Outline

1. Student Learning Targets, and Objectives
   - SLOs: QR-Factorization Least Squares Problems

2. Recap
   - Checking the Roadmap

3. Projectors
   - Idempotent Matrices; Range & Nullspace; Complementary
   - Orthogonal Projectors
   - Orthonormal and Non-Orthonormal Basis

4. The QR-Factorization
   - The Full and Reduced QR-Factorizations
   - Gram-Schmidt Orthogonalization
   - QR: Existence and Uniqueness

Student Learning Targets, and Objectives

Target: The QR-Factorization
   Objective: How to compute using the Gram-Schmidt Orthogonalization Method

Target: Building Blocks
   Objective: Projectors, Idempotent Matrices, Complementary Projectors
   Objective: Characterization of the SVD using Orthogonal Projectors

Recap

Student Learning Targets, and Objectives

SLOs: QR-Factorization Least Squares Problems

A Quick Check of the Roadmap

So far we have reviewed (or quickly introduced) basic linear algebra concepts, e.g.

- Vector and Matrix operations, including norms.
- Matrix properties (vocabulary): rank, range, nullspace, domain, Hermitian conjugate (adjoint), unitary...

Then we introduced the idea — from a geometrical perspective — of the Singular Value Decomposition $A = UΣV^*$ of a matrix.

Finally, we connected the SVD and its properties to the majority of the concepts introduced.

In a sense, with the SVD we have extracted all information from the matrix $A$ and we are "done."
A Quick Check of the Roadmap

Problem#1: We do not have a stable algorithm to compute the SVD. (We don’t even know what “stable” means!)

Problem#2: Even when we have such an algorithm (later in the semester), it will turn out to be quite computationally expensive.

The Approach: We will now start building our computational toolbox so that in the end we can implement a stable, effective algorithm for the SVD.

Along the way we will study other decompositions which may not be as complete as the SVD, but are cheaper to compute and are quite useful in certain applications.

Projectors

Definition (Projector)

A projector is a square matrix $P$ that satisfies $P^2 = P$.

Think, for instance of

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

as the projection of a vector in $\mathbb{R}^3$ onto the x-y plane in $\mathbb{R}^3$: — find a corner in the room; put a broom-stick in the corner and let it point into the room; observe the shadow on the floor. (This is making the assumption that the lighting is laser-based and arranged so that all light-rays go straight from ceiling-to-floor...)
Complementary Projectors

We notice that a projector $P$ separates $\mathbb{C}^m$ into two spaces.

Conversely, if $S_1, S_2 \subseteq \mathbb{C}^m$ such that $S_1 \cap S_2 = \{0\}$, and $S_1 + S_2 = \mathbb{C}^m$, then $S_1$ and $S_2$ are **complementary subspaces** and there exists a projector $P$ onto $S_1$ along $S_2$ such that $\text{range}(P) = S_1$, and $\text{null}(P) = S_2$.

An **orthogonal projector** is a projector that projects onto a subspace $S_1$ along a space $S_2$, where $S_1$ and $S_2$ are orthogonal.

Orthogonal Projectors

**Warning!!!**

Orthogonal projectors are not orthogonal/unitary matrices!!!

An **orthogonal projector** is a projector that is also Hermitian, i.e.

$$P^* = P, \quad \text{and} \quad P^2 = P$$

If $P = P^*$, then the inner product of $P\vec{x} \in S_1$ and $(I - P)\vec{y} \in S_2$ is zero:

$$\langle P\vec{x}, (I - P)\vec{y} \rangle = \vec{x}^*P^*(I - P)\vec{y} = \vec{x}^*(P - P^2)\vec{y} = 0$$
Orthogonal Projectors

We now show that if $P$ projects onto $S_1$ along $S_2$ ($S_1 \perp S_2$, and $S_1$ has dimension $n$), then $P = P^*$ — the construction will give us a very simple characterization of the projector in terms of the SVD!

We construct the SVD of $P$ as follows:

Let $\{\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_m\}$ be an orthonormal basis for $\mathbb{C}^m$, where $\{\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_n\}$ is a basis for $S_1$, and $\{\vec{q}_{n+1}, \vec{q}_{n+2}, \ldots, \vec{q}_m\}$ is a basis for $S_2$. We have

$$
\begin{cases}
P\vec{q}_j = \vec{q}_j, & j \leq n \\
P\vec{q}_j = 0, & j > n
\end{cases}
$$

Now, let $Q$ be the unitary $(m \times m)$ matrix whose $j$th column is $\vec{q}_j$.

Thus we have constructed an SVD of $P$:

$$
P = Q\Sigma Q^* = (Q^*)^*\Sigma^*Q^*
$$

and clearly $P$ is Hermitian.

$$
P^* = (Q\Sigma Q^*)^* = (Q^*)^*\Sigma^*Q^* = Q\Sigma Q^* = P.
$$

Orthogonal Projectors

Projection with an Orthonormal Basis

Since some singular values (in $\Sigma$) are zero, we can use the reduced SVD instead, i.e. we only keep the first $n$ columns in $Q$, and we end up with

$$
P = \hat{Q}\hat{Q}^* = \begin{pmatrix}
\vec{q}_1 & \cdots & \vec{q}_n
\end{pmatrix}
$$

where the columns of $Q \in \mathbb{C}^{m \times n}$ are orthonormal.

There is nothing magic about orthonormal vectors associated with the SVD — as long as the columns, $\vec{q}_j \in \mathbb{C}^m$, of $\hat{Q}$ are orthonormal, the matrix $P = QQ^*$ defines an orthogonal projection onto $S_1 = \text{range}(Q)$.

Projection with an Orthonormal Basis

The projection

$$
\vec{v} \mapsto P\vec{v}\quad \text{by}\quad QQ^*\vec{v} = \sum_{i=1}^{n}(\vec{q}_i\vec{q}_i^*)\vec{v}
$$

can be viewed as a sum on $n$ rank-one projections,

$$
P_i = \vec{q}_i\vec{q}_i^*
$$

where each such projection isolates the component in a single direction given by $\vec{q}_i$. These rank-one projectors will show up as building blocks in future algorithms.

For completeness, we note that the complement of a rank-one projector is a rank-$(m - 1)$ projector that eliminates the component in the direction of $\vec{q}_i$

$$
P_{\perp\vec{q}_i} = (I - \vec{q}_i\vec{q}_i^*)
$$
Projection with a Non-Orthonormal Basis

We can build an orthogonal projector from an arbitrary (not necessarily orthogonal) basis.

Let \( S_1 \) be the subspace spanned by the linearly independent vectors \( \{ \vec{a}_1, \ldots, \vec{a}_n \} \) and let \( A \) be the matrix with columns \( \vec{a}_j \).

\[
\vec{v} \quad \overset{P}{\mapsto} \quad \vec{y} \in \text{range}(A), \quad \vec{y} = A\vec{x}, \text{ some } \vec{x} \in \mathbb{C}^n \\
\vec{y} - \vec{v} \perp \text{range}(A) \\
\iff \vec{a}_j^*(\vec{y} - \vec{v}) = 0, \quad \forall j \\
\iff \vec{a}_j^*(A\vec{x} - \vec{v}) = 0, \quad \forall j \\
\iff A^*(A\vec{x} - \vec{v}) = 0 \\
\iff A^*A\vec{x} = A^*\vec{v} \\
\iff \vec{x} = (A^*A)^{-1}A^*\vec{v} \\
\iff \vec{y} = A(A^*A)^{-1}A^*\vec{v} = P\vec{v}
\]

Numerically Dangerous

The Reduced QR-Factorization

In many application we are interested in the column spaces spanned by a matrix \( A \), i.e. the spaces

\[
\text{span}(\vec{a}_1) \subseteq \text{span}(\vec{a}_1, \vec{a}_2) \subseteq \text{span}(\vec{a}_1, \vec{a}_2, \vec{a}_3) \subseteq \ldots
\]

We may, for instance, be looking for a minimum, or maximum of some quantity over each subspace.

**The QR-factorization** generates a sequence of orthonormal vectors \( \{ \vec{q}_1, \vec{q}_2, \vec{q}_3, \ldots \} \) that spans these spaces, i.e.

\[
\text{span}(\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_k) = \text{span}(\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_k), \quad k = 1, \ldots, n
\]

The reason for doing this is that it is much easier to work in an orthonormal basis.

Projections: Summary

The key thing we bring from the discussion on projections is the ability to identify how much of the “action” is directed in a certain set of directions, or subspace.

These ideas will be used, explicitly or implicitly, in many algorithms presented in this (and other) classes.

We now turn our attention to one of the “heavy-hitters” among numerical algorithms — the QR-factorization.
The Full QR-Factorization

As for the SVD, we can extend the QR-factorization by padding $\hat{Q}$ with an additional $(m - n)$ orthonormal columns, and zero-padding $\hat{R}$ with an additional $(m - n)$ rows of zeros:

$$\begin{align*}
\text{Figure: The Reduced QR-Factorization, } A &= \hat{Q}\hat{R} \\
\text{Figure: The Full QR-Factorization, } A &= QR
\end{align*}$$

In the full QR-factorization, the columns $\hat{a}_j$, $j > n$ are orthogonal to range($A$). If rank($A$) = $n$, they are an orthonormal basis for range($A$)$^\perp$ = null($A^*$), the space orthogonal to range($A$).

Algorithm: Classical Gram-Schmidt

We summarize our findings:

Algorithm (Classical Gram-Schmidt)

1. for $k \in \{1, \dots, n\}$ do
2. \quad $\hat{v}_k \leftarrow \hat{a}_k$
3. \quad for $i \in \{1, \dots, k - 1\}$ do
4. \quad \quad $r_{ik} \leftarrow \hat{a}_i^* \hat{a}_k$
5. \quad \quad $\hat{v}_k \leftarrow \hat{v}_k - r_{ik} \hat{a}_i$
6. \quad end for
7. \quad $r_{kk} \leftarrow \|\hat{v}_k\|_2$
8. \quad $\hat{q}_k \leftarrow \hat{v}_k/r_{kk}$
9. end for

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

Building the QR-Factorization — Gram-Schmidt Orthogonalization

The equations on slide 20 outline a method for computing reduced QR-factorizations.

At the $k$th step, we are looking to construct $\hat{q}_k \in \text{span}(\hat{a}_1, \ldots, \hat{a}_k)$ such that $\hat{q}_k \perp \text{span}(\hat{q}_1, \ldots, \hat{q}_{k-1})$

We simply take $\hat{a}_k$, and subtract all the projections onto the directions $\hat{q}_1, \ldots, \hat{q}_{k-1}$, and then normalize the resulting vector

$$(*)\quad \hat{v}_k = \hat{a}_k - (\hat{a}_1^* \hat{q}_1)\hat{q}_1 - \cdots - (\hat{a}_{k-1}^* \hat{q}_{k-1})\hat{q}_{k-1}$$

Computationally, it is more efficient to compute

$$(*)'\quad \hat{v}_k = \hat{a}_k - \hat{q}_1(\hat{a}_1^* \hat{q}_k) - \cdots - \hat{q}_{k-1}(\hat{q}_{k-1}^* \hat{a}_k)$$

The QR-Factorization: Existence and Uniqueness

Theorem (Existence of the QR-Factorization)

Every $A \in \mathbb{C}^{m \times n}$ ($m \geq n$) has a full QR-factorization, hence also a reduced QR-factorization.

Theorem (Uniqueness of the QR-Factorization)

Every $A \in \mathbb{C}^{m \times n}$ ($m \geq n$) of full rank has a unique reduced QR-factorization $A = \hat{Q}\hat{R}$, with $r_{kk} > 0$. 
1. If $A$ is full rank, the Gram-Schmidt algorithm gives the unique reduced QR-factorization.

2. If $A$ does not have full rank, then $\vec{v}_k = 0$ can occur during the iteration; if it does set $\vec{q}_k$ to be an arbitrary vector* orthogonal to $\text{span}(\vec{q}_1, \ldots, \vec{q}_{k-1})$, and proceed.

3. If $m > n$, follow Gram-Schmidt as described until $j = n$, then take an addition $(m - n)$ steps, introducing arbitrary orthogonal $\vec{q}_k$ in each step.

* Column pivoting (exchanges) may be necessary.

---

**Solving $A\vec{x} = \vec{b}$ by QR-Factorization**

If we have a QR-factorization algorithm handy, then we have the following “algorithm” for solving $A\vec{x} = \vec{b}$

1. Compute the QR-factorization $A = QR$.
2. Compute $\vec{y} = Q^*\vec{b}$.
3. Solve $R\vec{x} = \vec{y}$ for $\vec{x}$.

**Note:** Computing $Q^*\vec{b}$ is just a multiplication with a unitary matrix. Since $|\text{det}(Q^*)| = 1$ this completely numerically stable in the sense that errors will not be magnified. (We will quantify this soon.)

**Note:** Solving $R\vec{x} = \vec{y}$ is very easy (backward substitution) since $R$ is upper triangular.

**Note:** The bulk of the work is in computing the QR-factorization (2–3 times that of Gaussian Elimination).

---

**Homework #3 Due Date in Canvas/Gradescope**

Implement the reduced QR-factorization by **Classical Gram-Schmidt**.

1. Write a function which given $A \in \mathbb{C}^{m \times n}$ computes $Q \in \mathbb{C}^{m \times n}$, and $R \in \mathbb{C}^{n \times n}$ — in matlab/python you want to start something like this (e.g. file: qr_cgs.m, or qr_cgs.py):

   ```
   function [Q,R] = qr_cgs(A)
   % Indentation does not matter...
   % Implicit Return of Results
   def qr_cgs(A)
   # Indentation matters
   # Explicit Return of Results
   return Q, R
   ```

   See help function in matlab, or python_functions (clickable) for help on writing functions.

2. Validate your function — test that (i) $(A - QR) \approx 0$; (ii) $Q$ is unitary; and (iii) $R$ upper triangular. Show 3 test cases for $(3 \times 3)$, $(5 \times 5)$, and $(251 \times 251)$ matrices.

3. Compare the result for the $(3 \times 3)$, $(5 \times 5)$ cases with the built-in (“library”) version of the QR-factorization; comment on the similarities/differences.

   See help qr in matlab, or numpy.linalg.qr (clickable).

4. Can you find a non-zero matrix where your QR-factorization breaks?

   $\infty$. Hand in your code, and your validation/test-cases.

   $\infty \infty$. Appropriately “tag” all pages with the corresponding question(s) in Gradescope.