Overdetermined systems

We examined the case of systems $Ax = b$ where $A \in \mathbb{R}^{m \times n}$ and $m \geq n$. In general, a true solution does not exist. We define however the least squares solution as the vector $\hat{x}$ that minimizes

$$\hat{x} = \arg \min \| b - Ax \|_2^2$$

$\hat{x}$ is given by the solution to the system of normal equations

$$A^T A \hat{x} = A^T b \quad (1)$$

System (1) is square ($m \times n$) and invertible (if $A$ has linearly independent columns). However, the condition number of $A^T A$ could be very poor... for example, if $A$ was square, we would have $\text{cond} (A^T A) = [\text{cond} (A)]^2$.

An alternative method that does not suffer from this problematic conditioning is the $QR$ factorization.

Def. An $n \times n$ matrix $Q$ is called orthogonal iff

$$Q^T Q = QQ^T = I.$$
Thm Let $A \in \mathbb{R}^{m \times n}$ ($m > n$) have linearly independent columns. Then a decomposition

$$A = QR$$

exists, such that $Q \in \mathbb{R}^{m \times m}$ is orthogonal and $R \in \mathbb{R}^{m \times n}$ is upper triangular (i.e. $R = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{nn} \end{bmatrix}$).

Additionally, given that $A$ has linearly independent columns, we guarantee that $r_{ii} \neq 0$.

In Matlab, the QR decomposition is obtained via the $qr$ function, i.e.

$$[Q, R] = qr(A);$$

Now, let us write

$$Q = \begin{bmatrix} \hat{Q} & Q^* \end{bmatrix}$$

where $\hat{Q} \in \mathbb{R}^{m \times m}$ contains the first $m$ columns of $Q$, and $Q^* \in \mathbb{R}^{(m-n) \times m}$ contains the last $(m-n)$ columns.
Respectively, we write:

\[ A = \begin{bmatrix} \hat{R} \\ \mathbf{0} \end{bmatrix} \] where \( \hat{R} \in \mathbb{R}^{m \times n} \) (and upper triangular) contains the first \( n \) rows of \( R \).

\( \hat{R} \) is also non-singular (for lin. ind. columns of \( \hat{R} \))

We can verify the following:

\[ \hat{Q}^T \hat{Q} = \mathbf{I}_n \quad (\text{although } \hat{Q} \hat{Q}^T \neq \mathbf{I}_m !) \]

**Proof:**

\[ [\hat{Q}^T \hat{Q}]_{ij} = \sum_{k=1}^{m} [\hat{Q}^T]_{ik} [\hat{Q}]_{kj} \]

\[ = \sum_{k=1}^{m} [\hat{Q}]_{ki} [\hat{Q}]_{kj} = \sum_{k=1}^{m} [Q]_{ki} [Q]_{kj} \]

\[ = [Q^T Q]_{ij} = [I_m]_{ij} \]

\[ \rightarrow A = QR = \hat{Q} \cdot \hat{R} \]

(\( m \times n \)) (\( m \times m \)) (\( m \times n \)) (\( n \times n \)).

**Proof:** Similar.

The factorization \( A = \hat{Q} \cdot \hat{R} \) is the so-called economy-size QR factorization, and computed in Matlab as:

\[ [\hat{Q}, \hat{R}] = qr(A, 0) ; \]
Once we have $\hat{Q}$ & $\hat{R}$ computed, we observe that the normal equations can be written as:

$$A^T A \hat{x} = A^T b$$

using $A = QR$

$$\Rightarrow \quad \hat{R} \hat{Q} Q^T R x = \hat{R} \hat{Q} b$$

$$= In$$

$\hat{R}$ is invertible

$$\Rightarrow \quad (\hat{R}^T \hat{R} \hat{R} x) = (\hat{Q}^T \hat{Q} \hat{Q} b)$$

$$\Rightarrow \quad \boxed{\hat{R} x = \hat{Q} b} \quad \text{(\#)}$$

Least squares solution using $QR$ decomposition

Matlab: $[\hat{Q}, \hat{R}] = qr(A, 0)$

$$\hat{x} = \hat{Q}^T \hat{b}$$

$\hat{x} = \hat{R} \backslash \hat{x}$

**Benefit:** We can show that $\text{cond}(A^T A) = [\text{cond}(\hat{R})]^2$,

Thus, equation (\#) is much better conditioned than the normal equations system.
Numerical integration

We seek an algorithm to approximate the definite integral:

\[ I = \int_{a}^{b} f(x) \, dx \]

(or, the area below the graph of \( y = f(x) \))

Of course, in the fortuitous case where we know a function \( F(x) \) (the anti-derivative of \( f \)), s.t. \( F'(x) = f(x) \),

we can write:

\[ \int_{a}^{b} f(x) \, dx = F(b) - F(a) \]

e.g. \[ \left( \arctan(x) \right)' = \frac{1}{1+x^2} \], thus

\[ \int_{a}^{b} \frac{dx}{1+x^2} = \arctan(b) - \arctan(a). \]
However, this is not a practical algorithm, since:

→ The anti-derivative is not generally known.
→ Often, the anti-derivative may be significantly more
expensive to evaluate than \( f(x) \) itself,

(e.g. compare \( f(x) = \frac{1}{1+x^2} \) (easy) with \( F(x) = \arctan(x) \) (expensive)).

General methodology

→ Subdivide the interval of integration using the \( n+1 \) points
\( \{x_i\}_{i=0}^{n} \), with

\[
a = x_0 < x_1 < x_2 < \ldots < x_{n-1} < x_n = b
\]

→ In each interval \([x_i, x_{i+1}]\) approximate \( f(x) \) with
some simpler function, say a polynomial \( p^{(i)}(x) \), which
is easy to integrate. Approximate

\[
I_i = \int_{x_i}^{x_{i+1}} f(x) \, dx \approx \int_{x_i}^{x_{i+1}} p^{(i)}(x) \, dx.
\]

→ Compute the integral \( I = \int_{a}^{b} f(x) \, dx \)
as

\[
I = \sum_{i=0}^{n-1} I_i \approx \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} p^{(i)}(x) \, dx.
\]
Example: The rectangle rule

At each interval \([x_i, x_{i+1}]\) use the approximation

\[ p_i(x) = f(x_i) \] (the left endpoint!)

Thus we approximate:

\[
I_i = \int_{x_i}^{x_{i+1}} f(x) \, dx \approx \int_{x_i}^{x_{i+1}} f(x_i) \, dx = f(x_i) \cdot (x_{i+1} - x_i)
\]

(We often present this rule on a single interval \([a, b]\), as

\[
\int_{a}^{b} f(x) \, dx \approx f(a) \cdot (b-a)
\]
In the case where \( x_{i+1} - x_i = h = \text{const} \), we can write:

\[
I = \int_a^b f(x) \, dx = \sum_{i=0}^{n-1} I_i = \sum_{i=0}^{n-1} f(x_i) \cdot h = \]

\[
I \approx \frac{b-a}{n} \sum_{i=0}^{n-1} f(x_i)
\]

As in the case of interpolation, we can assess the error incurred by this approximation. There are 2 errors we actually focus on:

\[\rightarrow\] The local error \( \left| \int_{x_i}^{x_{i+1}} (f(x) - p^G(x)) \, dx \right| \) at each subinterval

\[\rightarrow\] The global error for the entire integral \( \int_a^b f(x) \, dx \).