just a bit faster than EGG
- At $n = 1210$:

method	time
LAPACK	9.00
EGG	0.64
AVW	0.49

Speed Comparison, real colleague case

- We are just a bit faster than EGG
- At $n = 1210$:

method	time
LAPACK	9.00
EGG	0.64
AVW	0.49

Accuracy Comparison, real colleague case

Maximum of residual $\|(\lambda I - A)v\|/\|A\|\|v\|$

degree	17	72	295	1210
LAPACK	1.7×10^{-13}	1.7×10^{-12}	6.3×10^{-12}	5.4×10^{-11}
EGG	1.9×10^{-13}	2.1×10^{-12}	7.4×10^{-12}	1.1×10^{-10}
AVW	4.3×10^{-09}	1.2×10^{-06}	4.6×10^{-07}	2.4×10^{-03}

Accuracy Comparison, real colleague case

Maximum of residual $\|(\lambda I - A)v\|/\|A\|\|v\|$

degree	17	72	295	1210
LAPACK	1.7×10^{-13}	1.7×10^{-12}	6.3×10^{-12}	5.4×10^{-11}
EGG	1.9×10^{-13}	2.1×10^{-12}	7.4×10^{-12}	1.1×10^{-10}
AVW	4.3×10^{-09}	1.2×10^{-06}	4.6×10^{-07}	2.4×10^{-03}

Ouch!

After Newton correction

Maximum of residual $\|(\lambda I - A)v\|/\|A\|\|v\|$

degree	17	72	295	1210
LAPACK	8.2×10^{-15}	3.5×10^{-14}	1.0×10^{-13}	1.6×10^{-12}
EGG	6.5×10^{-15}	3.3×10^{-14}	1.3×10^{-13}	2.1×10^{-12}
AVW	5.8×10^{-15}	5.8×10^{-13}	5.0×10^{-12}	4.3×10^{-05}

- A second Newton correction does the trick.

After Newton correction

Maximum of residual $\|(\lambda I - A)v\|/\|A\|\|v\|$

degree	17	72	295	1210
LAPACK	8.2×10^{-15}	3.5×10^{-14}	1.0×10^{-13}	1.6×10^{-12}
EGG	6.5×10^{-15}	3.3×10^{-14}	1.3×10^{-13}	2.1×10^{-12}
AVW	5.8×10^{-15}	5.8×10^{-13}	5.0×10^{-12}	4.3×10^{-05}

- A second Newton correction does the trick.

Final Remarks

- Our code is lightning fast, but
- it is unstable.
- Newton correction is helpful.
- Double-shift code disappointing.
- These are benign examples.
- Hybrid methods?

Final Remarks

- Our code is lightning fast, but
- it is unstable.
- Newton correction is helpful.
- Double-shift code disappointing.
- These are benign examples.
- Hybrid methods?

Final Remarks

- Our code is lightning fast, but
- it is unstable.
- Newton correction is helpful.
- Double-shift code disappointing.
- These are benign examples.
- Hybrid methods?

Final Remarks

- Our code is lightning fast, but
- it is unstable.
- Newton correction is helpful.
- Double-shift code disappointing.
- These are benign examples.
- Hybrid methods?

Final Remarks

- Our code is lightning fast, but
- it is unstable.
- Newton correction is helpful.
- Double-shift code disappointing.
- These are benign examples.
- Hybrid methods?

Final Remarks

- Our code is lightning fast, but
- it is unstable.
- Newton correction is helpful.
- Double-shift code disappointing.
- These are benign examples.
- Hybrid methods?

Final Remarks

- Our code is lightning fast, but
- it is unstable.
- Newton correction is helpful.
- Double-shift code disappointing.
- These are benign examples.
- Hybrid methods?

- Our code is lightning fast, but
- it is unstable.
- Newton correction is helpful.
- Double-shift code disappointing.
- These are benign examples.
- Hybrid methods?

Thank you for your attention.