
# Computational Mathematics
# Numerical Methods

Based on prof. Federico Poloni's lectures

Gemma Martini
Alessandro Cudazzo
Giulia Volpi

October 29, 2019

# Contents

# 1 20th of September 2018 — F. Poloni

## 1.1 A warm up

Before starting here is a small recap

- **Vector-Scalar product**:
  Let $x \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$ we call **multiple** of vector $x$ the following:

$$\lambda x = x\lambda = \begin{pmatrix} \lambda x_1 \\ \vdots \\ \lambda x_n \end{pmatrix}$$

  .

- **Vector-Vector product**:
  Let $x, y \in \mathbb{R}^n$. The product between those two vectors is computed as follows $x^T y = \sum_{i=1}^{n} x_i y_i$ and $x^T y \in \mathbb{R}$.

- **Scalar-Matrix product**:
  Let $A \in \mathbb{R}^{n \times m}$ and $\lambda \in \mathbb{R}$ we call the **scalar-matrix product** the following:

$$\lambda A = A\lambda = \begin{pmatrix} \lambda A_{11} & \lambda A_{12} & ... \\ \lambda A_{21} & \lambda A_{22} & ... \\ \vdots & \vdots & \ddots \end{pmatrix}$$

  .

- **Matrix-Vector product**:
  Given a matrix $A \in M(n, m, \mathbb{R})$ and a vector $v \in \mathbb{R}^m$ the **matrix-vector product** $Av = w \in \mathbb{R}^n$ is computed as follows:

$$w = Av = \begin{pmatrix} A_1 v \\ A_2 v \\ \vdots \\ A_m v \end{pmatrix}, \; w_i = \sum_{j=1}^{m} A_{ij} v_j$$

  This is the simple way, just a row-by-column vector product, the computational complexity of this operation is $O(n^2)$.
  The smart way to compute it: **linear combinations** of columns of A, e.g.:

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \\ A_{41} & A_{42} & A_{43} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}$$

with **linear combinations** we have:

$$\begin{pmatrix} A_{11} \\ A_{21} \\ A_{31} \\ A_{41} \end{pmatrix} v_1 + \begin{pmatrix} A_{12} \\ A_{22} \\ A_{32} \\ A_{42} \end{pmatrix} v_2 + \begin{pmatrix} A_{13} \\ A_{23} \\ A_{33} \\ A_{43} \end{pmatrix} v_3 = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix}$$

- **Matrix-Matrix Product**:
  Given two matrices $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{m \times k}$ we call **matrix-matrix product** the following: $C = AB$ such that $C_{ij} = A_i B^j$, where ${A_i}^T \in \mathbb{R}^m$ is the $i$-th row of $A$, $B^i$ is the $i$-th column of $B$ ($B^i \in \mathbb{R}^m$) and $C \in M(n, k, \mathbb{R})$. Notice that this product is **not commutative**: $AB \neq BA$ might not even make sense dimension-wise.

  Computational Cost: multiplying $m \times n$ and $n \times p$ requires $m(2n-1)p$ floating point operations (flops) and two matrix, A and B, are both $n \times n$ is $O(n^2)$. Forget about fancier algorithms (e.g. Strassen)

  **Order of operations**

  Usual algebra properties hold, e.g.: A(B + C) = AB + AC, A(BC) = (AB)C, etc...
  Parenthesization matters a lot: if $A, B \in \mathbb{R}^{n \times n}$, $v \in \mathbb{R}^n$, then $(AB)v$ costs $O(n^3)$, but $A(Bv)$ costs $O(n^2)$. Programming languages usually do not rearrange parentheses to help.

- **Image** of a matrix $A$ ($\mathrm{Im}(A)$): the set of vectors that can be obtained multiplying $A$ by any vector in the domain of $A$.
- **Kernel** of a matrix $A$ ($\ker(A)$): the set of vectors $w$ in its domain such that $Aw = 0$.
- Given a matrix $A \in M(n, \mathbb{R})$ we call **inverse** of $A$ the matrix $A^{-1}$ such that:

$$A^{-1}A = AA^{-1} = I_n = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}$$

  The **inverse of a product** (shoe-sock identity) is $(AB)^{-1} = B^{-1}A^{-1}$. Notice that this identity holds only for square matrices.

- The **transpose** of a matrix $A \in M(n, m, \mathbb{R})$ is $A^T$ such that $A^T_{ij} = A_{ji}$. The **transpose of a product** (shoe-sock identity) is $(AB)^T = B^T A^T$. ( This identity holds for square and rectangular matrices)

**Definition 1.1.** ***General linear group (GL)****: the general linear group of degree $n$ is the set of $n \times n$ invertible matrices, together with the operation of ordinary matrix multiplication*

**Fact 1.1.** *Let $A \in GL(n, \mathbb{R})$ (aka $A$ is a real square matrix of size $n$ and invertible), $B, C \in M(n, m, \mathbb{R})$ and we have the product $AB = AC$. If there is a matrix $M$ such that $MA = I$:*

$$(MA)B = (MA)C \iff B = C, \ M = A^{-1}$$

.

So $AB = AC$ does not usually imply $B = C$, $A$ must be invertible!

**Row and column vectors notation**

$$v = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}, \ v^T = \begin{pmatrix} 4 & 5 & 6 \end{pmatrix}$$

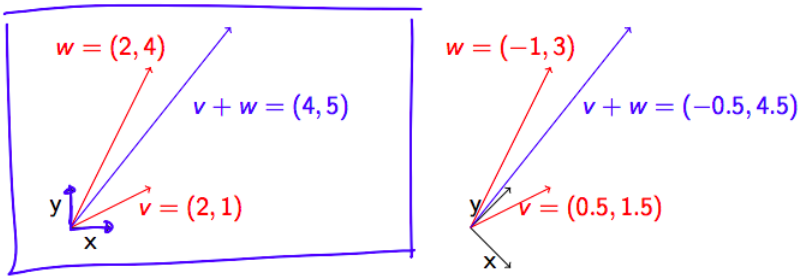
$v$ is a column vector in $\mathbb{R}^3$ (or a matrix in $\mathbb{R}^{3\times 1}$) and $v^T$ is a row vector (or a matrix in $\mathbb{R}^{1\times 3}$).

**Definition 1.2.** ***Basis****: a set $B$ of elements (vectors) in a vector space $V$ is called a* ***basis****, if every element of $V$ may be written in a unique way as a (finite) linear combination of elements of $B$. The coefficients of this linear combination are referred to as components or coordinates on $B$ of the vector. The elements of a basis are called basis vectors.*

**Canonical basis**: $w = w_1 e_1 + w_2 e_2 + w_3 e_3 + w_4 e_4$, e.g. for $m = 4$

$$e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \ e_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \ e_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \ e_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

The powerful idea behind linear algebra: many relations are true regardless of the basis we use. E.g. $w$, $v$ and $w + v$ in two different bases.



## 1.2 Solving Linear Systems

The objective of this course, for the part concerning numerical methods, is solving linear systems efficiently.

**Definition 1.3** (Linear system)**.** *Let $A \in M(n, m, \mathbb{R})$, $b \in \mathbb{R}^n$ and $x \in \mathbb{R}^m$. We term* ***linear system*** *the following:*

$$Ax = b$$

Our goal is to approximate such vector $x$, hence resulting in solving a minimum problem:

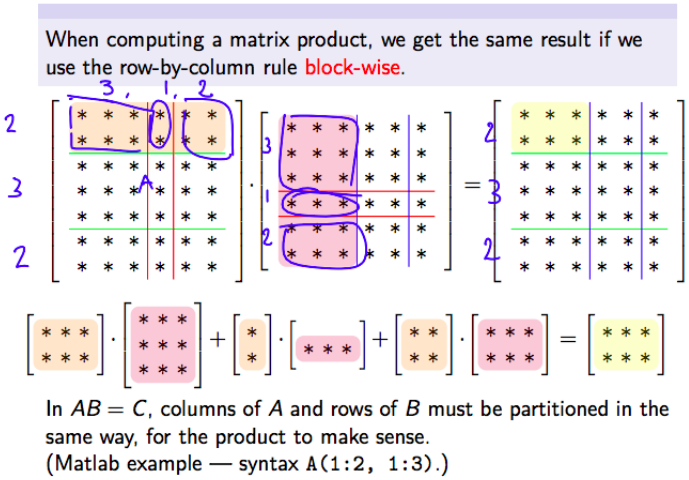
$$\min \|Ax - b\|$$

If we have a square and invertible A matrix solve a linear systems means: find coordinates $x_1, \ldots, x_m$ needed to write $b$ as linear combinations of the columns of (square) $A \in \mathbb{R}^{m \times m}$ and in this case, the solution is given by: $x = A^{-1}b$.
**Warning**: this is not the best way to solve a linear system on a computer!

**Something on Matlab . . .**

Notice that the machine precision is $10^{-16}$, so we should pay attention when making computations, since we may incur in some error (proportional to the size of the operands).
In Matlab a matrix is written as `A=[1, 2, 3; 4, 5, 6];`, where `[1, 2, 3]` is the first row of the matrix A.
The transpose of a matrix or a vector is denoted by `A'`.
The inverse of a square matrix is denoted by `inv(A)`.
If we are interested in only a part of our matrix `A` we may write `A[1:2, 1:3]` and obtain only the rows of `A` that go from 1 to 2 and those columns from 1 to 3.

**Definition 1.4** (Block multiplications)**.** *Let $A \in M(n, m, \mathbb{R})$ and let $B \in M(m, k, \mathbb{R})$. We can compute the result of a block of the matrix $AB$ as the product of the two blocks in $A$ and $B$ in the corresponding position.*




**Note**: Block operations usually give better performance: one matrix-matrix product performs faster than n matrix-vector products (even if they have the same number of flops). This is one of the reasons why library calls usually perform better than hand-coded loops (Blas/Lapack).

**Fact 1.2** (Block triangular matrices)**.** *Let $M \in M(n, m, \mathbb{R})$ and $B \in M(m, k, \mathbb{R})$ such that they are* ***block triangular****. Their product is a block triangular matrix as well, block triangular matrices are closed under products:*

$$MB = \begin{pmatrix} A & B \\ 0 & C \end{pmatrix} \begin{pmatrix} D & E \\ 0 & F \end{pmatrix} = \begin{pmatrix} AD & AE + BF \\ 0 & CF \end{pmatrix}$$

**Fact 1.3** (Properties of triangular matrices)**.**
*Let $M$ be a block triangular matrix, with all $A_{ii}$ square*

$$M = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1k} \\ 0 & A_{22} & \dots & A_{2k} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & A_{kk} \end{pmatrix}$$

1. *A block triangular matrix is invertible iff all diagonal blocks $A_{ii}$ are invertible;*
2. *The eigenvalues of a block triangular matrix are the union of the eigenvalues of each $A_{ii}$ block;*
3. *Let $M \in GL(n, m, \mathbb{R})$ such that $M = \begin{pmatrix} A & B \\ 0 & C \end{pmatrix}$ the inverse of $M$ is*

   $$M^{-1} = \begin{pmatrix} A^{-1} & -A^{-1}BC^{-1} \\ 0 & C^{-1} \end{pmatrix}.$$

4. *The product of two block (upper/lower) triangular matrices (with compatible block sizes) is still block triangular*

Why are we interested in block triangular matrices? They depict a situation as shown in Figure 1.1.


FIGURE 1.1: The adjacency matrix of a biparted graph has 0s in its bottom left part (Matlab syntax `A[p+1:n; 1:p]=0`).

**General principle**: matrix structures matter. Block triangular linear system has a cheaper system solution than a general system as shown in example 1.1.

**Example 1.1.** *2x2 block triangular linear system*

$$\begin{pmatrix} A & B \\ 0 & C \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} e \\ f \end{pmatrix}$$

*(Again, diagonal blocks are square and all dimensions are compatible.)*

$$\begin{pmatrix} Ax + By \\ Cy \end{pmatrix} = \begin{pmatrix} e \\ f \end{pmatrix} \implies y = C^{-1}f, x = A^{-1}(e - BC^{-1}f)$$

$$\begin{pmatrix} A & B \\ 0 & C \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} & -A^{-1}BC^{-1} \\ 0 & C^{-1} \end{pmatrix}$$

*Informal idea: we can start solving from the variables in C.*

## 1.3 Orthogonality

**Definition 1.5** (Norms)**.** *Let $x \in \mathbb{R}^n$. We "measure" their magnitude using so-called "norms".*

EUCLIDEAN: $\|x\|_2 = x^T x = \sqrt{\sum_{i=1}^{n} {x_i}^2}$;

NORM 1: $\|x\|_1 = \sum_{i=1}^{n} |x_i|$;

$p$-NORM: $|x|_p = \left( \sum_{i=1}^{n} |x_i|^p \right)^{1/p}$;

0-NORM: $\|x\|_0 = |\{i \ : \ |x_i| > 0\}|$;

$\infty$-NORM: $\|x\|_\infty = \max_{i=1,\dots,n} |x_i|$.

From now on in this part of th course we will refer to norm-2 only.

**Definition 1.6** (Orthogonal matrix)**.** *Let $A \in M(n, \mathbb{R})$ a square matrix. $A$ is orthogonal iff:*

- $A^T A = I_n$
- $AA^T = I_n$
- $A^{-1} = U^T$

*where $I_n$ is the identity matrix of size $n$ (1 on the diagonal, 0 elsewhere).*

# 2 26th of September 2018 — F. Poloni

## 2.1 Orthogonality (II)

In the previous lecture we introduced some sufficent conditions for matrix orthogonality.

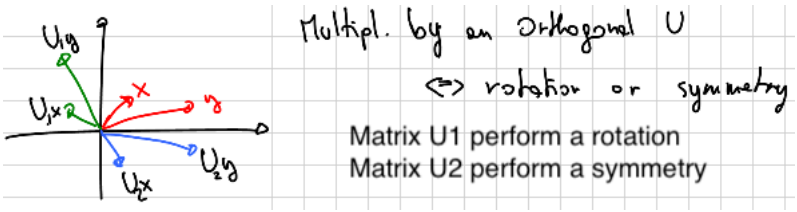
**Theorem 2.1.** *Let $U \in M(n, \mathbb{R})$ be an orthogonal matrix and let $x \in \mathbb{R}^n$. Then $\|Ux\| = \|x\|$.*

*Proof.* Instead of proving that $\|Ux\| = \|x\|$ we will prove $\|Ux\|^2 = \|x\|^2$:

$$\|Ux\|^2 = (Ux)^T(Ux) \overset{(1)}{=} x^T U^T U x = x^T I_n x = x^T x = \|x\|$$

where $\overset{(1)}{=}$ follows from the definition of transpose of a product. □

**Geometrically** an orthogonal preserve the norm, so a matrix $A$ represents a symmetry or a rotations on vector $x$ and these operations do not alter the size of vectors.




**Definition 2.1** (Orthogonality)**.** *Let $x, y \in R^n$ we say that $x$ and $y$ are **orthonormal** if $< x, y >= 0$.*

**Definition 2.2** (Orthonormality)**.** *Let $x, y \in R^n$ we say that $x$ and $y$ are **orthonormal** if $< x, y >= 0$ and $\|x\| = \|y\| = 1$.*

**Fact 2.2.** *Let us take $U \in M(n, \mathbb{R})$ such that $U$ is orthogonal. Then its columns $U^1, U^2, \ldots, U^n$ are **orthonormal** and the same holds for its rows.*

$$U^{i^T} U^j = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}$$

*and*

$$U_i {U_j}^T = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}$$

**Fact 2.3.** *Let $U, V \in M(n, \mathbb{R})$, such that $U$ and $V$ are orthogonal, then $UV$ is orthogonal. Orthogonal are closed under the product.*

*Proof.* $(UV)^T(UV) = V^T U^T U V = V^T I_n V = V^T V = I_n$ □

**Fact 2.4.** *We will often deal with tall thin rectangular matrices with orthonormal columns:*

$$U_1 = [u_1 \;\; u_2 \;\; \ldots \;\; u_n] \in \mathbb{R}^{m \times n} \quad (m \geq n)$$

*There exists a matrix $U_2$ s.t. $[U_1 \; U_2]$ is square orthogonal.*

## 2.2 Eigenvalues / Eigenvector

**Definition 2.3** (Eigenvectors and eigenvalues)**.** *Let $A \in M(n, \mathbb{R})$ and let $x \neq 0 \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$.*

*If $Ax = \lambda x$ we say that $x$ is an **eigenvector** of **eigenvalue** $\lambda$.*

**Fact 2.5.** *Let $A \in M(n, \mathbb{R})$ (real triangular matrix). The eigenvalues of $A$ are the scalars on the diagonal.*

**NOTE**: Eigenvectors and eigenvalues are interesting because we can use it to get a special decomposition of a matrix $A$.

**Eigendecomposition of a matrix**:
For almost almost all matrices $A \in \mathbb{R}^{n \times n}$ under some conditions we can decompose $A$ as:

$$A = V \Lambda V^{-1}$$

$$A = V \Lambda V^{-1} = \begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix} \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}$$

where $v_i$, $\forall i = 1, \ldots, n$ are eigenvectors of $A$ of eigenvalue $\lambda_i$ and $w_i =$ rows of $V^{-1}$.

Another way to see the diagonalized form of $A$ is the following:

$$A = V \Lambda V^{-1} = \sum_{i=1}^{n} v_i \lambda_i w_i^T =$$

$$\boxed{v_1}\ \boxed{\lambda_1}\ \boxed{w_1^T} + \boxed{v_2}\ \boxed{\lambda_2}\ \boxed{w_2^T} + \cdots + \boxed{v_n}\ \boxed{\lambda_n}\ \boxed{w_n^T}$$

**Something on Matlab ...**

Notice that in Matlab the eigenvalues and eigenvectors of a matrix are computed using the command `[V, Lambda] = eig(U)` and this operation has a computational complexity of $O(n^3)$.
We can check that the matrix `A` is equal to the decomposition in this way:
`A - V * Lambda * inv(V)` or `norm(A - V * Lambda * inv(V))` (both should be near to zero).

Notice that not all matrices $A \in M(n, \mathbb{R})$ allow a diagonal decomposition. It may happen that such a matrix is diagonalizable in $\mathbb{C}$ and its eigenvalues are complex.

**Fact 2.6.** *If this factorization with eigenvalues and eigenvectors holds, then:*

$$A = V\Lambda V^{-1} \Longrightarrow Av_i = v_i\lambda \ , \ \forall i = 1, \dots, n$$

This decomposition tell us the behavior under repeated application of a matrix $A$ to a vector $x$. This process allow to scale a general vector $x$.

**Something on Matlab ...**

e.g. `A = [1 1; 1 1]` and `x = [1 1]`
then `A * x` is equal to `[2 2]'` and `A * A * x` is equal to `[4 4]'`

**Fact 2.7.** *If $A \in M(n, \mathbb{R})$ is diagonalizable (aka may be written as $A = V\Lambda V^{-1}$) then:*

$$A^k x = \sum_{i=1}^{n} v_i \lambda_i^k w_i^T$$

*Proof.*

Linear algebra view point:

$$\begin{aligned} A^k x &= AA \dots Ax \\ &= V\Lambda \cancel{V^{-1}} \cancel{V} \Lambda \cancel{V^{-1}} \dots \cancel{V} \Lambda V^{-1} x \\ &= V\Lambda^k V^{-1} x \\ &= V \begin{pmatrix} {\lambda_1}^k & & & \\ & {\lambda_2}^k & & \\ & & \ddots & \\ & & & {\lambda_n}^k \end{pmatrix} V^{-1} x \end{aligned} \tag{2.1}$$

□

**NOTE**: if $A$ is not square, $Av$, $\lambda v$ have different sizes and it doesn't make sense to talk about eigenvalues.

### What can go wrong with eigenvalue decomposition

1. The eigenvalue decomposition is highly non-unique, we can:
   - Reorder eigenvalues/vectors
   - Replace an eigenvector $v_i$ with $2v_i$ , $3.5v_i$ ...
   - For matrices with repeated eigenvalues, even more possibilities:
     e.g. $I = VIV^{-}1$ for every invertible $V$

2. some matrices have only complex eigenvalues: e.g. $\begin{pmatrix} 2 & 4 \\ -3 & 3 \end{pmatrix}$

3. some matrices have fewer eigenvectors than we want and we can't use eigenvalue decomposition: e.g. $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$

Now, thanks to the eigenvalue decomposition we can prove the following:

**Theorem 2.8.** *Let $A \in M(n, \mathbb{R})$. If $|\lambda_i| < 1$ for all eigenvalues $\lambda_i$ of $A$ then $\lim_{k\to\infty} A^k x = 0$.*

**Theorem 2.9.** *Let $A \in M(n, \mathbb{R})$. If $\forall \lambda_i$ eigenvalues of $A$ $|\lambda_i| < |\lambda_1|$ then $A^k x \approx V^1 {\lambda_1}^k \alpha_1$.*

**Fact 2.10.** *Let $A \in M(n, \mathbb{R})$ be a diagonalizable matrix and let*

$$A = V\Lambda V^{-1} = \begin{pmatrix} V^1 & V^2 & \cdots & V^n \end{pmatrix} \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}$$

*Let us now consider a reordering of $V$'s columns and apply the same permutations to the "diagonal vector" of $\Lambda$ such that:*

$$\hat{V} = \begin{pmatrix} V^2 & V^1 & V^3 \cdots & V^n \end{pmatrix} \text{ and } \hat{\Lambda} = \begin{pmatrix} \lambda_2 & & & & \\ & \lambda_1 & & & \\ & & \lambda_3 & & \\ & & & \ddots & \\ & & & & \lambda_n \end{pmatrix}$$

*$A$ can be diagonalized through such $\hat{V}$ and $\hat{\Lambda}$: $A = V\Lambda V^{-1} = \hat{V}\hat{\Lambda}\hat{V}^{-1}$.*

Moreover, in the case of repeated eigenvalues

**Fact 2.11.** *Let $A \in M(n, \mathbb{R})$ a diagonalizable matrix such that $A = V\Lambda V^{-1}$, where $\lambda_1 = \lambda_2$ (without loss of generality). Then $V$ can be replaced by $\widetilde{V} = \begin{pmatrix} & & & & \\ V^1 + V^2 & V^1 - V^2 & V^3 & \cdots & V^n \\ & & & & \end{pmatrix}$.*

**Theorem 2.12** (Spectral theorem)**.** *Let $A \in S(n, \mathbb{R})$ (A is a real symmetric matrix). Then $A$ is diagonalizable $A = U\Lambda U^-1$, where eigenvalues are all real numbers and we can take $U$ orthogonal matrix.*

For symmetric matrices, nothing goes wrong: eigenvalues decomposition always exists (Spectral theorem). The matrix will not have complex eigenvalues and not fewer eigenvectors. We can choose an $U$ orthogonal because we said it was possible to reorder eigenvalues/vectors and replace eigenvector $v_i$.

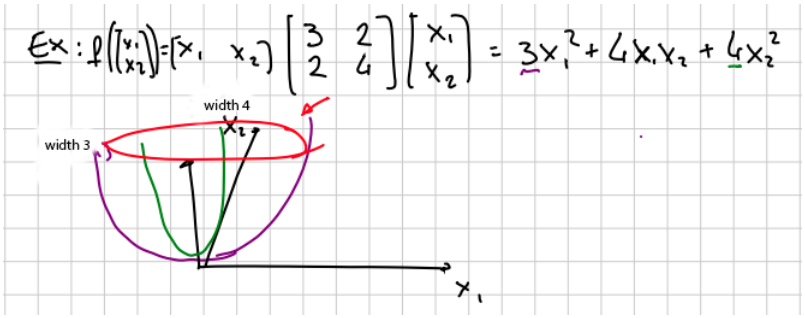
**Something on Matlab ...**

If we have an symmetric matrix `B` and we compute `[V, D] = eig(B)`, Matlab will always return an orthogonal matrix $V$.

**Quadratic forms**: for a fixed symmetric matrix $Q = Q^T$, consider $x \in \mathbb{R}^n$ and $Q \in \mathbb{R}^{n \times n}$ $f(x) = x^T Q x$ (Geometric idea: paraboloids):

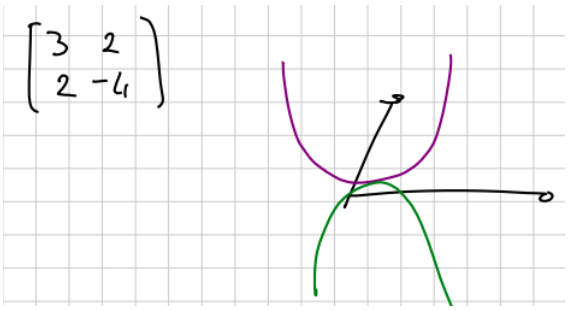
Let's see two example in a Geometric point of view:
Example 1:

$$f(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}) = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 3 & 2 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$




Example 2:

$$\text{with a matrix } Q = \begin{pmatrix} 3 & 2 \\ 2 & -4 \end{pmatrix}$$



**Fact 2.13.** *Let $Q \in S(n, \mathbb{R})$ (For a fixed symmetric matrix) and let $x \in \mathbb{R}^n$. Then*

$$\lambda_{min}\|x\|^2 \leq x^T Q x \leq \lambda_{max}\|x\|^2$$

*where $\lambda_{max}$ and $\lambda_{min}$ are respectively the eigenvalue of maximum value and the eigenvalue of minimum value.*

*Proof.*

Easy case with $Q = \Lambda$ diagonal:

$$x^T Q x = x^T \begin{pmatrix} \lambda_2 & & & & \\ & \lambda_1 & & & \\ & & \lambda_3 & & \\ & & & \ddots & \\ & & & & \lambda_n \end{pmatrix} x = \lambda_1 {x_1}^2 + \lambda_2 {x_2}^2 + \cdots + \lambda_n {x_n}^2$$

It is obvious that this sum is bounded by:

$$\lambda_{\min}({x_1}^2 + {x_2}^2 + \cdots + {x_n}^2) \leq \lambda_1 {x_1}^2 + \lambda_2 {x_2}^2 + \cdots + \lambda_n {x_n}^2 \leq \lambda_{\max}({x_1}^2 + {x_2}^2 + \cdots + {x_n}^2)$$

The following holds: $\lambda_{\min}({x_1}^2 + {x_2}^2 + \cdots + {x_n}^2) = \lambda_{\min} x^T x = \lambda_{\min}\|x\|^2$ and, on the other hand, $\lambda_{\max}({x_1}^2 + {x_2}^2 + \cdots + {x_n}^2) = \lambda_{\max} x^T x = \lambda_{\max}\|x\|^2$ and this proves the fact in the special case of diagonal matrix $Q$.

General case: Let us represent $Q$ through its eigendecomposition: $A = U\Lambda U^{-1} = U\Lambda U^T$, where $U$ is an orthogonal matrix.

$$x^T Q x = x^T U \Lambda U^T x \overset{(1)}{=} y^T \Lambda y$$

where $\overset{(1)}{=}$ is due to the change of variable $y = U^T x$ (that implies $y^T = x^T U$).

By the same argument used in the diagonal case,

$$\lambda_{\min}\|y\|^2 \leq y^T \Lambda y \leq \lambda_{\max}\|y\|^2$$

Now the point is that if we can replace $\|y\|^2$ with $\|x\|^2$ we have proved the theorem. In fact this is true, due to the orthogonality of matrix $U$ ( $UU^T = U^T U = I$.

□

**Corollary 2.14.** *Let $Q \in S(n, \mathbb{R})$ and let $x \in \mathbb{R}^n$. If $x \neq 0$, $\lambda_{min} \leq \frac{x^T Q x}{\|x\|^2} \leq \lambda_{max}$, where $\lambda_{max}$ and $\lambda_{min}$ are respectively the eigenvalue of maximum value and the eigenvalue of minimum value.*

**Definition 2.4** (Positive semidefinite)**.** *Let $Q \in S(n, \mathbb{R})$. If $\lambda_i \geq 0$ for each eigenvalue of $Q$ then $x^T Q x \geq 0$ for each vector $x$. $Q$ is called positive semidefinite.*

$$x^T Q x \geq 0 \, \|x\|^2 \geq 0$$

*This is Iff, so the reverse holds: if $x^T Q x \geq 0$ for all $x$, then eigenvalues of $Q$ are $\geq 0$.*

**Definition 2.5** (Positive definite)**.** *Let $Q \in S(n, \mathbb{R})$. If $\lambda_i > 0$ for each eigenvalue of $Q$ then $x^T Q x > 0$ for each vector $x \neq 0$. $Q$ is called positive semidefinite.*

$$x^T Q x \geq \lambda_{min} \, \|x\|^2 > 0 \, \|x\|^2 = 0$$

*This is Iff, so the reverse holds: if $x^T Q x > 0$ for all $x$, then eigenvalues of $Q$ are $> 0$.*

**Fact 2.15.** *Let $Q \in S(n, \mathbb{R})$. Iff $Q$ is **positive semidefinite** then $\lambda \geq 0 \; \forall \lambda$ eigenvalue of $Q$ iff $Q$ is **positive semidefinite**. On the other hand, all eigenvalues are **strictly** positive iff $Q$ is positive definite.*

*Proof.*
Let's prove that Iff $Q$ is **positive semidefinite** then $\lambda \geq 0 \; \forall \lambda$ by contradiction:
if $Q$ has a eigenvalue $\lambda_i < 0$ then:

$$v_i^T Q v_i = v_i^T \lambda v_i = \lambda_i \, \|v_i\|^2 < 0$$

for the eigenvector $v_i$ associated to $\lambda_i$.

□

**Fact 2.16.** *Let $B \in M(m, n, \mathbb{R})$ (possibly rectangular), $B^T B \in S(n, \mathbb{R})$ is a valid product and gives a square, symmetric matrix and is positive semidefinite.*

*Proof.*

Symmetry: $(B^T B)^T = B^T (B^T)^T = B^T B$

Positive definite: $x^T B^T B x = (Bx)^T (Bx) = \|Bx\|^2 \geq 0$

□

**Corollary 2.17.** *The same holds for $BB^T$, since we can define $C = B^T$.*

**Fact 2.18.** *Let $Q \in S(n, \mathbb{R})$. $A \succeq 0$ and $A$ invertible iff $Q$ is **strictly positive definite**.*

**Something on Matlab . . .**

In order to check if a matrix $A$ is positive definite in Matlab we can look at its eigenvalues (cfr. `eig(A)`).

**NOTE for complex matrices**:
Most of these properties work also for matrices with complex entries, with one change: replace $A^T$ with $\overline{A^T}$ (transpose + entrywise conjugate). Often denoted with $A^*$ or $A^H$.

The norm of a complex vector:

$$\|x\|_2^2 = x^*x = \overline{x_1}x_1 + \overline{x_2}x_2 + \cdots + \overline{x_n}x_n = |x_1|^2 + \cdots + |x_n|^2 \text{ which is always real} \geq 0$$

Some terminology changes:

- $UU^* = I$: unitary matrix (orthogonal + complex)
- $Q = Q^*$: Hermitian matrix (capital letter, after Charles Hermite).

# 3 28th of September 2018 — F. Poloni

## 3.1 Singular value decomposition (SVD)

We are left with the task of reaching a (sort of) "eigenvalue decomposition" when the target matix is not symmetric.
There are two way to generalize the eigenvalue decomposition to a nonsymmetric matrix A (with something that always exists):

**Definition 3.1** (Schur decomposition)**.** *Let $A \in M(n,\mathbb{R})$, $\exists\ U \in M(n,\mathbb{R})$ orthogonal matrix and $T \in M(n,\mathbb{R})$ triangular matrix such that $A = UTU^T$ and this is called* ***Schur decomposition****.*

What is really important for us is the **Singular value decomposition**, every square matrix $A$ can be written with **SVD** form.

Every square matrix A can be written as SVD.

**Definition 3.2** (Singular value decomposition)**.** *Let $A \in M(n,\mathbb{R})$, $\exists\ U,V \in M(n,\mathbb{R})$ orthogonal matrices ($V$ not necessary equal to $U$) and $\Sigma \in Diag(n,\mathbb{R})$ such that $A = U\Sigma V^T$ and this is called* ***Singular Value Decomposition****.*

$$A = \begin{pmatrix} u_1 & u_2 & \cdots & u_n \end{pmatrix} \cdot \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{pmatrix} \cdot \begin{pmatrix} {v_1}^T \\ {v_2}^T \\ \vdots \\ {v_n}^T \end{pmatrix} = \sum_{i=1}^{n} u_i \sigma_i {v_i}^T =$$

$$= u_1\sigma_1 v_1^T + u_2\sigma_2 v_2^T + \cdots + u_n\sigma_n v_n^T$$

*Where $\sigma_i$ are called* ***singular values*** *and they are sorted such that:*

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$$

*General fact on singular values:*

- *Singular values $\neq$ eigenvalues*
- *They are always positive and usually more spread apart than the eigenvalues.*

$$\sigma_1 \geq |\lambda_1| \ \ and \ \ |\lambda_m| \geq \sigma_m$$

*$\lambda_i$ is larger than the largest eigenvalue of a matrix $A$ and $\lambda_m$ is smaller than the smallest eigenvalue of a matrix $A$.*

The SVD can be defined also for a rectangular matrix A:

**Definition 3.3** (Rectangular matrices and SVD)**.** *Let $A \in M(m,n,\mathbb{R})$, there exist $U \in M(m,\mathbb{R})$ orthogonal, $V \in M(n,\mathbb{R})$ orthogonal and $\Sigma(m,n,\mathbb{R})$ diagonal in the sense that $\sum_{ij} = 0$ with $i \neq j$ (padded with zeros). Matrix $A$ has a* ***SVD factorization****, where $\Sigma$ has the following shape:*

- *case $m < n$ (e.g $m = 3, n = 5$)*

$$\begin{pmatrix} \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 \end{pmatrix}$$

- *case $m > n$ ( e.g $m = 5, n = 3$)*

$$\begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

**Definition 3.4** (Thin SVD)**.** *Let $A \in M(m,n,\mathbb{R})$, has a* ***thin SVD factorization****: we may restrict to compute only the first min(m, n) vectors that appear in this sum: thin SVD.*

$$A = \sum_{i=1}^{\min(m,n)} u_i \sigma_i {v_i}^T = u_1\sigma_1 v_1^T + u_2\sigma_2 v_2^T + \cdots + u_{min(m,n)}\sigma_{min(m,n)} v_{min(m,n)}^T$$

**Something on Matlab ...**

In Matlab the SVD decomposition is obtained through the command `svd(A)`, which return value is made of the three matrices $U, \Sigma, V$.
As an example, `[U, S, V] = svd(A)`. Notice that, if `svd(A)` is assigned to one variable, then such variable is an array of singular values.
The thin SVD can be compute with: `[U, S, V] = svd(A, 0)`

**Computational costs**

We are not going into details of algorithms for computing SVD, but we would like to add a consideration about the computational complexity of such an algorithm.

- `[U, S, V] = svd(A, 0)` (thin) costs $O(mn^2)$ ops for $A \in \mathbb{R}^{m\times n}$ or $A \in \mathbb{R}^{n\times m}$ with $m \geq n$
- `[U, S, V] = svd(A)` (non-thin) more expensive, because it has to store the large $m \times m$ factor. (But there are some tricks to store orthogonal matrices compactly, more about it later).

### 3.1.1 Properties of SVD

The SVD reveals rank, image, and kernel of a matrix.

**Definition 3.5** (Rank)**.** *Let $A \in M(n, \mathbb{R})$ we call the **rank** of $A$ the number of non-zero singular values.*

*Equivalently, the **rank** is the size of the column space.*

**Property 3.1.** *A matrix $A \in M(n, \mathbb{R})$ has rank $r$ iff all its eigenvalues starting from the $r+1$-th are 0, formally iff $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0$.*

Thanks to Property 3.1, we can somehow talk about an "even thinner" SVD, where all the 0s in the bottom right part of the matrix $\Sigma$, cancel out the latter columns of $U$ and the latter rows of $V$ (aka columns of $V^T$). A pictorial representation of the shape of $\Sigma$ can be found below.

$$\Sigma = \left(\begin{array}{cccc|c} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \ddots & & \\ & & & \sigma_r & \\ \hline & & & & 0 \\ & & & & & \ddots \\ & & & & & & 0 \\ \hline & & & & \multicolumn{2}{c}{\mathbf{0}} \end{array}\right)$$

This factorization represents $A$ as $\sum_{i=1}^{r} U_i \sigma_i V_i$.

An attentive reader may notice that $Ax = \sum_{i=1}^{r} U_i \sigma_i V_i x$, where the last three terms are dimensionally a scalar. It goes without saying that the image of $A$ is the span of $U_1, U_2, \ldots U_r$, hence $rk(A) = r$.

Moreover, $\ker(A) = span(V_{r+1}, V_{r+2}, \ldots V_n)$, since $V$ is orthgonal (proof: plugging in $x = V_j$, where $j > r$).

**Definition 3.6** (Matrix norm)**.** *Let $A \in M(m, n, \mathbb{R})$. We define the **matrix norm** of $A$ as*

$$\|A\| := \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{z=1} \|Az\|$$

*Where the norm may be any of the ones defined in Definition 1.5 second equality is introduced in order to work in a compact set, the one of normalized vectors $z$.*

Notice that $\|Ax\| \leq \|A\| \cdot \|x\|$.

**Property 3.2.** *Let $A$ and $B \in M(n, m, \mathbb{R})$ and let $x \in \mathbb{R}^n$, the following holds, for any norm defined in Definition 1.5:*

- $\|A\| \geq 0$ *(and the equality holds iff* $A = 0$*);*
- $\|\alpha A\| = |\alpha| \|A\|, \forall \alpha \in \mathbb{R}$*;*
- $\|A + B\| \leq \|A\| + \|B\|$*;*
- $\|AB\| \leq \|A\| \|B\|$*;*
- $\|Ax\| \leq \|A\| \|x\|$.

**Fact 3.3.** *Let* $A \in (n, m, \mathbb{R})$ *and let* $U \in M(m, n, \mathbb{R})$ *orthogonal, in the case of* 2*-norm* $\|A\|_2 = \|AU\|_2 = \|UA\|_2$.

*Proof.* $\|UA\|_2 = \max_{x \in R^n,\ x \neq 0} \frac{\|UAx\|_2}{\|x\|_2} \overset{(1)}{=} \max_{x \in R^n,\ x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \|A\|_2$, where $\overset{(1)}{=}$ follows from a property of vector norms.

$\|AU\|_2 = \max_{x \in R^n,\ x \neq 0} \frac{\|AUx\|_2}{\|x\|_2} \overset{(2)}{=} \max_{y \in R^n,\ y \neq 0} \frac{\|Ay\|_2}{\|y\|_2} = \|A\|_2$, where $\overset{(2)}{=}$ follows from the substitution $y = Ux$. □

**Definition 3.7** (Frobenius norm)**.** *Let* $A \in M(n, m, \mathbb{R})$*, we term* ***Frobenius norm*** *of* $A$ $\|A\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} (A)_{ij}{}^2}$.

Notice that all the properties enumerated in Property 3.2 hold for the Frobenius norm as well.

**Fact 3.4.** *Let* $A \in M(n, m, \mathbb{R})$ *and let* $A = U\Sigma V^T$ *be its singular value decomposition. The following hold:*

1. $\|A\|_2 = \|\Sigma\|_2 = \sigma_1$
2. $\|A\|_F = \|Sigma\|_F = \sum_{i=1}^{\min n,m} \sigma_i{}^2$

*Proof.* 1. The first equality follows from Proposition 3.3, while the second is proved as

follows:

$$\|\Sigma\|_2 = \max_{x\in\mathbb{R}^n,\ x\neq 0} \frac{\|\Sigma x\|_2}{\|x\|_2} = \max_{x\in\mathbb{R}^n,\ x\neq 0} \frac{\left\|\begin{pmatrix}\sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \\ & & 0 & \end{pmatrix}\cdot\begin{pmatrix}x_1\\x_2\\\vdots\\x_n\end{pmatrix}\right\|_2}{\left\|\begin{pmatrix}x_1\\x_2\\\vdots\\x_n\end{pmatrix}\right\|_2} = \max_{x\in\mathbb{R}^n,\ x\neq 0} \frac{\left\|\begin{pmatrix}\sigma_1 x_1\\\sigma_2 x_2\\\vdots\\\sigma_n x_n\\0\\\vdots\\0\end{pmatrix}\right\|_2}{\left\|\begin{pmatrix}x_1\\x_2\\\vdots\\x_n\end{pmatrix}\right\|_2}$$

$$= \frac{\sqrt{(\sigma_1 x_1)^2 + (\sigma_2 x_2)^2 + \cdots + (\sigma_n x_n)^2}}{\sqrt{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2}} \leq \frac{\sqrt{(\sigma_1 x_1)^2 + (\sigma_1 x_2)^2 + \cdots + (\sigma_1 x_n)^2}}{\sqrt{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2}}$$

$$= \sqrt{{\sigma_1}^2}\cdot\frac{\sqrt{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2}}{\sqrt{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2}} = \sigma_1 \tag{3.1}$$

The equality is achieved if we pick $x = e_1 = \begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix}$.

2. The proof of this assertion is similar to the other and it is left to the reader.

□

**Theorem 3.5** (Eckart-Younger)**.** *Let $A \in M(n,m,\mathbb{R})$ and let $A = U\Sigma V^T$ be its singular value decomposition.*

*The solution of* $\min_{rk(X)\leq k} \|A\| - X$ *is given by the* truncated SVD*:*

$$X = \begin{pmatrix}U^1 & U^2 & \cdots & U^k\end{pmatrix}\cdot\begin{pmatrix}\sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_k\end{pmatrix}\cdot\begin{pmatrix}V^1\\V^2\\\vdots\\V^k\end{pmatrix}$$

*Where the norm is both $\|\cdot\|_2$ and $\|\cdot\|_F$.*

**Fact 3.6.** *Let $A \in M(n,\mathbb{R})$ and let $A$ be invertible. The following holds: $\|A^{-1}\| = \frac{1}{\sigma_n}$*

*Proof.* Since $A$ is invertible, none of the $\sigma_i$ is 0, hence the smaller (namely $\sigma_n$) is not 0.

$$A^{-1} = (U\Sigma V^T)^{-1} \overset{(1)}{=} V^{T^{-1}}\Sigma^{-1}U^{-1} = V \begin{pmatrix} \frac{1}{\sigma_1} & & & \\ & \frac{1}{\sigma_2} & & \\ & & \ddots & \\ & & & \frac{1}{\sigma_n} \end{pmatrix} U^T$$

Where $\overset{(1)}{=}$ follows from the orthogonlaity of $V$ and $U$.

Notice that this is *almost* an SVD, because the values on the diagonal are not sorted in a decreasing order.

Plugging in the norm, we have:

$$\left\|A^{-1}\right\| = \left\| V \begin{pmatrix} \frac{1}{\sigma_1} & & & \\ & \frac{1}{\sigma_2} & & \\ & & \ddots & \\ & & & \frac{1}{\sigma_n} \end{pmatrix} U^T \right\| = \left\| \begin{pmatrix} \frac{1}{\sigma_1} & & & \\ & \frac{1}{\sigma_2} & & \\ & & \ddots & \\ & & & \frac{1}{\sigma_n} \end{pmatrix} \right\| = \frac{1}{\sigma_n}$$

□

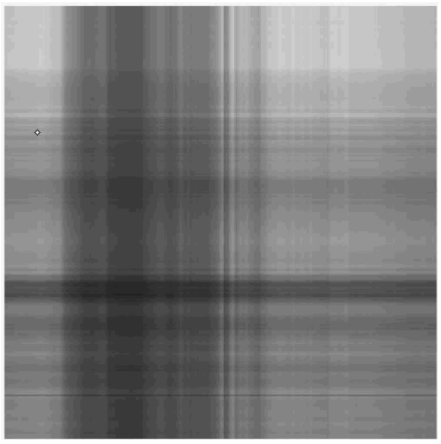
# 4 4th of October 2018 — F. Poloni

This lecture is about practical usage of the singular value decomposition and takes place almost wholly on Matlab.

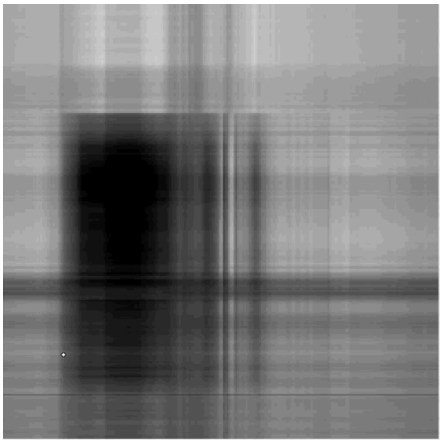
For example, given a certain image, that can be represented as a matrix of values in the range $[0, 255]$, the rank-1 SVD of such image, results in a very abstract picture, see Figure 4.1. The more we increase the rank, the better is the similarity of the approximated image with respect to the original one.

**Something on Matlab ...**

Given a certain matrix $A$, we can compute the SVD decomposition using the command `[U, S, V] = svd(A)`



(A) Rank 1



(B) Rank 2

(C) Rank 5

(D) Full rank

FIGURE 4.1: How the approximation of a matrix changes with respect to the different ranks.

**Definition 4.1** (Principal component analysis)**.** *Given a matrix $A$, we term* ***principal component analysis*** *the analysis of features of such matrix via the rows and columns of $U$ and $V$ respectively, where $U$ and $V$ are the matrices of the SVD decomposition.*

# 5 10th of October 2018 — F. Poloni

This lecture has the goal of introducing the concept of linear combinations.

**Definition 5.1** (Linear combination)**.** *In a very unformal way, we can define the goal of linear combination as the pursuit of obtaining a certain target vector $b \in \mathbb{R}^n$ using $m$ (in principle $m \neq n$) vectors $a_1, a_2, \ldots, a_m$ such that*

$$a_1 x_1 + a_2 x_2 + \cdots + a_m x_m = b$$

*where $x_i$ are properly chosen.*

The task of finding such vectors is called **solving a linear system** and it is formally written as $Ax = b$.

**Theorem 5.1.** *Let $A \in M(n, m)$ and let $b \in \mathbb{R}^n$. It holds that any linear system $Ax = b$ is solvable iff $A$ is invertible.*

We are interested in finding approximate solutions of such systems, where the proximity to the target is expressed in terms as $\|Ax - b\|$ that should be close to zero. A geometric inutition is displayed in Figure 5.1.

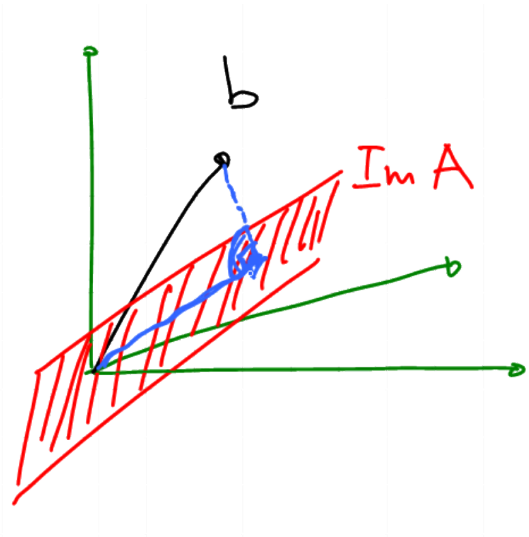



FIGURE 5.1: In this case the image of the matrix $A$ (in red) does not contain $b$ and the best one can do is to obtain a projection of $b$ in the plane $Im(A)$ (drawn in blue).

**Something on Matlab ...**

Matlab provides syntactic sugar to solve linear systems.
Before introducing such syntax let we just notice the following `5\ 2` $(= 2/5) \neq$ `5/2`.
The syntax to solve $Ax = b$ is `A\ b`, where the algorithm used in Matlab is not inverting the matrix $A$ and then performing the multiplication, but it is a more sophisticated and efficient one.

**Definition 5.2** (Linearly square problem)**.** *Let $A \in M(n, m, \mathbb{R})$ and let $b \in \mathbb{R}^n$, we term **linearly square problem** the task of computing $\min_{x \in \mathbb{R}^m} \|Ax - b\|_2$.*

**Something on Matlab ...**

In Matlab the syntax `.func` means that function func should be performed entry by entry of the non-scalar variable.

An example of a practical least square problem may be predicting the salary of NBA players, assuming that the income is obtained as a linear combination of some features.

**Definition 5.3** (Full rank matrix)**.** *Let $A \in M(n, m, \mathbb{R})$ we say that $A$ has **full column rank** if* $\ker A = \{0\}$.

*Equivalently, $rk(A) = n$ or alternatively $\nexists z \in \mathbb{R}^n \backslash \{0\}$ such that $Az = 0$.*

**Fact 5.2.** *Let $A \in M(n, m, \mathbb{R})$, the least square problem $\|Ax - b = 0\|$ has a unique solution iff $A$ has full column rank.*

**Theorem 5.3.** *Let $A \in M(n, m, \mathbb{R})$. $A$ has full column rank iff $A^T A$ is positive definite.*

*Proof.* $A$ has full column rank $\iff \|Az\| \neq 0, \forall z \in \mathbb{R}^m \backslash \{0\} \iff \|Az\|^2 \neq 0, \forall z \in \mathbb{R}^m \backslash \{0\} \iff 0 = (Az)^T Az = z^T A^T Az$ □

# 6 18th of October 2018 — F. Poloni

## 6.1 Least squares problem

**Fact 6.1.** *Given $A \in \mathcal{M}(m,n,\mathbb{R})$ the following conditions are equivalent:*

- *$A^TA$ is positive definite;*
- *$A$ has full column rank;*
- *the columns of $A$ are linear indipendent;*
- *$Ker(A) = \{0\}$.*

**Fact 6.2.** *Given $A \in \mathcal{M}(m,n,\mathbb{R})$, if $A^TA$ is **positive semidefinite** $f(x) = x^TQx - q^Tx + b^Tb$ is **strongly** (or **strictly**) **convex**. In other words, $f(x)$ has a **unique** minimum.*

We find the minimum by solving $\nabla f(x) = 0$,
where $f(x) = x^TA^TAx - 2b^TAx + b^Tb$
$\nabla f(x) = 2A^TAx - 2A^Tb$
How should we compute this gradient?
We know that $f(x+h) = f(x) + (\nabla f(x))^T\dot{h} + o(\|h\|)$

$$\begin{aligned} f(x+h) &= (x+h)^TA^TA(x+h) - 2b^TA(x+h) + b^Tb \\ &= \mathbf{x^TA^TAx} + \mathbf{x^TA^TAh} + \mathbf{h^TA^TAx} + \mathbf{h^TA^TAh} \mathbf{-2b^TAx} - 2b^TAh + \mathbf{b^Tb} \\ &= \mathbf{f(x)} + (\mathbf{2x^TA^TAh} - 2b^TAh) + \mathbf{o(\|h\|)} \\ &= f(x) + \mathbf{(A^TAx - 2A^Tb)^Th} + o(\|h\|) \end{aligned} \tag{6.1}$$

So, $\mathbf{\nabla f(x) = A^TAx - 2A^Tb}$
We would like to know when the gradient is 0.
$\nabla f(x) \stackrel{?}{=} 0 \Leftrightarrow A^TAx = A^Tb$
Since $A^TA$ is a **square** matrix and also **non singular** (which means invertible) we may find $x$ by solving a linear system $x = (A^TA)^{-1}(A^Tb)$ via:

- Gaussian elimination;
- LU factorization;
- QR factorization;
- Cholesky factorization (specialyzed method for positive definite matrices) Idea: $A^TA$ can be written as $A^TA = R^TR$, where $R$ is a square, upper triangular matrix.

Why do we need factorization method? Let's compute complexity:

- $A^TA \rightarrow 2mn^2$, where $m > n$;
- $A^Tb \rightarrow 2mn$;
- Solving $A^TAx = A^Tb$ with gaussian elimination has a computational complexity of $\frac{2}{3}n^3$;
- Cholesky factorization $A^TA = R^TR$ which has a cost of $\frac{1}{3}n^3$

### 6.1.1 Method of normal equations

This method solves least squares problem and takes his name from the fact that "normal" means orthogonal.

The key idea is using symmetry to skip half of the entries of $A^TA$

If $Ax = b$ can't be solved, since $A$ is tall and thin, we can multiply on both sides by $A^T$ and try again, since the matrix is square now.

The residual $Ax - b$ is orthogonal to any vector $v \in span(A) : (Av)^T(Ax - b) = 0$

Why?

$v^T(A^TAx - A^Tb) = 0$, see Figure 6.1.

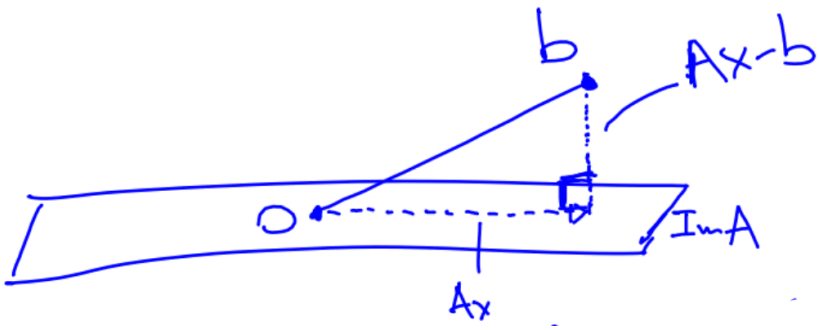



FIGURE 6.1: Geometric idea.

It's possible to find a close formula for solving this problem: $\min\|Ax - b\|$ is given by $x = (A^TA)^{-1}A^Tb$.

**Definition 6.1** (Moore-Penrose pseudoinverse)**.** *Let $A$ be a matrix in $\mathcal{M}(n, m, \mathbb{R})$, the* ***Moore-Penrose pseudoinverse*** *of $A$ with full column rank is* $\mathbf{A}^+ := (A^TA)^{-1}A^T$

So we can write $x = A^+b$ for the solution of a LS problem.

In particular, the solution of $\|\min Ax - (b_1 + b_2)\|$ is the sum of two solutions of $\min\|Ax_1 - b_1\|$ and $\min\|Ax_2 - b_2\|$.

**Obs:** $AA^+ = A(A^TA)^{-1}A^T = AA^{-1}A^{T^{-1}}A^T = I_{n\times m}$, but this doesn't hold for $A^+A = (A^TA)^{-1}(A^TA)$, which is not the identity matrix.

## 6.2 QR factorization

**Theorem 6.3.** $\forall A \in \mathcal{M}(m,m,\mathbb{R})$, $\exists Q \in \mathcal{O}(m,m,\mathbb{R})$ *(space of orthogonal matrices of size* $m \times m$*),* $\exists R \in \mathcal{M}(m,m,\mathbb{R})$ *upper triangular such that* $A = QR$

QR factorization isn't as powerful as SVD factorization.
Why are we interested in studying factorizations?

- They reveal properties: singularity, rank, ...;
- They may be an intermediate step in algorithms.

**Example 6.1.** *We would like to solve* $Ax = b$*, with* $A \in \mathcal{M}(m,m,\mathbb{R})$ *we may:*

1. *first compute the QR factorization* $(A = QR)$ *and then obtain* $x = A^{-1}b = R^{-1}Q^{-1}b$
2. *compute then* $c = Q^T b$
3. *and then* $x = R^{-1}c$

What's the computational cost?

1. $QR \rightarrow O(m^3)$
2. compute c $\rightarrow O(m^2)$
3. compute x $\rightarrow O(m^2)$

Let's analyze the case in which $A$ is a vector. Given $x \in \mathbb{R}^m$, we want to find an orthogonal matrix $Q$ such that $Qx$ has the form $\begin{pmatrix} s \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{pmatrix} = se_1$, where $s = \pm\|x\|$.

We denote $e_i$ the i-th column of the identity matrix.

**Definition 6.2** (Housholder reflector)**.** *Let* $U$ *be a vector in* $\mathbb{R}^m$*. An* ***Householder reflector*** *is a matrix* $H$ *such that* $H = I - \frac{2}{U^T U}UU^T$*.*

*We can observe that* $U^T U$ *is a scalar.*

*Since* $U^T U = \|U\|^2$*, another way of seeing* $H$ *may be* $H = I - \frac{2}{\|U\|^2}UU^T = I - 2vv^T$*, where* $v = \frac{1}{\|U\|}U$

**Lemma 6.4.** *Matrices of this form are orthogonal.*

*Proof.*

$$\begin{aligned} HH^T &= (I - \frac{2}{\|U\|^2}UU^T)(I - \frac{2}{\|U\|^2}UU^T) \\ &= II - \frac{2}{\|U\|^2}UU^T I - I\frac{2}{\|U\|^2}UU^T + \frac{2}{\|U\|^2}UU^T\frac{2}{\|U\|^2}UU^T \\ &= I - \frac{2}{\|U\|^2}UU^T - \frac{2}{\|U\|^2}UU^T + \frac{4}{\|U\|^4}UU^TUU^T \\ &= I - \frac{4}{\|U\|^2}UU^T + \frac{4}{\|U\|^4}U\|U\|^2U^T \\ &= I \end{aligned} \tag{6.2}$$

□

**Example 6.2.** $Hx = (I - \frac{2}{\|U\|^2}UU^T)x = x - \frac{2}{\|U\|^2}U(U^TU)$

$y = compute\_product(U, x)$

$a = U' * x$

$b = U' * U$

$y = x\frac{2*a}{b}U$

*All these operations are linear operations, so the complexity is $O(m)$, cheaper than generic matrix-vector product ($O(m^2)$).*

Can we find a matrix $H$ in this family that maps $x$ to $y$ (equivalently $Hx = y$)? The answer is given by the following lemma

**Lemma 6.5.** *$\forall x, y$ s.t $\|x\| = \|y\| \exists H$ s.t. $Hx = y$, choosing $u = x - y$.*

A geometric idea is given by the following:

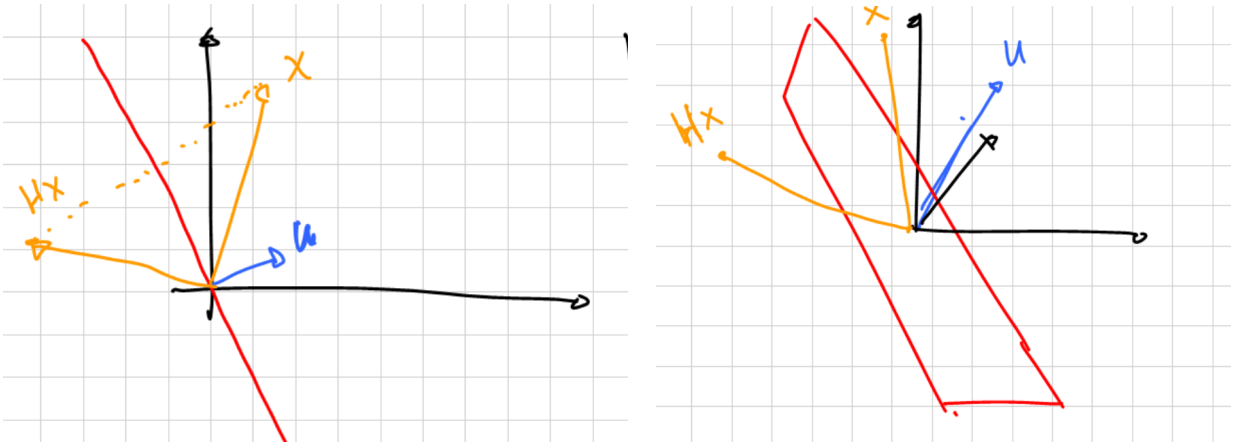



FIGURE 6.2: On the left in $\mathbb{R}^2$, on the right in $\mathbb{R}^3$.

What happens if $y = \begin{pmatrix} \|x\| \\ 0 \\ . \\ . \\ . \\ 0 \end{pmatrix}$?

$y = H^T x$, actually $H^T = H^{-1} = H$

Let's map $x$ to $y$:

$$U = x - y = \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{pmatrix} - \begin{pmatrix} s \\ 0 \\ . \\ . \\ . \\ 0 \end{pmatrix} = \begin{pmatrix} x_1 - s \\ x_2 \\ . \\ . \\ . \\ x_n \end{pmatrix}$$

where $s = \|x\|$.

**Something on Matlab ...**

`function[v,s] = householder_vector(x)`, where v and s are the returned values and x is the argument.

# 7 26th of October 2018 — F. Poloni

## 7.1 How to construct a QR factorization

In the previous lecture we introduced the $QR$ factorization and we defined what an Householder reflector is.

At the end of the lecture we gave a first MatLab implementation of `householder_vector`:

ALGORITHM 7.1 Householder vector Matlab implementation.

```
1        function[v,s] = householdervector(x)
2        s = norm(x);
3        v = x;
4        v(1) = v(1) - s;
5        v = v / norm(v);
```

What's the problem of this algorithm? That the subtraction may create a problem with machine numbers, if $s$ and $\|x\|$ are very close. If we take $\|x\| = -s$ the subtraction becomes and addition, and everything works well.

In the end, we would like to obtain this behaviour for every possible value for $x$ and $s$, so line 2 may be modified as `s = - sign(x(1)) * norm(x)`.

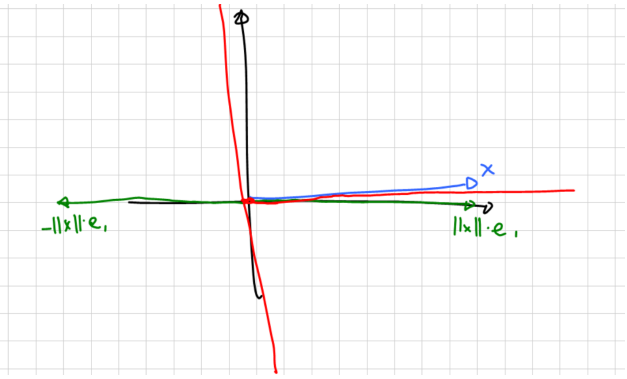



FIGURE 7.1: If $x$ is oriented as in the plot it's better if we choose $-\|x\|e_1$ verse, since it's opposite to $x$.

**Example 7.1.** *Given* $A = \begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix} \in \mathcal{M}(m, m, \mathbb{R})$, *where* $m = 5$, *we would like to calculate the* $QR$ *factorization of* $A$.

STEP 1 : *costruct a Householder matrix that sends* $A(:,1)$ *(first column of A) to a multiple of* $e_1$. *Then we have* $H_1A = \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix}$

STEP 2 : *take* $H_2 \in \mathcal{M}(m-1, m-1, \mathbb{R})$ *such that* $H_2A(2:end, 2) = \begin{pmatrix} \times \\ 0 \\ 0 \\ 0 \end{pmatrix}$ *and compute:*

$$\begin{aligned} Q_2(H_1A) &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & & & & \\ & & & H_2 & \\ 0 & & & & \\ 0 & & & & \end{pmatrix} \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix} \\ &= \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix} \end{aligned} \tag{7.1}$$

*And we denote* $Q_2 = \begin{pmatrix} I_{1\times 1} & 0 \\ 0 & H_2 \end{pmatrix}$, $Q_1 = H_1$;

STEP 3 : *take* $H_3 \in \mathcal{M}(m-2, m-2, \mathbb{R})$ *such that* $H_3A(3:end, 3) = \begin{pmatrix} \times \\ 0 \\ 0 \end{pmatrix}$ *and we compute:*

$$\begin{aligned} Q_3(Q_2Q_1A) &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & & & \\ & & & H_3 & \\ 0 & 0 & & & \\ 0 & 0 & & & \end{pmatrix} \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix} \\ &= \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix} \end{aligned} \tag{7.2}$$

*So,* $Q_3 = \begin{pmatrix} I_{2\times 2} & 0 \\ 0 & H_3 \end{pmatrix}$;

STEP 4 : *take* $H_4 \in \mathcal{M}(m-3, m-3, \mathbb{R})$ *such that* $H_4 A(4:end, 4) = \begin{pmatrix} \times \\ 0 \end{pmatrix}$ *and we compute:*

$$\begin{aligned} Q_4(Q_3Q_2Q_1A) &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & & \\ 0 & 0 & & H_4 & \\ 0 & 0 & & & \end{pmatrix} \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix} \\ &= \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{pmatrix} \end{aligned} \tag{7.3}$$

*Where,* $Q_4 = \begin{pmatrix} I_{3\times 3} & 0 \\ 0 & H_4 \end{pmatrix}$.

*In the end, since* $Q_i$ *is an orthogonal matrix and the product of orthogonal matrices is orthogonal,* $Q_1Q_2Q_3Q_4A = T$*, which is an upper triangular matrix.*

**Theorem 7.1** (Product of block matrices)**.** *Let* $I \in \mathcal{M}(k, k, \mathbb{R})$*, let* $H_i \in \mathcal{M}(m-k, m-k, \mathbb{R})$ *and let* $B_i \in \mathcal{M}(k, k, \mathbb{R})$*,* $C_i \in \mathcal{M}(k, m-k, \mathbb{R})$ *and* $A_i \in \mathcal{M}(m-k, m-k, \mathbb{R})$*, then the product between the two following block matrices is exactly the one showed below.*

$$\begin{pmatrix} I & 0 \\ 0 & H_i \end{pmatrix} \begin{pmatrix} B_i & C_i \\ 0 & A_i \end{pmatrix} = \begin{pmatrix} B_iI & C_i \\ 0 & H_iA_i \end{pmatrix}$$

*Proof.* It's trivial computation, using the definition of matrix product. □

### 7.1.1 Matlab implementation

ALGORITHM 7.2 First implementation of QR factorization.

```
function [Q, R] = myqr(A)
[m, n] = size(A);
Q = eye(m);
for j = 1:n
    v = householder_vector(A(j:end, j));
    H = eye(length(v)) - 2*v*v';
    A(j:end,j:end) = H * A(j:end,j:end);
    Q(:, j:end) = Q(:, j:end) * H;
end
R = A;
```

**Fact 7.2.** *The cost of this implementation when $A$ is a square matrix is $O(n^3 + (n-1)^3 + \cdots + 1^3)$. If $A$ is a rectangular matrix, then the computational complexity is $O(m \cdot n^2 + (m-1) \cdot (n-1)^2 + \cdots + (m-n+1)^3)$.*

*Proof.* Line 7 does a matrix product between matrices of size $n$, $n-1$, ..., 1, so the resulting cost is $O(m \cdot n^2 + (m-1) \cdot (n-1)^2 + \cdots + (m-n+1)^3)$.

□

We may design a faster algorithm, since $HA_j = A_j - 2v(v^T A_j)$.

ALGORITHM 7.3 More efficient implementation of QR factorization.

```
1  function [Q, A] = myqr(A)
2  [m, n] = size(A);
3  Q = eye(m);
4  for j = 1:n-1
5      [v, s] = householder_vector(A(j:end, j));
6      A(j,j) = s; A(j+1:end,j) = 0;
7      A(j:end,j+1:end) = A(j:end,j+1:end) - ...
8          2*v*(v'*A(j:end,j+1:end));
9      Q(:, j:end) = Q(:, j:end) - Q(:,j:end)*v*2*v';
10 end
```

Let's suppose that $A$ is square matrix, partitioned as:

$$A = \begin{pmatrix} A_1 & A_2 \end{pmatrix} = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_{1,1} & R_{1,2} \\ 0 & R_{2,2} \end{pmatrix}$$

Then we can recover the factorization of $A_1$ from the factorization of $A$, since $A_1 = Q \cdot R_{11}$.

**Fact 7.3** (Thin QR factorization)**.** *We may replace $Q \in \mathcal{M}(m, m, \mathbb{R})$ and $R \in \mathcal{M}(m, m, \mathbb{R})$ with $Q_1 \in \mathcal{M}(m, n, \mathbb{R})$ and $R_1 \in \mathcal{M}(n, n, \mathbb{R})$ and the same factorization holds: $A = QR = Q_1 R_1$. This is called* ***thin QR factorization****.*

*Proof.* $A_1 \in \mathcal{M}(m, n, \mathbb{R})$, $A = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \cdot \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = QR = Q_1 R_1 + Q_2 0 = Q_1 R_1$ □

In order to save space we may work in the following way:

$$QB = \begin{pmatrix} 1 & \times & \cdots & \times \\ 0 & & & \\ \vdots & & \text{I-2V}_1\text{V}_1^T & \\ 0 & & & \\ 0 & & & \end{pmatrix} \begin{pmatrix} 1 & \times & \times & \cdots & \times \\ 0 & 1 & \times & \cdots & \times \\ \vdots & 0 & & \text{I-2V}_2\text{V}_2^T & \\ 0 & \vdots & & & \\ 0 & 0 & & & \end{pmatrix} \cdots$$

$$\dots \begin{pmatrix} 1 & \times & \times & \cdots & & \times \\ 0 & 1 & \times & \cdots & & \times \\ \vdots & 0 & 1 & & & \\ 0 & \vdots & 0 & & \text{I-2V}_n\text{V}_n{}^T & \\ 0 & 0 & 0 & & & \end{pmatrix} B.$$

**Fun fact**

There are some libraries that store the $v_i$ vectors in the lower part of matrix $R$ which is upper triangular and has only zeros below the main diagonal.

## 7.2 How to use the thin QR factorization to solve a least squares problem

We would like to solve $\|Ax - b\|$ $\forall A \in \mathcal{M}(m, n, \mathbb{R})$ and $\forall B \in \mathbb{R}^n$ where $m > n$ (aka $A$ is a tall, thin matrix),through the $QR$ factorization. We would like to solve $\min \|Ax - b\|$ through the $QR$ factorization.

We may write first the $QR$ factorization of $A$, so $\forall A \in \mathcal{M}(m, n, \mathbb{R})$, $\exists Q \in \mathcal{M}(m, m, \mathbb{R})$, $\exists R \in \mathcal{M}(m, n, \mathbb{R})$ such that $A = QR$, where $Q = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix}$ and $R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$.

Then,

$$\begin{aligned} \|A\mathbf{x} - \mathbf{b}\| &= \left\|Q^T(A\mathbf{x} - \mathbf{b})\right\| = \left\|Q^TQR\mathbf{x} - Q^T\mathbf{b}\right\| \\ &= \left\|R\mathbf{x} - Q^T\mathbf{b}\right\| = \left\|\begin{pmatrix} R_1 \\ 0 \end{pmatrix}\mathbf{x} - \begin{pmatrix} {Q_1}^T \\ {Q_2}^T \end{pmatrix}\mathbf{b}\right\| \\ &= \left\|\begin{pmatrix} R_1\mathbf{x} - {Q_1}^T\mathbf{b} \\ {Q_2}^T\mathbf{b} \end{pmatrix}\right\| \end{aligned} \tag{7.4}$$

How can we pick $\mathbf{x}$ to minimize the norm of $A\mathbf{x} - \mathbf{b}$?

Can we choose $x$ such that $R_1\mathbf{x} - {Q_1}^T\mathbf{b} = 0$? Yes, we can, since this is a linear square system, so $\mathbf{x} = {R_1}^{-1}{Q_1}^T\mathbf{b}$

$\|A\mathbf{x} - \mathbf{b}\| = \left\|{Q_2}^T\mathbf{b}\right\|$

We used the fact that $R_1$ is invertible, but is it always true that $R_1$ is invertible?

**Lemma 7.4.** *$R_1$ is invertible $\Leftrightarrow$ $A$ has full column rank.*

*Proof.* $A$ has full column rank $\Leftrightarrow$ $A^TA$ is positive definite $\Leftrightarrow$ $A^TA$ is positive semidefinite and invertible, but $A^TA$ is positive semidefinite, so we only need to prove its invertibility.

Let's compute $QR^TQR = R^TQ^TQR = R^TR = \begin{pmatrix} {R_1}^T & 0 \end{pmatrix}\begin{pmatrix} R_1 \\ 0 \end{pmatrix} = {R_1}^TR_1$.

So, $A^TA$ is invertible $\Leftrightarrow$ $R_1$ is invertible. □

**Note**

${R_1}^T R_1$ is the Cholesky factorization of $A^T A$.

The computational complexity is asymptotically equal to the one of computing the $QR$ factorization, since the other operations are cheaper (the product $Q_1^T b$ costs $O(mn)$ and solving the triangular linear system by back-substitution costs $O(n^2)$).

# 8 7th of November 2018 — F. Poloni

## 8.1 Least squares problem with SVD

**Do you recall?**

**Tall thin SVD:** A matrix $A$ can be written as $A = USV^T$, where $U$ is orthogonal, $S$ is a diagonal matrix and $V$ is orthogonal as well. In the case of a tall, thin $A$ the decomposition has the following shape:

$$\begin{pmatrix} & & & \\ U^1 & U^2 & \cdots & U^m \\ & & & \end{pmatrix} \Sigma \begin{pmatrix} & & & \\ V^1 & V^2 & \cdots & V^n \\ & & & \end{pmatrix}^T$$

where

$$\Sigma = \begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & & & & & \sigma_n \end{pmatrix}$$

And if we denote $U_1$ the matrix obtained as the first $n$ columns of $U$ we have the tall, thin SVD: $A = U_1 \Sigma V^T$

We would like to see how we can solve a least squares problem through $SVD$:

$$
\begin{aligned}
\|Ax-b\| &= \left\|USV^Tx-b\right\| && \leftarrow \text{ Def. of SVD}\\
&= \left\|U^T(USV^Tx-b)\right\| && \leftarrow U^T\text{is orthogonal}\\
&= \left\|SV^Tx-U^Tb\right\| && \leftarrow \text{Distributivity + orthogonality}\\
&= \left\|Sy-U^Tb\right\| && \leftarrow y=V^Tx\\
&= \left\| \begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & & & & \sigma_n \\ & & 0 & & \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_n \end{pmatrix} - \begin{pmatrix} U^{1^T}b \\ U^{2^T}b \\ \vdots \\ U^{n^T}b \\ U^{n+1^T}b \\ \vdots \\ U^{m^T}b \end{pmatrix} \right\| && (8.1)\\
&= \left\| \begin{pmatrix} \sigma_1y_1 - U^{1^T}b \\ \sigma_2y_2 - U^{2^T}b \\ \vdots \\ \sigma_ny_n - U^{n^T}b \\ \sigma_{n+1}y_{n+1} - U^{n+1^T}b \\ \vdots \\ \sigma_my_m - U^{m^T}b \end{pmatrix} \right\|
\end{aligned}
$$

Where the first $n$ rows may be assigned to 0 iff $y_i = -\frac{U^{i^T}b}{\sigma_1}$ (if $\sigma_i \neq 0 \forall i$), while the latter $m-n$ do not depend on $y$. This process produces a solution $y$, but the variable change may

be inverted, so

$$
\begin{aligned}
x &= Vy && \leftarrow \text{Orthogonality of } V \\
&= V^1 y_1 + V^2 y_2 + \cdots + V^n y_n \\
&= V^1 \frac{1}{\sigma_1} U^{1T} b + V^2 \frac{1}{\sigma_2} U^{2T} b + \cdots + V^n \frac{1}{\sigma_n} U^{nT} b \\
&= V \begin{pmatrix} \frac{1}{\sigma_1} & & & & \\ & \frac{1}{\sigma_2} & & & \\ & & \ddots & & 0 \\ & & & \ddots & \\ & & & & \frac{1}{\sigma_n} \end{pmatrix} U^T b \\
&= V \begin{pmatrix} \frac{1}{\sigma_1} & & & & \\ & \frac{1}{\sigma_2} & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & \frac{1}{\sigma_n} \end{pmatrix} {U_1}^T b
\end{aligned}
\tag{8.2}
$$

Which depends only on the tall, thin SVD.

**Fact 8.1.** *The $\sigma_i$ are different from 0 iff $A$ has full column rank.*

*Proof.* $A$ has full column rank

$\Updownarrow$

$A^T A$ is invertible

$\Updownarrow$

$$
(USV^T)^T (USV^T) = VS^T U^T USV^T = V \begin{pmatrix} \sigma_1^2 & & & & \\ & \sigma_2^2 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & \sigma_n^2 \end{pmatrix} V^T \text{ is invertible}
$$

$\Updownarrow$

$\forall i \ \sigma_i \neq 0$

□

**Observation 8.1.** *This lemma also proves that the factorization is also a $QR$ factorization.*

**Note on Matlab syntax**

`svd(A, 0)` and `qr(A, 0)` express that we are only interested in the parts of the factorization without zeros, in case of a tall, thin matrix $A$.

We may observe that the computational complexity is $O(15n^3)$ for square matrices, while it's $O(mn^2)$ in the tall, thin case.

### 8.1.1 Behaviour in case of zeros as singular values

What happens when there are some zeros as singular values?

**Do you recall?**

We may recall that the singular values are ordered on the diagonal in decreasing order (the largerst in top left position). From this assumption, we may say that if there are some $\sigma_i = 0$ then they are in the bottom right part of the matrix.

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0$$

We also recall the following $\|Ax - b\| = \left\| \begin{pmatrix} \sigma_1 y_1 - U^{1T} b \\ \sigma_2 y_2 - U^{2T} b \\ \vdots \\ \sigma_n y_n - U^{nT} b \\ 0 \cdot y_{n+1} - U^{n+1T} b \\ \vdots \\ 0 \cdot y_m - U^{mT} b \end{pmatrix} \right\|$.

No matter what value we choose for $y_{r+1} \cdots y_n$, the value doesn't change since it's multiplied by 0. Therefore we get infinite solutions of the form $y = \begin{pmatrix} -\frac{U^{1T} b}{\sigma_1} \\ -\frac{U^{2T} b}{\sigma_2} \\ \vdots \\ -\frac{U^{rT} b}{\sigma_r} \\ * \end{pmatrix}$

We would like to make the solution unique, so we can modify the problem:

- taking the value that minimize the norm:

$$\min_{x \in \arg\min(\|Ax-b\|)} \|x\| . \tag{P2}$$

  Note that $\|x\| = \|y\|$, because $x = Vy$. It follows from the expression of $y$ that the choice that minimizes its norm is $y_{r+1} = \cdots = y_n = 0$.

-

The solution of $P2$ is given by

$$Vy = \begin{pmatrix} V^1 & V^2 & \cdots & V^n \end{pmatrix} \begin{pmatrix} -\frac{U^{1^T}b}{\sigma_1} \\ -\frac{U^{2^T}b}{\sigma_2} \\ \vdots \\ -\frac{U^{r^T}b}{\sigma_r} \\ 0 \end{pmatrix} = V^{1^T}\frac{1}{\sigma_1}U^{1^T}b + \cdots + V^{r^T}\frac{1}{\sigma_r}U^{r^T}b$$

What happens when working with machine precision? Let's make an example where $r = n-1$, so only the last singular value is 0. If the check of $\sigma_n = 0$ fails, $\frac{1}{\sigma_n}$ becomes very big. A way to circumvent this problem is to find the linear dependencies between the columns, so that the algorithm works correctly.

## 8.2 Truncated SVD

In many real world setups first singular components correspond to the most prominent features of the dataset, while the smallest ones are fine details and noise. Note, though, that in the sum $\sum_{i=1}^{n} V^i \frac{U^{i^T}b}{\sigma_i}$ the small singular values may have a large impact, because $\sigma_i$ is in the denominator.

We can modify the solution to cope with real world data problems:

$x = \sum_{i=1}^{n} V^i \frac{U^{i^T}b}{\sigma_i} \quad \longrightarrow \quad x_{trunc} = \sum_{i=1}^{k} V^i \frac{U^{i^T}b}{\sigma_i}$

for a certain $k$, ignoring small singular values.

Another way to modify the problem is the following.

## 8.3 Tikhonov regularization / ridge regression

The Tikhonov regularization is a smoother version of truncated SVD.

$$x_{Tik} = \arg\min_{x \in \mathbb{R}^n} \|Ax - b\|^2 + \alpha^2 \|x\|^2$$

**Fact 8.2.** *The Tikhonov regularization is equivalent to*

$$x_{Tik} = \arg\min_{x \in \mathbb{R}^n} \left\| \begin{pmatrix} A \\ \alpha I \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|^2 \tag{8.3}$$

We show that the two objective functions coincide.

*Proof.*

$$\begin{aligned}\left\|\begin{pmatrix}A\\ \alpha I\end{pmatrix}x-\begin{pmatrix}b\\ 0\end{pmatrix}\right\|^2 &= \left\|\begin{pmatrix}Ax\\ \alpha x\end{pmatrix}-\begin{pmatrix}b\\ 0\end{pmatrix}\right\|^2\\ &= \left\|\begin{pmatrix}Ax-b\\ \alpha x\end{pmatrix}\right\|^2\\ &= \|Ax-b\|^2+\|\alpha x\|^2\\ &= \|Ax-b\|^2+\alpha^2\|x\|^2\end{aligned} \tag{8.4}$$

□

**Fact 8.3.** *The solution of the Tikhonov regularization is given by the formula* $x_{Tik} = (A^TA+\alpha^2 I)^{-1}A^Tb$.

*Proof.* We start by writing the explicit solution of Equation (8.3) using the pseudoinverse

$$\begin{aligned}\begin{pmatrix}A\\ \alpha I\end{pmatrix}^+\begin{pmatrix}b\\ 0\end{pmatrix} &= \left(\begin{pmatrix}A\\ \alpha I\end{pmatrix}^T\begin{pmatrix}A\\ \alpha I\end{pmatrix}\right)^{-1}\begin{pmatrix}A\\ \alpha I\end{pmatrix}^T\begin{pmatrix}b\\ 0\end{pmatrix}\\ &= \left(\begin{pmatrix}A^T & \alpha I\end{pmatrix}\begin{pmatrix}A\\ \alpha I\end{pmatrix}\right)^{-1}\begin{pmatrix}A^T & \alpha I\end{pmatrix}\begin{pmatrix}b\\ 0\end{pmatrix}\\ &= (A^TA+\alpha^2 I)^{-1}A^Tb.\end{aligned} \tag{8.5}$$

□

**Fact 8.4.** *We can observe that* $(A^TA+\alpha^2 I)$ *is positive definite.*

*Proof.* $z^T\cdot(A^TA+\alpha^2 I)\cdot z = z^TA^TAz+\alpha^2 z^Tz \overset{(1)}{=} \alpha^2 z^Tz = \alpha^2\|x\|^2 > 0$. The equality (1) is obtained because $z^TA^TAz \geq 0$, since $A^TA$ is positive semidefinite. □

**Exercise 8.1.** *Show using the SVD of A that the Tikhonov / Ridge solution can be written as*

$$x = \sum_{i=1}^{n} V^i \frac{\sigma_i}{\sigma_i^2+\alpha^2} U^{iT} b.$$

*When* $\sigma_i \gg \alpha$, $\frac{\sigma_i}{\sigma_i^2+\alpha^2} \approx \frac{1}{\sigma_i}$*: similar to the 'true' solution.*
*When* $\sigma_i \ll \alpha$, $\frac{\sigma_i}{\sigma_i^2+\alpha^2} \approx \frac{\sigma_i}{\alpha^2} \approx 0$*: approximately ignoring small singular values.*

How can we choose $k$?

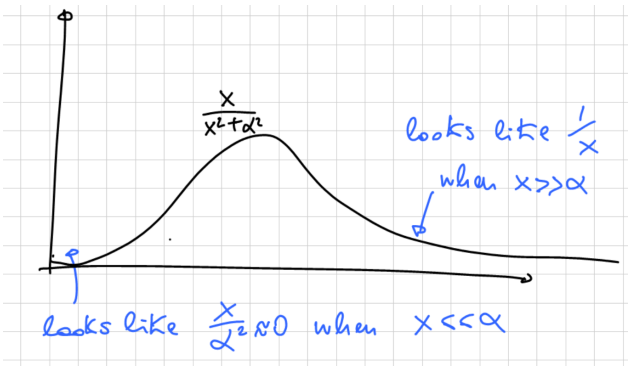



FIGURE 8.1: Here is the shape of the formula for the singular values.

# 9 9th of November 2018 — F. Poloni

## 9.1 Conditioning

Two lectures ago we introduced the QR factorization to solve least squares problems and we noticed that it has a computational complexity which is much worse thant the normal equations method.

Why did we introduce the QR factorization to solve the least squares problem, then? Altough it's more complex computationally speaking, it's much better than the normal equation for what concerns accuracy. Let's see why through an example:

**Example 9.1.** *Let $A \in \mathcal{M}(4,3,\mathbb{R})$ s.t.*

$$A = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 3 & 1 & 4 \\ 1 & 2 & 3+10^{-8} \end{pmatrix}$$

*In this case the normal equations method doesn't even find the order of magnitude correctly.*

At this point we may indroduce the problem of **sensitivity**:

**Definition 9.1** (Sensitivity)**.** *We call **sensitivity** the measure of how much the output of a problem changes when we perturb its input.*

*As an example, let $f(x,y) = x + 2y$. If we perturb the second parameter of $f$ as follows $\widetilde{y} = y + \delta$, we can compute the variation in the output value of the function $\boldsymbol{v}$ as*

$$\mathbf{v} = f(x,\widetilde{y}) - f(x,y) = x + 2(y+\delta) - (x+2y) = 2\delta$$

A good example of this behaviour is the temperature of water coming from the shower: in particular, when we rotate little the knob the water becomes too cold or too hot very fast. This function is plotted in Figure 9.1.

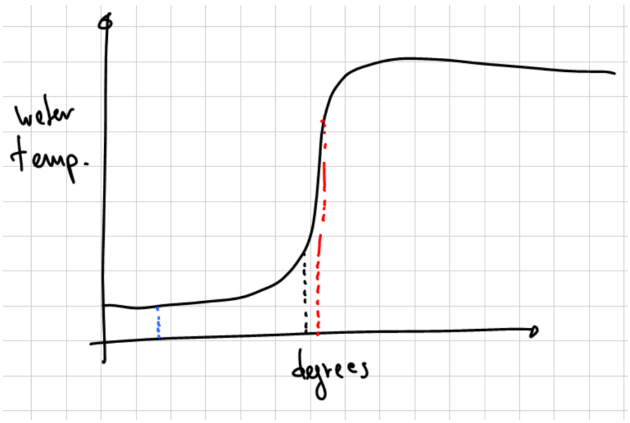



FIGURE 9.1: Geometric idea of temperature of the water in the shower

**Definition 9.2** (Absolute condition measure)**.** *The* ***absolute condition number*** *of a function $f$ is the* ***maximum*** *possible output change / input change ratio in the limit for a* ***small*** *change of the input.*

$$\kappa_{abs}(f,x) = \lim_{\varepsilon \to 0} \sup_{|\tilde{x}-x| \leq \varepsilon} \frac{|f(\tilde{x}) - f(x)|}{|\tilde{x} - x|}$$

We would like to focus on what this definition means.

Why are we interested in the limit of a very small change?

If we zoom-in a continuous function is gets basically linear (key idea of derivative) and then the ratio between the difference on the outputs and the one of the inputs is approximatively the derivative, as shown in Figure 9.2.

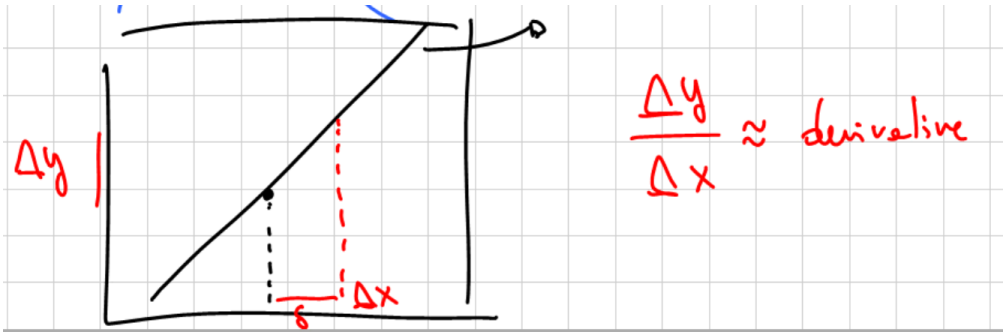



FIGURE 9.2: Geometric idea behind the derivative, as "zoom" of the function in a certain point.

We take a point $x$ and we cosider a ball of radius $\varepsilon$ and we compute the change in the output over the change in the input, then we take the maximum.

**Example 9.2.** *Let* $f(x) = x^2$.

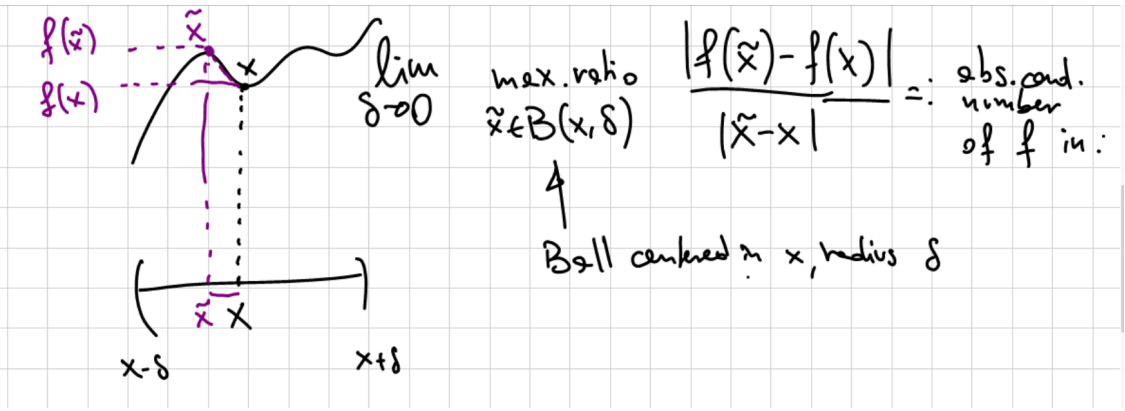



FIGURE 9.3: Geometric idea behind absolute condition number.

*If we perturb the input $x$ to $x+\delta$, then $f(\tilde{x}) = (x+\delta)^2 = x^2 + 2x\delta + \delta^2$, then we obtain the ratio*

$$r = \frac{|f(\tilde{x}) - f(x)|}{|\tilde{x} - x|} = \frac{|2x\delta + \delta^2|}{|\delta|} = |2x + \delta|$$

*If we denote $\varepsilon$ the radius of the ball, we obtain the following*

$$\max_{\substack{|\delta|<|\varepsilon| \\ \tilde{x}\in B(x,\varepsilon)}} r = |2x| + |\varepsilon|$$

*then*

$$\lim_{\varepsilon\to 0} \max_{|\delta|<|\epsilon|} r = \lim_{\varepsilon\to 0} |2x| + |\varepsilon| = |2x|$$

**Example 9.3.** *It's more interesting to see a multivariate function:*
*Let $f(x) = x^T Q x$, for instance for $x \in \mathbb{R}^2$ so that we can plot its graph in $\mathbb{R}^3$.*

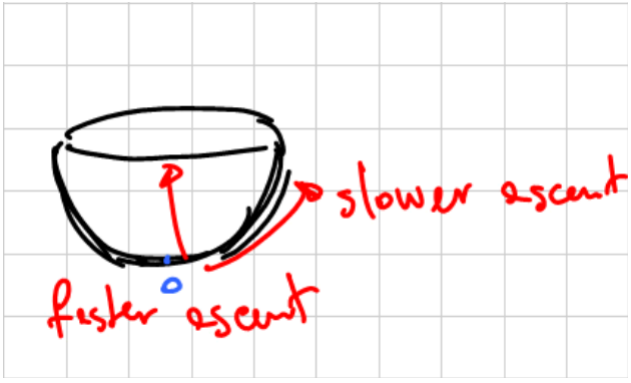



FIGURE 9.4: Paraboloid

*We shall take a general example where the cross-section are ellipses, so that there is a direction of faster and slower ascent; this is not just a circular "cup" seen in perspective. Note that these directions of faster ascent and lower ascent correspond to the eigenvectors of the matrix $Q$.*

*In this case the absolute condition number is* $\lim_{\varepsilon\to 0}\max_{\widetilde{x}\in B(x,\varepsilon)}\frac{\|f(\widetilde{x})-f(x)\|}{\|\widetilde{x}-x\|}$*, and one can see that the output/input ratio varies with the direction in which* $\tilde{x}$ *is, so we have to take a maximum in the whole ball* $B(x,\varepsilon)$*.*

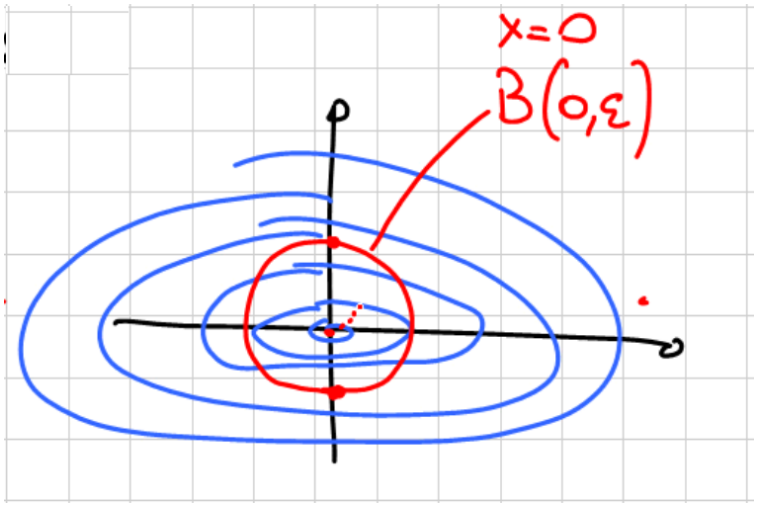



FIGURE 9.5: Level curves of a quadratic function ("seen from above").

At this point, an observation is mandatory: **any** absolute measure doesn't take into account the values of the function in other points, so we want to define the following

**Definition 9.3** (Relative error)**.** *The* ***relative error*** *of an approximation* $\widetilde{x}$ *to a quantity* $x$ *is* $\frac{\|\widetilde{x}-x\|}{\|x\|}$*.*

Here are some examples of good and bad accuracy:

- $\frac{|\tilde{x}-x|}{|x|}\approx 1$: **very bad** accuracy; it's just a number with the same order of magnitude.
- $\frac{|\tilde{x}-x|}{|x|}\approx 10^{-3}$: about 3 correct significant digits.
- $\frac{|\tilde{x}-x|}{|x|}\approx 10^{-16}$: about 16 correct digits; we **can't do better** typically (with `double` precision arithmetic).

**Definition 9.4** (Relative condition number)**.** *The* ***relative condition number*** *of a function* $f$ *is defined as*

$$\kappa_{rel}(f,\mathbf{x})=\lim_{\delta\to 0}\sup_{\frac{\|\tilde{\mathbf{x}}-\mathbf{x}\|}{\|\mathbf{x}\|}\le\delta}\frac{\frac{\|f(\tilde{\mathbf{x}})-f(\mathbf{x})\|}{\|f(\mathbf{x})\|}}{\frac{\|\tilde{\mathbf{x}}-\mathbf{x}\|}{\|\mathbf{x}\|}}=\kappa_{abs}(f,\mathbf{x})\frac{\|\mathbf{x}\|}{\|f(\mathbf{x})\|},$$

*i.e., we replace the absolute error* $\|\tilde{\mathbf{x}} - \mathbf{x}\|$ *with the relative error.*

### 9.1.1 Conditioning of linear systems

At this point we would like to compute the condition number of solving a linear system, i.e., the condition number of the function $f(A, b) = A^{-1}b$, perturbing the inputs $A$ and $b$, one at a time.

PERTURBING $b$ We want to compute the limit of the relative error $\frac{\left\|f(A,\widetilde{b})-f(A,b)\right\|}{\|f(A,b)\|}$, so we set $x = A^{-1}b$ and $\tilde{x} = A^{-1}\tilde{b}$, and we estimate *output error* $= \frac{\left\|\widetilde{x}-x\right\|}{\|x\|} =?$

1.
$$\begin{aligned} \|\widetilde{x} - x\| &= \left\|A^{-1}\widetilde{b} - A^{-1}b\right\| \\ &= \left\|A^{-1}(\widetilde{b} - b)\right\| \\ &\leq \left\|A^{-1}\right\| \left\|\widetilde{b} - b\right\| \end{aligned} \tag{9.1}$$

2. Since $\|b\| = \|Ax\| \leq \|A\| \|x\|$ we have $\frac{\left\|\widetilde{x}-x\right\|}{\|x\|} \leq \|A^{-1}\| \|A\| \frac{\left\|\widetilde{b}-b\right\|}{\|b\|}$

In the end, since *input error* $= \frac{\left\|\widetilde{b}-b\right\|}{\|b\|}$ we obtain

$$\kappa_{rel}(f, x) = \lim_{\varepsilon \to 0} \frac{\textit{output error}}{\textit{input error}} \leq \lim_{\varepsilon \to 0} \left\|A^{-1}\right\| \|A\| = \left\|A^{-1}\right\| \|A\|$$

We denote $\kappa(A) = \|A^{-1}\| \|A\|$ the **condition number of** $A$;

PERTURBING $A$ Given $Ax = b$ we obtain $(A + \Delta_A)(x + \Delta_x) = b$, where $\widetilde{A} = A + \Delta_A$ and $\widetilde{x} = x + \Delta_x$. Then we can expand as follows

$\cancel{Ax} + \Delta_A x + A\Delta_x + \Delta_A\Delta_x = \cancel{b}$

We can stop taking into account $\Delta_A\Delta_x$, since it's a sort of second order term ($\Delta_A\Delta_x = o(\|\Delta_A\| \|\Delta_x\|)$), so we get the following

$$\Delta_A x + A\Delta_x = 0$$

$$\Delta_x = -A^{-1}\Delta_A x$$

then $\|\Delta_x\| \leq \|A^{-1}\| \|\Delta_A\| \|x\|$, which implies $\frac{\|\Delta_x\|}{\|x\|} \leq \|A^{-1}\| \frac{\|\Delta_A\|}{\|A\|}$.

We obtain that *relative output error* $\leq \kappa(A) \cdot$ *relative input error*.

We only proved an inequality, but it turns out that it is tight: for every $A$ and $b$ there is a possible choice of the perturbation $\tilde{x}$ that attains equality.

In the end, in both cases, the error in the output is the error in the input (namely $b$ or $A$) times the condition number.

## 9.2 Condition number, SVD, and distance to singularity

**Fact 9.1.** *$\kappa(A) = \frac{\sigma_1}{\sigma_n}$, i.e., $\kappa(A)$ is the ratio between the smallest and the largest singular value.*

So we can say that if a matrix is close to a singular matrix, then its condition number is going to be large.

*Proof.* Let $A = USV^T$, then $\|A\| = \left\|USV^T\right\| = \|S\| = \sigma_1$, since

$$S = \begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & & & & & \sigma_n \end{pmatrix} \text{ and } \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n.$$

It's also true that $\|A^{-1}\| = \left\|(USV)^{-1}\right\| = \left\|VS^{-1}U^T\right\| = \|S^{-1}\| = \sigma_n$, since

$$S = \begin{pmatrix} \frac{1}{\sigma_1} & & & & \\ & \frac{1}{\sigma_2} & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & & & & & \frac{1}{\sigma_n} \end{pmatrix} \text{ and } \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n.$$

In the end $\kappa(A) = \frac{\|A\|}{\|A^{-1}\|} = \frac{\sigma_1}{\sigma_n}$ □

**Fact 9.2.** *The relative distance between $A$ and the closest singular matrix is $\frac{1}{\kappa(A)}$.*

**Do you recall?**

**Eckart-Young theorem:** the closest matrix to $A$ that has rank $\leq n-1$ is

$$\hat{A} = U \begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \sigma_n \\ & & & & & 0 \end{pmatrix} V^T$$

*Proof.*

*bau* (9.2)

$$
\begin{aligned}
\left\|A-\hat{A}\right\| &= \left\| U \left( \begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & & & & & \sigma_n \end{pmatrix} - \begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \sigma_{n-1} \\ & & & & & 0 \end{pmatrix} \right) V^T \right\| \\
&= \left\| U \begin{pmatrix} 0 & & & & \\ & \cdot & & & \\ & & \cdot & & \\ & & & 0 & \\ & & & & \sigma_n \end{pmatrix} V^T \right\| \\
&= \sigma_n
\end{aligned}
\tag{9.3}
$$

Thus, $\left\|A-\hat{A}\right\| = \sigma_n$. We know already that $\|A\| = \sigma_1$, so we just need to take the ratio. □

We analyzed the conditioning of linear systems, but the main problem we want to study in this course is least squares problem.

## 9.3 Conditioning of least squares problem

We need two quantities to be able to measure the conditioning of least squares problem:

✯ $\kappa(A)$ Let $A \in \mathcal{M}(m,n,\mathbb{R})$, with $m > n$ (tall, thin $A$). We *define* $\kappa(A) = \frac{\sigma_1}{\sigma_n}$. Note that we cannot use the other definition $\kappa(A) = \|A\|\,\|A^{-1}\|$, since $A^{-1}$ does not exist for a non-square $A$. However, one can verify that $\|A\|\,\|A^{+}\| = \frac{\sigma_1}{\sigma_n} = \kappa(A)$, where $A^+$ is the pseudoinverse.

**Observation 9.1.** *Note that $\frac{1}{\kappa(A)}$ is the relative distance to the closest $\hat{A}$ without full column rank.*

✯ $\theta$ The second quantity needed is the angle between $Ax$ and $b$, see Figure 9.6. $\theta = \arccos \frac{\|Ax\|}{\|b\|}$

Now we can express the theorem:

**Theorem 9.3** (Trefethen, Bau)**.** *Consider the linear least squares problem* $\min \|Ax-b\|$*, with $A \in \mathbb{R}^{m\times n}$ with full column rank. Its relative condition number with respect to the input $b$ is*

$$\kappa_{rel,b\to x} \leq \frac{\kappa(A)}{\cos\theta}$$

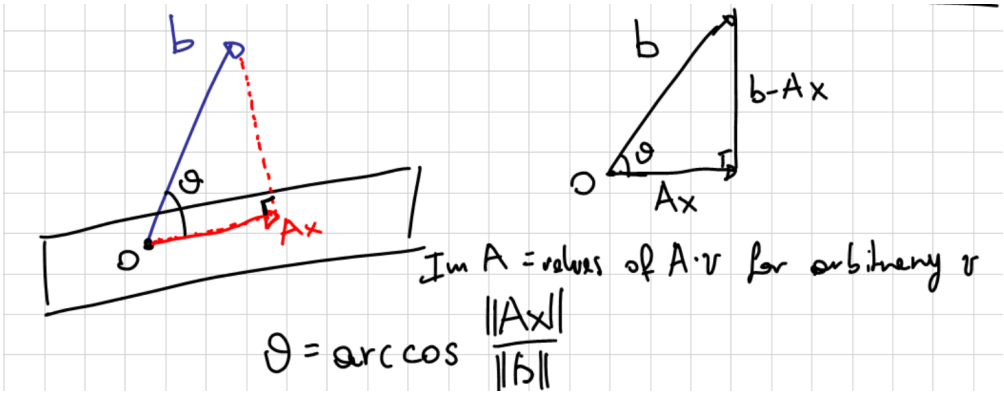



FIGURE 9.6: The triangle in the figure (the one whose cathets are $Ax$ and $b - Ax$) is a square triangle.

*and with respect to $A$ it is*

$$\kappa_{rel,A\to x} \leq \kappa(A) + \kappa(A)^2 \tan\theta$$

*where $\theta$ is the angle such that $\cos\theta = \frac{\|Ax\|}{\|b\|}$.*

At this point we have two condition numbers and they both depend on $\kappa(A)$ and $\theta$.

**Observation 9.2.**

SPECIAL CASE 1: $\theta \approx 90°$ *We can see from the figure that a big change of $b$ induces a small perturbation of $Ax$. No matter what the conditioning of $A$ is, a small (relative) perturbation in $b$ can change a large (relative) perturbation in $x$ and $Ax$, see Figure 9.7.*

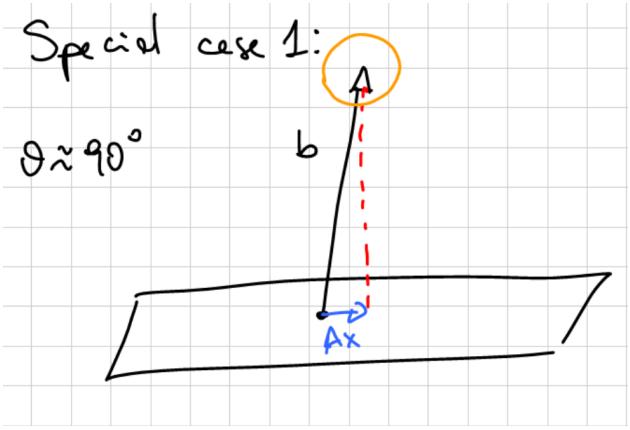



FIGURE 9.7: Special case 1

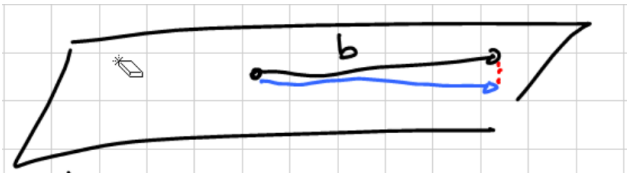



FIGURE 9.8: Special case 2.

SPECIAL CASE 2: $\theta \approx 0°$ *When b is almost in plane with $Im(x)$. In this case cond $\approx \kappa(A)$, see Figure 9.8.*

GENERAL CASE: $\theta$ FAR FROM $0°$ AND $90°$ *In the more general case, cond $\approx \kappa(A)^2$.*

# 10 15th of November 2018 — F. Poloni

## 10.1 Stability of algorithms

In this lecture we will try to answer the questio: "Is our algorithm (using floating point) going to compute a good approximation of the answer?"

It is related to sensitivity / conditioning but different. Depends on how we perform the computation.

**Do you recall?**

Computers work with IEEE arithmetic and the basic idea is the following.
TL;DR: floating point numbers are numbers in base-2 scientific (exponential) notation.
`double` (64-bit numbers):

$$\pm 1.\underbrace{01001011101\ldots101}_{\text{52 binary digits}} \cdot 2^{\pm \underbrace{101\ldots01}_{\text{10 binary digits}}}.$$

We use 1 bit for the sign, 52 bits for the "mantissa" and 11 bits for the exponent and its sign.
Some of these combinations of bits are reserved for special numbers, e.g. `Inf` and `NaN`, `-0`.

This system is subject to approximation errors, exactly like the "usual" decimal arithmetic: for example, if we do $\frac{1}{3} = 0.33333\ldots$ and if we do $\frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 0.99999\ldots \neq 1$.

Whenever we store a number on a computer we need to approximate it, using the representable numbers, as shown in Figure 10.1.


FIGURE 10.1: We have $2^{52}$ equispaced numbers between $\frac{1}{2}$ and 1 and between 1 and 2, and also $2^{52}$ between 2 and 4 and so on and so forth, so we have the same number of integers, although the space is enlarging.

Not all numbers are exactly representable, take 0.1 (decimal). It's a periodic number when written in binary, hence we can't represent it exactly as a machine number.

**Definition 10.1** (Error bound)**.** *For each $x \in \pm[10^{-308}, 10^{308}]$, there is an exactly representable number $\tilde{x}$ such that $\frac{|\tilde{x}-x|}{|x|} \leq \mathsf{u}$, with $\mathsf{u} = 2^{-52} \approx 2 \cdot 10^{-16}$.*

Let us assume that we have the best possible algorithm that returns $\tilde{y} = f(\tilde{x})$ which is the best representation of $f(\tilde{x})$, then

$$\begin{aligned}\frac{|\tilde{y}-y|}{|y|} &\leq \kappa_{rel}(f,x)\frac{|\tilde{x}-x|}{|x|} + o(\frac{|\tilde{x}-x|}{|x|}) \\ &\leq \kappa_{rel}(f,x)\mathsf{u} + o(\mathsf{u}).\end{aligned}$$

In practice we may ignore $o(u)$, since it's small o of $2^{-16}$.

When we are approximating a result, we are approximating it up to a relative error of order of magnitude $10^{-16}$.

$$a \oplus b = (a+b)(1+\delta), \quad |\delta| \leq \mathsf{u}$$

$$\Updownarrow$$

$$\frac{|(a \oplus b) - (a+b)|}{|a+b|} = \delta$$

We would like to compute the error on the function $f(a,b) = a^T b$.

$$a = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}, b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \; so \; f(a,b) = a_1b_1 + a_2b_2 + a_3b_3$$

Denoting $\odot$ the approximated product and $\oplus$ the approximated sum on a computer, we can write the following:

$$\begin{aligned}\text{computer result} &= a_1 \odot b_1 \oplus a_2 \odot b_2 \oplus a_3 \odot b_3 \\ &= \Big[\Big[a_1b_1(1+\delta) + a_2b_2(1+\delta_2)\Big](1+\delta_3) + a_3b_3(1+\delta_4)\Big](1+\delta_5) \\ &= a_1b_1(1+\delta_1)(1+\delta_3)(1+\delta_5) + a_2b_2(1+\delta_2)(1+\delta_3)(1+\delta_5) \\ &+ a_3b_3(1+\delta_4)(1+\delta_5) \\ &= a_1b_1(1+\delta_1+\delta_3+\delta_5+O(u^2)) \\ &+ a_2b_2(1+\delta_1+\delta_3+\delta_5+O(u^2)) \\ &+ a_3b_3(1+\delta_4+\delta_5+O(u^2)) \\ &\approx a_1b_1(1+\delta_1+\delta_3+\delta_5) + a_2b_2(1+\delta_1+\delta_3+\delta_5) + a_3b_3(1+\delta_4+\delta_5)\end{aligned} \tag{10.1}$$

Where $O(u^2)$ comes from the summation of $\delta_i\delta_j$, for some $i$, $j$ and allows us to do an approximation up to second order terms of precision.

The absolute error then is:

$$\begin{aligned}\mathrm{Err}_a &= |a_1b_1(\cancel{1} + \delta_1 + \delta_3 + \delta_5) + a_2b_2(\cancel{1} + \delta_1 + \delta_3 + \delta_5) + a_3b_3(\cancel{1} + \delta_4 + \delta_5) + \cancel{a_1b_1} + \cancel{a_2b_2} + \cancel{a_3b_3}| \\ &\overset{(1)}{\leq} |a_1b_1|3u + |a_2b_2|3u + |a_3b_3|2u \\ &\leq (|a_1b_1| + |a_2b_2| + |a_3b_3|)3u\end{aligned} \tag{10.2}$$

Where $\overset{(1)}{\leq}$ follows from the observation that $|\delta_i| \leq u$.

The result that we can obtain is weaker than we would have expected: *if* $a_ib_i \geq 0, \forall i = 1, 2, 3$, then

$$\frac{|\text{computer result} - (a_1b_1 + a_2b_2 + a_3b_3)|}{a_1b_1 + a_2b_2 + a_3b_3} \leq 3u$$

which means that the algorithm is stable.

However, if $a_1b_1$, $a_2b_2$ and $a_3b_3$ have different signs, then we can bound the error not with $a_1b_1 + a_2b_2 + a_3b_3$, but only with $|a_1b_1| + |a_2b_2| + |a_3b_3|$. This might be a lot larger than what we want to compute.

**Example 10.1.** *Take* $\varepsilon = 10^{-16}$. *Compute* $\begin{pmatrix}1 & -1 & 0\end{pmatrix}\begin{pmatrix}1+\varepsilon \\ 1 \\ 1\end{pmatrix} = (1+\varepsilon) - 1 + 0 = \varepsilon$.

*In this case we have a subtraction between two very close numbers.*

$|$*computer result*$| \leq u(i + |1(1+\varepsilon)| + |-1 \cdot 1| + |0 \cdot 1|) = (2+\varepsilon)u$, *which means that we have* 10 *correct digits.*

*The problem is in the fact that we have a very small result* ($\varepsilon = O(10^{-6})$) *and a very small error (of the same order of the result), which implies a large relative error.*

An attentive reader might have noticed that it's very long to make this computation. Therefore, since we computed it on easy examples, we have to observe that we can't afford it for SVD or other complicated methods.

Wilkinson trick (from the 60's) comes to help us, to simplify this computation for more complicated problems.

**Idea:** see the computer result as the exact output of an algorithm run on a slightly perturbed input.

$$\begin{aligned}\tilde{y} &= \dots \\ &= ((a_1b_1(1+\delta_1) + a_2b_2(1+\delta_2))(1+\delta_4) + a_3b_3(1+\delta_3))(1+\delta_5) \\ &= a_1\widetilde{b_1} + a_2\widetilde{b_2} + a_3\widetilde{b_3}\end{aligned} \tag{10.3}$$

where

$$\begin{aligned}\widetilde{b_1} &= b_1(1+\delta_1)(1+\delta_4)(1+\delta_5) = b_1 + 3ub_1 + o(u), \\ \widetilde{b_2} &= b_2(1+\delta_2)(1+\delta_4)(1+\delta_5) = b_2 + 3ub_2 + o(u), \\ \widetilde{b_3} &= b_3(1+\delta_3)(1+\delta_5) = b_3 + 2ub_3 + o(u),\end{aligned} \tag{10.4}$$

$$\widetilde{a_i} = a_i \quad i = 1, 2, 3.$$

And the relative error is $\frac{\|\widetilde{b}-b\|}{\|b\|} \leq 3u + o(u)$.
Hence,

$$\frac{\|\tilde{y}-y\|}{\|y\|} \leq \kappa_{rel,b}\frac{\left\|\widetilde{b}-b\right\|}{\|b\|} \leq k_{rel,b}3u$$

This isn't bad, because it's within a factor 3 of the optimal error for a perfect algorithm. Computing the conditioning is much easier than making all the calculations of $\delta$s.

**Definition 10.2** (Backward stability of an algorithm)**.** *An algorithm to compute $\mathbf{y} = f(\mathbf{x})$ is called **backward stable** if the computed output $\tilde{\mathbf{y}}$ can be written as $\tilde{\mathbf{y}} = f(\tilde{\mathbf{x}})$, where $\tilde{\mathbf{x}} = \mathbf{x} + O(\mathsf{u}\,\|\mathbf{x}\|)$ (exact function, perturbed input).*

**Observation 10.1.** *In real-life usage, this $O()$ notation often hides polynomial factors in the dimension $n$. Although this may look an illicit simplification, we observe that these factors are much more harmless than the error that we could make otherwise.*

**Theorem 10.1.** *Backward stable algorithms are as accurate as theoretically possible (given the condition number of a problem), up to some factor that depends only on the dimension (e.g., $n$, $2n^2 + 18n$, ...).*

*Proof.*

$$\frac{\|\tilde{\mathbf{y}}-\mathbf{y}\|}{\|\mathbf{y}\|} \leq \kappa_{rel}(f,\mathbf{x})\frac{\|\tilde{\mathbf{x}}-\mathbf{x}\|}{\|\mathbf{x}\|} = \kappa_{rel}(f,\mathbf{x})O(\mathsf{u})$$

while the best attainable accuracy is $\kappa_{rel}(f,\mathbf{x})\mathsf{u}$. □

We may ask ourselves if it's possible to perturb the input in order to get $\widetilde{y}$ for every possible algorithm and the answer is no. Let us see a counterexample, where $f(a,b) = a^T b$ (vector-vector product in the order that produces a matrix).

We observe that in general it's not true that the computed approximation of $f(a,b)$ has rank 1.

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} = \begin{bmatrix} x_1y_1 & x_1y_2 & x_1y_3 \\ x_2y_1 & x_2y_2 & x_2y_3 \\ x_3y_1 & x_3y_2 & x_3y_3 \end{bmatrix} = M$$

While

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \odot \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} = \begin{bmatrix} x_1 \odot y_1 & x_1 \odot y_2 & x_1 \odot y_3 \\ x_2 \odot y_1 & x_2 \odot y_2 & x_2 \odot y_3 \\ x_3 \odot y_1 & x_3 \odot y_2 & x_3 \odot y_3 \end{bmatrix} = \widetilde{M}$$

And we are looking for $\widetilde{x}$ and $\widetilde{y}$ such that $\widetilde{M} = \widetilde{x}^T\widetilde{y}$

**Example 10.2.** *Example (with exaggerated errors):*

$$\begin{bmatrix}1\\2\\3\end{bmatrix}\begin{bmatrix}4 & 5 & 6\end{bmatrix} = \begin{bmatrix}4 & 5 & 6\\8 & 10 & 12\\12 & 15 & 18\end{bmatrix}$$

*While*

$$\begin{bmatrix}1\\2\\3\end{bmatrix} \odot \begin{bmatrix}4 & 5 & 6\end{bmatrix} = \begin{bmatrix}4.01 & 4.99 & 6.01\\7.99 & 10.01 & 12.02\\11.98 & 15.02 & 17.97\end{bmatrix}$$

*and this second matrix doesn't have rank* 1.

## 10.2 Backward stability of QR factorization

**Do you recall?**

A generic step of the computation of the QR factorization of a matrix has the following shape:

$$\begin{pmatrix} \mathrm{I} & \\ & \mathrm{H}\ \mathbf{u}_k \end{pmatrix}\begin{pmatrix}*&*&*&*&*\\0&*&*&*&*\\0&0&*&*&*\\0&0&*&*&*\\0&0&*&*&*\end{pmatrix} = \begin{pmatrix}*&*&*&*&*\\0&*&*&*&*\\0&0&*&*&*\\0&0&0&*&*\\0&0&0&*&*\end{pmatrix}$$

Each step of the QR factorization in backward stable.

Let us compute the backward stability of QR factorization:

With some tedious computations like the ones above, one can show that *one* step of the QR factorization is backward stable, i.e., $\widetilde{R}_{k-1} = R_{k-1} + \Delta_{R_{k-1}}$, where $\left\|\Delta_{R_{k-1}}\right\| \leq O(u)\left\|R_{k-1}\right\|$ and this allows us to write the following

$$\frac{\left\|\widetilde{R}_{k-1} - R_{k-1}\right\|}{\|R_{k-1}\|} \leq O(u)$$

At a generic step $k$ we have that:

$$R_k = Q_k R_{k-1}$$

$$\Updownarrow$$

$$\widetilde{R_k} = Q_k(R_{k-1} + \Delta_{R_{k-1}})$$

The idea is doing this procedure one step after the other, starting from step 1, where $R_0 = A$.

$$\widetilde{R_1} = Q_1\widetilde{R_0} = Q_1(R_0 + \Delta_{R_0}) = Q_1(A + \Delta_{R_0})$$

$$\widetilde{R_2} = Q_2\widetilde{R_1} = Q_2(R_1 + \Delta_{R_1}) = Q_2Q_1(A + \Delta_{R_0}) + Q_2\Delta_{R_1} = Q_2Q_1(A + \Delta_{R_0} + {Q_1}^T\Delta_{R_1})$$

We go on combining errors like this and we see that all the quantities $(A + \Delta_{R_j})$ are perturbations of the original matrix $A$.

In the end we may observe that the final computed $R_n$ is the exact result obtained from $A$ plus $n$ perturbations.

**Observation 10.2.** *We should notice that the norm of each perturbation is small with respect to the norm of $A$.*

*Formally,* $\|\Delta_{R_0}\| \leq u\,\|R_0\| = u\,\|A\|$ *and* $\left\|{Q_1}^T\Delta R_1\right\| \overset{(1)}{=} \|\Delta R_1\| \leq u\,\|R_1\| \overset{(2)}{=} u\,\|A\|$.

*Where* $\overset{(1)}{=}$ *and* $\overset{(2)}{=}$ *hold because $Q_i$ are orthogonal.*

**Mantra**

Orthogonal transformations are the key for stability.

### 10.2.1 Stability of algorithms for least-squares problems

Let us see how various algorithms to solve LS problems (implemented in Matlab) behave in relation to backward stability.

**Least squares problem via QR**

STEP 1: Computing a thin QR (`qr(A, 0);`) → backward stable;

STEP 2: (`Q1' * b;`)→ backward stable;

STEP 3: (`R1  c;`)→ backward stable;

Scrivere meglio

**Least squares problem via SVD**

STEP 1: Computing a SVD (`svd(A, 0);`) → backward stable;

STEP 2: (`U' * b;`)→ backward stable;

STEP 3: (`c ./ diag(S);`)→ backward stable;

STEP 4: (`V * d;`)→ backward stable;

**Least squares problem via Normal Equations**

STEP 1: `C = A' * A;`

STEP 2: `d = A' * b;`

STEP 3: `x = C  d;`

We can also prove that some algorithms aren't backward stable, like normal equations. The issue here is that the same input is needed in more than one operation, so it should satisfy more than one equation and this may lead to a solution set which is empty. Also, in the last equation we solve a linear system with matrix $C$, and this gives an error of the order of $\kappa(A)^2$, never of $\kappa(A)$: $\kappa(C) = \kappa(A^T A) = \kappa(A)^2$.

Let us take the example of two lectures ago.

**Example 10.3.** *Let $A \in \mathcal{M}(4, 3, \mathbb{R})$ s.t.*

$$A = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 3 & 1 & 4 \\ 1 & 2 & 3 + 10^{-8} \end{pmatrix}$$

*We may observe that $A$ is at distance $10^{-8}$ from a matrix without full column rank, hence $\kappa \approx 10^8$.*

*What is the condition number of solving this least squares problem? It's about $10^8$, since in that problem we generated $b = A \begin{pmatrix} 3 & 4 & 5 \end{pmatrix}$, so $b$ lies in the image of $A$ ($Im(A)$). The geometric idea in Figure 10.2.*

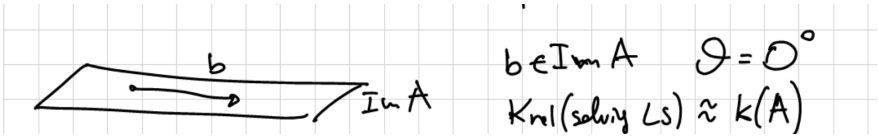



FIGURE 10.2: $b$ lies (at least almost, because of numerical error) on the plane of $Im(A)$. In this case $\kappa_{rel}$(solving LS) $\approx \kappa(A)$.

A brief recap may be found in Table 1.

### 10.2.2 A posteriori checks

Let's assume that Matlab gives us a solution of a problem. We want to be able to check how good this result is.

**Example 10.4.** *Suppose we have solved a linear system...*

```
» A = randn(4, 4); b = randn(4, 1);
» x = A \ b;
» A * x - b
ans =
```

| | **Normal eqns** | **QR** | **SVD** |
|---|---|---|---|
| $m \approx n$ | $\frac{4}{3}n^3$ | $\frac{4}{3}n^3$ | $\approx 13n^3$ |
| $m \gg n$ | $mn^2$ | $2mn^2$ | $2mn^2$ |
| | Unstable when *cond* $\approx \kappa(A)$ | Backward stable | Backward stable; reveals info on sensitivity, allows regularization |

TABLE 1: Brief recap of the complexities of the algorithms we studied for solving the least squares problem. The last row takes into account the stability.

```
           0
-1.3878e-17
           0
 2.2204e-16
```

**Definition 10.3** (Residual)**.** *Let $A \in \mathcal{M}(m, \mathbb{R})$ and $\in \mathbb{R}^m$, and $x$ be the solution of $Ax = b$. For a given $\widetilde{x}$ we define **residual** the following $\mathbf{r} = A\widetilde{x} - b$.*

Assume that $\|r\|$ is small; does this mean that $\widetilde{x}$ is close to the exact solution $x$?

**Theorem 10.2.** *Let $A \in \mathbb{R}^{m\times m}, \in \mathbb{R}^m$, and $x$ be the solution of $Ax = b$.*

*For a given $\widetilde{x}$, the relative error of $\widetilde{x}$ is bounded by the condition of matrix $A$ times the ratio between the norm of the residual and the norm of $b$.*

$$\frac{\|x - \widetilde{x}\|}{\|x\|} \leq \kappa(A)\frac{\|\mathbf{r}\|}{\|b\|}.$$

This theorem tells us that $\widetilde{x}$ is "close to the solution" apart from a factor which is the conditioning of matrix $A$.

*Proof.* Follows from the perturbation results for linear systems. The idea is that $\widetilde{x}$ is the exact solution of the perturbed system

$$A\widetilde{x} \overset{*}{=} b + \mathbf{r} = \widetilde{b}$$

Where $\overset{*}{=}$ follows from the definition of $r$ and $\frac{\|\widetilde{b}-b\|}{\|b\|} = \frac{\|r\|}{\|b\|}$.
A relative perturbation of size $\frac{\mathbf{r}}{b}$ is amplified by $\kappa(A)$. □

It's important to notice that also computing $A \odot x \ominus b$ is an approximated operation. We choose to simplify things and ignore this error.

## 10.3 A posteriori check for Least Squares Problems

Can we make an analogous check for the Least Squares Problem?

Take a LSP $\min \|Ax - b\|$, with $A$ tall thin, with full column rank.

The problem is that $\|Ax - b\|$ isn't small at all, indeed it could be as large as $b$, as you can see from Figure 10.3.

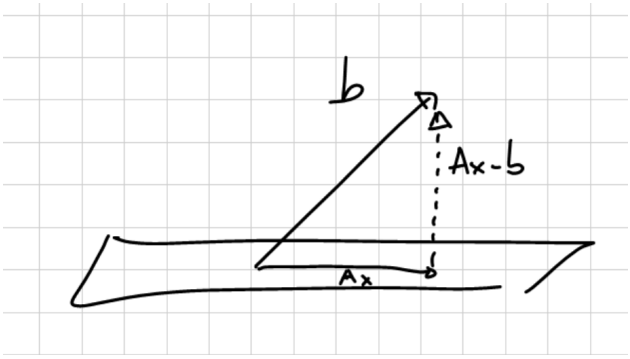



FIGURE 10.3: $A$ can be as large as $b$ if $b$ is perfectly orthogonal.

**Observation 10.3.** *If you solve LSP via QR,* $\|Ax - b\| = \left\| \begin{pmatrix} R_1 x - {Q_1}^T b \\ -{Q_2}^T b \end{pmatrix} \right\|$. *We said that the entries in the second block are fixed irrespective of* $x$*, but we could make the entries in the first block zero, by choosing* $x = R_1^{-1} Q^T b$*. This information let us infer something about the values of the vectors in Figure 10.3, in particular the minimum of the value that we can get is* $\left\| {Q_2}^T b \right\| = \|Ax - b\|$.

*With some algebra we may also check that* $\left\| {Q_1}^T b \right\| = \|Ax\|$.

Since this is a minimum problem, we know that the gradient of the function is small near the optimum value: $\min \|Ax - b\|^2 = \min x^T A^T Ax - 2b^T Ax + b^T b$.

$\nabla_{\widetilde{x}} f = 2(A^T Ax - A^T b) \to 0$

**Theorem 10.3.** $\frac{\|\widetilde{x} - x\|}{x} \leq \kappa(A)^2 \frac{\|A^T Ax - A^T b\|}{\|A^T b\|}$. *Although we might have wanted to have the condition number of the problem, instead of the condition number of* $A$ *and this could lead to underestimating the error.*

Another idea could be using as error the first entry of the vector obtained via QR (namely ${R_1}^T x - {Q_1}^T b$), by imposing $R_1 x = {Q_1}^T b$

We may observe that this is a truly backward stable measure:

given $r = \left\| {R_1}^T \widetilde{x} - {Q_1}^T b \right\|$, there exists $\widetilde{b}$ with $\left\| \widetilde{b} - b \right\| = \|r\|$ such that $\widetilde{x}$ is the exact solution of $\min \left\| Ax - \widetilde{b} \right\|$.

We have proved the following

**Fact 10.4.** $\frac{\|\widetilde{x} - x\|}{\|x\|} \leq \kappa_{rel,LS} \frac{\|\widetilde{b} - b\|}{\|b\|} = \kappa_{rel,LS} \frac{\|r\|}{\|b\|}$.

**Theorem 10.5.** *Let $A = Q_1 R_1$ be a thin QR factorization. Let $\mathbf{r}_1 = Q_1^T(A\widetilde{\mathbf{x}} - \mathbf{b})$. Then, $\widetilde{\mathbf{x}}$ is the exact solution of the LS problem:*

$$\min \|A\mathbf{x} - (\mathbf{b} + Q_1\mathbf{r}_1)\|$$

*so the backward error of $\widetilde{\mathbf{x}}$ is $\|Q_1\mathbf{r}_1\| = \|\mathbf{r}_1\|$.*

*Proof.* Idea: replay the solution of a LS problem with QR factorization, and use ${Q_1}^T T Q_1 = I$. You will get in the first block $R_1 x = {Q_1}^T b + \mathbf{r}_1$, i.e., ${Q_1}^T(Ax - b) = \mathbf{r}_1$, which is verified by $\widetilde{x}$. □

# 11 21st of November 2018 — F. Poloni

In this lecture we address the problem of solving linear systems exactly.

Someone could observe that this subject has already been studied in the numerical linear algebra course, but we are interested in computing the solution to this problem quickly when the dimensions are large and the matrix $A$ is sparse.

Since the complexity of Gauss method is cubic, this algorithm is unfeasible for large inputs.

Let us see some real life examples, where the matrices are large and sparse.

Local function on graphs: A **local function on graphs** is a function that depend on few nearby vertices. This kind of functions lead to a sparse adjacency matrix $A$, as can be observed in Figure 11.1;

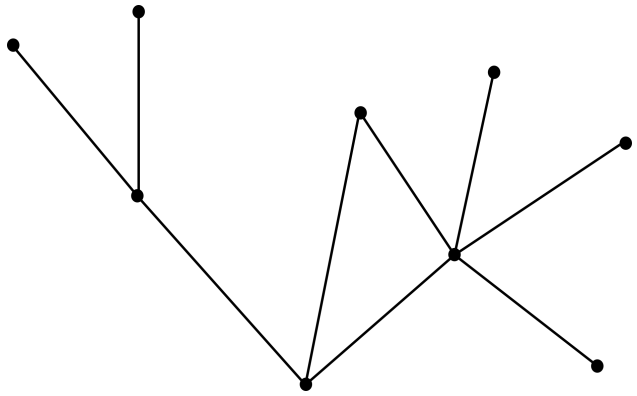


Figure 11.1: A local function on graph.

Images: Take an $m \times m$ image and blur it (each pixel is obtained as the average of its neighbours). $T : \mathcal{M}(m, \mathbb{R}) \to \mathcal{M}(m, \mathbb{R})$ such that $T(A)_{ij} = \frac{1}{9}(A_{i-1j} + A_{i-1j-1} + A_{i-1j+1} + A_{ij} + A_{ij-1} + A_{ij+1} + A_{i+1j} + A_{i+1j-1} + A_{i+1j+1})$. $T$ may be written as a matrix that maps all the $m$ images to a set of $m$ blurred images and has the following shape $T \in \mathcal{M}(m^2, \mathbb{R})$ such that the $(i, j)$-th row of $T$ has exactly 9 entries with value $\frac{1}{9}$ and all the others are 0. The non zero entries correspond to $A_{i-1j}, \ldots, A_{i+1j+1}$;

KKT systems: constrained optimization;

Engineering problem: To check stability of a bridge, it gets split into small cells. It can be proven that the stress on each of these cells corresponds to the force applied by the neighbours. In the end, this local phenomenon may be represented by a sparse matrix.

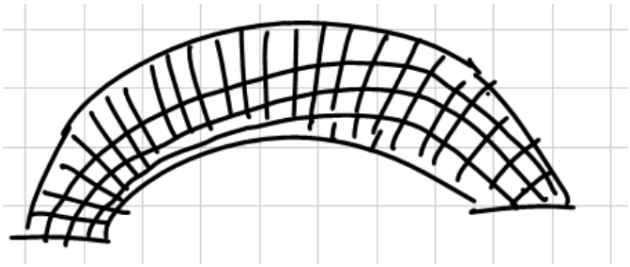


FIGURE 11.2: Graphic idea of a bridge partitioned into small blocks

## 11.1 Gaussian elimination and LU factorization

Gaussian elimination can be seen as a factorization: $A = LU$. The intuition is to proceed iteratively, multiplying each time for a new matrix, just like QR factorization.

Since the idea of Gauss elimination is to add multiples of row 1 to all the rows from 2 to $n$ to kill off $A_{2:end,1}$ we have that:

STEP 1:

$$\begin{pmatrix} 1 & & & & \\ * & 1 & & & \\ * & & 1 & & \\ * & & & 1 & \\ * & & & & 1 \end{pmatrix} \begin{pmatrix} \circledast & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{pmatrix} = \begin{pmatrix} \circledast & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix},$$

Where the $\circledast$ is called **pivot**.

$$L_1 A = A_1$$
$$(L_1)_{k1} = -\frac{A_{k1}}{A_{11}}, \quad k = 2, 3, \ldots, m$$

STEP 2: we multiply for a matrix that has an "identity frame" and inside does the same $L_1$ was doing before.

$$\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & * & 1 & & \\ & * & & 1 & \\ & * & & & 1 \end{pmatrix} \begin{pmatrix} * & \circledast & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix} = \begin{pmatrix} * & \circledast & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \end{pmatrix}$$

$$L_2 A_1 = A_2$$
$$(L_2)_{k2} = \frac{(A_1)_{k2}}{(A_1)_{22}}, \quad k = 3, \ldots, m$$

STEP 3: we go on and

$$\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & * & 1 & \\ & & * & & 1 \end{pmatrix} \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & \circledast & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \end{pmatrix} = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & \circledast & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \end{pmatrix}$$

$$L_3 A_2 = A_3$$

$$(L_3)_{k3} = \frac{(A_2)_{k3}}{(A_2)_{33}}, \quad k = 4, \dots, m$$

STEP 4: one more operation

$$\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & * & 1 \end{pmatrix} \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & \circledast & * \\ 0 & 0 & 0 & * & * \end{pmatrix} = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & \circledast & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix}$$

$$L_4 A_3 = A_4$$

$$(L_4)_{k4} = \frac{(A_3)_{k4}}{(A_3)_{44}}, \quad k = 5, \dots, m$$

In the generic case we have $L_{m-1} L_{m-2} \dots L_1 A = U$, where $U$ is upper triangular, or $A = \underbrace{L_1^{-1} L_2^{-1} \dots L_{m-1}^{-1}}_{=L} U$, with $U$ upper triangular and $L$ lower triangular.

**Theorem 11.1.** *Let $A \in \mathcal{M}(m, \mathbb{R})$ such that we do not encounter zero pivots in the algorithm. $A$ admits a factorization $A = LU$, where $L$ is lower triangular with ones on its diagonal, and $U$ is upper triangular.*

**Observation 11.1** (Stroke of luck)**.** *The product of the ${L_i}^{-1}$'s (denoted $L$) can be computed for free, since the following holds:*

$$\begin{bmatrix} 1 & & & & \\ -a_2 & 1 & & & \\ -a_3 & & 1 & & \\ -a_4 & & & 1 & \\ -a_5 & & & & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & -b_3 & 1 & & \\ & -b_4 & & 1 & \\ & -b_5 & & & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & -c_4 & 1 & \\ & & -c_5 & & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & -d_5 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & & & & \\ a_2 & 1 & & & \\ a_3 & b_3 & 1 & & \\ a_4 & b_4 & c_4 & 1 & \\ a_5 & b_5 & c_5 & d_5 & 1 \end{bmatrix}$$

ALGORITHM 11.1 LU factorization, Matlab implementation.

```
1  function [L, U] = lu_factorization(A)
2    m = size(A, 1);
3    L = eye(m);
4    U = A;
5    for k = 1 : m - 1
6      % compute "multipliers"
7      L(k+1:end, k) = U(k+1:end, k) / U(k, k);
8      % update U
9      U(k+1:end, k) = 0;
10     U(k+1:end, k+1:end) = U(k+1:end, k+1:end) ...
11         - L(k+1:end, k) * U(k, k+1:end);
12   end
```

The idea behind the implementation of Algorithm 11.1 is shown in Figure 11.3.

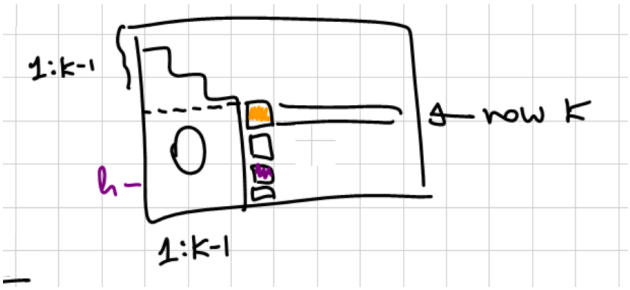



FIGURE 11.3: Assuming that we are at step $k$, we have that the $h$-th multiplier is expressed as $\frac{A_{hk}}{A_{kk}}$ and this muliplier goes to the right position in $L$.

**Observation 11.2.** *The computational complexity of this algorithm is concentrated at lines* 10, 11 *and the cost of this operation is* $O((m-1)^2 + (m-2)^2 + \cdots + 2^2 + 1)$. *So we have that for a dense matrix the computational complexity is* $\frac{2}{3}m^3 + O(m^2)$, *in other words half as much as QR factorization.*

**Something on Matlab ...**

**Implementation of \ in Matlab:** We consider important to remark how the operator \ is implemented in Matlab. It works using the LU factorization, making some checks and changing the algorithm as follows:

- upper/lower triangular systems: back-substitution ($O(n^2)$);
- non-triangular linear systems: LU with partial pivoting (then throw away the factors);
- symmetric and/or sparse systems: uses appropriate LU variants (will see in the following).
- non-square matrices: solves the system "in the least squares sense $\min_x \|Ax-b\|_2$".

Why isn't there any check on the orthogonality of the matrix? Because in that case $A^{-1} = A^T$, and hence it is easy to solve the system. Well, in order to find out that a matrix is orthogonal we need to compute $A^T A$, which is too costly.
**Obs:** LU without pivoting isn't stable, as shown in Section 11.1.1, so Matlab uses pivoting.

missing

### 11.1.1 Stability of LU

A downside of this approach is that it's not numerically stable. The intuition is that the condition is bad whenever the matrix $A$ has a very small pivot.

Let us see an example:

**Example 11.1.**

$$A = \begin{bmatrix} 10^{-30} & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 10^{30} & 1 \end{bmatrix} \begin{bmatrix} 10^{-30} & 1 \\ 0 & 1-10^{30} \end{bmatrix}$$

*In this case the $LU$ factorization produces $L, U$ with norm much larger than $\|A\|$.*

Luckily, it's easy to circumvent this issue multiplying $L_i$s by some permutation matrices (which swap rows in order to keep "large" pivots), as follows

$$L_{m-1}P_{m-1}\dots L_2P_2L_1P_1A = U$$

**Observation 11.3.** *Thanks to another "stroke of luck" we can reorder those factors:*

$$L_{m-1}P_{m-1}\dots L_2P_2L_1P_1 = \widehat{L}_{m-1}\widehat{L}_{m-2}\dots\widehat{L}_1P_{m-1}P_{m-2}\dots P_1$$

*where $\widehat{L}_i$ have the same structure as the $L_i$.*

We can now introduce the following

**Theorem 11.2.** *Let $A \in \mathcal{M}(m, \mathbb{R})$. $A$ admits a factorization $A = PLU$, where $P$ is a permutation matrix, $L$ is lower triangular with ones on its diagonal, and $U$ is upper triangular.*

What about the stability of this improvement to LU factorization, called **LU with partial pivoting**?

It's not stable at all, in the worst case it may happen that $\|U\| / \|A\|$ may grow as $\approx 2^m$, although matrices for which this happens are very rare.

### 11.1.2 Gaussian elimination on sparse matrices

Given a sparse matrix

$$A = \begin{pmatrix}
* & * & & & * & * & & * & \\
 & * & & * & & * & * & & \\
* & & * & * & * & & * & & \\
* & & & * & & & * & & \\
 & * & & & & * & & * & * \\
* & & & * & & * & * & & \\
* & * & & & & & * & & * \\
* & * & * & * & & & & & \\
* & & * & & & * & & & *
\end{pmatrix}$$

Gaussian elimination causes some fill-in, due to the sum of a multiple of the first row, which has non zero entries in different positions:

$$\begin{pmatrix}
* & * & & & * & * & & * & \\
 & * & & * & & * & * & & \\
* & & * & * & * & & * & & \\
* & & & * & & & * & & \\
 & * & & & & * & & * & * \\
* & & & * & & * & * & & \\
* & * & & & & & * & & * \\
* & * & * & * & & & & & \\
* & & * & & & * & & & *
\end{pmatrix} \rightarrow \begin{pmatrix}
* & * & & & * & * & & * & \\
 & * & & * & & * & * & & \\
0 & * & * & * & * & * & * & * & \\
0 & * & & * & * & * & * & * & \\
 & * & & & & * & & * & * \\
0 & * & & * & * & * & * & * & \\
0 & * & & & * & * & * & * & * \\
0 & * & * & * & * & * & & * & \\
0 & * & * & & * & * & & * & *
\end{pmatrix}$$

We may observe that the computational complexity of sparse LU is linear in the number of non zero entries of the final matrix, obtained by the algorithm, which is possibly much larger than the number of non zeros in $A$.

How to circumvent this problem? At each step we may use as pivot row the most sparse one. This computation may be done in a more sofisticate way, considering the "relative" position of non zeros between couples of rows.

Because of this trade-off the choice is made in relation to the needs of the implementation. We won't study any algorithm that deals with sparse matrices, since they are very complicated and make use of heuristics.

There are some lucky cases in which the fill-in is almost none, for example a matrix that only has 5 diagonals which entries are different from 0 (called **tridiagonal**). In this particular case $L$ is tridiagonal and lower triangular and $U$ is tridiagonal and upper triangular, as shown below.

$$A = \begin{pmatrix} * & * & * & 0 & 0 & 0 & \cdots & 0 \\ * & * & * & * & 0 & 0 & \cdots & 0 \\ * & * & * & * & * & 0 & \cdots & 0 \\ 0 & * & * & * & * & * & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & * & * & * & * \\ 0 & 0 & 0 & \cdots & * & * & * & * \\ 0 & 0 & 0 & \cdots & 0 & * & * & * \end{pmatrix}$$

$$L = \begin{pmatrix} * & 0 & 0 & 0 & 0 & \cdots & 0 \\ * & * & 0 & 0 & 0 & \cdots & 0 \\ * & * & * & 0 & 0 & \cdots & 0 \\ & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ & & * & * & * & 0 & 0 \\ & & & * & * & * & 0 \\ & & & & * & * & * \end{pmatrix} \quad U = \begin{pmatrix} * & * & * & & & & & \\ 0 & * & * & * & & & & \\ 0 & 0 & * & * & * & & & \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & & \\ 0 & \cdots & 0 & 0 & * & * & * & \\ 0 & \cdots & 0 & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & * & * \end{pmatrix}$$

**Observation 11.4.** *We should remark that if we are interested in high-performance computing we need to pay attention to the blocking, because we go from vector-vector operation to matrix-matrix operation and some of these operations may be performed more efficiently. Parallel/multithreaded implementations are available by means of parallel libraries for Matlab.*

# 12 23rd of November 2018 — F. Poloni

## 12.1 Gaussian elimination on symmetric matrices

**Do you recall?**

In Gaussian elimination we had $A$ and we multiplied it by $L_1$ in order to get a big chunk of 0s in the first column

$$\begin{pmatrix} 1 & & & & \\ * & 1 & & & \\ * & & 1 & & \\ * & & & 1 & \\ * & & & & 1 \end{pmatrix} \begin{pmatrix} \circledast & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{pmatrix} = \begin{pmatrix} \circledast & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix},$$

Let us consider an upgrade of Gaussian elimination in the case of $A \in S(m, \mathbb{R})$. Let us see what happens if we multiply $L_1 A$ on the right by the transpose of $L_1$:

Step 1:

$$\begin{pmatrix} 1 & & & & \\ * & 1 & & & \\ * & & 1 & & \\ * & & & 1 & \\ * & & & & 1 \end{pmatrix} \begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{pmatrix} \begin{pmatrix} 1 & * & * & * & * \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix} = \begin{pmatrix} * & 0 & 0 & 0 & 0 \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix}$$

Step 2:

$$\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & * & 1 & & \\ & * & & 1 & \\ & * & & & 1 \end{pmatrix} \begin{pmatrix} * & 0 & 0 & 0 & 0 \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix} \begin{pmatrix} 1 & & & & \\ & 1 & * & * & * \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix} = \begin{pmatrix} * & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & 0 \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \end{pmatrix}$$

Step $m$:

$$L_{m-1} L_{m-2} \dots L_1 A L1^T \dots L_{m-2}^T L_{m-1}^T = D,$$

where $D$ is diagonal, or

$$A = L_1 L_2 \dots L_{m-1} D L_{m-1}^T \dots L_2^T L_1^T = L D L^T.$$

**Observation 12.1** (Stroke of luck)**.** *Notice that the stroke of luck of Observation 11.1 holds in this case too, hence we pay nothing to compute matrix $L$.*

$$\begin{bmatrix} 1 & & & \\ -a_2 & 1 & & \\ -a_3 & & 1 & \\ -a_4 & & & 1 \\ -a_5 & & & & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & -b_3 & 1 & & \\ & -b_4 & & 1 & \\ & -b_5 & & & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & -c_4 & 1 & \\ & & -c_5 & & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & -d_5 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & & & & \\ a_2 & 1 & & & \\ a_3 & b_3 & 1 & & \\ a_4 & b_4 & c_4 & 1 & \\ a_5 & b_5 & c_5 & d_5 & 1 \end{bmatrix}$$

**Theorem 12.1** (Symmetric Gaussian elimination)**.** *Let $A \in S(m, \mathbb{R})$ such that during Gaussian elimination we don't encounter any 0 pivot. $A$ admits a factorization $A = LDL^T$, where $L$ is lower triangular with ones on its diagonal, and $D$ is diagonal.*

A Matlab implementation of symmetric Gaussian elimination is shown in Algorithm 12.1.

ALGORITHM 12.1 Symmetric Gaussian factorization, Matlab implementation.

```
function [L, D] = ldl_factorization(A)
  m = size(A, 1);
  L = eye(m); D = zeros(m);
  for k = 1:m-1
    D(k, k) = A(k, k);
          L(k+1:end, k) = A(k+1:end, k) / A(k, k);
          A(k+1:end, k+1:end) = A(k+1:end, k+1:end) ...
        - L(k+1:end, k) * A(k, k+1:end);
  end
  D(m, m) = A(m, m);
```

It is possible to make an optimization of this algorithm: since $A$ is supposed to be symmetric, we only need to update the lower triangular part of $A$, since the rest is mirrored by symmetry, hence the computational complexity is half that of Gaussian elimination.

This algorithm is not backward stable, exactly like the one on non symmetric matrices. Pivoting may be performed in order to improve stability. It comes without saying that the row swap should be done consistently on the columns to preserve symmetry.

Of course there are some matrices (like the ones with all 0s on the diagonal) that cannot be "pivoted". There are workarounds, though. As an example, Matlab's `[L, D, P] = ldl(A)` produces matrices such that $P^TAP = LDL^T$, where $D$ may have $2 \times 2$ diagonal blocks.

**Do you recall?**

We recall the characterization of **positive definite matrix** $A \in M(m, \mathbb{R})$: all its eigenvalues are strictly positive. In other words, $A$ is positive definite if $\forall z \neq 0 \in \mathbb{R}^m$ $z^TAz > 0$.

**Lemma 12.2.** *In the context of positive definite matrices the following holds:*

1. *Let $A$ be a symmetric matrix. $A$ is positive definite if and only if $MAM^T$ is so, for some invertible $M \in M(m, \mathbb{R})$. Formally, $\forall A \in S(m, \mathbb{R})$ s.t. $A \succ 0 \Leftrightarrow \exists M \in GL(m, \mathbb{R})$ s.t. $MAM^T \succ 0$;*

2. *Let $A$ a symmetric positive definite matrix such that $A = \left[\begin{smallmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{smallmatrix}\right]$, then $A_{11}$ and $A_{22}$ are, too. Formally, $\forall A \in S(m, \mathbb{R})$ s.t. $A \succ 0$ and $A = \left[\begin{smallmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{smallmatrix}\right] \Rightarrow A_{11} \succ 0$ and $A_{22} \succ 0$.*

*Proof.*

1.

   $\Rightarrow$) $A \in S(m, \mathbb{R})$ and $A \succ 0 \Longrightarrow MAM^T \in S(m, \mathbb{R})$ and $MAM^T \succ 0$.
   Take $z \in \mathbb{R}^m$, $z \neq 0$ $z^T MAM^T z = y^T A y > 0$, where we performed a variable change $y = M^T z$. Notice that $y \neq 0$ because $M$ is invertible (and $ker(M) = \{0\}$). The symmetry of the matrix $MAM^T$ follows from $(MAM^T)^T = M^{T^T} A^T M^T = MAM^T$;

   $\Leftarrow$) $MAM^T \in S(m, \mathbb{R})$ and $MAM^T \succ 0 \Longrightarrow A \in S(m, \mathbb{R})$ and $A \succ 0$.
   This proof follows from the previous arrow, where the substitution is $z = M^{-1} y$.

2. $A = \left[\begin{smallmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{smallmatrix}\right]$ positive definite $\Longrightarrow$ $A_{11}$ and $A_{22}$ are positive definite too.
   Since $A$ is positive definite, its scalar product is greater than zero with all the vectors in $\mathbb{R}^m$.

   $A_{11}$) Let us take $\mathbf{z} = \left[\begin{smallmatrix} \mathbf{z}_1 \\ 0 \end{smallmatrix}\right]$.
   $\left[\begin{smallmatrix} {\mathbf{z}_1}^T & 0 \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} \mathbf{z}_1 \\ 0 \end{smallmatrix}\right] = {\mathbf{z}_1}^T A_{11} \mathbf{z_1} > 0$, $\forall \mathbf{z}_1 \in \mathbb{R}^{sizeof A_{11}}$

   $A_{22}$) Let us take $\mathbf{z} = \left[\begin{smallmatrix} 0 \\ \mathbf{z}_2 \end{smallmatrix}\right]$.
   $\left[\begin{smallmatrix} 0 & {\mathbf{z}_2}^T \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} 0 \\ \mathbf{z}_2 \end{smallmatrix}\right] = {\mathbf{z}_2}^T A_{22} \mathbf{z_2} > 0$, $\forall \mathbf{z}_2 \in \mathbb{R}^{sizeof A_{22}}$

□

**Corollary 12.3.** *Let $A \in M(n, \mathbb{R})$ such that $A$ is positive definite. When computing the $LDL^T$ factorization of $A$, at each step we have $D_{kk} > 0$, hence we need no pivoting technique.*

*Proof.* From the first point of Lemma 12.2 we have that, since $A \succ 0$, $L_1 A {L_1}^T$ is positive definite. Thanks to the second point of the same lemma we have $L_1 A {L_1}^T = \begin{pmatrix} * & 0 & 0 & 0 & 0 \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix}$ and so the first and the second diagonal blocks are positive definite ($D_{11} > 0$ and $D_{22} \succ 0$).

Notice that, since $A$ is positive definite $A_{11} \succ 0$, but it's a scalar, hence $A_{11} > 0$ and this implies no breakdown case. □

We may introduce another kind of factorization.

## 12.2 Cholesky factorization

The key idea is to write the diagonal matrix of the Gaussian elimination $D$ as product of $D^{1/2}$ times itself:

$$D = \begin{pmatrix} d_{11} & & & \\ & d_{22} & & \\ & & \ddots & \\ & & & d_{mm} \end{pmatrix} = \begin{pmatrix} \sqrt{d_{11}} & & & \\ & \sqrt{d_{22}} & & \\ & & \ddots & \\ & & & \sqrt{d_{mm}} \end{pmatrix} \cdot \begin{pmatrix} \sqrt{d_{11}} & & & \\ & \sqrt{d_{22}} & & \\ & & \ddots & \\ & & & \sqrt{d_{mm}} \end{pmatrix}$$

The LDL factorization may be rewritten as follows

$$A = LDL^T = LD^{1/2}(D^{1/2^T}L^T) = CC^T,$$

where $D^{1/2} = \text{diag}(D_{11}^{1/2}, D_{22}^{1/2}, \ldots, D_{mm}^{1/2})$, and $C$ is lower triangular (but not anymore with ones on the diagonal).

In Matlab the Cholesky factorization of a positive definite matrix is performed by the function `chol(A);` and returns $C^T$.

**Observation 12.2.** *We will not discuss stability further, but Cholesky is always backward stable even without pivoting ($\|C\| = \|A\|^{1/2}$).*

**Observation 12.3.** *In a sparse matrix, we can choose the (symmetric) permutation with the only goal of reducing fill-in. The same considerations about $LU$ factorization hold in this case too.*

## 12.3 Krylov subspace methods

In this part we will discuss some different techniques to solved linear systems, inspired from optimization algorithms.

**Example 12.1.** *Let us consider the optimization problem* $\min \frac{1}{2}x^TAx + b^Tx$, *where* $A \succ 0$. *We know that the solution to this problem is* $x = A^{-1}b$. *Assume we solve this problem via a gradient descent-type method, starting from* $x_0 = 0$.

STEP 1: $x_0 = 0$, $\nabla f(x_0) = -b$;

STEP 2: $x_1 =$*some multiple of* $b \in Span(b)$

$f(x_1) = Ax_1 - b$

$\nabla f(x_1) =$ *some multpiple of* $Ab$ + *some multiple of* $b \in span(b)$;

STEP 3: $x_2 =$ *mult. of* $x_1$ + *mul. of* $\nabla f(x_1)$ + *mult. of* $x_0 + \cdots \in span(Ab_1, b)$;

STEP 4: $x_3 =$ *mult.of* $x_2 + \cdots \nabla f(x_2)$+ *previous iterates* $\cdots$

$Ax_2 - b = A \cdot (sAb + tb) - b = sA^2b + tAb - b \in span(b, Ab, A^2b)$, *where* $s$ *and* $t$ *are scalars;*

STEP 5: $x_4 \in span(b, Ab, A^2b, A^3b)$.

*Notice that we can make some linear combinations of the vectors we have available and* $A(A(sb + tAb) + uAb + vb) + e(sb + tAb) + fAb + gb \in span(b, Ab, A^2b, A^3b)$.

**Idea:** first compute the basis of $span(b, Ab, A^2b, A^3b)$, then look for the best solution inside this subspace.

The following family of algorithms "uses" the matrix $A$ only by computing matrix-vector products.

**Observation 12.4.** *The cost of multiplying a sparse matrix $A$ with a vector $z$ is $O(nnz(A))$, where $nnz(A)$ is the number of non zero entries of $A$.*

*Proof.* Let us assume the matrix $A$ is stored as a vector, whose entries are $(i, j, A_{ij})$.

The matrix-vector product would then be

1. `x=zeros`
2. `for (i, j,` $A_{ij}$ `) such that` $A_{ij} \neq 0$
3. $x_i = x_i + A_{ij} * z_j$
4. `end`

□

From now on we will consider to have a function `compute_product_with_A` that we use to compute matrix-vector products with $A$. In particular, $x =$`compute_product_with_A(z)` computes $x = Az$, given $z$. This function will be the only way in which the matrix $A$ appears in our algorithm. If $A$ is sparse, hence, these algorithms become particularly fast. Moreover, if we somehow have matrices that are not really sparse, but for which there exists a clever implementation of the matrix-vector product, this class of algorithms will give good results.

**Definition 12.1** (Krylof subspace)**.** *Let $A \in M(m, \mathbb{R})$ and let $b \in \mathbb{R}^m$. The **Krilov subspace** of index $n$ is $\mathbf{K_n}(\mathbf{A}, \mathbf{b}) = span(b, Ab, A^2b, \ldots, A^{n-1}b)$.*

*Equivalently, $k \in K_n(A, b) \iff \exists \alpha_1, \ldots, \alpha_{n-1} \in \mathbb{R}^m$ s.t. $v = \alpha_0 b + \alpha_1 Ab + \alpha_2 A^2 b + \cdots + \alpha_{n-1} A^{n-1} b$.*

*Equivalently, $(\alpha_0 + \alpha_1 A + \alpha_2 A^2 + \cdots + \alpha_{n-1} A^{n-1})b = p(A)b$ for a polinomial $p$ of degree such that $deg(p) \leq n - 1$.*

**Observation 12.5** (Properties)**.**

1. *$v, w \in K_n(A, b) \Rightarrow \alpha + \beta W \in K_n(A, b)$;*
2. *$v \in K_n(A, b) \Rightarrow Av \in K_n(A, b)$. Proof. Let us take $v = \alpha_0 b + \cdots + \alpha_{n-1} b$, then $Av = A(\alpha_0 b + \cdots + \alpha_{n-1} b) = \alpha_0 Ab + \cdots + \alpha_{n-1} A^n b$;*

3. $\dim(K_n(A,b)) \leq n$. *It is exactly* $n$ *if* $b, Ab, A^2b, \ldots, A^{n-1}b$ *are linearly independent.*

4. *Let us assume* $dim(K_n(A,b)) \leq n$. *In the second point, if* $A^{n-1}$ *was really necessary* $\alpha_{n-1} \neq 0$ *or* $v \in K_n(A,b)$, $v \notin K_{n-1}(A,b)$, *equivalently then* $A^n b$ *is really necessary to write* $Av$, *i.e.* $Av \in K_{n+1}(A,b)$ *but* $Avnot \in K_n(A,b)$.

5. *We may observe that* $\dim(K_1(A,b)) < \dim(K_2(A,b)) < \cdots < \dim(K_{n_{max}}(A,b)) = \dim(K_{n_{max}+1}(A,b)) = \cdots$.

# 13 29th of November 2018 — F. Poloni

In this lecture we address the problem of designing methods that use Krylov spaces to solve linear systems.

We need alternative methods to Gaussian elimination because matrices are too large.

### 13.0.1 Naive idea

Find first a "good" search subspace, then look for best approximation of the solution of $Ax = b$ in that space.

Let us assume that our space is the image of a matrix $V$ $Im(V)$. $\forall\ x \in Im(V)$ such $x$ may be written as $x = V^1 y_1 + V^2 y_2 + \cdots + V^n y_n$, where $V^i$ are the columns of the matrix $V$.

The idea is too find a vector $y$ that satisfies $\min_{y\in\mathbb{R}} \|AVy - b\|$, which is a least squares problem.

### 13.0.2 Improvement

A good search space is $Im(V) = K_n(A, b) = \text{span}(b, Ab, A^2b, \ldots, A^{n-1}b)$, where $K_n(A, b)$ is the Krylov space.

The issue here is that $V = \begin{pmatrix} b & Ab & \cdots & A^{n-1}b \end{pmatrix}$ is a bad basis, because it is ill conditioned.

Working with that $V$ is problematic: its columns "tend" to be aligned with the dominant eigenvector and this means that $V$ is close to a rank 1 matrix.

We need a better basis for $K_n(A, b)$, in particular an orthogonal basis.

**Do you recall?**

An orthogonal basis is a basis in which each couple of vectors are orthogonal

## 13.1 Arnoldi algorithm

The idea behind this algorithm is to build an orthogonal basis of $K_n(A, b)$ incrementally.

The algorithm at a generic step, takes an orthogonal basis for $K_n(A, b)$ and adds a vector to produce one of $K_{n+1}(A, b)$.

Let us assume that we start with $\{q_1, q_2, \ldots, q_n\}$, orthogonal basis of $K_n(A, b)$.

We also assume that $q_n = \alpha_0 b + \alpha_1 Ab + \cdots + \alpha_{n-1}A^{n-1}b = p(A)b$, where we impose $\alpha_{n-1} \neq 0$.

$p(A) = \alpha_0 I + \alpha_1 A + \cdots + \alpha_{n-1}A^{n-1}$ has degree exactly 1.

Step 1: $K_1(A, b) = span(b)$, $q_1 = \frac{q_1}{\|q_1\|}$

Generic step:

1. produce a vector $K_{n+1}(A,b) - K_n(A,b) : w = Aq_n$
2. $w \in K_{n+1}(A,b) = 0\, w = q_1\beta_1 + q_1\beta_2 + \cdots + q_n\beta_n + q_{n+1}\beta_{n+1}$, where $q_1, q_2, \ldots, q_n, q_{n+1}$ is an orthogonal basis of $K_{n+1}(A,b)$. In this context $q_1, q_2, \ldots, q_n$ are know, while $q_{n+1}$ is still to be determined.
   Notice that $\forall i = 1, \ldots, n$ holds the following ${q_i}^T w = {q_i}^T q_1\beta_1 + \cdots + {q_i}^T q_n\beta_n + {q_i}^T q_{n+1}\beta_{n+1} = {q_i}^T q_i\beta_i = \beta_i$, because of the orthonormality of the basis. Now we can compute $q_{n+1}\beta_{n+1} = w - q_1\beta_1 - q_2\beta_2 + \cdots + q_n\beta_n = z$. If we choose $\beta_{n+1} = \|z\|$ we have that $q_{n+1} = \frac{z}{\|z\|}$ and this produces a valid choice of $q_{n+1}$. We still need to prove that it has norm 1 and it's orthogonal to all the other vectors in the basis.

An implementation of this algorithm is shown in Algorithm 13.1.

ALGORITHM 13.1 Arnoldi algorithm Matlab implementation.

```
function Q = arnoldi(A, b, n)
Q = zeros(length(b), n); %will be filled in
H = zeros(n+1, m);
Q(:, 1) = b / norm(b);
for j = 1 : n
  w = A * Q(:, j);
  for i = 1:j
  % not what we showed earlier here, but stabler
    betai = Q(:, i)' * w;
    w = w - betai * Q(:, i);
    H(i, j) = betai;
  end
  nrm = norm(w);
  H(j+1, j) = nrm;
  Q(:, j+1) = w / nrm;
end
```

Notice that we presented an algorithm where $\beta_i = {q_i}^T w$ for $i = 1, \ldots, n$, then $w \leftarrow w - \beta_1 q_1 - \beta_2 q_2 - \cdots - \beta_n q_n$.

In the implementation we compute $\beta_i = {q_1}^T w$, $w \leftarrow w - \beta_1 q_1$, $\beta_2 = {q_2}^T w$, $w \leftarrow w - \beta_2 q_2$. Why? It is more stable.

At step $j$

$$Aq_j = \beta_{1,j}q_1 + \beta_{2,j}q_2 + \cdots + \beta_{j,j}q_j + \beta_{j+1,j}q_{j+1} = Q \begin{bmatrix} \beta_{1,j} \\ \beta_{2,j} \\ \vdots \\ \beta_{j+1,j} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

$$Q = \begin{bmatrix} q_1 & q_2 & \cdots & q_n & q_{n+1} \\ & & & & \end{bmatrix}$$

$$AQ = A \begin{bmatrix} q_1 & q_2 & \cdots & q_n & q_n \\ & & & & \end{bmatrix} = Q \begin{bmatrix} \beta_{1,1} & \beta_{1,2} & \beta_{1,3} & \cdots & \beta_{1,n} \\ \beta_{2,1} & \beta_{2,2} & \beta_{2,3} & \cdots & \beta_{2,n} \\ 0 & \beta_{3,2} & \beta 3,3 & \cdots & \beta_{3,n} \\ 0 & 0 & \beta_{4,3} & \cdots & \beta_{4,n} \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \beta_{n+1,n} \end{bmatrix}$$

Hence the values $q_i$, $\beta_{i,j}$ computed by Arnoldi satisfy $AQ_n = Q_{n+1}H$, where $H \in M(n+1, n, \mathbb{R})$ and looks like a triangular matrix plus a diagonal below, namely $H = \begin{bmatrix} \beta_{1,1} & \beta_{1,2} & \beta_{1,3} & \cdots & \beta_{1,n} \\ \beta_{2,1} & \beta_{2,2} & \beta_{2,3} & \cdots & \beta_{2,n} \\ 0 & \beta_{3,2} & \beta 3,3 & \cdots & \beta_{3,n} \\ 0 & 0 & \beta_{4,3} & \cdots & \beta_{4,n} \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \beta_{n+1,n} \end{bmatrix}$

Notation: $Q_{n+1} = \begin{bmatrix} q_1 & q_2 & \cdots & q_n & q_{n+1} \\ & & & & \end{bmatrix}$, $Q_n = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \\ & & & \end{bmatrix}$

such that $Q_n \in M(m, n, \mathbb{R})$, while $Q_{n+1} \in M(m, n+1, \mathbb{R})$, $A \in M(m, \mathbb{R})$ and $b \in M(m, n)$.

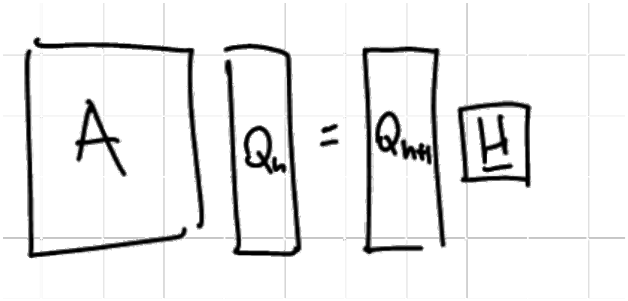



FIGURE 13.1: We started form a matirx $A$ which has many entries and it gets factorized by the product of two smaller matrices

For every matrix $A$ there exist and Arnoldi factorization.

We don't like that we multiply $A$ by two different matrices on the left and on the right, but we may improve the algorithm by using the same matrix.

$AQ_n = (Q_n|q_{n+1}) \cdot matriToDraw = Q_nH + q_{n+1} \cdot (0, \ldots, 0, *) = Q_nH + q_{n+1} \cdot tre_{n+1}\beta_{j+1,j}$, where $e_i = 0 \ldots 010 \ldots 0$.

Last remark: $AQ_n = Q_{n+1}H$ doesn't allow a factorization of the matrix $A$, because $Q_n$ isn't invertible, because $Q_n$ is tall, thin and it doesn't have an inverse.

Once we see how Arnoldi works, we would like to see how to use it.

At some point this assumption must become false. For instance, assume we arrive at step

$m = n$: $q_1, q_2, \ldots, q_m$ are a basis of $\mathbb{R}^m$, so

$$Aq_m = \beta_1 q_1 + \cdots + \beta_m q_m + 0$$

(without an additional term $\beta_{m+1} q_{m+1}$.)

Arnoldi factorization for $m = n$ (assuming nothing broke down before):

$AQ_m = Q_m H$ is a factorization into square matrices, formally $Q_m, H, A, \in M(m, \mathbb{R})$, so we can write something that we couldn't write before:

$A = A_m H {Q_m}^T$ we recall that $H$ is upper triangular plus another diagonal before.

**Definition 13.1.** *Let $H \in M(m, \mathbb{R})$ such that $H_{ij} = 0 \forall i > j + 1$. $H$ is called* ***Hessemberg matrix****.*

**Fact 13.1.** *The QR factorization an Hessemberg matrix $H \in M(m, \mathbb{R})$ can be computed in $O(m^2)$ operations.*

This approach may be used, but in practice, since $A$ is large and sparse we do not want to go until the end.

Let us analyze what happens if at some point $K_{n+1}(A, b) = K_n(A, b)$?

$$Aq_n = \beta_1 q_1 + \cdots + \beta_n q_n + 0$$

for instance, if $b$ is an eigenvector of $A$, it happens already at $n = 1$.

In the implementation Algorithm 13.1 if we have a breakdown at step $n + 1$ this means that $w = A_q n$ has already enough $q_i$s, so $q_{n+1} \beta_{n+1} = 0$.

Problem: $q_{n+1} = \frac{z}{\|z\|}$ division by 0. We need to change the definition $\beta_{n+1} = \|z\| = 0$. At this point we don't get a basis of the Krylov space, but we can still go on as "nothing happened", as long as these vectors are orthonormal. We go on until the end we get

$$AQ_m = H_m Q_m,$$

$$H_m = \left[\begin{array}{cccc|cccc} * & \cdots & * & * & * & \cdots & \cdots & * \\ * & \cdots & * & * & * & \cdots & \cdots & * \\ 0 & \ddots & * & * & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & * & * & * & \cdots & \cdots & * \\ \hline & & & 0 & * & \cdots & * & * \\ & & & & * & \cdots & * & * \\ & & & & 0 & \ddots & * & * \\ & & & & 0 & 0 & * & * \end{array}\right].$$

The blocks are square.

This is good news, because it allows us to make a lot of manipulations.

$$A = Q_m H_m Q_m^T = \begin{bmatrix} Q_n & \widehat{Q} \end{bmatrix} \begin{bmatrix} H_n & L \\ 0 & M \end{bmatrix} \begin{bmatrix} Q_n & \widehat{Q} \end{bmatrix}^T$$

**Do you recall?**

We said two things about block triangular matrices:

1. The eigenvalues of the matrix are the eigenvalues of the diagonal blocks, more formally, $eigh(A) = eigh(H) = eigh(H_n) \cup eigh(M)$
2. We can solve linear systems on block matrices more easily. In this case,

$$\begin{aligned} x = A^{-1}b = Q_m H^{-1} {Q_m}^T b = Q_m H^{-1} \begin{bmatrix} {q_1}^T \\ {q_2}^T \\ \vdots \\ {q_m}^T \end{bmatrix} b \overset{(1)}{=} Q_m H^{-1} \begin{bmatrix} \|b\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ = \begin{bmatrix} Q_n & \widehat{Q} \end{bmatrix} \begin{bmatrix} {H_n}^{-1} & -{H_n}^{-1} L M^{-1} \\ 0 & M^{-1} \end{bmatrix} \begin{bmatrix} \|b\| \, \mathbf{e}_1 \\ 0 \end{bmatrix} \\ = \begin{bmatrix} Q_n & \widehat{Q} \end{bmatrix} \begin{bmatrix} \|b\| \, H_n^{-1} \mathbf{e}_1 \\ 0 \end{bmatrix} = Q_n \, \|b\| \, H_n^{-1} \mathbf{e}_1. \end{aligned} \tag{13.1}$$

   where $\overset{(1)}{=}$ follows from the fact that $b = \|b\| \, q_1$ and $q_1$ is orthogonal to all the other $q_i$.

An attentive reader may notice that at step $j$, when encountering $B_{j+1,j} = 0$, we know the following:

- some eigenvalues of $A$ (those of $H_n$)
- Given an eigencouple $v, \lambda$ such that $H_n v = \lambda v$ then $Q_n v$ is an eigenvector of $A$ with eigenvalue $\lambda$
- the solution $x = Q_n {H_n}^{-1} \|b\| \, e_1$ of $Ax = b$ and this is called lucky breakdown.

The point is that we have what we need to compute the solution at last step before the breakdown.

**Theorem 13.2.** ***'Lucky breakdown'****: if it happens at an early step, we can solve linear systems (or compute some eigenvalues) cheaply: costs $n$ matrix-vector products + $O(mn^2)$.*

What happens when there is no breakdown? After $n$ steps of Arnoldi,

1. if $H_n v = \lambda v$ is an eigenpair of $H_n$. $Q_n v$, $\lambda$ is an approximation of an eigenpair of $A$.
2. $\widetilde{x} = Q_n {H_n}^{-1} \|b\| \, e_1$ is an approximantion of the solution $x$ of $Ax = b$

# 14 5th of December 2018 — F. Poloni

**Do you recall?**

**Arnoldi factorization:** we wanted to build an orthonormal base $\{q_1, q_2, \dots, q_n\}$ of the Krylov space $K_n(A, b) = \{b, Ab, A^2b, \dots, A^{n-1}b\}$. We showed that the matrix $A$ could be more of less factorized as $AQ_n = Q_{n+1}H_n = Q_nH_n + q_{n+1}h_{n+1,n}{e_{n+1}}^T$. Moreover, we concluded that we could approximate the eigenvalues of the matrix $A$ through the eigenvalues of matrix $H_n$, while the eigenvectors are obtained as $Q_nv$, where $v$ is an eigenvector of the matrix $H_n$.

In the first part of the lecture we run some experiments on MatLab and we observed that the eigenvalues are approximated better and better starting from the boundaries. Notice that the approximation is not always good, but it is good enough if we take into account the complexity of the algorithm.

We are interested in explaining the convergence of Arnoldi method.

## 14.1 Convergence of Arnoldi

The eigenvalues of $H_n$ are eigenvalues of a "nearby matrix" obtained by taking $\widetilde{H}_m$ (result of the full process) and replacing $\widetilde{H}_{n+1,n}$ with zero.

$$\widetilde{H}_m = \left[\begin{array}{cccc|cccc} * & \cdots & * & * & * & \cdots & \cdots & * \\ * & \cdots & * & * & * & \cdots & \cdots & * \\ 0 & \ddots & * & * & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & * & * & * & \cdots & \cdots & * \\ \hline & & & \textcircled{0} & * & \cdots & * & * \\ & & & & * & \cdots & * & * \\ & & & & 0 & \ddots & * & * \\ & & & & 0 & 0 & * & * \end{array}\right] \rightarrow H_m = \left[\begin{array}{cccc|cccc} * & \cdots & * & * & * & \cdots & \cdots & * \\ * & \cdots & * & * & * & \cdots & \cdots & * \\ 0 & \ddots & * & * & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & * & * & * & \cdots & \cdots & * \\ \hline & & & \textcircled{*} & * & \cdots & * & * \\ & & & & * & \cdots & * & * \\ & & & & 0 & \ddots & * & * \\ & & & & 0 & 0 & * & * \end{array}\right]$$

We expect that this change does not lead to a significative change in the eigenvalues, in other words, the eigenvalues of $\widetilde{H}_m$ differ from the eigenvalues of $H_m$ by $|h_{n+1,n}|$. Formally, $\left\|\widetilde{H}_m - H_m\right\| = h_{n+1,n}$.

### 14.1.1 Better explanation

The space $K_n(A,b) = \text{span}(b, Ab, \dots, A^{n-1}b)$ contains the right "features" to represent the eigenvectors of $A$ with largest eigenvalues: if $A = V\Lambda V^{-1}$ is diagonalizable, then

$$\begin{aligned} A^k b &= (V\Lambda V^{-1}) \cdots (V\Lambda V^{-1}) b \\ &= V\Lambda^k V^{-1} b \\ &= \begin{pmatrix} V^1 & V^2 & \cdots & V^m \end{pmatrix} \cdot \begin{pmatrix} {\lambda_1}^n & & & \\ & {\lambda_2}^n & & \\ & & \ddots & \\ & & & {\lambda_m}^n \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix} \\ &= V^1 \lambda_1^k c_1 + V^2 \lambda_2^k c_2 + \cdots + V^m \lambda_m^k c_m \end{aligned} \tag{14.1}$$

where $c = V^{-1}b$.

$A^k b$ is a linear combination of the eigenvectors $V^i$ in which those with largest $|\lambda_i|$ are "more prominent", in other words as $n$ increases the components involving the largest $|\lambda_i|$s grow faster.

This also tells us that $\text{span}(V^1, V^2, \dots, V^m)$ (that are the eigenvectors associated to largest eigenvalues in modulus) "represent well" $K_m(A,b) = \text{span}(b, Ab, \dots, A^{m-1}b)$.

We cannot provide a formal proof of the convergence, because there are some counter examples.

**Example 14.1.** *Let $A \in M(m, \mathbb{R})$, such that* $A = \begin{pmatrix} 0 & & & 1 \\ 1 & 0 & & \\ & 1 & \ddots & \\ & & \ddots & \\ & & 1 & 0 \end{pmatrix}$

*In this case the eigenvalues are 0 until the last iteration. In the last step the eigenvalues become correct.*

*Notice that in this case the absolute values of the eigenvalues are the same, hence we are not able to do the trick explained above.*

**Observation 14.1** (Matlab syntax)**.** *The command* `[V, D] = eigs(A)` *does not work on sparse matrices $A$. In this case we may run Arnoldi method and use the best values obtained by Arnoldi:* `[V, D] = eigs(A, n)`*, which computes approximations of the top-n (largest in modulus) eigenvalues.*

*Notice that the Matlab "implementation" (the quotes are because the Arnoldi method de facto is not implmented in Matlab) of the Arnoldi method uses some tricks to converge fast.*

*It is possible to use the command* `eigs` *and pass to it a $\lambda$-function which computes the matrix-vector product and this is useful when the matrix and the vector have a particular shape.*

We would like to understand how to compute some eigenvalues that are not the biggest.

**Lemma 14.1.** *Let $A \in M(m, \mathbb{R})$ and let $(\lambda_1, \mathbf{v}_1), \dots, (\lambda_k, \mathbf{v}_k)$ be the eigenvalues/vectors of $A$. The following holds:*

1. *$(\lambda_i + \alpha, \mathbf{v}_i)$ are eigenvalues/vectors of $A + \alpha I$;*
2. *$(\frac{1}{\lambda_i}, \mathbf{v}_i)$ are eigenvalues/vectors of $A^{-1}$;*
3. *$(\lambda_i^k, \mathbf{v}_i)$ are eigenvalues/vectors of $A^k$.*

*Proof.* Let us omit the subscript $i$ to ease notation:

1. $(A + \alpha I)\mathbf{v} = A\mathbf{v} + \alpha\mathbf{v} = \lambda\mathbf{v} + \alpha\mathbf{v} = (\lambda + \alpha)\mathbf{v}$;
2. $(\lambda^{-1}\mathbf{v})$ is an eigenpair of $A^{-1}$. We need to check that $A^{-1}\mathbf{v} = \lambda^{-1}\mathbf{v}$. If we multiply by $\lambda A$ both sides: $\lambda \cancel{A A^{-1}}\mathbf{v} = \lambda A \lambda^{-1}\mathbf{v} \Leftrightarrow \lambda\mathbf{v} = \cancel{\lambda\lambda^{-1}} A\mathbf{v}$, which is true by definition of eigenvalue/vector of $A$;
3. (by induction) $A^2\mathbf{v} = A(A\mathbf{v}) = A \cdot \lambda\mathbf{v} = \lambda A v = \lambda\lambda v = \lambda^2 v$.

□

This lemma gives us the chance to use Arnoldi algorithm to compute the eigenvalues of a slightly modified matrix $B = (A - \mu I)^{-1}$. If $(\lambda, v)$ is an eigenpair of $A$, then $(\lambda - \mu^{-1}, v)$ is an eigenpair of $B$.

When is that $(\lambda - \mu^{-1})$ is large? When $\lambda$ and $\mu$ are close.

If $A$ has eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_m$, then the eigenvalues of $B$ are $\nu_1 = \frac{1}{\lambda_1 - \mu}, \nu_2 = \frac{1}{\lambda_2 - \mu}, \dots, \nu_m = \frac{1}{\lambda_m - \mu}$.

The problem is that such matrix $B$ is not sparse when $A$ is sparse.

We perform a trick to overcome this issue: we pass the function `eig` a $\lambda$-function that computes the product $Bz = A - \mu I^{-1} z$ without computing $B$ explicitly. The idea is to use factorizations: $A - \mu I = LU$, then $Bz = U^{-1}(L^{-1}z)$, that we can compute by back-substitution.

```
% computes 5 eigenvalues closest to mu=2
» fl = eigs(A, 5, mu);
» fl = eigs(A, 5, 'SM'); %smallest magnitude
» fl = eigs(A, 5, 'LM'); %largest magnitude
```

Notice that `eigs(A, 6, 1)` needs to factorize $A - 1 \cdot I = LU$, which might induce a lot of fill-in.

As final observation, the equivalents to Matlab's `eigs` function are `scipy.linalgs.eigs` for Python and `arpack` for C/C++ and Fortran.

# 15 7th of December 2018 — F. Poloni

In this lecture we are interested in using Arnoldi method to solve linear systems.

We can use the *sparse eigenvalues funtion* that we saw in last lecture and the **G**eneralized **M**inimum **RES**idual (GMRES).

Our task is to approximate the solution of a large-scale linear system of the form $Ax = b$ and our approach is to look for "the closest thing to solution" inside $K_n(A, b)$.

Through Arnoldi of $A$, $b$ and $n$, we obtained $[Q, H]$ and we can approximate the solution $x$ as $Q_1 y_1 + Q_2 y_2 + \cdots + Q_n y_n = Qy$, which is a good approximation of the solution inside $K_n(A, b)$, formally

$$\min_{x \in K_n(A,b)} \|Ax - b\|, \ x = Q_n y.$$

which is equivalent to $\min_{y \in \mathbb{R}^n} \|AQ_n y - b\|$.

We can perform some more reductions and:

$$\begin{aligned} \|AQ_n y - b\| &\overset{(1)}{=} \|Q_{n+1} \underline{H}_n y - b\| \\ &\overset{(2)}{=} \|Q_{n+1} \underline{H}_n y - Q_{n+1} \|b\| e_1\| \\ &= \|Q_{n+1} \cdot (\underline{H}_n y - \|b\| e_1)\| \\ &\overset{(3)}{=} \|\underline{H}_n y - \|b\| e_1\|. \end{aligned} \tag{15.1}$$

where $\overset{(1)}{=}$ is due to the equivalence $AQ_n = Q_{n+1} H_n$, with $H_n \in M(n+1, n)$, $\overset{(2)}{=}$ follows from the fact that $q_1 = \frac{b}{\|b\|}$ and $\overset{(3)}{=}$ is explained recalling that $Q_{n+1}$ is an orthogonal rectangular matrix in $M(mn + 1)$ and $\|z\| = \|Q_{n+1} z\|$, since $z^T z = z^T {Q_{n+1}}^T Q_{n+1} z$.

We got a LS problem of size $(n + 1) \times n$ (small), where $y \in \mathbb{R}^n$ and $e_1 \in \mathbb{R}^{n+1}$; moreover $\underline{H}$ has the following shape

$$H = \left[\begin{array}{cccc|cccc} * & \cdots & * & * & * & \cdots & \cdots & * \\ * & \cdots & * & * & * & \cdots & \cdots & * \\ 0 & \ddots & * & * & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & * & * & * & \cdots & \cdots & * \\ \hline & & & \circledast & * & \cdots & * & * \\ & & & & * & \cdots & * & * \\ & \mathbf{0} & & & 0 & \ddots & * & * \\ & & & & 0 & 0 & * & * \end{array}\right]$$

hence it is quite sparse.

$qr(H)$ can be computed in $O(n^2)$ using the fact that $H$ is 'almost triangular' (Hessenberg matrix), although it is not a big optimization, since $n$ Arnoldi steps need to be computed first.

Notice that instead of doing a QR at the end, we can compute QRs of $\underline{H}_1, \underline{H}_2, \ldots$ and update them at each step. This allows us to compute at each step a residual $\|Ax_n - b\|$ that we can use as stopping criterion.

**Something on Matlab ...**

Matlab has `gmres(A, b)` (and Python has `scipy.sparse.linalg.gmres`).

To estimate the convergence of GMRES we can see $x$ as a plynomial ($x = p(A)b$, such that $p(t) = \alpha_0 + \alpha_1 t + \cdots + \alpha_{n-1}t^{n-1}$ is a polynomial of degree $n-1$).

As far as the residual is concerned $Ax - b = A \cdot p(A) \cdot b - b = A \cdot (\alpha_0 I + \alpha_1 A + \cdots + \alpha_{n-1}A^{n-1}) \cdot b - b = q(A) \cdot b$, where $q(t) = t \cdot p(t) - 1$. If $A = V\Lambda V^{-1}$ diagonalizable, then $A^k = V \cdot \begin{bmatrix} {\lambda_1}^k & & \\ & \ddots & \\ & & {\lambda_m}^k \end{bmatrix} V^{-1}$ and

$$q(A) = V \begin{bmatrix} q(\lambda_1) & & \\ & \ddots & \\ & & q(\lambda_m) \end{bmatrix} V^{-1}.$$

All this computation was needed to write the residual GMRES in a clearer form:

$$\begin{aligned}
\min_{x \in K_n(A,b)} \|Ax - b\| &= \min_{\substack{q(x)=xp(x)-1 \\ \text{of degree} \le n}} \|Ap(A)b - b\| \\
&= \min_{\substack{q(x)=xp(x)-1 \\ \text{of degree} \le n}} \|q(A)b\| \\
&\le (\min_{\cdots} \|q(A)\|) \cdot \|b\| \\
&= \min_{\cdots} \left\| V \begin{bmatrix} q(\lambda_1) & & \\ & \ddots & \\ & & q(\lambda_m) \end{bmatrix} V^{-1} \right\| \\
&\le \min_{\cdots} \|V\| \cdot \left\| \begin{bmatrix} q(\lambda_1) & & \\ & \ddots & \\ & & q(\lambda_m) \end{bmatrix} \right\| \cdot \left\| V^{-1} \right\| \\
&\le K(V) \cdot \left\| \min_{\cdots} \begin{bmatrix} q(\lambda_1) & & \\ & \ddots & \\ & & q(\lambda_m) \end{bmatrix} \right\|
\end{aligned} \tag{15.2}$$

If $A$ has very few distinct eigenvalues ($k \le n$ of them), then we can find $q$ such that $q(\lambda_i) = 0$ for all $i$ and $q(0) = -1$, hence $n$ steps of GMRES give us the exact solution.

If $A$ has eigenvalues clustered in $n$ points in the plane, we can find a polynomial $q$ such that $q(\lambda_i)$ is small for all $i$.

Notice that Gauss operations on the rows of any matrix $A$ (e.g. swapping rows or scalar multiplication of a row) change its eigenvalues, without changing the solution.

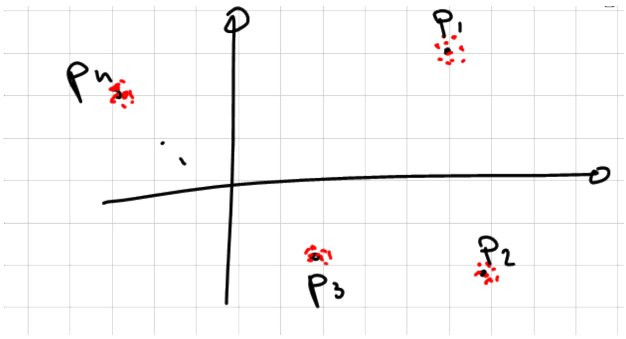



FIGURE 15.1: In this picture the eigenvalues are clustered around $P_1, P_2, P_3$ and $P_4$. We can find a polynomial $q$ such that $q(\text{red points}) \approx 0$.

More generally, given $P \in M(n, \mathbb{R})$ we can change the problem $Ax = b$ to $PAx = Pb$. If $P$ is invertible, the two systems have the same solution. However, the spectrum of $PA$ may be much better (in the above sense) than the spectrum of $A$, leading to a faster solution with GMRES.

In particular, this happens if we manage to find $P \approx A^{-1}$. The perfect choice would be $P = A^{-1}$, but, of course, if we knew $A^{-1}$ we would already have a way to solve linear systems: just compute the matrix multiplication $A^{-1}b$.

There are various techniques (often problem-dependent) to build effective *preconditioners* $P$. One comes from approximate LU factorizations of $A$ (in a suitable sense); they are known as *incomplete LU* preconditioners.

# 16 13th of December 2018 — F. Poloni

## 16.1 Lanczos algorithm

**Do you recall?**

In last lecture we saw how to factorize a matrix $A \in M(m, \mathbb{R})$ with Arnoldi (i.e. $AQ_n = Q_{n+1}\underline{H}_n = Q_n\underline{H}_n + h_{n+1,n}q_{n+1}{e_1}^T$).

If $A$ is **symmetric**, something special happens: $\underline{H}_n = Q_m^T A Q_m$ is symmetric as well, so it is a **tridiagonal** matrix. This improves the complexity of the Arnoldi process, because many iterations of the for loop (`j = 1 : n`) are not needed anymore, we need only two iterations.

This symmetric variant of Arnoldi is called *Lanczos algorithm*, and such algorithm reduces the cost to $n$ matrix products + $O(mn)$.

Suppose $A = A^T$ is positive definite. Then, we can find the solution to $Ax = b$ by minimizing the (strictly convex) function $f(x) = \frac{1}{2}x^T Ax - b^T x$.

Surprisingly, conjugate gradient on this problem can be interpreted as a Krylov subspace method.

The pseudocode of such algorithm can be found in Algorithm 16.1, where $x_k$ is the current iterate, $r_k = b - Ax_k = -\nabla f(x_k)$ is the residual and $d_k$ is the search direction.

ALGORITHM 16.1 Pseudocode for the conjugate gradient method.

```
1:  procedure CG_ITERATION
2:      x_0 ← 0;
3:      r_0 ← b;
4:      d_0 ← b;
5:      for k = 1:n do
6:          α_k ← (r_{k-1}^T r_{k-1})/(d_{k-1}^T A d_{k-1});
7:          x_k ← x_{k-1} + α_k d_{k-1};
8:          r_k ← r_{k-1} − α_k A d_{k-1};
9:          β_k ← (r_k^T r_k)/(r_{k-1}^T r_{k-1});
10:         d_k ← r_k + β_k d_{k-1};
11:     end for
12: end procedure
```

Notice that the search direction (line 10) is modified adding a multiple of the previous search direction to the residual and $\beta_k$ is chosen such that $d_k$ and $d_{k-1}$ are A-orthogonal (formally, ${d_k}^T A d_{k-1} = 0$).

Conversely, the next point is chosen in order to minimize the objective function $f(x_{k-1}\alpha_k d_{k-1})$.

As far as the complexity is concerned, the space complexity is constant (three vectors) and the time complexity is O(mn).

**Theorem 16.1.** $K_k(A,b) = span(x_1, x_2, \dots, x_k) = span(d_0, d_1, \dots, d_{k-1}) = span(r_0, r_1, \dots, r_{k-1})$.

**Theorem 16.2.** *The residuals are orthogonal and the search directions are A-orthogonal. Formally,* ${r_j}^T r_k = {d_i}^T A d_k = 0,\ \forall\ i < k$.

*Proof.* By induction: Let us assume we proved the thesis ${r_j}^T r_k = 0$ for $k-1, k-2, \dots, 0$. Since $x_k = x_{k-1}\alpha_k d_{k-1}$, the residual $r_k = b - Ax_k = b - A(x_{k-1} + \alpha_k d_{k-1}) = b - Ax_{k-1} - \alpha_k A d_{k-1} = r_{k-1} - \alpha_k A d_{k-1}$.

Let us compute ${r_j}^T r_k = {r_j}^T \cdot (r_{k-1} - \alpha_k A d_{k-1})$.

- If $i < k-1$ then ${r_j}^T r_{k-1} - {r_j}^T \alpha_k A d_{k-1} = 0$, because the first term is 0 from induction hypothesis and ${r_j}^T \alpha_k A d_{k-1} = 0$, because $r_j \in K_{k-1}(A,b) = span(d_0, d_1, \dots, d_{k-2})$.
- If $i = k-1$, $0 = {r_{k-1}}^T \cdot (r_{k-1} - \alpha_k A d_{k-1}) = {r_{k-1}}^T r_{k-1} - \alpha_k {r_{k-1}}^T A d_{k-1}$ holds if $\alpha_k = \frac{r_{k-1}^T r_{k-1}}{r_{k-1}^T A d_{k-1}}$. We are left with proving that $\alpha_k = \frac{r_{k-1}^T r_{k-1}}{r_{k-1}^T A d_{k-1}} = \alpha_k = \frac{r_{k-1}^T r_{k-1}}{d_{k-1}^T A d_{k-1}}$.

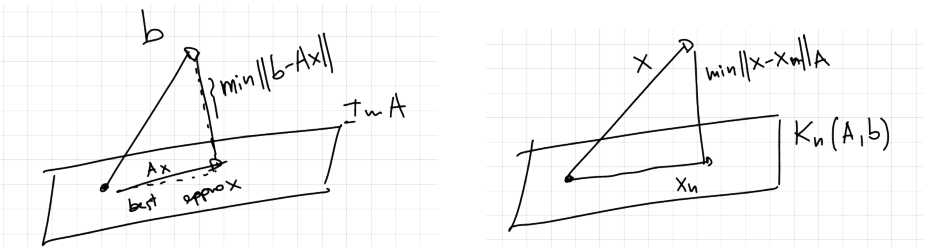
  This is true, since $d_{k-1} = r_{k-1} + \beta_{k-1} d_{k-2}$, so ${d_{k-1}}^T A d_{k-1} = (r_{k-1} + \beta_{k-1} d_{k-2})^T A d_{k-1} = {r_{k-1}}^T A d_{k-1} + \beta_{k-1} d_{k-2} {A d_{k-1}}^T$ and the second part is equal to 0 by induction.

□

Notice that this base is not orthonormal, we need to rescale it to obtain an orthonormal one, moreover, $\frac{1}{\|r_i\|} r_i$ coincides (up to a sign) with the $q_i$ obtained with Arnoldi.

We are left with writing the equation we need to solve at each iteration, namely we need to ensure that $r_k = b - Ax_k$ is orthogonal to all vectors of $K_k(A,b)$ which is equivalent to requiring ${Q_k}^T \cdot (b - Ax_k) = 0$ or, equivalently, $\|b\| \cdot e_1 = H_k y_k$.

In figure Figure 16.1 we can see a comparison between Arnoldi algorithm and the conjugate gradient.




(A) Arnoldi (B) Conjugate gradient

FIGURE 16.1: Traditional orthogonality (Arnoldi) leads to the minimization of the 2-norm, while in the conjugate gradient we impose A-orthogonality and we get a good approximation in several norms.

As far as convergence speed is concerned,

**Theorem 16.3.** *$x_k$ is the best approximation of the exact (and unknown) solution $x$ to $Ax = b$ in $K_k(A,b)$ in the A-norm, i.e.* $x_k = arg \min_{z \in K_k(A,b)} (x - z)^T A (x - z)$

**Theorem 16.4.** *Let $\lambda_{\max}$, $\lambda_{\min}$ be the maximum/minimum eigenvalue of $A$; then, CG converges with rate*

$$\|x - x_k\| \leq \left( \frac{\sqrt{\lambda_{\max}} - \sqrt{\lambda_{\min}}}{\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}}} \right)^k \|x - x_0\| .$$

We can rewrite it in terms of a more familiar quantity: for a positive definite matrix, eigenvalues and singular values coincide, hence

$$\frac{\sqrt{\lambda_{\max}} - \sqrt{\lambda_{\min}}}{\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}}} = \frac{\sqrt{\sigma_1} - \sqrt{\sigma_m}}{\sqrt{\sigma_1} + \sqrt{\sigma_m}} = \frac{\sqrt{\frac{\sigma_1}{\sigma_m}} - 1}{\sqrt{\frac{\sigma_1}{\sigma_m}} + 1} = \frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}.$$

For large values of $\kappa(A)$, this is approximately $1 - \frac{2}{\sqrt{\kappa(A)}}$, while if $\kappa(A) \approx 1$ the convergence speed is very high.

As for GMRES, if $A$ has only $n$ different eigenvalues, then this minimum reaches 0 after $n$ steps. If the eigenvalues of $A$ are 'clustered', one can construct polynomials such that $q(\lambda)$ is small for each $\lambda$ then fast convergence is implied.