Fast and backward stable computation of roots of polynomials

David S. Watkins

Department of Mathematics
Washington State University

July, 2017

Problem

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

Problem

- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n \qquad \text{(monic)}$
- Find the zeros.

Problem

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

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

Problem

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

Problem

- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n \qquad$ (monic)
- Find the zeros.
- 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

Problem

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

Problem

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

Problem

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

Problem

- $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + x^n$ (monic)
- Find the zeros.
- 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. Can we do better?

Our Paper

- Yes!

Our Paper

- Yes!
- Jared L. Aurentz, Thomas Mach, Raf Vandebril, and D. S. W.

Our Paper

- Yes!
- Jared L. Aurentz, Thomas Mach, Raf Vandebril, and D. S. W.
- Fast and backward stable computation of roots of polynomials

Our Paper

- Yes!
- Jared L. Aurentz, Thomas Mach, Raf Vandebril, and D. S. W.
- Fast and backward stable computation of roots of polynomials
- SIAM J. Matrix Anal. Appl. 36 (2015) pp. 942–973

Our Paper

- Yes!
- Jared L. Aurentz, Thomas Mach, Raf Vandebril, and D. S. W.
- Fast and backward stable computation of roots of polynomials
- SIAM J. Matrix Anal. Appl. 36 (2015) pp. 942–973
- SIAM Outstanding Paper prize, 2017

Our Paper

- Yes!
- Jared L. Aurentz, Thomas Mach, Raf Vandebril, and D. S. W.
- Fast and backward stable computation of roots of polynomials
- SIAM J. Matrix Anal. Appl. 36 (2015) pp. 942–973
- SIAM Outstanding Paper prize, 2017
- I'll tell you about this . . .

Our Paper

- Yes!
- Jared L. Aurentz, Thomas Mach, Raf Vandebril, and D. S. W.
- Fast and backward stable computation of roots of polynomials
- SIAM J. Matrix Anal. Appl. 36 (2015) pp. 942–973
- SIAM Outstanding Paper prize, 2017
- I'll tell you about this . . .
- . . . and also some more recent developments.

Unitary-plus-rank-one Structure

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

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

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

- 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.

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

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

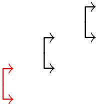
Storage Scheme, Part I

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

Storage Scheme, Part I

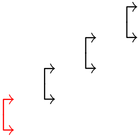
$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix} = \begin{bmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ & 0 & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix}$$

Storage Scheme, Part I



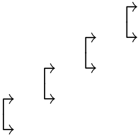
$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix} = \begin{bmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ & 0 & * & * & * \\ & & \color{red}{0} & * & * \\ & & & * & * \end{bmatrix}$$

Storage Scheme, Part I



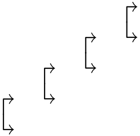
$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix} = \begin{bmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ & 0 & * & * & * \\ & & 0 & * & * \\ & & & \color{red}{0} & * \end{bmatrix}$$

Storage Scheme, Part I



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

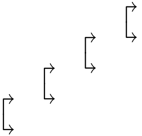
Storage Scheme, Part I



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

- **Def:** *Core Transformation*

Storage Scheme, Part I

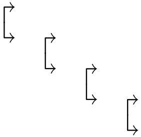


$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix} = \begin{bmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ & 0 & * & * & * \\ & & 0 & * & * \\ & & & 0 & * \end{bmatrix}$$

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

Storage Scheme, Part I

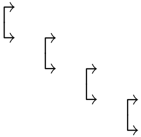
$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix} =$$



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

Storage Scheme, Part I

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

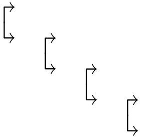


$$\begin{bmatrix} * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \\ & & & & * \end{bmatrix}$$

$$A = QR$$

Storage Scheme, Part I

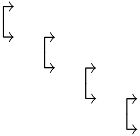
$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix} =$$



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

$$A = QR$$

$$Q =$$



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:

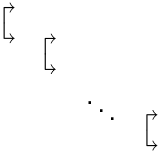
$$
\begin{aligned}
A &= QR \\
&= \begin{bmatrix} 0 & \cdots & 0 & 1 \\ 1 & & & 0 \\ & \ddots & & \vdots \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & -a_1 \\ & 1 & & -a_2 \\ & & \ddots & \vdots \\ & & & -a_0 \end{bmatrix}
\end{aligned}
$$

Storage Scheme, Part II

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

$$A = QR$$

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

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

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)

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

Storage Scheme, Part II

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

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

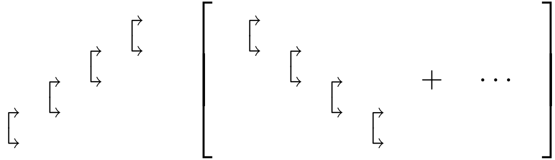
$$
= \left[\begin{array}{cccc|c} 1 & & & 0 & 0 \\ & \ddots & & \vdots & \vdots \\ & & 1 & 0 & 0 \\ & & & 0 & 1 \\ \hline & & & 1 & 0 \end{array}\right] + \left[\begin{array}{cccc|c} 0 & & & -a_1 & 0 \\ & \ddots & & \vdots & \vdots \\ & & 0 & -a_{n-1} & 0 \\ & & & -a_0 & 0 \\ \hline & & & -1 & 0 \end{array}\right]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

Storage Scheme, Part II

- Leaving out a few steps,

- Leaving out a few steps, we get

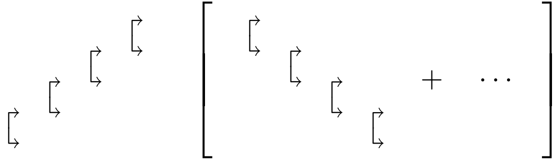
$\underline{R} =$



Storage Scheme, Part II

- Leaving out a few steps, we get

$\underline{R} =$

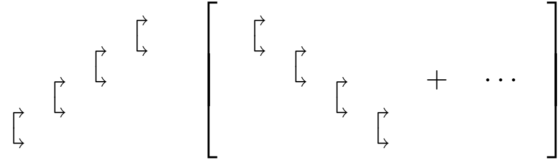


- Bonus: Redundant information (Read our paper.)

Storage Scheme, Part II

- Leaving out a few steps, we get

$\underline{R} =$

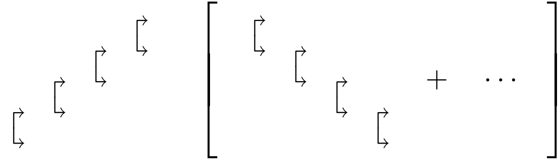


- Bonus: Redundant information (Read our paper.)
- We can ignore the rank-one part!

Storage Scheme, Part II

- Leaving out a few steps, we get

$$\underline{R} =$$

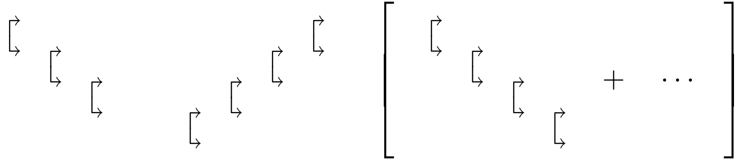


- Bonus: Redundant information (Read our paper.)
- We can ignore the rank-one part!
- Storage is $O(n)$.

Altogether we have

$$A = QR$$

$=$



- A is stored entirely in terms of core transformations.

Working with Core Transformations

Working with Core Transformations

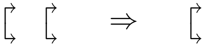
- We want to perform iterations of Francis's algorithm on this Structure.

Working with Core Transformations

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

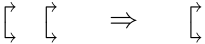
Working with Core Transformations

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

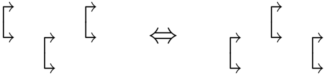


Working with Core Transformations

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

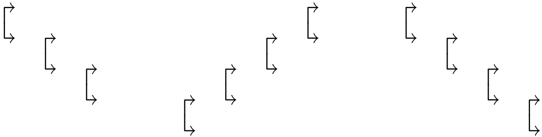


- Turnover (aka shift through, Givens swap, . . .)

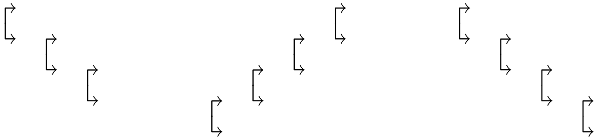


Francis Iteration (the core chase)

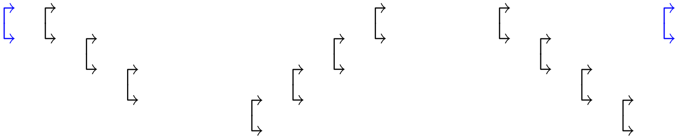
- ignoring rank-one part . . .

$A =$ 

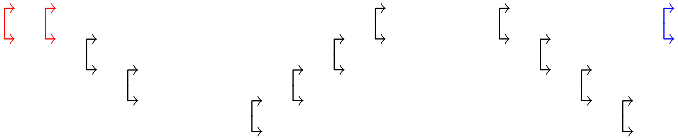
The Core Chase



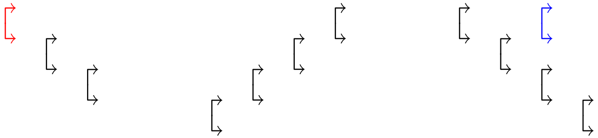
The Core Chase



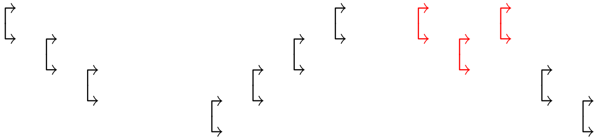
The Core Chase



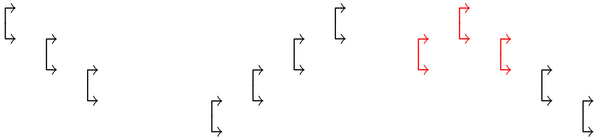
The Core Chase



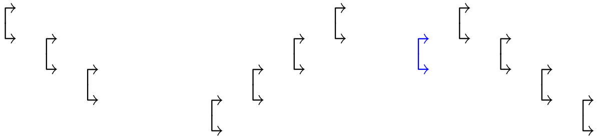
The Core Chase



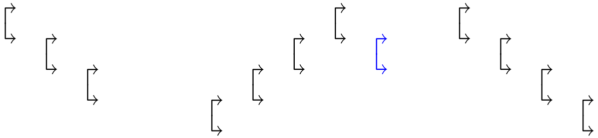
The Core Chase



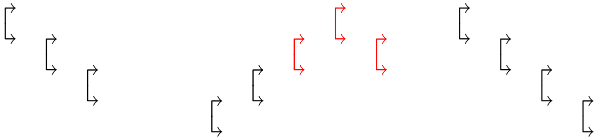
The Core Chase



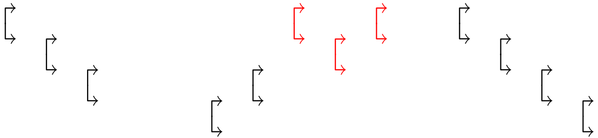
The Core Chase



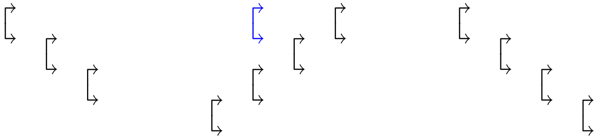
The Core Chase



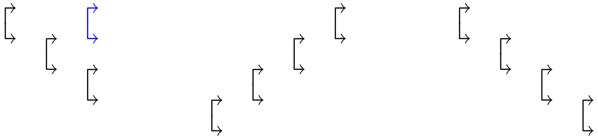
The Core Chase



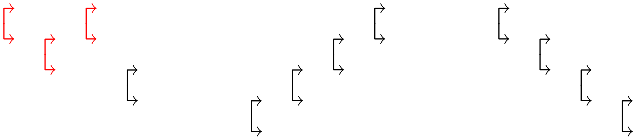
The Core Chase



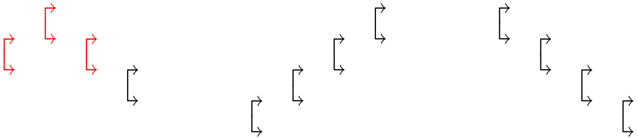
The Core Chase



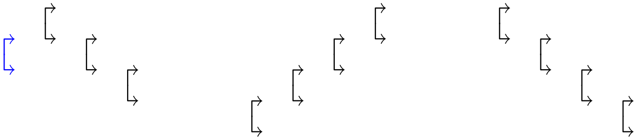
The Core Chase



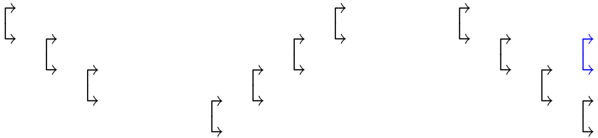
The Core Chase



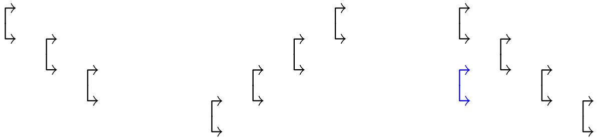
The Core Chase



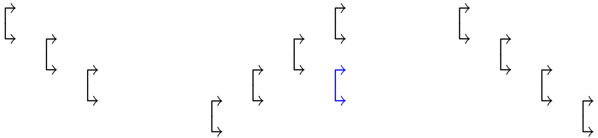
The Core Chase



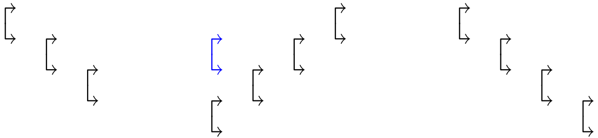
The Core Chase



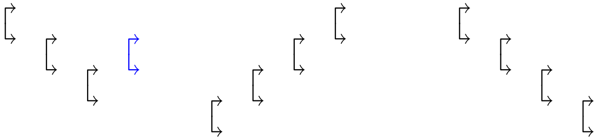
The Core Chase



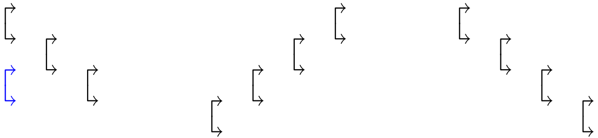
The Core Chase



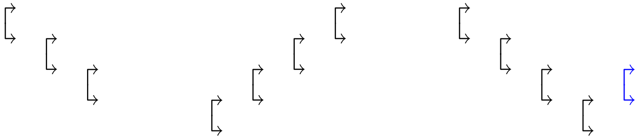
The Core Chase



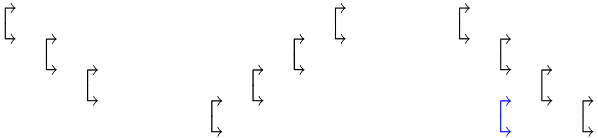
The Core Chase



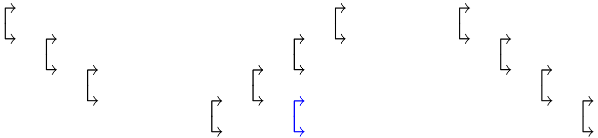
The Core Chase



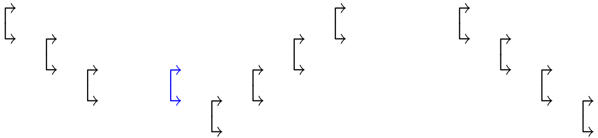
The Core Chase



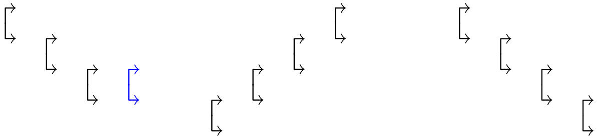
The Core Chase



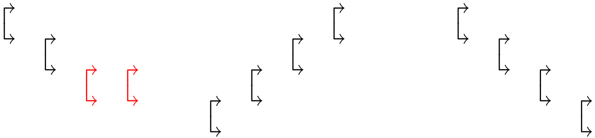
The Core Chase



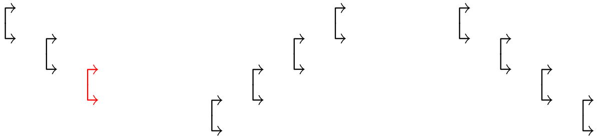
The Core Chase



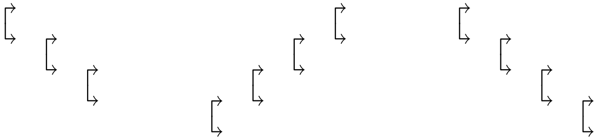
The Core Chase



The Core Chase



The Core Chase



Done!

- iteration complete!

Done!

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

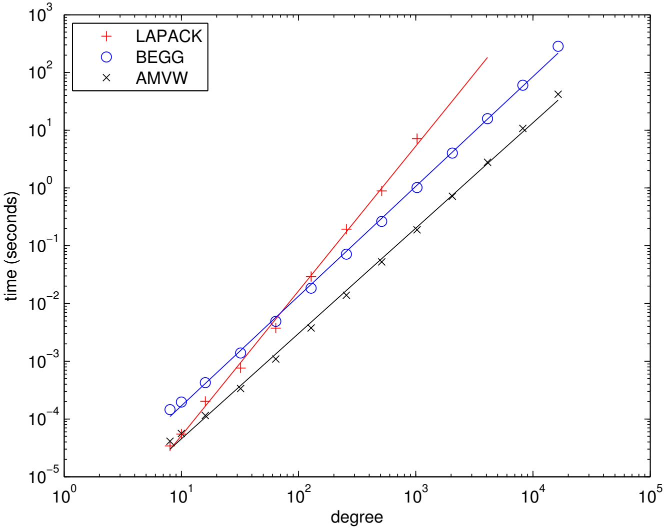
Done!

- 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



At degree 1000

method	time
LAPACK	7.2
BEGG	1.2
AMVW	0.2

Companion Pencil, a variant

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ (not monic)

Companion Pencil, a variant

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ (not monic)
- Divide by a_n,

Companion Pencil, a variant

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ (not monic)
- Divide by a_n, or ...

Companion Pencil, a variant

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ (not monic)
- 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

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ (not monic)
- 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

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ (not monic)
- 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),
- ... but is this really better?

Companion Pencil, a variant

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ (not monic)
- 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),
- ...but is this really better?
- Additional collaborator: Leonardo Robol

Companion Pencil, a variant

- $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ (not monic)
- 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),
- ... but is this really better?
- Additional collaborator: Leonardo Robol
- We can also handle matrix polynomials. (story for another day)

Our Book

- Monograph in Progress (130+ pp.)

Our Book

- Monograph in Progress (130+ pp.)
  - Jared Aurentz
  - Leonard Robol
  - Thomas Mach
  - Raf Vandebril
  - David Watkins

Backward Stability Odyssey

- These algorithms are obviously backward stable

Backward Stability Odyssey

- These algorithms are obviously backward stable
- because they act entirely on unitary matrices,

Backward Stability Odyssey

- These algorithms are obviously backward stable
- because they act entirely on unitary matrices,
- but it took us a while to write down a correct argument.

Backward Stability Odyssey

- These algorithms are obviously backward stable
- because they act entirely on unitary matrices,
- but it took us a while to write down a correct argument.
- First written attempt (horrible)

Backward Stability Odyssey

- These algorithms are obviously backward stable
- because they act entirely on unitary matrices,
- but it took us a while to write down a correct argument.
- First written attempt (horrible)
- Second attempt was much better (2015 paper) . . .

Backward Stability Odyssey

- These algorithms are obviously backward stable
- because they act entirely on unitary matrices,
- but it took us a while to write down a correct argument.
- First written attempt (horrible)
- Second attempt was much better (2015 paper) ...
- ...but there was one one more thing!

Backward Stability Odyssey

- These algorithms are obviously backward stable
- because they act entirely on unitary matrices,
- but it took us a while to write down a correct argument.
- First written attempt (horrible)
- Second attempt was much better (2015 paper) ...
- ... but there was one one more thing!
- Corrected in companion pencil paper.

Backward Stability Odyssey

- These algorithms are obviously backward stable
- because they act entirely on unitary matrices,
- but it took us a while to write down a correct argument.
- First written attempt (horrible)
- Second attempt was much better (2015 paper) ...
- ... but there was one one more thing!
- Corrected in companion pencil paper. We also exploited the structure of the backward error to get a better result.

Backward Stability Odyssey

- These algorithms are obviously backward stable
- because they act entirely on unitary matrices,
- but it took us a while to write down a correct argument.
- First written attempt (horrible)
- Second attempt was much better (2015 paper) . . .
- . . . but there was one one more thing!
- Corrected in companion pencil paper. We also exploited the structure of the backward error to get a better result. Rejected!

Backward Stability Odyssey

- These algorithms are obviously backward stable
- because they act entirely on unitary matrices,
- but it took us a while to write down a correct argument.
- First written attempt (horrible)
- Second attempt was much better (2015 paper) ...
- ... but there was one one more thing!
- Corrected in companion pencil paper. We also exploited the structure of the backward error to get a better result. Rejected!
- Search for examples.

Backward Stability Odyssey

- These algorithms are obviously backward stable
- because they act entirely on unitary matrices,
- but it took us a while to write down a correct argument.
- First written attempt (horrible)
- Second attempt was much better (2015 paper) . . .
- . . . but there was one one more thing!
- Corrected in companion pencil paper. We also exploited the structure of the backward error to get a better result. Rejected!
- Search for examples.
- Companion matrix code amazingly robust.

Backward Stability Odyssey

- These algorithms are obviously backward stable
- because they act entirely on unitary matrices,
- but it took us a while to write down a correct argument.
- First written attempt (horrible)
- Second attempt was much better (2015 paper) ...
- ... but there was one one more thing!
- Corrected in companion pencil paper. We also exploited the structure of the backward error to get a better result. Rejected!
- Search for examples.
- Companion matrix code amazingly robust.
- Take a closer look at backward error.

- Show computed roots of p are exact roots of a nearby polynomial $\hat{p}$.

Backward Stability Odyssey

- Show computed roots of p are exact roots of a nearby polynomial $\hat{p}$.
- Must $\hat{p}$ be monic?

Backward Stability Odyssey

- Show computed roots of p are
  exact roots of a nearby polynomial $\hat{p}$.
- Must $\hat{p}$ be monic? This makes a difference!

- Show computed roots of p are exact roots of a nearby polynomial $\hat{p}$.
- Must $\hat{p}$ be monic? This makes a difference!
- If monic: $\| a - \hat{a} \| \lesssim u \| a \|^2$ (a is coefficient vector of p)

Backward Stability Odyssey

- Show computed roots of p are exact roots of a nearby polynomial $\hat{p}$.
- Must $\hat{p}$ be monic? This makes a difference!
- If monic: $\| a - \hat{a} \| \lesssim u \| a \|^2$ (a is coefficient vector of p)
- if not: $\| a - \hat{a} \| \lesssim u \| a \|$

Backward Stability Odyssey

- Show computed roots of p are exact roots of a nearby polynomial $\hat{p}$.
- Must $\hat{p}$ be monic? This makes a difference!
- If monic: $\| a - \hat{a} \| \lesssim u \| a \|^2$ (a is coefficient vector of p)
- if not: $\| a - \hat{a} \| \lesssim u \| a \|$
- good as we could hope for

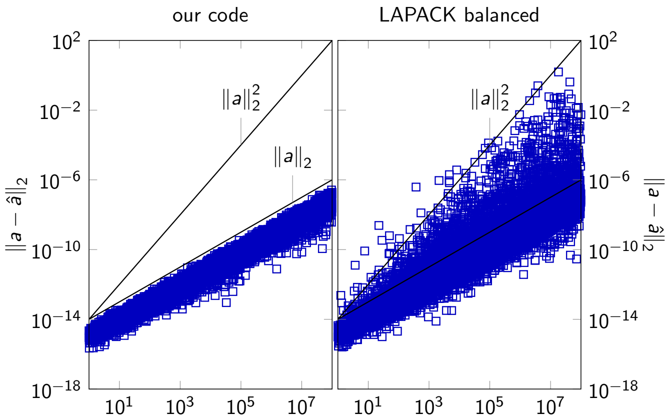
Backward Stability Odyssey

- Show computed roots of p are exact roots of a nearby polynomial $\hat{p}$.
- Must $\hat{p}$ be monic? This makes a difference!
- If monic: $\| a - \hat{a} \| \lesssim u \| a \|^2$ (a is coefficient vector of p)
- if not: $\| a - \hat{a} \| \lesssim u \| a \|$
- good as we could hope for
- confirmed by numerical experiments

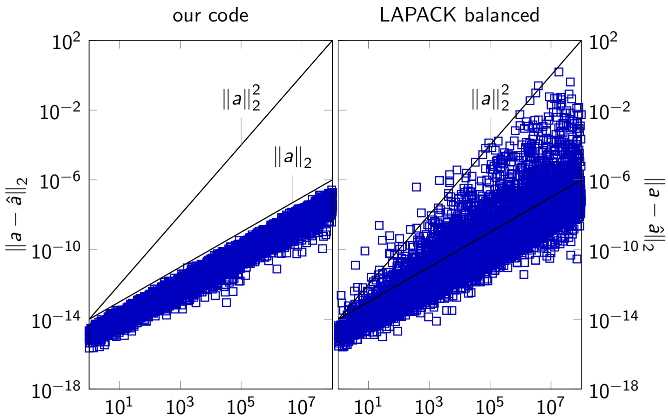
Backward Stability Odyssey

- Show computed roots of p are exact roots of a nearby polynomial $\hat{p}$.
- Must $\hat{p}$ be monic? This makes a difference!
- If monic: $\| a - \hat{a} \| \lesssim u \| a \|^2$ (a is coefficient vector of p)
- if not: $\| a - \hat{a} \| \lesssim u \| a \|$
- good as we could hope for
- confirmed by numerical experiments
- Meaning: Most of the error is "parallel" to p and is therefore irrelevant.

Nice Picture

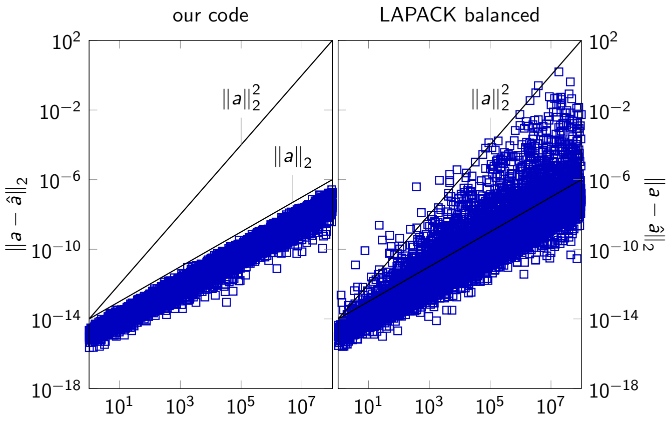


Nice Picture

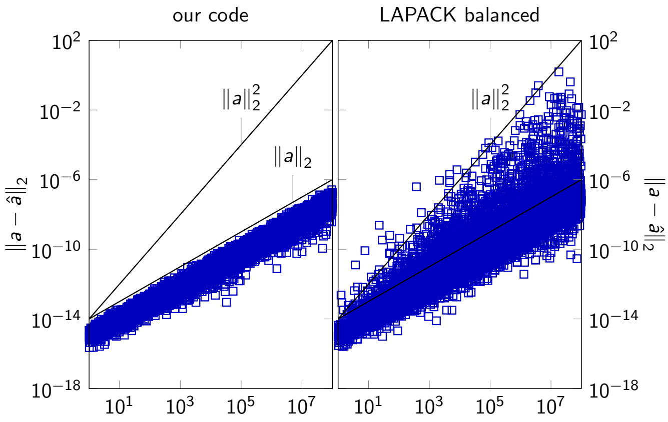


Our code is not just faster,

Nice Picture



Our code is not just faster, it is also more accurate!



Our code is not just faster, it is also more accurate!

Thank you for your attention.