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

David S. Watkins

Department of Mathematics
Washington State University

Providence, June 2013

Collaborators

This is joint work with

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

Zeros of a polynomial (monomial basis)

- $p(z) = z^n - c_1 z^{n-1} - c_2 z^{n-2} - \cdots - c_n = 0$
- companion matrix

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

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

Zeros of a polynomial (monomial basis)

- $p(z) = z^n - c_1 z^{n-1} - c_2 z^{n-2} - \cdots - c_n = 0$
- companion matrix

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

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

Zeros of a polynomial (monomial basis)

- $p(z) = z^n - c_1 z^{n-1} - c_2 z^{n-2} - \cdots - c_n = 0$
- 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)

- $p(z) = z^n - c_1 z^{n-1} - c_2 z^{n-2} - \cdots - c_n = 0$
- 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)

- $p(z) = z^n - c_1 z^{n-1} - c_2 z^{n-2} - \cdots - c_n = 0$
- 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)

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

- $p_k(z) b_{k+1,k} = z p_{k-1}(z) - \sum_{j=1}^{k} p_{j-1}(z) b_{jk}$
- 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)

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

- $p_k(z) b_{k+1,k} = z p_{k-1}(z) - \sum_{j=1}^{k} p_{j-1}(z) b_{jk}$
- 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)

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

- $p_k(z) b_{k+1,k} = z p_{k-1}(z) - \sum_{j=1}^{k} p_{j-1}(z) b_{jk}$
- 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)

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

- $p_k(z) b_{k+1,k} = z p_{k-1}(z) - \sum_{j=1}^{k} p_{j-1}(z) b_{jk}$
- 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)

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

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

Zeros of a polynomial (3-term recurrence)

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

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

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

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

- Newton basis
- $p_k(z)\alpha_k = (z - \beta_k)p_{k-1}(z)$
- $p(z) = p_n(z) - c_1 p_{n-1}(z) - c_2 p_{n-2}(z) - \cdots - c_n p_0(z) = 0$

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

- bidiagonal plus spike

Zeros of a polynomial (2-term recurrence)

- Newton basis
- $p_k(z)\alpha_k = (z - \beta_k)p_{k-1}(z)$
- $p(z) = p_n(z) - c_1 p_{n-1}(z) - c_2 p_{n-2}(z) - \cdots - c_n p_0(z) = 0$

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

- bidiagonal plus spike

Zeros of a polynomial (2-term recurrence)

- Newton basis
- $p_k(z)\alpha_k = (z - \beta_k)p_{k-1}(z)$
- $p(z) = p_n(z) - c_1 p_{n-1}(z) - c_2 p_{n-2}(z) - \cdots - c_n p_0(z) = 0$

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

- bidiagonal plus spike

Zeros of a polynomial (1-term recurrence)

- monomial basis
- $p_k(z) = zp_{k-1}(z)$
- $p(z) = p_n(z) - c_1 p_{n-1}(z) - c_2 p_{n-2}(z) - \cdots - c_n p_0(z) = 0$

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

- one diagonal plus spike
- companion matrix

Zeros of a polynomial (1-term recurrence)

- monomial basis
- $p_k(z) = zp_{k-1}(z)$
- $p(z) = p_n(z) - c_1 p_{n-1}(z) - c_2 p_{n-2}(z) - \cdots - c_n p_0(z) = 0$

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

- 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.

Fast Methods

We have developed great methods that are . . .

- lightning fast
- unstable

Fast Methods

We have developed great methods that are . . .

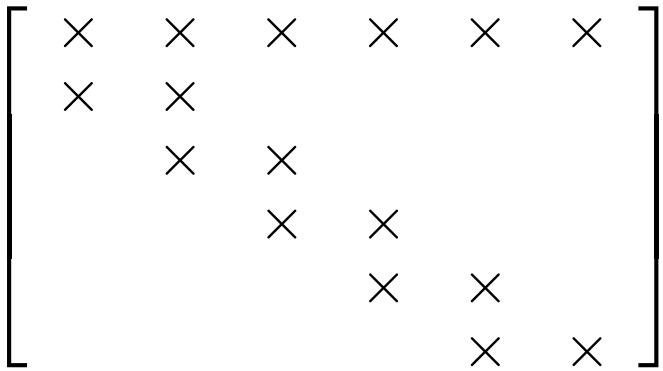
- lightning fast
- unstable

Fast Methods

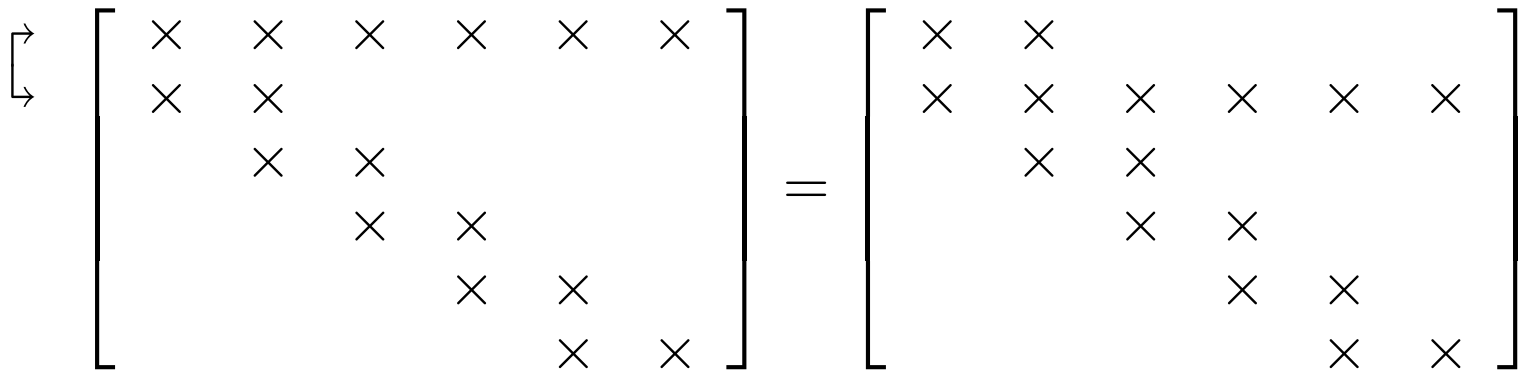
We have developed great methods that are . . .

- lightning fast
- unstable

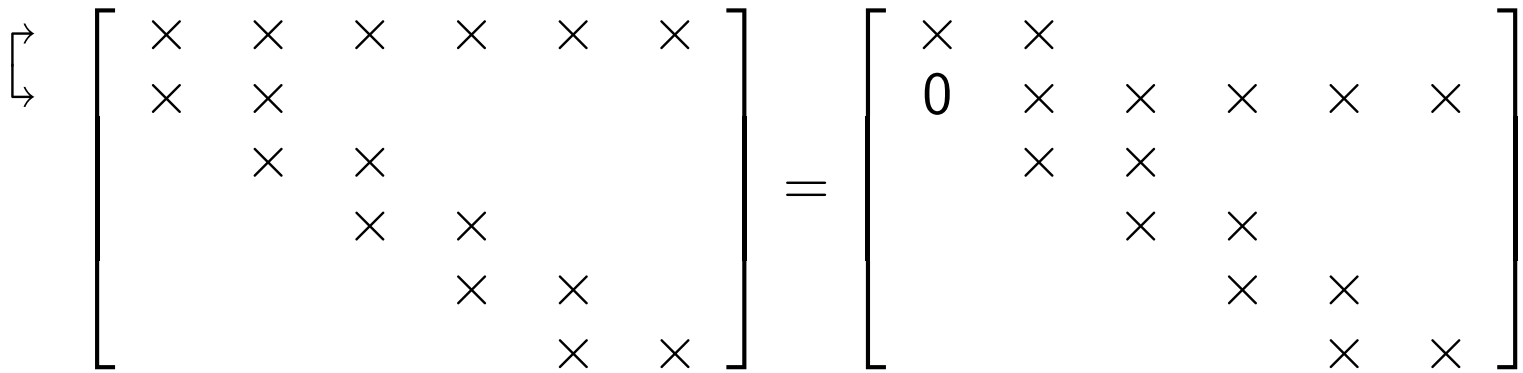
A useful factorization (by Gaussian elimination)



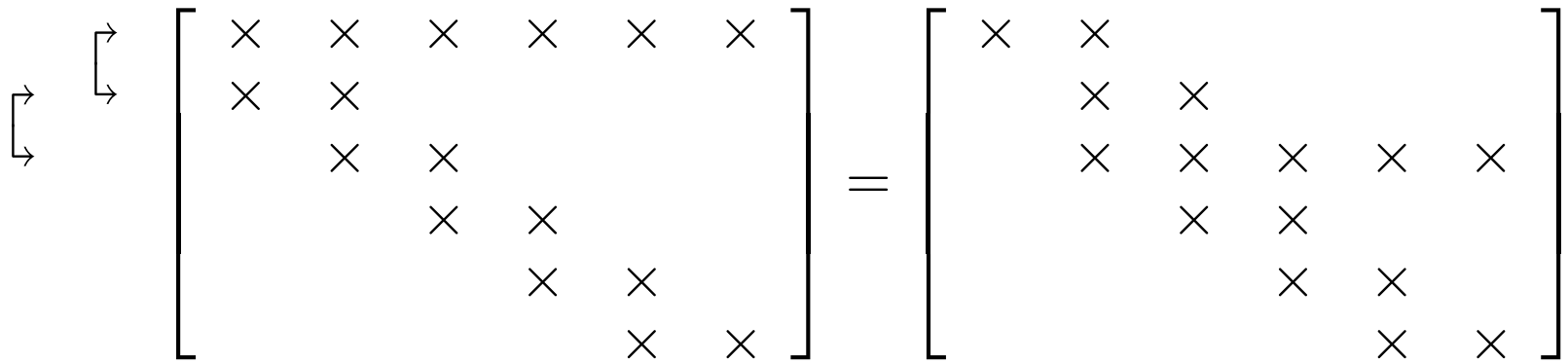
A useful factorization (by Gaussian elimination)



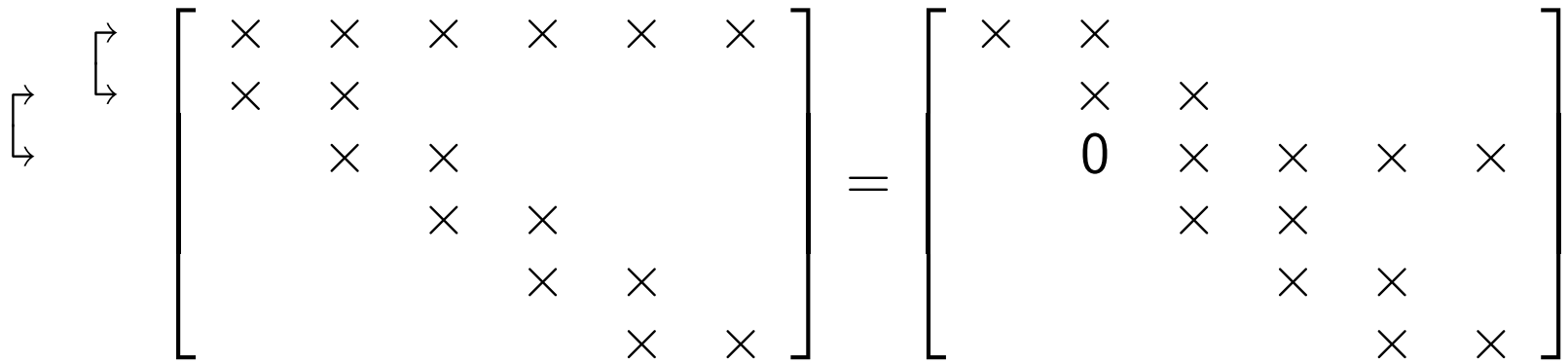
A useful factorization (by Gaussian elimination)



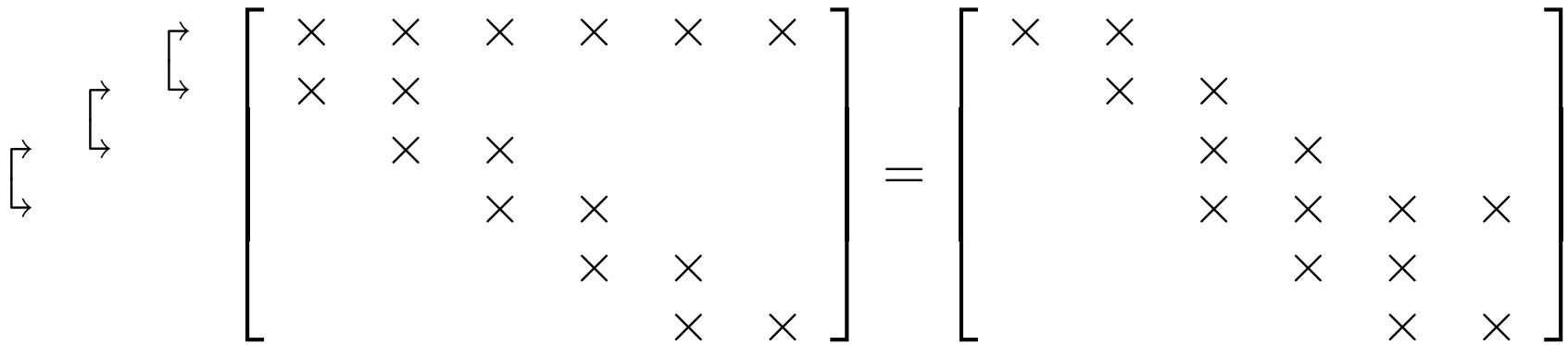
A useful factorization (by Gaussian elimination)



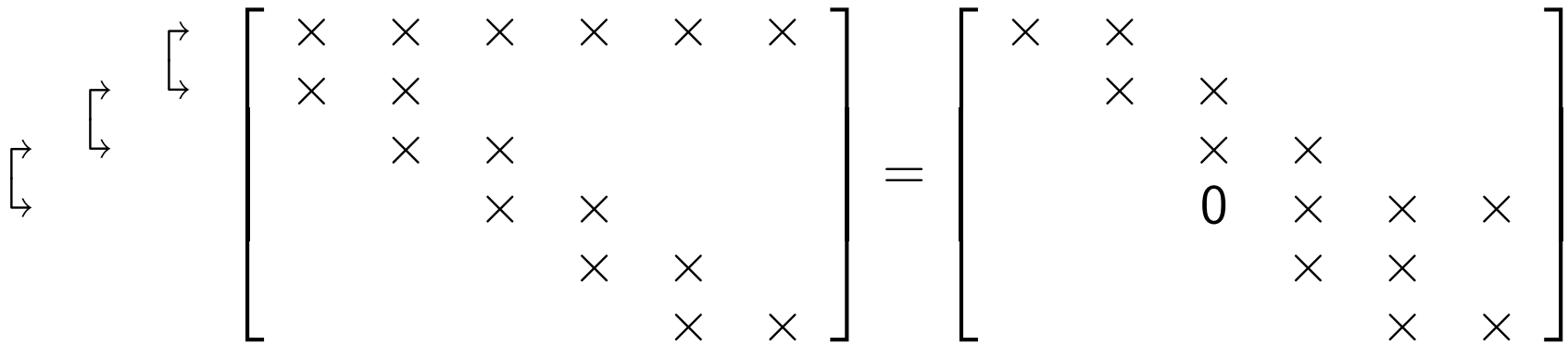
A useful factorization (by Gaussian elimination)



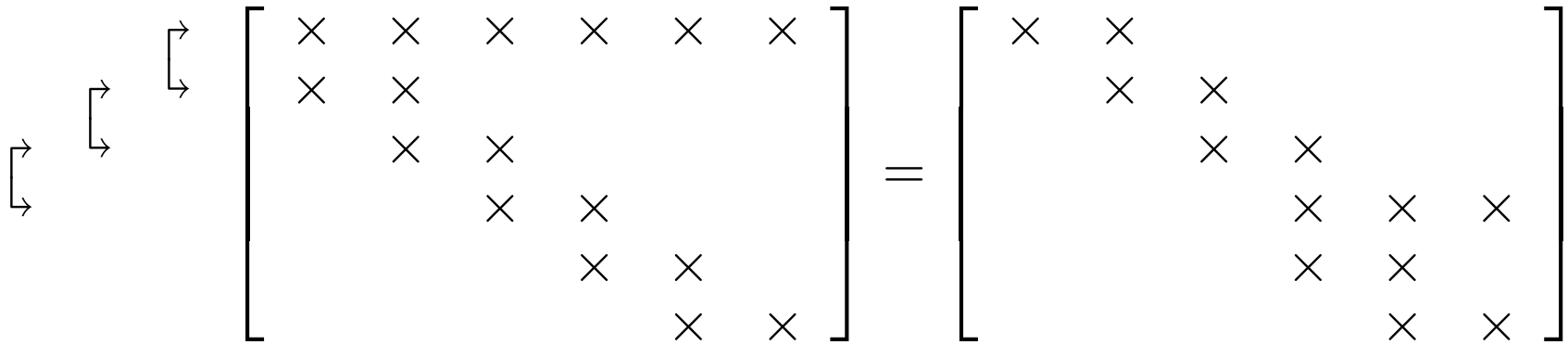
A useful factorization (by Gaussian elimination)



A useful factorization (by Gaussian elimination)



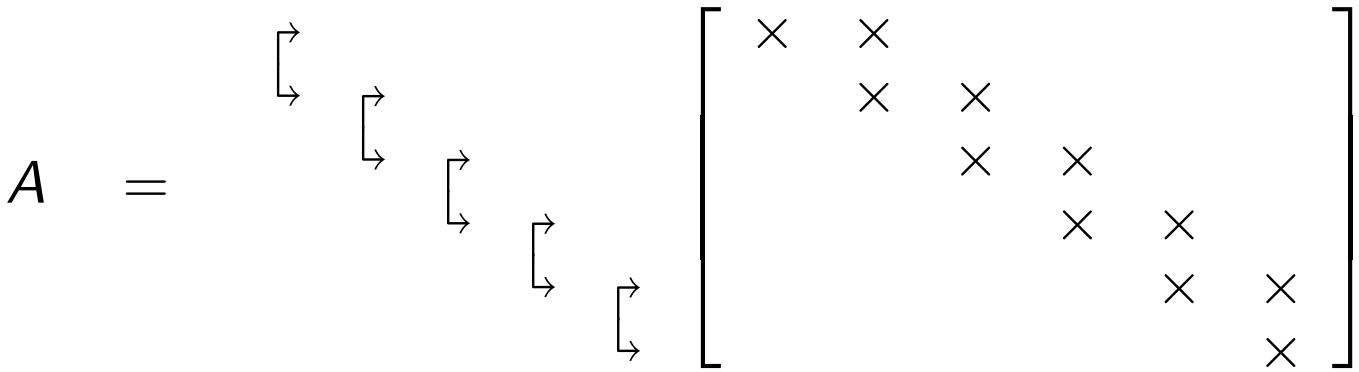
A useful factorization (by Gaussian elimination)



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

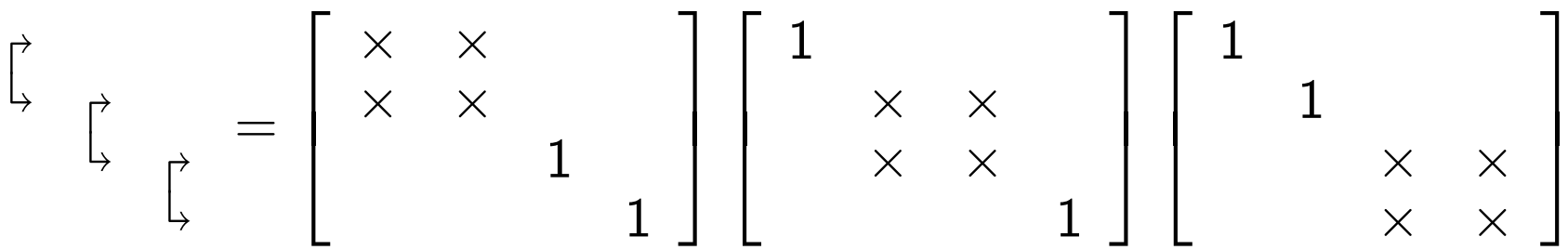
A useful factorization (by Gaussian elimination)

- We get the factorization

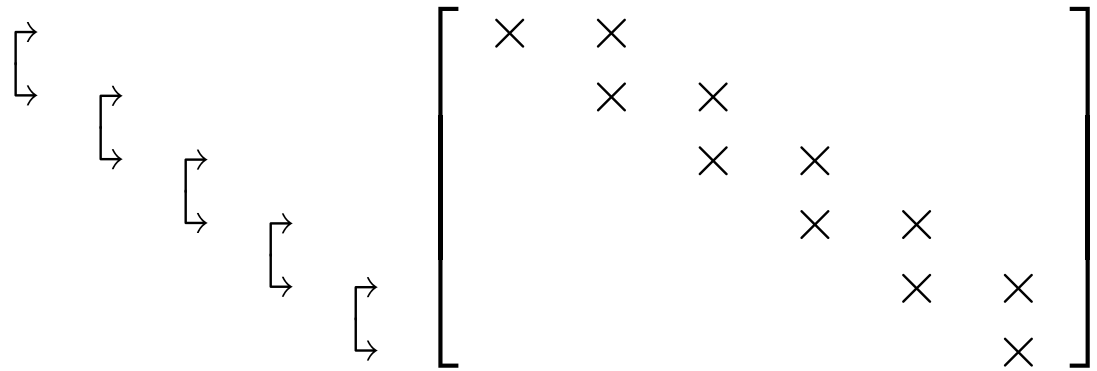


- core transformations times banded upper triangular
- band width = length of recurrence

What core transformations look like

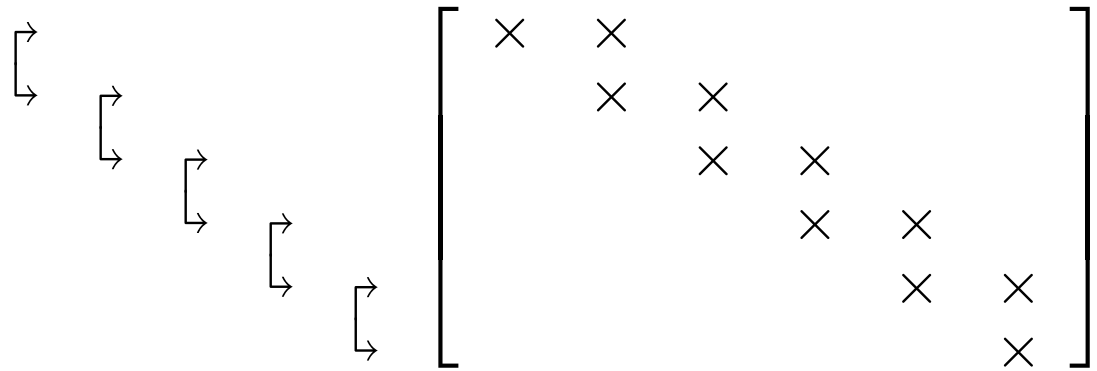


Bulge Chasing Algorithm



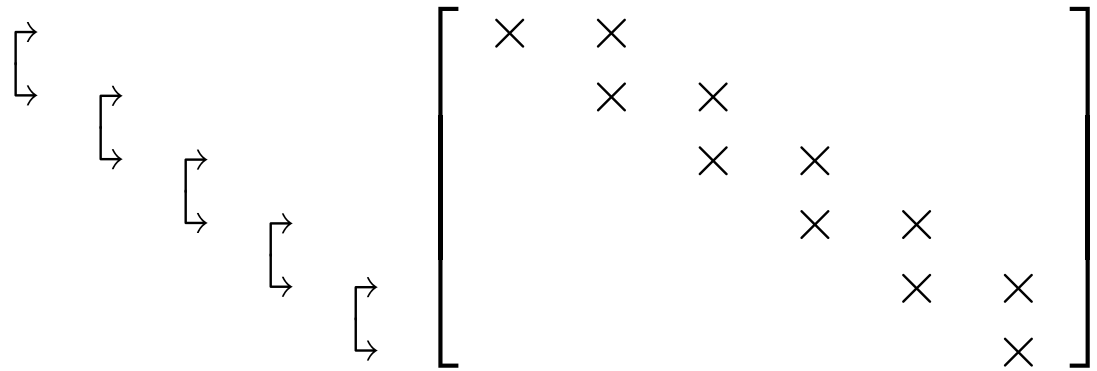
- We will develop a nonunitary bulge-chasing algorithm
- that preserves this factorization.
- Note: Storage is $O(n)$ (about $6n$ for this case).
- We have single-shift and double-shift variants.

Bulge Chasing Algorithm



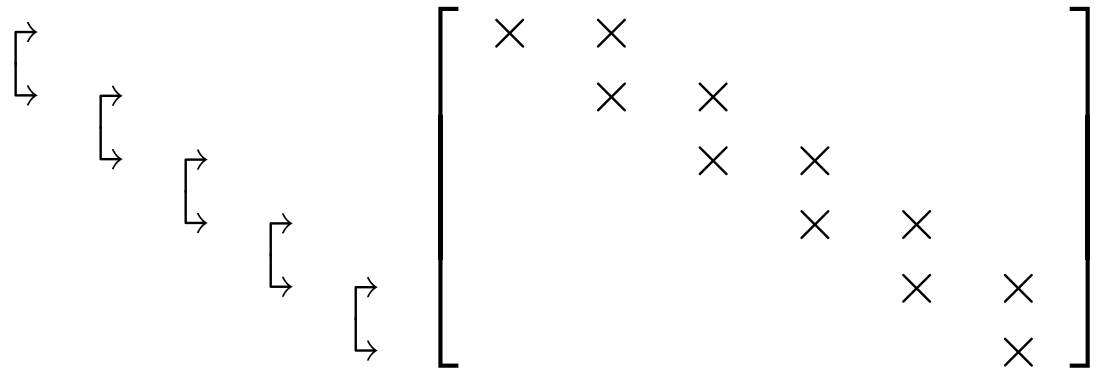
- We will develop a nonunitary bulge-chasing algorithm
- that preserves this factorization.
- Note: Storage is $O(n)$ (about $6n$ for this case).
- We have single-shift and double-shift variants.

Bulge Chasing Algorithm



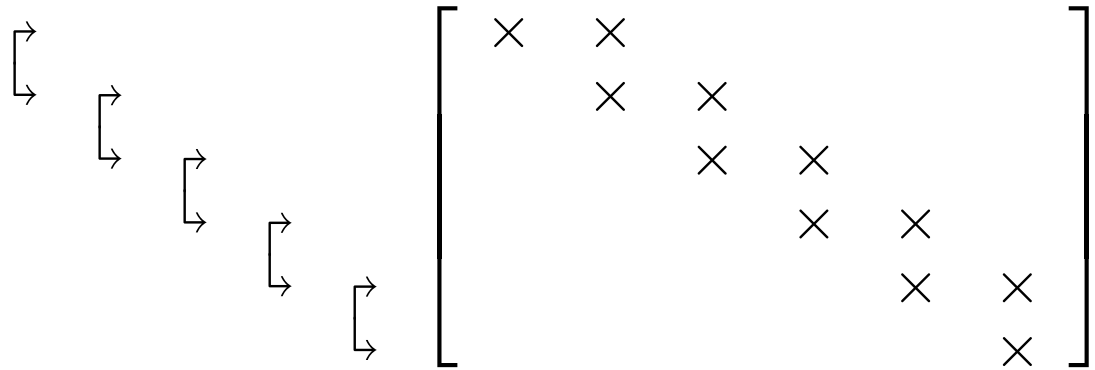
- We will develop a nonunitary bulge-chasing algorithm
- that preserves this factorization.
- Note: Storage is $O(n)$ (about $6n$ for this case).
- We have single-shift and double-shift variants.

Bulge Chasing Algorithm

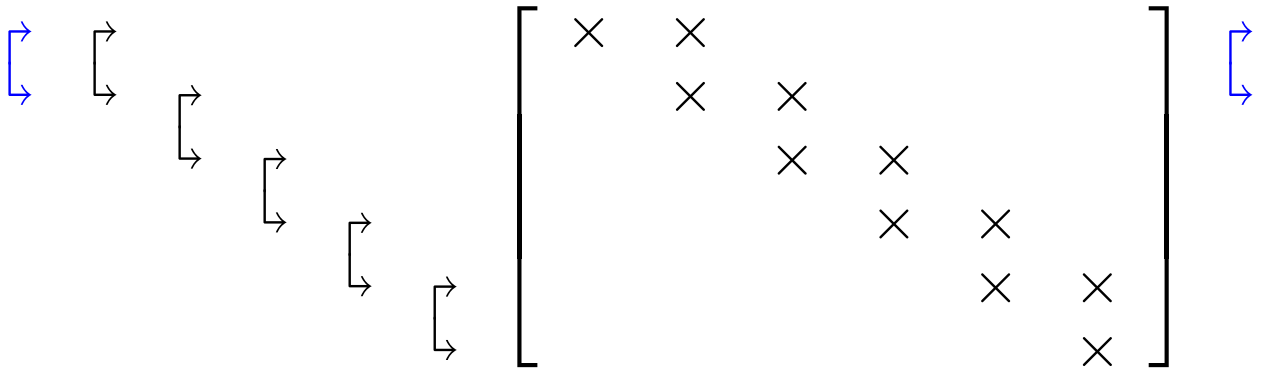


- We will develop a nonunitary bulge-chasing algorithm
- that preserves this factorization.
- Note: Storage is $O(n)$ (about $6n$ for this case).
- We have single-shift and double-shift variants.

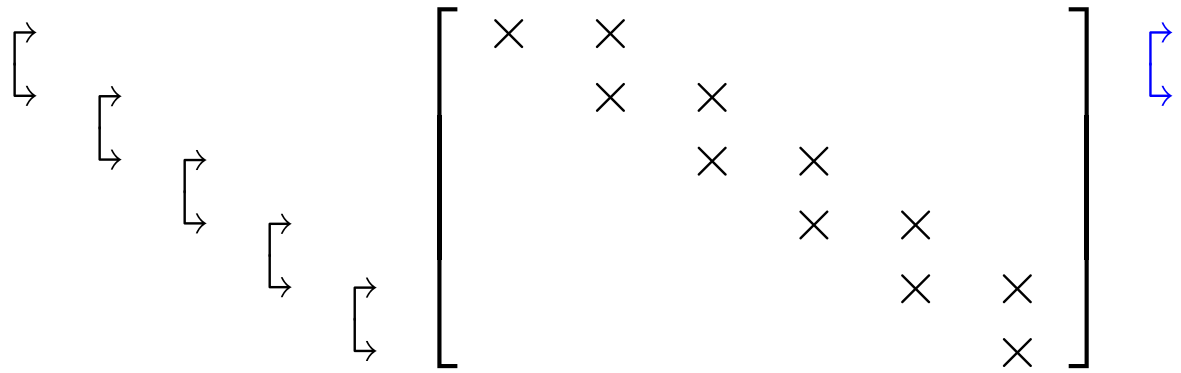
Bulge Chasing Algorithm (single-shift variant)



Bulge Chasing Algorithm (single-shift variant)

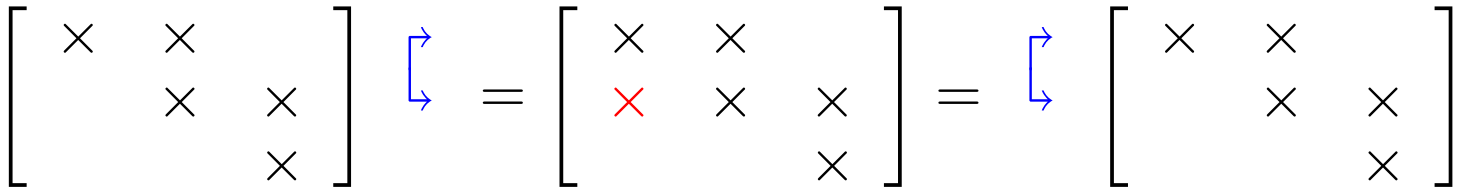


Bulge Chasing Algorithm (single-shift variant)



Passing Gauss transform through triangular matrix

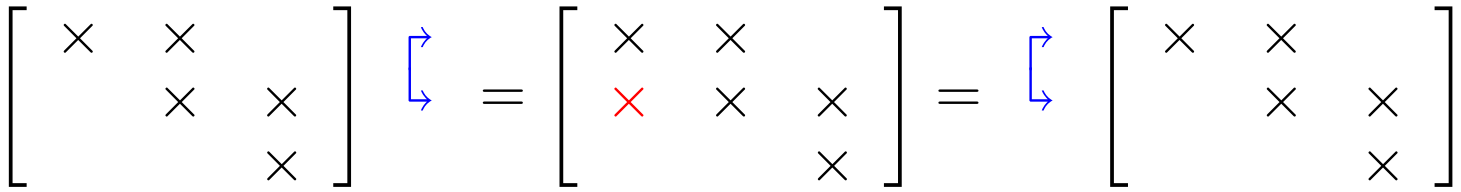
- Blue core transformations are Gauss transforms.



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

Passing Gauss transform through triangular matrix

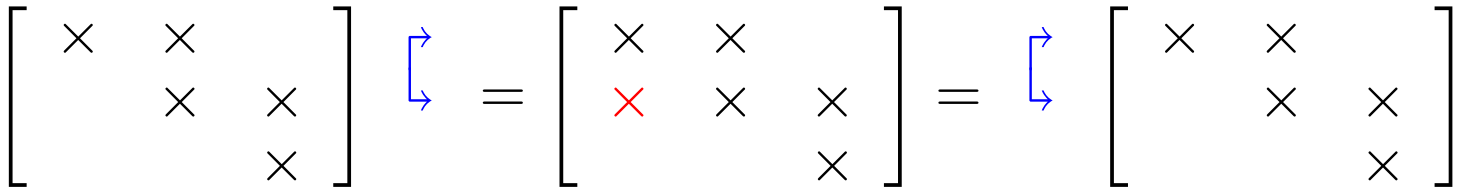
- Blue core transformations are Gauss transforms.



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

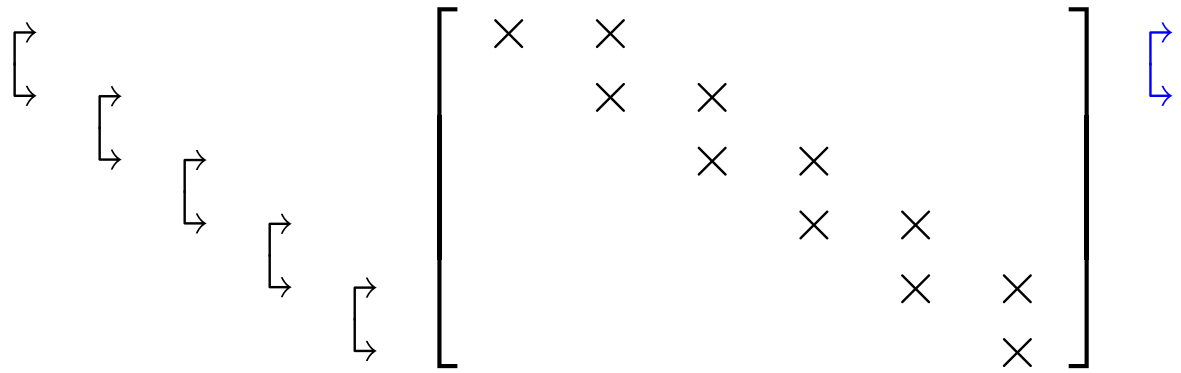
Passing Gauss transform through triangular matrix

- Blue core transformations are Gauss transforms.

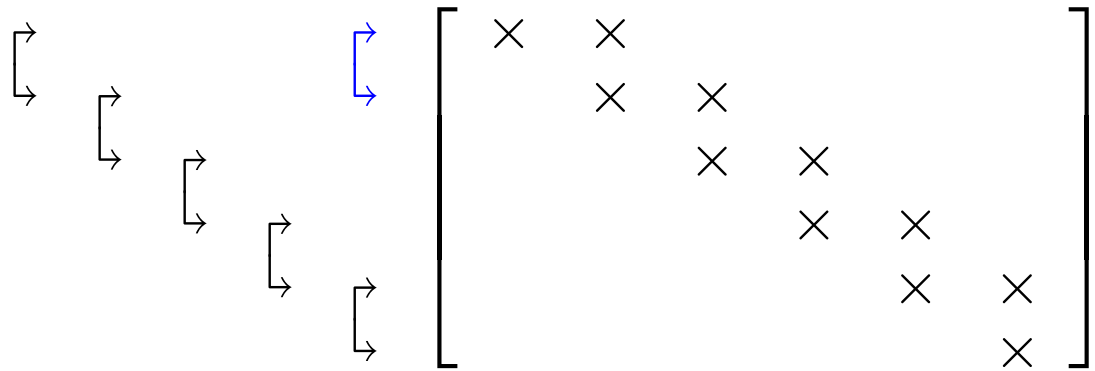


- We do this at every opportunity.
- From this point on, we suppress the triangular matrix.

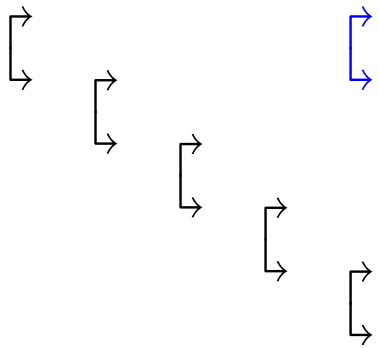
Bulge Chasing Algorithm (single-shift variant)



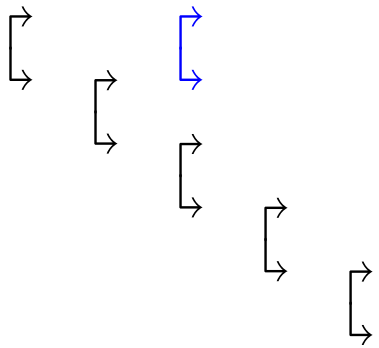
Bulge Chasing Algorithm (single-shift variant)



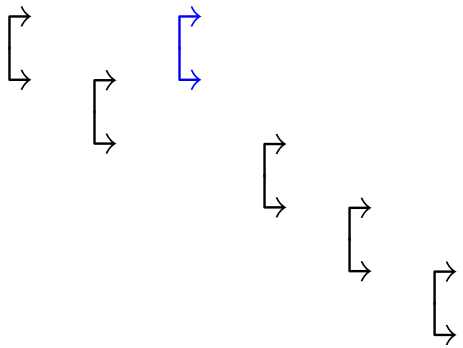
Bulge Chasing Algorithm (single-shift variant)



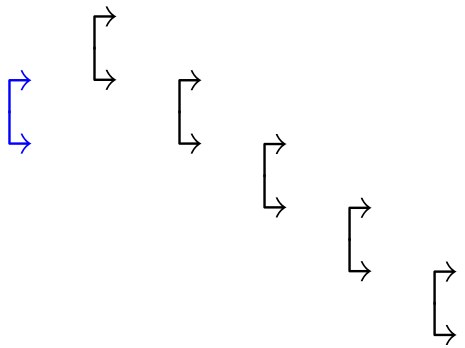
Bulge Chasing Algorithm (single-shift variant)



Bulge Chasing Algorithm (single-shift variant)

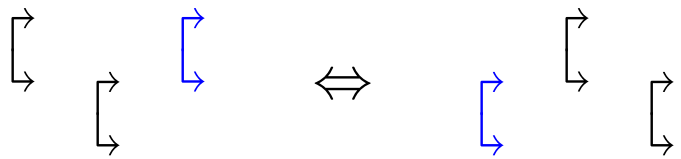


Bulge Chasing Algorithm (single-shift variant)



Turnover operation

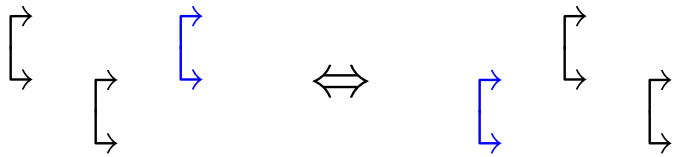
- Can we do this?



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

Turnover operation

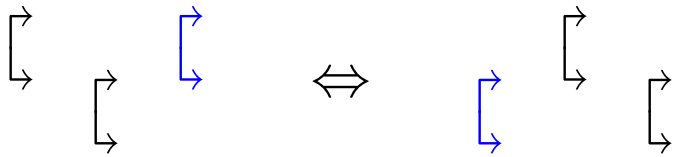
- Can we do this?



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

Turnover operation

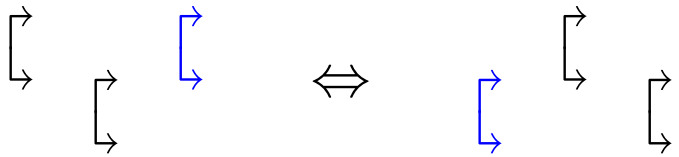
- Can we do this?



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

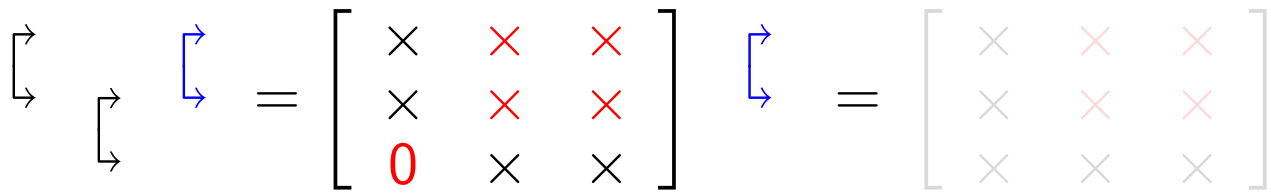
Turnover operation

- Can we do this?



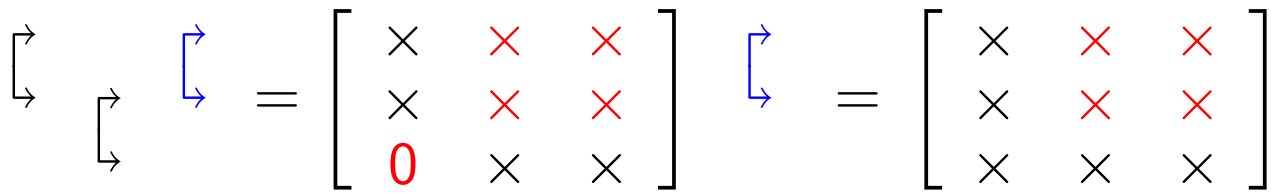
- crucial question
- Pessimistic answer: No! (not always)
- Optimistic answer: Yes! (almost always)

Turnover operation



- Red submatrix still has rank one.

Turnover operation



- Red submatrix still has rank one.

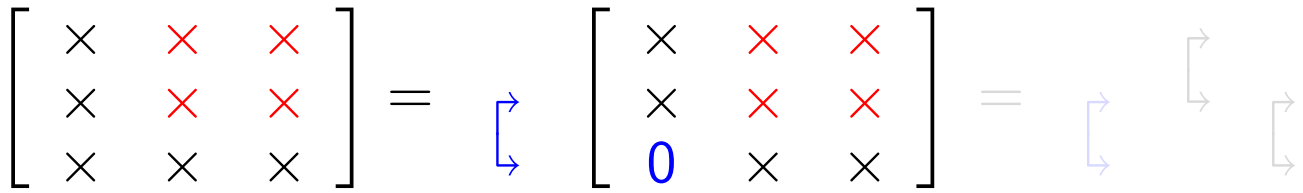
Turnover operation

- So . . .

- Gauss transform does not disturb 2×2 submatrix.

Turnover operation

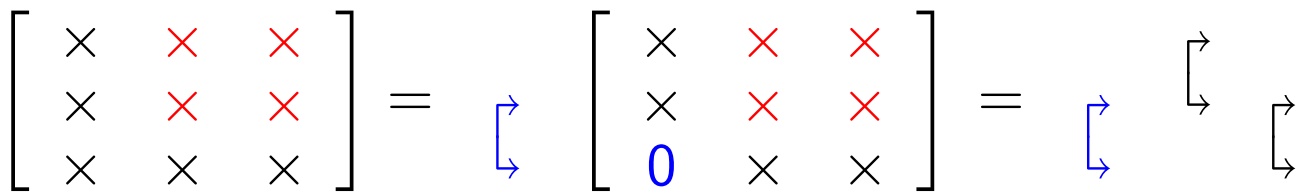
- So . . .



- Gauss transform does not disturb 2×2 submatrix.

Turnover operation

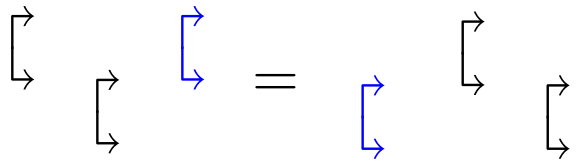
- So ...



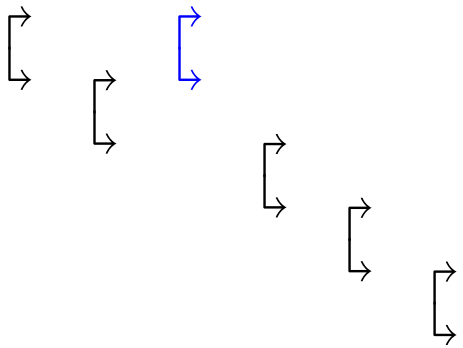
- Gauss transform does not disturb 2×2 submatrix.

Turnover operation

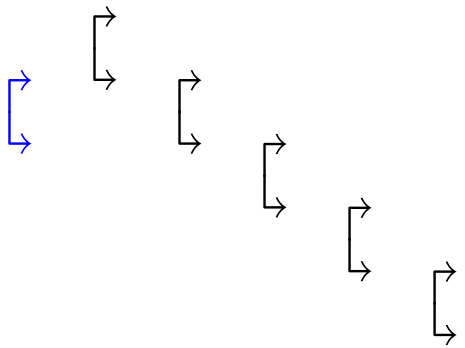
- Complete turnover looks like this:



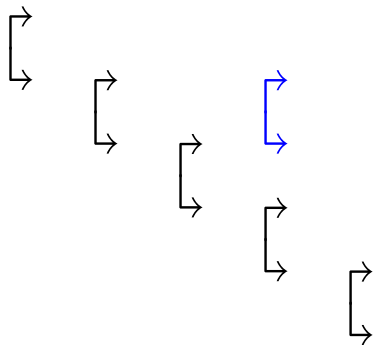
Bulge Chasing Algorithm (single-shift variant)



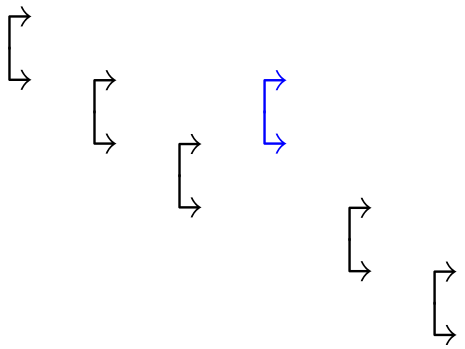
Bulge Chasing Algorithm (single-shift variant)



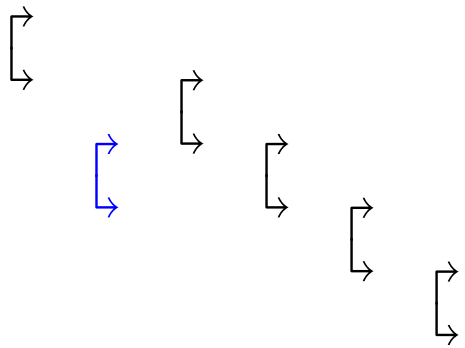
Bulge Chasing Algorithm (single-shift variant)



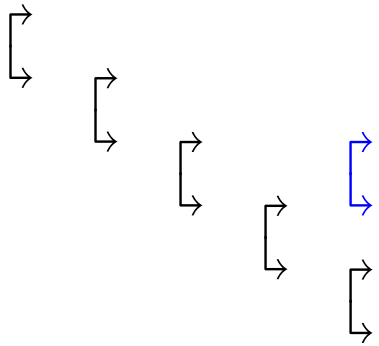
Bulge Chasing Algorithm (single-shift variant)



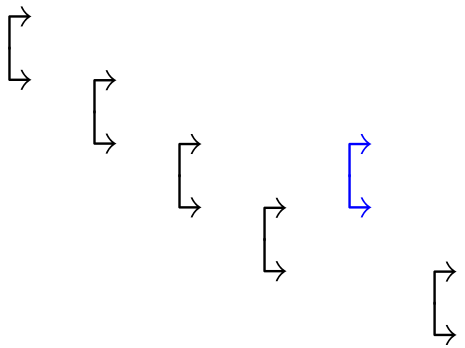
Bulge Chasing Algorithm (single-shift variant)



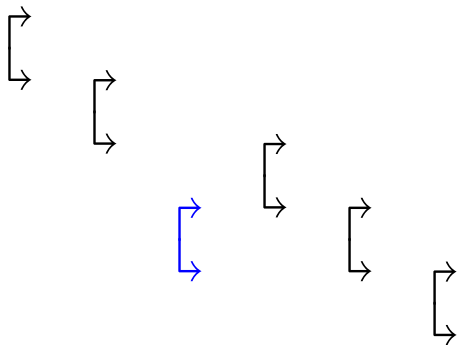
Bulge Chasing Algorithm (single-shift variant)



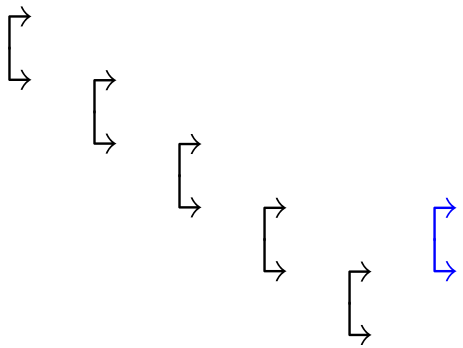
Bulge Chasing Algorithm (single-shift variant)



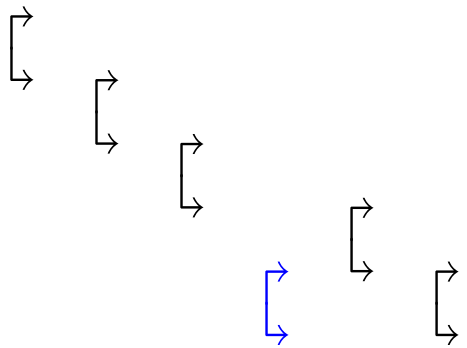
Bulge Chasing Algorithm (single-shift variant)



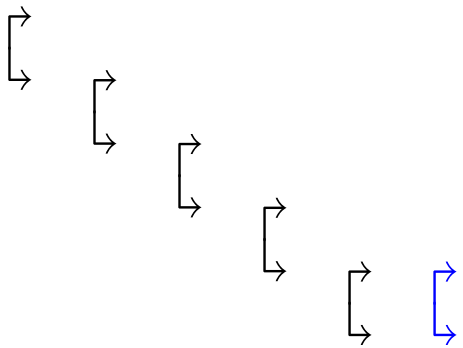
Bulge Chasing Algorithm (single-shift variant)



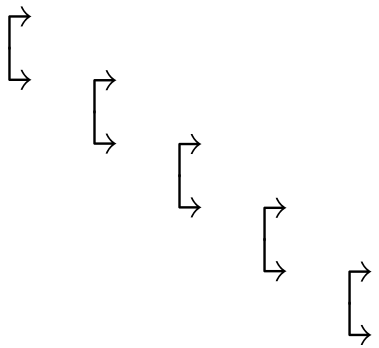
Bulge Chasing Algorithm (single-shift variant)



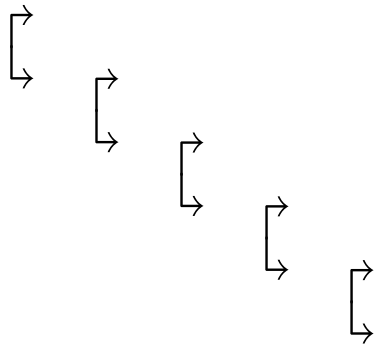
Bulge Chasing Algorithm (single-shift variant)



Bulge Chasing Algorithm (single-shift variant)

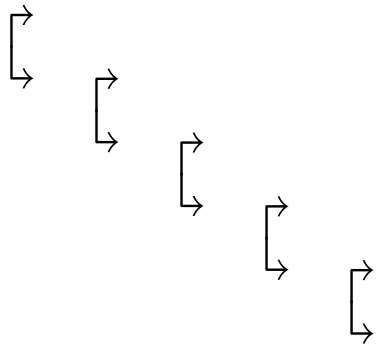


Bulge Chasing Algorithm (single-shift variant)



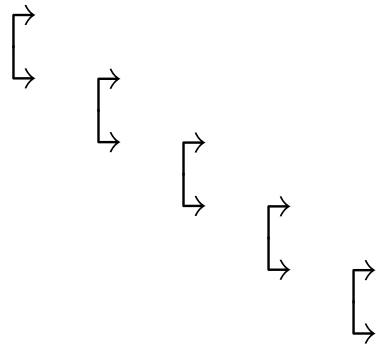
- Done!
- Now repeat again, again, and again

Bulge Chasing Algorithm (single-shift variant)



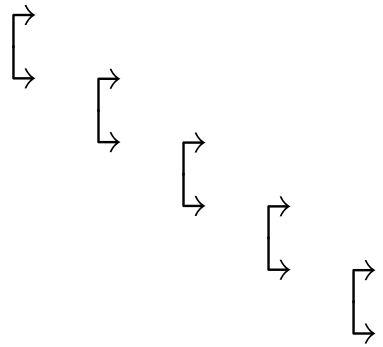
- Done!
- Now repeat again, again, and again

Bulge Chasing Algorithm (single-shift variant)



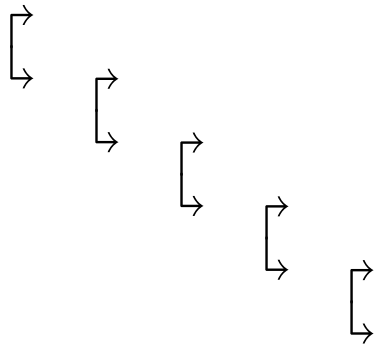
- Done!
- Now repeat again, again, and again

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.

Stability issue

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

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.

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.

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.

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.

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.

Stability issue

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

Stability issue

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

Stability issue

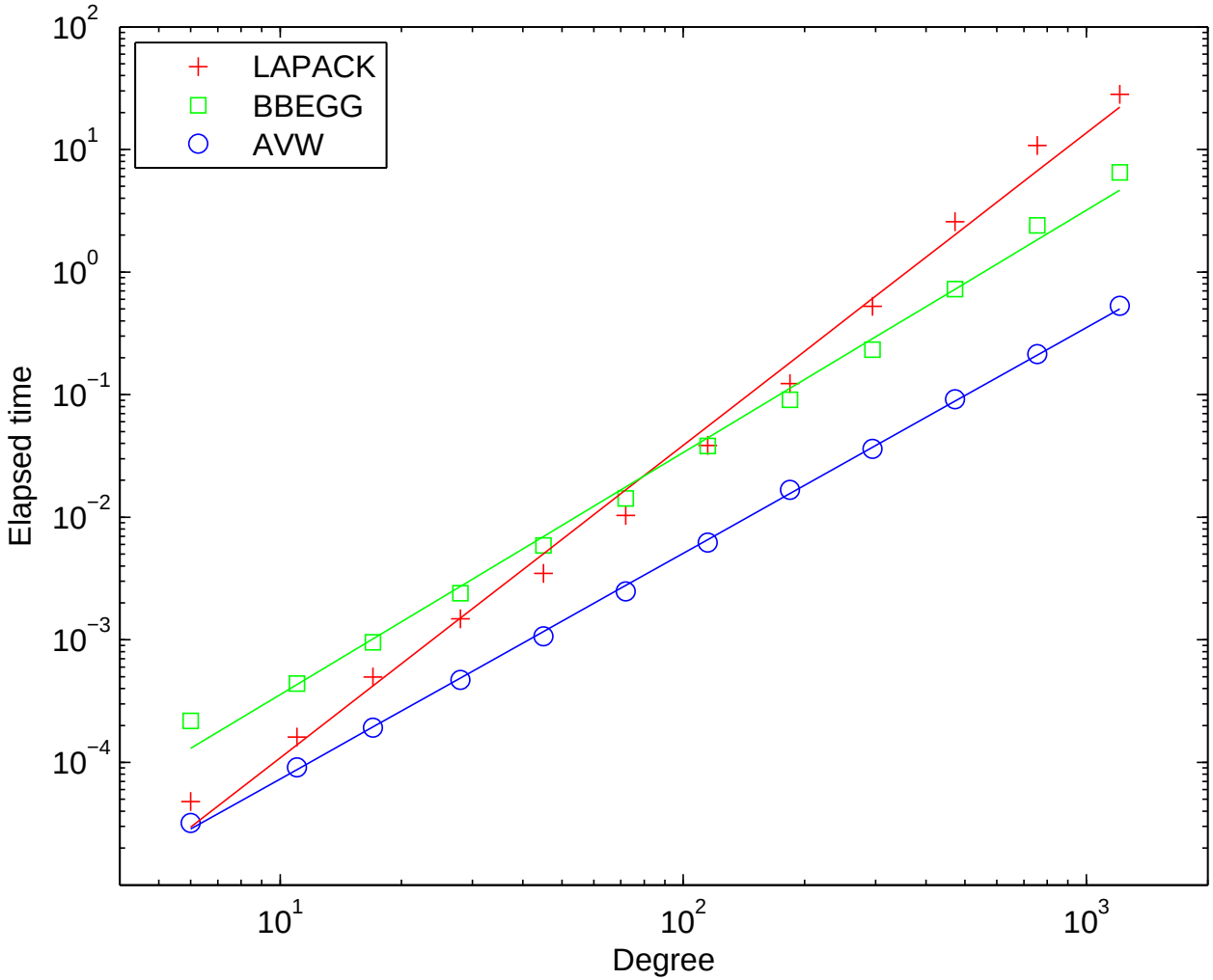
- This is a structured, implicitly-shifted LR algorithm.
- Unstable! Can we trust the results? No,
- but we can do *a posteriori* tests.
- $\| (\lambda I - A)v \| = |p(\lambda)|$
- $| \frac{p(\lambda)}{p'(\lambda)} |$ is estimate of error.
- Newton correction
- Cost: $O(n^2)$ flops for n roots.

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 $\quad \|(\lambda I - A)v\| / \|A\| \|v\|$

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}

Accuracy Comparison, companion case

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}

Accuracy Comparison, companion case

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}

Accuracy Comparison, companion case

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)

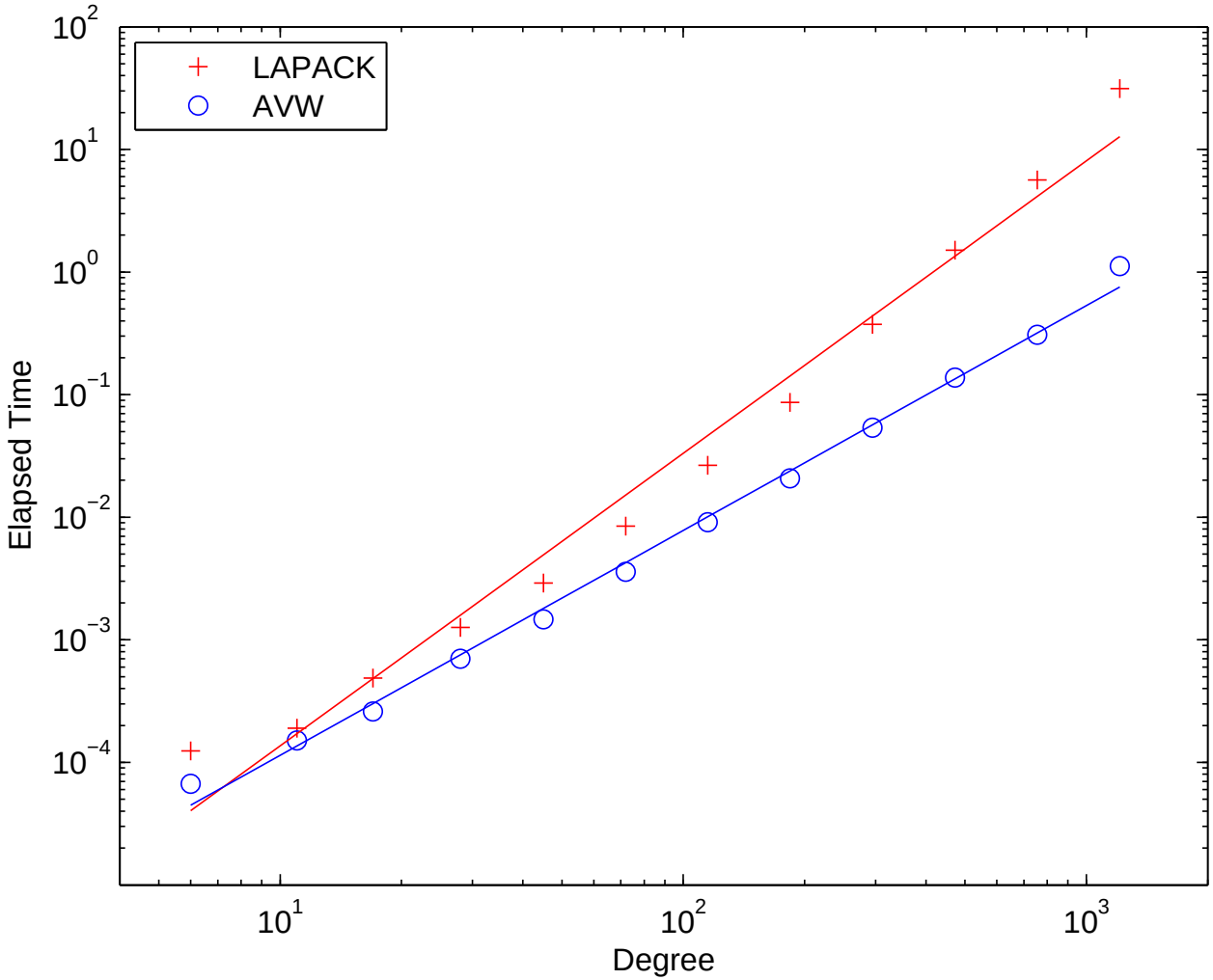
Second example: Chebyshev basis, complex, single-shift code

- 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 $\quad \|(\lambda I - A)v\|/\|A\|\|v\|$

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}

Accuracy Comparison, colleague case

After Newton correction

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

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}

Accuracy Comparison, colleague case

After Newton correction

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

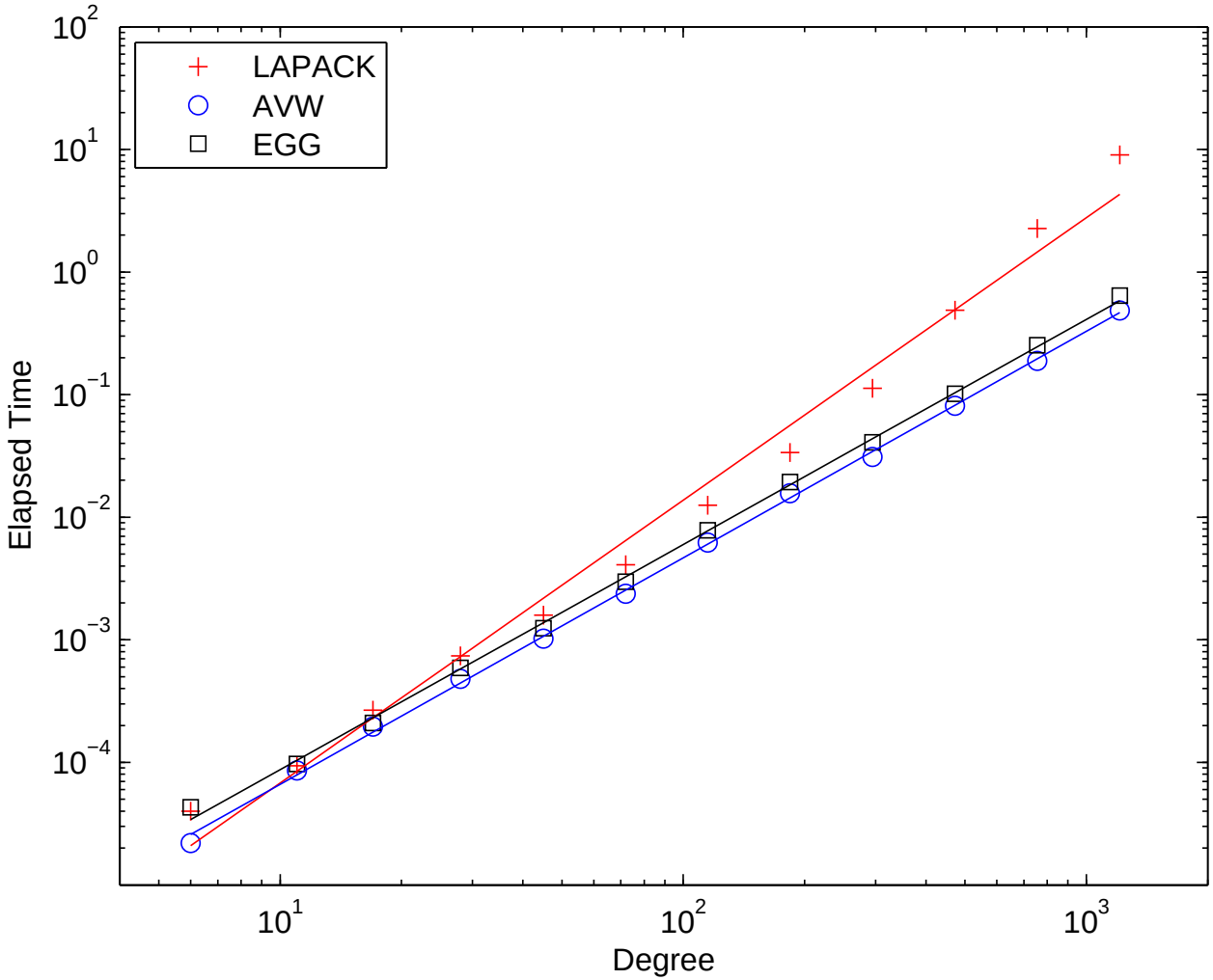
degree	17	72	295	1210
LAPACK	9.8×10^{-16}	6.6×10^{-16}	4.9×10^{-16}	2.1×10^{-16}
AVW	1.7×10^{-15}	3.2×10^{-16}	2.1×10^{-16}	3.4×10^{-16}

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 $\quad \|(\lambda I - A)v\|/\|A\|\|v\|$

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 $\quad \|(\lambda I - A)v\|/\|A\|\|v\|$

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!

Accuracy Comparison, real colleague case

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.

Accuracy Comparison, real colleague case

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?

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?

Thank you for your attention.