Fast stable computation of the eigenvalues of unitary-plus-rank-one matrices, including companion matrices

David S. Watkins

Department of Mathematics
Washington State University

AMS Sectional Meeting, April 22, 2017

Who Cares?

- unitary-plus-rank-one?

Who Cares?

- unitary-plus-rank-one?
- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n$

Who Cares?

- unitary-plus-rank-one?
- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n$
- Find the zeros.

Who Cares?

- unitary-plus-rank-one?
- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n$
- Find the zeros.
- Another first question ...

Who Cares?

- unitary-plus-rank-one?
- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n$
- Find the zeros.
- Another first question ...
- Other bases?

Who Cares?

- unitary-plus-rank-one?
- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n$
- Find the zeros.
- Another first question ...
- Other bases? $\quad p(x) = \sum_{j=0}^{n} c_j T_j(x)$

Companion Matrix

- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n$

Companion Matrix

- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n$
- companion matrix

Companion Matrix

- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n$
- companion matrix

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

Companion Matrix

- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n$
- 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 Matrix

- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n$
- 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

Companion Matrix

- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n$
- 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
- upper Hessenberg

Companion Matrix

- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n$
- 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
- upper Hessenberg
- Francis's (implicitly-shifted QR) algorithm

Companion Matrix

- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n$
- 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
- upper Hessenberg
- Francis's (implicitly-shifted QR) algorithm
- structure not fully exploited

Unitary-plus-rank-one Structure

- Companion matrix is unitary-plus-rank-one:

Unitary-plus-rank-one Structure

- Companion matrix is unitary-plus-rank-one:

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

Unitary-plus-rank-one 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 to get a faster algorithm.

Unitary-plus-rank-one 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 to get a faster algorithm.
- Francis's algorithm preserves this structure.

Unitary-plus-rank-one 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 to get a faster algorithm.
- Francis's algorithm preserves this structure.
- Who's "we"?

Our International Research Group

This is joint work with

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

Our International Research Group

This is joint work with

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

as well as

- **Leonardo Robol** (Pisa $\rightarrow$ KU Leuven $\rightarrow$ Pisa)

Companion Pencil, a variant

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$

Companion Pencil, a variant

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$
- Divide by a_n,

Companion Pencil, a variant

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$
- Divide by a_n, or ...

Companion Pencil, a variant

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$
- Divide by a_n, or ...
- companion pencil:

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

Companion Pencil, a variant

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$
- Divide by a_n, or ...
- companion pencil:

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

- We can handle this too (for a price),

Companion Pencil, a variant

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$
- Divide by a_n, or ...
- companion pencil:

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

- We can handle this too (for a price),
- ...and matrix polynomial problems,

Companion Pencil, a variant

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$
- Divide by a_n, or ...
- companion pencil:

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

- We can handle this too (for a price),
- ... and matrix polynomial problems,
- ... but we will focus on the companion *matrix* problem today.

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
- unitary-plus-rank-one structure exploited

Some of the Competitors

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

- Fortran codes available
- unitary-plus-rank-one structure exploited
- 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
- unitary-plus-rank-one structure exploited
- 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
- unitary-plus-rank-one structure exploited
- evidence of backward stability
- quasiseparable generator representation
- We 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
- unitary-plus-rank-one structure exploited
- evidence of backward stability
- quasiseparable generator representation
- We 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
- unitary-plus-rank-one structure exploited
- evidence of backward stability
- quasiseparable generator representation
- We do something else.
- Our method is faster, and we can prove backward stability.

Storage Scheme, Part I

Storage Scheme, Part I

Store Hessenberg matrix in QR decomposed form

$$A = QR$$

- Q is unitary, R is upper triangular

Storage Scheme, Part I

Store Hessenberg matrix in QR decomposed form

$$A = QR$$

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

Storage Scheme, Part I

Store Hessenberg matrix in QR decomposed form

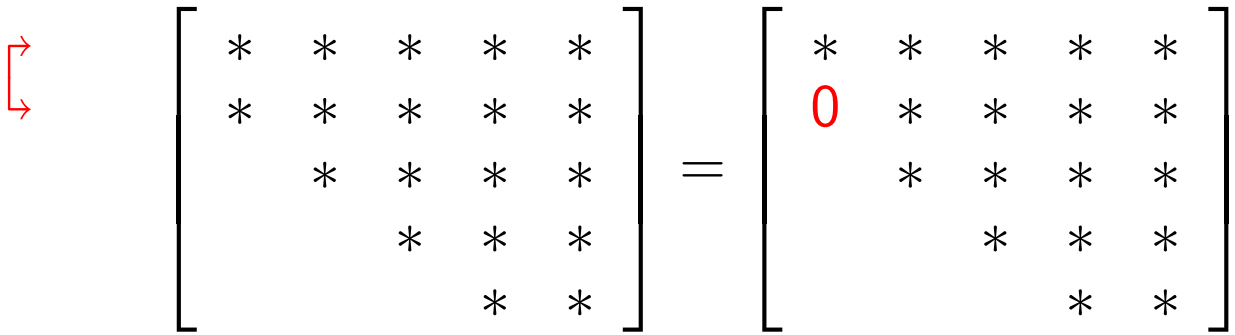
$$A = QR$$

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

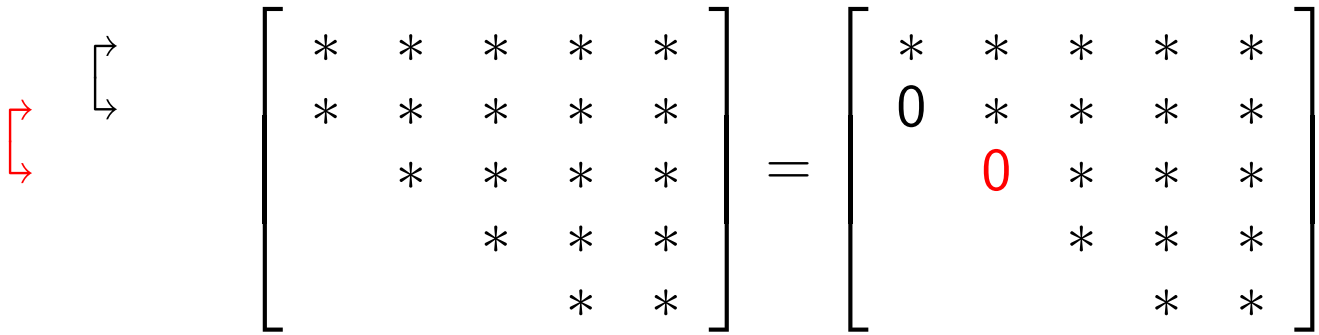
Storage Scheme, Part I

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

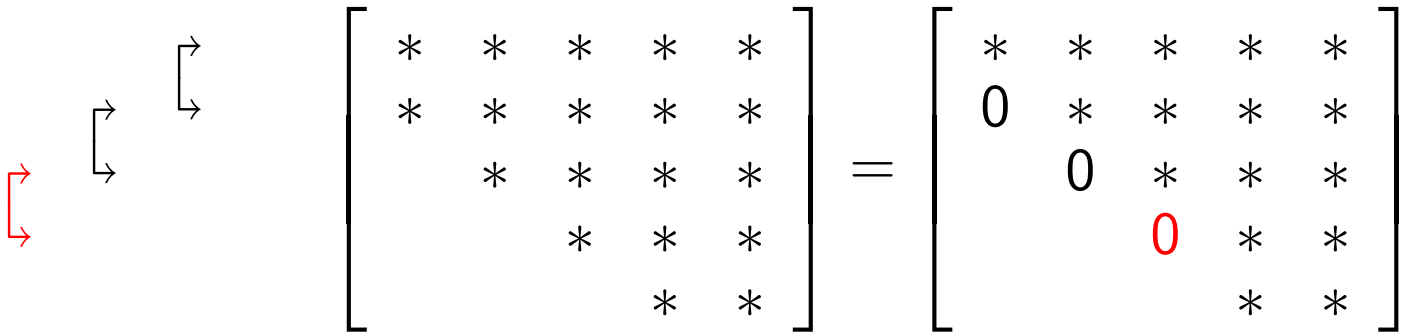
Storage Scheme, Part I



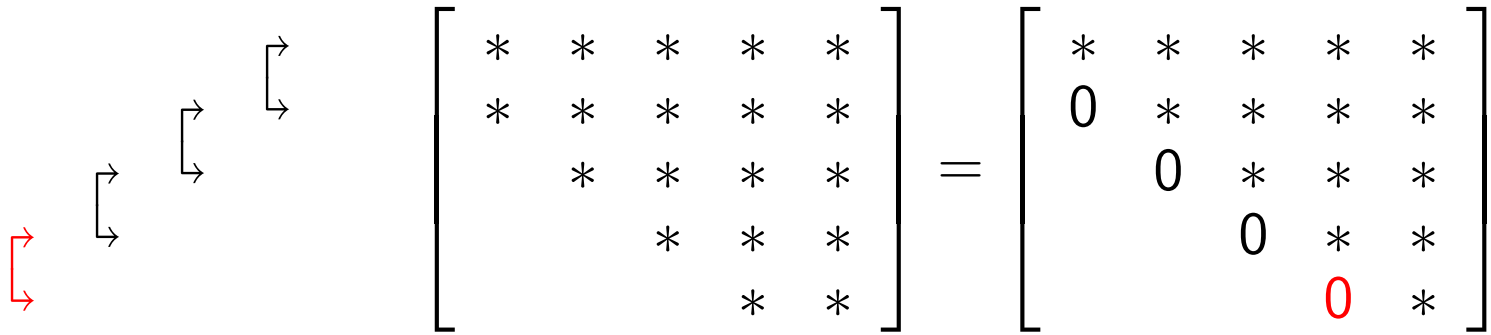
Storage Scheme, Part I



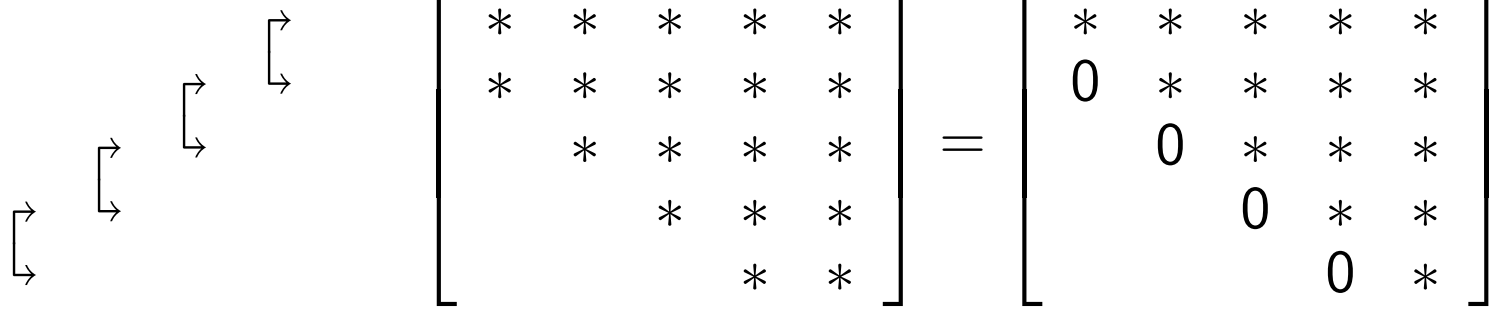
Storage Scheme, Part I



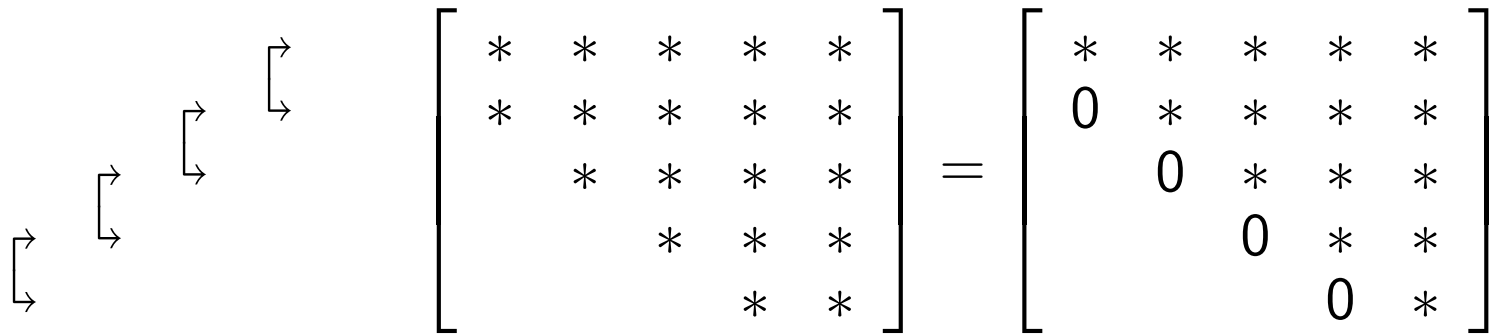
Storage Scheme, Part I



Storage Scheme, Part I

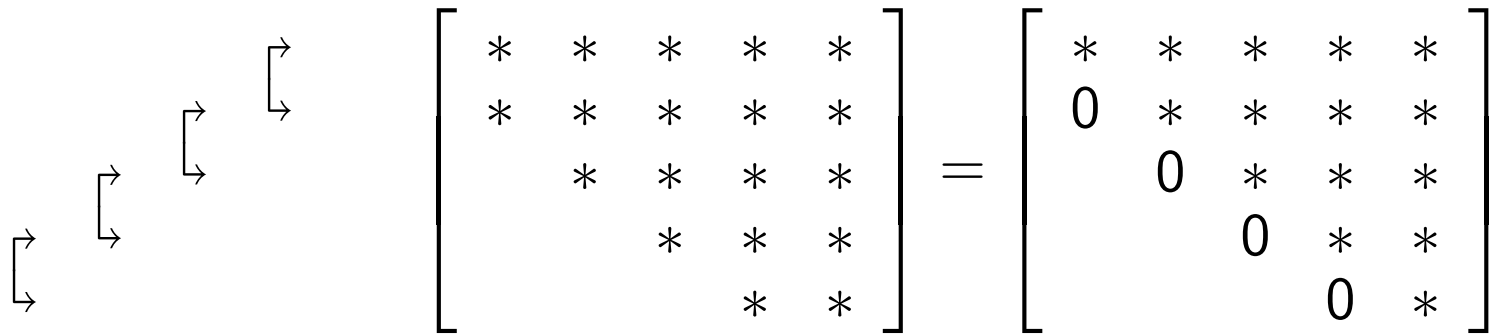


Storage Scheme, Part I



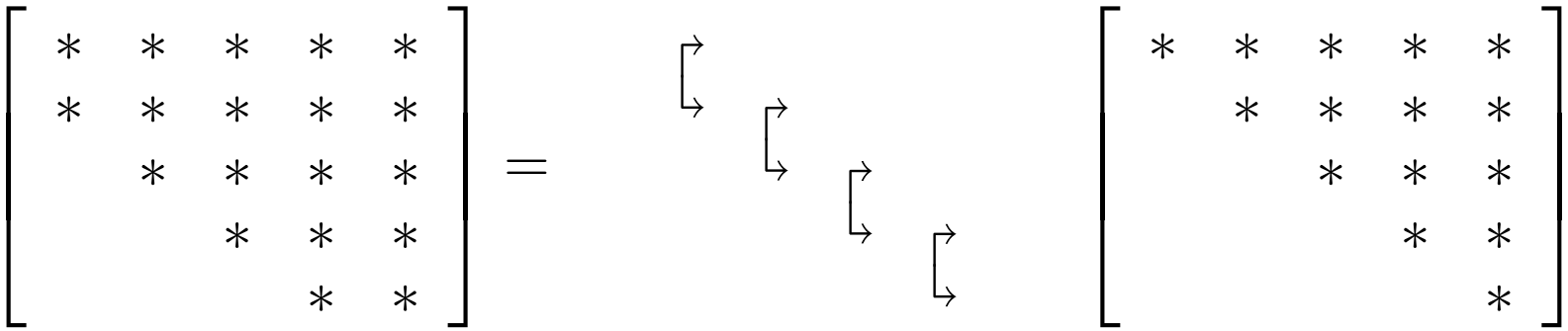
- **Def:** *Core Transformation*

Storage Scheme, Part I

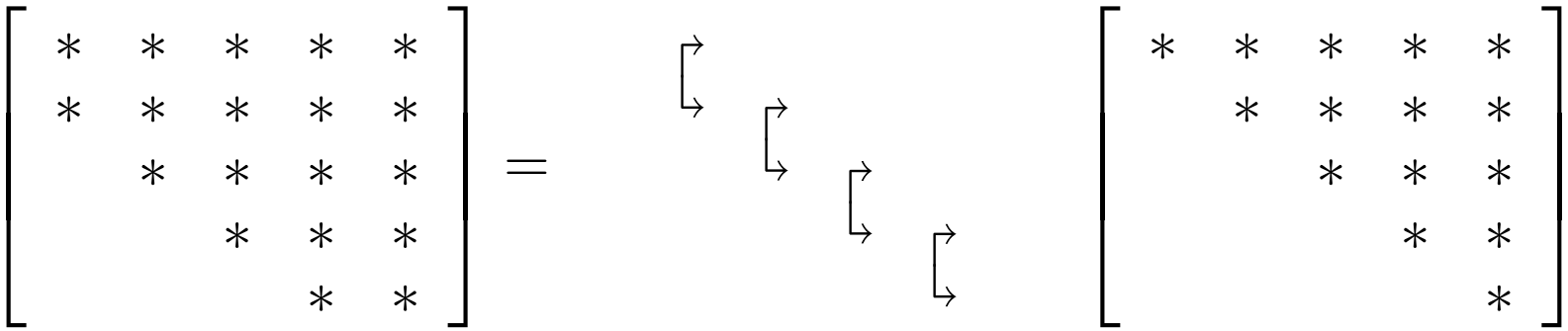


- **Def:** *Core Transformation*
- Now invert the core transformations to move them to the other side.

Storage Scheme, Part I

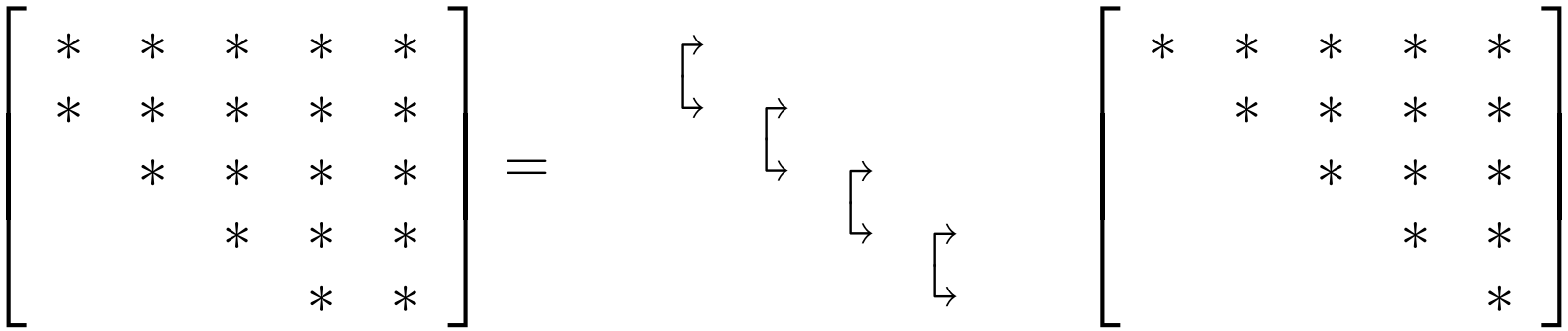


Storage Scheme, Part I

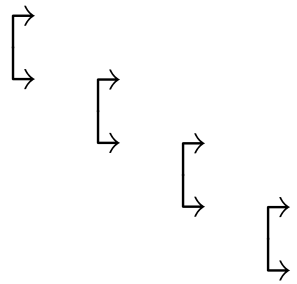


$$A = QR$$

Storage Scheme, Part I



$$A = QR$$

$$Q =$$ 

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

Storage Scheme, Part II

- Now, how do we store R?

Storage Scheme, Part II

- Now, how do we store R?
- R is also unitary-plus-rank-one:

$$A = QR$$

Storage Scheme, Part II

- Now, how do we store R?
- R is also unitary-plus-rank-one:

$$A = QR$$

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

Storage Scheme, Part II

- Now, how do we store R?
- R is also unitary-plus-rank-one:

$$A = QR$$

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

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

Storage Scheme, Part II

- Adjoin a row and column for wiggle room. (not obvious)

Storage Scheme, Part II

- Adjoin a row and column for wiggle room. (not obvious)

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

Storage Scheme, Part II

- Adjoin a row and column for wiggle room. (not obvious)

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

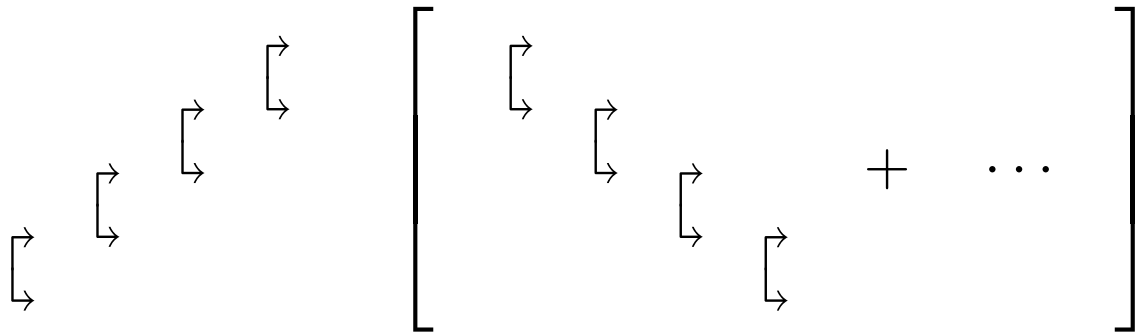
$$= \left[\begin{array}{cccc|c} 1 & & & 0 & 0 \\ & \ddots & & \vdots & \vdots \\ & & 1 & 0 & 0 \\ & & & 0 & 1 \\ \hline & & & 1 & 0 \end{array}\right] + \left[\begin{array}{cccc|c} 0 & & & -a_1 & 0 \\ & \ddots & & \vdots & \vdots \\ & & 0 & -a_{n-1} & 0 \\ & & & -a_0 & 0 \\ \hline & & & -1 & 0 \end{array}\right]$$

Storage Scheme, Part II

- Leaving out a few steps,

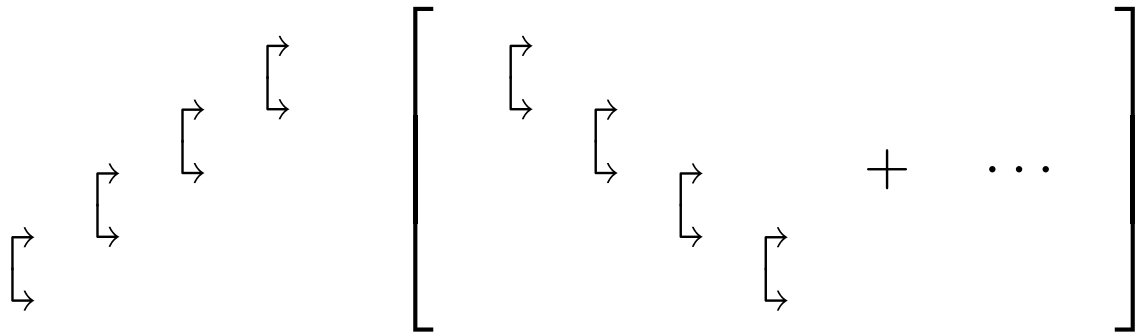
Storage Scheme, Part II

- Leaving out a few steps, we get

$\hat{R} =$ 

Storage Scheme, Part II

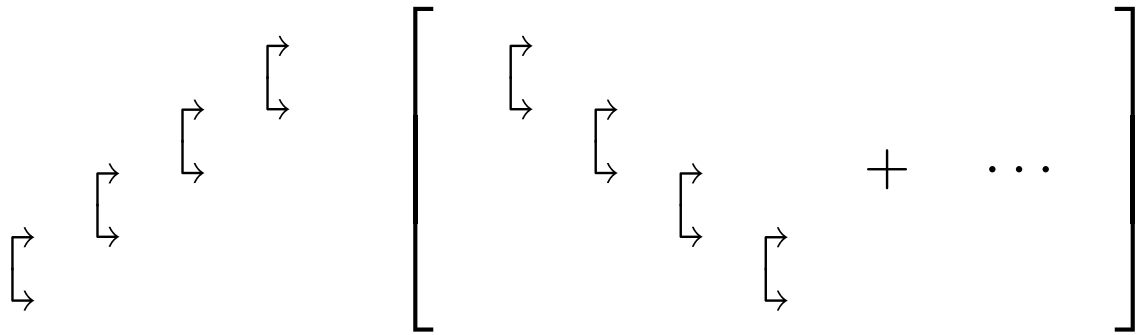
- Leaving out a few steps, we get

$\hat{R} =$ 

- Bonus: Redundant information (Read our paper.)

Storage Scheme, Part II

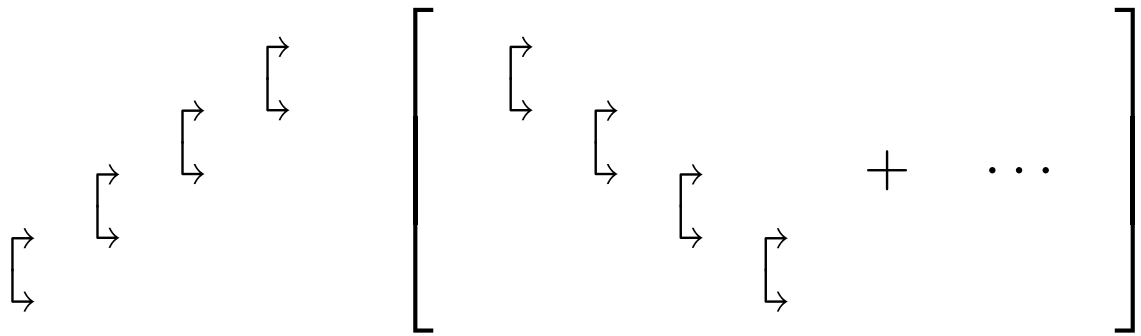
- Leaving out a few steps, we get

$\hat{R} =$ 

- Bonus: Redundant information (Read our paper.)
- We can ignore the rank-one part!

Storage Scheme, Part II

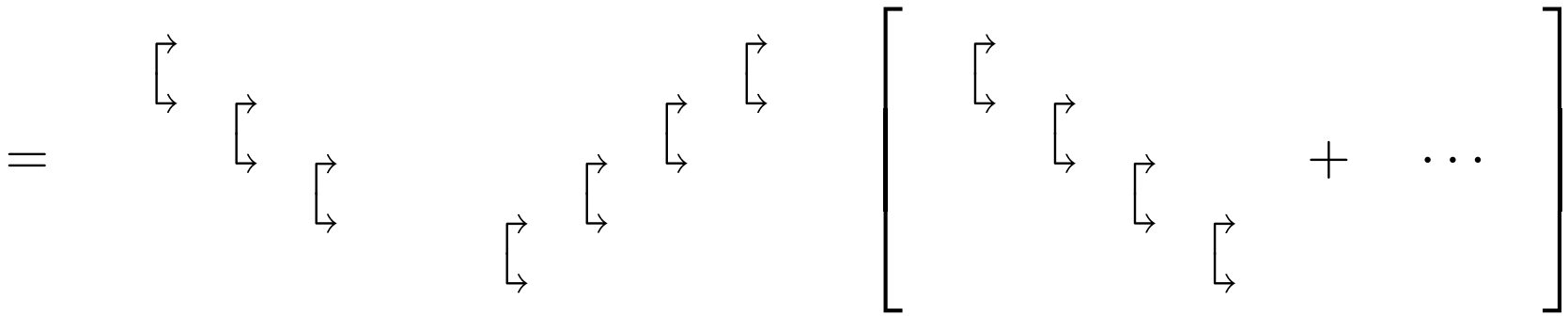
- Leaving out a few steps, we get

$\hat{R} =$ 

- Bonus: Redundant information (Read our paper.)
- We can ignore the rank-one part!
- Storage is $O(n)$.

Altogether we have

$A = QR$



- A is stored entirely in terms of core transformations.

Working with Core Transformations

Working with Core Transformations

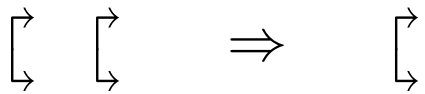
- We want to perform iterations of Francis's algorithm on this Structure.

Working with Core Transformations

- We want to perform iterations of Francis's algorithm on this Structure.
- Two important operations:

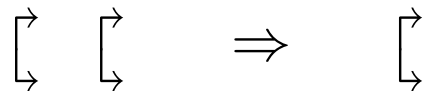
Working with Core Transformations

- We want to perform iterations of Francis's algorithm on this Structure.
- Two important operations:
- Fusion

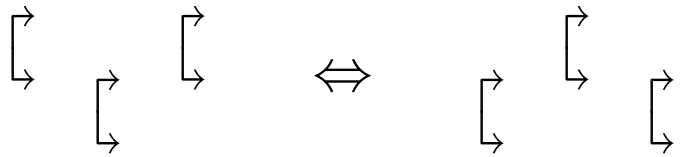


Working with Core Transformations

- We want to perform iterations of Francis's algorithm on this Structure.
- Two important operations:
- Fusion

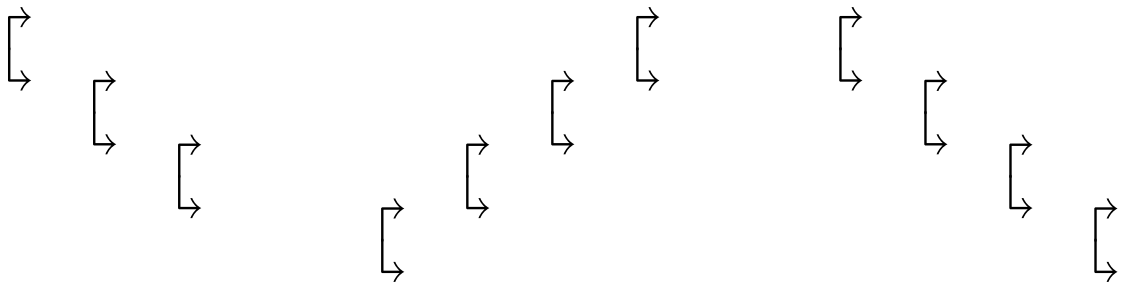


- Turnover (aka shift through, Givens swap, . . .)

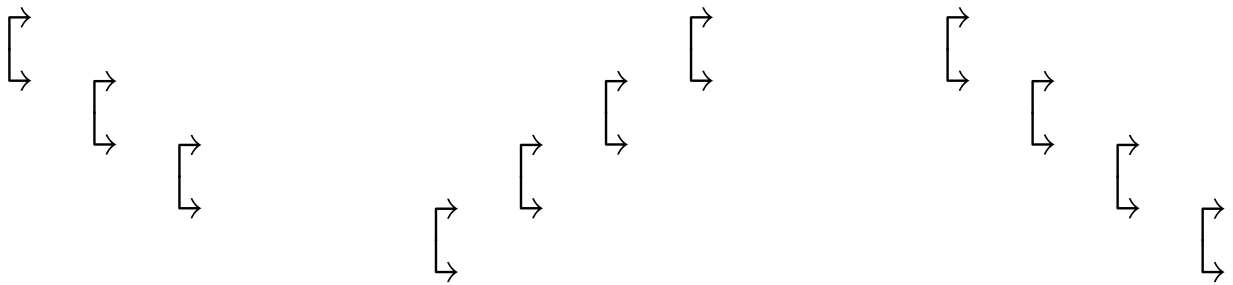


Francis Iteration (the core chase)

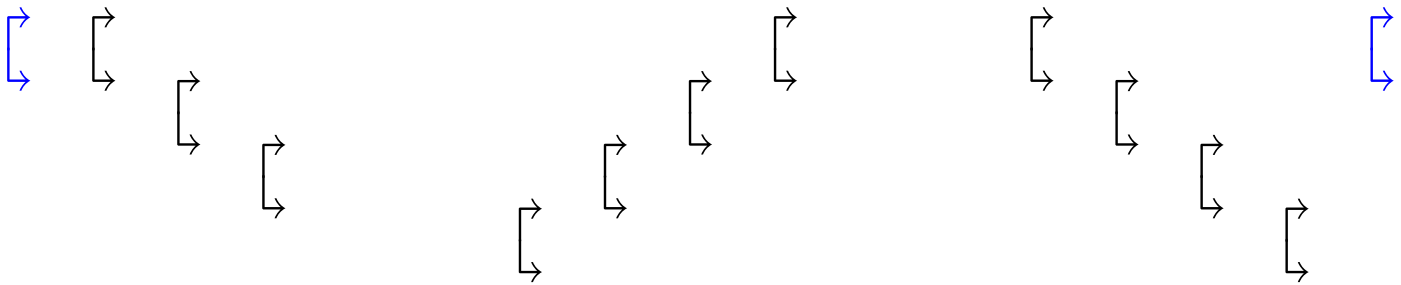
- ignoring rank-one part . . .

$A =$ 

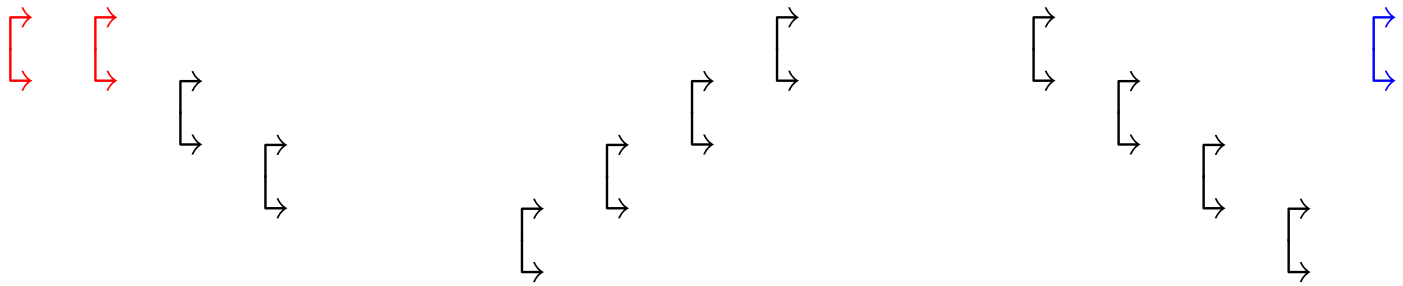
The Core Chase



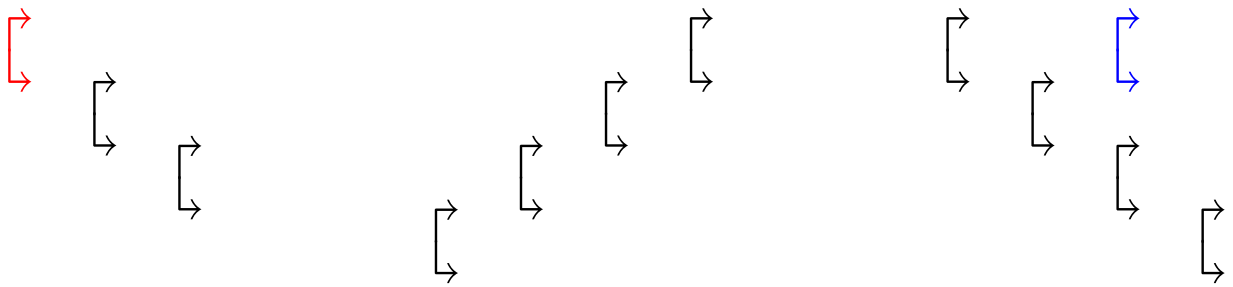
The Core Chase



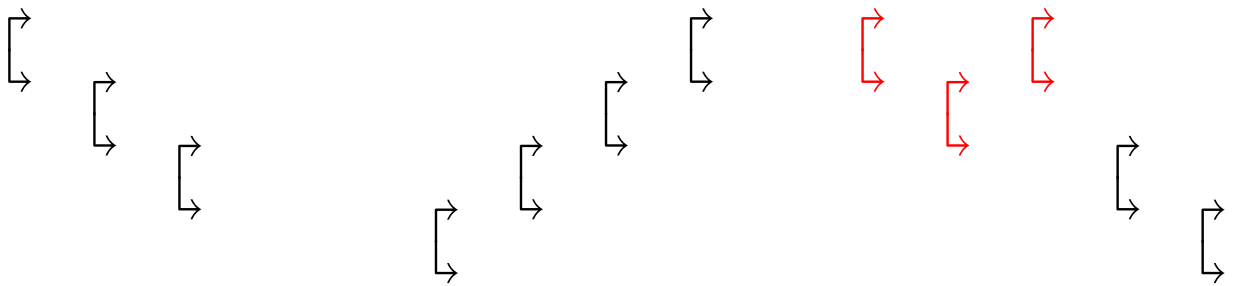
The Core Chase



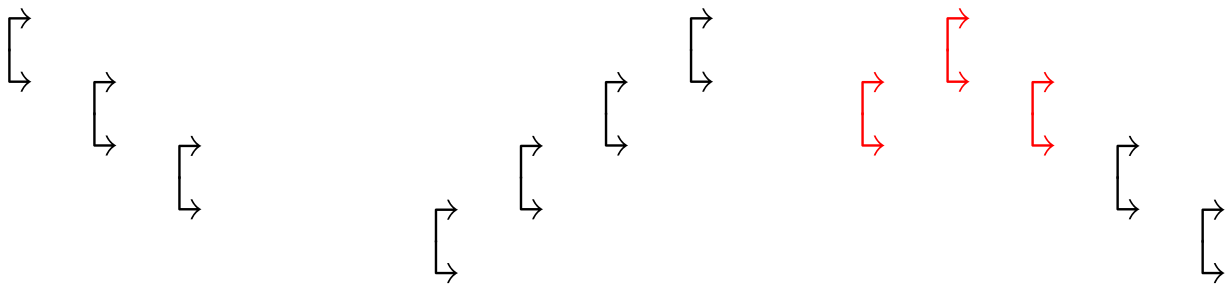
The Core Chase



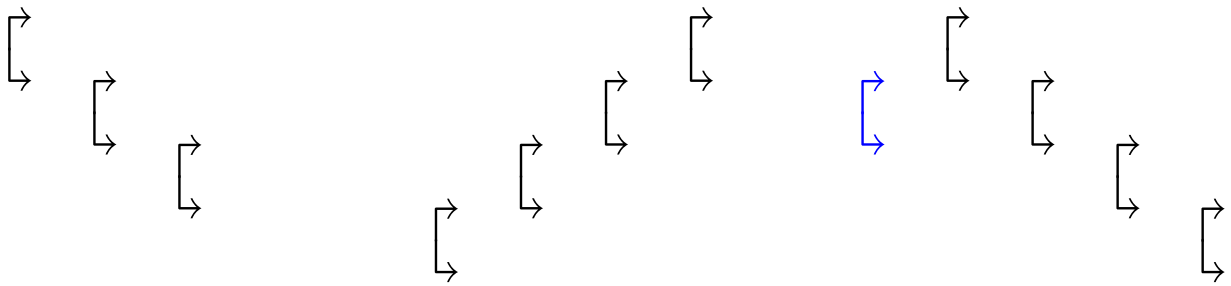
The Core Chase



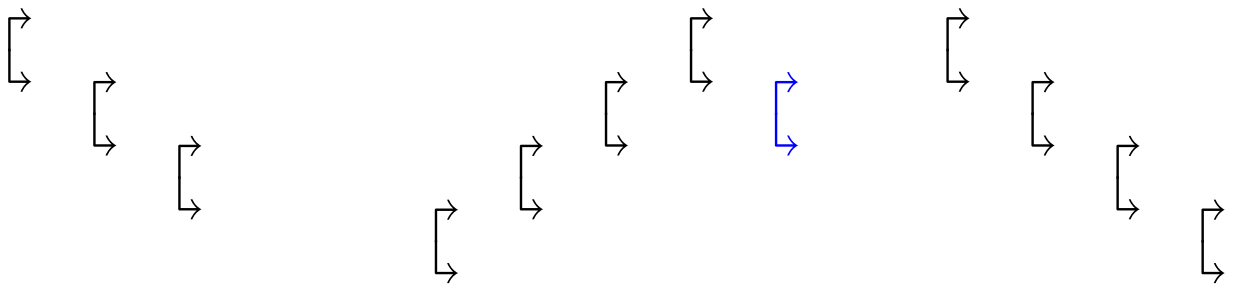
The Core Chase



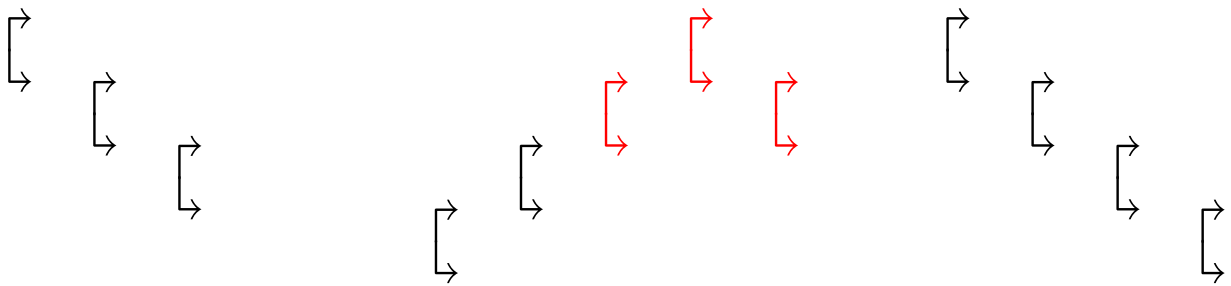
The Core Chase



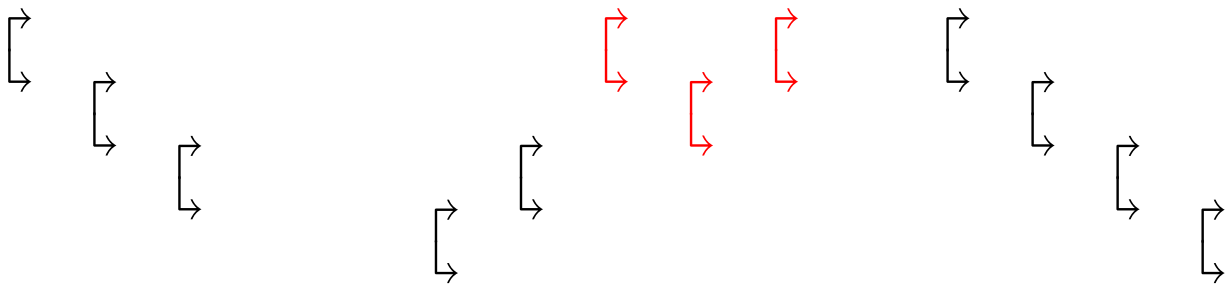
The Core Chase



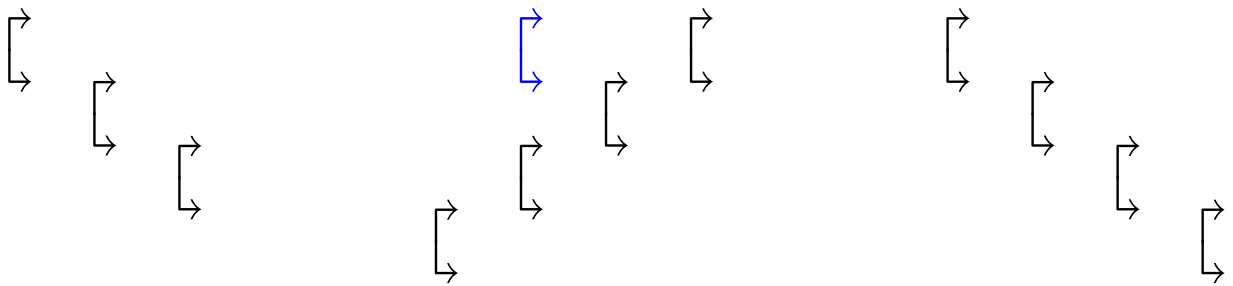
The Core Chase



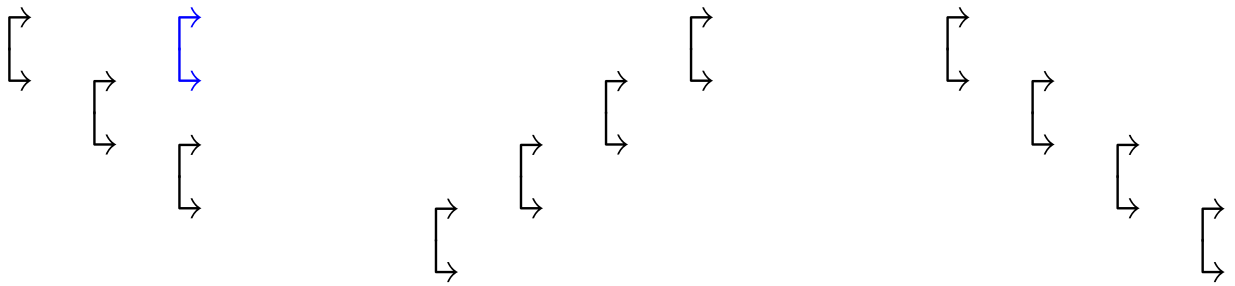
The Core Chase



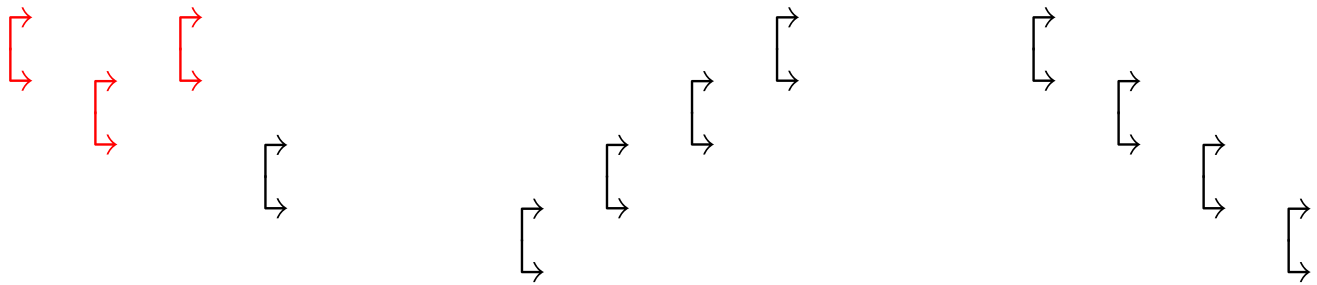
The Core Chase



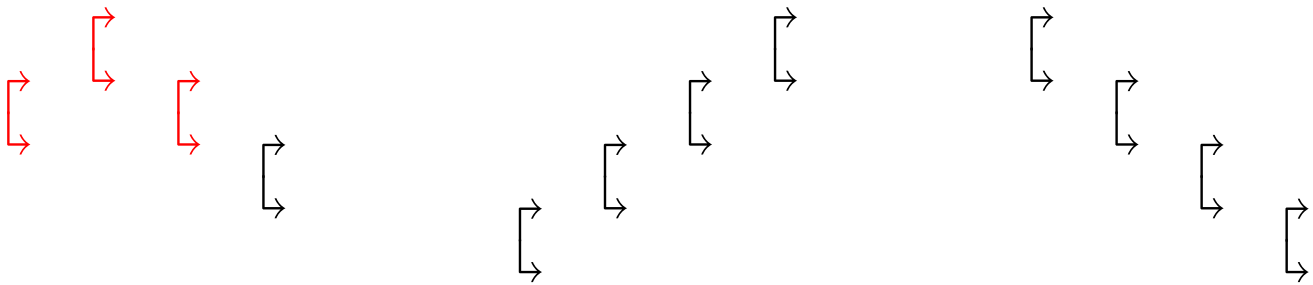
The Core Chase



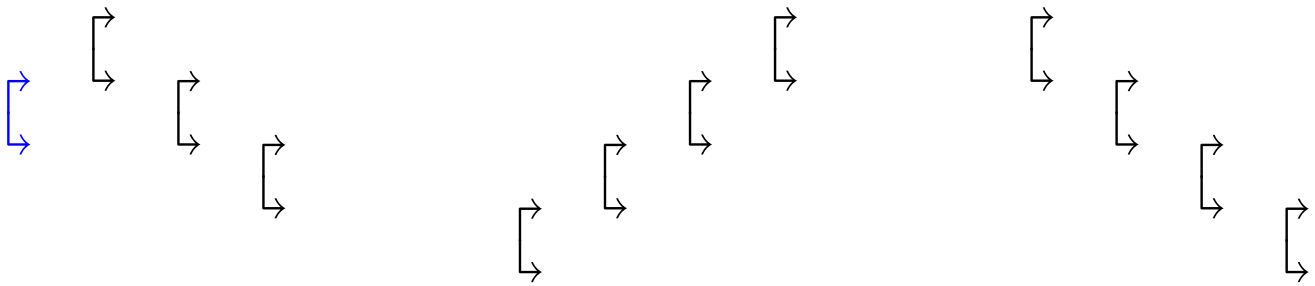
The Core Chase



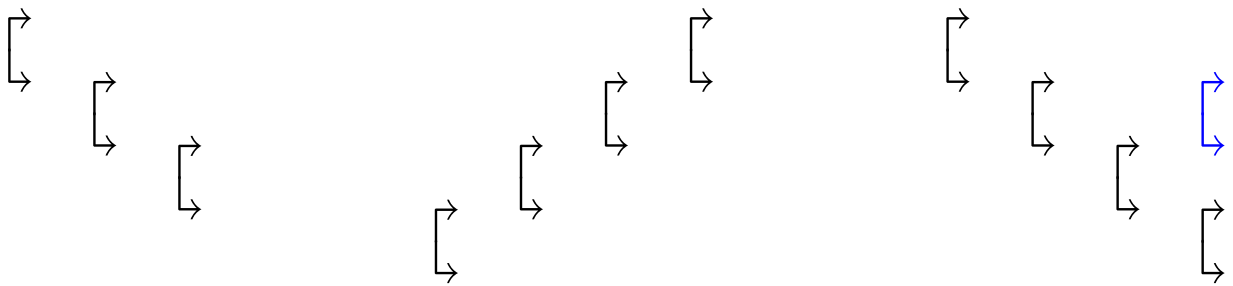
The Core Chase



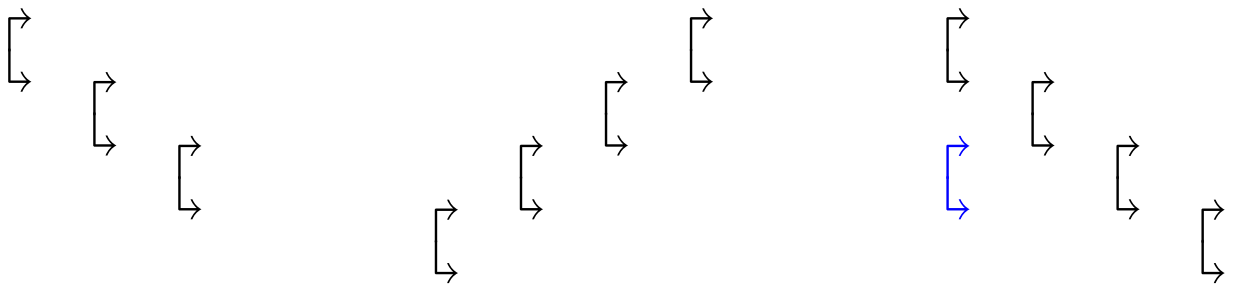
The Core Chase



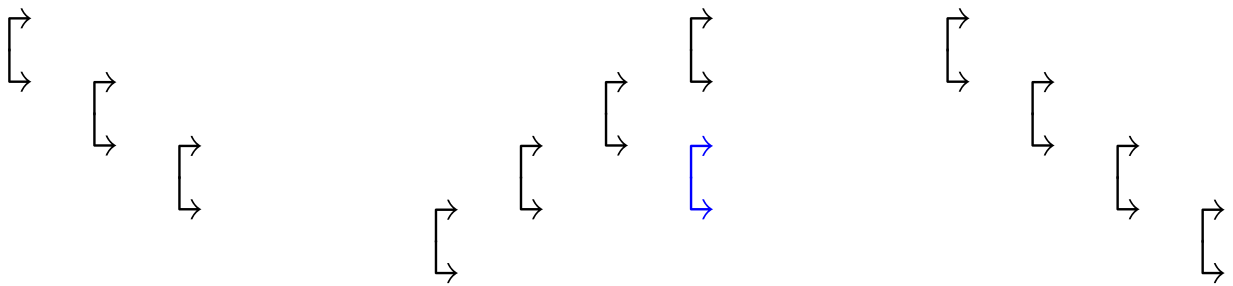
The Core Chase



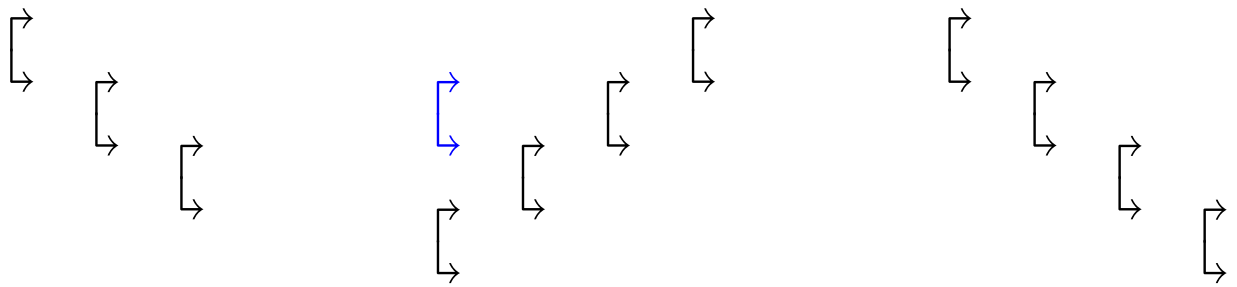
The Core Chase



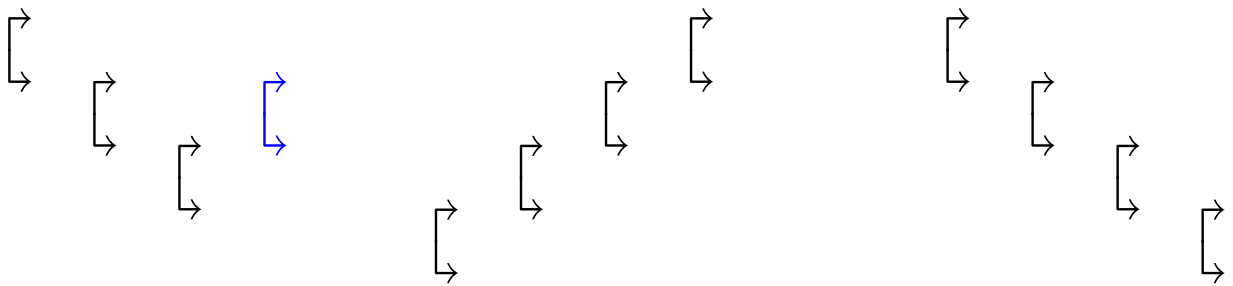
The Core Chase



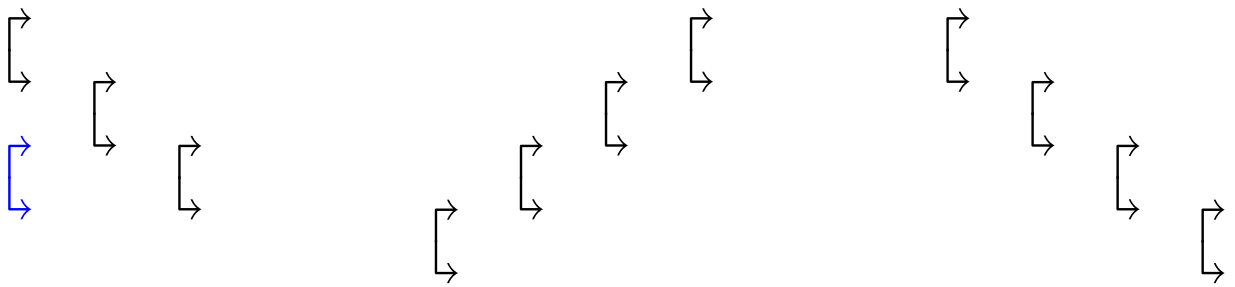
The Core Chase



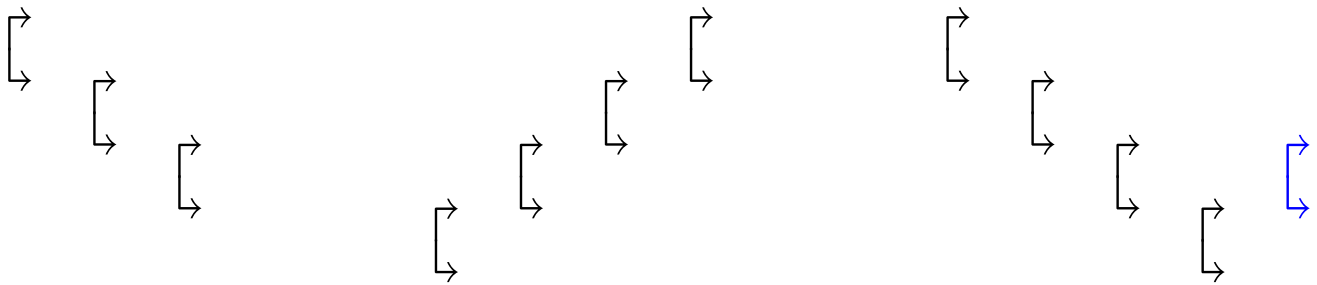
The Core Chase



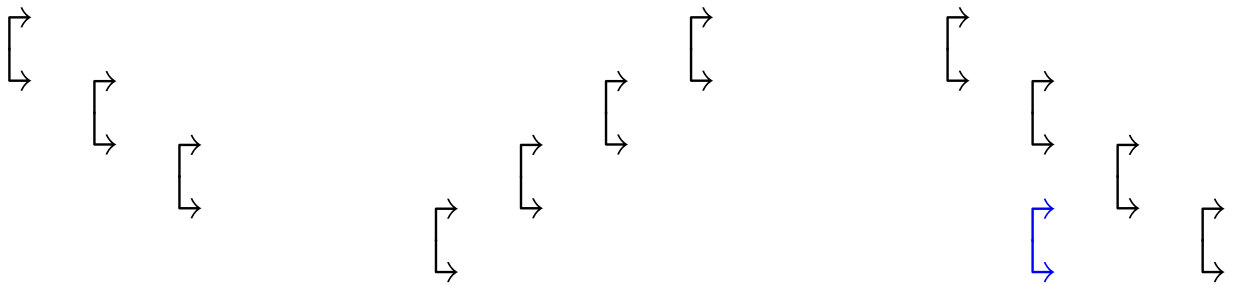
The Core Chase



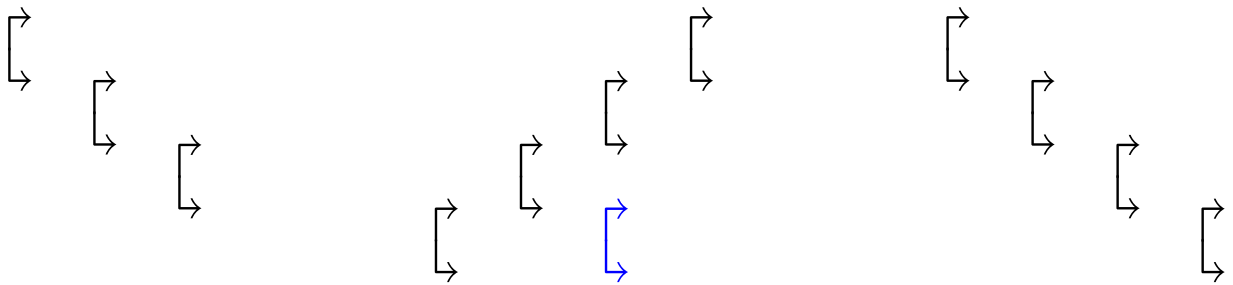
The Core Chase



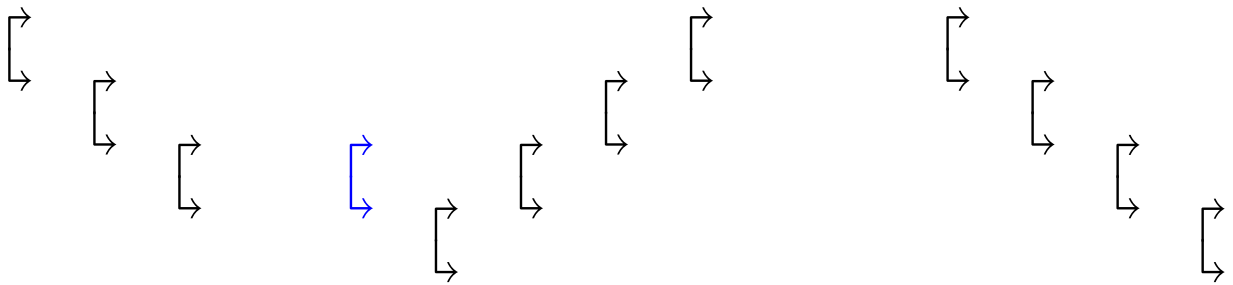
The Core Chase



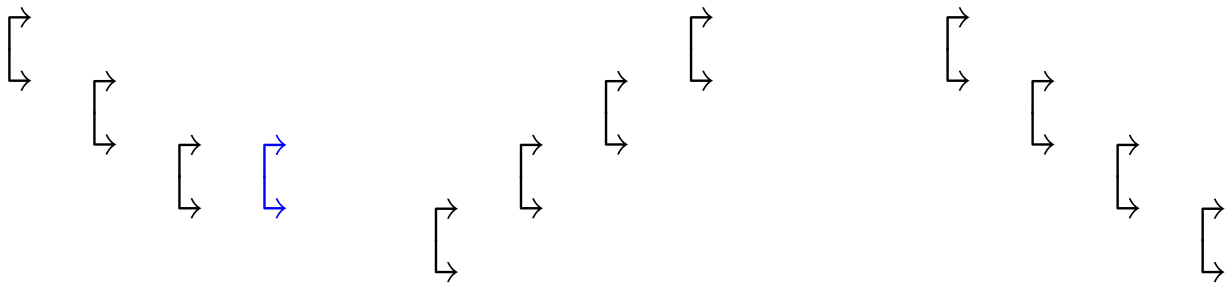
The Core Chase



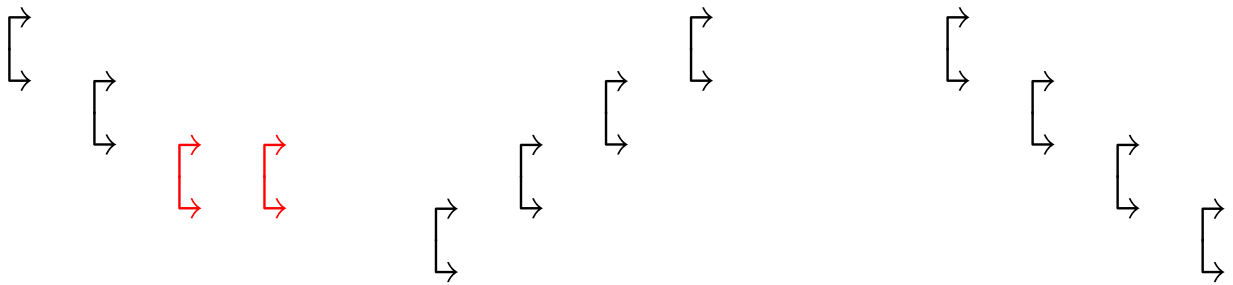
The Core Chase



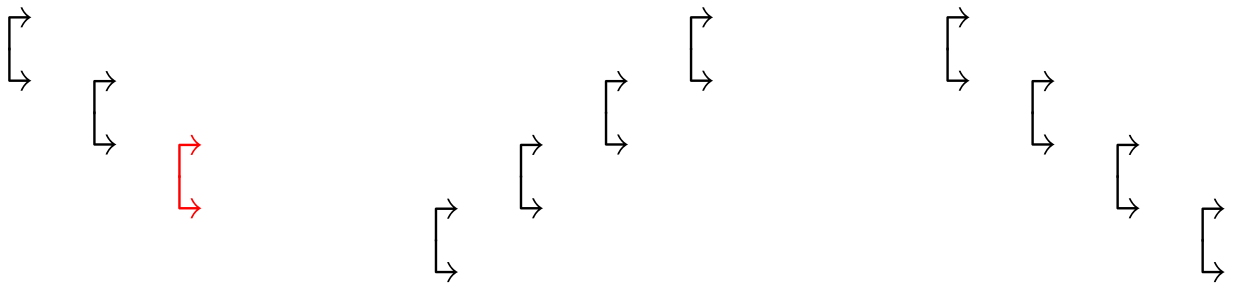
The Core Chase



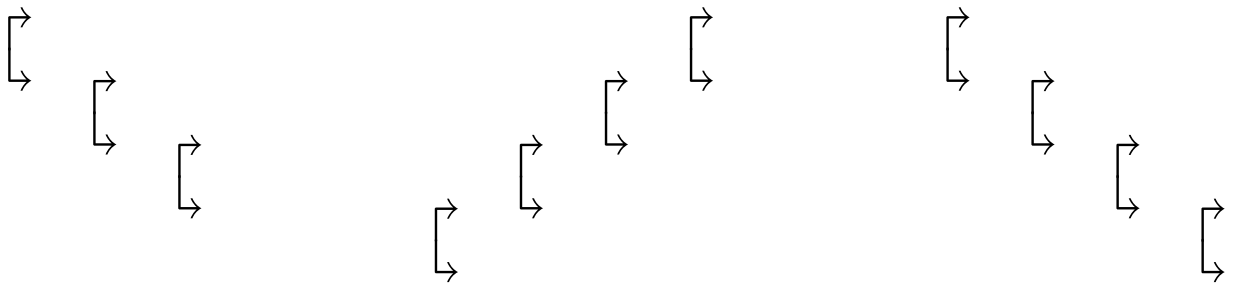
The Core Chase



The Core Chase



The Core Chase



Done!

- iteration complete!

Done!

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

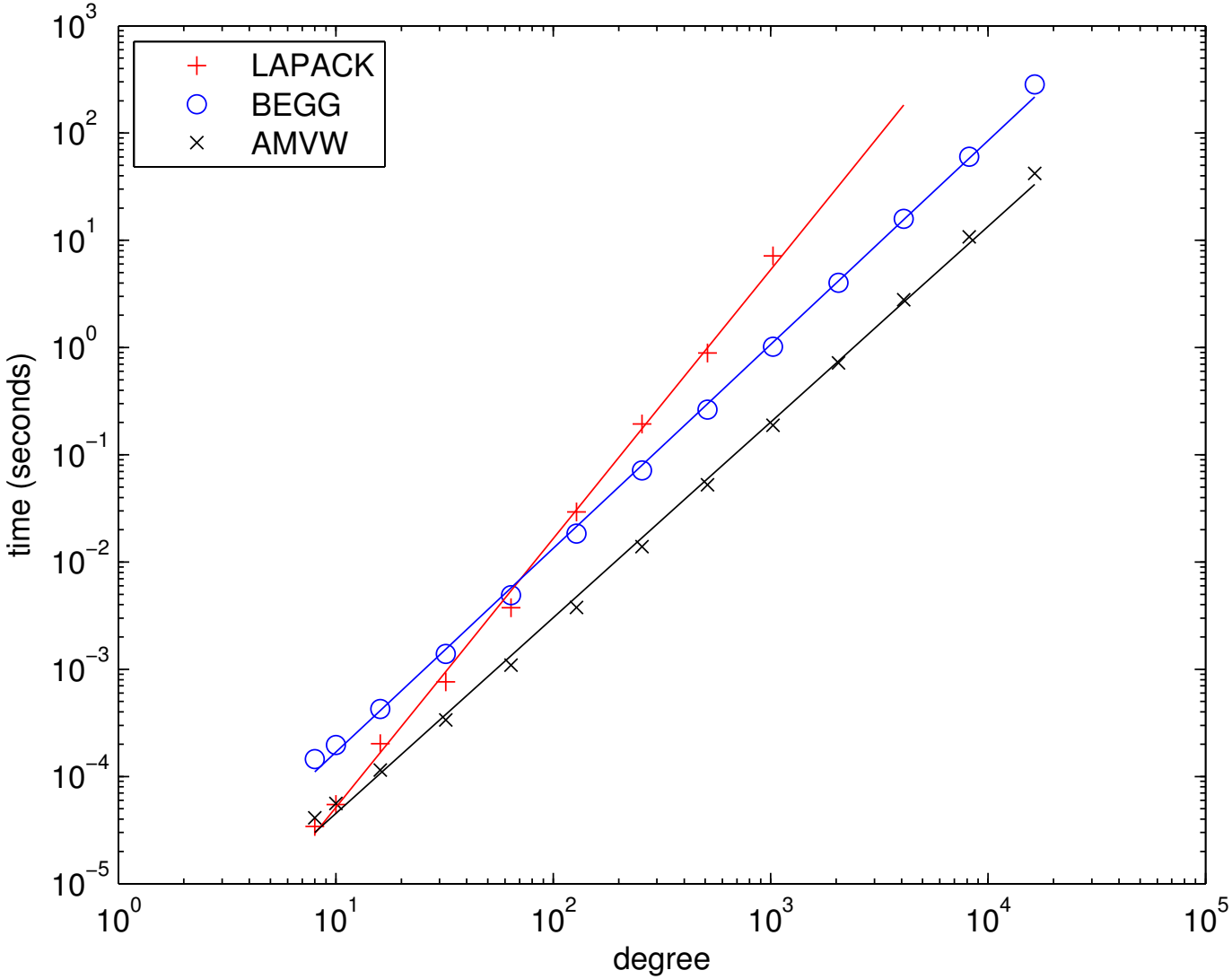
Done!

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

Done!

- iteration complete!
- Cost: $3n$ turnovers/iteration, so $O(n)$ flops/iteration.
- $O(n)$ iterations in all.
- Total flop count is $O(n^2)$.

Performance



Performance

At degree 1000

method	time
LAPACK	7.2
BEGG	1.2
AMVW	0.2

See our Papers

- For details see . . .

See our Papers

- For details see . . .
- Jared L. Aurentz, Thomas Mach, Raf Vandebril, and David S. Watkins, *Fast and backward stable computation of roots of polynomials,* SIAM J. Matrix Anal. Appl., 36 (2015), pp. 942–973.

See our Papers

- For details see . . .
- Jared L. Aurentz, Thomas Mach, Raf Vandebril, and David S. Watkins, *Fast and backward stable computation of roots of polynomials,* SIAM J. Matrix Anal. Appl., 36 (2015), pp. 942–973.
- Co-winner of SIAM Best Paper Prize (2017).

See our Papers

- For details see . . .
- Jared L. Aurentz, Thomas Mach, Raf Vandebril, and David S. Watkins, *Fast and backward stable computation of roots of polynomials,* SIAM J. Matrix Anal. Appl., 36 (2015), pp. 942–973.
- Co-winner of SIAM Best Paper Prize (2017).
- Jared L. Aurentz, Thomas Mach, Leonardo Robol, Raf Vandebril, and David S. Watkins, *Roots of polynomials: on twisted QR methods for companion matrices and pencils,* submitted for publication, arXiv:1611.02435.

See our Papers

- For details see . . .
- Jared L. Aurentz, Thomas Mach, Raf Vandebril, and David S. Watkins, *Fast and backward stable computation of roots of polynomials,* SIAM J. Matrix Anal. Appl., 36 (2015), pp. 942–973.
- Co-winner of SIAM Best Paper Prize (2017).
- Jared L. Aurentz, Thomas Mach, Leonardo Robol, Raf Vandebril, and David S. Watkins, *Roots of polynomials: on twisted QR methods for companion matrices and pencils,* submitted for publication, arXiv:1611.02435.
- Jared L. Aurentz, Thomas Mach, Leonardo Robol, Raf Vandebril, and David S. Watkins, *Fast and backward stable computation of the eigenvalues of matrix polynomials,* submitted for publication.

See our Papers

- For details see . . .
- Jared L. Aurentz, Thomas Mach, Raf Vandebril, and David S. Watkins, *Fast and backward stable computation of roots of polynomials,* SIAM J. Matrix Anal. Appl., 36 (2015), pp. 942–973.
- Co-winner of SIAM Best Paper Prize (2017).
- Jared L. Aurentz, Thomas Mach, Leonardo Robol, Raf Vandebril, and David S. Watkins, *Roots of polynomials: on twisted QR methods for companion matrices and pencils,* submitted for publication, arXiv:1611.02435.
- Jared L. Aurentz, Thomas Mach, Leonardo Robol, Raf Vandebril, and David S. Watkins, *Fast and backward stable computation of the eigenvalues of matrix polynomials,* submitted for publication.

Thank you for your attention.