Fast and backward stable computation of the eigenvalues of matrix polynomials

Leonardo Robol, KU Leuven
<leonardo.robol@cs.kuleuven.be>

joint work with Jared L. Aurentz (University of Oxford), Thomas Mach (Nazarbayev University), Raf Vandebril (KU Leuven) and David S. Watkins (WSU)

24 Oct 2016, Luminy

The problem

We are interested in solving Polynomial Eigenvalue Problems, that is finding the *eigenvalues* λ such that:

$$\det P(\lambda)v = 0, \qquad P(\lambda) = \sum_{j=0}^{d} P_j \lambda^j, \qquad P_j \in \mathbb{C}^{m\times m}.$$

The problem

We are interested in solving Polynomial Eigenvalue Problems, that is finding the *eigenvalues* λ such that:

$$\det P(\lambda)v = 0, \qquad P(\lambda) = \sum_{j=0}^{d} P_j \lambda^j, \qquad P_j \in \mathbb{C}^{m\times m}.$$

- Only square ($m \times m$) matrix polynomials.
- We are mainly interested in the *eigenvalues*.
- We want a method with an optimal asymptotic cost.

The classical way

The usual strategy to solve a PEP is to build the Frobenius companion linearization:

$$W - \lambda V = \begin{bmatrix} 0_m & \dots & 0_m & -P_0 \\ I_m & & & -P_1 \\ & \ddots & & \vdots \\ & & I_m & -P_{d-1} \end{bmatrix} - \lambda \begin{bmatrix} I_m & & & \\ & \ddots & & \\ & & I_m & \\ & & & P_d \end{bmatrix}$$

and solve it via the QZ method. The cost is cubic in the size $\Longrightarrow \mathcal{O}(m^3 d^3)$ flops.

We claim that we can do better than that!

What should be the optimal cost?

Let's look at some specific cases we are all familiar with:

What should be the optimal cost?

Let's look at some specific cases we are all familiar with:

$d = 1$ Eigenvalues of an $m \times m$ matrix pencil: we expect the cost to be $\mathcal{O}(m^3)$.

What should be the optimal cost?

Let's look at some specific cases we are all familiar with:

- $d = 1$ Eigenvalues of an $m \times m$ matrix pencil: we expect the cost to be $\mathcal{O}(m^3)$.
- $m = 1$ Roots of a scalar polynomial: exploiting the structure we are able to do that in $\mathcal{O}(d^2)$ flops (in a proven backward stable way!).

What should be the optimal cost?

Let's look at some specific cases we are all familiar with:

- $d = 1$ Eigenvalues of an $m \times m$ matrix pencil: we expect the cost to be $\mathcal{O}(m^3)$.
- $m = 1$ Roots of a scalar polynomial: exploiting the structure we are able to do that in $\mathcal{O}(d^2)$ flops (in a proven backward stable way!).

By combining the above remarks, we aim for $\mathcal{O}(d^2 m^3)$ flops.

Some background

We extend the work of Aurentz et al. for the scalar case, which exploits writing a scalar companion matrix in a special structured form.

Some background

We extend the work of Aurentz et al. for the scalar case, which exploits writing a scalar companion matrix in a special structured form.

Let $p(\lambda)$ be a scalar (monic) polynomial of degree d. Its roots are the eigenvalues of the companion matrix:

$$\begin{bmatrix} & & -p_0 \\ 1 & & -p_1 \\ & \ddots & \vdots \\ & 1 & -p_{d-1} \end{bmatrix} = \begin{bmatrix} & & 1 \\ 1 & & \\ & \ddots & \\ & 1 & \end{bmatrix} + \begin{bmatrix} -p_0 - 1 \\ -p_1 \\ \vdots \\ -p_{d-1} \end{bmatrix} e_d^T$$

Unitary plus low rank structure

The companion matrix is unitary plus rank 1, and this structure is preserved by the QR (or QZ) iterations. How to parametrize it?

Unitary plus low rank structure

The companion matrix is unitary plus rank 1, and this structure is preserved by the QR (or QZ) iterations. How to parametrize it?

Step 0: Enlarge the matrix by adding a row and a column:

$$\begin{bmatrix} & & & -p_0 & 1 \\ 1 & & & -p_1 & \vdots \\ & \ddots & & \vdots & \vdots \\ & & 1 & -p_{d-1} & 0 \\ 0 & \dots & \dots & 0 & 0 \end{bmatrix}$$

Unitary plus low rank structure

The companion matrix is unitary plus rank 1, and this structure is preserved by the QR (or QZ) iterations. How to parametrize it?

Step 1: Compute its QR factorization

$$\mathcal{Q}R = \begin{bmatrix} & & & 1 & 0 \\ 1 & & & & \\ & \ddots & & & \\ & & 1 & & 0 \\ & & & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & & & -p_1 & 0 \\ & \ddots & & \vdots & \vdots \\ & & 1 & -p_{d-1} & \vdots \\ & & & -p_0 & 1 \\ 0 & \dots & \dots & 0 & 0 \end{bmatrix}$$

Unitary plus low rank structure

The companion matrix is unitary plus rank 1, and this structure is preserved by the QR (or QZ) iterations. How to parametrize it?

Step 2: Decompose the R factor as unitary plus rank 1:

$$R = Y + xy^T = \begin{bmatrix} 1 & & & 0 & 0 \\ & \ddots & & 0 & \vdots \\ & & 1 & \vdots & \vdots \\ & & & 0 & 1 \\ 0 & \dots & \dots & 1 & 0 \end{bmatrix} + \begin{bmatrix} -p_1 \\ \vdots \\ -p_{d-1} \\ -p_0 \\ -1 \end{bmatrix} e_d^T$$

Unitary plus low rank structure

The companion matrix is unitary plus rank 1, and this structure is preserved by the QR (or QZ) iterations. How to parametrize it?

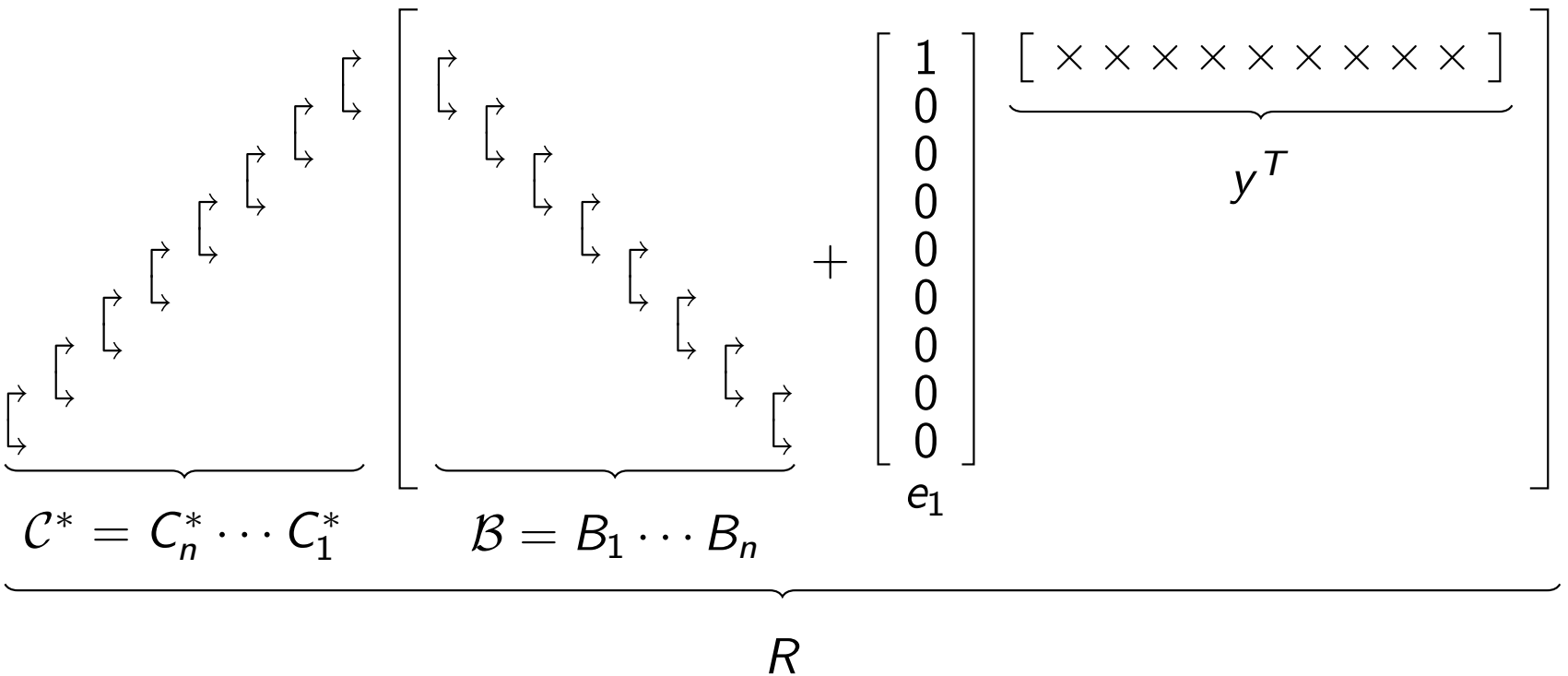
Step 3: "Roll up" the column with the coefficients

$$\mathcal{C} \begin{bmatrix} -p_1 \\ \vdots \\ -p_{d-1} \\ -p_0 \\ -1 \end{bmatrix} = \alpha e_1,$$

and set $\mathcal{B} := \mathcal{C}^* Y$, $y = \alpha e_d^T$, so $W = \mathcal{Q}\mathcal{C}^*(\mathcal{B} + e_1 y^T)$.

Core transformations

The matrices $\mathcal{Q}, \mathcal{C}$, and $\mathcal{B}$ are sequences of *core transformations*, that are essentially 2×2 matrices. We can write R as:



Representing the leading coefficients

The same kind of factorization can be done for the leading coefficient, but there is no need for the $\mathcal{Q}$.

Representing the leading coefficients

The same kind of factorization can be done for the leading coefficient, but there is no need for the $\mathcal{Q}$.

We have

$$V = \mathcal{C}_V^*(\mathcal{B}_V + e_1 y_V^T), \qquad W = \mathcal{Q}\mathcal{C}_W^*(\mathcal{B}_W + e_1 y_V^T).$$

Representing the leading coefficients

The same kind of factorization can be done for the leading coefficient, but there is no need for the $\mathcal{Q}$.

We have

$$V = \mathcal{C}_V^*(\mathcal{B}_V + e_1 y_V^T), \qquad W = \mathcal{Q}\mathcal{C}_W^*(\mathcal{B}_W + e_1 y_V^T).$$

Basic idea for the QZ: perform unitary similarities with core transformations until W is upper triangular.

Representing the leading coefficients

The same kind of factorization can be done for the leading coefficient, but there is no need for the $\mathcal{Q}$.

We have

$$V = \mathcal{C}_V^*(\mathcal{B}_V + e_1 y_V^T), \qquad W = \mathcal{Q}\mathcal{C}_W^*(\mathcal{B}_W + e_1 y_V^T).$$

Basic idea for the QZ: perform unitary similarities with core transformations until W is upper triangular.

In our language, that means $\mathcal{Q} = I$.

The matrix polynomial case

We know how to perform the QZ efficiently on unitary plus rank 1 matrices. What about unitary plus rank m?

- Treating the unitary plus rank m directly is difficult, and easily leads to high complexities in m, or instabilities.

The matrix polynomial case

We know how to perform the QZ efficiently on unitary plus rank 1 matrices. What about unitary plus rank m?

- Treating the unitary plus rank m directly is difficult, and easily leads to high complexities in m, or instabilities.
- Unitary plus rank m representations can be cumbersome to work with: it is very difficult to get efficient implementations.

The matrix polynomial case

We know how to perform the QZ efficiently on unitary plus rank 1 matrices. What about unitary plus rank m?

- Treating the unitary plus rank m directly is difficult, and easily leads to high complexities in m, or instabilities.
- Unitary plus rank m representations can be cumbersome to work with: it is very difficult to get efficient implementations.
- . . . so we do what mathematician always do: we go back to the previous case.

A nice factorization trick

Let W be the (block) companion of a matrix polynomial $P(\lambda)$. Assume the constant coefficient P_0 is lower triangular. Then we have:

$$W = W_1 \dots W_m,$$

where W_i are scalar companion matrices of polynomial having as coefficients (some of) the coefficients of $P(\lambda)$, appropriately permuted.

A nice factorization trick

Let W be the (block) companion of a matrix polynomial $P(\lambda)$. Assume the constant coefficient P_0 is lower triangular. Then we have:

$$W = W_1 \dots W_m,$$

where W_i are scalar companion matrices of polynomial having as coefficients (some of) the coefficients of $P(\lambda)$, appropriately permuted.

In particular, the factorization can be computed at *no cost*!

An example

$$\begin{bmatrix} 0 & & & \alpha_1 \\ 1 & & & \vdots \\ & \ddots & & \vdots \\ & & \ddots & \alpha_{md-1} \\ & & 1 & \alpha_{md} \end{bmatrix} \begin{bmatrix} 0 & & & \beta_1 \\ 1 & & & \vdots \\ & \ddots & & \vdots \\ & & \ddots & \beta_{md-1} \\ & & 1 & 0 \end{bmatrix}$$

An example

$$\begin{bmatrix} & & & \alpha_1 & 0 \\ & & & \vdots & \beta_1 \\ 1 & & & \vdots & \vdots \\ & \ddots & & \vdots & \vdots \\ & & 1 & \alpha_{md} & \beta_{md-1} \end{bmatrix}$$

Almost the same trick for the leading coefficient

We can factor the leading coefficient in a similar way, if we assume P_d to be upper triangular

$$V = V_1 \dots V_d$$

Almost the same trick for the leading coefficient

We can factor the leading coefficient in a similar way, if we assume P_d to be upper triangular

$$V = V_1 \dots V_d$$

V is upper triangular, and V_j are all elementary Gauss transformations. Again, no computations needed!

Almost the same trick for the leading coefficient

We can factor the leading coefficient in a similar way, if we assume P_d to be upper triangular

$$V = V_1 \dots V_d$$

V is upper triangular, and V_j are all elementary Gauss transformations. Again, no computations needed!

A quick look at Gaussian decomposition

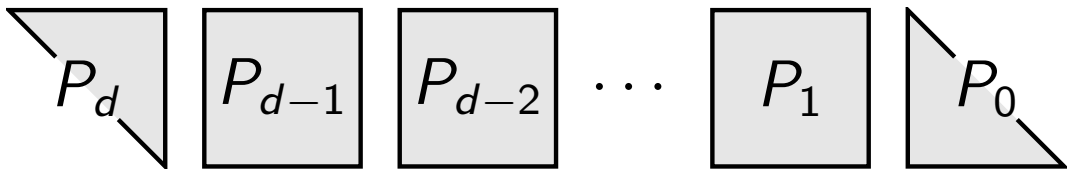
$$\begin{bmatrix} 1 & & & \alpha_1 \\ & \ddots & & \vdots \\ & & \ddots & \vdots \\ & & & \alpha_{md} \end{bmatrix} \begin{bmatrix} 1 & & \beta_1 & \\ & \ddots & \vdots & \\ & & \beta_{md-1} & \\ & & & 1 \end{bmatrix}$$

A quick look at Gaussian decomposition

$$\begin{bmatrix} 1 & & \beta_1 & \alpha_1 \\ & \ddots & \vdots & \vdots \\ & & & \vdots \\ & & \beta_{md-1} & \\ & & & \alpha_{md} \end{bmatrix}$$

The required structure

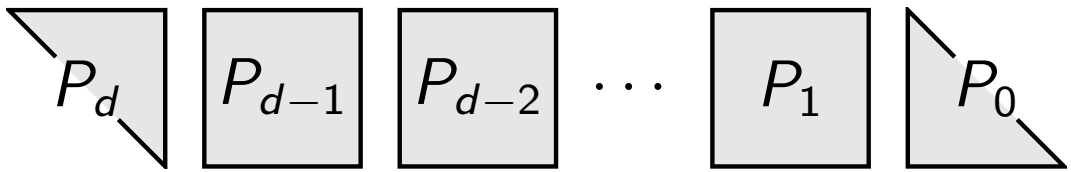
In order to compute these decompositions we need the initial structure of the matrix polynomial to be



Not restrictive, can be computed by QZ-like algorithm.

The required structure

In order to compute these decompositions we need the initial structure of the matrix polynomial to be



Not restrictive, can be computed by QZ-like algorithm.

Our pencil is now

$$W_1 \dots W_m - \lambda V_1 \dots V_m,$$

which we can solve as a product eigenvalue problem!

What's missing

Hessenberg-Triangular reduction The matrices V_j are upper triangular, but each of the W_j is upper Hessenberg, thus the product is m-Hessenberg. We need to eliminate $m-1$ subdiagonals.

Chasing We can implement the chasing passing rotations through the triangular factors.

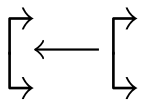
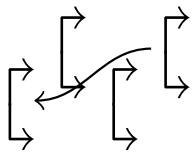
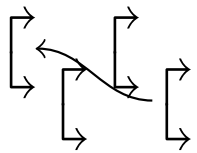
What's missing

Hessenberg-Triangular reduction The matrices V_j are upper triangular, but each of the W_j is upper Hessenberg, thus the product is m-Hessenberg. We need to eliminate $m-1$ subdiagonals.

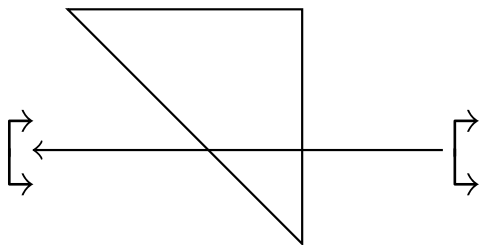
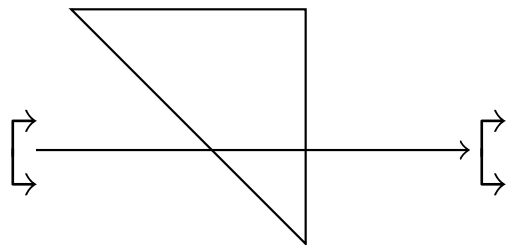
Chasing We can implement the chasing passing rotations through the triangular factors.

To understand how this works, we need some basic operations with core transformations.

Using core transformations

Fusion:  Turnover:  ,  .

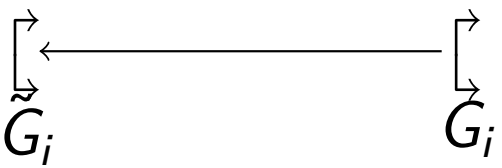
And then we need to pass rotation through upper triangular matrices:

 or 

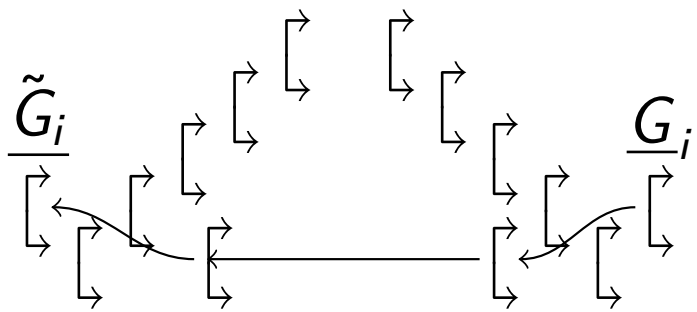
Exploiting the structure

Since we have structured upper triangular matrices, we can pass rotation through them in a fast way:

$$(\text{recall that } R = \mathcal{C}^*(\mathcal{B} + e_1 y^T))$$



is computed as



Hessenberg-Triangular reduction

The QZ and the Hessenberg-Triangular reduction are (formally) equivalent to the Hessenberg reduction and QR on WV^{-1}.

Hessenberg-Triangular reduction

The QZ and the Hessenberg-Triangular reduction are (formally) equivalent to the Hessenberg reduction and QR on WV^{-1}.

We can write our factorization in a structured form:

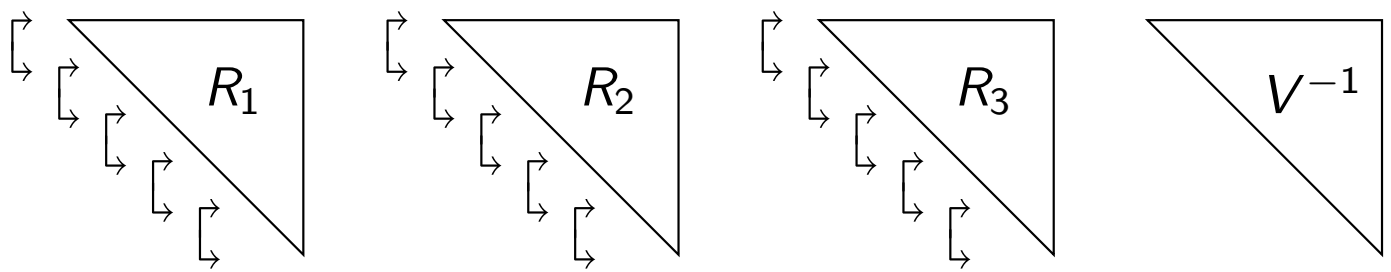
$$WV^{-1} = W_1 \dots W_m V^{-1} = \mathcal{Q}R_1 \dots \mathcal{Q}R_m V^{-1}$$

Hessenberg-Triangular reduction

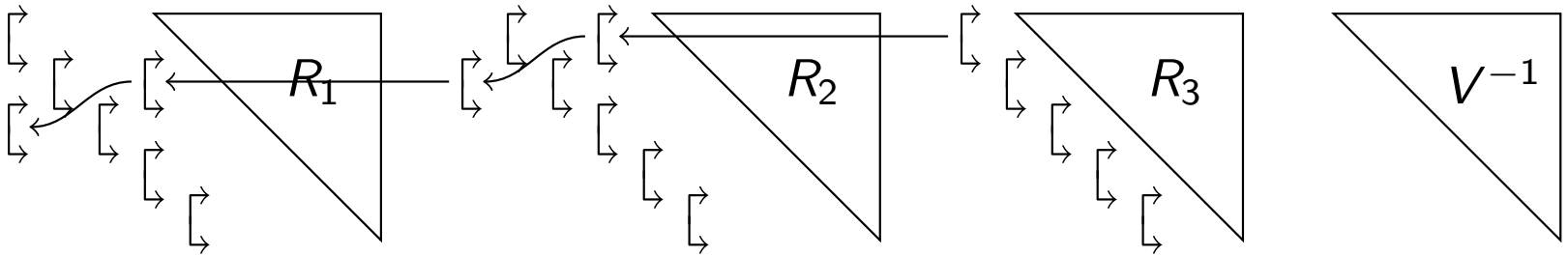
The QZ and the Hessenberg-Triangular reduction are (formally) equivalent to the Hessenberg reduction and QR on WV^{-1}.

We can write our factorization in a structured form:

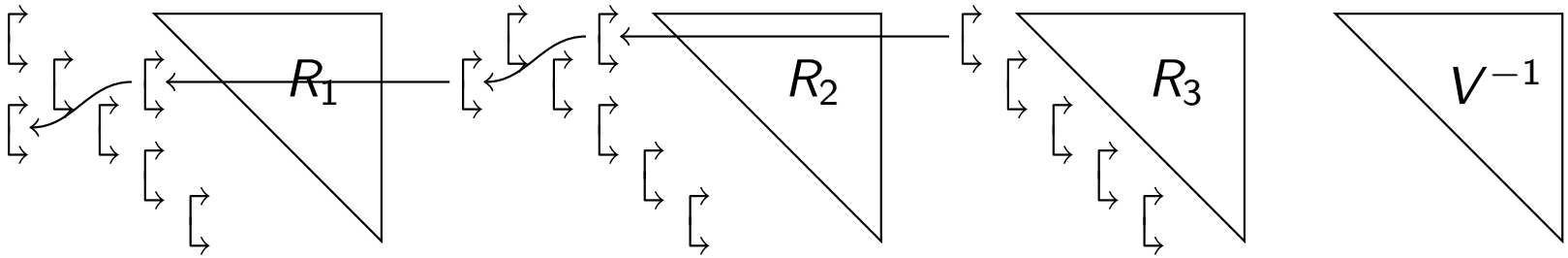
$$WV^{-1} = W_1 \dots W_m V^{-1} = \mathcal{Q} R_1 \dots \mathcal{Q} R_m V^{-1}$$

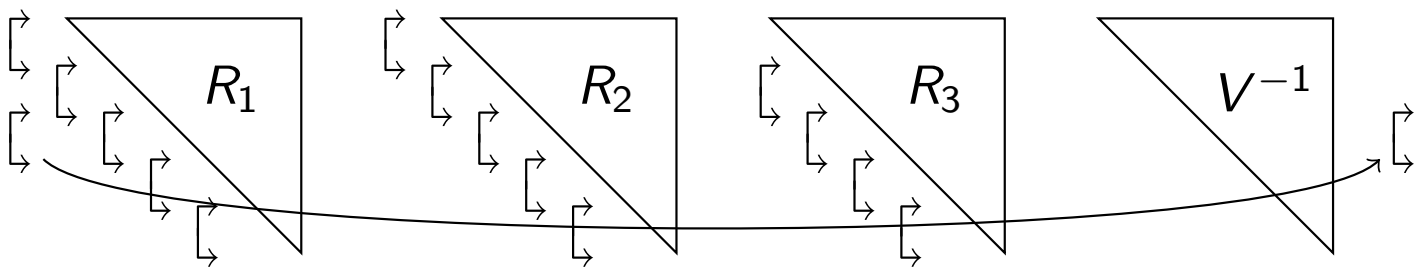


Hessenberg-Triangular reduction

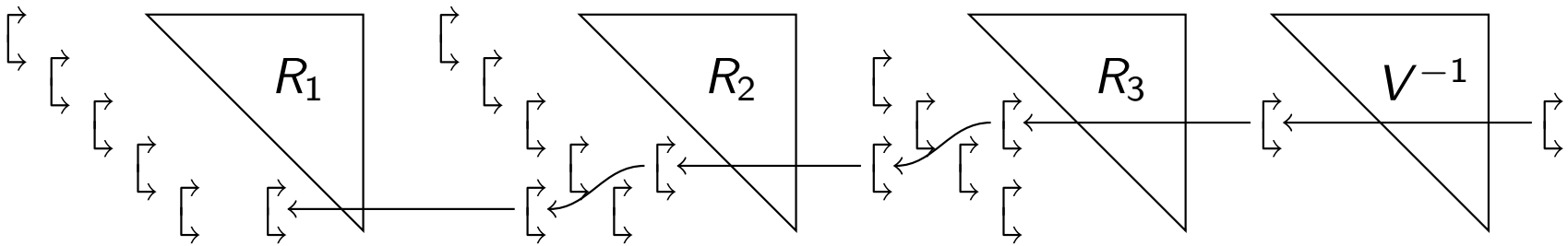


Hessenberg-Triangular reduction

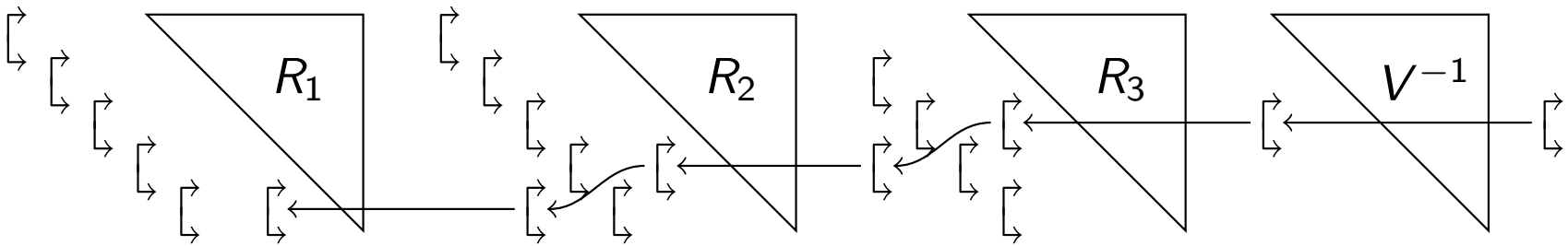


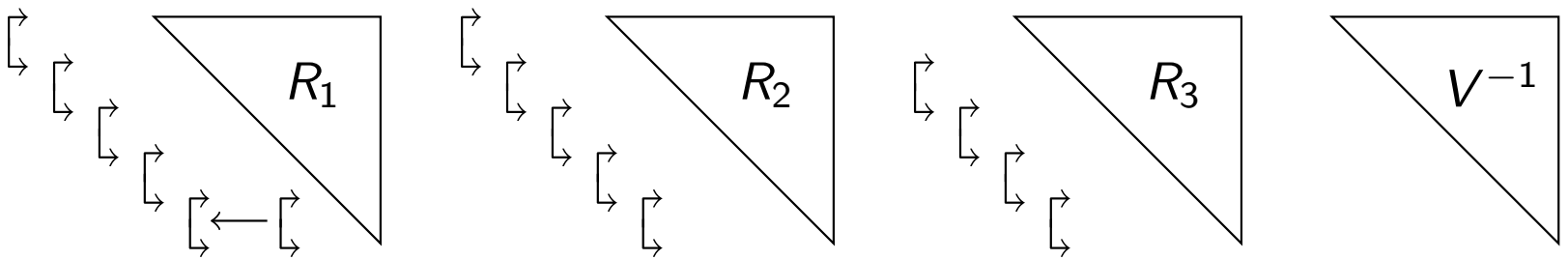


Hessenberg-Triangular reduction

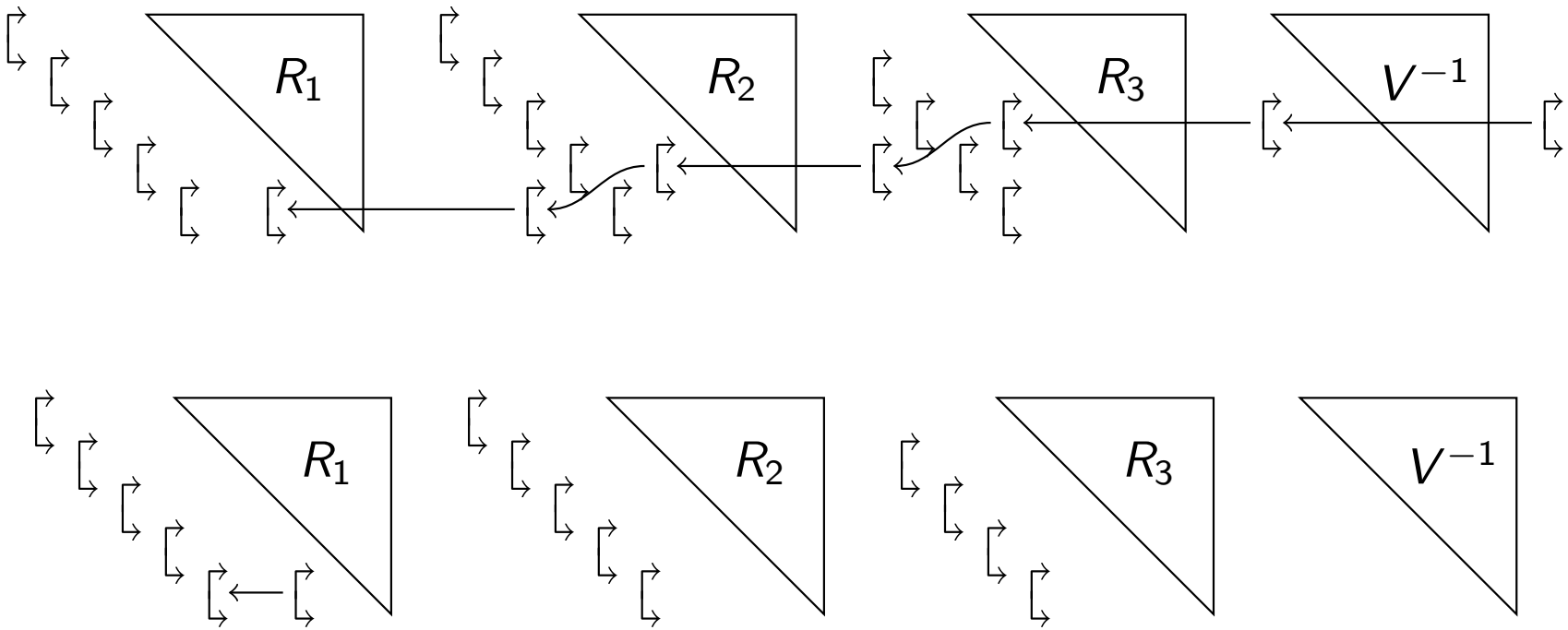


Hessenberg-Triangular reduction





Hessenberg-Triangular reduction



$\mathcal{O}(md)$ turnovers per core transformation and $\mathcal{O}(m^2d)$ transformations to annihilate $\implies$ $\mathcal{O}(m^3d^2)$ flops.

Chasing core transformations

The same ideas can be used to chase core transformations until $\mathcal{Q} \approx I$. We put a $\tilde{}$ on the matrices and transformations obtained after HT-reduction.

Chasing core transformations

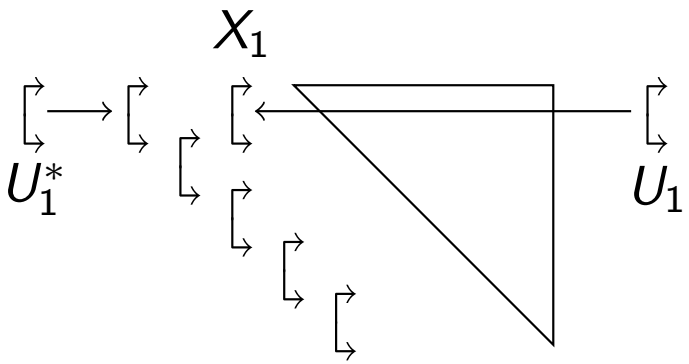
The same ideas can be used to chase core transformations until $\mathcal{Q} \approx I$. We put a $\tilde{}$ on the matrices and transformations obtained after HT-reduction.

$$\tilde{W}\tilde{V}^{-1} = \underbrace{\quad}_{\tilde{\mathcal{Q}}} \underbrace{\quad}_{\tilde{R}_1\tilde{R}_2\tilde{R}_3\tilde{V}_3^{-1}\tilde{V}_2^{-1}\tilde{V}_1^{-1}} .$$

Chasing core transformations

The same ideas can be used to chase core transformations until $\mathcal{Q} \approx I$. We put a $\tilde{}$ on the matrices and transformations obtained after HT-reduction.

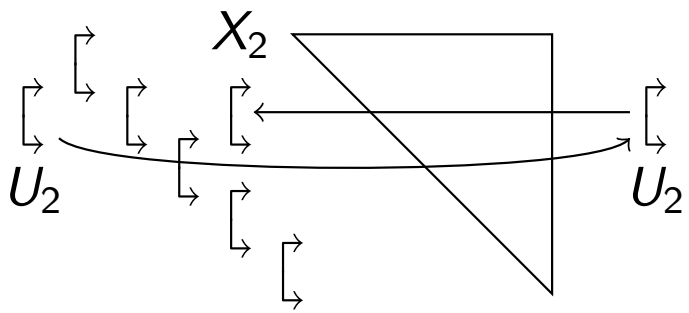
We compute the first transformation U_1:



Chasing core transformations

The same ideas can be used to chase core transformations until $\mathcal{Q} \approx I$. We put a $\tilde{}$ on the matrices and transformations obtained after HT-reduction.

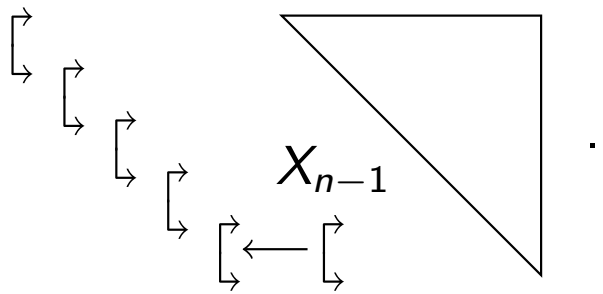
and then we chase it by executing a turnover and similarity:



Chasing core transformations

The same ideas can be used to chase core transformations until $\mathcal{Q} \approx I$. We put a $\tilde{}$ on the matrices and transformations obtained after HT-reduction.

until we fuse it at the bottom:



.

A quick flop count

A QZ sweep can be completed moving a core transformation down of $\mathcal{O}(md)$ rows.

A quick flop count

A QZ sweep can be completed moving a core transformation down of $\mathcal{O}(md)$ rows.

Each "movement" costs $\mathcal{O}(m)$ passthroughs and 1 turnover, thus a QZ sweep amounts to $O(m^2 d)$ flops.

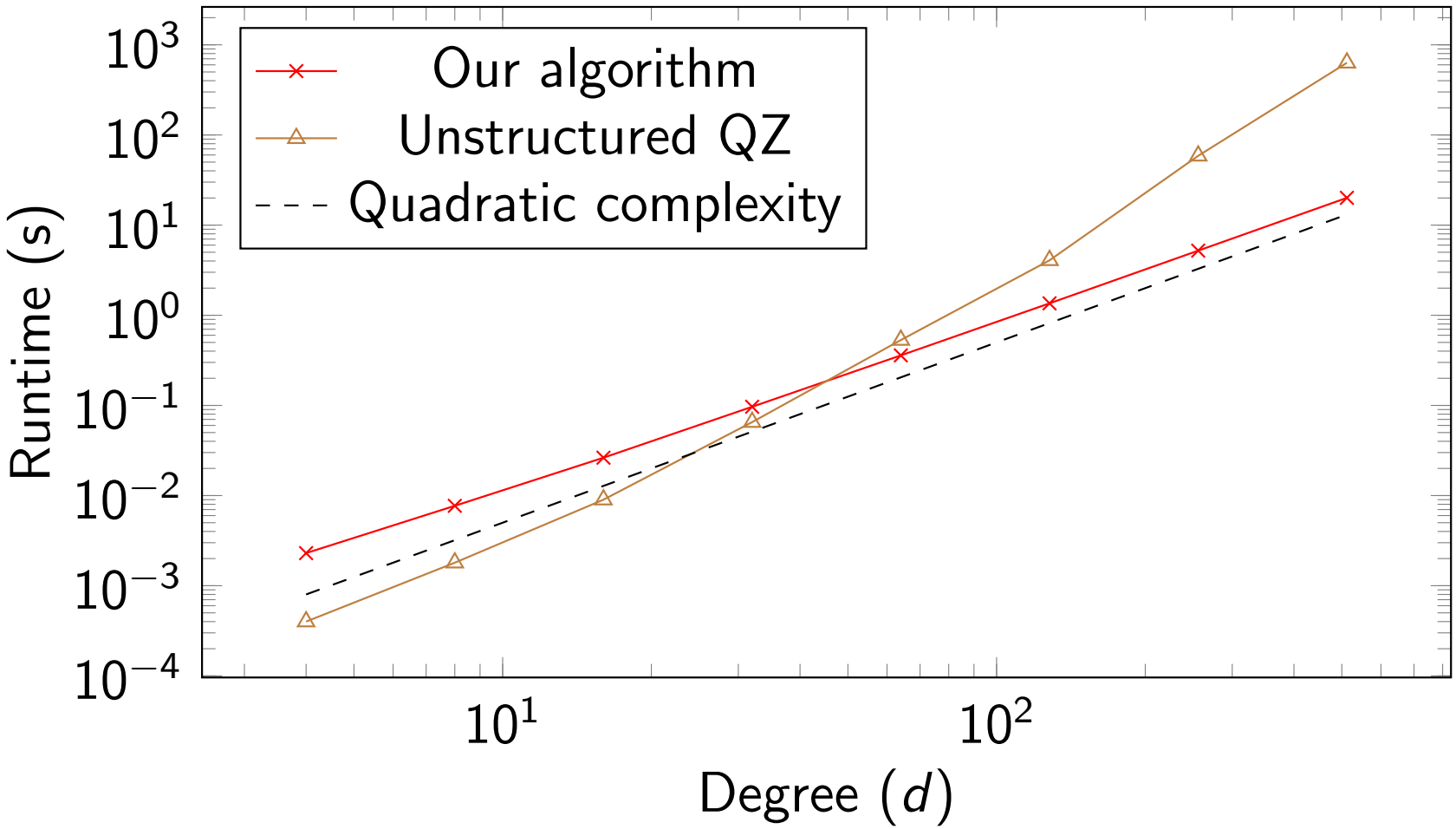
A quick flop count

A QZ sweep can be completed moving a core transformation down of $\mathcal{O}(md)$ rows.

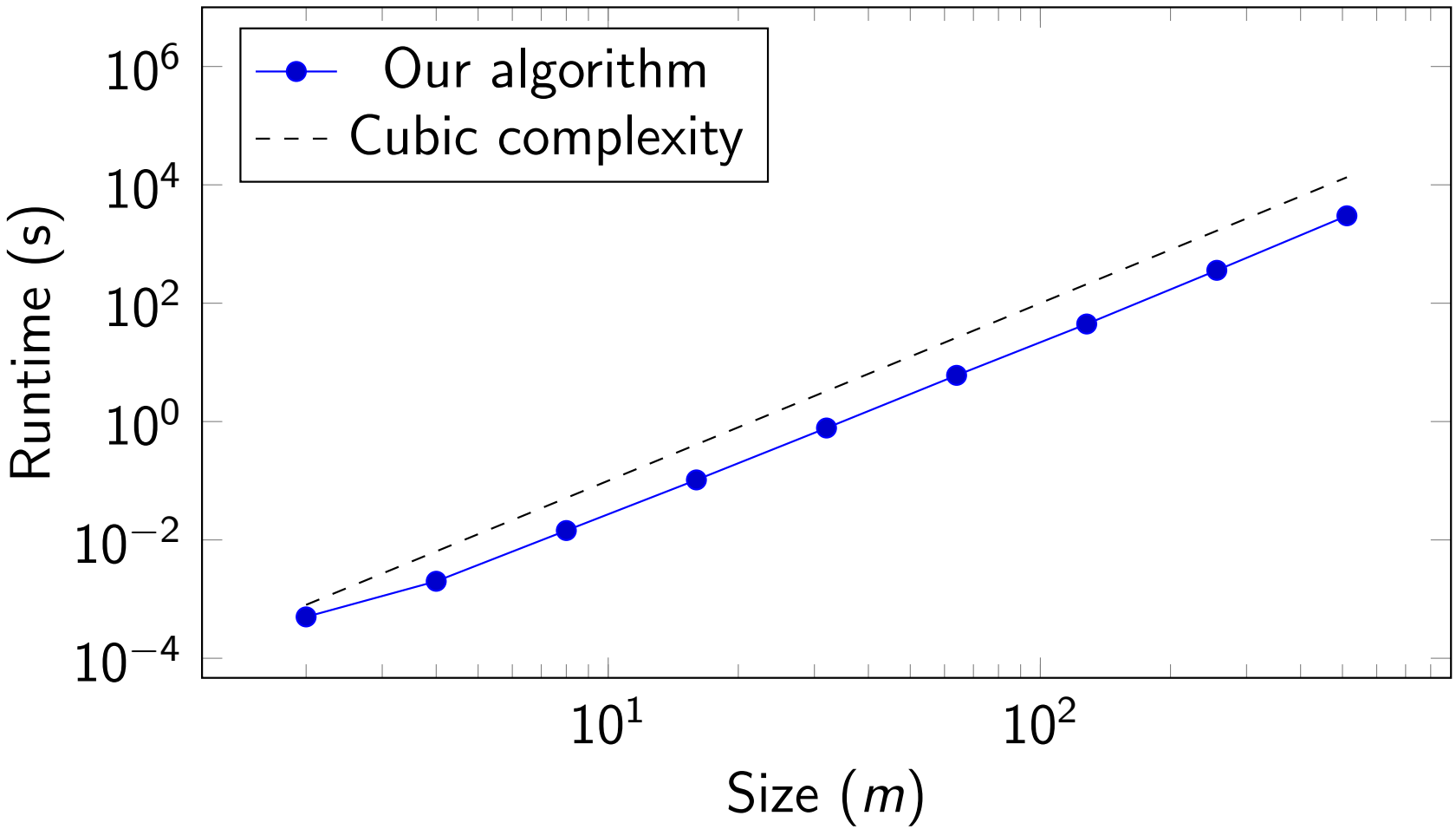
Each "movement" costs $\mathcal{O}(m)$ passthroughs and 1 turnover, thus a QZ sweep amounts to $O(m^2 d)$ flops.

Assuming $\mathcal{O}(md)$ steps are required to reach convergence, the cost is $\mathcal{O}(m^3 d^2)$.

Complexity in d (4 x 4 matrix polynomial)



Complexity in m ($d = 10$)



Backward stability

- Each passthrough and turnover introduces a small backward error of the order of the unit roundoff u on each of the unitary factors.

Backward stability

- Each passthrough and turnover introduces a small backward error of the order of the unit roundoff u on each of the unitary factors.
- We can prove that the rank 1 part (of each factor) can be recovered from the unitary one.

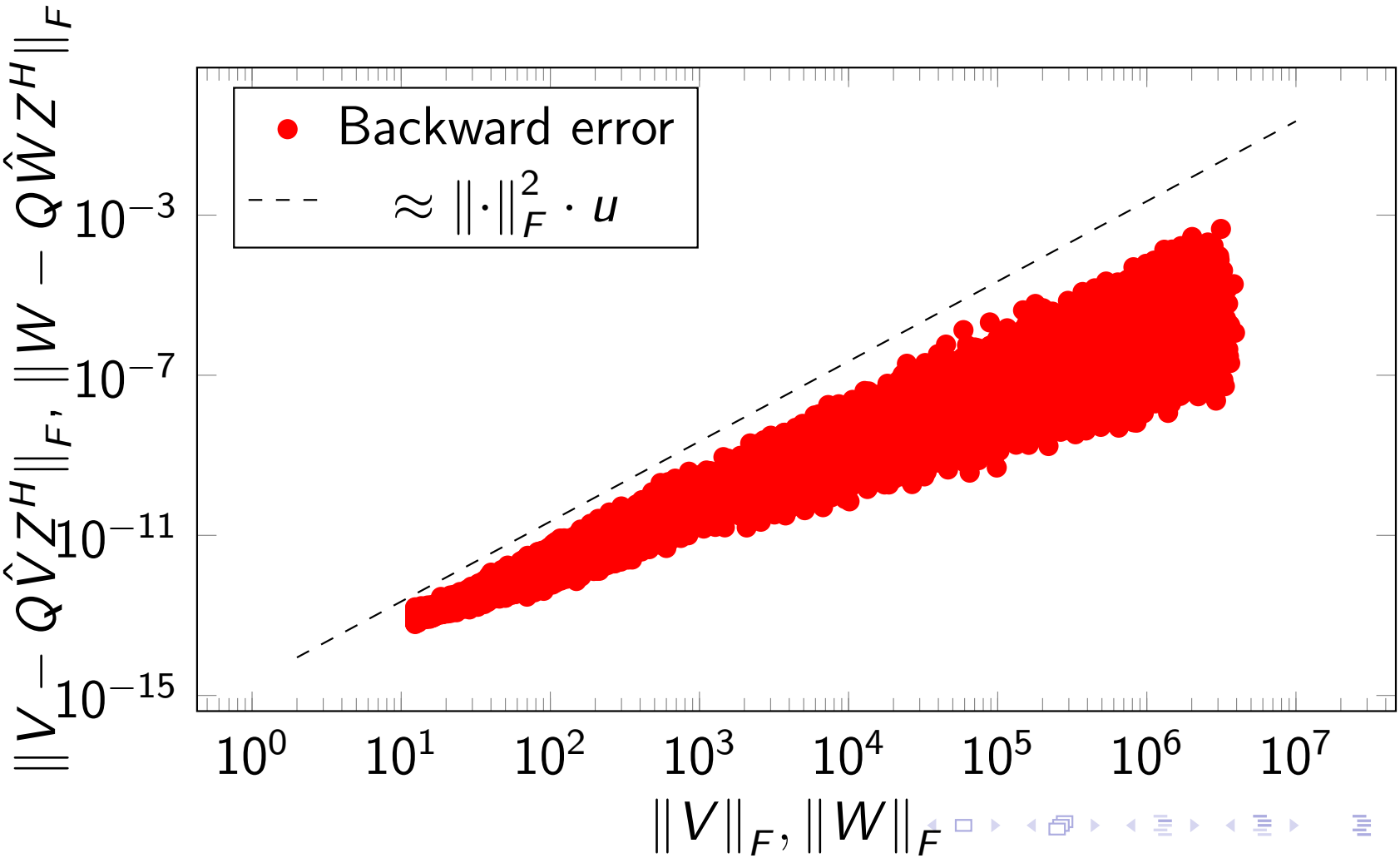
Backward stability

- Each passthrough and turnover introduces a small backward error of the order of the unit roundoff u on each of the unitary factors.
- We can prove that the rank 1 part (of each factor) can be recovered from the unitary one.
- $\mathcal{O}(u)$ backward error on the unitary part introduces a $\mathcal{O}(\|y_j\|^2 u)$ backward error on the rank 1 part.

Backward stability

- Each passthrough and turnover introduces a small backward error of the order of the unit roundoff u on each of the unitary factors.
- We can prove that the rank 1 part (of each factor) can be recovered from the unitary one.
- $\mathcal{O}(u)$ backward error on the unitary part introduces a $\mathcal{O}(\|y_j\|^2 u)$ backward error on the rank 1 part.
- Total backward error of the Schur form (on the original matrix pencil): $\mathcal{O}(\|V\|^2 u)$ and $\mathcal{O}(\|W\|)^2 u)$.

Backward stability



Conclusions

- The algorithm has the "optimal" complexity $\mathcal{O}(m^3 d^2)$.

Conclusions

- The algorithm has the "optimal" complexity $\mathcal{O}(m^3 d^2)$.
- It is fast in practice even for small degrees.

Conclusions

- The algorithm has the "optimal" complexity $\mathcal{O}(m^3 d^2)$.
- It is fast in practice even for small degrees.
- It is proven backward stable.

Conclusions

- The algorithm has the "optimal" complexity $\mathcal{O}(m^3 d^2)$.
- It is fast in practice even for small degrees.
- It is proven backward stable.

Thank you for your attention!