Numerical Linear Algebra for Computational Science and Information Engineering

Orthogonalization

Nicolas Venkovic
nicolas.venkovic@tum.de

Chair of Computational Mathematics
School of Computation, Information and Technology
Technical University of Munich



Outline I

1. QR factorization
   Section 4 in Darve & Wootters (2021) 1
2. Householder reflections
   Section 4.1 in Darve & Wootters (2021) 5
3. Givens rotations
   Section 4.2 in Darve & Wootters (2021) 11
4. CholeskyQR 14
5. Tall-and-skinny QR 17
6. Gram-Schmidt procedures
   Section 4.3 in Darve & Wootters (2021) 21

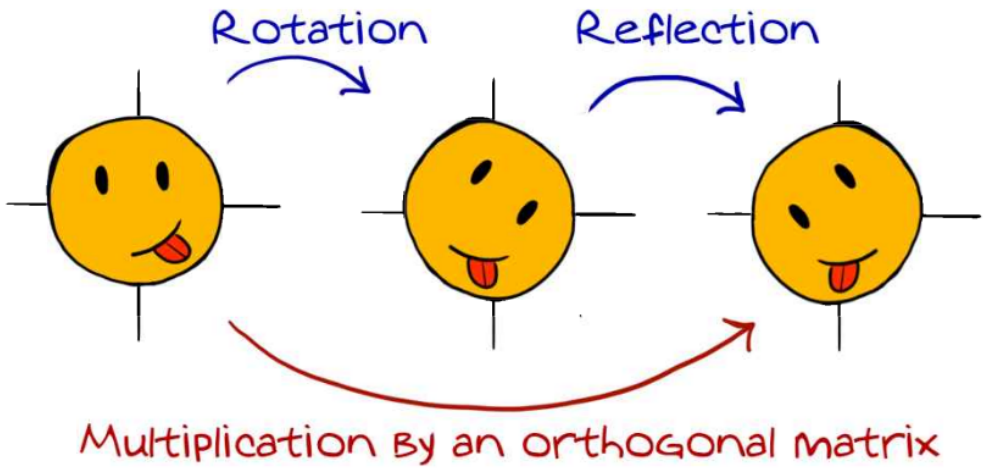
QR factorization

Section 4 in Darve & Wootters (2021)

QR factorization

- A **QR factorization decomposes** a matrix as the **product of an orthogonal matrix Q with an upper-triangular matrix R.**
- Recall that a matrix Q is orthogonal if $Q^T Q = I$.
- If a matrix is orthogonal, then $\|Qx\|_2 = \|x\|_2$ for all x, i.e., Q doesn't change the length of vectors.

  The operations that do not change the length of vectors are **rotations** and **reflections**, so an orthogonal matrix can be thought of as a map that combines a rotation with a reflection.



Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

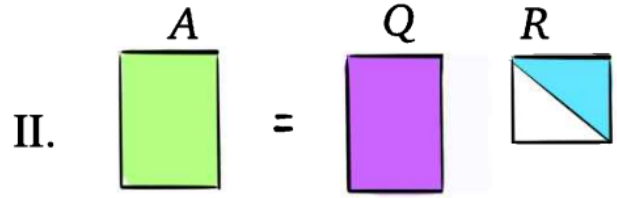
QR factorization, cont'd

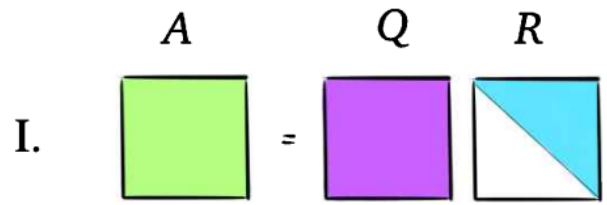
QR factorization

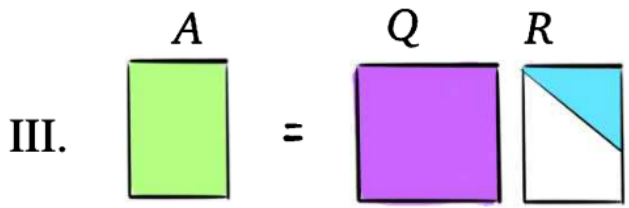
- Let A be a real $m \times n$ matrix with $m \geq n$. Then, there is an orthogonal matrix Q and an upper-triangular matrix R such that $A = QR$. This is called the **QR factorization**.
- When A is complex, there is still a factorization $A = QR$, but Q is unitary, i.e., $Q^H Q = I$.

For the rest this lecture, we assume A is real, but QR factorizations do exist for complex matrices.

- There are different forms of QR factorizations, depending on the shape of A:







Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

Applications of the QR factorization

The QR factorization has several applications in numerical linear algebra:

1. It can be used to solve least-squares problems, i.e., problems of the form $\arg\min_x \|Ax - b\|_2$ where A is tall and skiny.
2. It is used as part of eigen- and singular value algorithms for small dense matrices.
3. It is also used in iterative methods to solve linear systems and compute eigenvalues, such as in Krylov methods.

QR factorization and least-squares problems

To see why the QR factorization can be useful, let's look briefly at the least-squares problem:

- Let $A \in \mathbb{R}^{m \times n}$ with $m > n$.We want to find x such that Ax is closest to b in Euclidean distance That is

$$x^* = \arg\min_x \|Ax - b\|_2.$$

To do this, we use the QR factorization, with a square Q, i.e., case III from slide #2:

$$\|Ax - b\|_2 = \|Q^T(Ax - b)\|_2 = \|Q^T(QRx - b)\|_2 = \|Rx - Q^T b\|_2$$

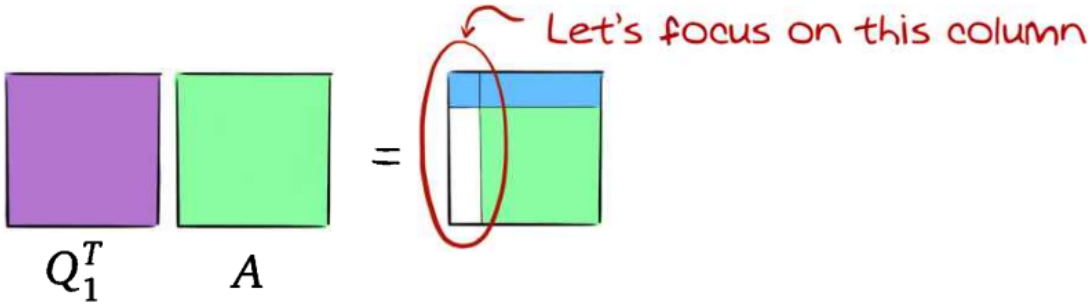
where we used the fact that for any vector y, $\|Q^T y\|_2 = \|y\|_2$. As it turns out, it is easier to find x that minimizes $\|Rx - Q^T b\|_2$ than it is to minimize $\|Ax - b\|_2$.

Householder reflections

Section 4.1 in Darve & Wootters (2021)

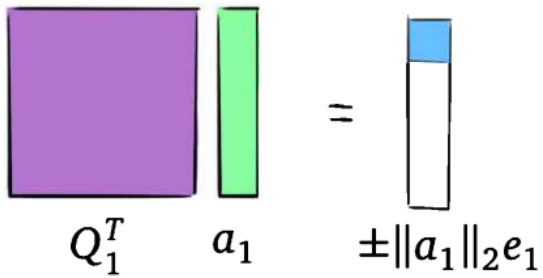
Householder reflections

- Householder reflections are one of the most reliable methods to compute a QR factorization with a square Q, i.e., cases I and III.
- That is, we ask the question, does there exists a matrix Q s.t. $Q^T A = R$.
- Our goal is thus to create zero entries below the diagonal. Starting by the first column, we have:



- We need to apply an orthogonal transformation Q_1^T to transform the first column of A into a vector in the direction of e_1.

  Let's write $A = [a_1 | \dots | a_n]$. Then, since Q_1^T does not change the norm of a_1, we should have:



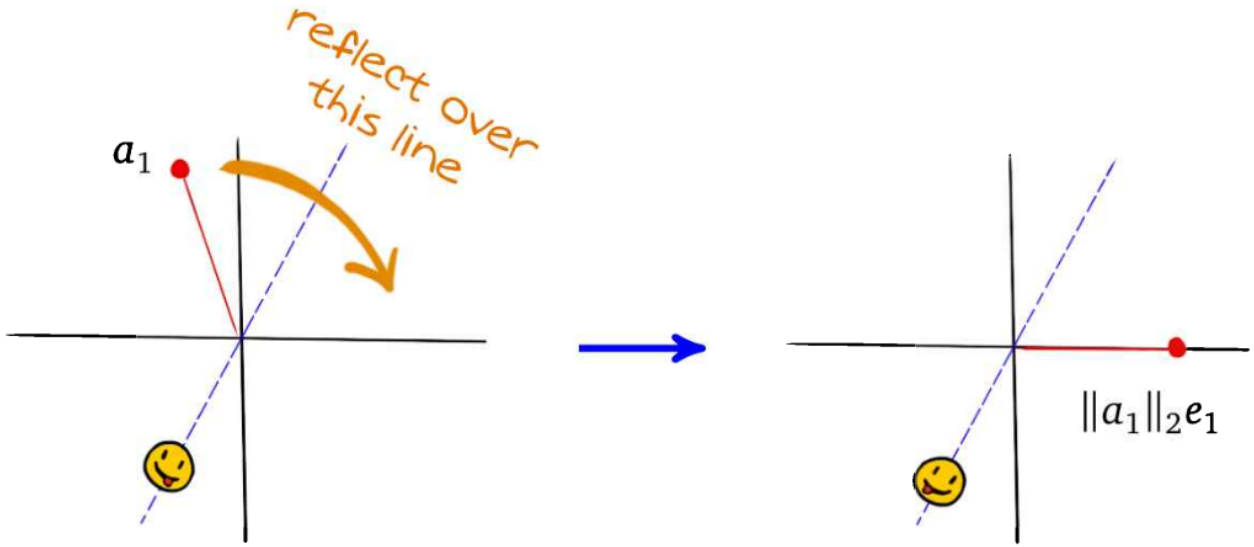
Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

Householder reflections, cont'd$_1$

- A logical choice for Q_1^T would be a rotation that maps a_1 parallel to e_1.

  However, rotations in high dimensions are not so easy to set up.

  Thus, we instead choose Q_1^T to be a **reflection** that maps a_1 parallel to e_1.
  In 2D, it looks like this:



Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

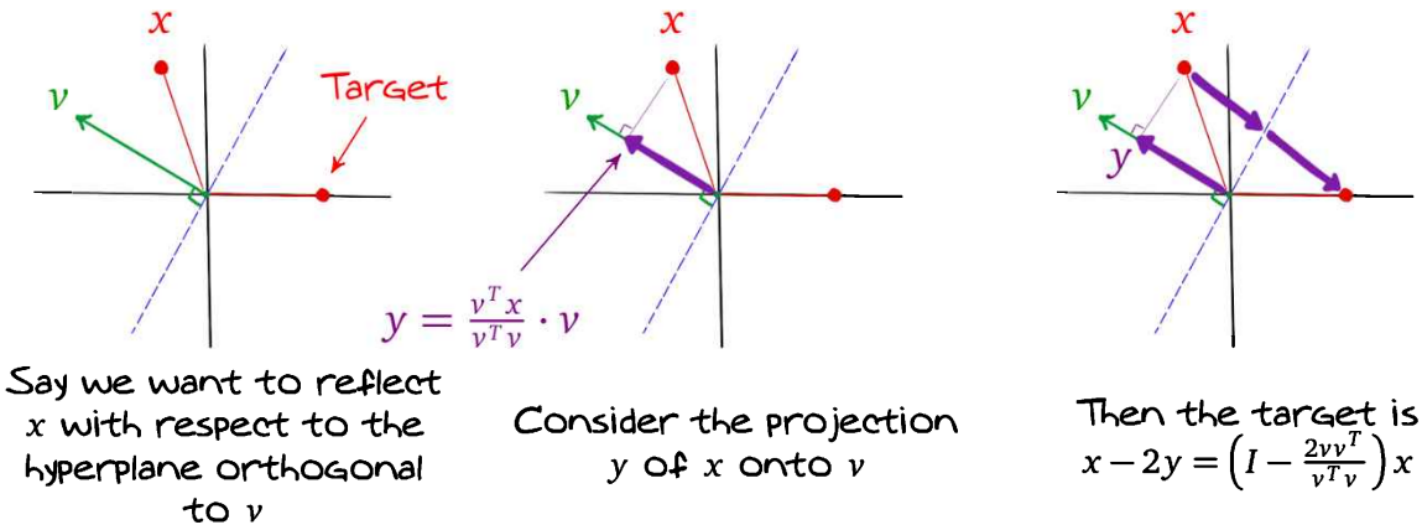
Now that we have an idea of what the reflection should be doing, we need to figure out its mathematical formula.

Householder reflections, cont'd$_2$

- In higher dimensions, a reflection is defined with respect to a hyperplane represented by a normal vector v:



Given v, we can reason geometrically (back in 2D) about what a reflection is:



Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

Householder reflections, cont'd$_3$

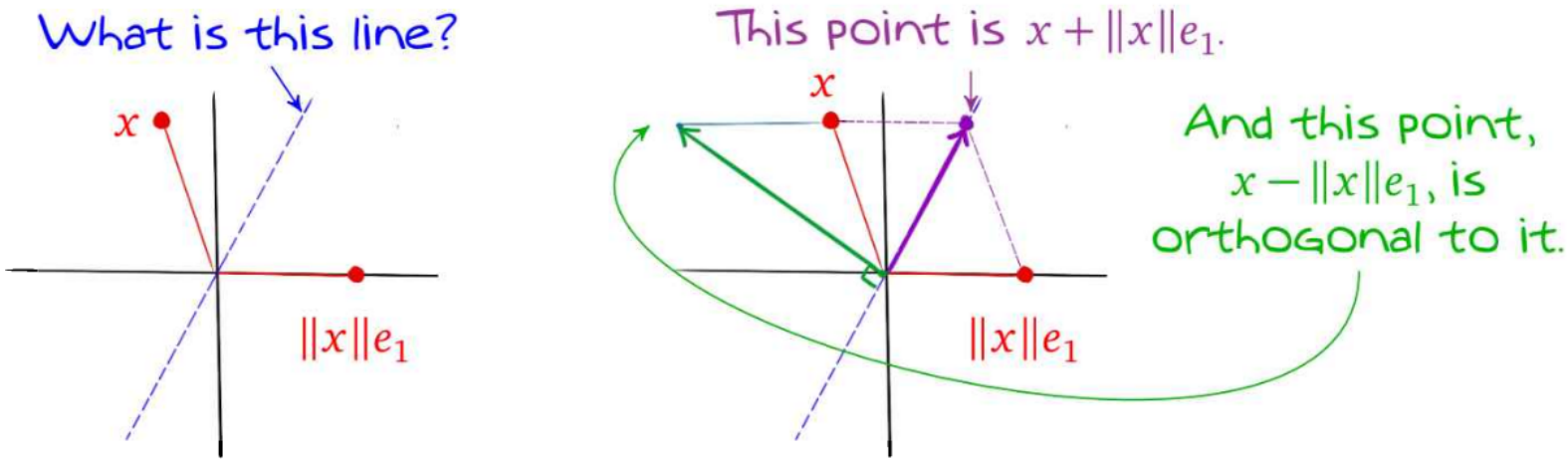
Reflection

Let P be the matrix which represents a reflection over the hyperplane orthogonal to some vector v. Then P is given by

$$P = I - \beta v v^T \text{ where } \beta = \frac{2}{v^T v}.$$

- Now we need to pick v to arrive at a Householder reflection, i.e., to get a transformation from x to $\|x\|_2 e_1$.

  The following geometric argument shows that $v = x - \|x\|_2 e_1$ will work



Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

Householder reflections, cont'd$_3$

Let $v = x - \|x\|_2 e_1$, then we see that

$$\begin{aligned}
Px &= \left(I - 2\frac{vv^T}{v^T v} \right) x \\
&= \left(I - 2\frac{(x - \|x\|_2 e_1)(x - \|x\|_2 e_1)^T}{(x - \|x\|_2 e_1)^T (x - \|x\|_2 e_1)} \right) x \\
&= \left(I - 2\frac{(x - \|x\|_2 e_1)(x - \|x\|_2 e_1)^T}{2(\|x\|_2^2 - 2\|x\|_2 x_1)} \right) x \\
&= x - \frac{(x - \|x\|_2 e_1)(\|x\|_2^2 - \|x\|_2 x_1)}{(\|x\|_2^2 - 2\|x\|_2 x_1)} \\
&= x - (x - \|x\|_2 e_1) \\
&= \|x\|_2 e_1
\end{aligned}$$

so that, indeed, a reflection over the hyperplane orthogonal to $v = x - \|x\|_2 e_1$, is a Householder reflection of x.

Iterating Householder reflections

- Now that we know how to operate a first Householder reflection from a_1 to $\|a_1\|_2 e_1$, we can apply a series of Householder transformations to progressively reduce A to a upper-triangular form.

  We proceed by first zeroing entries in the first column, then in the second column, and so on.

  In the end, for $A \in \mathbb{R}^{m \times n}$ with $m \geq n$, we have

  $$Q_{n-1}^T \dots Q_1^T A = R,$$

  which is equivalent to

  $$A = Q_1 \dots Q_{n-1} R = QR$$

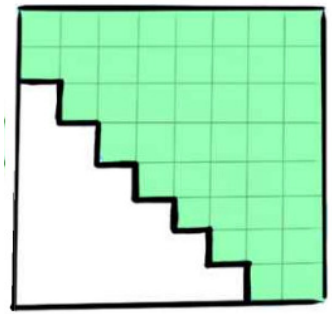
  where $Q = Q_1 \dots Q_{n-1} \in \mathbb{R}^{m \times m}$, and R has zeros in the $m - n$ rows if $m > n$.
- In practice, when reducing A to R, the matrix P is never formed explicitly, instead we compute $PA = A - \beta v(v^T A)$ which carries a cost $\mathcal{O}(2mn)$, instead of $\mathcal{O}(m^2 n)$ when P is assembled and applied, i.e., as P is dense.

Givens rotations

Section 4.2 in Darve & Wootters (2021)

Givens rotations

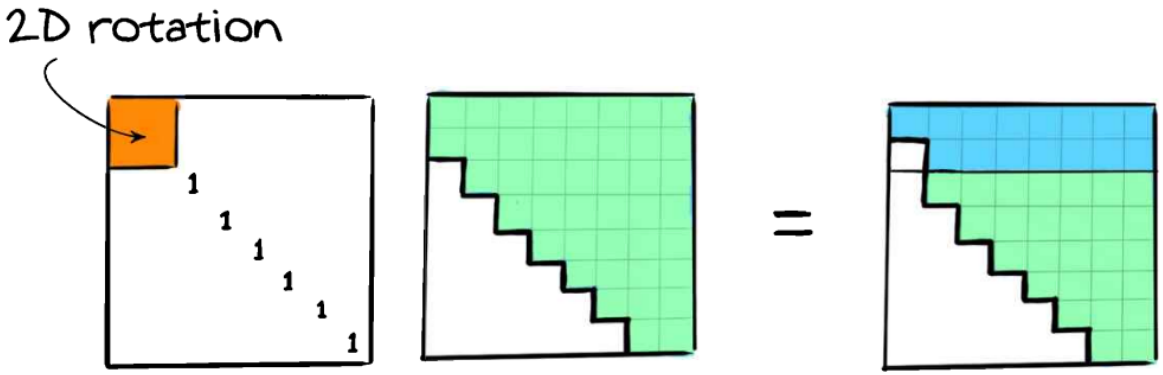
- When the matrix A is upper Hessenberg, i.e., where $a_{ij} = 0$ for all $i > j + 1$, most of the subdiagonal components are already zero, so that using Householder transformations in this situation is a bit of an overkill:



A

Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

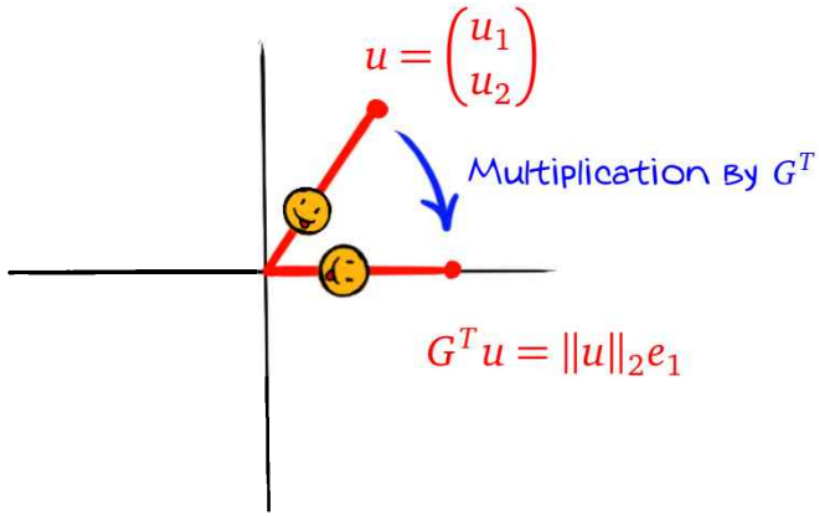
- On every column of A, only one entry needs to be zeroed, so that 2D rotations, which are easy to set up, can be deployed for the job:



Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

Givens rotations, cont'd$_1$

- Zeroing a single subdiagonal entry can be reduced to considering a 2D vector $u = (u_1, u_2)$ and finding a rotation G^T such that the vector u becomes aligned with e_1:



Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

With some algebra, we find:

Givens rotation

A **Givens rotation** which rotates $u = (u_1, u_2)^T$ to $\|u\|_2 e_1$ is the 2 x 2 matrix defined by

$$G^T = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}, \quad c = \frac{u_1}{\|u\|_2}, \quad s = -\frac{u_2}{\|u\|_2}.$$

Givens rotations, cont'd$_2$

- For an upper Hessenberg matrix A of size $m \times n$, we can compute its QR factorization using a sequence of Givens rotations.
- The algorithm is as follows:
  1. For each column $j = 1, \ldots, n-1$:
  2. Construct a Givens rotation matrix G^T that zeros $a_{j+1,j}$
  3. Apply G^T to rows j and $j+1$ of A

CholeskyQR

CholeskyQR factorization

- The **CholeskyQR** algorithm builds an economic factorization (i.e., case III) of $A \in \mathbb{R}^{m \times n}$, typically for $n \ll m$.
- It proceeds by first obtaining the R factor through a Cholesky factorization of the Gram matrix $A^T A$, then retrieving the Q factor by forward substitution:

  The algorithm is as follows:

  1. $X := A^T A$ // BLAS 3
  2. Find the upper triangular R s.t. $R^T R = X$ // Cholesky factorization
  3. $Q := AR^{-1}$ // triangular solves
- CholeskyQR reaches higher arithmetic intensity and requires less synchronizations than Householder QR:

  $$\implies \text{favored for distributed implementations.}$$
- But, unlike Householder QR, CholeskyQR suffers from instability, with

  $$\text{LOO}(Q) := \|I_n - Q^T Q\|_2 = \mathcal{O}(u \cdot \kappa(A)).$$

Reorthogonalized variants

- The lack of stability of an orthogonlization procedure can be partly remedied by repetition. The repetition of CholeskyQR is referred to as **CholeskyQR2**:
  1. $(Q_1, R_1) \leftarrow\!\shortmid \text{CholeskyQR}(A)$
  2. $(Q, R_2) \leftarrow\!\shortmid \text{CholeskyQR}(Q_1)$
  3. $R := R_2 R_1$

  Repeating CholeskyQR significantly improves orthogonality,yielding $\text{LOO}(Q) = \mathcal{O}(u)$ under the condition that $\kappa(A) = \mathcal{O}(u^{-1/2})$.
- Another way to improve the stability of CholeskyQR is to shift the Gram matrix, decreasing its condition number, thus improving the stability of the Cholesky factorization. The resulting **Shifted CholeskyQR** is given by:
  1. $X := A^T A$
  2. $s := 11(mn + n(n+1))u\|A\|_2^2$ // calculate shift
  3. $X := X + sI_n$ // shift Gram matrix
  4. Find the upper triangular R s.t. $R^T R = X$ // Cholesky factorization
  5. $Q := AR^{-1}$ // triangular solves

  which ensures $\kappa(Q) = \mathcal{O}(u^{-1/2})$ as long as $\kappa(A) = \mathcal{O}(u^{-1})$.

Reorthogonalized variants, cont'd

- Therefore, Shifted CholeskyQR can be used as a preconditioner to CholeskyQR2. This procedure is referred to as **Shifted CholeskyQR3**:
  1. $(Q_1, R_1) \leftarrow\!\shortmid$ Shifted CholeskyQR(A)
  2. $(Q, R_2) \leftarrow\!\shortmid$ CholeskyQR2(Q_1)
  3. $R := R_2 R_1$

  which yields LOO$(Q) = \mathcal{O}(u)$ as long as $\kappa(A) = \mathcal{O}(u^{-1})$.

Yamamoto, Y., Nakatsukasa, Y., Yanagisawa, Y., & Fukaya, T. (2015). Roundoff error analysis of the CholeskyQR2 algorithm. Electron. Trans. Numer. Anal, 44(01), 306-326.

Fukaya, T., Kannan, R., Nakatsukasa, Y., Yamamoto, Y., & Yanagisawa, Y. (2020). Shifted Cholesky QR for computing the QR factorization of ill-conditioned matrices. SIAM Journal on Scientific Computing, 42(1), A477-A503.

Tall-and-skinny QR

Tall-and-skinny QR (TSQR)

- Householder QR is unconditionally stable but memory-bound, and CholeskyQR variants, although they achieve high arithmetic intensity, offer limited stability.
- Tall-and-skinny (TSQR) algorithms offer both unconditional stability and high arithmetic intensity.

  TSQR is particularly relevant when only the upper triangular factor R is needed.
- The key idea of TSQR is to **partition the matrix A into blocks**and compute QR factorizations hierarchically.

Tall-and-skinny QR (TSQR), cont'd$_1$

- For a matrix $A \in \mathbb{R}^{m \times n}$ with $m \gg n$, we partition A into p blocks:

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_p \end{bmatrix}$$

  where each $A_i \in \mathbb{R}^{(m/p) \times n}$.
- We then compute the QR factorization of each block independently:

$$A_i = Q_i R_i, \quad i = 1, \ldots, p.$$

- This gives us:

$$A = \begin{bmatrix} Q_1 R_1 \\ Q_2 R_2 \\ \vdots \\ Q_p R_p \end{bmatrix} = \begin{bmatrix} Q_1 & & & \\ & Q_2 & & \\ & & \ddots & \\ & & & Q_p \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_p \end{bmatrix}.$$

Tall-and-skinny QR (TSQR), cont'd$_2$

- Next, we need to compute the QR factorization of the stacked R matrices:

$$\begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_p \end{bmatrix} = \tilde{Q}\tilde{R}$$

  where $\tilde{Q} \in \mathbb{R}^{(pn)\times n}$ and $\tilde{R} \in \mathbb{R}^{n\times n}$.
- The final QR factorization is then:

$$A = \begin{bmatrix} Q_1 & & & \\ & Q_2 & & \\ & & \ddots & \\ & & & Q_p \end{bmatrix} \tilde{Q}\tilde{R} = Q\tilde{R}$$

  where $Q = \text{diag}(Q_1, Q_2, \ldots, Q_p)\tilde{Q}$.
- **Key advantage**: Each block can be processed independently, making TSQR highly parallelizable.

Tall-and-skinny QR (TSQR), cont'd$_3$

- The TSQR algorithm is as follows:
  1. Partition A into p blocks: $A = [A_1^T, A_2^T, \ldots, A_p^T]^T$
  2. Parallelizable step: Compute $(Q_i, R_i) = \mathrm{QR}(A_i)$ for $i = 1, \ldots, p$
  3. Stack the R factors: $\tilde{A} = [R_1^T, R_2^T, \ldots, R_p^T]^T$
  4. Compute $(\tilde{Q}, \tilde{R}) = \mathrm{QR}(\tilde{A})$
  5. If only R is needed, return $\tilde{R}$. Otherwise, $Q = \mathrm{diag}(Q_1, \ldots, Q_p)\tilde{Q}$
- The loss of orthogonality of TSQR is such that $\mathrm{LOO}(Q) = \mathcal{O}(u)$ irrespective of A, i.e., TSQR is unconditionally stable.
- TSQR combines the best of both worlds: numerical stability of Householder QR with high arithmetic intensity and parallelizability.

  $\Longrightarrow$ TSQR favored for high-performance implementations.

Gram-Schmidt procedures

Section 4.3 in Darve & Wootters (2021)

Gram-Schmidt procedures

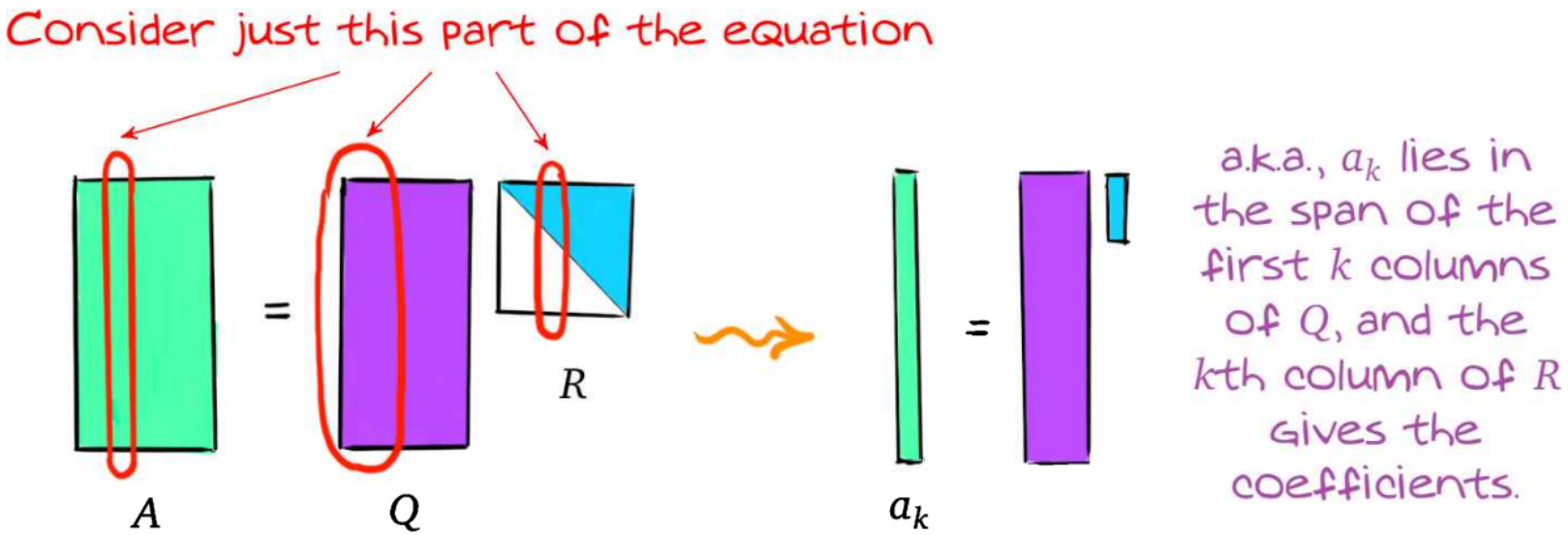
- Householder reflections and Givens rotations produce a square matrix $Q \in \mathbb{R}^{m \times m}$, even when $A \in \mathbb{R}^{m \times n}$ with $m > n$, i.e., case III.

  On the other hand, **Gram-Schmidt** procedures will **produce a rectangular, tall-and-skinny matrix** $Q \in \mathbb{R}^{m \times n}$, i.e., like in case II.
- Another peculiarity of **Gram-Schmidt** procedures is that they **work column-by-column**, i.e., to compute q_i in $Q = [q_1, \ldots, q_n]$, you only need access to a_i from $A = [a_1, \ldots, a_n]$ and $q_1, \ldots, q_{k-1}$.

  This feature of the Gram-Schmidt procedures is particularly **useful in Krylov methods** where A is not available all at once, and the new column a_i to orthogonalize is only available after performing a full iteration of computations.
- The first k columns $q_1, \ldots, q_k$ formed by Gram-Schmidt procedure in Q are an **orthonormal basis** of the subspace spanned by $a_1, \ldots, a_k$.

Gram-Schmidt procedures, cont'd$_1$

- Visualizing the column $a_k = QR_{:,k}$,and the fact that $R_{:,k}$ has k non-zero entries followed by $m - k$ zeros on the subdiagonal,we can write $a_k = Q_{:,1:k}R_{1:k,k}$:



Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

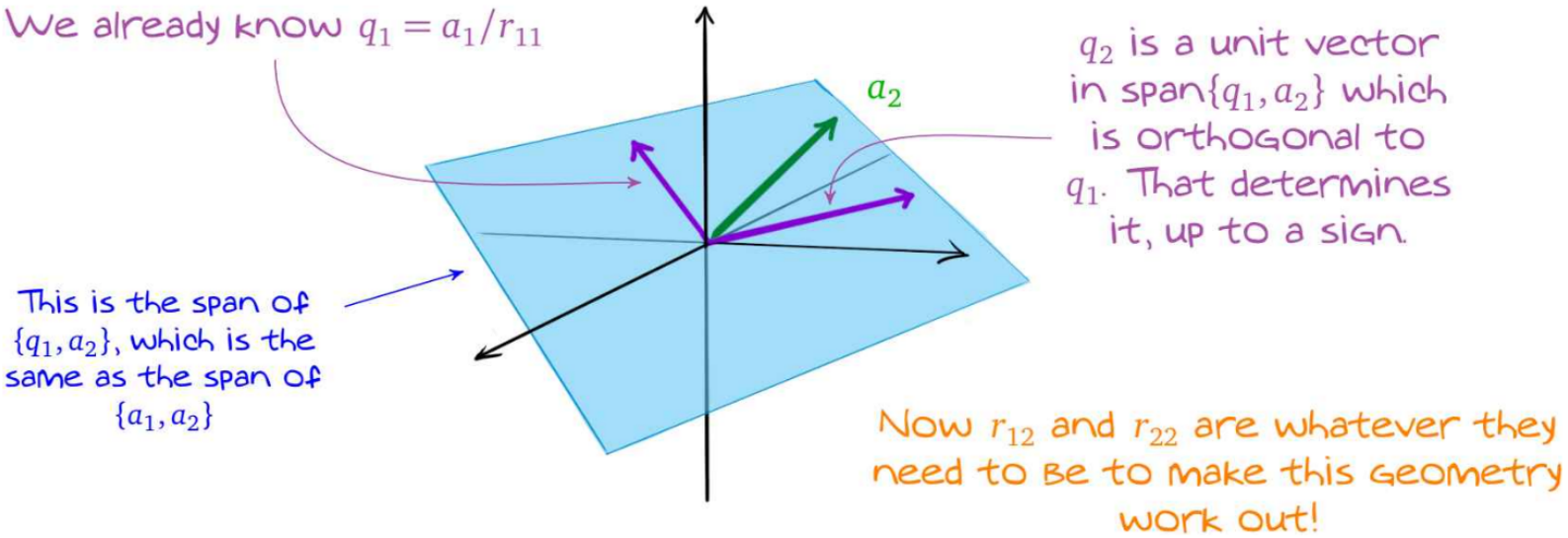
That is, a_k is formed by linear combination of $q_1, \ldots, q_k$:

$$a_k = r_{1k}q_1 + \cdots + r_{kk}q_k.$$

- Thus, instead of searching for the matrix Q that makes $Q^T A$ upper triangular, we search for the upper triangular matrix R such that every a_k is given by linear combination of $q_1, \ldots, q_k$.

Gram-Schmidt procedures, cont'd$_2$

- First, we have $a_1 = r_{11}q_1$, where q_1 has unit norm, so that we set $r_{11} := \|a_1\|_2$ and $q_1 := a_1/r_{11}$.
- Then, we continue iteratively, i.e., $a_2 = r_{12}q_1 + r_{22}q_2$ where q_2 is a unit vector in $\mathrm{span}\{q_1, a_2\} = \mathrm{span}\{a_1, a_2\}$, orthogonal to q_1:



Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

Then r_{12} and r_{22} are found to close the system.

Classical Gram-Schmidt (CGS) procedure

- More formally, for each $1 \leq k \leq n$, we write

$$a_k = \sum_{i=1}^{k} r_{ik} q_i = r_{kk} q_k + \sum_{i=1}^{k-1} r_{ik} q_i$$

Assuming we already know q_j and r_{ij} for all $j < k$ and $i \leq j$, we can then use this formula to find expressions for the r_{ik}'s and q_k.

First, multiplying by q_i^T and invoking the orthonormality of the basis given by $q_1, \ldots, q_k$, we get

$$q_i^T a_k = \sum_{j=1}^{k} r_{jk} q_i^T q_j = r_{ik} q_i^T q_i \implies r_{ik} = q_i^T a_k \text{ for } i < k.$$

Next, to find r_{kk}, we have $q_k r_{kk} = a_k - \sum_{i=1}^{k-1} r_{ik} q_i$ where q_k has unit norm so that

$$r_{kk} = \left\| a_k - \sum_{i=1}^{k-1} r_{ik} q_i \right\|_2 .$$

Classical Gram-Schmidt (CGS) procedure, cont'd$_1$

Note that r_{kk} could also be chosen to be negative. However, it is standard to let R have positive components on the diagonal.

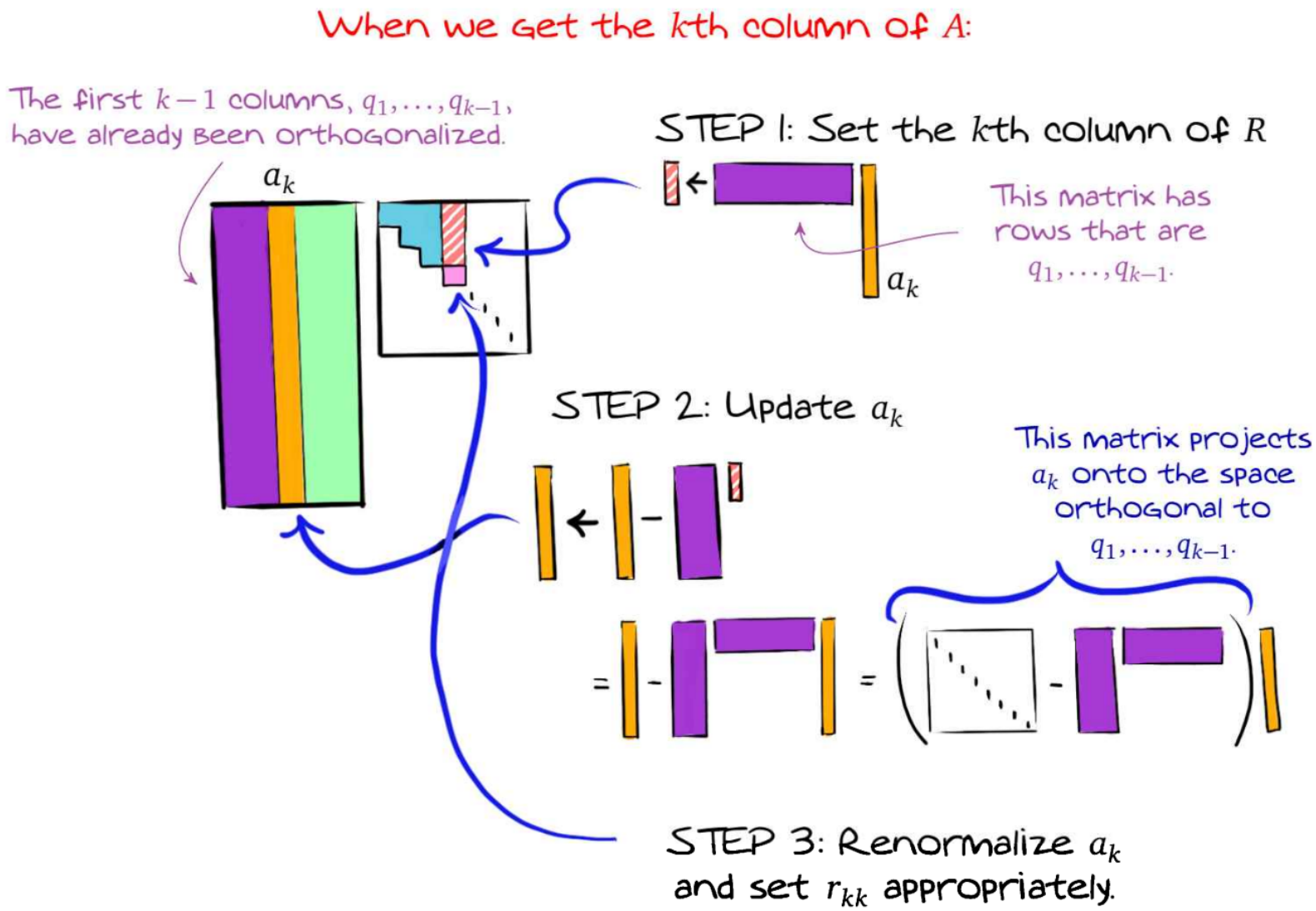
Finally, we have

$$q_k = \frac{1}{r_{kk}} \left(a_k - \sum_{i=1}^{k-1} r_{ik} q_i \right).$$

This procedure is referred to as the **classical Gram-Schmidt** algorithm.

We see indeed that, in order to compute q_k, you need access to a_k and $q_1, \ldots, q_{k-1}$.

Classical Gram-Schmidt (CGS) procedure, cont'd$_2$



Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

Classical Gram-Schmidt (CGS) procedure, cont'd$_3$

- So the CGS algorithm is implemented as follows:

1. $r_{11} := \|a_1\|_2$; $q_1 := a_1/r_{11}$
2. For each $k = 2, \ldots, n$:
3. $\quad R_{1:k-1,k} := Q_{:,1:k-1}^T a_k$ // BLAS 2
4. $\quad q_k := a_k - Q_{:,1:k-1} R_{1:k-1,k}$ // BLAS 2
5. $\quad r_{kk} := \|q_k\|_2$; $q_k := q_k/r_{kk}$

so that CGS relies on two BLAS 2 calls per iteration.

Similarly, we can write

1. $r_{11} := \|a_1\|_2$; $q_1 := a_1/r_{11}$
2. For each $k = 2, \ldots, n$:
3. $\quad q_k := \Pi_{k-1} a_k$
4. $\quad r_{kk} := \|q_k\|_2$; $q_k := q_k/r_{kk}$

where $\Pi_{k-1} := I_m - Q_{:,1:k-1} Q_{:,1:k-1}^T$ is an **orthogonal projector** onto the subspace $\mathrm{range}(Q_{:,1:k-1})^\perp$ so, indeed, q_k is made orthogonal to the previously formed vectors $q_1, \ldots, q_{k-1}$.

Instability of CGS

- CGS is known to **not** being very **stable**.
- For example, consider the matrix $A = \begin{bmatrix} 1 & 1 & 1 \\ \varepsilon & 0 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & \varepsilon \end{bmatrix}$.

  If we assume ε^2 is smaller than the unit roundoff u, then the Q matrix generated by CGS is $Q = \begin{bmatrix} 1 & 0 & 0 \\ \varepsilon & -1/\sqrt{2} & -1/\sqrt{2} \\ 0 & 1/\sqrt{2} & 0 \\ 0 & 0 & 1/\sqrt{2} \end{bmatrix}$.

  Then we see that q_2 and q_3 are far from being orthogonal as we have $q_2^T q_3 = 1/2$.
- With CGS, the loss of orthogonality depends on A, i.e., $\mathrm{LOO}(Q) = \mathcal{O}(u \cdot \kappa^{n-1}(A))$, irrespective of $\kappa(A)$, although this bound is not sharp.

Re-orthogonalization, CGS2

- An alternative is to orthogonalize twice by CGS, leading to CGS2:

1. $r_{11} := \|a_1\|_2$; $q_1 := a_1/r_{11}$
2. For each $k = 2, \ldots, n$:
3. $\quad q_k := \Pi_{k-1} a_k$
4. $\quad q_k := \Pi_{k-1} q_k$
5. $\quad r_{kk} := \|q_k\|_2$; $q_k := q_k/r_{kk}$

where $\Pi_{k-1} := I_m - Q_{:,1:k-1} Q_{:,1:k-1}^T$.

1. $r_{11} := \|a_1\|_2$; $q_1 := a_1/r_{11}$
2. For each $k = 2, \ldots, n$:
3. $\quad R_{1:k-1,k} := Q_{:,1:k-1}^T a_k$
4. $\quad q_k := a_k - Q_{:,1:k-1} R_{1:k-1,k}$
5. $\quad S_{1:k-1} := Q_{:,1:k-1}^T q_k$
6. $\quad q_k := q_k - Q_{:,1:k-1} S_{1:k-1}$
7. $\quad r_{kk} := \|q_k\|_2$; $q_k := q_k/r_{kk}$

- The loss of orthogonality becomes $\mathrm{LOO}(Q) = \mathcal{O}(u)$ under the assumption that $\kappa(A) = \mathcal{O}(u^{-1})$.
- However, CGS2 requires $4mn^2$ FLOPs instead of $2mn^2$ for CGS.

Modified Gram-Schmidt, MGS

- Another alternative to CGS, referred to as modified Gram-Schmidt (MGS), is obtained by letting $\Pi_{k-1} := (I_m - q_{:,k-1}q_{:,k-1}^T)\dots(I_m - q_{:,1}q_{:,1}^T)$ in

1. $r_{11} := \|a_1\|_2$; $q_1 := a_1/r_{11}$
2. For each $k = 2, \dots, n$:
3. $\quad q_k := \Pi_{k-1}a_k$
5. $\quad r_{kk} := \|q_k\|_2$; $q_k := q_k/r_{kk}$

Assuming perfect arithmetic, this is equivalent to CGS, but it relies on BLAS 1 instead BLAS 2 operations:

1. $r_{11} := \|a_1\|_2$; $q_1 := a_1/r_{11}$
2. For each $k = 2, \dots, n$:
3. $\quad q_k := a_k$
4. $\quad$ For each $\ell = 1, \dots, k-1$:
3. $\quad\quad r_{\ell k} := q_\ell^T q_k$ // BLAS 1
4. $\quad\quad q_k := q_k - r_{\ell k}q_\ell$ // BLAS 1
5. $\quad r_{kk} := \|q_k\|_2$; $q_k := q_k/r_{kk}$

Modified Gram-Schmidt, MGS, cont'd

- The loss of orthogonality of MGS is $\mathrm{LOO}(Q) = \mathcal{O}(u \cdot \kappa(A))$, irrespective of $\kappa(A)$, so that it is more stable than CGS.
- Considering once again the matrix $A = \begin{bmatrix} 1 & 1 & 1 \\ \varepsilon & 0 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & \varepsilon \end{bmatrix}$ where $\varepsilon^2 < u$,

  MGS yields a Q matrix $A = \begin{bmatrix} 1 & 0 & 0 \\ \varepsilon & -1/\sqrt{2} & -1/\sqrt{6} \\ 0 & 1/\sqrt{2} & -1/\sqrt{6} \\ 0 & 0 & \sqrt{2}/\sqrt{3} \end{bmatrix}$.

  Contrarily to CGS, we see that q_2 and q_3 are exactly orthogonal, i.e., $q_2^T q_3 = 0$, and q_1 is nearly orthogonal to q_2 and q_3, with $|q_1^T q_2| = \varepsilon/\sqrt{2}$ and $|q_1^T q_3| = \varepsilon/\sqrt{6}$.
- In practice, MGS and CGS2 are used instead of CGS.
- MGS is often preferred by default, but CGS2 is more stable and reaches higher arithmetic intensity.