

MA3227 Numerical Analysis II

Lecture 5: Least Squares

Simon Etter



2019/2020

Least Squares

Introduction

Our aim in the next few weeks will be to develop numerical algorithms which can solve the finite-difference-discretised Poisson equation in dimensions $d > 1$ more efficiently than LU factorisation.

One such class of algorithms are *subspace algorithms*: given $V \in \mathbb{R}^{N \times k}$, approximate the solution $x = A^{-1}b$ by

$$x_k = V_k y_k \quad \text{where} \quad y_k = \arg \min \|AV_k y_k - b\|_2. \quad (1)$$

This lecture prepares for our discussion of subspace algorithms by showing how to solve the above least squares problem.

Least squares problem

Given $A \in \mathbb{R}^{N \times k}$ and $b \in \mathbb{R}^N$ with $N > k$ and $\text{rank}(A) = k$, find

$$x := \arg \min \|Ax - b\|_2. \quad (2)$$

Note that the A, x in (2) are equal to AV_k, y_k in (1), respectively. This change in notation is so that I can follow the standard notation both for least squares as well as subspace problems.

Least Squares

Least squares via QR factorisation

Assume $QR = A$ with

$$Q = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \in \mathbb{R}^{N \times N} \text{ orthogonal with } Q_1 \in \mathbb{R}^{N \times k},$$
$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \in \mathbb{R}^{N \times k} \text{ with } R_1 \in \mathbb{R}^{k \times k} \text{ upper-triangular.}$$

Orthogonal means $Q^T Q = I$. Since Q is square, this implies $Q^T = Q^{-1}$.
Upper-triangular means $R_1[i, j] = 0$ if $i > j$.

In pictorial form:

$$\underbrace{\begin{pmatrix} \text{gray square} \end{pmatrix}}_A = \underbrace{\begin{pmatrix} \text{square with vertical dashed line} \end{pmatrix}}_Q \underbrace{\begin{pmatrix} \text{square with gray upper triangle} \end{pmatrix}}_R$$

Least Squares

Least squares via QR factorisation

We observe that

- ▶ $\text{rank}(R_1) = k$ since $\text{rank}(A) = k$, and
- ▶ $\|Qv\|_2 = \|v\|_2$ since $\|Qv\|_2^2 = v^T Q^T Q v = v^T v = \|v\|_2^2$.

From the second point, it follows that

$$\|Ax - b\|_2 = \|Q(Rx - Q^T b)\|_2 = \|Rx - Q^T b\|_2,$$

or in block-matrix form

$$\left\| \begin{pmatrix} R_1 \\ 0 \end{pmatrix} x - \begin{pmatrix} Q_1^T b \\ Q_2^T b \end{pmatrix} \right\|_2.$$

From the first point, it further follows that the solution to the least squares problem is given by

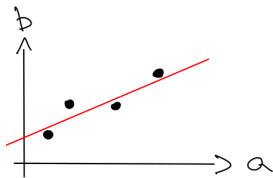
$$x = R_1^{-1} Q_1^T b.$$

Least Squares

Example

Given data vectors $a, b \in \mathbb{R}^N$ defining points (a_i, b_i) in the 2d plane,

find $c_0, c_1 \in \mathbb{R}$ minimising
$$\sum_{i=1}^N (c_0 + c_1 a_i - b_i)^2.$$



This is a least squares problem with

$$A = \begin{pmatrix} 1 & a \end{pmatrix}, \quad x = \begin{pmatrix} c_0 & c_1 \end{pmatrix}^T, \quad b = b.$$

See `example()`.

Least Squares

Real-world example: pricing of diamonds

Price v of diamond is primarily determined by four parameters:

- ▶ $w > 0$: carat (weight)
- ▶ $p \in P = \{I2, \dots, IF\}$: clarity (lack of defects in crystal structure)
- ▶ $c \in C = \{J, \dots, D\}$: colour (how white the diamond is)
- ▶ $q \in Q = \{\text{Fair}, \dots, \text{Ideal}\}$: cut quality

Assume price is determined as follows:

$$v = x_0 + x_w w + \sum_{\hat{p} \in P \setminus \{p^*\}} x_{\hat{p}} \delta_{p\hat{p}} + \sum_{\hat{c} \in C \setminus \{c^*\}} x_{\hat{c}} \delta_{c\hat{c}} + \sum_{\hat{q} \in Q \setminus \{q^*\}} x_{\hat{q}} \delta_{q\hat{q}}.$$

Parameters x can be determined through linear least squares fit.

See `05_least_squares_diamonds.jl`.

Least Squares

Discussion

- ▶ It can be shown that the factorisation $QR = A$ described in the previous slide always exists. We will discuss an algorithm for computing such Q, R later in this lecture.
- ▶ $x = R_1^{-1} Q_1^T b$ is easily computed once Q_1, R_1 are available since R_1 is an upper-triangular matrix.
- ▶ $x = R_1^{-1} Q_1^T b$ has a nice geometric interpretation, see next slide.

Least Squares

Interpretation of $x = R_1^{-1} Q_1^T b$

Observations:

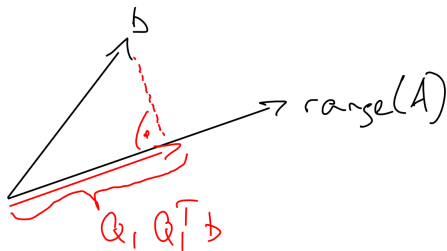
- ▶ Columns of Q_1 provide an orthogonal basis for $\text{range}(A)$.
Proof. Orthogonality of columns of Q_1 follows from orthogonality of Q .
 $\text{range}(A) = \text{range}(Q_1)$ holds since $A = Q_1 R_1$ and R_1 is invertible.
- ▶ $Q_1 Q_1^T$ is the orthogonal projector onto $\text{range}(A)$.
Proof. $Q_1 Q_1^T$ is projector because $(Q_1 Q_1^T)^2 = Q_1 Q_1^T Q_1 Q_1^T = Q_1 Q_1^T$.
Projector is orthogonal since the projected vector $Q_1 Q_1^T b$ and the residual $b - Q_1 Q_1^T b$ are orthogonal for all b :
 $(Q_1 Q_1^T b)^T (b - Q_1 Q_1^T b) = Q_1 Q_1^T b - Q_1 Q_1^T Q_1 Q_1^T b = 0$.
- ▶ Ax is the orthogonal projection of b onto $\text{range}(A)$.
Proof. $Ax = Q_1 R_1 R_1^{-1} Q_1^T b = Q_1 Q_1^T b$.

Interpretation of $x = R_1^{-1} Q_1^T b$:

- ▶ $Q_1^T b = Q_1^T (Q_1 Q_1^T b)$ computes the coordinates of the projection $Q_1 Q_1^T b$ with respect to Q_1 .
- ▶ Multiplying by R_1^{-1} translates these coordinates into coordinates with respect to A .

Least Squares

Interpretation of $x = R_1^{-1} Q_1^T b$ (continued)



Least Squares

QR factorisation

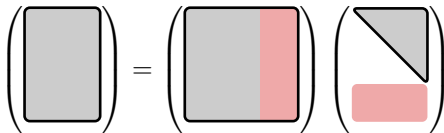
Given $A \in \mathbb{R}^{N \times k}$, find

- ▶ orthogonal matrix Q ($Q^T Q = I$), and
- ▶ upper-triangular matrix R ($R[i, j] = 0$ if $i > j$)

such that $QR = A$.

There are two versions of the QR factorisation:

- ▶ Fat QR: $Q \in \mathbb{R}^{N \times N}$, $R \in \mathbb{R}^{N \times k}$ (gray and red).
- ▶ Thin QR: $Q \in \mathbb{R}^{N \times k}$, $R \in \mathbb{R}^{k \times k}$ (gray only).


$$\left(\begin{array}{c} \text{gray rectangle} \end{array} \right) = \left(\begin{array}{c} \text{gray square} \quad \text{red rectangle} \end{array} \right) \left(\begin{array}{c} \text{gray triangle} \\ \text{red rectangle} \end{array} \right)$$

We have seen that thin QR is enough for solving $\arg \min_x \|Ax - b\|_2$.
The following slides will present an algorithm for computing this thin QR.

Least Squares

Computing the QR factorisation

Let us write $A = QR$ column by column:

$$A[:, 1] = Q[:, 1] R[1, 1],$$

$$A[:, 2] = Q[:, 1] R[1, 2] + Q[:, 2] R[2, 2],$$

$$A[:, 3] = Q[:, 1] R[1, 3] + Q[:, 2] R[2, 3] + Q[:, 3] R[3, 3]$$

...

Imposing $\|Q[:, 1]\|_2 = 1$ on the first line yields

$$R[1, 1] = \|A[:, 1]\|_2,$$

$$Q[:, 1] = \frac{A[:, 1]}{R[1, 1]}.$$

Least Squares

Computing the QR factorisation (continued)

Let us rewrite the second line:

$$A[:, 2] - Q[:, 1] R[1, 2] = Q[:, 2] R[2, 2].$$

Imposing $Q[:, 1]^T Q[:, 2] = 0$ yields

$$Q[:, 1]^T A[:, 2] - Q[:, 1]^T Q[:, 1] R[1, 2] = 0$$

which is equivalent to (recall that $Q[:, 1]^T Q[:, 1] = 1$)

$$R[1, 2] = Q[:, 1]^T A[:, 2].$$

$R[2, 2]$ and $Q[:, 2]$ are determined by imposing $\|Q[:, 2]\|_2 = 1$:

$$R[2, 2] = \|A[:, 2] - Q[:, 1]R[1, 2]\|_2,$$

$$Q[:, 2] = \frac{A[:, 2] - Q[:, 1]R[1, 2]}{R[2, 2]}.$$

Least Squares

Computing the QR factorisation (continued)

Continuing this procedure yields the following algorithm.

Algorithm 1 Classical Gram-Schmidt

```
1: for  $\ell = 1, \dots, k$  do  
2:    $\tilde{q}_\ell = A[:, \ell]$   
3:   for  $m = 1, \dots, \ell - 1$  do  
4:      $R[m, \ell] = Q[:, m]^T A[:, \ell]$   
5:      $\tilde{q}_\ell = \tilde{q}_\ell - Q[:, m] R[m, \ell]$   
6:   end for  
7:    $R[\ell, \ell] = \|\tilde{q}_\ell\|_2$   
8:    $Q[:, \ell] = \tilde{q}_\ell / R[\ell, \ell]$   
9: end for
```

Interpretation:

- ▶ Lines 3-6 subtract the components of $A[:, \ell]$ in the directions of the $Q[:, 1 : \ell - 1]$.
- ▶ Line 8 normalises the remainder.

Least Squares

Computing the QR factorisation (continued)

In exact arithmetic, the algorithm on the previous slide is equivalent to the following (proof will follow).

Algorithm 2 Modified Gram-Schmidt

```
1: for  $\ell = 1, \dots, k$  do  
2:    $\tilde{q}_\ell = A[:, \ell]$   
3:   for  $m = 1, \dots, \ell - 1$  do  
4:      $R[m, \ell] = Q[:, m]^T \tilde{q}_\ell$   
5:      $\tilde{q}_\ell = \tilde{q}_\ell - Q[:, m] R[m, \ell]$   
6:   end for  
7:    $R[\ell, \ell] = \|\tilde{q}_\ell\|_2$   
8:    $Q[:, \ell] = \tilde{q}_\ell / R[\ell, \ell]$   
9: end for
```

Modified Gram-Schmidt produces more accurate results in the presence of rounding errors (no further justification, just accept this).

Least Squares

Proof that classical and modified Gram-Schmidt are equivalent

In any iteration of the for-loop over m , we have

$$\tilde{q}_\ell = A[:, \ell] - \sum_{m'=1}^{m-1} Q[:, m'] R[m', \ell].$$

Since the columns of Q are orthogonal, it follows

$$Q[:, m]^T \tilde{q}_\ell = Q[:, m]^T A[:, \ell] - \sum_{m'=1}^{m-1} \underbrace{Q[:, m]^T Q[:, m']}_{=0} R[m', \ell].$$

Implementing the Gram-Schmidt algorithm

See `modified_gram_schmidt()`.

Least Squares

Example: breakdown of Gram-Schmidt

So far we always assumed $\text{rank}(A) = k$, i.e. the columns of A are linearly independent.

Let us now consider the matrix

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 + 10^{-15} \end{pmatrix}.$$

A is “close” to having rank 1 in the sense that it would have rank 1 if we dropped the 10^{-15} entry in the bottom right corner.

Let us compute the QR factorisation of this matrix and check the orthogonality of Q .

Least Squares

Example: breakdown of Gram-Schmidt (continued)

```
julia> A = [  
    1      1  
    1      1+1e-15  
];
```

```
julia> Q,R = modified_gram_schmidt(A);
```

```
julia> Q'*Q  
2×2 Array{Float64,2}:  
 1.0      0.196116  
 0.196116 1.0
```

Observation: Q is not orthogonal at all!

To understand what is going on, we must understand how computers represent real numbers.

Least Squares

Floating-point representation

The floating-point representation of a number $a \in \mathbb{R}$ is

$$a = \pm x.xxx \times 10^y$$

where $x.xxx$ represents a fixed number of significant digits (four in this case) and y is an exponent such that the first significant digit is not 0.

Examples for representation with three significant digits:

Number	Floating-point representation
$\pi = 3.1415\dots$	3.14×10^0
$\frac{1}{\sqrt{2}} = 0.7071\dots$	7.07×10^{-1}
-100.1	-1.00×10^2

Remark (ignore if this confuses you)

On a computer, the significant digits are represented as a binary number (e.g. 3 is represented as 1.1×2^1). However, we will continue working with a decimal representation for simplicity.

Least Squares

Floating-point arithmetic

All standard operations ($+$, $-$, \times , \div , $\sqrt{\cdot}$, etc.) are implemented such that they produce the exact result rounded to a fixed number of significant digits.

Example, assuming two significant digits:

Operation	Exact result	Result in FP arithmetic	Rounding error
1.1×1.1	1.21	1.2×10^0	0.01
1.1×1.0	1.1	1.1×10^0	0

Discussion

The above rule (“FP result = exact result + rounding”) applies for single operations. In this case, it guarantees that the resulting error is small relative to the exact result.

For example, for the first of the above examples we get a relative error of

$$\frac{|1.21 - 1.2|}{|1.2|} \approx 0.01.$$

However, rounding errors may blow up once we start to chain operations.

Least Squares

Example

Let us compute $1.1 \times 1.1 - 1.2$ in two-digits floating-point arithmetic.

$$1.1 \times 1.1 - 1.2 \longrightarrow \text{round}(1.21) - 1.2 \longrightarrow 1.2 - 1.2 = 0.$$

The exact result is 0.01, so the relative error is

$$\frac{|0.01-0|}{|0.01|} = 1.$$

Floating-point arithmetic leads to a 100% error in the result!

Rule-of-thumb

Rounding errors blow up if we subtract two numbers a, b which have a small difference $|b - a|$ and at least one of which already contains rounding errors.

Least Squares

Breakdown of Gram-Schmidt (continued)

We now have the knowledge required to understand the breakdown of Gram-Schmidt when applied to the matrix

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 + 10^{-15} \end{pmatrix}.$$

The place where things go wrong is when we subtract from the second column the component in the direction of the first column. This happens on line 5, which in the given context may be written as

$$\tilde{q}_2 = A[:, 2] - Q[:, 1] R[1, 2].$$

This operation satisfies both conditions for blow-up of rounding errors:

- ▶ \tilde{q}_2 is small because $A[:, 2]$ and $Q[:, 1] \propto A[:, 1]$ are almost collinear.
- ▶ The term $Q[:, 1] R[1, 2]$ is the result of floating-point computations and hence contains rounding errors.

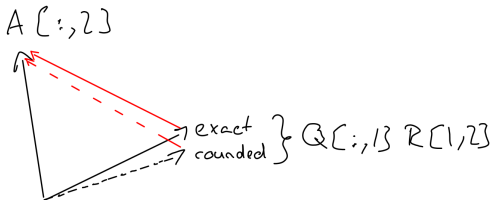
We conclude that the numerically computed \tilde{q}_2 is the exact \tilde{q}_2 plus some rounding errors which are large relative to the exact values of \tilde{q}_2 .

It is this rounding-error-component which spoils the orthogonality between \tilde{q}_2 and $Q[:, 1]$.

Least Squares

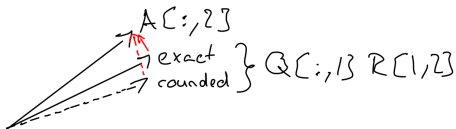
Breakdown of Gram-Schmidt (illustration)

If $A[:, 1]$ and $A[:, 2]$ are far from being linearly dependent:



Difference between solid and dashed red arrows is small in relative terms.

If $A[:, 1]$ and $A[:, 2]$ are almost linearly dependent:



Diff. between solid and dashed red arrows may be large in relative terms.

Least Squares

Breakdown of Gram-Schmidt (conclusion)

Our findings may be summarised as follows:

Gram-Schmidt is unreliable if the columns of A are almost linearly dependent.

This observation will play an important role in the next lecture.

Discussion

With a bit of extra work, one can show that any algorithm for computing $QR = A$ will significantly amplify rounding errors if columns of A are almost linearly dependent.

The reason for this is that computing $QR = A$ for an almost rank-deficient matrix A is an ill-conditioned problem.

The concept of conditioning was introduced in Numerical Analysis I. The precise definition will not be important for our purposes, but I will use the term

“ill-conditioned” repeatedly. You may replace “ill-conditioned” with “likely to give a wrong result when evaluated on a computer” if this is easier for you.

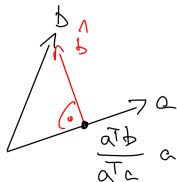
Least Squares

Summary

- ▶ $x = \arg \min \|Ax - b\|_2 \iff x = R^{-1}Q^T b$ where $QR = A$.
- ▶ Basic orthogonalisation operation: given $a, b \in \mathbb{R}^N$, set

$$\hat{b} := b - \frac{a^T b}{a^T a} a.$$

Then, $\hat{b} \perp a$ and $\text{span}\{a, b\} = \text{span}\{a, \hat{b}\}$.



- ▶ Gram-Schmidt algorithm for computing the QR factorisation is the repeated application of the above basic operation.
- ▶ Computing $QR = A$ is an ill-conditioned problem if columns of A are almost linearly dependent.