Core-Chasing Algorithms for the Eigenvalue Problem

David S. Watkins

Department of Mathematics
Washington State University

July, 2016

Our International Research Group

This is joint work with

- **Jared Aurentz** (Oxford $\rightarrow$ Madrid)
- **Thomas Mach** (KU Leuven $\rightarrow$ Astana)
- **Raf Vandebril** (KU Leuven)

Today's Topic

Today's Topic

- The matrix eigenvalue problem

Today's Topic

- The matrix eigenvalue problem
- $A \in \mathbb{C}^{n \times n}$

Today's Topic

- The matrix eigenvalue problem
- $A \in \mathbb{C}^{n \times n}$
- Find the eigenvalues (. . . vectors, invariant subspaces)

Today's Topic

- The matrix eigenvalue problem
- $A \in \mathbb{C}^{n \times n}$
- Find the eigenvalues (. . . vectors, invariant subspaces)
- Possible structures
  - unitary
  - unitary-plus-rank-one (companion)
  - symmetric
  - symmetric-plus-rank-one (colleague)

Today's Topic

- The matrix eigenvalue problem
- $A \in \mathbb{C}^{n \times n}$
- Find the eigenvalues (. . . vectors, invariant subspaces)
- Possible structures
  - unitary
  - unitary-plus-rank-one (companion)
  - symmetric
  - symmetric-plus-rank-one (colleague)
  - . . . or no special structure

John Francis

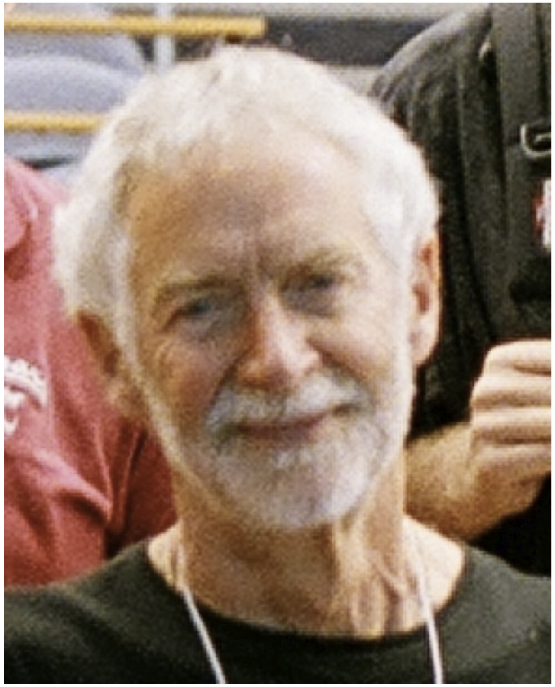


photo: Frank Uhlig, 2009

John Francis

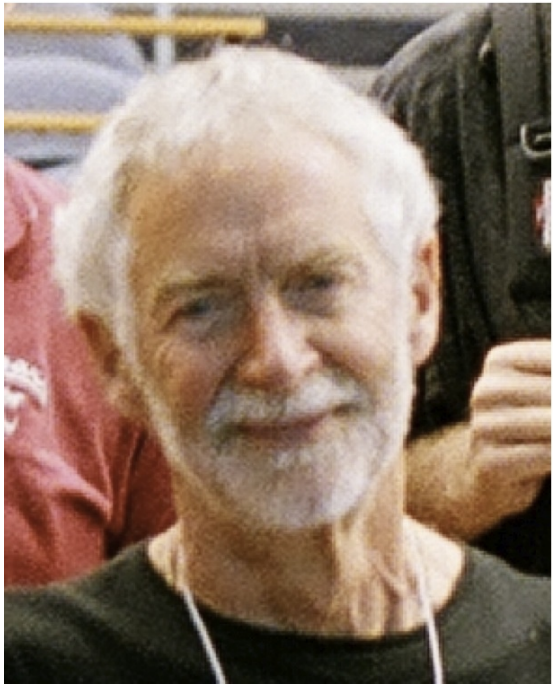


photo: Frank Uhlig, 2009

- invented the winning algorithm in 1959.

John Francis

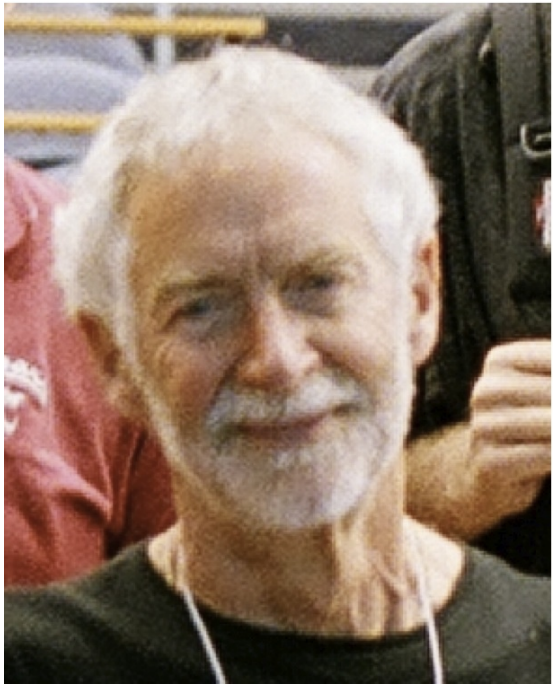


photo: Frank Uhlig, 2009

- invented the winning algorithm in 1959.
- implicitly shifted QR algorithm

John Francis

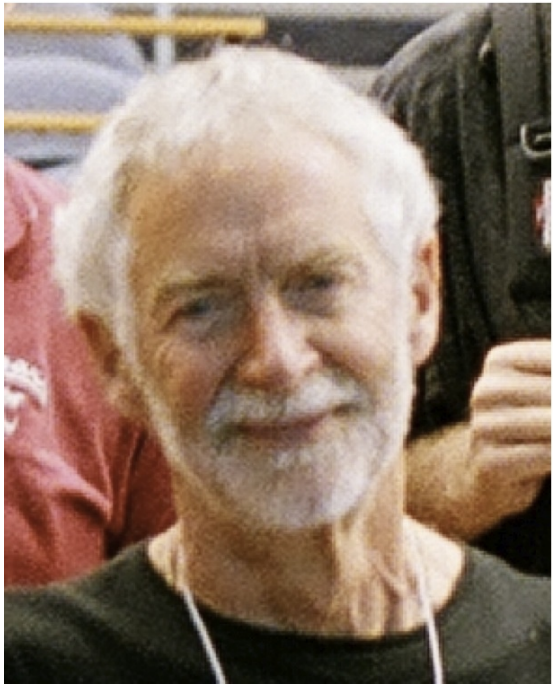


photo: Frank Uhlig, 2009

- invented the winning algorithm in 1959.
- implicitly shifted QR algorithm
- Our algorithms are all variants of this.

- Francis's algorithm . . .

- Francis's algorithm . . .
- . . . is a bulge chasing algorithm.

- Francis's algorithm . . .
- . . . is a bulge chasing algorithm.
- We turn it into a core chasing algorithm.

- Francis's algorithm . . .
- . . . is a bulge chasing algorithm.
- We turn it into a core chasing algorithm.
- Instead of chasing bulges, we chase core transformations.

Core Transformations

- What is a core transformation?

Core Transformations

- What is a core transformation?
- It's a unitary matrix, and

Core Transformations

- What is a core transformation?
- It's a unitary matrix, and
- it's essentially 2×2
- $C_2 = \begin{bmatrix} 1 & & & & \\ & \times & \times & & \\ & \times & \times & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}$

Core Transformations

- What is a core transformation?
- It's a unitary matrix, and
- it's essentially 2×2
- $C_2 = \begin{bmatrix} 1 & & & & \\ & \times & \times & & \\ & \times & \times & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}$
- Ex: Givens rotator, reflector, ...

Core Transformations

- What is a core transformation?
- It's a unitary matrix, and
- it's essentially 2×2
- $C_2 = \begin{bmatrix} 1 & & & & \\ & \times & \times & & \\ & \times & \times & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}$
- Ex: Givens rotator, reflector, ...
- We just wanted a generic term.

Core Transformations

$$\begin{bmatrix} 1 & & \\ & \times & \times \\ & \times & \times \end{bmatrix} \begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & 0 & \times \end{bmatrix}$$

Core Transformations

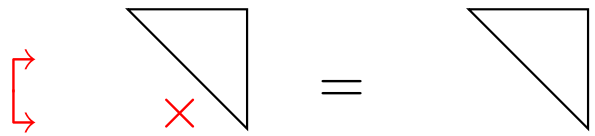
$$\begin{bmatrix} 1 & & \\ & \times & \times \\ & \times & \times \end{bmatrix} \begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & 0 & \times \end{bmatrix}$$

Abbreviated notation

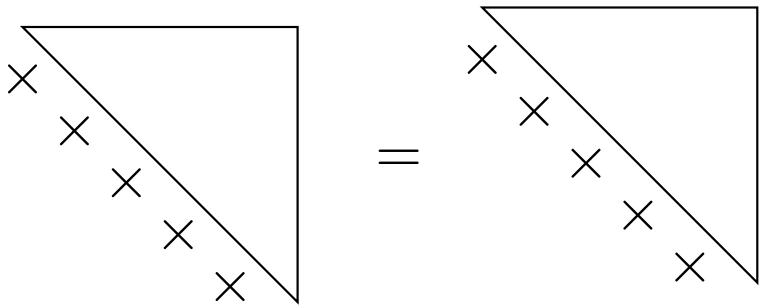
Core Transformations

$$\begin{bmatrix} 1 & & \\ & \times & \times \\ & \times & \times \end{bmatrix} \begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & 0 & \times \end{bmatrix}$$

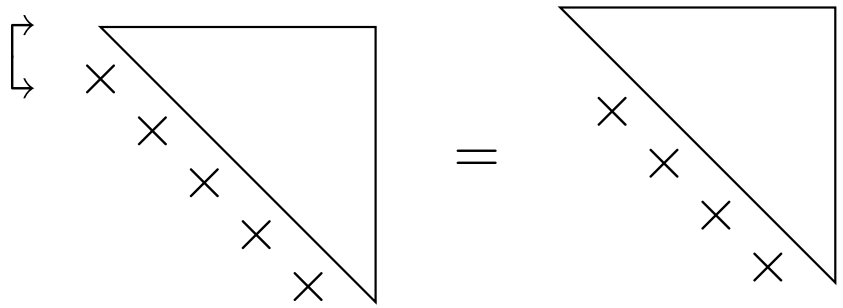
Abbreviated notation



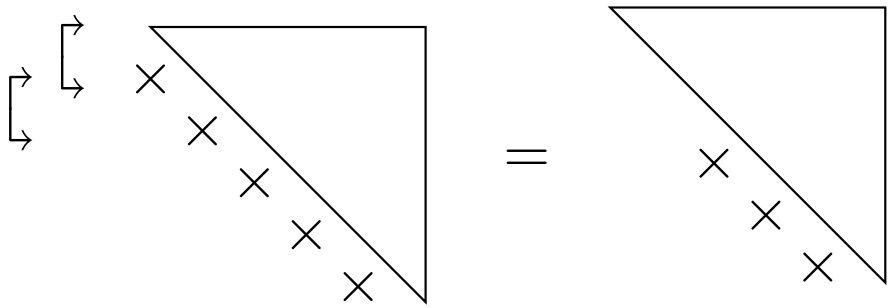
Hessenberg QR decomposition



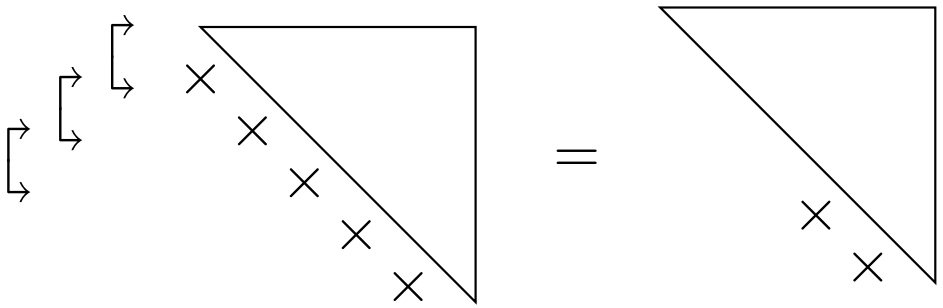
Hessenberg QR decomposition



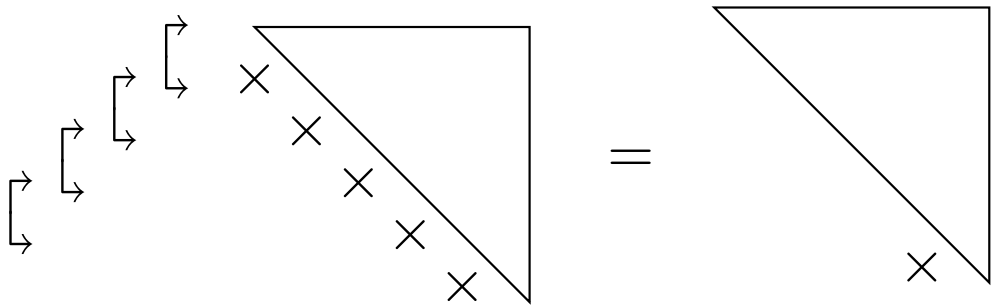
Hessenberg QR decomposition



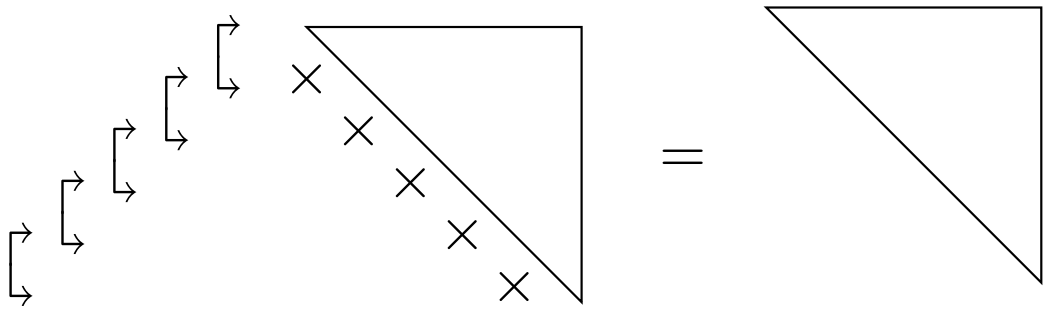
Hessenberg QR decomposition



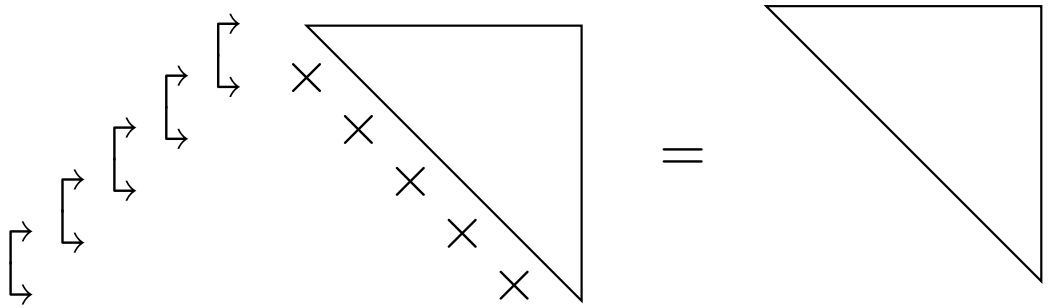
Hessenberg QR decomposition



Hessenberg QR decomposition

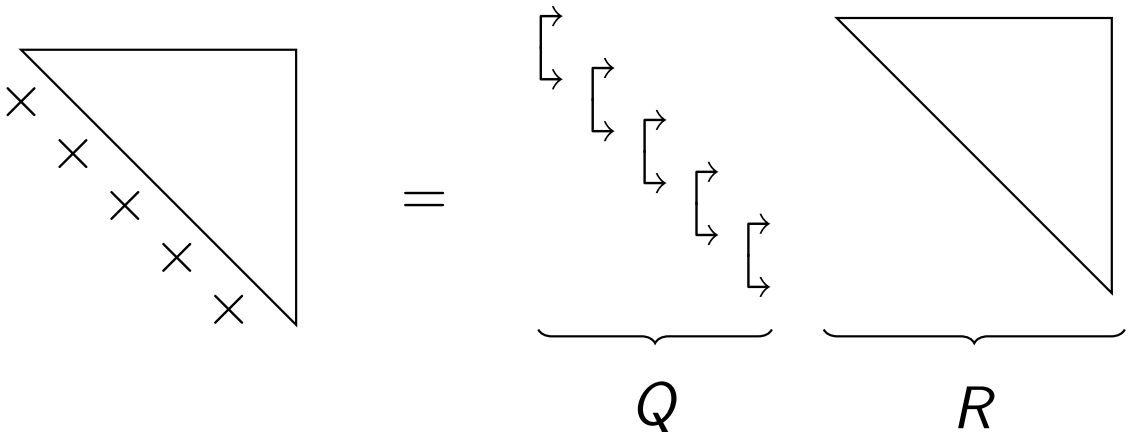


Hessenberg QR decomposition



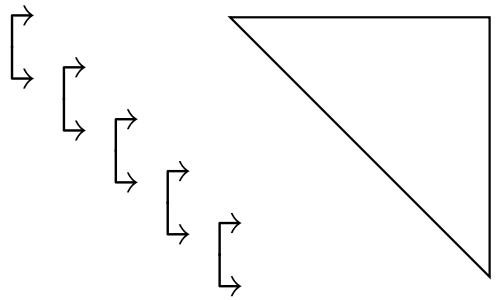
Now invert the core transformations.

Hessenberg QR decomposition



Our algorithms operate on the matrix in QR decomposed form.

$$A = QR =$$

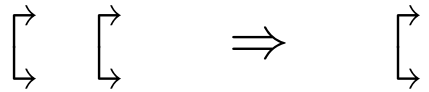


This is not inefficient.

We apply Francis's algorithm to this factored form.

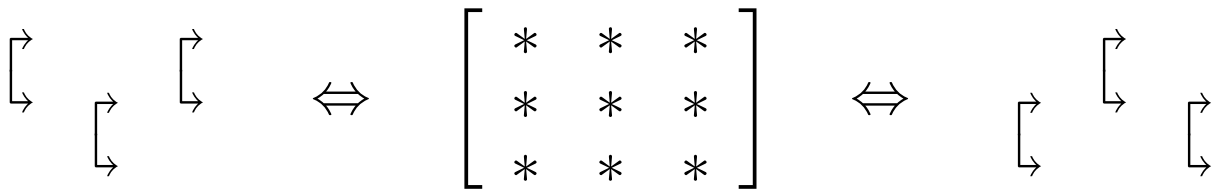
Operating on Core Transformations

Fusion



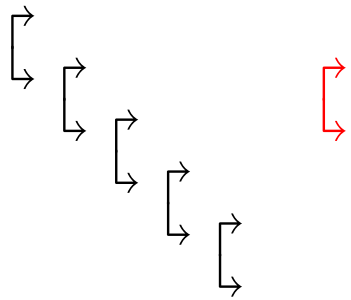
Operating on Core Transformations

Turnover



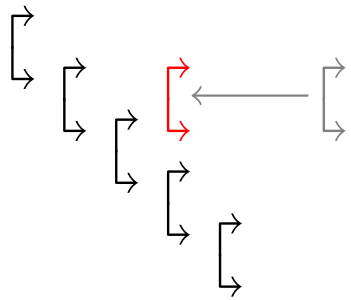
Operating on Core Transformations

Turnover as a shift-through operation



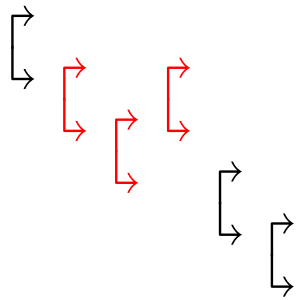
Operating on Core Transformations

Turnover as a shift-through operation



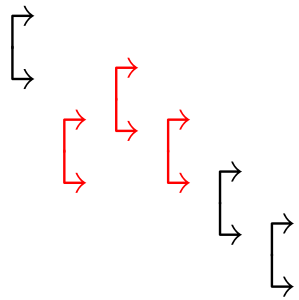
Operating on Core Transformations

Turnover as a shift-through operation



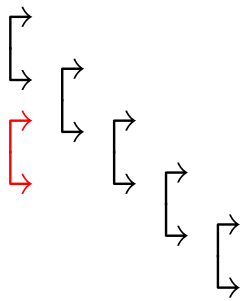
Operating on Core Transformations

Turnover as a shift-through operation



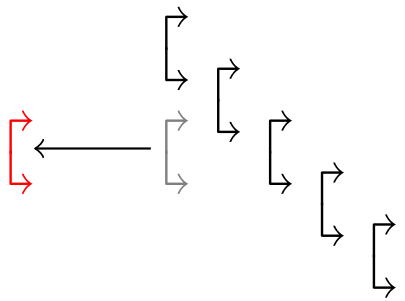
Operating on Core Transformations

Turnover as a shift-through operation



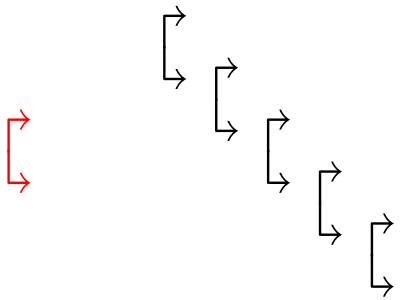
Operating on Core Transformations

Turnover as a shift-through operation



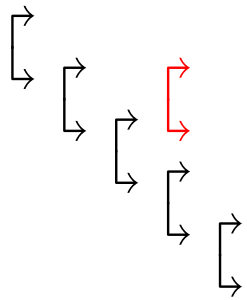
Operating on Core Transformations

Turnover as a shift-through operation



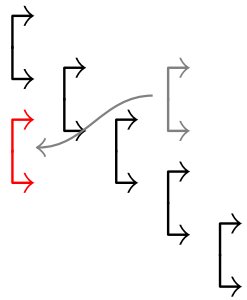
Operating on Core Transformations

Turnover as a shift-through operation : Abbreviated notation



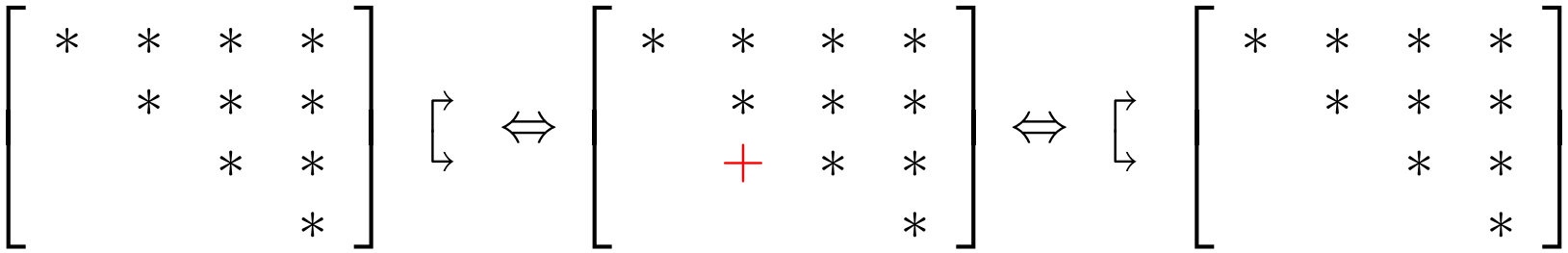
Operating on Core Transformations

Turnover as a shift-through operation : Abbreviated notation



Operating on Core Transformations

Passing a core transformation through a triangular matrix

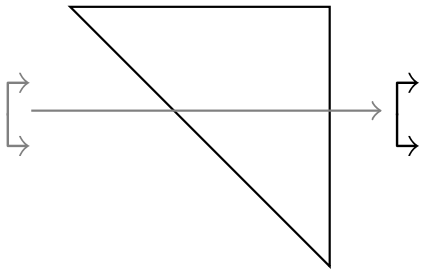


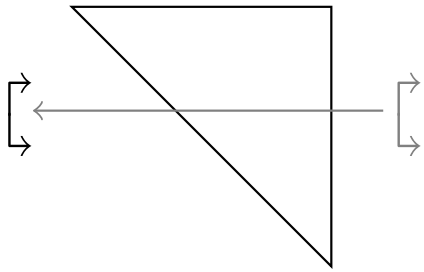
Cost is $O(n)$ flops.

Operating on Core Transformations

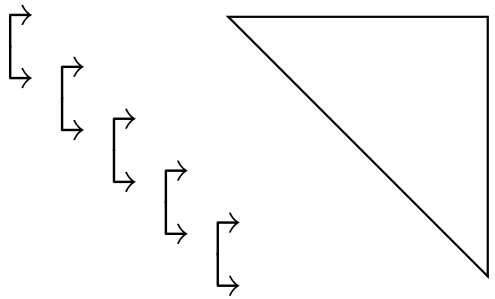
Passing a core transformation through a triangular matrix

Abbreviated Notation:

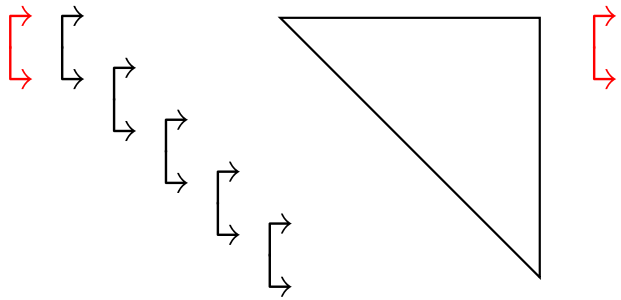




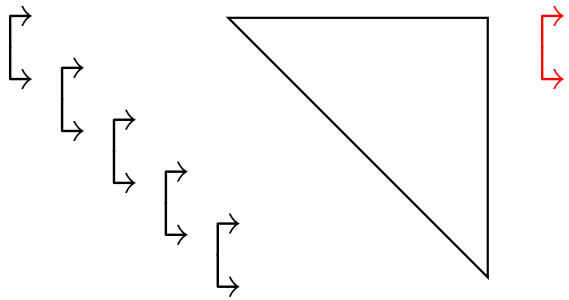
Core Chasing



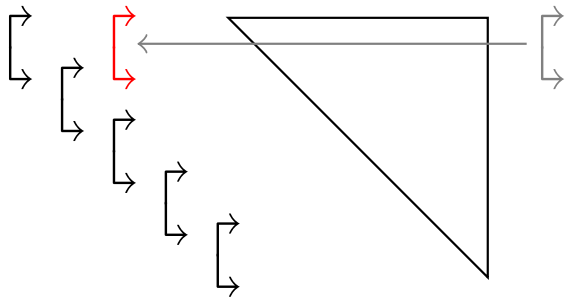
Core Chasing



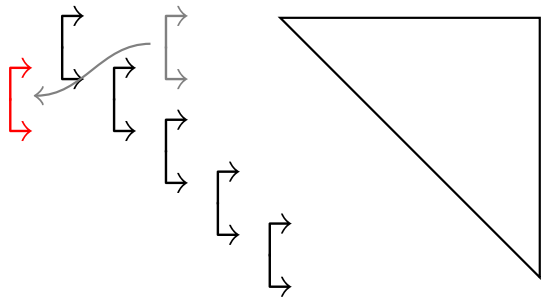
Core Chasing



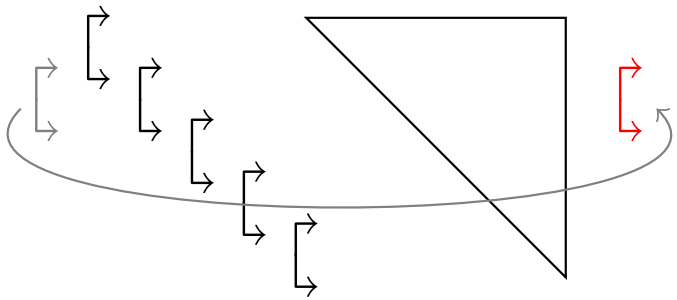
Core Chasing



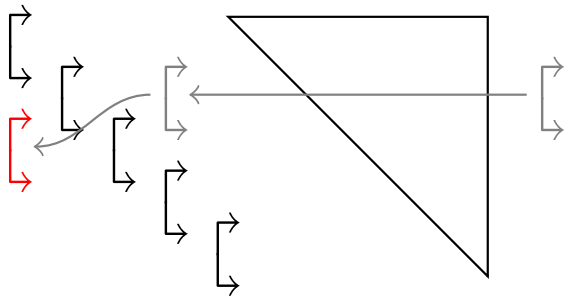
Core Chasing



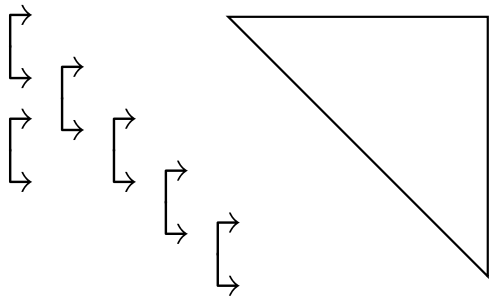
Core Chasing



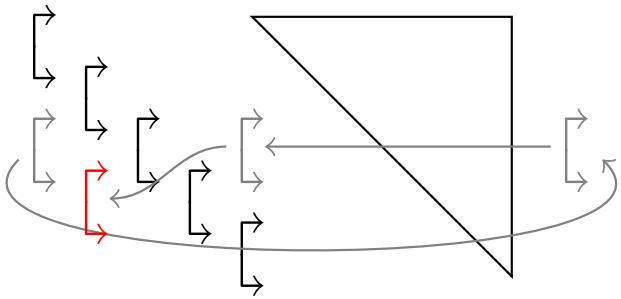
Core Chasing



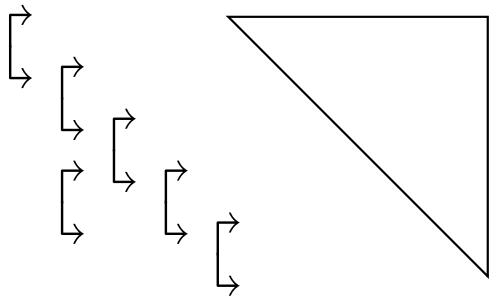
Core Chasing



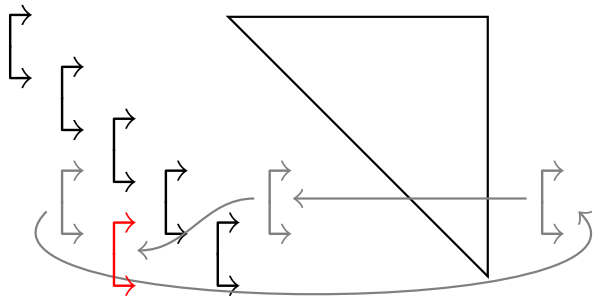
Core Chasing



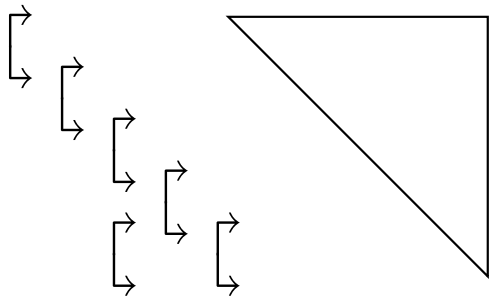
Core Chasing



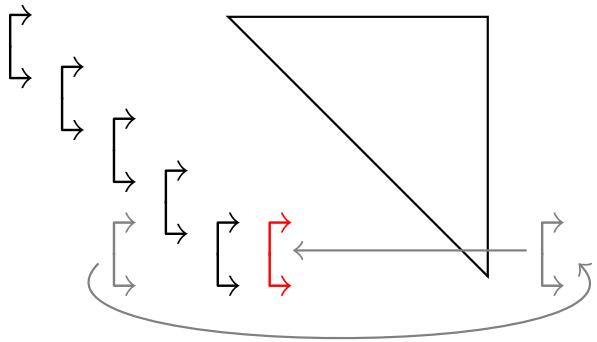
Core Chasing



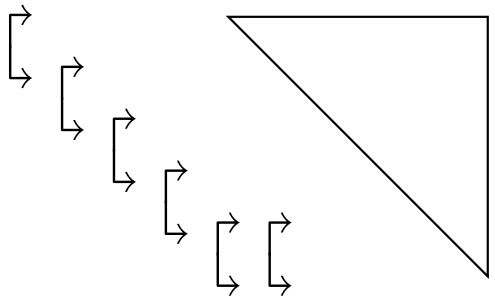
Core Chasing



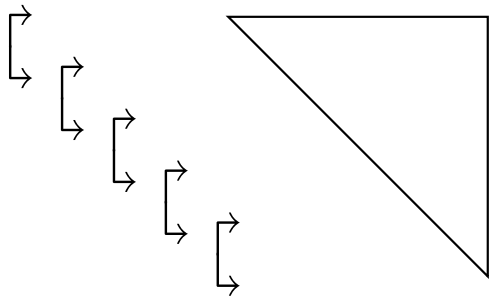
Core Chasing



Core Chasing



Core Chasing



Cost

Cost

- Most arithmetic in passing-through operation

Cost

- Most arithmetic in passing-through operation
- $O(n^2)$ flops per iteration . . .
- $O(n^3)$ total flops . . .
- about the same as for standard Francis iteration.

Are there any advantages?

Are there any advantages?

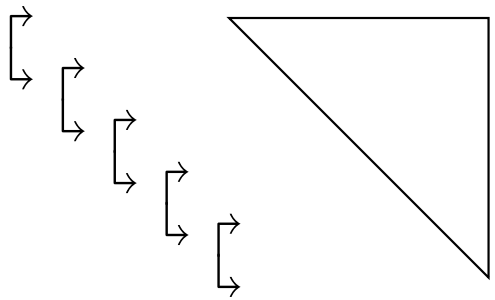
- unitary case

Are there any advantages?

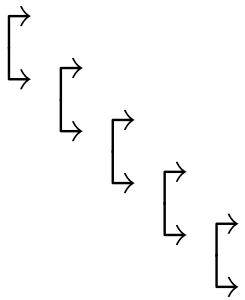
- unitary case
- companion case (unitary-plus-rank-one)

Unitary Case

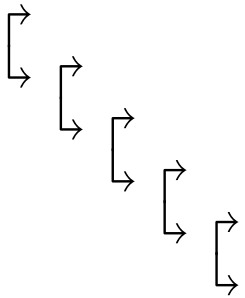
Unitary Case

$A = QR =$ 

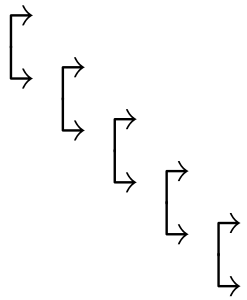
Unitary Case

$A = QR =$ 

Unitary Case

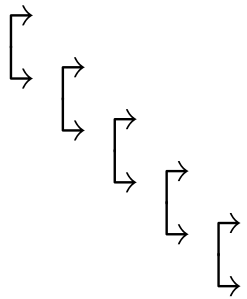
$A = QR =$ 

Unitary Case

$A = QR =$ 

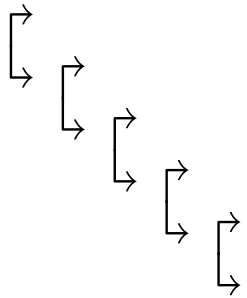
- Cost is $O(n)$ flops per iteration,

Unitary Case

$A = QR =$ 

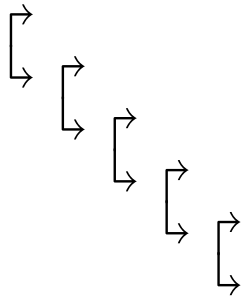
- Cost is $O(n)$ flops per iteration, $O(n^2)$ flops total.

Unitary Case

$A = QR =$ 

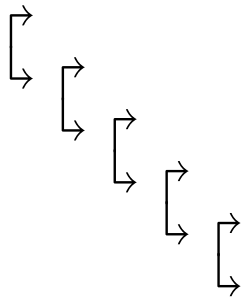
- Cost is $O(n)$ flops per iteration, $O(n^2)$ flops total.
- Storage requirement is $O(n)$.

Unitary Case

$A = QR =$ 

- Cost is $O(n)$ flops per iteration, $O(n^2)$ flops total.
- Storage requirement is $O(n)$.
- Gragg (1986)

Unitary Case

$A = QR =$ 

- Cost is $O(n)$ flops per iteration, $O(n^2)$ flops total.
- Storage requirement is $O(n)$.
- Gragg (1986)
- Ammar, Reichel, M. Stewart, Bunse-Gerstner, Elsner, He, W, ...

Companion Case

Companion Case

- $p(x) = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_0 = 0$
- monic polynomial

Companion Case

- $p(x) = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_0 = 0$
- monic polynomial
- companion matrix

$$A = \begin{bmatrix} 0 & & \cdots & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & -a_1 \\ & 1 & \ddots & \vdots & \vdots \\ & & \ddots & 0 & -a_{n-2} \\ & & & 1 & -a_{n-1} \end{bmatrix}$$

- . . . get the zeros of p by computing the eigenvalues.

Companion Case

- $p(x) = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_0 = 0$
- monic polynomial
- companion matrix

$$A = \begin{bmatrix} 0 & & \cdots & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & -a_1 \\ & 1 & \ddots & \vdots & \vdots \\ & & \ddots & 0 & -a_{n-2} \\ & & & 1 & -a_{n-1} \end{bmatrix}$$

- . . . get the zeros of p by computing the eigenvalues.
- MATLAB's `roots` command

Cost of solving companion eigenvalue problem

Cost of solving companion eigenvalue problem

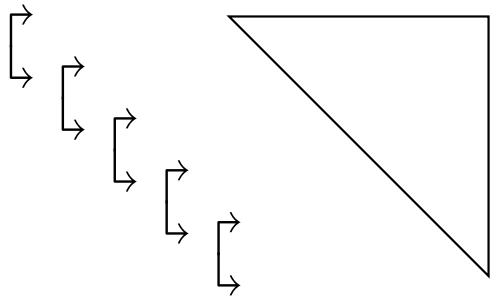
- If structure not exploited:
  - $O(n^2)$ storage, $O(n^3)$ flops
  - Francis's algorithm

Cost of solving companion eigenvalue problem

- If structure not exploited:
  - $O(n^2)$ storage, $O(n^3)$ flops
  - Francis's algorithm
- If structure exploited:
  - $O(n)$ storage, $O(n^2)$ flops
  - several methods proposed
  - data-sparse representation + Francis's algorithm

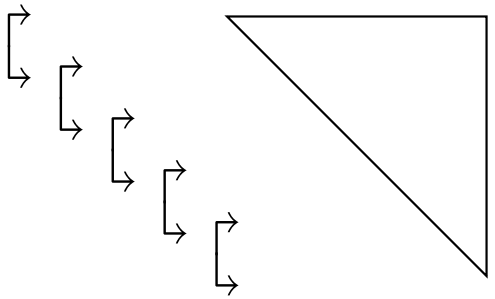
Representation of R

We store the QR decomposed form.

$A = QR =$ 

Representation of R

We store the QR decomposed form.

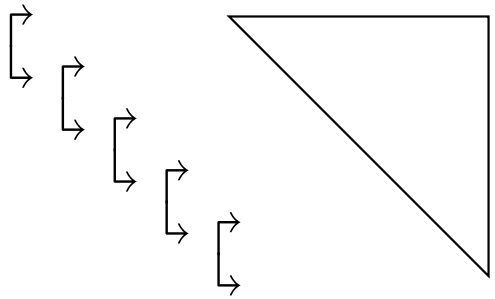
$$A = QR = \quad$$ 

where

$$R = \begin{bmatrix} 1 & 0 & \cdots & -a_1 \\ & 1 & & -a_2 \\ & & \ddots & \vdots \\ & & & -a_0 \end{bmatrix}.$$

Representation of R

We store the QR decomposed form.

$$A = QR = \quad$$ 

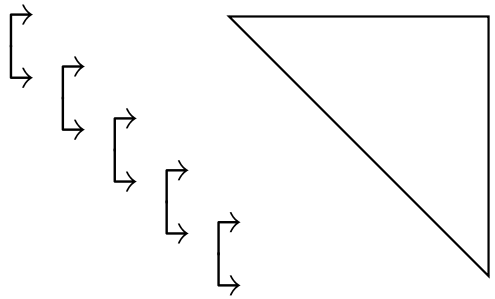
where

$$R = \begin{bmatrix} 1 & 0 & \cdots & -a_1 \\ & 1 & & -a_2 \\ & & \ddots & \vdots \\ & & & -a_0 \end{bmatrix}.$$

- This is unitary-plus-rank-one.

Representation of R

We store the QR decomposed form.

$$A = QR = \quad$$ 

where

$$R = \begin{bmatrix} 1 & 0 & \cdots & -a_1 \\ & 1 & & -a_2 \\ & & \ddots & \vdots \\ & & & -a_0 \end{bmatrix}.$$

- This is unitary-plus-rank-one.
- How do we store it?

Representation of R

- Jared covered this yesterday.

Representation of R

- Jared covered this yesterday.
- Add a row and column to R.

Representation of R

- Jared covered this yesterday.
- Add a row and column to R.

$$\tilde{R} = \left[\begin{array}{cccc|c} 1 & 0 & \cdots & -a_1 & 0 \\ & 1 & & -a_2 & 0 \\ & & \ddots & \vdots & \vdots \\ & & & -a_0 & 1 \\ \hline 0 & 0 & \cdots & 0 & 0 \end{array}\right].$$

Representation of R

- Jared covered this yesterday.
- Add a row and column to R.

$$\tilde{R} = \left[\begin{array}{cccc|c} 1 & 0 & \cdots & -a_1 & 0 \\ & 1 & & -a_2 & 0 \\ & & \ddots & \vdots & \vdots \\ & & & -a_0 & 1 \\ \hline 0 & 0 & \cdots & 0 & 0 \end{array}\right].$$

- This is still unitary-plus-rank-one.

Representation of R

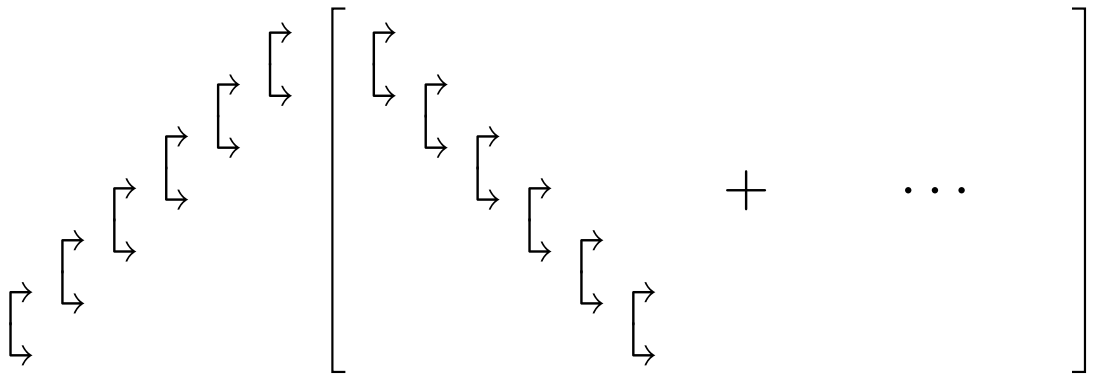
$$\tilde{R} =$$

Representation of R

$$\tilde{R} = C_n^* \cdots C_1^* (B_1 \cdots B_n + e_1 y^T)$$

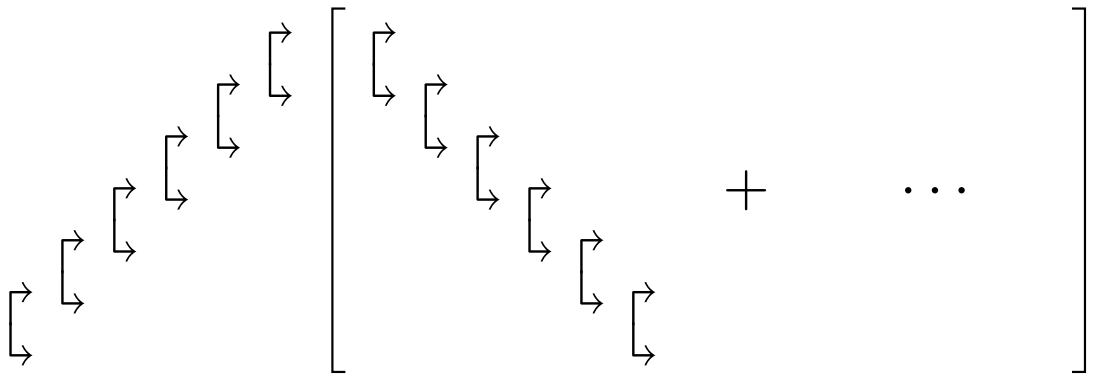
Representation of R

$$\tilde{R} = C_n^* \cdots C_1^* (B_1 \cdots B_n + e_1 y^T)$$



Representation of R

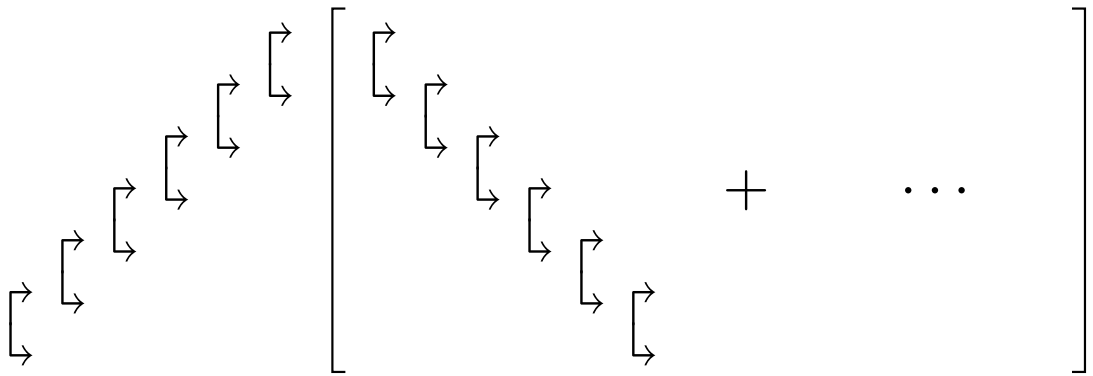
$$\tilde{R} = C_n^* \cdots C_1^* (B_1 \cdots B_n + e_1 y^T)$$



. . . and we don't have to store the rank-one part!

Representation of R

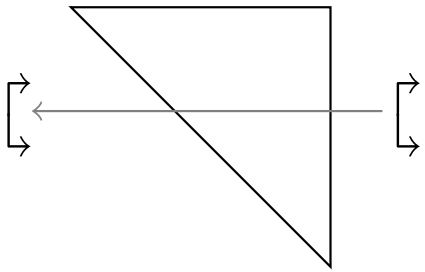
$$\tilde{R} = C_n^* \cdots C_1^* (B_1 \cdots B_n + e_1 y^T)$$



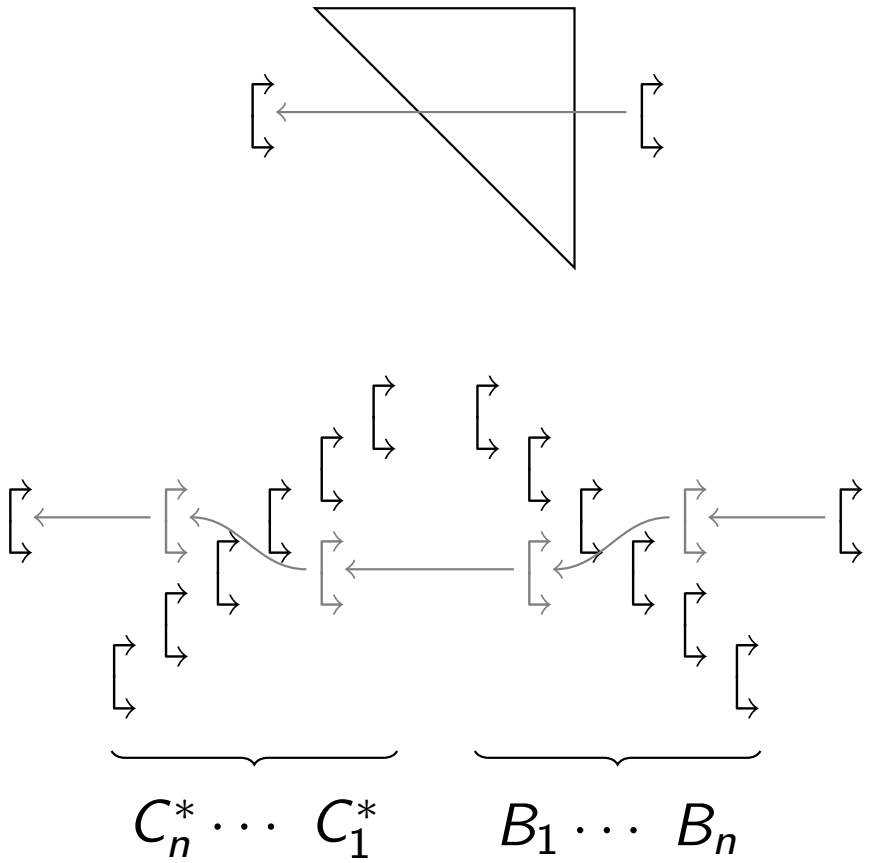
. . . and we don't have to store the rank-one part!

Storage is $O(n)$.

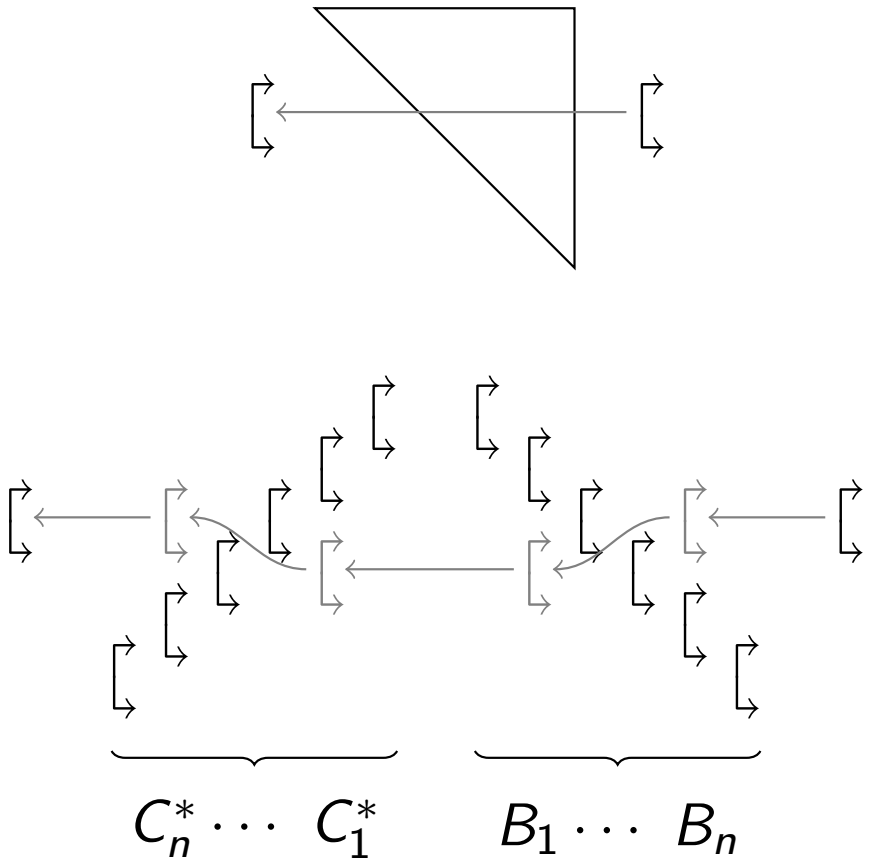
Passing a core transformation through R



Passing a core transformation through R



Passing a core transformation through R



Cost: $O(n)$ flops per iteration, $O(n^2)$ flops total.

The rank-one part

Recovering y^T from the core transformations

The rank-one part

Recovering y^T from the core transformations

$$0 \quad = \quad e_{n+1}^T \tilde{R}$$

The rank-one part

Recovering y^T from the core transformations

$$\begin{aligned} 0 &= e_{n+1}^T \tilde{R} \\ &= e_{n+1}^T C_n^* \cdots C_1^* (B_1 \cdots B_n + e_1 y^T) \end{aligned}$$

The rank-one part

Recovering y^T from the core transformations

$$\begin{aligned} 0 &= e_{n+1}^T \tilde{R} \\ &= e_{n+1}^T C_n^* \cdots C_1^* (B_1 \cdots B_n + e_1 y^T) \\ &= e_{n+1}^T C^* (B + e_1 y^T) \end{aligned}$$

The rank-one part

Recovering y^T from the core transformations

$$\begin{aligned} 0 &= e_{n+1}^T \tilde{R} \\ &= e_{n+1}^T C_n^* \cdots C_1^* (B_1 \cdots B_n + e_1 y^T) \\ &= e_{n+1}^T C^* (B + e_1 y^T) \\ &= e_{n+1}^T C^* B + (e_{n+1}^T C^* e_1) y^T \end{aligned}$$

The rank-one part

Recovering y^T from the core transformations

$$\begin{aligned} 0 &= e_{n+1}^T \tilde{R} \\ &= e_{n+1}^T C_n^* \cdots C_1^* (B_1 \cdots B_n + e_1 y^T) \\ &= e_{n+1}^T C^* (B + e_1 y^T) \\ &= e_{n+1}^T C^* B + (e_{n+1}^T C^* e_1) y^T \end{aligned}$$

so

$$y^T = -\rho^{-1} e_{n+1}^T C^* B, \quad \text{where} \quad \rho = e_{n+1}^T C^* e_1.$$

The rank-one part

Recovering y^T from the core transformations

$$\begin{aligned} 0 &= e_{n+1}^T \tilde{R} \\ &= e_{n+1}^T C_n^* \cdots C_1^* (B_1 \cdots B_n + e_1 y^T) \\ &= e_{n+1}^T C^* (B + e_1 y^T) \\ &= e_{n+1}^T C^* B + (e_{n+1}^T C^* e_1) y^T \end{aligned}$$

so

$$y^T = -\rho^{-1} e_{n+1}^T C^* B, \quad \text{where} \quad \rho = e_{n+1}^T C^* e_1.$$

Note: $|\rho| = \|y\|_2^{-1} \neq 0$.

The rank-one part

Recovering y^T from the core transformations

$$\begin{aligned} 0 &= e_{n+1}^T \tilde{R} \\ &= e_{n+1}^T C_n^* \cdots C_1^* (B_1 \cdots B_n + e_1 y^T) \\ &= e_{n+1}^T C^* (B + e_1 y^T) \\ &= e_{n+1}^T C^* B + (e_{n+1}^T C^* e_1) y^T \end{aligned}$$

so

$$y^T = -\rho^{-1} e_{n+1}^T C^* B, \quad \text{where} \quad \rho = e_{n+1}^T C^* e_1.$$

Note: $|\rho| = \|y\|_2^{-1} \neq 0$.

- This gives us a way to compute y at any time.

The rank-one part

Recovering y^T from the core transformations

$$\begin{aligned} 0 &= e_{n+1}^T \tilde{R} \\ &= e_{n+1}^T C_n^* \cdots C_1^* (B_1 \cdots B_n + e_1 y^T) \\ &= e_{n+1}^T C^* (B + e_1 y^T) \\ &= e_{n+1}^T C^* B + (e_{n+1}^T C^* e_1) y^T \end{aligned}$$

so

$$y^T = -\rho^{-1} e_{n+1}^T C^* B, \quad \text{where} \quad \rho = e_{n+1}^T C^* e_1.$$

Note: $|\rho| = \|y\|_2^{-1} \neq 0$.

- This gives us a way to compute y at any time.
- In fact we never need to use it, ...

The rank-one part

Recovering y^T from the core transformations

$$\begin{aligned} 0 &= e_{n+1}^T \tilde{R} \\ &= e_{n+1}^T C_n^* \cdots C_1^* (B_1 \cdots B_n + e_1 y^T) \\ &= e_{n+1}^T C^* (B + e_1 y^T) \\ &= e_{n+1}^T C^* B + (e_{n+1}^T C^* e_1) y^T \end{aligned}$$

so

$$y^T = -\rho^{-1} e_{n+1}^T C^* B, \quad \text{where} \quad \rho = e_{n+1}^T C^* e_1.$$

Note: $|\rho| = \|y\|_2^{-1} \neq 0$.

- This gives us a way to compute y at any time.
- In fact we never need to use it, . . .
- . . . but it comes in handy for the backward error analysis.

Other things we can do

We can also handle

Other things we can do

We can also handle

- generalized eigenvalue problem
- companion pencil

Other things we can do

We can also handle

- generalized eigenvalue problem
- companion pencil
- matrix polynomial eigenvalue problems (with L. Robol)

Other things we can do

We can also handle

- generalized eigenvalue problem
- companion pencil
- matrix polynomial eigenvalue problems (with L. Robol)
- symmetric-plus-rank-one (via Cayley transform)

Other things we can do

We can also handle

- generalized eigenvalue problem
- companion pencil
- matrix polynomial eigenvalue problems (with L. Robol)
- symmetric-plus-rank-one (via Cayley transform)
- completely unstructured problems

Other things we can do

We can also handle

- generalized eigenvalue problem
- companion pencil
- matrix polynomial eigenvalue problems (with L. Robol)
- symmetric-plus-rank-one (via Cayley transform)
- completely unstructured problems
- generalizations of Hessenberg form.

Other things we can do

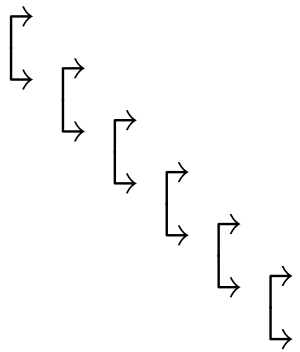
We can also handle

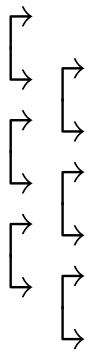
- generalized eigenvalue problem
- companion pencil
- matrix polynomial eigenvalue problems (with L. Robol)
- symmetric-plus-rank-one (via Cayley transform)
- completely unstructured problems
- generalizations of Hessenberg form.

Monograph in progress (100+ pp?)

Beyond Hessenberg form

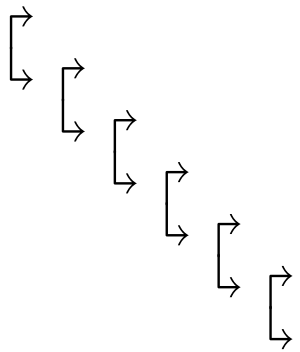
Form of Q can be upper Hessenberg, so-called CMV, . . .

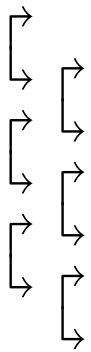




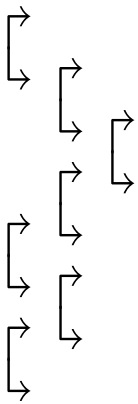
Beyond Hessenberg form

Form of Q can be upper Hessenberg, so-called CMV, . . .





or "random" twisted . . .



Backward Stability

Backward Stability

- All of these algorithms are normwise backward stable.

Backward Stability

- All of these algorithms are normwise backward stable.
- This is “obvious”,

Backward Stability

- All of these algorithms are normwise backward stable.
- This is "obvious", but our storage schemes are unorthodox.

Backward Stability

- All of these algorithms are normwise backward stable.
- This is "obvious", but our storage schemes are unorthodox.
- So let's take a closer look.

Backward Stability

- All of these algorithms are normwise backward stable.
- This is "obvious", but our storage schemes are unorthodox.
- So let's take a closer look.

In exact arithmetic . . .

$$\hat{A} = U^* A U$$

Backward Stability

- All of these algorithms are normwise backward stable.
- This is "obvious", but our storage schemes are unorthodox.
- So let's take a closer look.

In exact arithmetic . . .

$$\hat{A} = U^* A U$$

In floating-point arithmetic . . .

$$\hat{A} = U^*(A + E)U$$

Is it true that $\|E\| \approx u\|A\|$? (u is unit roundoff.)

Backward Stability

- All of these algorithms are normwise backward stable.
- This is "obvious", but our storage schemes are unorthodox.
- So let's take a closer look.

In exact arithmetic . . .

$$\hat{A} = U^* A U$$

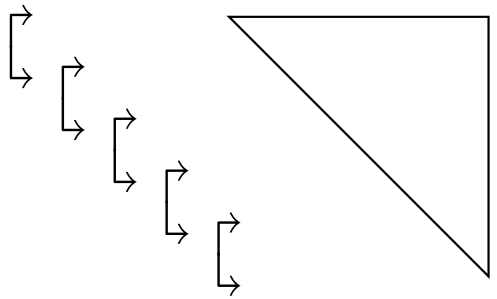
In floating-point arithmetic . . .

$$\hat{A} = U^*(A + E)U$$

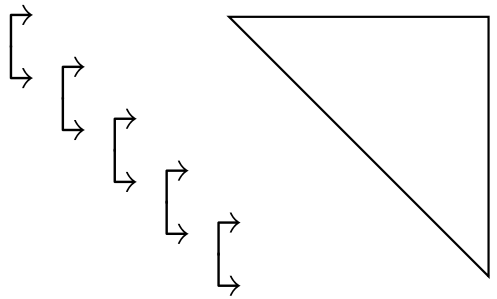
Is it true that $\|E\| \approx u\|A\|$? (u is unit roundoff.)

If so, the algorithm is normwise backward stable.

Backward Stability

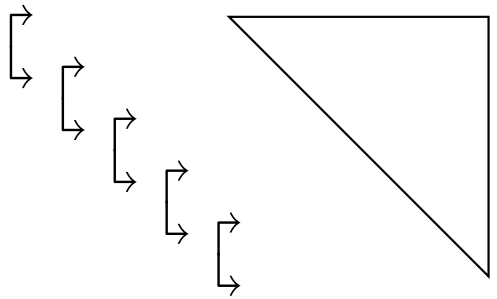
$A = QR =$ 

Backward Stability

$A = QR =$ 

$$\hat{A} = \hat{Q}\hat{R} = U^*QRU = (U^*QV)(V^*RU)$$

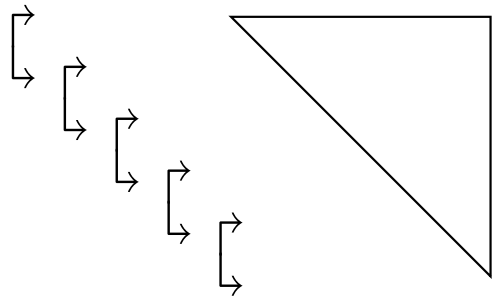
Backward Stability

$A = QR =$ 

$$\hat{A} = \hat{Q}\hat{R} = U^*QRU = (U^*QV)(V^*RU)$$

$$\hat{Q} = U^*QV \qquad \hat{R} = V^*RU$$

Backward Stability

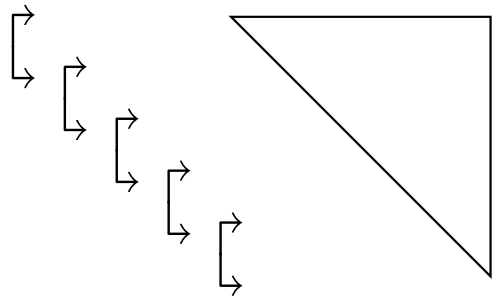
$A = QR =$ 

$$\hat{A} = \hat{Q}\hat{R} = U^*QRU = (U^*QV)(V^*RU)$$

$$\hat{Q} = U^*QV \qquad \hat{R} = V^*RU$$

$$\hat{Q} = U^*(Q + E_q)V \qquad \hat{R} = V^*(R + E_r)U$$

Backward Stability

$A = QR =$ 

$$\hat{A} = \hat{Q}\hat{R} = U^*QRU = (U^*QV)(V^*RU)$$

$$\hat{Q} = U^*QV \qquad \hat{R} = V^*RU$$

$$\hat{Q} = U^*(Q + E_q)V \qquad \hat{R} = V^*(R + E_r)U$$

$$\| E_q \| \approx u \qquad \| E_r \| \approx u \| R \| = u \| A \|$$

. . . assuming R is stored in the conventional way.

Backward Stability

$$\hat{A} = \hat{Q}\hat{R} = U^* QVV^* RU = U^* QRU = U^* AU$$

Backward Stability

$$\hat{A} = \hat{Q}\hat{R} = U^*QVV^*RU = U^*QRU = U^*AU$$

$$\begin{aligned} \hat{A} = \hat{Q}\hat{R} &= U^*(Q + E_q)(R + E_r)U \\ &= U^*(QR + E_qR + QE_r + E_qE_r)U \\ &= U^*(A + E)U, \end{aligned}$$

where

$$E = E_qR + QE_r + O(u^2), \quad \text{so} \quad \|E\| \approx u\|R\|.$$

Backward Stability

Now consider the unitary-plus-rank-one case: $R = C^*(B + e_1 y^T)$.

Backward Stability

Now consider the unitary-plus-rank-one case: $R = C^*(B + e_1 y^T)$.

Is it still true that $\hat{R} = V^*(R + E_r)U$ with $\| E_r \| \approx u \| R \|$?

Backward Stability

Now consider the unitary-plus-rank-one case: $R = C^*(B + e_1 y^T)$.

Is it still true that $\hat{R} = V^*(R + E_r)U$ with $\| E_r \| \approx u \| R \|$?

$$R = C^*(B + e_1 y^T) = C^* B + C^* e_1 y^T$$

Backward Stability

Now consider the unitary-plus-rank-one case: $R = C^*(B + e_1 y^T)$.

Is it still true that $\hat{R} = V^*(R + E_r)U$ with $\| E_r \| \approx u \| R \|$?

$$R = C^*(B + e_1 y^T) = C^* B + C^* e_1 y^T$$

$$R_u = C^* B \qquad R_o = C^* e_1 y^T$$

Backward Stability

Now consider the unitary-plus-rank-one case: $R = C^*(B + e_1 y^T)$.

Is it still true that $\hat{R} = V^*(R + E_r)U$ with $\| E_r \| \approx u \| R \|$?

$$R = C^*(B + e_1 y^T) = C^* B + C^* e_1 y^T$$

$$R_u = C^* B \qquad R_o = C^* e_1 y^T$$

$$\hat{R}_u = V^* R_u U \qquad \hat{R}_o = V^* R_o U$$

Backward Stability

Let's consider the unitary part first.

Backward Stability

Let's consider the unitary part first.

$$\hat{R}_u = V^* R_u U = V^* C^* B U = (V^* C^* W)(W^* B U) = \hat{C}^* \hat{B}$$

Backward Stability

Let's consider the unitary part first.

$$\hat{R}_u = V^* R_u U = V^* C^* B U = (V^* C^* W)(W^* B U) = \hat{C}^* \hat{B}$$

$$\hat{C} = W^* C V \qquad \hat{B} = W^* B U$$

Backward Stability

Let's consider the unitary part first.

$$\hat{R}_u = V^* R_u U = V^* C^* B U = (V^* C^* W)(W^* B U) = \hat{C}^* \hat{B}$$

$$\hat{C} = W^* C V \qquad \hat{B} = W^* B U$$

$$\hat{C} = W^*(C + E_c)V \qquad \hat{B} = W^*(B + E_b)U$$

$$\| E_c \| \approx u \qquad \| E_b \| \approx u$$

Backward Stability

Let's consider the unitary part first.

$$\hat{R}_u = V^* R_u U = V^* C^* B U = (V^* C^* W)(W^* B U) = \hat{C}^* \hat{B}$$

$$\hat{C} = W^* C V \qquad \hat{B} = W^* B U$$

$$\hat{C} = W^*(C + E_c)V \qquad \hat{B} = W^*(B + E_b)U$$

$$\| E_c \| \approx u \qquad \| E_b \| \approx u$$

Put them together!

Backward Stability

Now consider the rank-one part.

Backward Stability

Now consider the rank-one part.

$$R_o = C^* e_1 y^T,$$

Backward Stability

Now consider the rank-one part.

$$R_o = C^* e_1 y^T, \quad \text{where} \quad y^T = -\rho^{-1} e_{n+1}^T C^* B$$

Backward Stability

Now consider the rank-one part.

$$R_o = C^* e_1 y^T, \quad \text{where} \quad y^T = -\rho^{-1} e_{n+1}^T C^* B$$

so

$$R_o = -\rho^{-1} C^* e_1 e_{n+1}^T C^* B$$

Backward Stability

Now consider the rank-one part.

$$R_o = C^* e_1 y^T, \quad \text{where} \quad y^T = -\rho^{-1} e_{n+1}^T C^* B$$

so

$$R_o = -\rho^{-1} C^* e_1 e_{n+1}^T C^* B$$

$$\hat{R}_o = V^* R_o U = -\rho^{-1} V^* C^* e_1 e_{n+1}^T C^* B U$$

Backward Stability

Now consider the rank-one part.

$$R_o = C^* e_1 y^T, \quad \text{where} \quad y^T = -\rho^{-1} e_{n+1}^T C^* B$$

so

$$R_o = -\rho^{-1} C^* e_1 e_{n+1}^T C^* B$$

$$\hat{R}_o = V^* R_o U = -\rho^{-1} V^* C^* e_1 e_{n+1}^T C^* B U$$

$$\hat{R}_o = V^* R_o U = -\rho^{-1} V^* (C + E_c)^* e_1 e_{n+1}^T (C + E_c)^* (B + E_b) U$$

Backward Stability

Now consider the rank-one part.

$$R_o = C^* e_1 y^T, \quad \text{where} \quad y^T = -\rho^{-1} e_{n+1}^T C^* B$$

so

$$R_o = -\rho^{-1} C^* e_1 e_{n+1}^T C^* B$$

$$\hat{R}_o = V^* R_o U = -\rho^{-1} V^* C^* e_1 e_{n+1}^T C^* B U$$

$$\hat{R}_o = V^* R_o U = -\rho^{-1} V^* (C + E_c)^* e_1 e_{n+1}^T (C + E_c)^* (B + E_b) U$$

Backward stability follows.

Advantages

Advantages of our approach

Advantages

Advantages of our approach

- Fast algorithms for certain special structures

Advantages

Advantages of our approach

- Fast algorithms for certain special structures
  - unitary
  - unitary-plus-rank-one
  - symmetric
  - symmetric-plus-rank-one

Advantages

Advantages of our approach

- Fast algorithms for certain special structures
  - unitary
  - unitary-plus-rank-one
  - symmetric
  - symmetric-plus-rank-one
- . . . but also recommended for problems with no special structure.

Advantages

Advantages of our approach

- Fast algorithms for certain special structures
  - unitary
  - unitary-plus-rank-one
  - symmetric
  - symmetric-plus-rank-one
- . . . but also recommended for problems with no special structure.
  - chase multiple bulges for parallelism, BLAS3 speed (in theory)

Advantages

Advantages of our approach

- Fast algorithms for certain special structures
  - unitary
  - unitary-plus-rank-one
  - symmetric
  - symmetric-plus-rank-one
- . . . but also recommended for problems with no special structure.
  - chase multiple bulges for parallelism, BLAS3 speed (in theory)
  - better deflation strategy

Deflation

Deflation

- Conventional deflation:

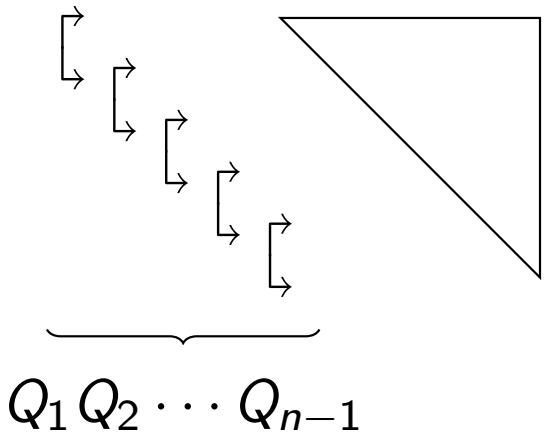
  Decouple when $|a_{j+1,j}| < u\,(|a_{jj}| + |a_{j+1,j+1}|)$.

Deflation

- Conventional deflation:

  Decouple when $|a_{j+1,j}| < u\,(|a_{jj}| + |a_{j+1,j+1}|)$.

- Our deflation:



$$Q_1 Q_2 \cdots Q_{n-1}$$

$$Q_j = \begin{bmatrix} c_j & -s_j \\ s_j & \overline{c_j} \end{bmatrix}$$

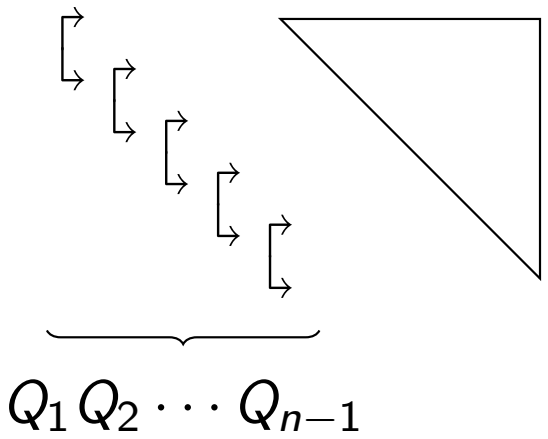
Decouple when $|s_j| < u$.

Deflation

- Conventional deflation:

  Decouple when $|a_{j+1,j}| < u\,(|a_{jj}| + |a_{j+1,j+1}|)$.

- Our deflation:



$$Q_1 Q_2 \cdots Q_{n-1}$$

$$Q_j = \begin{bmatrix} c_j & -s_j \\ s_j & \overline{c_j} \end{bmatrix}$$

  Decouple when $|s_j| < u$.

- Stronger backward stability (Mach and Vandebril 2014)

Quick Summary

Quick Summary

- We took a new look at Francis's algorithm.

Quick Summary

- We took a new look at Francis's algorithm.
- considered QR decomposed form.

Quick Summary

- We took a new look at Francis's algorithm.
- considered QR decomposed form.
- chased cores instead of bulges.

Quick Summary

- We took a new look at Francis's algorithm.
- considered QR decomposed form.
- chased cores instead of bulges.
- We demonstrated some advantages.

Quick Summary

- We took a new look at Francis's algorithm.
- considered QR decomposed form.
- chased cores instead of bulges.
- We demonstrated some advantages.

Thank you for your attention.