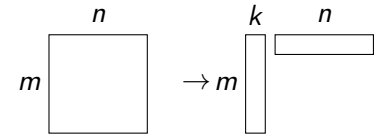
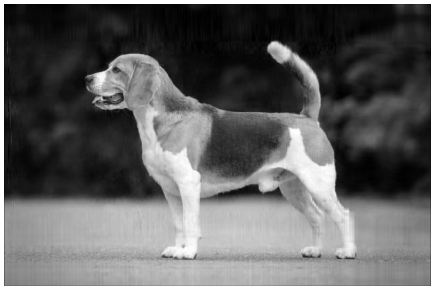
Harnessing inexactness in scientific computing

Lecture 9: low-rank approximations

Theo Mary (CNRS)
theo.mary@lip6.fr
https://perso.lip6.fr/Theo.Mary/

Elisa Riccietti (ENS Lyon)
elisa.riccietti@ens-lyon.fr
https://perso.ens-lyon.fr/elisa.riccietti/





M2 course at ENS Lyon, 2024–2025
Slides available on course webpage

Introduction

QR decomposition methods

Randomization

Mixed precision

Gram-based QR/LRA

Final project

Rank and numerical rank

In the following, B is a dense matrix of size $m \times n$ with $m \geq n$.

Definition (Rank)

The rank k of B is defined as the smallest integer such that there exist matrices X and Y of size $m \times k$ and $n \times k$ such that

$$B = XY^T.$$

Definition (Numerical rank)

The numerical rank k_ε of B at accuracy ε is defined as the smallest integer such that there exists a matrix $\widetilde{B}$ of rank k_ε such that

$$\|B - \widetilde{B}\| \leq \varepsilon.$$

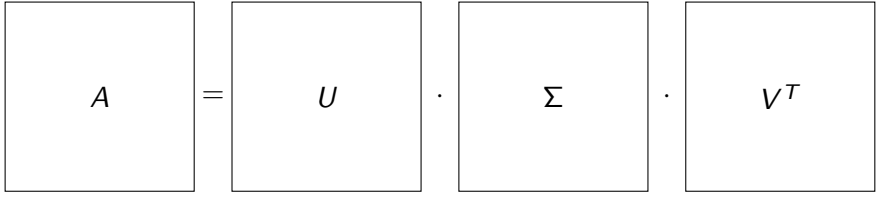
Theorem (Eckart-Young)

Let $U\Sigma V^T$ be the SVD decomposition of B and let us note $\sigma_i = \Sigma_{i,i}$ its singular values. Then $\widetilde{B} = U_{1:m,1:k}\Sigma_{1:k,1:k}V^T_{1:n,1:k}$ is the optimal rank-k approximation of B and

$$\|B - \widetilde{B}\|_2 = \sigma_{k+1}.$$

Remark: in $\|\cdot\|_2$,

$$k_\varepsilon = \min_{1 \leq k \leq \min(m,n)} \sigma_{k+1} \leq \varepsilon.$$



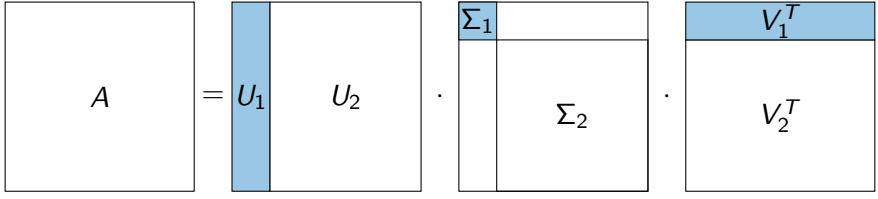
Theorem (Eckart-Young)

Let $U\Sigma V^T$ be the SVD decomposition of B and let us note $\sigma_i = \Sigma_{i,i}$ its singular values. Then $\widetilde{B} = U_{1:m,1:k}\Sigma_{1:k,1:k}V^T_{1:n,1:k}$ is the optimal rank-k approximation of B and

$$\|B - \widetilde{B}\|_2 = \sigma_{k+1}.$$

Remark: in $\|\cdot\|_2$,

$$k_\varepsilon = \min_{1 \leq k \leq \min(m,n)} \sigma_{k+1} \leq \varepsilon.$$



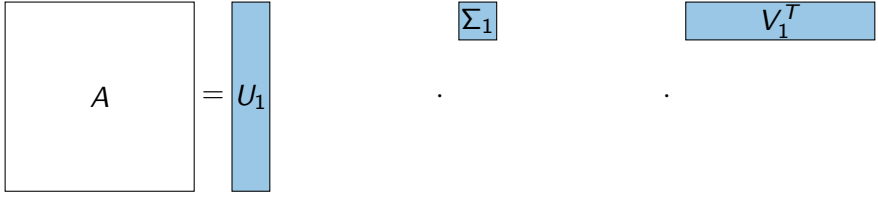
Theorem (Eckart-Young)

Let $U\Sigma V^T$ be the SVD decomposition of B and let us note $\sigma_i = \Sigma_{i,i}$ its singular values. Then $\widetilde{B} = U_{1:m,1:k}\Sigma_{1:k,1:k}V^T_{1:n,1:k}$ is the optimal rank-k approximation of B and

$$\|B - \widetilde{B}\|_2 = \sigma_{k+1}.$$

Remark: in $\|\cdot\|_2$,

$$k_\varepsilon = \min_{1 \leq k \leq \min(m,n)} \sigma_{k+1} \leq \varepsilon.$$



If $k_\varepsilon = \min(m, n)$ then B is said to be full-rank, otherwise it is rank-deficient. A class of rank-deficient matrices of particular interest are low-rank matrices.

Definition (Low-rank matrix)

B is said to be low-rank (for a given accuracy ε) if its numerical rank k_ε is small enough such that its rank-k_ε approximation $\widetilde{B} = XY^T$ requires less storage than the full-rank matrix B, i.e., if

$$k_\varepsilon(m + n) \leq mn.$$

In that case, $\widetilde{B}$ is said to be a low-rank approximation of B and ε is called the low-rank threshold.

In the following, for the sake of simplicity, we refer to the numerical rank of a matrix at accuracy ε simply as its "rank".

Computing low-rank approximations

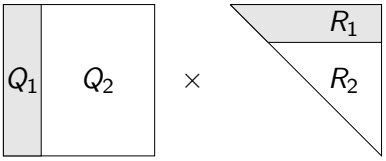
How to compute a low-rank approximation?

- Optimal truncated SVD costs $O(mn^2)$ flops, with a big constant in the $O()$ $\Rightarrow$ too expensive for large matrices.

Computing low-rank approximations

How to compute a low-rank approximation?

- Optimal truncated SVD costs $O(mn^2)$ flops, with a big constant in the $O()$ $\Rightarrow$ too expensive for large matrices.
- Can rely instead on QR factorization.

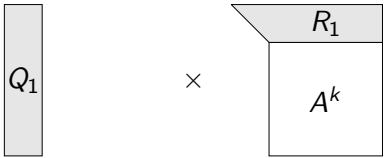


$$\|A - Q_1 R_1\| \leq \|R_2\|$$

Computing low-rank approximations

How to compute a low-rank approximation?

- Optimal truncated SVD costs $O(mn^2)$ flops, with a big constant in the $O()$ $\Rightarrow$ too expensive for large matrices.
- Can rely instead on QR factorization. The factorization need not be computed entirely but can actually be truncated:

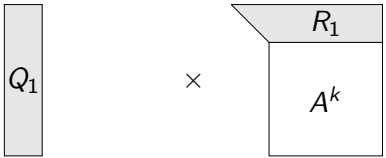


$$\|A - Q_1 R_1\| \leq \|A^k\|$$

Because $\|A^k\|$ is monotonically decreasing for $k = 1, \ldots, n$, the factorization can be interrupted as soon as $\|A^k\| \leq \varepsilon$.

How to compute a low-rank approximation?

- Optimal truncated SVD costs $O(mn^2)$ flops, with a big constant in the $O()$ $\Rightarrow$ too expensive for large matrices.
- Can rely instead on QR factorization. The factorization need not be computed entirely but can actually be truncated:



$$\|A - Q_1 R_1\| \leq \|A^k\|$$

Because $\|A^k\|$ is monotonically decreasing for $k = 1, \ldots, n$, the factorization can be interrupted as soon as $\|A^k\| \leq \varepsilon$.

- However, is the computed k close to the numerical rank k_ε, i.e., are such methods rank-revealing ?

Introduction

QR decomposition methods

Randomization

Mixed precision

Gram-based QR/LRA

Final project

Gram–Schmidt

- The Gram–Schmidt computes an orthonormal basis Q of the columns of a matrix $A \in \mathbb{R}^{m \times n}$

Classical (CGS)

for $j = 1\colon n$ **do**
  $q_j = a_j$
  for $i = 1\colon j-1$ **do**
    $r_{ij} = q_i^T a_j$
    $q_j = q_j - r_{ij} q_i$
  end for
  $r_{jj} = \|q_j\|_2$
  $q_j = q_j / r_{jj}$
end for

Modified (MGS)

$Q = A$
for $j = 1\colon n$ **do**
  $r_{jj} = \|q_j\|_2$
  $q_j = q_j / r_{jj}$
  for $i = j+1\colon n$ **do**
    $r_{ji} = q_i^T q_j$
    $q_i = q_i - r_{ji} q_j$
  end for
end for

- However, unstable in finite precision: $\|I - Q^T Q\|$ is of order $\kappa(A)^2 u$ for CGS and $\kappa(A) u$ for MGS
- Flop count $\sim \sum_{j=1}^{n} 4m(n-j) \sim 2mn^2$
- **MATLAB demo**

Householder transformation

- Householder transformations are defined as $H = I - 2vv^T$ where $\|v\|_2 = 1$
- Given a column vector x, we can use a Householder transformation to zero out all its coefficients except the first, i.e., transform it into a multiple of e_1
- Indeed, define $w = x - s\|x\|_2 e_1$ where $s = \pm 1$ and $v = w/\|w\|$, then $Hx = s\|x\|e_1$
- $s = \pm 1$ chosen to avoid cancellation

Householder QR

$R = A$
$Q = I$
for $j = 1 \colon n$ **do**
  $\alpha = \|A(j \colon m, j)\|_2$
  $s = -\operatorname{sign}(A(j,j))$
  $w = A(j \colon m, j) - s\alpha e_1$
  $v = w/\|w\|_2$
  $H = I - 2vv^T$
  $R(j \colon m, j \colon n) = H \cdot R(j \colon m, j \colon n)$
  $Q(1 \colon m, j \colon n) = Q(1 \colon m, j \colon n) \cdot H$
end for

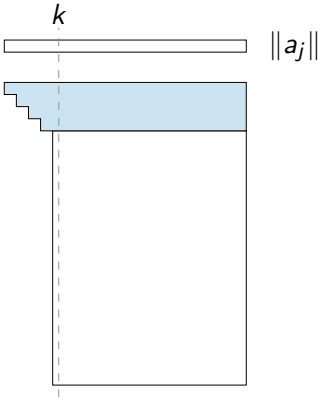
- Householder QR successively applies Householder transformations to zero out all subdiagonal coefficients of A
- We obtain $R = H_n \cdots H_2 \cdot H_1 \cdot A$ and $Q = I \cdot H_1 \cdot H_2 \cdots H_n$. Alternatively, need not form Q which can be implicitly represented by storing the H_j transforms

- Numerically stable: $\|I - Q^T Q\| = O(u)$ regardless of $\kappa(A)$
- Flop count $\sim \sum_{j=1}^{n} 4(m-j)(n-j) \sim 2mn^2 - 2n^3/3$ without forming Q, extra $2mn^2$ for forming Q
- **MATLAB demo**

Truncated Householder QR

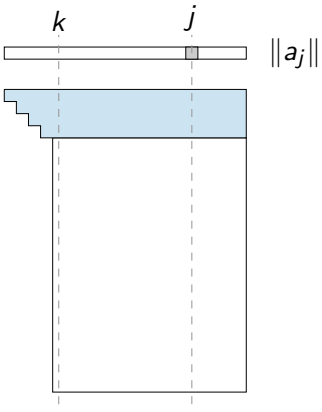
- Householder QR factorization can be stopped after k steps, yielding truncated (rank-k) factors
- Assuming $k \ll n$, flop count becomes $\sim 4mnk$
- How to ensure that k is close to the numerical rank ? $\Rightarrow$ use column pivoting to choose the best column at each step

Householder QR with column pivoting (QRCP)



At step $k = 1, \ldots, n$

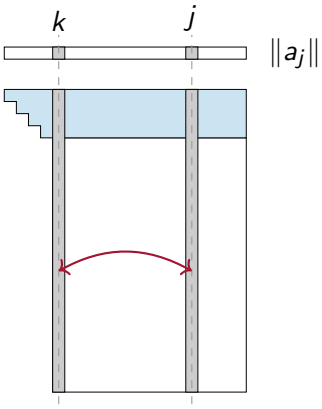
Householder QR with column pivoting (QRCP)



At step $k = 1, \ldots, n$

1. select column j of largest norm

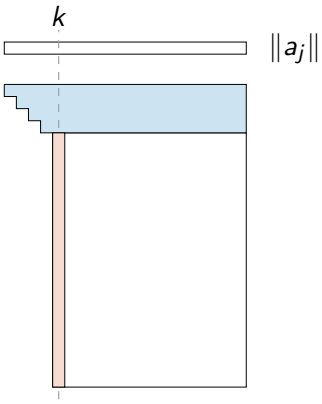
Householder QR with column pivoting (QRCP)



At step $k = 1, \ldots, n$

1. select column j of largest norm
2. permute columns k and j

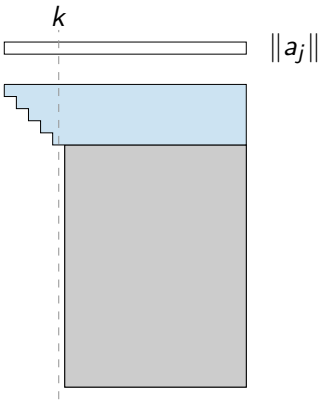
Householder QR with column pivoting (QRCP)



At step $k = 1, \ldots, n$

1. select column j of largest norm
2. permute columns k and j
3. reduce column k via Householder transform

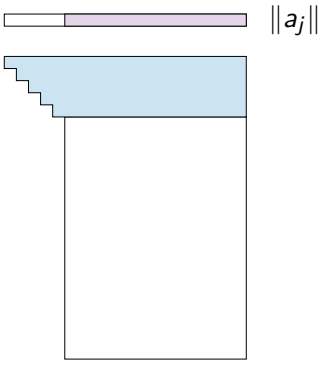
Householder QR with column pivoting (QRCP)



At step $k = 1, \ldots, n$

1. select column j of largest norm
2. permute columns k and j
3. reduce column k via Householder transform
4. update trailing submatrix (at least row k)

Householder QR with column pivoting (QRCP)

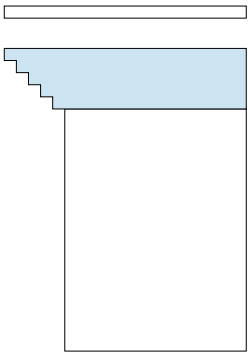


At step $k = 1, \ldots, n$

1. select column j of largest norm
2. permute columns k and j
3. reduce column k via Householder transform
4. update trailing submatrix (at least row k)
5. update column norms

How? At step k, we remove row k, hence
$\|a_j^{(k+1)}\|_2^2 = \|a_j^{(k)}\|_2^2 - a_{kj}^2$
Risk of heavy cancellation in finite precision!

Householder QR with column pivoting (QRCP)



At step $k = 1, \ldots, n$

1. select column j of largest norm
2. permute columns k and j
3. reduce column k via Householder transform
4. update trailing submatrix (at least row k)
5. update column norms

   How? At step k, we remove row k, hence
   $\|a_j^{(k+1)}\|_2^2 = \|a_j^{(k)}\|_2^2 - a_{kj}^2$
   Risk of heavy cancellation in finite precision!

- This is a BLAS-2 algorithm. Partial block version exists but still poor computational efficiency and parallelization.
- **MATLAB demo**

- Let $A \in \mathbb{R}^{m \times n}$ and let $\Omega \in \mathbb{R}^{n \times \ell}$ be a random Gaussian matrix ($\omega_{ij} \sim \mathcal{N}(0,1)$)
- Let $B = A\Omega$. A and B have the same range. Moreover, since the columns of Ω are independent, it is likely that the columns of B are also linearly independent.
- Let $Q = \mathtt{qr}(B)$ and let $\ell = k + p$. Then we have

$$\mathbb{E}(\|A - QQ^T A\|_F) \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{i>k} \sigma_i^2\right)^{1/2}$$

$\Rightarrow$ if $p \geq k$ ($\ell \geq 2k$), the rank-ℓ approximation $A \approx Q(Q^T A)$ is nearly as good as the best rank-k approximation. Moreover, taking small p (e.g., $\ell = k + 5$) is sufficient to get a good approximation up to constants.

From the rank-ℓ approximation $A \approx Q(Q^T A)$, a rank-k approximation XY^T can be efficiently recovered with the following algorithm.

Halko, Martinsson, Tropp (2011)

Input: $A \in \mathbb{R}^{m \times n}$, rank k, oversampling p
Output: $X \in \mathbb{R}^{m \times k}$, $Y \in \mathbb{R}^{n \times k}$ such that $A \approx XY^T$

$\Omega \leftarrow \texttt{randn}(n, k+p)$
$B \leftarrow A\Omega$
$Q \leftarrow \texttt{qr}(B)$
$C \leftarrow A^T Q$
$ZY^T \leftarrow \texttt{LRA}(C, k)$
$X \leftarrow QZ$

- Cost:
  - Matrix products: $4mn\ell$ flops
  - QR: $O(m\ell^2)$ flops
  - LRA: $O(n\ell k)$ flops

- Matrix multiplication is the bottleneck
  $\Rightarrow$ BLAS-3, very efficient!
- Can be applied to A^T to interchange m and n
- Can use a structured Ω (sparse, FFT, ...) to reduce the flop count of the matmul
- **MATLAB demo**

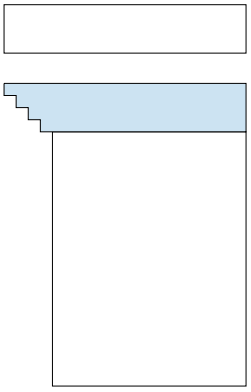
Fixed-accuracy randomized LRA

- In practice k is often not known: we'd rather choose a target accuracy ε and let the algorithm find k_ε. This can be accomplished by adapting the algorithm as follows.
  Martinsson and Voronin (2016)
- The choice of the block size b presents a tradeoff between efficiency and risk of "overshooting" the rank
- Here, Q is orthonormalized with block MGS
- Flop count $\sim 6mnq$, where $q \approx b \times \lceil k/b \rceil$. Increased constant (6 instead of 4) due to the need to downdate A to keep track of the error norm.
- **MATLAB demo**

Input: $A \in \mathbb{R}^{m \times n}$, tolerance ε, block size b
Output: $X \in \mathbb{R}^{m \times k}$, $Y \in \mathbb{R}^{n \times k}$ such that $A \approx XY^T$

Initialize Q and B to empty matrices.
repeat
$\Omega \leftarrow \texttt{randn}(n, b)$
$Y = A\Omega$
$Q_b = \texttt{qr}(Y - Q(Q^T Y))$
$B_b = Q_b^T A$
$Q \leftarrow [Q \ Q_b]$
$B \leftarrow \begin{bmatrix} B \\ B_b \end{bmatrix}$
$A \leftarrow A - Q_b B_b$
until $\|A\| \leq \varepsilon$
$ZY^T = \texttt{truncSVD}(B, \varepsilon)$
$X = QZ$

QR with randomized pivoting (QRRP)

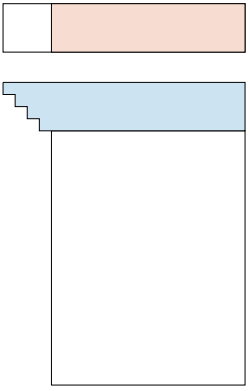


Alternatively, randomization can be used to efficiently select the pivots in Householder QR.

Martinsson et al. (2017) Duersch and Gu (2017)

Compute a sample $S = \Omega A$ using a random matrix Ω. At step $k = 1 : b : n$

QR with randomized pivoting (QRRP)



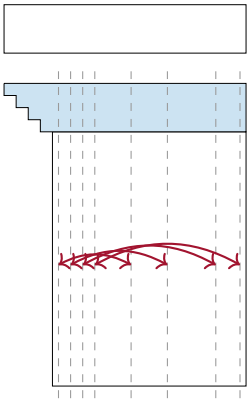
Alternatively, randomization can be used to efficiently select the pivots in Householder QR.

Martinsson et al. (2017) Duersch and Gu (2017)

Compute a sample $S = \Omega A$ using a random matrix Ω. At step $k = 1 : b : n$

1. compute QR of S with B-G pivoting to select the "best" b columns

QR with randomized pivoting (QRRP)



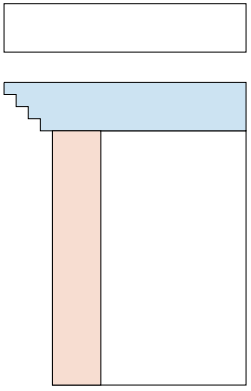
Alternatively, randomization can be used to efficiently select the pivots in Householder QR.

Martinsson et al. (2017) Duersch and Gu (2017)

Compute a sample $S = \Omega A$ using a random matrix Ω. At step $k = 1 : b : n$

1. compute QR of S with B-G pivoting to select the "best" b columns
2. permute the selected columns upfront

QR with randomized pivoting (QRRP)



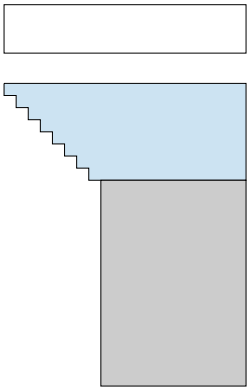
Alternatively, randomization can be used to efficiently select the pivots in Householder QR.

Martinsson et al. (2017) Duersch and Gu (2017)

Compute a sample $S = \Omega A$ using a random matrix Ω. At step $k = 1 : b : n$

1. compute QR of S with B-G pivoting to select the "best" b columns
2. permute the selected columns upfront
3. reduce b columns via Householder transform

QR with randomized pivoting (QRRP)



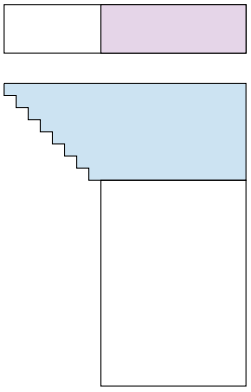
Alternatively, randomization can be used to efficiently select the pivots in Householder QR.

Martinsson et al. (2017) Duersch and Gu (2017)

Compute a sample $S = \Omega A$ using a random matrix Ω. At step $k = 1 : b : n$

1. compute QR of S with B-G pivoting to select the "best" b columns
2. permute the selected columns upfront
3. reduce b columns via Householder transform
4. update trailing submatrix

QR with randomized pivoting (QRRP)



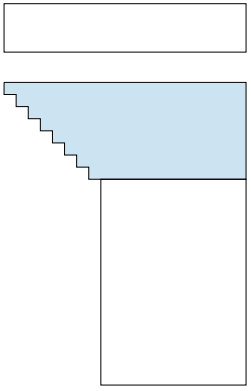
Alternatively, randomization can be used to efficiently select the pivots in Householder QR.

Martinsson et al. (2017) Duersch and Gu (2017)

Compute a sample $S = \Omega A$ using a random matrix Ω. At step $k = 1 : b : n$

1. compute QR of S with B-G pivoting to select the "best" b columns
2. permute the selected columns upfront
3. reduce b columns via Householder transform
4. update trailing submatrix
5. update S

QR with randomized pivoting (QRRP)



Alternatively, randomization can be used to efficiently select the pivots in Householder QR.

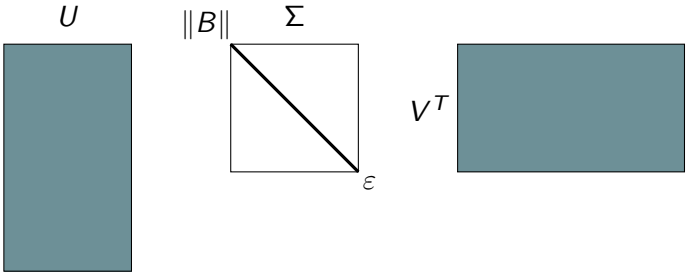
Martinsson et al. (2017) Duersch and Gu (2017)

Compute a sample $S = \Omega A$ using a random matrix Ω. At step $k = 1 : b : n$

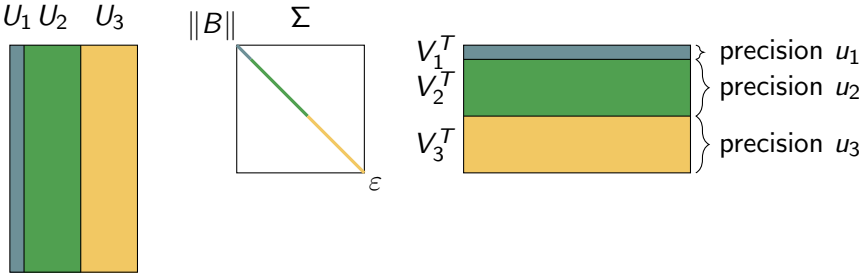
1. compute QR of S with B-G pivoting to select the "best" b columns
2. permute the selected columns upfront
3. reduce b columns via Householder transform
4. update trailing submatrix
5. update S

- High efficiency and parallelization
- Norm of the trailing submatrix is indirectly available through the sample: $\|s_j\| = \sqrt{b}\|a_j\|$ works well in practice

Adaptive precision low rank compression



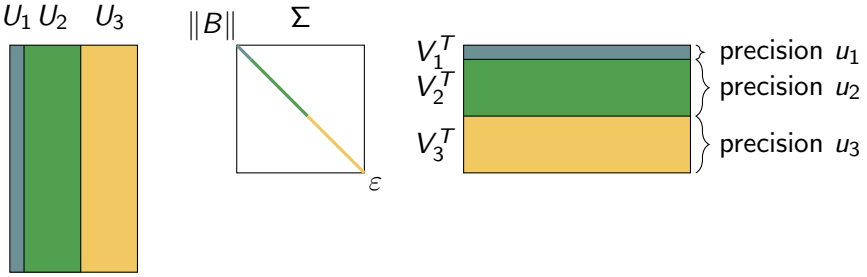
Adaptive precision low rank compression



- **Adaptive precision compression:** partition U and V into q groups of decreasing precisions $u_1 \leq \varepsilon < u_2 < \ldots < u_q$

  Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M. (2022)

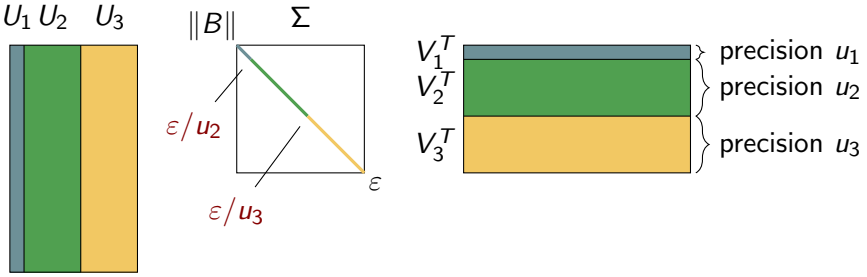
Adaptive precision low rank compression



- **Adaptive precision compression:** partition U and V into q groups of decreasing precisions $u_1 \leq \varepsilon < u_2 < \ldots < u_q$

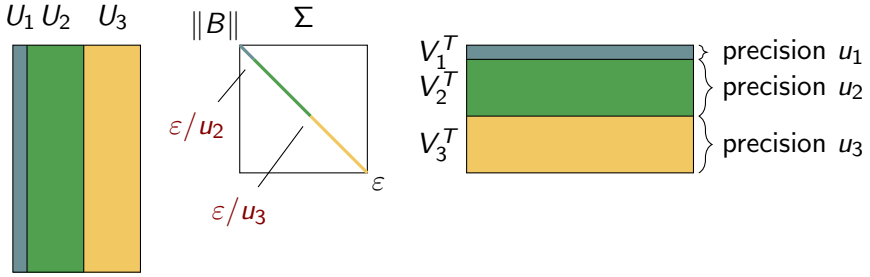
  Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M. (2022)
- Why does it work? $B = \mathbf{B_1} + \mathbf{B_2} + \mathbf{B_3}$ with $|B_i| \leq O(\|\Sigma_i\|)$

Adaptive precision low rank compression



- **Adaptive precision compression:** partition U and V into q groups of decreasing precisions $u_1 \leq \varepsilon < u_2 < \ldots < u_q$
  Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M. (2022)
- Why does it work? $B = \mathbf{B_1} + \mathbf{B_2} + \mathbf{B_3}$ with $|B_i| \leq O(\|\Sigma_i\|)$
- With p precisions and a partitioning such that $\|\Sigma_k\| \leq \varepsilon \|B\|/u_k$, $\|B - \widehat{U}_\varepsilon \Sigma_\varepsilon \widehat{V}_\varepsilon\| \lesssim (2p-1)\varepsilon\|B\|$

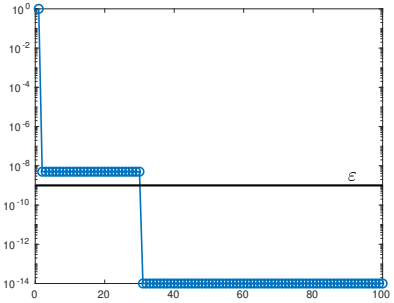
Adaptive precision low rank compression



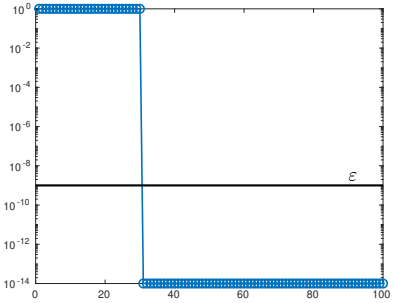
- **Adaptive precision compression:** partition U and V into q groups of decreasing precisions $u_1 \leq \varepsilon < u_2 < \ldots < u_q$
  Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M. (2022)
- Why does it work? $B = \mathbf{B_1} + \mathbf{B_2} + \mathbf{B_3}$ with $|B_i| \leq O(\|\Sigma_i\|)$
- With p precisions and a partitioning such that $\|\Sigma_k\| \leq \varepsilon \|B\|/u_k$, $\|B - \widehat{U}_\varepsilon \Sigma_\varepsilon \widehat{V}_\varepsilon\| \lesssim (2p-1)\varepsilon\|B\|$
- Applicable to any LRA decomposition with rank-1 components of decaying norm (in particular, QRCP)

Examples of spectrum

Both matrices have ε-rank 30 (with $\varepsilon = 10^{-9}$) but present very different potential for mixed precision

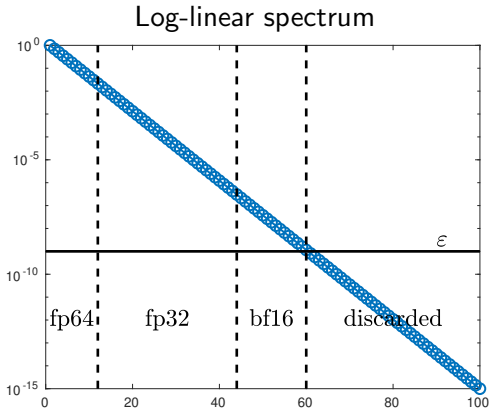


Large gain
(almost all in lower precision)



No gain
(all in higher precision)

Examples of spectrum



Householder QR in finite precision

Let's assume a sequence of Householder transforms H^i $i = 1, \ldots, k$ computed and applied in precision u to a vector $b = b^0$

$$\widehat{b^1} = H^1(b^0 + \Delta b^0), \qquad \|\Delta b^0\| \leq mu\|b^0\|$$

Householder QR in finite precision

Let's assume a sequence of Householder transforms H^i $i = 1, \ldots, k$ computed and applied in precision u to a vector $b = b^0$

$$
\begin{array}{lcll}
\widehat{b^1} & = & H^1(b^0 + \Delta b^0), & \|\Delta b^0\| \leq mu\|b^0\| \\
& \cdots & & \\
\widehat{b^i} & = & H^i(\widehat{b^{i-1}} + \Delta b^{i-1}), & \|\Delta b^{i-1}\| \leq mu\|\widehat{b^{i-1}}\| \\
& \cdots & & \\
\widehat{b^k} & = & H^k(\widehat{b^{k-1}} + \Delta b^{k-1}), & \|\Delta b^{k-1}\| \leq mu\|\widehat{b^{k-1}}\|
\end{array}
$$

Householder QR in finite precision

Let's assume a sequence of Householder transforms H^i $i = 1, \ldots, k$ computed and applied in precision u to a vector $b = b^0$

$$\begin{array}{lcll}
\widehat{b^1} & = & H^1(b^0 + \Delta b^0), & \|\Delta b^0\| \leq mu\|b^0\| = mu\|b\| \\
& \ldots & & \\
\widehat{b^i} & = & H^i(\widehat{b^{i-1}} + \Delta b^{i-1}), & \|\Delta b^{i-1}\| \leq mu\|\widehat{b^{i-1}}\| = mu\|b\| \\
& \ldots & & \\
\widehat{b^k} & = & H^k(\widehat{b^{k-1}} + \Delta b^{k-1}), & \|\Delta b^{k-1}\| \leq mu\|\widehat{b^{k-1}}\| = mu\|b\|
\end{array}$$

In conclusion

$$\widehat{b^k} = H^k \ldots H^1(b + \Delta b), \ \|\Delta b\| \leq kmu\|b\|$$

Householder QR in finite precision

Let's assume a sequence of Householder transforms H^i $i = 1, \ldots, k$ computed and applied in precision u to a vector $b = b^0$

$$\begin{array}{rcll}
\widehat{b^1} & = & H^1(b^0 + \Delta b^0), & \|\Delta b^0\| \leq mu\|b^0\| = mu\|b\| \\
& \ldots & & \\
\widehat{b^i} & = & H^i(\widehat{b^{i-1}} + \Delta b^{i-1}), & \|\Delta b^{i-1}\| \leq mu\|\widehat{b^{i-1}}\| = mu\|b\| \\
& \ldots & & \\
\widehat{b^k} & = & H^k(\widehat{b^{k-1}} + \Delta b^{k-1}), & \|\Delta b^{k-1}\| \leq mu\|\widehat{b^{k-1}}\| = mu\|b\|
\end{array}$$

In conclusion

$$\widehat{b^k} = H^k \ldots H^1(b + \Delta b), \ \|\Delta b\| \leq kmu\|b\|$$

and, consequently,

$$\|A - \widehat{Q}\widehat{R}\| \leq c_{mn}u\|A\| \quad \text{(Higham, Chap. 19)}$$

In the specific case of the QR factorization, the H transforms have a peculiar structure:

$$H^i \widehat{b}^{i-1} = \begin{bmatrix} I^{i-1} & \\ & \bar{H}^i \end{bmatrix} \begin{bmatrix} \widehat{b}^{i-1}_{1:i-1} \\ \widehat{b}^{i-1}_{i:m} \end{bmatrix}$$

and, therefore,

$$\|\Delta b^{i-1}\| \leq (m-i)u \|\widehat{b}^{i-1}_{i:m}\|$$

In the specific case of the QR factorization, the H transforms have a peculiar structure:

$$H^i \widehat{b}^{i-1} = \begin{bmatrix} I^{i-1} & \\ & \bar{H}^i \end{bmatrix} \begin{bmatrix} \widehat{b}^{i-1}_{1:i-1} \\ \widehat{b}^{i-1}_{i:m} \end{bmatrix}$$

and, therefore,

$$\|\Delta b^{i-1}\| \leq (m-i)u \|\widehat{b}^{i-1}_{i:m}\|$$

Introducing mixed precision

Because all the H^i and $\bar{H}^i$ are unitary transformations, $\|\widehat{b}^{i-1}_{i:m}\|$ will be monotonically decreasing for $i = 1, \ldots, k \longrightarrow$ starting at some i u can be increased without increasing the error

Theorem

Assume that a truncated QR factorization is computed such that $k \leq n$ Householder transformations are computed and applied to a matrix $A \in \mathbb{R}^{m \times n}$ using p different precisions of increasing unit roundoff u^i. Let k^i be the number of transformations that are computed using precision i. The computed $\widehat{R}_i$ and $\widehat{Q}_i$ satisfy

$$\|A - \sum_{i=1}^{p} \widehat{Q}_i \widehat{R}_i\| \leq \|A^{p+1}\| + \sum_{i=1}^{p} c_{mk^i} u^i \|A^i\|.$$

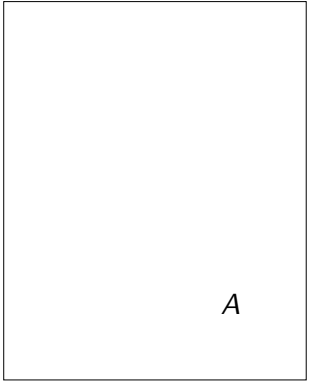
where A^i is the trailing submatrix after $\sum_{j=1}^{i-1} k_j$ transformations.

Buttari, M., Pacteau (2024)

Truncated Householder QR in mixed precision

Using this result into an algorithm:

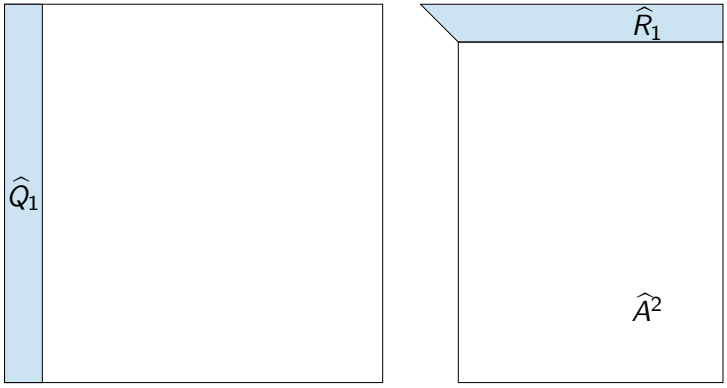
1. start the factorization with $u_1 \leq \varepsilon$



Truncated Householder QR in mixed precision

Using this result into an algorithm:

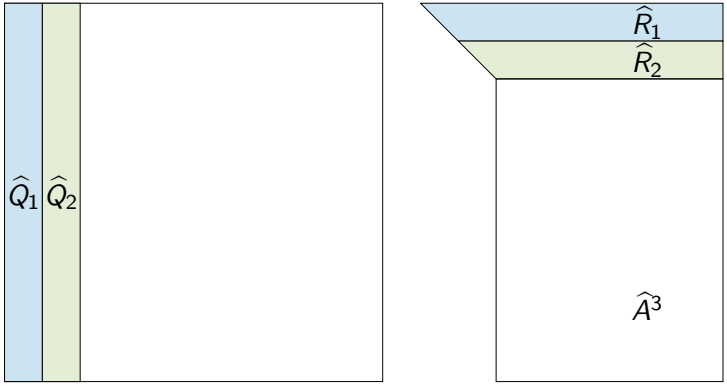
1. start the factorization with $u_1 \leq \varepsilon$
2. if after k_1 transformations $\|A^2\| \leq \varepsilon/u_2\|A\|$, switch to prec. u_2



Truncated Householder QR in mixed precision

Using this result into an algorithm:

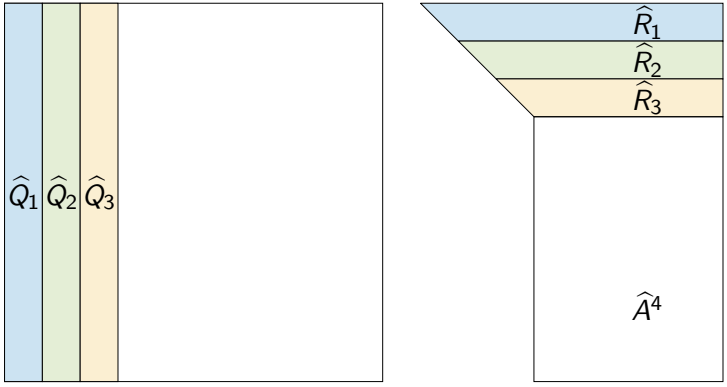
1. start the factorization with $u_1 \leq \varepsilon$
2. if after k_1 transformations $\|A^2\| \leq \varepsilon/u_2\|A\|$, switch to prec. u_2
3. same for precisions $2, \ldots, p$



Truncated Householder QR in mixed precision

Using this result into an algorithm:

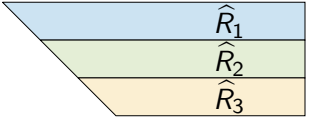
1. start the factorization with $u_1 \leq \varepsilon$
2. if after k_1 transformations $\|A^2\| \leq \varepsilon/u_2\|A\|$, switch to prec. u_2
3. same for precisions $2, \ldots, p$
4. if after $k_1 + \cdots + k_p$ transformations $\|A^{p+1}\| \leq \varepsilon\|A\|$, stop



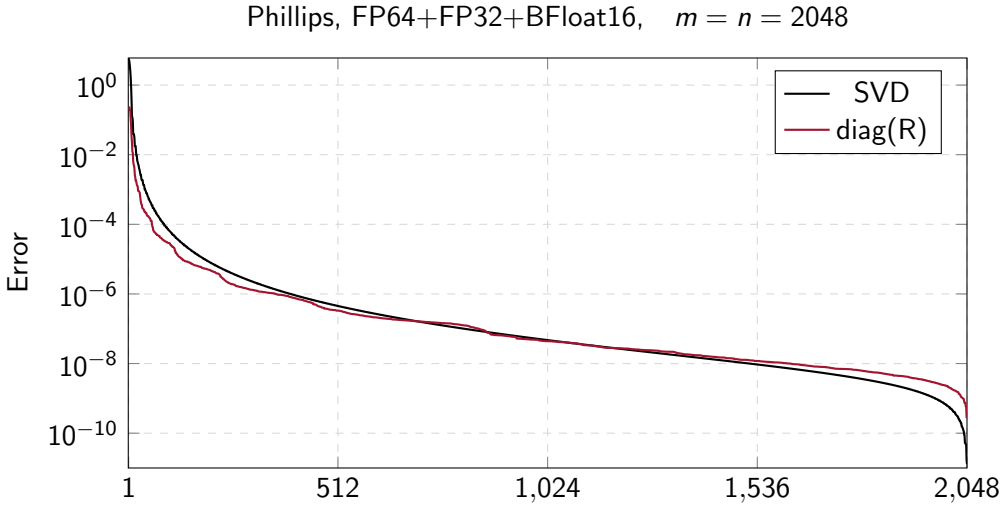
Truncated Householder QR in mixed precision

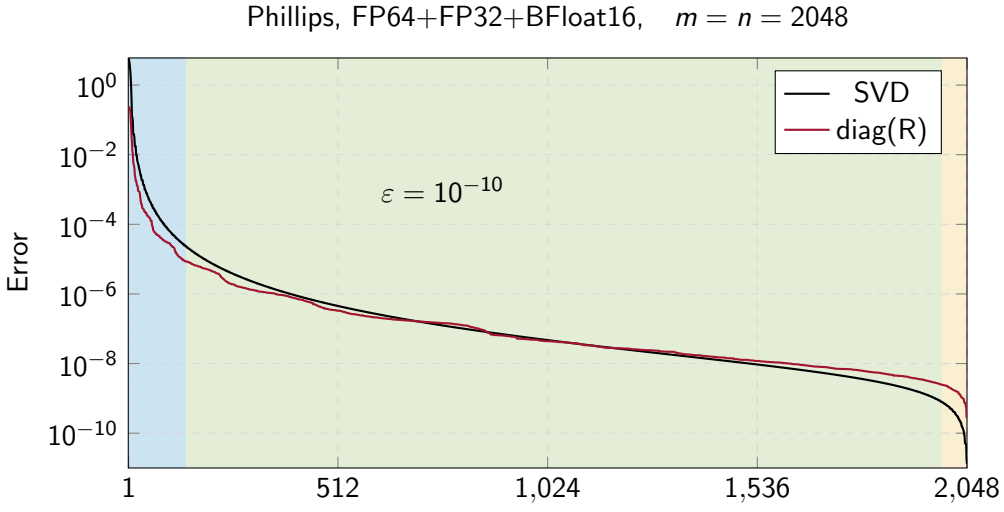
Using this result into an algorithm:

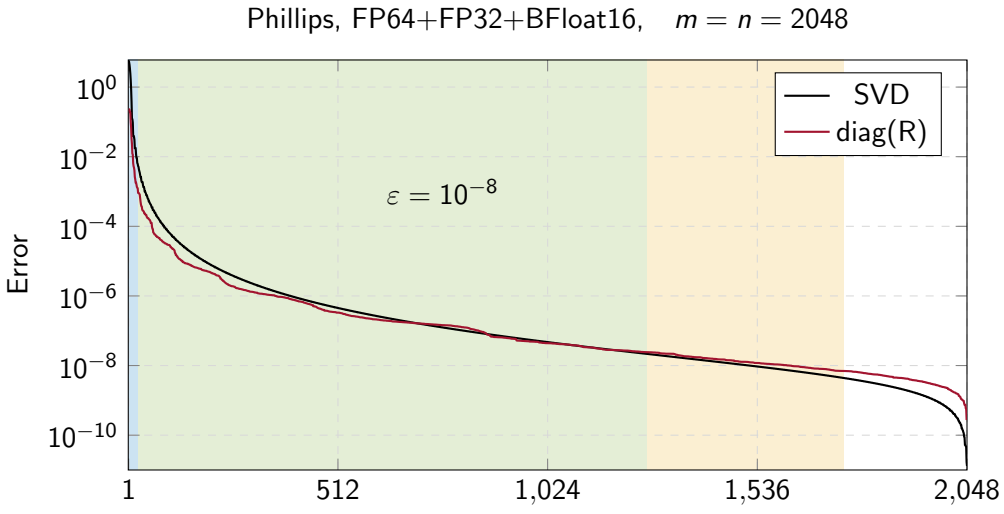
1. start the factorization with $u_1 \leq \varepsilon$
2. if after k_1 transformations $\|A^2\| \leq \varepsilon/u_2\|A\|$, switch to prec. u_2
3. same for precisions $2, \ldots, p$
4. if after $k_1 + \cdots + k_p$ transformations $\|A^{p+1}\| \leq \varepsilon\|A\|$, stop

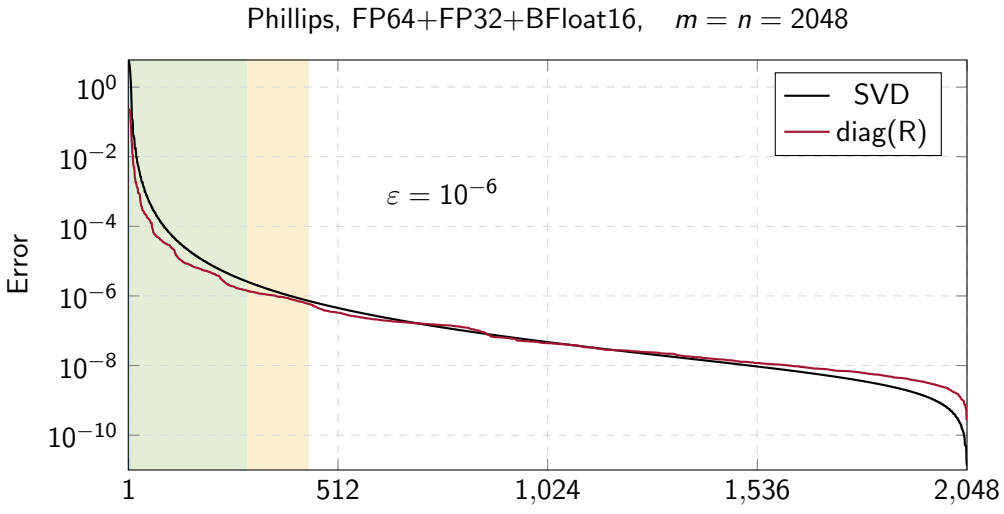


$$\|A - \widehat{Q}_1\widehat{R}_1 - \widehat{Q}_2\widehat{R}_2 - \widehat{Q}_3\widehat{R}_3\| \leq \beta\varepsilon\|A\|$$

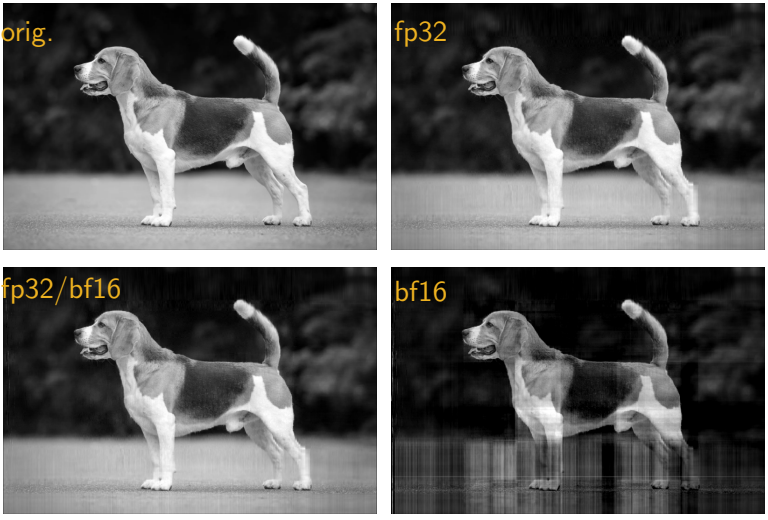

Phillips, FP64+FP32+BFloat16, $m = n = 2048$
Error
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
1
512
1,024
1,536
2,048
SVD
diag(R)


Phillips, FP64+FP32+BFloat16, $m = n = 2048$
Error
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
$\varepsilon = 10^{-10}$
SVD
diag(R)
1
512
1,024
1,536
2,048


Phillips, FP64+FP32+BFloat16, $m = n = 2048$
Error
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
$\varepsilon = 10^{-8}$
SVD
diag(R)
1
512
1,024
1,536
2,048

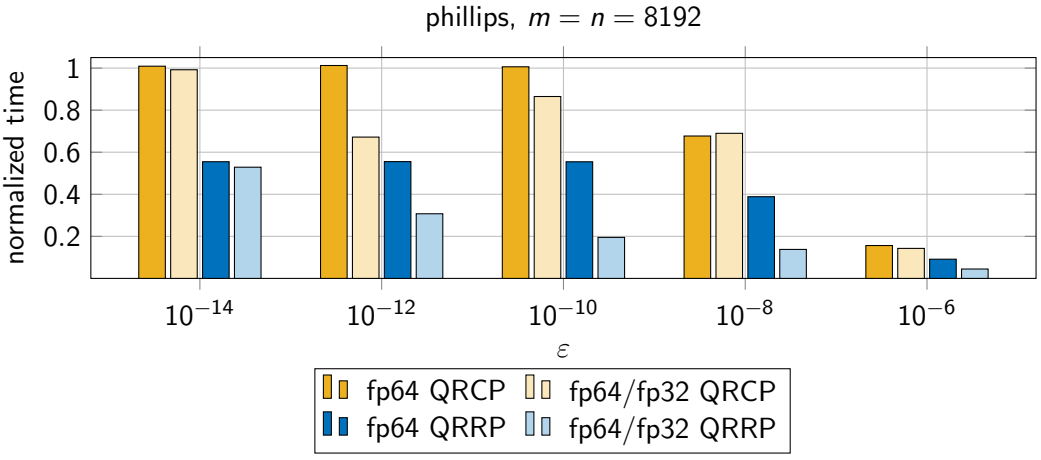

Phillips, FP64+FP32+BFloat16, $m = n = 2048$
Error
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
$\varepsilon = 10^{-6}$
SVD
diag(R)
1
512
1,024
1,536
2,048

Experiments: Julia, image compression



With $\varepsilon = 0.04$ the rank is 191 but only 13 steps are done in fp32 and the rest in bf16 (original size is 1057×1600)

Experiments: Fortran, performance



$$
\begin{aligned}
&G \leftarrow A^T A \\
&R^T R = \texttt{chol}(G) \\
&Q = AR^{-1}
\end{aligned}
$$

- The Cholesky QR algorithm computes the QR factorization of A via the Cholesky factorization of the Gram matrix $G = A^T A$
- Unstable: $\|I - Q^T Q\| \propto \kappa(G)u = \kappa(A)^2 u$
- But very efficient, almost entirely BLAS-3. Flop count $\sim 2mn^2 + n^3/3$

$$
\begin{aligned}
&G \leftarrow A^T A \\
&R^T R = \texttt{chol}(G) \\
&Q = AR^{-1}
\end{aligned}
$$

- The Cholesky QR algorithm computes the QR factorization of A via the Cholesky factorization of the Gram matrix $G = A^T A$
- Unstable: $\|I - Q^T Q\| \propto \kappa(G)u = \kappa(A)^2 u$
- But very efficient, almost entirely BLAS-3. Flop count $\sim 2mn^2 + n^3/3$

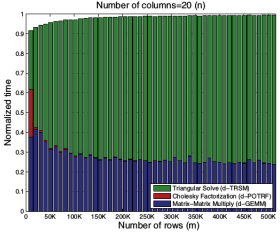
- CholQR2: even when Q is far from orthonormal, it can still be well conditioned, i.e., $\kappa(Q) \ll \kappa(A)$. An idea is thus to reapply CholQR to Q. Then $\|I - Q^T Q\| \propto u$ provided that $\kappa(A)^2 u \ll 1$.
- **MATLAB demo**

Mixed precision Cholesky QR

Yamazaki et al. (2015) :

1. $B = A^T A$ **in precision** u_{high}
2. Cholesky factorization: $R^T R = B$ **in precision** u_{high}
3. $Q = AR^{-1}$ **in precision** u_{low}

- Roughly half of the flops in precision u_{low}, but greater fraction in time



(a) d-CholQR time breakdown.

- $\|Q^T Q - I\| = O(\kappa(A)^2 u_{\text{high}} + \kappa(A) u_{\text{low}})$

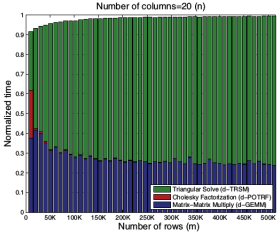
$\Rightarrow$ if $\kappa(A) \geq u_{\text{low}}/u_{\text{high}}$ no impact on stability

Mixed precision Cholesky QR

Yamazaki et al. (2015) :

1. $B = A^T A$ **in precision** u_{high}
2. Cholesky factorization: $R^T R = B$ **in precision** u_{high}
3. $Q = AR^{-1}$ **in precision** u_{low}

- Roughly half of the flops in precision u_{low}, but greater fraction in time

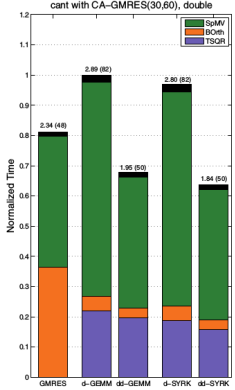


(a) d-CholQR time breakdown.

- $\|Q^T Q - I\| = O(\kappa(A)^2 u_{\text{high}} + \kappa(A) u_{\text{low}})$

$\Rightarrow$ if $\kappa(A) \geq u_{\text{low}}/u_{\text{high}}$ no impact on stability

Application to communication-avoiding GMRES



$$
\begin{aligned}
&G \leftarrow A^T A \\
&U\Sigma V^T = \mathtt{svd}(G) \\
&\bar{Q}R = \mathtt{qr}(S^{1/2}U^T) \\
&Q = AR^{-1}
\end{aligned}
$$

- In CholQR(2), Cholesky breaks down if G becomes indefinite, i.e., when $\kappa(A)^2 u \gg 1$
- This issue can be circumvented by replacing Cholesky by SVD
- Useful when combined with repeated orthonormalization (e.g., SVDQR2), since this allows to successfully orthonormalize A even when $\kappa(A)^2 u \gg 1$
- **MATLAB demo**

$$
\begin{aligned}
&\Omega \leftarrow \texttt{randn}(n, m) \\
&S \leftarrow \Omega A \\
&Q_S R_S \leftarrow \texttt{qr}(S) \\
&\widetilde{A} \leftarrow A R_S^{-1} \\
&QR \leftarrow \texttt{cholQR}(\widetilde{A})
\end{aligned}
$$

- Use random projection to inexpensively compute an n-dimension QR factorization $Q_S R_S$
- Use R_S to precondition A: $\kappa(\widetilde{A}) = \kappa(AR_S^{-1}) \ll \kappa(A)$
- Apply CholQR (or any of its variants) to $\widetilde{A}$
- The R-factor of A can be recoved as RR_S if needed

  Balabanov (2022) Garrison and Ipsen (2024)
- **MATLAB demo**

Gram SVD

- So far we have seen Gram-based algorithms for QR. What about for LRA?
- The SVD of A is closely related to the EVD of G:

$$A = U\Sigma V^T \Rightarrow G = V\Sigma^2 V^T$$

- The Gram SVD algorithm exploits this fact by computing the LRA $(AV_k) \times V_k^T$ where V_k are the truncated eigenvectors of G (singular vectors of A)

Input: $A \in \mathbb{R}^{m\times n}$, rank k or tolerance ε
Output: $X \in \mathbb{R}^{m\times k}$, $Y \in \mathbb{R}^{n\times k}$ such that $A \approx XY^T$
$G \leftarrow A^T A$
$V\Lambda V^T \leftarrow \texttt{eig}(G)$
Truncate V into V_k (such that $\|\Lambda_{k+1:\,n}\| \leq \varepsilon^2$)
$X \leftarrow AV_k$ and $Y \leftarrow V_k$

- Gram SVD is unstable in finite precision. However, despite the fact that $\kappa(G) = \kappa(A)^2$, we can prove that the computed LRA satisfies M. (2024)

$$\|A - (AV_k)V_k^T\| \leq \min(\kappa(A)u, \sqrt{u})\|A\|$$

- **MATLAB demo**

Objective

- Compute solution to linear system $Ax = b$
- $A \in \mathbb{R}^{n \times n}$ is ill conditioned

Preconditioned iterative method

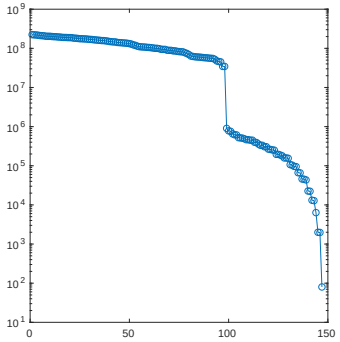
1. Compute preconditioner M^{-1} such that $M^{-1} \approx A^{-1}$, e.g.,
   - Low precision LU factorization
   - Incomplete LU factorization
   - Block Low-Rank LU factorization
2. Solve $Ax = b$ via some iterative method (e.g., GMRES) preconditioned by M^{-1}, e.g., with leeft-preconditioning, $M^{-1}Ax = M^{-1}b$

- Convergence to solution may be slow or fail

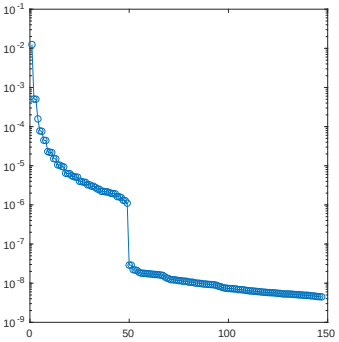
$\Rightarrow$ **Objective: accelerate convergence**

Key observation

Matrix lund_a ($n = 147$, $\kappa(A) = 2.8e{+}06$)



SVD of A



SVD of A^{-1}

- Often, A is ill conditioned due to a small number of small singular values
- Then, A^{-1} is numerically low-rank

Factorization error might be low-rank?

Assume $M = A + \Delta A$ and consider the error

$$E = M^{-1}A - I = M^{-1}(M + \Delta A) - I$$
$$= M^{-1}\Delta A \approx A^{-1}\Delta A$$

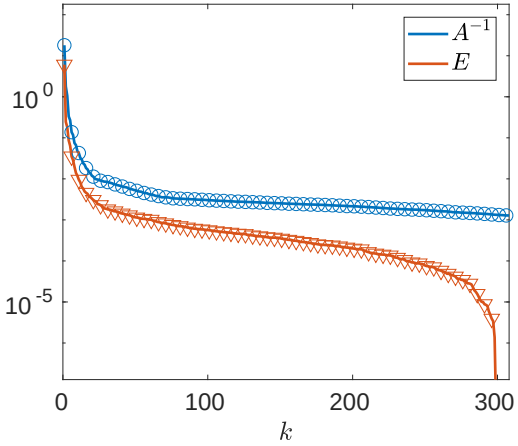
Does E retain the low-rank property of A^{-1}?

A novel preconditioner

Consider the preconditioner $M_k = M(I + E_k)$ with E_k a rank-k approximation to E.

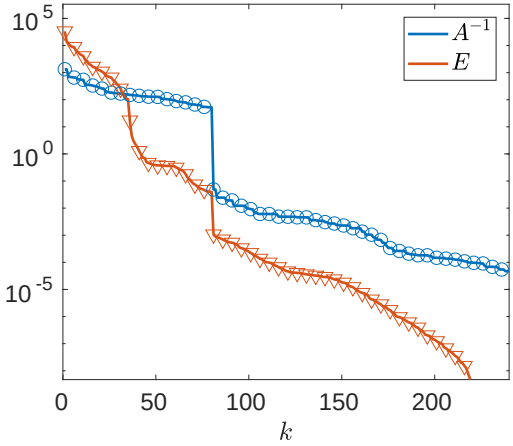
- If $E = E_k$, $M_k = A$
- If $E \approx E_k$ for some small k, M_k^{-1} can be computed cheaply via Sherman-Morrison-Woodbury formula

Typical SV distributions of A^{-1} and E



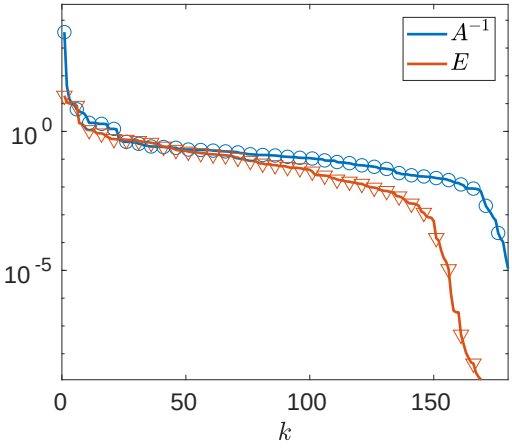
Matrix cz308

Typical SV distributions of A^{-1} and E



Matrix steam1

Typical SV distributions of A^{-1} and E



Matrix rajat14

Suggestions

- Gather some test matrices for which A^{-1} is numerically low-rank (you can generate them randomly, or take a look at Suitesparse collection for real-life problems)
- Prepare a reference solver (suggestion: use MATLAB's `gmres`) and some reference preconditioners M (e.g., MATLAB's `ilu`, or low precision `lu`)[1] (Lecture 9)
- If you use sparse matrices, remember Lecture 6 and look up MATLAB's reordering tools (e.g., `dissect`)
- How to compute a rank-k approximation of E ? Explicitly forming E is not a good idea! You should rather use a method that only requires matrix–vector multiplies...
- Perform some numerical experiments and test the role of k (or ε), etc.
- Should one build a fixed-rank (k) or fixed-accuracy (ε) LRA of E?
- Should one use left or right preconditioning? (note that M_k is defined differently in either case)
- Can refer to Higham and M. (2019) for some guidance

[1] Either using MATLAB's `single` or simulating low precision by computing `lu`$(A + \Delta A)$ for a random perturbation ΔA