Numerical Linear Algebra for Computational Science and Information Engineering

Orthogonalization and Least-Squares Problems

Nicolas Venkovic
nicolas.venkovic@tum.de

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

Outline I

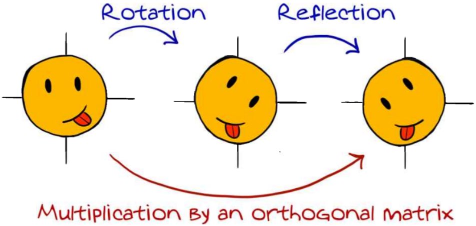
QR factorization

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 $Q^TQ = I$.
- 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.

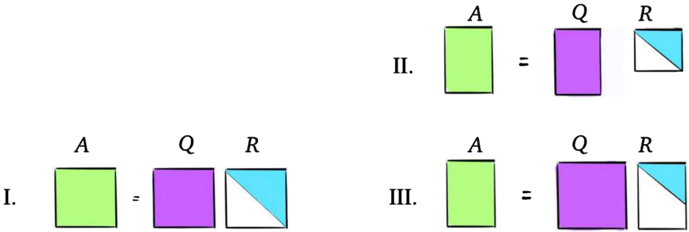
QR factorization, cont'd

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, $\|Q^T y\|_2 = \|y\|_2$ because

$$\|Q^T y\|_2^2 = (Q^T y)^T (Q^T y) = y^T Q Q^T y = y^T (Q^T Q)^T y = y^T I^T y = y^T y = \|y\|_2^2.$$

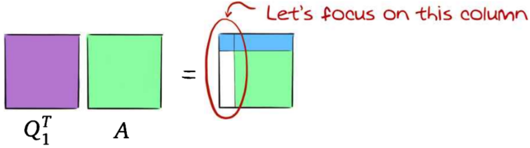
As it turns out, it is easier to find x that minimizes $\|Rx - Q^T b\|_2$ than it is to minimize $\|Ax - b\|_2$.

Householder reflections

Section 4.1 in Darve & Wootters (2021)

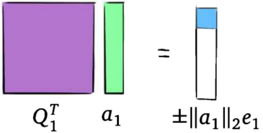
Householder reflections

- 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:

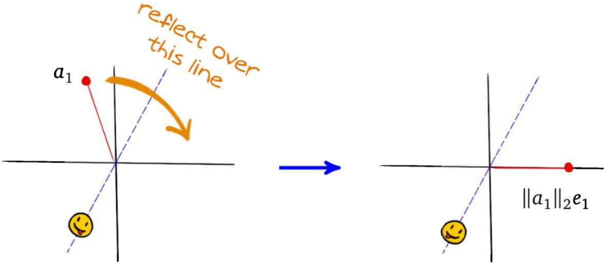


Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

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'll instead choose Q_1^T to be a **reflection** that maps a_1 parallel to e_1:

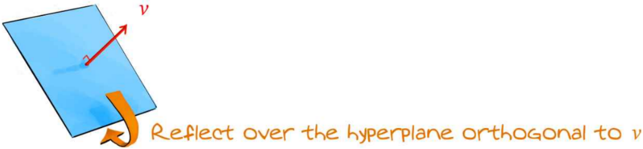


Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

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

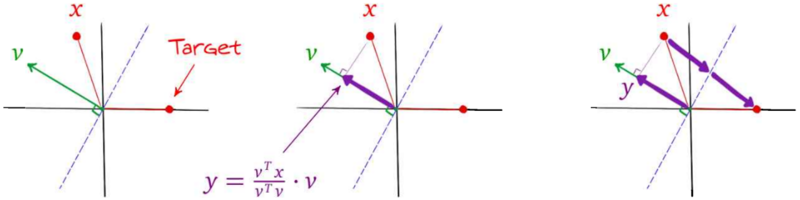
Householder reflections, cont'd$_2$

- Let us consider reflections in general. A reflection is defined by a vector:



Reflect over the hyperplane orthogonal to v

Given v, we can reason geometrically about what a reflection is:



Say we want to reflect x with respect to the hyperplane orthogonal to v

Consider the projection y of x onto v

Then the target is $x - 2y = \left(I - \frac{2vv^T}{v^T v}\right)x$

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

Householder reflections, cont'd$_3$

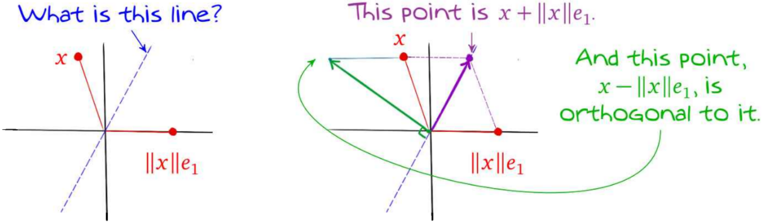
Reflection

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


What is this line?
x
$\|x\|e_1$
This point is $x+\|x\|e_1$.
x
And this point, $x-\|x\|e_1$, is orthogonal to it.
$\|x\|e_1$

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

$$\begin{aligned}
Px &= \left(I - 2\frac{vv^T}{v^Tv}\right)x \\
&= \left(I - 2\frac{(x-\|x\|_2e_1)(x-\|x\|_2e_1)^T}{(x-\|x\|_2e_1)^T(x-\|x\|_2e_1)}\right)x \\
&= \left(I - 2\frac{(x-\|x\|_2e_1)(x-\|x\|_2e_1)^T}{2(\|x\|_2^2 - 2\|x\|_2x_1)}\right)x \\
&= x - \frac{(x-\|x\|_2e_1)(\|x\|_2^2 - \|x\|_2x_1)}{(\|x\|_2^2 - 2\|x\|_2x_1)} \\
&= x - (x - \|x\|_2e_1) \\
&= \|x\|_2e_1
\end{aligned}$$

so that, indeed, a reflection over the hyperplane orthogonal to $v = x - \|x\|_2e_1$, 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

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

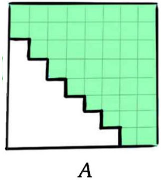
  where $Q = Q_1 \dots Q_{n-1} \in \mathbb{R}^{m \times m}$, and R has zeros in the $m - n$ rows if $m > n$.
- In practice, when reducing A to R, the matrix P is never formed explicitly, instead we compute $PA = A - \beta v(v^T A)$ which carries a cost $\mathcal{O}(2mn)$, instead of $\mathcal{O}(m^2 n)$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

Givens rotations

Section 4.2 in Darve & Wootters (2021)

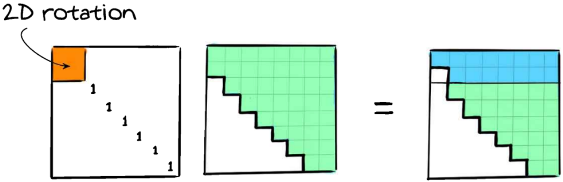
Givens rotations

- 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, and using Householder transformations in this situation is a bit of an overkill:



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

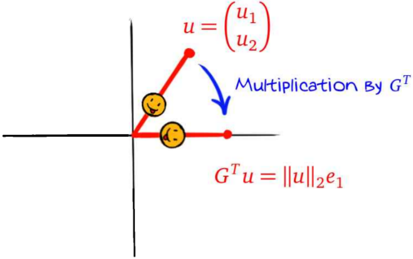
- On every column of A, only one entry needs to be zeroed, so that 2D rotations, which are easy to set up, can be deployed for the job:



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

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

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 $j = 1, \dots, n-1$ :
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^TA, then retrieving the Q factor by forward substitution:

  The algorithm is as follows:

  1. $X := A^TA$ // BLAS 3
  2. Find the upper triangular R s.t. $R^TR = X$ // 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

$$\text{LOO}(Q) \in \mathcal{O}(u \cdot \kappa(A)).$$

Reorthogonalized variants

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

  Repeating CholeskyQR significantly improves orthogonality, yielding LOO$(Q) \in \mathcal{O}(u)$ under the condition that $\kappa(A) \in \mathcal{O}(u^{-1/2})$.
- 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 $\kappa(Q) \in \mathcal{O}(u^{-1/2})$ as long as $\kappa(A) \in \mathcal{O}(u^{-1})$.

Reorthogonalized variants, cont'd

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

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

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:

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

- 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 $Q = \mathrm{diag}(Q_1, Q_2, \dots, Q_p)\tilde{Q}$.

- **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 $(Q_i, R_i) = \text{QR}(A_i)$ 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 $(\tilde{Q}, \tilde{R}) = \text{QR}(\tilde{A})$
  5. If only R is needed, return $\tilde{R}$. Otherwise, $Q = \text{diag}(Q_1, \ldots, Q_p)\tilde{Q}$
- The loss of orthogonality of TSQR is such that $\text{LOO}(Q) \in \mathcal{O}(u)$ 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.

$\implies$ 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.

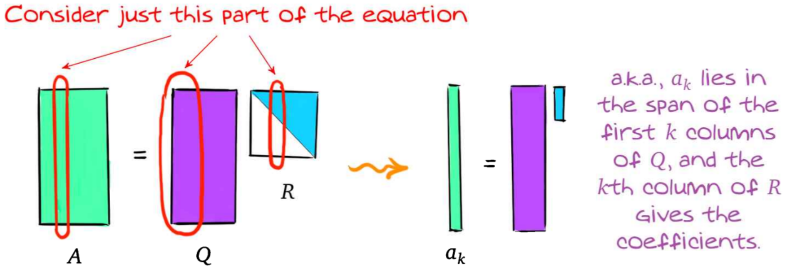
- Another peculiarity of **Gram-Schmidt** procedures is that they **work column-by-column**, i.e., to compute q_i in $Q = [q_1, \dots, q_n]$, you only need access to a_i from $A = [a_1, \dots, a_n]$ and $q_1, \dots, q_{k-1}$.

  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 an performing a full iteration of computations.

- The first k columns $q_1, \dots, q_k$ formed by Gram-Schmidt procedure in Q are an **orthonormal basis** of the subspace spanned by $a_1, \dots, a_k$.

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 $a_k = Q_{:,1:k}R_{1:k,k}$:


Consider just this part of the equation
A
=
Q
R
a_k
=
a.k.a., a_k lies in the span of the first k columns of Q, and the kth column of R gives the coefficients.

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

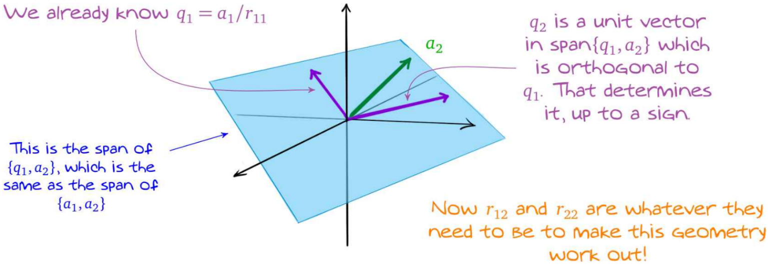
That is, a_k is formed by linear combination of $q_1, \dots, q_k$:

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

- Thus, instead of searching for the matrix Q that makes Q^TA upper triangular, we are rather going to search for the upper triangular matrix R such that every a_k is given by linear combination of $q_1, \dots, q_k$.

Gram-Schmidt procedures, cont'd$_2$

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


We already know $q_1 = a_1/r_{11}$
a_2
q_2 is a unit vector in span$\{q_1, a_2\}$ which is orthogonal to q_1. That determines it, up to a sign.
This is the span of $\{q_1, a_2\}$, which is the same as the span of $\{a_1, a_2\}$
Now r_{12} and r_{22} are whatever they need to be to make this geometry work out!

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 $q_1, \dots, q_k$, we get

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

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

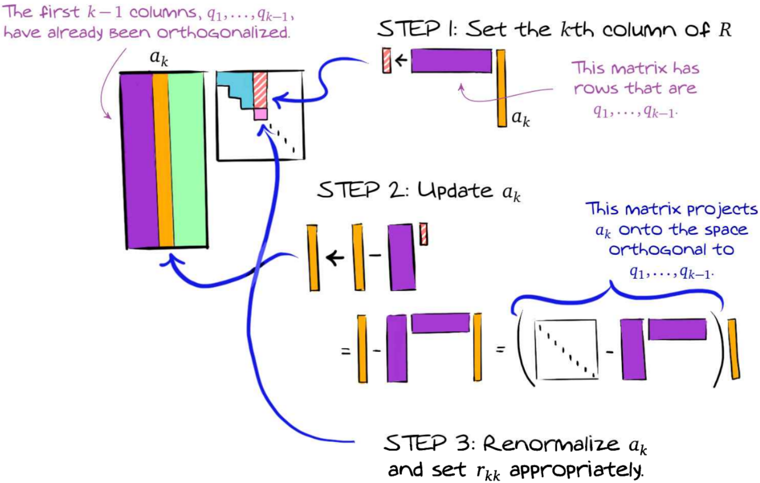
$$q_k = \frac{1}{r_{kk}} \left(a_k - \sum_{i=1}^{k-1} r_{ik} q_i \right).$$

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 $q_1, \dots, q_{k-1}$.

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

When we get the kth column of A:



The first $k-1$ columns, $q_1,\ldots,q_{k-1}$, have already been orthogonalized.

STEP 1: Set the kth column of R

This matrix has rows that are $q_1,\ldots,q_{k-1}$.

STEP 2: Update a_k

This matrix projects a_k onto the space orthogonal to $q_1,\ldots,q_{k-1}$.

STEP 3: Renormalize a_k and set r_{kk} appropriately.

Now we've turned a_k into $q_k \perp q_1,\ldots,q_{k-1}$! Move on to the next column.

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:

```
$r_{11} := \|a_1\|_2$; $q_1 := a_1/r_{11}$
For each $k = 2, \dots, n$ :
    $R_{1:k-1,k} := Q_{:,1:k-1}^T a_k$ // BLAS 2
    $q_k := a_k - Q_{:,1:k-1} R_{1:k-1,k}$ // BLAS 2
    $r_{kk} := \|q_k\|_2$; $q_k := q_k/r_{kk}$
```

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

Similarly, we can write

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

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 $q_1, \dots, q_{k-1}$.

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 $q_2^T q_3 = 1/2$.
- Numerical stability is measured with respect to the loss of orthogonality, LOO, defined by LOO$(Q) := \|I_m - Q^T Q\|_2$.
- With CGS, the LOO depends on A, i.e., $\mathrm{LOO} \in \mathcal{O}(u \cdot \kappa^{n-1}(A))$, irrespective of $\kappa(A)$, although this bound is not sharp.

Re-orthogonalization, CGS2

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

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

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

```
$r_{11} := \|a_1\|_2$; $q_1 := a_1/r_{11}$
For each $k = 2, \dots, n$ :
    $R_{1:k-1,k} := Q_{:,1:k-1}^T a_k$
    $q_k := a_k - Q_{:,1:k-1} R_{1:k-1,k}$
    $S_{1:k-1} := Q_{:,1:k-1}^T q_k$
    $q_k := q_k - Q_{:,1:k-1} S_{1:k-1}$
    $r_{kk} := \|q_k\|_2$; $q_k := q_k/r_{kk}$
```

- The loss of orthogonality becomes $\mathrm{LOO}(Q) \in \mathcal{O}(u)$ under the assumption that $\kappa(A) \in \mathcal{O}(u^{-1})$.
- 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

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

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

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

Modified Gram-Schmidt, MGS, cont'd

- The loss of orthogonality of MGS is $\mathrm{LOO}(Q) \in \mathcal{O}(u \cdot \kappa(A))$, 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 $A = \begin{bmatrix} 1 & 0 & 0 \\ \varepsilon & -1/\sqrt{2} & -1/\sqrt{6} \\ 0 & 1/\sqrt{2} & -1/\sqrt{6} \\ 0 & 0 & \sqrt{2}/\sqrt{3} \end{bmatrix}$.

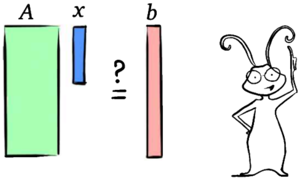
  Contrarily to CGS, we see that q_2 and q_3 are exactly orthogonal, i.e., $q_2^T q_3 = 0$, and q_1 is nearly orthogonal to q_2 and q_3, with $|q_1^T q_2| = \varepsilon/\sqrt{2}$ and $|q_1^T q_3| = \varepsilon/\sqrt{6}$.
- In practice, MGS and CGS2 are used instead of CGS.
- MGS is often preferred by default,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

Least-squares problems

Section 4.4 in Darve & Wootters (2021)

Geometry of least-squares

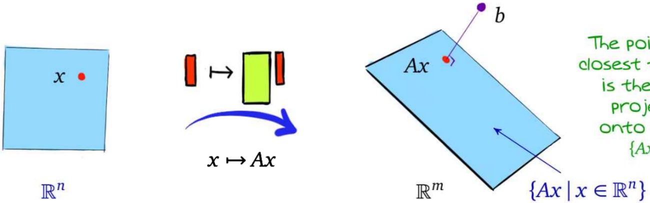
- One of the applications of the QR decomposition is the solving of least-squares problems $\arg\min_x \|Ax - b\|_2$:



Since A is tall and skinny, there are more equations than unknowns! So there may not be some x so that $Ax = b$. The least-squares problem is to find the x so that Ax is closest to b, in 2-norm.

with a tall-and-skinny matrix $A \in \mathbb{R}^{m\times n}$ and a vector $b \in \mathbb{R}^n$.

- To minimize the 2-norm from a point b to a subspace $\{Ax,\ x \in \mathbb{R}^n\}$, we can just do an orthogonal projection:



The point Ax which is closest to b in 2-norm is the orthogonal projection of b onto the subspace $\{Ax : x \in \mathbb{R}^n\}$.

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

Method of normal equations

- As per property of orthogonal projections, the x that minimizes $\|Ax-b\|_2$ has an error $e := Ax - b$ which is orthogonal to the range of A. This can be written as

  $$A^T(Ax-b) = 0. \tag{1}$$

  Assuming A is full-rank, this equation can be used to solve for x by a method called **normal equations**.

  Eq. (1) may also be derived from calculus, namely, the optimal x which minimizes the cost function

  $$f(x) = \|Ax-b\|_2^2 = (Ax-b)^T(Ax-b) = x^TA^TAx - 2x^TA^Tb + b^Tb$$

  is obtained for $\nabla f(x) = 0$ where

  $$\nabla f(x) = 2A^TAx - 2A^Tb,$$

  which equivalently yields Eq. (1).

Method of normal equations, cont'd

- Assuming A is full-rank, $A^T A$ is SPD so that we may compute its Cholesky factorization and solve for x in $A^T A x = A^T b$.

Normal equations

Finding the solution x to the least-suqares problem $\arg\min \|Ax - b\|_2$ by solving the system $A^T A x = A^T b$ is called the method of **normal equations**.

- Since the condition number of $A^T A$ is the square of that of A, the method of normal equations can run into issues when A is poorly conditioned.
- For cases where A is poorly conditioned, the QR factorization can be used to yield a more accurate computation of the solution x to the least-squares problem.

QR factorization for least-squares problems

- The origin of the method of normal equations stems from saying that the error $Ax - b$ is orthogonal to the range of A.

  But if we know a QR factorization $A = QR$ where $Q \in \mathbb{R}^{m \times n}$, then the range of A is the same as the range of Q.

  The orthogonality condition can then be re-stated as

  $$Q^T(Ax - b) = 0. \tag{2}$$

  Since Q is orthogonal, it is necessarily well-conditioned, and the conditioning problem of the method of normal equations can be avoided.

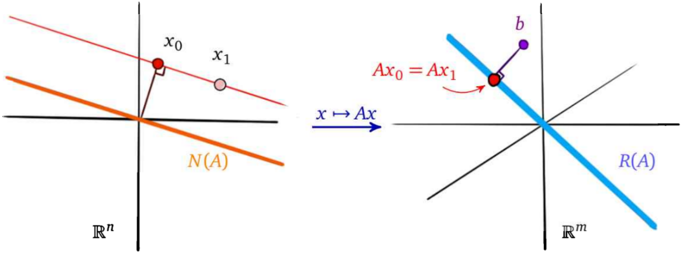
  Since $A = QR$, due to the orthogonality of Q, we have $Q^T A = R$ so that Eq. (2) becomes

  $$Rx = Q^T b$$

  where R is non-singular as long as A is non-singular, so that there exists a unique solution x.

Case of rank-deficient A

- If A is rank-deficient, the null space of A is non-trivial. Then, for some x that minimizes $\|Ax-b\|_2$, there are infinitely many $\delta x \in \mathrm{null}(A)$ such that $A(x+\delta x)=Ax$. Hence, the solution to the least-squares problem is not unique.
- In case of non-uniqueness of solution, one can search for the unique x_0 which minimizes both $\|Ax-b\|_2$ and $\|x\|_2$:



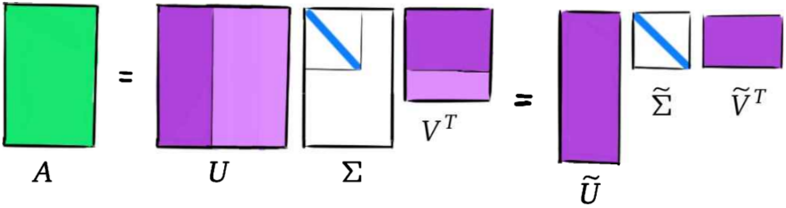
Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

We can see the x_0 we are after is orthogonal to the null space of A, while any other solution x_1 is of the form $x_0+\delta x$.

SVD method for solving least-squares with rank-deficient A

- Let $A \in \mathbb{R}^{m\times n}$ be of rank $r < n < m$ have an SVD given by $A = U\Sigma V^T$ with $U \in \mathbb{R}^{m\times m}, V \in \mathbb{R}^{n\times n}$ and $\Sigma \in \mathbb{R}^{m\times n}$ where Σ has zeros from row $r+1$ to m.

  Then we can ignore the columns of U and V that correspond to zeros in Σ to create the thin SVD $A = \widetilde{U}\widetilde{\Sigma}\widetilde{V}^T$:



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

- Now, $Ax = 0$ if and only if $\widetilde{V}^T x = 0$, which means that the null space of A is the same as that of $\widetilde{V}^T$, i.e., $\text{null}(A) = \text{null}(\widetilde{V}^T)$.

SVD method for solving least-squares with rank-deficient A

- We know that any solution x to the least-squares problem satisfies

$$
\begin{aligned}
A^T A x &= A^T b \\
(\widetilde{U}\widetilde{\Sigma}\widetilde{V}^T)^T \widetilde{U}\widetilde{\Sigma}\widetilde{V}^T x &= (\widetilde{U}\widetilde{\Sigma}\widetilde{V}^T)^T b \\
\widetilde{V}\widetilde{\Sigma}^T\widetilde{\Sigma}\widetilde{V}^T x &= \widetilde{V}\widetilde{\Sigma}^T\widetilde{U}^T b \\
\widetilde{\Sigma}\widetilde{V}^T x &= \widetilde{U}^T b
\end{aligned}
$$

  where $r < n$ so that $\widetilde{\Sigma}\widetilde{V}^T$ is not full-column-rank and this equation admits infinitely many solutions.

- However, we can find one solution as follows.

  First, let's solve the system $\widetilde{\Sigma}\omega = \widetilde{U}^T b$ for $\omega \in \mathbb{R}^r$. This gives

$$
\omega_i = \frac{\widetilde{u}_i^T b}{\widetilde{\sigma}_{ii}}
$$

  where $\widetilde{U} = [\widetilde{u}_1, \dots, \widetilde{u}_r]$ and $\widetilde{\Sigma} = \operatorname{diag}(\widetilde{\sigma}_{11}, \dots, \widetilde{\sigma}_{rr})$.

SVD method for solving least-squares with rank-deficient A

Then, since $\omega = \widetilde{\Sigma}^{-1}\widetilde{U}^T b$, we have

$$\widetilde{\Sigma}\widetilde{V}^T(\widetilde{V}\omega) = \widetilde{\Sigma}\widetilde{V}^T\widetilde{V}\tilde{\Sigma}^{-1}\widetilde{U}^T b = \widetilde{U}^T b$$

so that $x_0 := \widetilde{V}\omega$ is solution of $\widetilde{\Sigma}\widetilde{V}^T x_0 = \widetilde{U}^T b$ and thus, as explained before, it is also solution of the least-squares problem.

Note that $x_0 := \widetilde{V}\omega$ is the solution with smallest norm.

To see this, we need to show $x_0 \perp \mathrm{null}(A)$. First, let

$$\mathrm{null}(\widetilde{V}^T) = \{y \in \mathbb{R}^n,\ \widetilde{V}^T y = 0\}$$

and consider that for each $y \in \mathrm{null}(\widetilde{V}^T)$, we have

$$x_0^T y = (\widetilde{V}\omega)^T y = \omega\widetilde{V}^T y = 0$$

so that $x_0 \perp \mathrm{null}(\widetilde{V}^T)$.

But since $\mathrm{null}(\widetilde{V}^T) = \mathrm{null}(A)$, we have that $x_0 \perp \mathrm{null}(A)$.

LSQR

Iterative solve of normal equations

- We saw that the least-squares solution x_* of $\min_{x\in\mathbb{R}^n} \|Ax-b\|_2$ for $A\in\mathbb{R}^{m\times n}$ and $b\in\mathbb{R}^m$ with $m>n$ is such that $A^TAx_*=A^Tb$, i.e., x_* is solution of the normal equation.
- For very large matrices, the cost of computing a QR factorization by Householder QR, or even by CholeskyQR, can be prohibitive. When the matrix is sparse, computing a QR factorization is generally an overkill.
- In future lectures, we will look into iterative methods to solve linear systems of the form $Bx=b$ with a square matrix $B\in\mathbb{R}^{n\times n}$.

  In particular, if A is full-rank, one can use the conjugate gradient algorithm to solve $A^TAx_*=b$.

  However, in practice, for cases where A is ill-conditioned, this approach can suffer from significantly delayed convergence.
- **LSQR** is an algorithm proposed by Paige and Sanders (1982) which, in case of exact arithmetic, reproduces the iterates of the conjugate gradient algorithm applied to the normal equation but, in practice, is **more reliable**.

Paige, C. C., & Saunders, M. A. (1982). LSQR: An algorithm for sparse linear equations and sparse least squares. ACM Transactions on Mathematical Software (TOMS), 8(1), 43-71.

Bidiagonalization

- Bidiagonalization is a procedure proposed by Golub and Kahan (1965) which reduces any general matrix $A \in \mathbb{R}^{m\times n}$ into lower bidiagonal form. Let $x_0 \in \mathbb{R}^n$ be an initial approximation of x_* with residual $r_0 := b - Ax_0$. Starting the bidiagonalization procedure with r_0 goes as follows:

$$\beta_1 u_1 = r_0,\ \alpha_1 v_1 = A^T u_1$$

$$\left.\begin{aligned}\beta_{i+1}u_{i+1} &= Av_i - \alpha_i u_i\\ \alpha_{i+1}v_{i+1} &= A^T u_{i+1} - \beta_{i+1}v_i\end{aligned}\right\}\text{for } i = 1, 2\ldots$$

where the scalars $\alpha_i \geq 0$ and $\beta_i \geq 0$ are chosen so that $\|u_i\|_2 = \|v_i\|_2 = 1$. Let $U_k := [u_1, \ldots, u_k]$, $V_k := [v_1, \ldots, v_k]$,

$$B_k := \begin{bmatrix}\alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_k & \alpha_k\end{bmatrix} \text{ and } \underline{B}_k := \begin{bmatrix}\alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_k & \alpha_k \\ & & & \beta_{k+1}\end{bmatrix}.$$

Golub, G., & Kahan, W. (1965). Calculating the singular values and pseudo-inverse of a matrix. Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis, 2(2), 205-224.

Paige, C. C. (1974). Bidiagonalization of matrices and solution of linear equations. SIAM Journal on Numerical Analysis, 11(1), 197-209.

Bidiagonalization, cont'd

Then, we have

$$U_{k+1}(\beta_1 e_1) = r_0 \qquad (e_1 := I_{k+1}[:,1])$$

$$AV_k = U_{k+1}\underline{B}_k = U_k B_k + \beta_{k+1} u_{k+1} e_k^T \qquad (e_k := I_k[:,k])$$

$$A^T U_{k+1} = V_k \underline{B}_k^T + \alpha_{k+1} v_{k+1} e_{k+1}^T. \qquad (e_{k+1} := I_{k+1}[:,k+1])$$

In exact arithmetic, we have $U_k^T U_k = I_k$ and $V_k^T V_k = I_k$.
Clearly, $U_{k+1}^T A V_k = \underline{B}_k$, $U_k^T A V_k = B_k$, $\alpha_k = u_k^T A v_k$ and $\beta_k = u_k^T A v_{k-1}$.
Moreover, the columns of U_k and V_k are orthonormal bases of Krylov subspaces of AA^T and $A^T A$, respectively, i.e.,

$$\text{range}(U_k) = \mathcal{K}_k(AA^T, u_1) = \text{span}\left\{u_1, AA^T u_1, \dots, (AA^T)^{k-1} u_1\right\},$$

$$\text{range}(V_k) = \mathcal{K}_k(A^T A, v_1) = \text{span}\left\{v_1, A^T A v_1, \dots, (A^T A)^{k-1} v_1\right\}.$$

Bidiagonalization is an orthogonal equivalence transformation which plays a key role in the iterative solve of singular value decompositions.

Golub, G., & Kahan, W. (1965). Calculating the singular values and pseudo-inverse of a matrix. Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis, 2(2), 205-224.

Paige, C. C. (1974). Bidiagonalization of matrices and solution of linear equations. SIAM Journal on Numerical Analysis, 11(1), 197-209.

LSQR

- LSQR defines a sequence of iterates $x_1, x_2, \ldots, x_k$ which approximate the solution x_* of $\min_{x \in \mathbb{R}^n} \|b - Ax\|_2$ by

  $$x_k \in x_0 + \mathcal{K}_k(A^T A, v_1).$$

  That is, we search for $x_k = x_0 + V_k y_k$ such that

  $$x_k = \underset{x \in x_0 + \text{range}(V_k)}{\arg\min} \|b - Ax\|_2 \implies y_k = \underset{y \in \mathbb{R}^k}{\arg\min} \|b - A(x_0 + V_k y)\|_2$$
  $$= \underset{y \in \mathbb{R}^k}{\arg\min} \|r_0 - U_{k+1} \underline{B}_k y\|_2$$
  $$= \underset{y \in \mathbb{R}^k}{\arg\min} \|U_{k+1}(\beta_1 e_1) - U_{k+1} \underline{B}_k y\|_2$$
  $$= \underset{y \in \mathbb{R}^k}{\arg\min} \|\beta_1 e_1 - \underline{B}_k y\|_2.$$

  In exact arithmetic, the LSQR iterates exhibit monotonic decrease of residual norm, i.e., $\|r_{k+1}\|_2 \leq \|r_k\|_2$.

LSQR, cont'd

- A basic implementation of the LSQR algorithm goes as follows:

```
$r_0 := b - Ax_0$
$u_1 := r_0$, $\beta_1 := \|u_1\|_2$, $u_1 := u_1/\beta_1$
$v_1 := A^T u_1$, $\alpha_1 := \|v_1\|_2$, $v_1 := v_1/\alpha_1$
for $i = 1, 2, \dots$
  $u_{i+1} := Av_i - \alpha_i u_i$, $\beta_{i+1} := \|u_{i+1}\|_2$, $u_{i+1} := u_{i+1}/\beta_{i+1}$
  $v_{i+1} := A^T u_{i+1} - \beta_{i+1} v_i$, $\alpha_{i+1} := \|v_{i+1}\|_2$, $v_{i+1} := v_{i+1}/\alpha_{i+1}$
  $y_i := \underset{y \in \mathbb{R}^i}{\arg\min} \|\beta_1 e_1 - \underline{B}_i y\|_2$
```

When convergence is achieved, the iterate $x_i := x_0 + V_i y_i$ is formed.

Convergence monitoring is reliant on the evaluation of $\|r_i\|_2$ and $\|A^T r_i\|_2$. Note that we have

$$\|r_i\|_2 = \|b - A(x_0 + V_i y_i)\|_2 = \|\underline{t}_i\|_2 \text{ where } \underline{t}_i := \beta_1 e_1 - \underline{B}_i y_i$$
$$\|A^T r_i\|_2 = \|A^T U_{i+1} \underline{t}_i\|_2 = \|(V_i \underline{B}_i^T + \alpha_{i+1} v_{i+1} e_{i+1}^T) \underline{t}_i\|_2 = \|B_{i+1}^T \underline{t}_i\|_2$$

so that convergence can be monitored without forming neither x_i nor r_i.