Numerical Matrix Analysis
Notes #8
The QR-Factorization: — Least Squares Problems

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 2023
(Revised: February 13, 2023)

Student Learning Targets, and Objectives

SLOs: QR-Factorization Least Squares Problems

1. Student Learning Targets, and Objectives
   - Target Linear Least Squares Problems
     - Objective Discrepancy Measure: Residual
     - Objective Relation to the Maximum Likelyhood Estimate
     - Objective Polynomial Fitting, and the Vandermonde Matrix
     - Objective The Moore-Penrose Pseudo-Inverse of a Matrix

2. Recap

3. Least Squares Problems
   - Problem, Language...
   - Problem Set-up: the Vandermonde Matrix
   - Formal Statement

4. LSQ: The Solution
   - Pseudo-Inverse
     - The Moore-Penrose Matrix Inverse
   - 3.5 Algorithms for the LSQ Problem

Previously (Gram-Schmidt and Householder)

Computing the QR-factorization 3 ways:

**Gram-Schmidt Orthogonalization** — Modified vs. Classical.

**Householder Triangularization**

<table>
<thead>
<tr>
<th>Modification</th>
<th>Gram-Schmidt</th>
<th>Householder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerically stable*</td>
<td>Even better stability</td>
<td></td>
</tr>
<tr>
<td>Useful for iterative methods</td>
<td>Not as useful for iterative methods</td>
<td></td>
</tr>
</tbody>
</table>

**“Triangular Orthogonalization”**

\[ AR_1 R_2 \ldots R_n = \hat{Q} \]

**“Orthogonal Triangularization”**

\[ Q_n \ldots Q_2 Q_1 A = R \]

\[ \text{Work} \sim 2mn^2 \text{ flops} \]

\[ \text{Work} \left( \sim 2mn^2 - \frac{2n^3}{3} \right) \text{ flops} \]

Note: No \( Q \) at this lower cost!!!
Least Squares

Least squares data/model fitting is used everywhere; — social sciences, engineering, statistics, mathematics, “data science” . . .

In our language, the problem is expressed as an **overdetermined system**

\[ Ax = \vec{b}, \quad A \in \mathbb{C}^{m \times n}, \ m \gg n. \]

Since \( A \) is “tall and skinny,” we have more equations than unknowns. \( \sim \) Very likely to be inconsistent.

The least squares solution is defined by

\[ \vec{x}_{LS} = \arg \min_{\vec{x} \in \mathbb{C}^n} \| \vec{b} - A\vec{x} \|_2^2. \]

**Least Squares: Some Language**

The quantity \( \vec{r}(\vec{x}) = \vec{b} - A\vec{x} \) is known as the **residual**, and since our problem is overdeterned, we cannot (in general) hope to find an \( \vec{x}^* \) such that \( \vec{r}(\vec{x}^*) = \vec{0}. \)

Minimizing some norm of \( \vec{r}(\vec{x}) \) is a close second best.

This (among other things, like e.g. checking that large matrices contain zeros) is why we needed the discussion of norms back in [Lecture#3].

The choice of the 2-norm leads to a problem that is easy to work with, and it is usually the correct choice for statistical reasons — computing the least squares solution yields the **Maximum Likelihood Estimate** (under certain conditions — independent identically distributed variables, etc...)

**Least-Squares: Problem Set-Up**

So... How do we fit (polynomial) models to data?!? We flip back to [Lecture#2] and express our system using the **Vandermonde matrix**

\[ A = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^d \\ 1 & x_2 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^d \end{bmatrix}, \quad \vec{c} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_d \end{bmatrix}, \quad \vec{b} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}, \]

where the fitting polynomial is described using the coefficients \( \vec{c} \)

\[ p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_d x^d. \]

Given the locations of the points \( \vec{x} \), and a particular set of coefficients \( \vec{c} \), the matrix-vector product \( \vec{p} = A\vec{c} \) evaluates the polynomial in those points, i.e. \( \vec{p}^T = (p(x_1), p(x_2), \ldots, p(x_m)). \)
Least-Squares: Thinking About Projectors

We can think of the least squares problem as the problem of finding the vector in $\text{range}(A)$ which is closest to $\vec{b}$.

Since we are measuring using the 2-norm, “closest” $\equiv$ closest in the sense of Euclidean distance.

We look to minimize the residual, $\vec{r} = \vec{b} - Ax$.

The minimum residual must be orthogonal to $\text{range}(A)$.

\[
\begin{align*}
\vec{b} & \\
\vec{r} = \vec{b} - Ax & \\
\text{range}(A) & \\
\vec{y} = Ax = \vec{Pb} & \\
\end{align*}
\]

Least Squares: Formal Statement

Theorem (Linear Least Squares)

Let $A \in \mathbb{C}^{m \times n}$ ($m \geq n$), and $\vec{b} \in \mathbb{C}^m$ be given. A vector $\vec{x} \in \mathbb{C}^n$ minimizes the residual norm $\|\vec{r}\|_2 = \|\vec{b} - A\vec{x}\|_2$, thereby solving the least squares problem, if and only if $\vec{r} \perp \text{range}(A)$, that is

\[
A^*\vec{r} = 0, \quad \iff \quad A^*A\vec{x} = A^*\vec{b}, \quad \iff \quad A\vec{x} = P\vec{b}
\]

where the orthogonal projector $P \in \mathbb{C}^{m \times m}$ maps $\mathbb{C}^m$ onto $\text{range}(A)$. The $(n \times n)$ system $A^*A\vec{x} = A^*\vec{b}$ (the normal equations), is non-singular if and only if $A$ has full rank $\iff$ The solution $\vec{x}^*$ is unique if and only if $A$ has full rank.

Language: The Pseudo-Inverse

Hence, if $A$ has full rank, the least squares-solution $\vec{x}_{LS}$ is uniquely determined by

\[
\vec{x}_{LS} = (A^*A)^{-1}A^*\vec{b}.
\]

The matrix

\[
A^\dagger \equiv (A^*A)^{-1}A^*
\]

is known as a pseudo-inverse of $A$.

With this notation and language, the least squares problem comes down to computing one or both of

\[
\vec{x} = A^\dagger\vec{b}, \quad \vec{y} = P\vec{b}
\]

We will look at $(3 + \frac{1}{2})$ algorithms for accomplishing this.
### Take#1 — The Normal Equations

The classical / straight-forward / bone-headed (?) way to solve the least squares problem is to solve the normal equations:

$$A^* A \tilde{x} = A^* \tilde{b}.$$

The preferred way of doing this is by computing the Cholesky factorization (details to follow [Notes#17](https://en.wikipedia.org/wiki/Conjugate_gradient_method#On_the_normal_equations)).

Given $R$ is an upper triangular matrix, the equivalent system

$$R^* R \tilde{x} = A^* \tilde{b}, \quad (A^* = (R^* R)^{-1} A^*),$$

can be solved by a forward and a backward substitution sweep.

---

**Sidetone:** There are specialized iterative schemes, e.g. CGNE (Conjugate Gradient on the Normal Equations) which are useful in certain circumstances (sparse $A$-matrix); see


---

### Take#2 — The SVD

If we compute the reduced SVD

$$A = \hat{U} \Sigma V^*,$$

then we can use $\hat{U}$ to express the projector $P = \hat{U} \hat{U}^*$, and end up with the linear system of equations

$$\hat{U} \Sigma V^* \tilde{x} = \hat{U} \hat{U}^* \tilde{b},$$

and we get $\tilde{x}_{LS}$ by

$$\tilde{x}_{LS} = V \Sigma^{-1} \hat{U}^* \tilde{b}.$$

Here, the pseudo-inverse is expressed as

$$A^* = V \Sigma^{-1} \hat{U}^*.$$

---

### Take#3½ — The Q-less QR-Factorization

Say we computed $\hat{R}$ using the Householder Q-less QR-factorization, but "forgot" to compute $Q^* \tilde{b}$, is everything lost?!?

No, we can still compute $\tilde{x}_{LS}$ using the following sequence

$$\begin{align*}
\tilde{x} & \leftarrow R^{-1} R^* (A^* \tilde{b}) \\
\tilde{r} & \leftarrow \tilde{b} - A \tilde{x} \\
\tilde{e} & \leftarrow R^{-1} R^* (A^* \tilde{r}) \\
\tilde{x} & \leftarrow \tilde{x} + \tilde{e},
\end{align*}$$

The first step solves the "semi-normal equations"

$$R^* R \tilde{x} = A^* \tilde{b}.$$
Algorithms for Least Squares: Comments

<table>
<thead>
<tr>
<th>Method</th>
<th>Work (flops)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal Equations</td>
<td>(\sim (mn^2 + \frac{3n^3}{3}))</td>
<td>Fastest, sensitive to roundoff errors. Not recommended.</td>
</tr>
<tr>
<td>QR-Factorization</td>
<td>(\sim (2mn^2 - \frac{2n^3}{3}))</td>
<td>Your everyday choice. Can run into trouble when (A) is close to rank-deficient.</td>
</tr>
<tr>
<td>SVD</td>
<td>(\sim (2mn^2 + 11n^3))</td>
<td>The Big Hammer more stable than the QR approach, but requires more computational work.</td>
</tr>
</tbody>
</table>

Comment
If \(m \gg n\), then the work for QR and SVD are both dominated by the first term, \(2mn^2\), and the computational cost of the SVD is not excessive. However, when \(m \approx n\) the cost of the SVD is roughly 10 times that of the QR-factorization.

Looking Forward

We can now compute (and have a “serious” use for) one of the big important tools of numerical linear algebra — the QR-factorization.

Next, we finally(?) formalize the discussion on “numerical stability,” and then we take another look at some of our algorithms in the light of stability considerations.

HW#4

1. Implement modified Gram-Schmidt QR-factorization.

   Write a function which given an \(A \in \mathbb{C}^{m \times n}\) computes \(Q \in \mathbb{C}^{m \times n}\), and \(R \in \mathbb{C}^{n \times n}\) — \texttt{qr.mgs(A)} \rightarrow Q, R.

   Work through experiment #1 and #2 in “Lecture 9” of Trefethen & Bau. Make sure your versions of classical and modified GS can reproduce figure 9.1.

   Note that depending on your coding environment, you may have to use larger (and worse conditioned) matrices to achieve the loss of orthogonality in classical Gram-Schmidt.

2. Do exercises 9.1(a,b), and 9.2(a,b).

   For additional (non-mandatory) fun do exercises 9.1(c) and 9.2(c).