

# Core-Chasing Algorithms for Eigenvalue Computation

David S. Watkins

Department of Mathematics  
Washington State University

November, 2015

# 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)
- Many applications

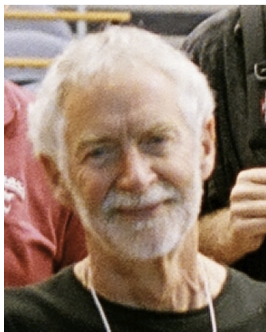
# Today's Topic

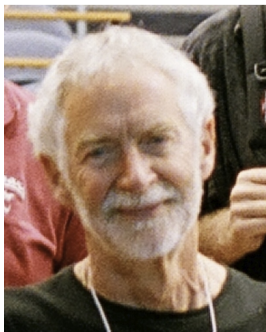
- The matrix eigenvalue problem
- $A \in \mathbb{C}^{n \times n}$
- Find the eigenvalues ( . . . vectors, invariant subspaces)
- Many applications
- Interest dates back to the very beginning of the electronic computing era.

# Today's Topic

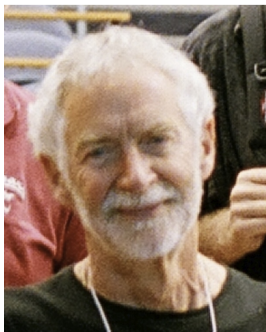
- The matrix eigenvalue problem
- $A \in \mathbb{C}^{n \times n}$
- Find the eigenvalues ( . . . vectors, invariant subspaces)
- Many applications
- Interest dates back to the very beginning of the electronic computing era.
- Nobody knew how to do it.



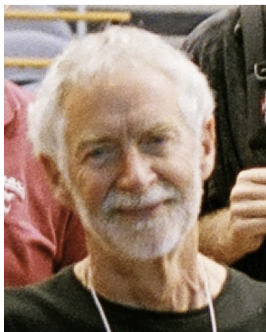




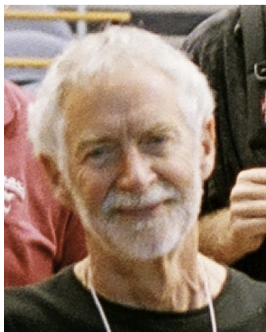
- invented the winning algorithm in 1959.



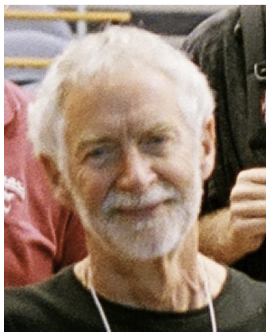
- invented the winning algorithm in 1959.
- commonly called: QR algorithm



- invented the winning algorithm in 1959.
- commonly called: QR algorithm
- more precisely: implicitly shifted QR algorithm



- invented the winning algorithm in 1959.
- commonly called: QR algorithm
- more precisely: implicitly shifted QR algorithm
- better yet: Francis's algorithm,



- invented the winning algorithm in 1959.
- commonly called: QR algorithm
- more precisely: implicitly shifted QR algorithm
- better yet: Francis's algorithm, bulge-chasing algorithm.

# Francis's algorithm (superficial description)

# Francis's algorithm (superficial description)

- upper Hessenberg form

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

- unitary similarity transformation
- direct method ( $O(n^3)$  flops)



# Francis's algorithm (superficial description)

- upper Hessenberg form

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

- unitary similarity transformation
- direct method ( $O(n^3)$  flops)
- Francis: Iterate

# Francis's algorithm (superficial description)

- upper Hessenberg form

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

- unitary similarity transformation
- direct method ( $O(n^3)$  flops)
- Francis: Iterate
- Drive toward triangular form.

# Francis's algorithm (superficial description)

- upper Hessenberg form

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

- unitary similarity transformation
- direct method ( $O(n^3)$  flops)
- Francis: Iterate
- Drive toward triangular form.
- (Galois theory)

# Francis's algorithm (superficial description)

Chasing the bulge

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

# Francis's algorithm (superficial description)

Chasing the bulge

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ + & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix}$$

# Francis's algorithm (superficial description)

Chasing the bulge

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & + & * & * & * \\ & & & * & * \end{bmatrix}$$

# Francis's algorithm (superficial description)

Chasing the bulge

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & + & * \\ & & & & * \end{bmatrix}$$

# Francis's algorithm (superficial description)

Chasing the bulge

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



# Francis's algorithm (superficial description)

Chasing the bulge

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

- iteration complete!

# Francis's algorithm (superficial description)

Chasing the bulge

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

- iteration complete!
- repeated iterations  $\Rightarrow$  triangular form

# Francis's algorithm (superficial description)

Chasing the bulge

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

- iteration complete!
- repeated iterations  $\Rightarrow$  triangular form
- This is the *single-shift* algorithm.

# Francis's algorithm (superficial description)

Chasing the bulge

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

- iteration complete!
- repeated iterations  $\Rightarrow$  triangular form
- This is the *single-shift* algorithm.
- *Double-shift* algorithm chases a bigger bulge.

## Computational Cost

- $O(n^2)$  flops per iteration
- $O(n)$  total iterations
- $O(n^3)$  total flops

For details see ...

For details see ...

- Golub and Van Loan, *Matrix Computations*, 4th Ed.

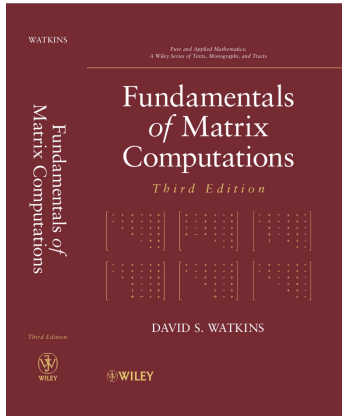
For details see ...

- Golub and Van Loan, *Matrix Computations*, 4th Ed.
- Watkins, *Fundamentals of Matrix Computations*, 3rd Ed.



For details see ...

- Golub and Van Loan, *Matrix Computations*, 4th Ed.
- Watkins, *Fundamentals of Matrix Computations*, 3rd Ed.



# My History with this Topic

# My History with this Topic

- Understanding the QR algorithm, SIAM Rev., 1982

# My History with this Topic

- Understanding the QR algorithm, SIAM Rev., 1982
- Fundamentals of Matrix Computations, Wiley, 1991

# My History with this Topic

- Understanding the QR algorithm, SIAM Rev., 1982
- Fundamentals of Matrix Computations, Wiley, 1991
- Some perspectives on the eigenvalue problem, 1993
- QR-like algorithms—an overview of convergence theory and practice, AMS proceedings, 1996
- QR-like algorithms for eigenvalue problems, JCAM, 2000
- The Matrix Eigenvalue Problem: GR and Krylov Subspace Methods, SIAM, 2007
- The QR algorithm revisited, SIAM Rev., 2008
- Fundamentals of Matrix Computations, 3rd Ed., 2010
- Francis's Algorithm, Amer. Math. Monthly, 2011

# My History with this Topic

- Understanding the QR algorithm, SIAM Rev., 1982
- Fundamentals of Matrix Computations, Wiley, 1991
- Some perspectives on the eigenvalue problem, 1993
- QR-like algorithms—an overview of convergence theory and practice, AMS proceedings, 1996
- QR-like algorithms for eigenvalue problems, JCAM, 2000
- The Matrix Eigenvalue Problem: GR and Krylov Subspace Methods, SIAM, 2007
- The QR algorithm revisited, SIAM Rev., 2008
- Fundamentals of Matrix Computations, 3rd Ed., 2010
- Francis's Algorithm, Amer. Math. Monthly, 2011

... but we're still not done!

This is joint work with

- **Jared Aurentz** (Oxford)
- **Thomas Mach** (KU Leuven)
- **Raf Vandebril** (KU Leuven)

# A new look at an old algorithm



# A new look at an old algorithm

Store in QR decomposed form

$$A = QR$$

- $Q$  is unitary,  $R$  is upper triangular

# A new look at an old algorithm

Store in QR decomposed form

$$A = QR$$

- $Q$  is unitary,  $R$  is upper triangular
- looks inefficient!

# A new look at an old algorithm

Store in QR decomposed form


$$A = QR$$

- $Q$  is unitary,  $R$  is upper triangular
- looks inefficient! but it's not!

# A new look at an old algorithm

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

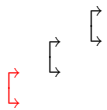
# A new look at an old algorithm


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

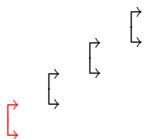
# A new look at an old algorithm

$$\left[ \begin{array}{ccccc} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{array} \right] = \left[ \begin{array}{ccccc} * & * & * & * & * \\ 0 & * & * & * & * \\ & 0 & * & * & * \\ & & * & * & * \\ & & & * & * \end{array} \right]$$

# A new look at an old algorithm

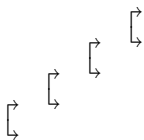

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix} = \begin{bmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ & 0 & * & * & * \\ & & 0 & * & * \\ & & & * & * \end{bmatrix}$$

# A new look at an old algorithm


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



# A new look at an old algorithm



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

# A new look at an old algorithm


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

- **Def:** *Core Transformation*

# A new look at an old algorithm


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

- **Def:** *Core Transformation*
- Now invert the core transformations to move them to the other side.

# A new look at an old algorithm

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix} = \begin{matrix} \rightarrow \\ \leftarrow \\ \rightarrow \\ \leftarrow \\ \rightarrow \\ \leftarrow \\ \rightarrow \\ \leftarrow \end{matrix} \begin{bmatrix} * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \\ & & & & * \end{bmatrix}$$

# A new look at an old algorithm

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix} = \begin{matrix} \left. \begin{matrix} \rightarrow \\ \rightarrow \end{matrix} \right\} & \left. \begin{matrix} \rightarrow \\ \rightarrow \end{matrix} \right\} & \left. \begin{matrix} \rightarrow \\ \rightarrow \end{matrix} \right\} & \left. \begin{matrix} \rightarrow \\ \rightarrow \end{matrix} \right\} \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{matrix} \begin{bmatrix} * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \\ & & & & * \end{bmatrix}$$

$$A = QR$$

# A new look at an old algorithm

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix} = \begin{matrix} \left[ \right] & & & & \\ & \left[ \right] & & & \\ & & \left[ \right] & & \\ & & & \left[ \right] & \\ & & & & \left[ \right] \end{matrix} \begin{bmatrix} * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \\ & & & & * \end{bmatrix}$$

$$A = QR$$

$$Q = \begin{matrix} \left[ \right] & & & & \\ & \left[ \right] & & & \\ & & \left[ \right] & & \\ & & & \left[ \right] & \\ & & & & \left[ \right] \end{matrix}$$

$Q$  requires only  $O(n)$  storage space.

# A new look at an old algorithm

## Manipulating core transformations

# A new look at an old algorithm

## Manipulating core transformations

- Fusion

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



# A new look at an old algorithm

## Manipulating core transformations

- Fusion

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

- Turnover (aka shift through, Givens swap, ...)

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

# A new look at an old algorithm

## Manipulating core transformations

- Fusion

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

- Turnover (aka shift through, Givens swap, ...)

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

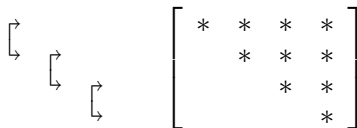
- Passing a core transformation through a triangular matrix (cost  $O(n)$ )

$$\left[ \begin{array}{cccc} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \end{array} \right] \left[ \begin{array}{c} \rightarrow \\ \leftarrow \end{array} \right] \Leftrightarrow \left[ \begin{array}{cccc} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \end{array} \right] \left[ \begin{array}{c} \rightarrow \\ \leftarrow \end{array} \right] \Leftrightarrow \left[ \begin{array}{cccc} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \end{array} \right]$$

(Note: A red '+' sign is present in the bottom row of the second matrix, under the second column.)

# A new look at an old algorithm

## Francis's algorithm on the QR decomposed form (a core chasing algorithm)

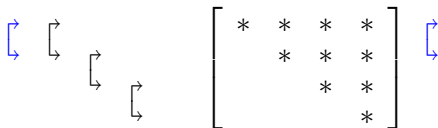


The diagram illustrates the core chasing process in Francis's algorithm. On the left, three horizontal arrows point to the right, with a vertical double-headed arrow bracket positioned above each arrow, indicating the movement of the core. On the right, a matrix is shown with asterisks representing non-zero elements. The matrix is upper triangular, with non-zero elements in the first four rows and columns, forming a core of size 4x4.

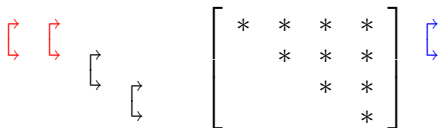
$$\left[ \begin{array}{cccc} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \end{array} \right]$$

# A new look at an old algorithm

## Francis's algorithm on the QR decomposed form (a core chasing algorithm)

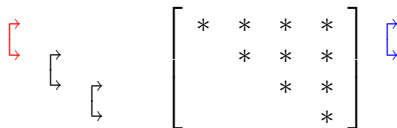


## Francis's algorithm on the QR decomposed form (a core chasing algorithm)

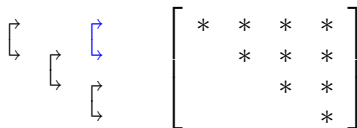


# A new look at an old algorithm

## Francis's algorithm on the QR decomposed form (a core chasing algorithm)



## Francis's algorithm on the QR decomposed form (a core chasing algorithm)





## Francis's algorithm on the QR decomposed form (a core chasing algorithm)



## Francis's algorithm on the QR decomposed form (a core chasing algorithm)



## Francis's algorithm on the QR decomposed form (a core chasing algorithm)



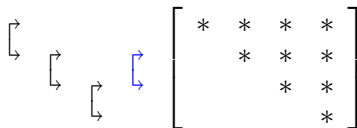
# A new look at an old algorithm

## Francis's algorithm on the QR decomposed form (a core chasing algorithm)

$$\left[ \begin{array}{cccc} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \end{array} \right]$$

The diagram illustrates the core chasing process in Francis's algorithm. It shows a matrix with asterisks representing non-zero elements. A blue double-headed arrow on the right indicates the core being chased.

## Francis's algorithm on the QR decomposed form (a core chasing algorithm)



The diagram illustrates the core-chasing process of Francis's algorithm. It shows a sequence of four horizontal double-headed arrows pointing to the right, with a blue double-headed arrow pointing to the right. These arrows are positioned to the left of a 4x4 matrix. The matrix is shown in a large square bracket and contains asterisks representing non-zero elements. The non-zero elements are located at the following positions: (1,1), (1,2), (1,3), (1,4), (2,2), (2,3), (3,3), (3,4), and (4,4). The blue arrow points to the (2,3) element, indicating the current core being chased.

$$\left[ \begin{array}{cccc} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \end{array} \right]$$

## Francis's algorithm on the QR decomposed form (a core chasing algorithm)

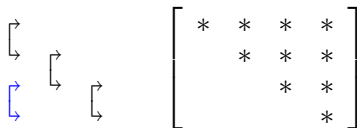
The diagram illustrates Francis's algorithm on a QR decomposed matrix. The matrix is represented as a 4x4 grid of asterisks (\*). The matrix is enclosed in large square brackets. To the left of the matrix, there are three red double-headed vertical brackets and one black double-headed horizontal bracket. The red brackets are positioned between the first and second columns, the second and third columns, and the third and fourth columns. The black bracket is positioned between the first and second rows. This indicates the movement of the core (the 2x2 submatrix) from the top-left towards the bottom-right of the matrix.

$$\left[ \begin{array}{cccc} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \end{array} \right]$$

## Francis's algorithm on the QR decomposed form (a core chasing algorithm)

$$\left[ \begin{array}{cccc} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \end{array} \right]$$

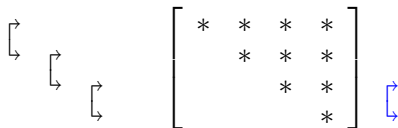
## Francis's algorithm on the QR decomposed form (a core chasing algorithm)





# A new look at an old algorithm

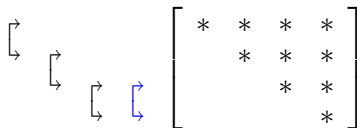
## Francis's algorithm on the QR decomposed form (a core chasing algorithm)



The diagram illustrates Francis's algorithm on the QR decomposed form. It shows a sequence of three horizontal double-headed arrows pointing to the right, indicating the movement of the core. To the right of these arrows is a matrix structure represented by a large square bracket containing asterisks. The asterisks are arranged in a staircase pattern: the first row has four asterisks, the second row has three, the third row has two, and the fourth row has one. To the right of the matrix is a vertical double-headed arrow pointing up and down, indicating the movement of the core. The entire diagram is set against a white background.

$$\begin{array}{c} \longleftrightarrow \\ \longleftrightarrow \\ \longleftrightarrow \end{array} \left[ \begin{array}{cccc} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \end{array} \right] \begin{array}{c} \updownarrow \end{array}$$

## Francis's algorithm on the QR decomposed form (a core chasing algorithm)



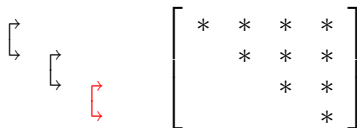
The diagram illustrates the core chasing process in Francis's algorithm. It shows a sequence of four horizontal double-headed arrows of decreasing length, positioned to the left of a matrix. The arrows are white, black, black, and blue, respectively, indicating the progression of the algorithm. The matrix is a 4x4 upper triangular matrix with asterisks representing non-zero elements. The non-zero elements are located at the following positions: (1,1), (1,2), (1,3), (1,4), (2,2), (2,3), (2,4), (3,3), (3,4), and (4,4).

$$\left[ \begin{array}{cccc} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \end{array} \right]$$

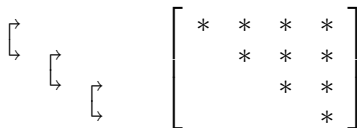
## Francis's algorithm on the QR decomposed form (a core chasing algorithm)

$$\left[ \begin{array}{cccc} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \end{array} \right]$$

## Francis's algorithm on the QR decomposed form (a core chasing algorithm)

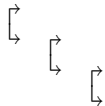


## Francis's algorithm on the QR decomposed form (a core chasing algorithm)


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

# A new look at an old algorithm

## Francis's algorithm on the QR decomposed form (a core chasing algorithm)


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

Done!

## 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)

## Are there any advantages?

- unitary case
- companion case (unitary-plus-rank-one)
- general case: efficient cache use

# Unitary Case

# Unitary Case

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

# Unitary Case

$$A = QR = \begin{matrix} \left[ \right. \\ \left[ \right. \\ \left[ \right. \end{matrix}$$



# Unitary Case

$$A = QR = \begin{matrix} \left[ \right. & & & \\ & \left[ \right. & & \\ & & \left[ \right. & \\ & & & \left[ \right. \end{matrix}$$

$$A = QR = \begin{bmatrix} \rightarrow & & \\ & \rightarrow & \\ & & \rightarrow \end{bmatrix}$$

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

$$A = QR = \begin{bmatrix} \rightarrow & & \\ & \rightarrow & \\ & & \rightarrow \end{bmatrix}$$

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

# Unitary Case

$$A = QR = \begin{bmatrix} \rightarrow \\ \rightarrow \\ \rightarrow \end{bmatrix}$$

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

# Unitary Case

$$A = QR = \begin{bmatrix} \rightarrow \\ \rightarrow \\ \rightarrow \end{bmatrix}$$

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

# Unitary Case

$$A = QR = \begin{matrix} \left[ \right. & & \\ & \left[ \right. & \\ & & \left[ \right. \end{matrix}$$

- 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

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

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

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

- ... 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
  - data-sparse representation + 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
  - data-sparse representation + Francis's algorithm
  - several methods proposed

# Some of the Competitors

- Chandrasekaran, Gu, Xia, Zhu (2007)
- Bini, Boito, Eidelman, Gemignani, Gohberg (2010)
- Boito, Eidelman, Gemignani, Gohberg (2012)

# Some of the Competitors

- Chandrasekaran, Gu, Xia, Zhu (2007)
- Bini, Boito, Eidelman, Gemignani, Gohberg (2010)
- Boito, Eidelman, Gemignani, Gohberg (2012)
  
- Fortran codes available



# Some of the Competitors

- Chandrasekaran, Gu, Xia, Zhu (2007)
  - Bini, Boito, Eidelman, Gemignani, Gohberg (2010)
  - Boito, Eidelman, Gemignani, Gohberg (2012)
- 
- Fortran codes available
  - evidence of backward stability

# Some of the Competitors

- Chandrasekaran, Gu, Xia, Zhu (2007)
  - Bini, Boito, Eidelman, Gemignani, Gohberg (2010)
  - Boito, Eidelman, Gemignani, Gohberg (2012)
- 
- Fortran codes available
  - evidence of backward stability
  - quasiseparable generator representation

# Some of the Competitors

- Chandrasekaran, Gu, Xia, Zhu (2007)
  - Bini, Boito, Eidelman, Gemignani, Gohberg (2010)
  - Boito, Eidelman, Gemignani, Gohberg (2012)
- 
- Fortran codes available
  - evidence of backward stability
  - quasiseparable generator representation
  - We will do something else.

# Some of the Competitors

- Chandrasekaran, Gu, Xia, Zhu (2007)
  - Bini, Boito, Eidelman, Gemignani, Gohberg (2010)
  - Boito, Eidelman, Gemignani, Gohberg (2012)
- 
- Fortran codes available
  - evidence of backward stability
  - quasiseparable generator representation
  - We will do something else.
  - Our method is faster,

# Some of the Competitors

- Chandrasekaran, Gu, Xia, Zhu (2007)
- Bini, Boito, Eidelman, Gemignani, Gohberg (2010)
- Boito, Eidelman, Gemignani, Gohberg (2012)
  
- Fortran codes available
- evidence of backward stability
- quasiseparable generator representation
- We will do something else.
- Our method is faster, and **we can prove backward stability.**

# Structure

- Companion matrix is unitary-plus-rank-one

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

- We exploit this structure.

... but we store the QR decomposed form

$$A = QR$$



... but we store the QR decomposed form

$$A = QR$$

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

... but we store the QR decomposed form

$$A = QR$$

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

$$= \begin{matrix} \left[ \right] & & & \\ \left[ \right] & \left[ \right] & & \\ & \ddots & & \\ & & \left[ \right] & \end{matrix} \begin{bmatrix} 1 & 0 & \cdots & -a_1 \\ & 1 & & -a_2 \\ & & \ddots & \vdots \\ & & & -a_0 \end{bmatrix}$$

# Structure

- How do we store  $R$  compactly?

# Structure

- How do we store  $R$  compactly?
- $R$  is unitary-plus-rank one.

# Structure

- How do we store  $R$  compactly?
- $R$  is unitary-plus-rank one.
- Adjoin a row and column for wiggle room. (not obvious)

# Structure

- How do we store  $R$  compactly?
- $R$  is unitary-plus-rank one.
- Adjoin a row and column for wiggle room. (not obvious)

$$\hat{R} = \left[ \begin{array}{ccc|c} 1 & & -a_1 & 0 \\ & \ddots & \vdots & \vdots \\ & & 1 & -a_{n-1} \\ \hline & & & -a_0 \\ & & & 0 \end{array} \right]$$

# Structure

- How do we store  $R$  compactly?
- $R$  is unitary-plus-rank one.
- Adjoin a row and column for wiggle room. (not obvious)

$$\hat{R} = \left[ \begin{array}{ccc|c} 1 & & -a_1 & 0 \\ & \ddots & \vdots & \vdots \\ & & 1 & -a_{n-1} & 0 \\ & & & -a_0 & 1 \\ \hline & & & 0 & 0 \end{array} \right]$$
$$= \left[ \begin{array}{ccc|c} 1 & & 0 & 0 \\ & \ddots & \vdots & \vdots \\ & & 1 & 0 & 0 \\ & & & 0 & 1 \\ \hline & & & 1 & 0 \end{array} \right] + \left[ \begin{array}{ccc|c} 0 & & -a_1 & 0 \\ & \ddots & \vdots & \vdots \\ & & 0 & -a_{n-1} & 0 \\ \hline & & & -a_0 & 0 \\ & & & -1 & 0 \end{array} \right]$$

# Representation of $R$

- $R = P^T \hat{R} P$



# Representation of $R$

- $R = P^T \hat{R} P$
- $\hat{R} = U + xy^T$ , where

# Representation of $R$

- $R = P^T \hat{R} P$
- $\hat{R} = U + xy^T$ , where

$$xy^T = \left[ \begin{array}{c} -a_1 \\ \vdots \\ -a_{n-1} \\ -a_0 \\ \hline -1 \end{array} \right] \left[ \begin{array}{cccc|c} 0 & \cdots & 0 & 1 & 0 \end{array} \right]$$

# Representation of $R$

- $R = P^T \hat{R} P$
- $\hat{R} = U + xy^T$ , where

$$xy^T = \left[ \begin{array}{c} -a_1 \\ \vdots \\ -a_{n-1} \\ -a_0 \\ \hline -1 \end{array} \right] \left[ \begin{array}{cccc|c} 0 & \cdots & 0 & 1 & 0 \end{array} \right]$$

- Next step: Roll up  $x$ .

# Representation of $R$

$$\begin{bmatrix} x \\ x \\ x \\ x \end{bmatrix} = \begin{bmatrix} x \\ x \\ x \\ x \end{bmatrix}$$

# Representation of $R$

$$\left[ \begin{array}{c} x \\ x \\ x \\ x \end{array} \right] = \left[ \begin{array}{c} x \\ x \\ x \\ 0 \end{array} \right]$$

# Representation of $R$

$$\begin{array}{c} \left[ \begin{array}{c} \rightarrow \\ \rightarrow \end{array} \right] \left[ \begin{array}{c} x \\ x \\ x \\ x \end{array} \right] = \left[ \begin{array}{c} x \\ x \\ 0 \\ 0 \end{array} \right] \end{array}$$

# Representation of $R$

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

# Representation of $R$

$$\begin{array}{c} \left. \begin{array}{c} \rightarrow \\ \rightarrow \end{array} \right\} \\ \left. \begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \end{array} \right\} \\ \left. \begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{array} \right\} \end{array} \begin{bmatrix} x \\ x \\ x \\ x \end{bmatrix} = \begin{bmatrix} x \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$C_1 \cdots C_{n-1} C_n x = \alpha e_1 \quad (\text{w.l.g. } \alpha = 1)$$



# Representation of $R$

- $C_1 \cdots C_{n-1} C_n x = e_1$

# Representation of $R$

- $C_1 \cdots C_{n-1} C_n x = e_1$
- $Cx = e_1$

# Representation of $R$

- $C_1 \cdots C_{n-1} C_n x = e_1$
- $Cx = e_1$
- $C^* e_1 = x$

# Representation of $R$

- $C_1 \cdots C_{n-1} C_n x = e_1$
- $Cx = e_1$
- $C^* e_1 = x$
- $\hat{R} = U + xy^T = U + C^* e_1 y^T = C^*(CU + e_1 y^T)$

# Representation of $R$

- $C_1 \cdots C_{n-1} C_n x = e_1$
- $Cx = e_1$
- $C^* e_1 = x$
- $\hat{R} = U + xy^T = U + C^* e_1 y^T = C^*(CU + e_1 y^T)$
- $\hat{R} = C^*(B + e_1 y^T)$

# Representation of $R$

- $C_1 \cdots C_{n-1} C_n x = e_1$
- $Cx = e_1$
- $C^* e_1 = x$
- $\hat{R} = U + xy^T = U + C^* e_1 y^T = C^*(CU + e_1 y^T)$
- $\hat{R} = C^*(B + e_1 y^T)$
- $B$  is upper Hessenberg (and unitary)

# Representation of $R$

- $C_1 \cdots C_{n-1} C_n x = e_1$
- $Cx = e_1$
- $C^* e_1 = x$
- $\hat{R} = U + xy^T = U + C^* e_1 y^T = C^*(CU + e_1 y^T)$
- $\hat{R} = C^*(B + e_1 y^T)$
- $B$  is upper Hessenberg (and unitary) so  $B = B_1 \cdots B_n$ .

# Representation of $R$

- $C_1 \cdots C_{n-1} C_n x = e_1$
- $Cx = e_1$
- $C^* e_1 = x$
- $\hat{R} = U + xy^T = U + C^* e_1 y^T = C^*(CU + e_1 y^T)$
- $\hat{R} = C^*(B + e_1 y^T)$
- $B$  is upper Hessenberg (and unitary) so  $B = B_1 \cdots B_n$ .
- $R = P^T C^*(B + e_1 y^T)P = P^T C_n^* \cdots C_1^*(B_1 \cdots B_n + e_1 y^T)P$



# Representation of $R$

- $C_1 \cdots C_{n-1} C_n x = e_1$
- $Cx = e_1$
- $C^* e_1 = x$
- $\hat{R} = U + xy^T = U + C^* e_1 y^T = C^*(CU + e_1 y^T)$
- $\hat{R} = C^*(B + e_1 y^T)$
- $B$  is upper Hessenberg (and unitary) so  $B = B_1 \cdots B_n$ .
- $R = P^T C^*(B + e_1 y^T)P = P^T C_n^* \cdots C_1^*(B_1 \cdots B_n + e_1 y^T)P$
- $O(n)$  storage
- Bonus: Redundancy! No need to keep track of  $y$ .

# Representation of $R$

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

# Representation of $R$

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

$$= \begin{matrix} & & & \begin{matrix} \left[ \begin{matrix} \rightarrow \\ \leftarrow \end{matrix} \right] \\ \rightarrow \\ \leftarrow \end{matrix} \\ & & \begin{matrix} \left[ \begin{matrix} \rightarrow \\ \leftarrow \end{matrix} \right] \\ \rightarrow \\ \leftarrow \end{matrix} \\ & \begin{matrix} \left[ \begin{matrix} \rightarrow \\ \leftarrow \end{matrix} \right] \\ \rightarrow \\ \leftarrow \end{matrix} \\ \begin{matrix} \left[ \begin{matrix} \rightarrow \\ \leftarrow \end{matrix} \right] \\ \rightarrow \\ \leftarrow \end{matrix} \end{matrix} \left[ \begin{matrix} \rightarrow \\ \leftarrow \end{matrix} \right] \left[ \begin{matrix} \rightarrow \\ \leftarrow \end{matrix} \right] \left[ \begin{matrix} \rightarrow \\ \leftarrow \end{matrix} \right] \left[ \begin{matrix} \rightarrow \\ \leftarrow \end{matrix} \right] + \cdots \end{matrix}$$

**Altogether we have**

$$A = QR$$

$$= \begin{matrix} \begin{matrix} \left[ \begin{matrix} \rightarrow \\ \rightarrow \\ \rightarrow \end{matrix} \right] \\ \left[ \begin{matrix} \rightarrow \\ \rightarrow \end{matrix} \right] \\ \left[ \begin{matrix} \rightarrow \end{matrix} \right] \end{matrix} & \begin{matrix} \left[ \begin{matrix} \rightarrow \\ \rightarrow \\ \rightarrow \end{matrix} \right] \\ \left[ \begin{matrix} \rightarrow \\ \rightarrow \end{matrix} \right] \\ \left[ \begin{matrix} \rightarrow \end{matrix} \right] \end{matrix} & \left[ \begin{matrix} \left[ \begin{matrix} \rightarrow \\ \rightarrow \\ \rightarrow \end{matrix} \right] \\ \left[ \begin{matrix} \rightarrow \\ \rightarrow \end{matrix} \right] \\ \left[ \begin{matrix} \rightarrow \end{matrix} \right] \\ + \dots \end{matrix} \right] \end{matrix}$$

# Francis Iterations

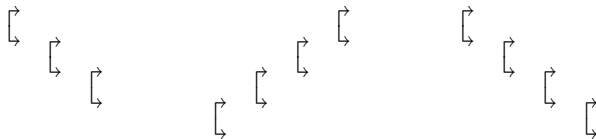
- We have complex single-shift code ...
- real double-shift code.

# Francis Iterations

- We have complex single-shift code . . .
- real double-shift code.
- We describe single-shift case for simplicity.

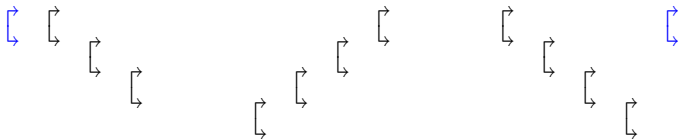


# The Core Chase

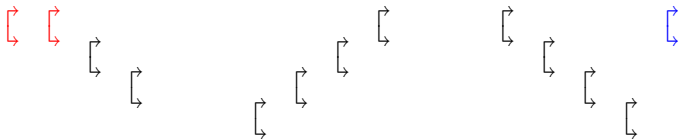




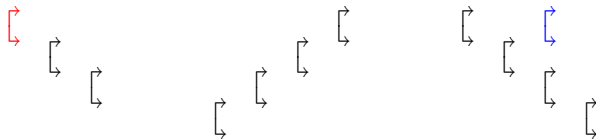
# The Core Chase



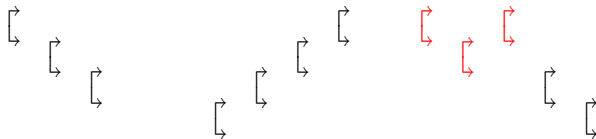
# The Core Chase



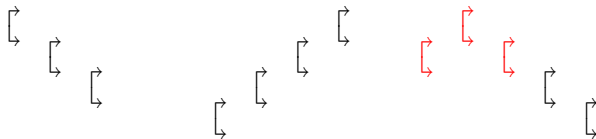
# The Core Chase



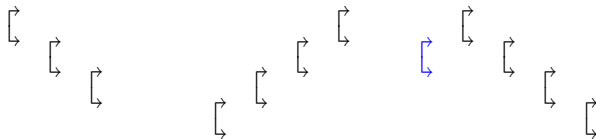
# The Core Chase



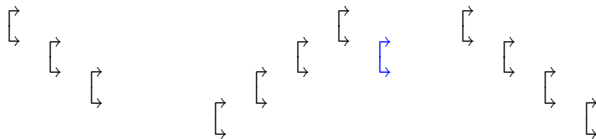
# The Core Chase



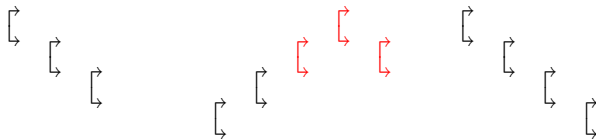
# The Core Chase



# The Core Chase

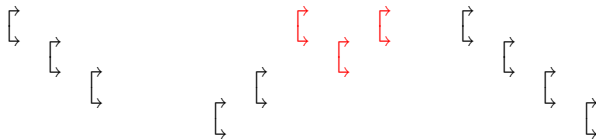


# The Core Chase

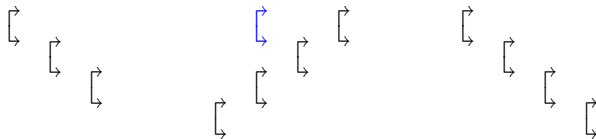




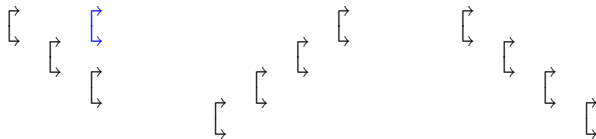
# The Core Chase



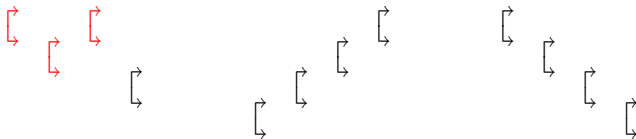
# The Core Chase



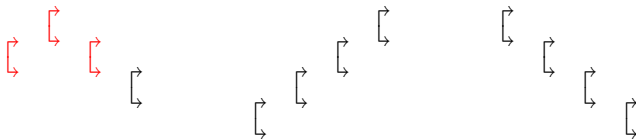
# The Core Chase



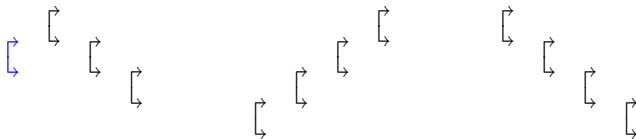
# The Core Chase



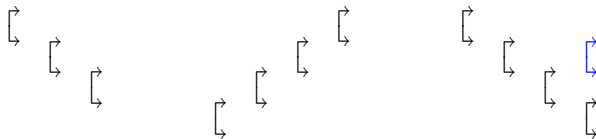
# The Core Chase



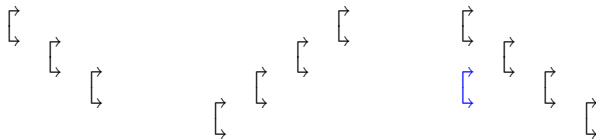
# The Core Chase



# The Core Chase

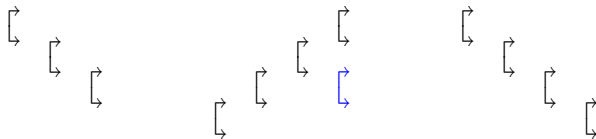


# The Core Chase

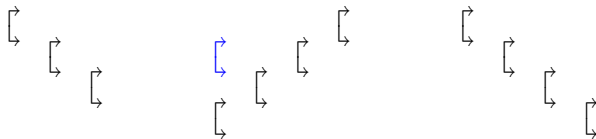




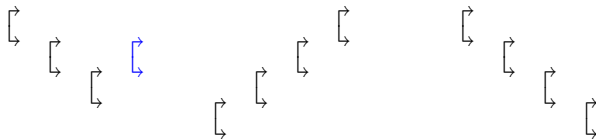
# The Core Chase



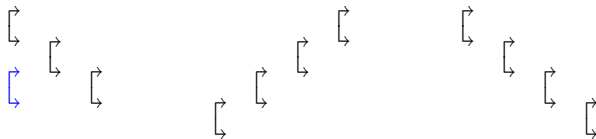
# The Core Chase



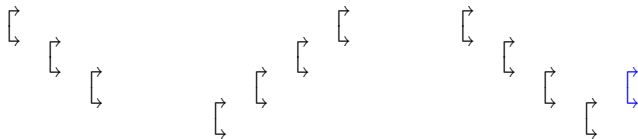
# The Core Chase



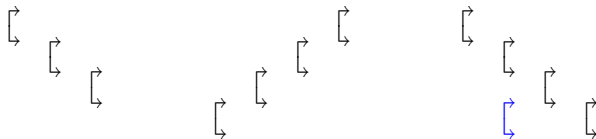
# The Core Chase



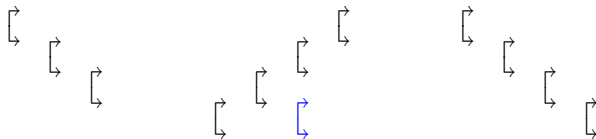
# The Core Chase



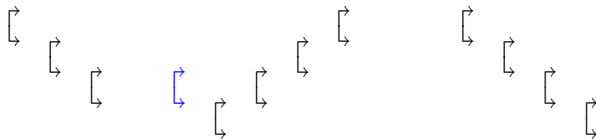
# The Core Chase



# The Core Chase

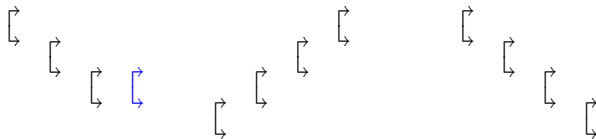


# The Core Chase

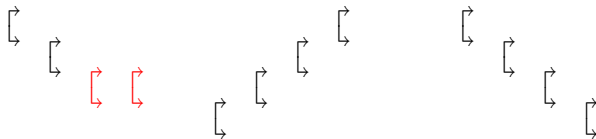




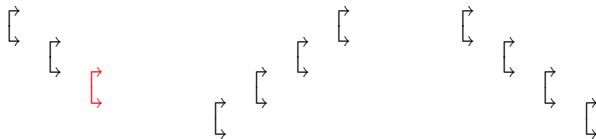
# The Core Chase



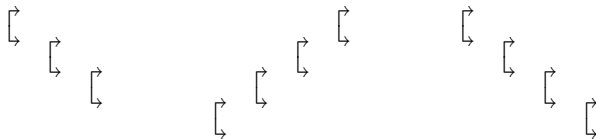
# The Core Chase



# The Core Chase



# The Core Chase

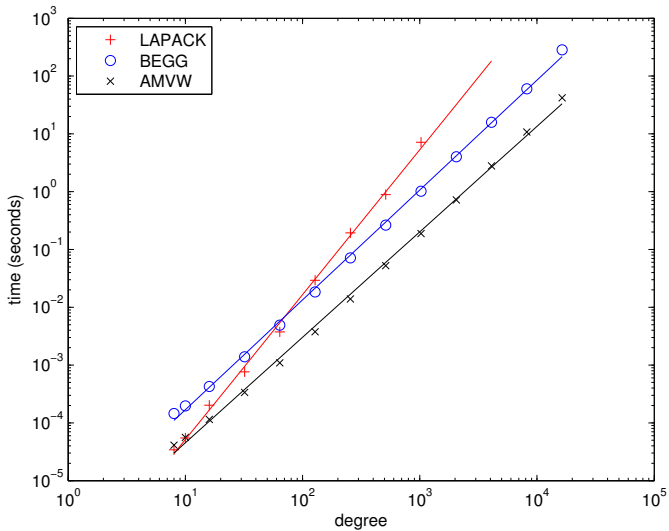


- iteration complete!

- iteration complete!
- Cost:  $3n$  turnovers/iteration, so  $O(n)$  flops/iteration

- iteration complete!
- Cost:  $3n$  turnovers/iteration, so  $O(n)$  flops/iteration
- Double-shift iteration is similar.
- (Chase two core transformations instead of one.)

# Performance





**At degree 1000**

method	time
LAPACK	7.2
BEGG	1.2
AMVW	0.2

# See our paper for . . .

- Paper in SIAM J. Matrix Anal. Appl. has

# See our paper for . . .

- Paper in SIAM J. Matrix Anal. Appl. has
- . . . more timings,

# See our paper for . . .

- Paper in SIAM J. Matrix Anal. Appl. has
- . . . more timings,
- . . . accuracy comparisons,

# See our paper for . . .

- Paper in SIAM J. Matrix Anal. Appl. has
- . . . more timings,
- . . . accuracy comparisons,
- . . . proof of backward stability.

# Summary

- We took a new look at Francis's algorithm

# Summary

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



# Summary

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

# Summary

- We took a new look at Francis's algorithm
- considered QR decomposed form
- We demonstrated some advantages.
  - unitary case

# Summary

- We took a new look at Francis's algorithm
- considered QR decomposed form
- We demonstrated some advantages.
  - unitary case
  - unitary-plus-rank-one case (companion)

# Summary

- We took a new look at Francis's algorithm
- considered QR decomposed form
- We demonstrated some advantages.
  - unitary case
  - unitary-plus-rank-one case (companion)
  - efficient cache use (not demonstrated today)

- We took a new look at Francis's algorithm
- considered QR decomposed form
- We demonstrated some advantages.
  - unitary case
  - unitary-plus-rank-one case (companion)
  - efficient cache use (not demonstrated today)

Thank you for your attention.