Numerical Matrix Analysis

Notes #7 — The QR-Factorization and Least Squares Problems: Gram-Schmidt and Householder

Peter Blomgren
⟨blomgren@sdsu.edu⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://terminus.sdsu.edu/

Spring 2024
(Revised: February 19, 2024)

Outline

Student Learning Targets, and Objectives

Target Gram-Schmidt Orthogonalization
- **Objective** "Classical" vs. Modified
- **Objective** Mathematically Equivalent
- **Objective** Numerically Divergent

Target Quatifying Compuational "Speed"
- **Objective** Computational Complexity

Target Orthogonalization Alternatives
- **Objective** Householder Reflections
- **Objective** (Givens Rotations)

Last Time (Projections; Classical Gram-Schmidt)

Orthogonal and non-orthogonal projectors

$$P = P^2, \qquad \left[P = P^* \right].$$

Projection with an orthonormal, and arbitrary, basis

$$P = \widehat{Q}\widehat{Q}^*, \qquad P = A(A^*A)^{-1}A^*.$$

Rank-one projections, rank-$(m-1)$ complementary projections

$$P = \vec{q}\vec{q}^*, \qquad P_\perp = I - \vec{q}\vec{q}^*.$$

QR-Factorization, using classical Gram-Schmidt orthogonalization.

Algorithm: Classical Gram-Schmidt — ∃ Movies

Algorithm (Classical Gram-Schmidt)

```
for k ∈ {1,...,n} do
    v_k ← a_k
    for i ∈ {1,...,k−1} do
        r_ik ← q_i^* a_k            /* projection */
        v_k ← v_k − r_ik q_i        /* projection */
    end for
    r_kk ← ||v_k||_2
    q_k ← v_k / r_kk
end for
```

Mathematically, we are done. Numerically, however, we can run into trouble due to roundoff errors.

Classical Gram-Schmidt: Revisited ⇝ The Modified Gram-Schmidt Method

Let $A \in \mathbb{C}^{m\times n}$, $m \geq n$, be a full-rank matrix with columns $\vec{a}_k$. With orthogonal projectors P_k we can express the Gram-Schmidt orthogonalization using the formulas

$$\vec{q}_k = \frac{P_k \vec{a}_k}{\|P_k \vec{a}_k\|_2}, \quad k = 1, \dots, n$$

The projector P_k must be an $(m \times m)$-matrix of rank $(m-(k-1))$ which projects the space $\mathbb{C}^m$ orthogonally onto the space orthogonal to $\mathrm{span}(\vec{q}_1, \dots, \vec{q}_{k-1})$. $(P_1 = I)$.

Note: $\vec{q}_k \in \mathrm{span}(\vec{a}_1, \dots, \vec{a}_k)$ and $\vec{q}_k \perp \mathrm{span}(\vec{q}_1, \dots, \vec{q}_{k-1})$; therefore this description is equivalent to the algorithm on slide 5.

We can represent the projector $P_k = (I - \widehat{Q}_{k-1}\widehat{Q}_{k-1}^*)$ where $\widehat{Q}_{k-1}$ is the $(m \times (k-1))$-matrix $[\vec{q}_1\ \vec{q}_2\ \dots\ \vec{q}_{k-1}]$.

A Hard Test Problem — Matlab-centric Notation

Let U and V be two randomly selected (80×80) unitary matrices

```
[U,∼] = qr(randn(80,80));   [V,∼] = qr(randn(80,80));
```

Build a matrix A with singular values $2^{-1}, 2^{-2}, \dots, 2^{-80}$:
(condition number — $\kappa(A) = 2^{79} \approx 10^{23}$)

```
S = diag(2.^(-1:-1:-80));   A = U * S * V';
```

Finally we compute the QR-factorization using both classical and modified Gram-Schmidt

```
[QC,RC] = qr_cgs(A);HW#3   [QM,RM] = qr_mgs(A);HW#4
```

Now, the diagonals of RC and RM contain the recovered singular values.

Burning Questions: What is the modified Gram-Schmidt method?!?
... and why do we need it?!?

Classical Gram-Schmidt: The Bad News — 1 of 2

Unfortunately, classical Gram-Schmidt is not numerically stable — in finite precision, the vectors $\vec{q}_k$ may lose orthogonality...

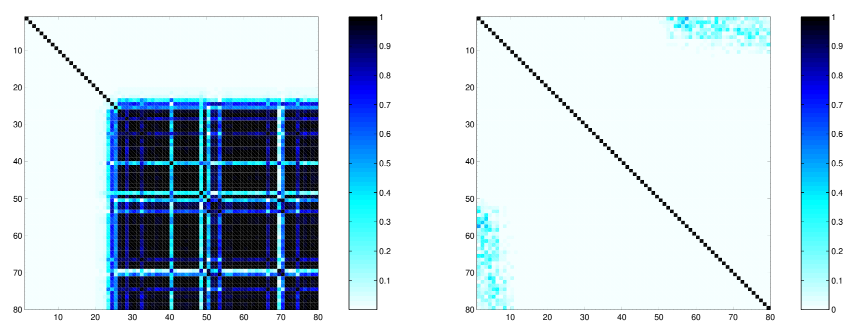


Figure: Comparing Q^*Q (which should be the identity matrix) for classical (left) and modified (right) Gram-Schmidt on a particularly hard problem where $\sigma_1 = 2^{-1}$ and $\sigma_{80} = 2^{-80}$. We see that CGS completely loses orthogonality after 20-some steps; whereas MGS does not suffer this catastrophic breakdown.

Classical Gram-Schmidt: The Bad News

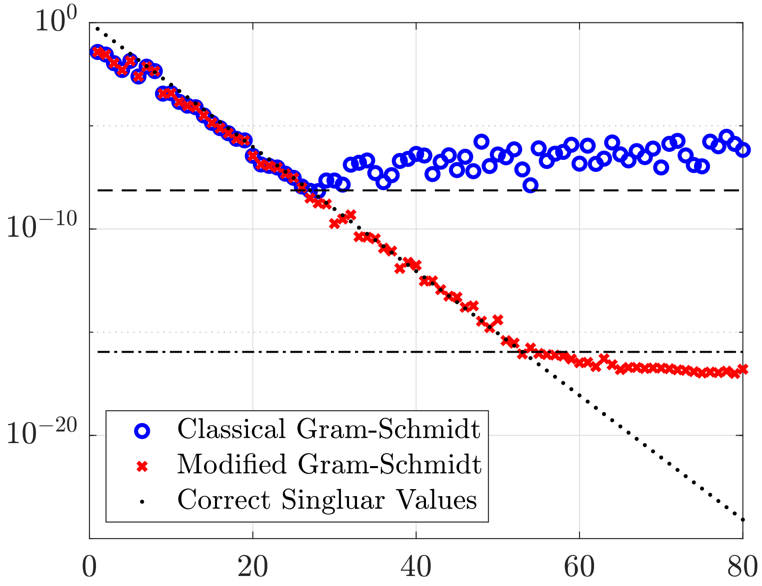


Figure: The performance of classical (**blue** **'o's**) and modified (**red** **'x's**) Gram-Schmidt on a particularly hard problem where $\sigma_1 = \frac{1}{2}$ and $\sigma_{80} = \frac{1}{2^{80}}$. C-GS identifies the first $\sim$ 26 singular values (down to the size $\sim \sqrt{\epsilon_{\text{mach}}}$), whereas M-CG identifies $\sim$ 54 singular values (down to the size $\sim \epsilon_{\text{mach}}$).

What is "Machine Epsilon," ϵ_{mach}???

Machine Epsilon is the smallest positive value for which $1.0 + \epsilon > 1.0$.

In most (double-precision / 64-bit) computational environments $\epsilon_{\text{mach}} \sim 2.22 \times 10^{-16}$, which means we can compute with AT MOST 15 significant (base-10) digits.

Algorithm (Find Machine Epsilon)

```
eps = 1.0
while (1.0 + eps > 1.0) do
    eps ← eps/2
end while
eps ← eps * 2
```


An Improvement: Modified Gram-Schmidt

For each k in classical Gram-Schmidt, we compute one orthogonal projection of rank $(m-(k-1))$:

$$\vec{v}_k = P_k \vec{a}_k.$$

Modified Gram-Schmidt computes the same — **mathematically equivalent** quantity — by a sequence of $(k-1)$ projections of rank $(m-1)$:

$$P_1 = I, \qquad P_k = P_{\perp \vec{q}_{k-1}} \cdots P_{\perp \vec{q}_1}, \quad k > 1,$$

where

$$P_{\perp \vec{q}_k} = I - \vec{q}_k \vec{q}_k^*, \quad k > 1,$$

thus

$$\tilde{\mathbf{v}}_\mathbf{k} = \mathbf{P}_{\perp \tilde{\mathbf{q}}_{\mathbf{k}-1}} \cdots \mathbf{P}_{\perp \tilde{\mathbf{q}}_1} \tilde{\mathbf{a}}_\mathbf{k}.$$

Algorithm: Modified Gram-Schmidt

Algorithm (Modified Gram-Schmidt)

for $k \in \{1, \ldots, n\}$ **do**
  $\vec{v}_k \leftarrow \vec{a}_k$
end for
for $i \in \{1, \ldots, n\}$ **do**
  $r_{ii} \leftarrow \|\vec{v}_i\|_2$
  $\vec{q}_i \leftarrow \vec{v}_i / r_{ii}$
  for $k \in \{(i+1), \ldots, n\}$ **do**
    $r_{ik} \leftarrow \vec{q}_i^* \vec{v}_k$
    $\vec{v}_k \leftarrow \vec{v}_k - r_{ik} \vec{q}_i$
  end for
end for

The ordering of the computation is the key... in step $\#i$, we make all the remaining columns orthogonal to column $\#i$.

In practice, usually we let $\vec{v}_i$ overwrite $\vec{a}_i$, in order to save storage.

We can also let $\vec{q}_i$ overwrite $\vec{v}_i$ to save additional storage.

Comparison: Modified/Classical Gram-Schmidt

Algorithm (Modified vs. Classical Gram-Schmidt)

Modified:

```
for k ∈ {1,...,n} do
    v_k ← a_k
end for
for i ∈ {1,...,n} do
    r_ii ← ||v_i||_2
    q_i ← v_i / r_ii
    for k ∈ {(i+1),...,n} do
        r_ik ← q_i^* v_k
        v_k ← v_k − r_ik q_i
    end for
end for
```

Classical:

```
for k ∈ {1,...,n} do
    v_k ← a_k
    for i ∈ {1,...,k−1} do
        r_ik ← q_i^* a_k
        v_k ← v_k − r_ik q_i
    end for
    r_kk ← ||v_k||_2
    q_k ← v_k / r_kk
end for
```

Clearly, unexpected subtle differences can have a huge impact on the result.

Recap
Gram-Schmidt
Gram-Schmidt and Householder: Different Views of QR
Bad News for the Classical Version
Improving Gram-Schmidt
I Feel the Need for Speed!!!

Why is $\vec{q}_i^*\vec{v}_k \neq \vec{q}_i^*\vec{a}_k$???

In **infinite precision**, they are the same:

$\vec{v}_k$ contains only the part of $\vec{a}_k \perp \operatorname{span}(\vec{q}_1,\dots,\vec{q}_{k-1})$, *i.e*

$$\vec{a}_k = \vec{v}_k + \vec{a}_k^{\ddagger}, \quad \text{where } \vec{a}_k^{\ddagger} \in \operatorname{span}(\vec{q}_1,\dots,\vec{q}_{k-1})$$

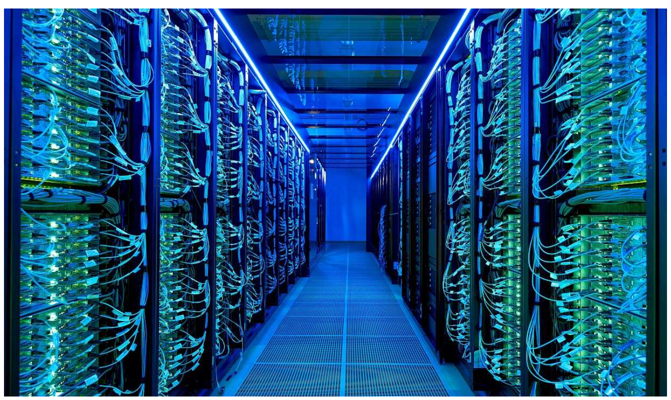
in the sense that:

$$\vec{q}_i^*\vec{a}_k = \vec{q}_i^*(\vec{v}_k + \vec{a}_k^{\ddagger}) = \vec{q}_i^*\vec{v}_k + \underbrace{\vec{q}_i^*\vec{a}_k^{\ddagger}}_{0} = \vec{q}_i^*\vec{v}_k$$

However, *numerically*, throwing out the (infinite-precision) 0 is better than "mixing in" the numerical errors from the computation of $\vec{q}_i^*\vec{a}_k^{\ddagger}$.

Recap
Gram-Schmidt
Gram-Schmidt and Householder: Different Views of QR
Bad News for the Classical Version
Improving Gram-Schmidt
I Feel the Need for Speed!!!

Counting Work: Ancient, Old, and Somewhat Recent Measures



How fast is fast???

Recap
Gram-Schmidt
Gram-Schmidt and Householder: Different Views of QR
Bad News for the Classical Version
Improving Gram-Schmidt
I Feel the Need for Speed!!!

Counting Work: Ancient, Old, and Somewhat Recent Measures

We need some measure of how fast, or slow, an algorithm is...

In **ancient times** multiplications (an divisions) where a lot slower than additions (and subtractions) $T_{*,/} \gg T_{+,-}$; so one would count the number of multiplications.

Then the chip designers figured out how to make multiplications faster, so $T_{*,/} \approx T_{+,-}$, so in the **old days** one would count all operations.

Last week, processors where so fast that **memory accesses** dominated the processing time; in particular **cache-misses**, so we end up with a completely different model... (see next slide)

Yesterday, processors suddenly had multiple cores, and hence multiple memory pathways...

This morning we have to deal with GPUs with tens of thousands of cores, FPGAs...

Counting Work: The (Single-Threaded) Memory Access Latency Model

If we have three cache-levels (L1, L2, and L3), some average hit-rate (and hence miss-rate) for each level and the time it takes to access that cache-level (the hit-cycle-time), then we end up with a measure for the average memory access latency per memory access

$$\begin{aligned} T \;\sim\; & (\texttt{L1\_hit\_rate} * \texttt{L1\_hit\_cycle\_time}) \\ & + (\texttt{L1\_miss\_L2\_hit\_rate} * \texttt{L2\_hit\_cycle\_time}) \\ & + (\texttt{L2\_miss\_L3\_hit\_rate} * \texttt{L3\_hit\_cycle\_time}) \\ & + (\texttt{L3\_miss\_rate} * \texttt{[S]DRAM\_latency}) \end{aligned}$$

If this does not scare you, please get a Ph.D. in algorithm design!

Meanwhile, the rest of us will count **"flops"**, *i.e.* floating-point operations (multiplications and additions)!

11th Generation Intel Core Cache Structure (Already Outdated)

2.4.1 Intel® Smart Cache Technology

The Intel® Smart Cache Technology is a shared Last Level Cache (LLC).

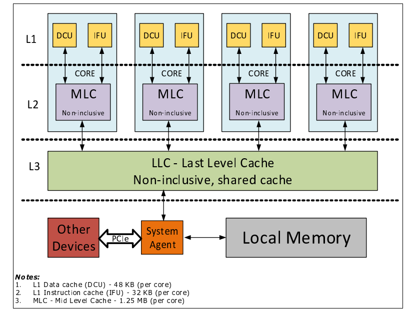
- The LLC is non-inclusive.
- The LLC may also be referred to as a 3^{rd} level cache.
- The LLC is shared between all IA cores as well as the Processor Graphics.

2.4.2 IA Core Level 1 and Level 2 Caches

The 1^{st} level cache is divided into a data cache (DFU) and an instruction cache (IFU). The processor 1^{st} level cache size is 48 KB for data and 32 KB for instructions. The 1^{st} level cache is an 8-way associative cache.

The 2^{nd} level cache holds both data and instructions. It is also referred to as mid-level cache or MLC.
The processor 2^{nd} level cache size is 1.25 MB and is a 20-way non-inclusive associative cache.

Figure 2-4. Processor Cache Hierarchy



Source: *11^{th} Generation Intel Core^{TM} Processor Datasheet, Volume 1 of 2, pages 35–36.*
`https://cdrdv2.intel.com/v1/dl/getContent/631121`

See also: https://www.intel.com/content/www/us/en/products/docs/processors/core/core-technical-resources.html

Counting Work: Gram-Schmidt Orthogonalization

Theorem (Computational Complexity of Modified Gram-Schmidt)

The modified Gram-Schmidt orthogonalization algorithm requires

$$\sim 2mn^2 \quad \textit{flops}$$

to compute the QR-factorization of an $(m \times n)$ matrix.

Here we have assumed that complex arithmetic is just as fast as real arithmetic. This is not true in general.

$$\begin{aligned} c_1 \cdot c_2 &= [r_1 \cdot r_2 - i_1 \cdot i_2] + i\,[r_1 \cdot i_2 + r_2 \cdot i_1] \\ c_1 + c_2 &= [r_1 + r_2] + i\,[i_1 + i_2] \end{aligned}$$

Hence, the complex multiplication consists of 4 real multiplications and 2 real additions; and the complex addition consists of 2 real additions. Also, we need *at least* double the amount of memory accesses.

Counting Flops

The Outer Loop: for $i \in \{1, \dots, n\}$
  The Inner Loop: for $k \in \{(i+1), \dots, n\}$
    r_{ik} is formed by an m-inner product -- requiring m multiplications and $(m-1)$ additions
    $\vec{v}_k$ requires m multiplications and m subtractions
  End Inner Loop
End Outer Loop

$$\begin{aligned} \texttt{Work} \;&\sim\; \sum_{i=1}^{n} \sum_{k=i+1}^{n} 4m \;\sim\; \sum_{i=1}^{n} 4m(n-i) \\ &\sim\; 4mn^2 - 4mn^2/2 \;\sim\; 2mn^2 \end{aligned}$$

Note that to *leading order* summation is "just like" integration:

$$\sum_{i=0}^{n} i^p \sim \frac{n^{(p+1)}}{(p+1)}$$

Exact Summation Formula — For Reference

$$\sum_{i=0}^{n} i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^{p} \frac{B_k}{p-k+1}\binom{p}{k}(n+1)^{p-k+1},$$

where B_k are Bernoulli numbers:

$$B_k(n) = \sum_{\ell=0}^{k}\sum_{\nu=0}^{\ell}(-1)^{\nu}\binom{\ell}{\nu}\frac{(n+\nu)^k}{\ell+1}.$$

Gram-Schmidt as Triangular Orthogonalization — 1 of 3

Each outer loop in the modified Gram-Schmidt algorithm can be seen as a right-multiplication by a square upper triangular matrix.

E.g. Iteration#1

$$\begin{bmatrix} | & | & & | \\ \vec{v}_1 & \vec{v}_2 & \cdots & \vec{v}_n \\ | & | & & | \end{bmatrix} \underbrace{\begin{bmatrix} \frac{1}{r_{11}} & -\frac{r_{12}}{r_{11}} & -\frac{r_{13}}{r_{11}} & \cdots \\ & 1 & & \\ & & 1 & \\ & & & \ddots \end{bmatrix}}_{R_1} = \begin{bmatrix} | & | & & | \\ \vec{q}_1 & \vec{v}_2^{(2)} & \cdots & \vec{v}_n^{(2)} \\ | & | & & | \end{bmatrix}$$

The correct triangular matrix $(R_k) \rightsquigarrow$ (one additional) orthogonal vector $(\vec{q}_k)$.

Gram-Schmidt as Triangular Orthogonalization — 2 of 3

E.g. Iteration#2

$$\begin{bmatrix} | & | & & | \\ \vec{q}_1 & \vec{v}_2^{(2)} & \cdots & \vec{v}_n^{(2)} \\ | & | & & | \end{bmatrix} \underbrace{\begin{bmatrix} 1 & & & \\ & \frac{1}{r_{22}} & -\frac{r_{23}}{r_{22}} & \cdots \\ & & 1 & \\ & & & \ddots \end{bmatrix}}_{R_2} = \begin{bmatrix} | & | & & | \\ \vec{q}_1 & \vec{q}_2 & \cdots & \vec{v}_n^{(3)} \\ | & | & & | \end{bmatrix}$$

When we are done we have

$$A\,\underbrace{R_1R_2\dots R_n}_{\widehat{R}^{-1}} = \widehat{Q} \quad \Leftrightarrow \quad A = \widehat{Q}\widehat{R}$$

"Bookkeeping" and naming $\rightsquigarrow \widehat{R}^{-1} \rightsquigarrow \widehat{R}$ (which is also triangular).

Gram-Schmidt as Triangular Orthogonalization — 3 of 3

This formulation of the QR-factorization shows that we can **think** of the modified Gram-Schmidt algorithm as a method of **triangular orthogonalization**.

We apply a sequence of triangular operations from the right of the matrix A in order to reduce it to a matrix $\widehat{Q}$ with orthonormal columns.

In practice we **do not** explicitly form the matrices R_i and multiply them together.

However, this view tells us something about the structure of modified Gram-Schmidt.

Note: From now on when we say "Gram-Schmidt" we mean the modified version.

Final Comment: Gram-Schmidt Orthogonalization

Comment (Advantages and Disadvantages)

"The Gram-Schmidt process is inherently numerically unstable. While the application of the projections has an appealing geometric analogy to orthogonalization, the orthogonalization itself is prone to numerical error. A significant advantage however is the ease of implementation, which makes this a useful algorithm to use for prototyping if a pre-built linear algebra library is unavailable."

— Wikipedia, `https://en.wikipedia.org/wiki/QR_decomposition#Advantages_and_disadvantages`

Householder Triangularization — A More Stable Alternative

Householder triangularization is another way of computing the QR-factorization:

Gram-Schmidt	**Householder**
Numerically stable$^{(?)}$ **Useful for iterative methods**	**Even better stability** Not as useful for iterative methods
"Triangular Orthogonalization" $AR_1R_2\ldots R_n = \widehat{Q}$	**"Orthogonal Triangularization"** $Q_n \ldots Q_2Q_1A = R$

Gram-Schmidt: "Build triangular matrices that create ortogonal vectors"
Householder: "Build orthogonal transformations that create triangular matrices"

Householder Triangularization — By Picture

$$\underbrace{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}}_{A} \xrightarrow{Q_1} \underbrace{\begin{bmatrix} * & * & * \\ \mathbf{0} & * & * \\ \mathbf{0} & * & * \\ \mathbf{0} & * & * \\ \mathbf{0} & * & * \end{bmatrix}}_{Q_1A} \xrightarrow{Q_2} \underbrace{\begin{bmatrix} \times & \times & \times \\ & * & * \\ & \mathbf{0} & * \\ & \mathbf{0} & * \\ & \mathbf{0} & * \end{bmatrix}}_{Q_2Q_1A} \xrightarrow{Q_3} \underbrace{\begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & & * \\ & & \mathbf{0} \\ & & \mathbf{0} \end{bmatrix}}_{Q_3Q_2Q_1A}$$

$\mathbf{0}$ represents a new zero.
$*$ represents a modified entry.
$\times$ represents an unchanged entry.

The Big Question: How do we find the unitary matrices Q_k ?!?

Householder Reflections

The matrices Q_k are of the form

$$Q_k = \begin{bmatrix} I & 0 \\ 0 & F \end{bmatrix},$$

where I is the $((k-1)\times(k-1))$ identity, and F is an $((m-k+1)\times(m-k+1))$ unitary matrix.

The matrix F is responsible for introducing zeros into the kth column.

Let $\vec{x} \in \mathbb{C}^{m-k+1}$ be the last $(m-k+1)$ entries in the kth column.

$$\vec{x} = \begin{bmatrix} \times \\ \times \\ \vdots \\ \times \end{bmatrix} \quad \xrightarrow{F} \quad F\vec{x} = \begin{bmatrix} \pm\|\vec{x}\|_2 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \pm\|\vec{x}\|_2\,\vec{e}_1.$$

Householder Reflections: A Geometric View

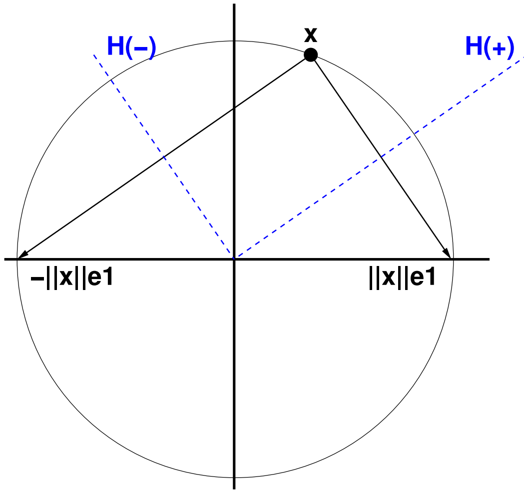


Figure: We can view the two points $\pm\|\vec{x}\|_2\vec{e}_1$ as reflections across the hyperplanes, $H_\pm$, orthogonal to $\vec{v}_\pm = \pm\|\vec{x}\|_2\vec{e}_1 - \vec{x}$.

Note: $\vec{e}_1 \in \mathbb{R}^m$ is a unit vector (for the appropriate m) in the first coordinate direction.

Householder Reflections: As Projectors

We now use our knowledge of projectors and note that for any $\vec{y} \in \mathbb{C}^m$, the vector $P\vec{y}$ defined by

$$P\vec{y} = \left[I - \frac{\vec{v}\vec{v}^*}{\vec{v}^*\vec{v}}\right]\vec{y} = \vec{y} - \vec{v}\left[\frac{\vec{v}^*\vec{y}}{\vec{v}^*\vec{v}}\right],$$

is the orthogonal projection of $\vec{y}$ **onto** the space H.

However, in order to **reflect across** the space H we must move the point twice as far, *i.e.*

$$F\vec{y} = \left[I - 2\frac{\vec{v}\vec{v}^*}{\vec{v}^*\vec{v}}\right]\vec{y} = \vec{y} - 2\vec{v}\left[\frac{\vec{v}^*\vec{y}}{\vec{v}^*\vec{v}}\right].$$

Householder Reflections: Which One ?!?

In the real case we have two possibilities, *i.e.*

$$\vec{v}_\pm = \pm\|\vec{x}\|_2\vec{e}_1 - \vec{x}, \quad \Rightarrow \quad F_\pm = I - 2\frac{\vec{v}_\pm\vec{v}_\pm^*}{\vec{v}_\pm^*\vec{v}_\pm}.$$

Mathematically, both choices give us an algorithm which produces a triangularization of A. However, from a numerical point of view, the choice which **moves** $\vec{x}$ **the farthest** is optimal.

If $\vec{x}$ and $\|\vec{x}\|_2\vec{e}_1$ are too close, then the vector $\vec{v} = (\|\vec{x}\|_2\vec{e}_1 - \vec{x})$ used in the reflection operation is the difference between two quantities that are almost the same — catastrophic **cancellation** may occur.

Therefore, we select

$$\tilde{\mathbf{v}} = -\mathrm{sign}(\mathbf{x}_1)\|\tilde{\mathbf{x}}\|\tilde{\mathbf{e}}_1 - \tilde{\mathbf{x}} \overset{*}{\equiv} \mathrm{sign}(\mathbf{x}_1)\|\tilde{\mathbf{x}}\|\tilde{\mathbf{e}}_1 + \tilde{\mathbf{x}}.$$

(∗) We can take out the minus sign since $\vec{v}$ always appears "squared" in the reflector.

`Algorithm`: Householder QR-Factorization

Algorithm (Householder QR-Factorization)

1: **for** $k \in \{1, \dots, n\}$ **do**
2: $\quad\vec{x} \leftarrow A(\texttt{k:m,k})$
3: $\quad\vec{v}_k \leftarrow \mathrm{sign}(x_1)\|\vec{x}\|_2\vec{e}_1 + \vec{x}$
4: $\quad\vec{v}_k \leftarrow \vec{v}_k / \|\vec{v}_k\|_2$
5: $\quad A(\texttt{k:m,k:n}) \leftarrow A(\texttt{k:m,k:n}) - 2\vec{v}_k(\vec{v}_k^* A(\texttt{k:m,k:n}))$
6: **end for**

$A(\texttt{k:m,k})$	Denotes the kth thru mth rows, in the kth column of A — a vector quantity.
$A(\texttt{k:m,k:n})$	Denotes the kth thru mth rows, in the kth thru nth columns of A — a matrix quantity.

Householder-QR: Where is the Q ?!?

At the completion of the Householder QR-factorization, the modified matrix A contains R (of the full QR-factorization), but Q is nowhere to be found.

Often, we only need Q implicitly, as in the **action** of Q on something. *I.e.* if we need $Q^*\vec{b}$, we can add the line

$$\vec{b}(\mathtt{k:m}) \leftarrow \vec{b}(\mathtt{k:m}) - 2\vec{v}_k(\vec{v}_k^*\vec{b}(\mathtt{k:m}))$$

to the loop; or store the generated vectors $\vec{v}_k$, and *a posteriori* compute

for $k \in \{1,\ldots,n\}$ **do**
  $\vec{b}(\mathtt{k:m}) \leftarrow \vec{b}(\mathtt{k:m}) - 2\vec{v}_k(\vec{v}_k^*\vec{b}(\mathtt{k:m}))$
end for

Householder-QR: Where is the Q ?!?

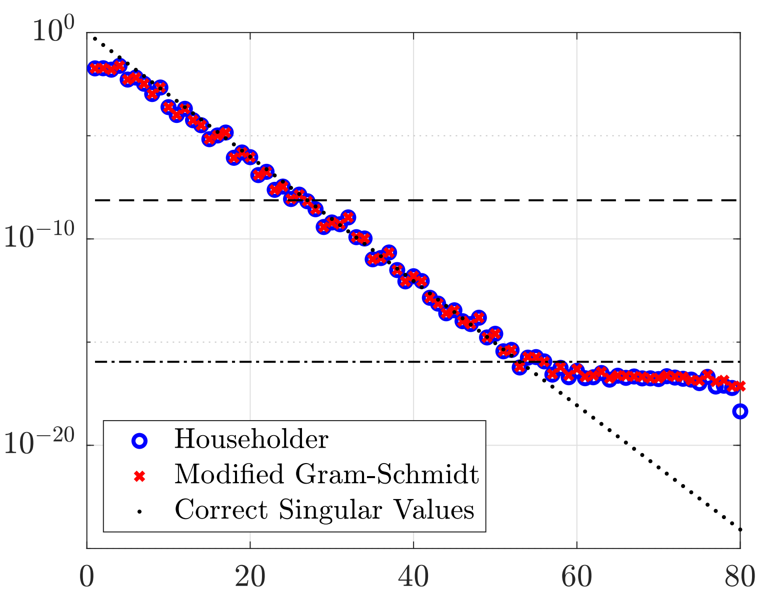
If we need $Q\vec{x}$, then we must store the generated vectors $\vec{v}_k$, and compute

for $k \in \{n,\ldots,1\}$ **do**
  $\vec{x}(\mathtt{k:m}) \leftarrow \vec{x}(\mathtt{k:m}) - 2\vec{v}_k(\vec{v}_k^*\vec{x}(\mathtt{k:m}))$
end for

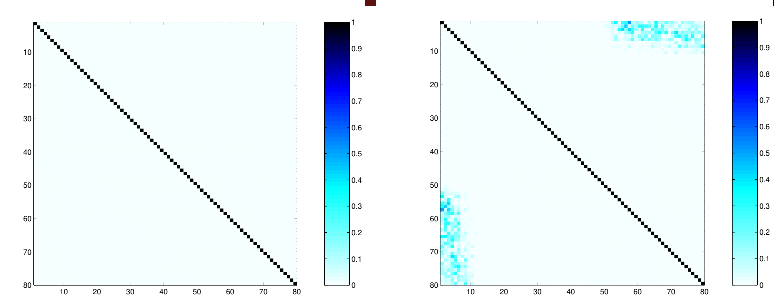
We can also use this algorithm to explicitly generate Q

$Q \leftarrow I_{n\times n}$
for $k \in \{n,\ldots,1\}$ **do**
  $Q(\mathtt{k:m,k:n}) \leftarrow Q(\mathtt{k:m,k:n}) - 2\vec{v}_k(\vec{v}_k^* Q(\mathtt{k:m,k:n}))$
end for

Comparison: Householder vs. Gram-Schmidt (modified)



Q-Orthogonality: Householder, Modified-GS, and Classical-GS



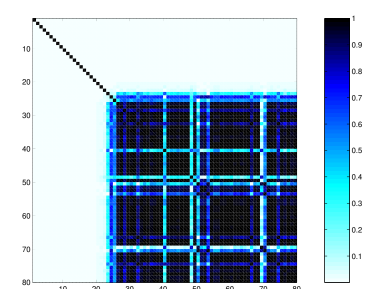


Figure: Entries of Q^*Q for Householder (top-left), GS_{mod} (top-right) and classical (left) Gram-Schmidt.

Householder-QR: Work

mgs: $\sim 2mn^2$

The dominating work is done in the operation

$$A(\texttt{k:m,k:n}) \leftarrow A(\texttt{k:m,k:n}) - 2\vec{v}_k(\vec{v}_k^* A(\texttt{k:m,k:n}))$$

Each entry in $A(\texttt{k:m,k:n})$ is "touched" by 4 flops per iteration (2 from the inner product, 1 scalar multiplication, and 1 subtraction).

The size of the sub-matrix $A(\texttt{k:m,k:n})$ is $((m-k+1)\times(n-k+1))$, so we get

$$\sum_{k=1}^{n}(m-k+1)(n-k+1) \sim \sum_{k=1}^{n}(m-k)(n-k) \sim \sum_{k=1}^{n}\left(mn + k^2 - k(m+n)\right)$$

$$\sim mn^2 + \frac{n^3}{3} - \frac{n^2}{2}(m+n) \sim \frac{mn^2}{2} - \frac{n^3}{6}$$

Hence, the work of Householder-QR is $\boxed{\sim 2mn^2 - \frac{2n^3}{3}}$ flops.

Final Comment: Householder Reflections

Comment (Advantages and Disadvantages)

"The use of Householder transformations is inherently the most simple of the numerically stable QR decomposition algorithms due to the use of reflections as the mechanism for producing zeroes in the R matrix. However, the Householder reflection algorithm is **bandwidth heavy and not parallelizable***, as every reflection that produces a new zero element changes the entirety of both Q and R matrices."*

— Wikipedia, `https://en.wikipedia.org/wiki/QR_decomposition#Advantages_and_disadvantages_2`