Computational Mathematics

Based on prof. Antonio Frangioni's and
prof. Federico Poloni's lectures

Gemma Martini

June 17, 2020

A sincere thank you to
Alessandro Cudazzo, Donato Meoli,
Giulia Volpi and all those who helped me
improving these notes in style and contents.

Contents

0.1 20th of September 2018 — F. Poloni

0.1.1 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 $A \in M(n, m, R)$ and $\lambda \in \mathbb{R}$ we call the **scalar-matrix product** the following:
  $$\lambda A = A\lambda = \begin{pmatrix} \lambda A_{11} & \lambda A_{12} & \cdots & \lambda A_{1m} \\ \lambda A_{21} & \lambda A_{22} & \cdots & \lambda A_{2m} \\ \vdots & \vdots & \ddots & \\ \lambda A_{n1} & \lambda A_{n2} & \cdots & \lambda A_{nm} \end{pmatrix}$$

- **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 $A \in M(n, m, R)$ and $B \in M(m, k, R)$ 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.

  As long as the complexity is concerned, multiplying two matrices $m \times n$ and $n \times k$ requires $O(mnk)$ floating point operations (flops). Forget about fancier algorithms (e.g. Strassen)

  Order of operations

  Usual algebra properties hold, e.g.: $A(B+C) = AB+AC, A(BC) = (AB)C, \ldots$.
  Parenthesization matters a lot: if $A, B \in M(n, \mathbb{R})$, $v \in \mathbb{R}^n$, 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 0.1.1. ***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 0.1.1. *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 equality $AB = AC$. If there is a matrix M such that $MA = I$, the following holds*

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

In general, $AB = AC$ does not imply $B = C$; it holds only when A is 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 $M(3, 1, \mathbb{R})$) and v^T is a row vector (or a matrix in $M(1, 3, \mathbb{R})$).

Definition 0.1.2 (Basis)**.** *We call* ***basis*** *a set B of elements (vectors) in a vector space V 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.*

Definition 0.1.3 (Canonical basis)**.** *We term* ***canonical basis*** *of a vector space $\mathbb{R}^n$ the basis made of all the column of the $n \times n$ identity matrix $I_n \in M(n, \mathbb{R})$.*

Example 0.1.1. *In $\mathbb{R}^4$ the* ***canonical basis*** *is $\mathcal{B} = \{e_1, e_2, e_3, e_4\}$ such that*

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

and each vector $w \in R^4$ can be written as $w = w_1e_1 + w_2e_2 + w_3e_3 + w_4e_4$.

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.

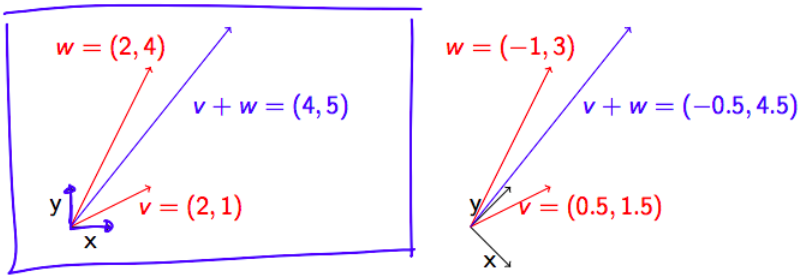


FIGURE 0.1: How a change of basis reflects on the space.

Definition 0.1.4 (Triangular matrix)**.** *Let $A \in M(n, \mathbb{R})$. We term A* ***upper triangular*** *if $(A)_{ij} = 0$ for each $i < j$. Conversely, we term A* ***lower triangular*** *if $(A)_{ij} = 0$ for each $j < i$. The set of all triangular $n \times n$ real matrices is a group and it is denoted as $T(n, \mathbb{R})$.*

Definition 0.1.5 (Diagonal matrix)**.** *Let $A \in M(n, \mathbb{R})$. We term A* ***diagonal*** *if $(A)_{ij} = 0$ for each $i \neq j$. The set of all diagonal $n \times n$ real matrices is a group and it is denoted as $D(n, \mathbb{R})$.*

Definition 0.1.6 (Symmetric matrix)**.** *Let $A \in M(n, \mathbb{R})$. We term A* ***symmetric*** *if $(A)_{ij} = (A)_{ji}$ for each $i, j = 1, \ldots, n$. The set of all diagonal $n \times n$ real matrices is a group and it is denoted as $D(n, \mathbb{R})$.*

0.1.2 Solving Linear Systems

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

Definition 0.1.7 (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:

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

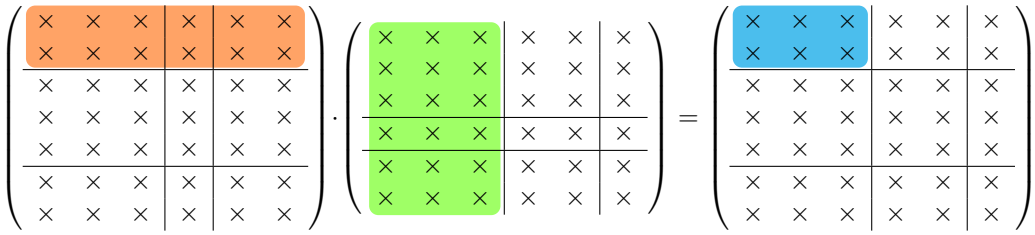
If we have a square and invertible mtrix $A \in GL(n, \mathbb{R})$ solving a linear system means: find those coordinates $x_1, \ldots, x_n$ needed to write b as a linear combination of the columns of (square) A and in this case, the solution is given by: $x = A^{-1}b$.
Warning: this is not the best way 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.
Notice that in Matlab both vector and matrices are 1-based.

Definition 0.1.8 (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.*

Observation 0.1.1. *When computing a matrix product, we get the same result if we use the row-by-column rule* block-wise.



Notice that 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 0.1.2 (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. In other words, 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 0.1.3 (Properties of block triangular matrices)**.**
Let M be a block triangular matrix, where all the blocks on the diagonal are square

$$M = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ 0 & A_{22} & \cdots & A_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & A_{nn} \end{pmatrix}$$

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 $M \in GL(n, \mathbb{R})$ such that $M = \begin{pmatrix} A & B \\ 0 & C \end{pmatrix}$ the inverse of M is*

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

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 0.2.

FIGURE 0.2: The adjacency matrix of a biparted graph has 0s in its bottom left part (Matlab syntax `A[p+1:n; 1:p]=0`), which means that the edges from a connected component and the other are in one direction only.

General principle: matrix structures matter. Block triangular linear system has a cheaper system solution than a general system as shown in Example 0.1.2.

Example 0.1.2. *Let us take a* 2×2 *block triangular linear system*

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

(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 multiplied by C.

0.1.3 Orthogonality

Definition 0.1.9 (Norms)**.** *Let* $x \in \mathbb{R}^n$. *We "measure" its 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: $\|x\|_0 = |\{i \ : \ |x_i| > 0\}|$, *which accounts for* $1 - \#$*of* 0 *entries;*

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

From now on in this part of the course, if not explicitly specified, we will refer to norm-2 only.

Definition 0.1.10 (Scalar product)**.** *Let* $v, w \in \mathbb{R}^n$ *we term* ***standard scalar product*** *between* v *and* w *the real number* $v^T w = \sum\limits_{i=1}^{n} v_i w_i$.

Definition 0.1.11 (Orthogonal vectors)**.** *Let* $v, w \in \mathbb{R}^n$*. We say that* v *is* ***orthogonal*** *to* w *(in symbols* $x \perp y$*) if their scalar product is zero.*

Formally, $x^T v = 0$.

Definition 0.1.12 (Orthogonal matrix)**.** *Let* $A \in M(n, \mathbb{R})$ *a square matrix. We term* A ***orthogonal*** *if* $A^T A = A A^T = I_n$ *where* I_n *is the identity matrix of size* n *(*1 *on the diagonal,* 0 *elsewhere) or equivalenlty if* $A^{-1} = U^T$.

The set of all orthogonal matrices in $M(n, \mathbb{R})$ *is a group and it called orthogonal group and denotes as* $O(n, \mathbb{R})$.

Fact 0.1.4. *Let* $A \in O(n, \mathbb{R})$, $\forall x \in \mathbb{R}^n$ *we have that* $\|Ax\| = \|x\|$.

Proof.

$$\|Ax\| = \sqrt{(Ax)^T Ax} = \sqrt{x^T A^T A x} = \sqrt{x^T I_n x} = \sqrt{x^T x} = \|x\|$$

□

0.2 26th of September 2018 — F. Poloni

0.2.1 Orthogonality (II)

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

Theorem 0.2.1. *Let $U \in O(n, \mathbb{R})$ and let $x \in \mathbb{R}^n$. Then $\|Ux\| = \|x\|$.*

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

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

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

Geometrically an orthogonal transformations (aka matrices) preserve the norm, so an orthogonal matrix A represents a symmetry or a rotation these operations do not alter the size of vectors.

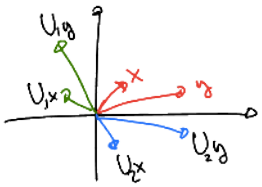


FIGURE 0.3: Matrix U_1 represents a rotation, while U_2 is a symmetry operation.

Definition 0.2.1 (Orthonormality)**.** *Let $x, y \in R^n$ we say that x and y are **orthonormal** if $< x, y >= 0$ and $\|x\| = \|y\| = 1$.*

Fact 0.2.2. *Let us take $U \in)(n, \mathbb{R})$. 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 0.2.3. *The set of orthogonal matrices is closed under product operations. Let $U, V \in O(n, \mathbb{R})$, then $U \cdot V$ is orthogonal.*

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

Since we will often deal with tall-thin rectangular matrices with orthonormal columns as U_1

$$U_1 = \begin{pmatrix} U^1 & U^2 & \cdots & U^n \end{pmatrix} \in M(m,n,\mathbb{R}) \text{where} m \geq n$$

the following fact may come in handy

Fact 0.2.4. *$\forall U_1 \in M(m,n,\mathbb{R})$ where $M \geq n$ and the columns of u_1 are orthogonal $\exists U_2$ s.t. $\begin{pmatrix} U_1 & U_2 \end{pmatrix} \in O(m,\mathbb{R})$.*

0.2.2 Eigenvalues / Eigenvector

Definition 0.2.2 (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** λ.*

Theorem 0.2.5. *Let $A \in M(n,\mathbb{R})$ (real triangular matrix). The eigenvalues of A are the scalars on the diagonal.*

Terminology

In this notes we refer to the columns to a generic matrix A as A^1 for the first column, A^2 for the second column and so on and so forth.
Conversely, we represent the rows of a matrix A as A_i.

Definition 0.2.3 (Diagonalizable matrix)**.** Some *matrices $A \in M(n,\mathbb{R})$ under some conditions can be decomposed as:*

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

$$A = V\Lambda V^{-1} = \begin{pmatrix} V^1 & V^2 & \cdots & V^n \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 & & 0 & \\ & \lambda_2 & & \\ & 0 & \ddots & \\ & & & \lambda_n \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}$$

where $V \in GL(n,\mathbb{R})$ is the matrix that has as columns the eigenvectors of A (V^i, λ_i) $\forall i = 1,\ldots,n$ and $w_i = {V_i}^{-1}$ are the rows of the inverse of matrix V.

Fact 0.2.6. *$\forall V^i \in \mathbb{R}$ eigenvectors of A they are still eigenvector of the diagonalized form of $A = V\Lambda V^{-1}$*

Proof.

$$AV^i = V\Lambda V^{-1}V^i = V\Lambda e_i = \lambda_i V^i$$

□

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

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

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

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 close to zero).

Notice that not all matrices $A \in M(n, \mathbb{R})$ allow a diagonal decomposition. It may happen that such a matrix A is diagonalizable in $\mathbb{C}$ and its eigenvalues are complex.
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 0.2.7. *If $A \in M(n, \mathbb{R})$ is diagonalizable (aka may be written as $A = V\Lambda V^{-1}$) then $A^k x = \sum_{i=1}^{n} \lambda_i^k \alpha_i V^i$, for some $\alpha_i \in \mathbb{R}$.*

Proof. Let us write x in the base of $\mathbb{R}^n$ made of the linearly independent columns of V:

$$x = V^1\alpha_1 + V^2\alpha_2 + \cdots + V^n\alpha_n$$

for some $\alpha_i \in \mathbb{R}$.

- Algebraic view point:

$$\begin{aligned} Ax &= A \cdot (V^1\alpha_1 + V^2\alpha_2 + \cdots + V^n\alpha_n) \\ &= AV^1\alpha_1 + AV^2\alpha_2 + \cdots + AV^n\alpha_n \\ &= \lambda_1 V^1\alpha_1 + \lambda_2 V^2\alpha_2 + \cdots + \lambda_n V^n\alpha_n \\ &= V^1(\lambda_1\alpha_1) + V^2(\lambda_2\alpha_2) + \cdots + V^n(\lambda_n\alpha_n) \end{aligned} \tag{0.2.1}$$

Then

$$\begin{aligned} A^2x &= A \cdot \Big(V^1(\lambda_1\alpha_1) + V^2(\lambda_2\alpha_2) + \cdots + V^n(\lambda_n\alpha_n) \Big) \\ &= AV^1\lambda_1\alpha_1 + AV^2\lambda_2\alpha_2 + \cdots + AV^n\lambda_n\alpha_n \\ &= {\lambda_1}^2V^1\alpha_1 + {\lambda_2}^2V^2\alpha_2 + \cdots + A{\lambda_n}^2V^n\alpha_n \end{aligned} \tag{0.2.2}$$

The thesis follows inductively.

- Linear algebra view point:

$$\begin{aligned} A^kx &= A \cdot A \cdot \ldots \cdot A \cdot x \\ &= V\Lambda\cancel{V^{-1}}\cancel{V}\Lambda\cancel{V^{-1}}\ldots\cancel{V}\Lambda V^{-1}x \\ &= V\Lambda^kV^{-1}x \\ &= V \begin{pmatrix} {\lambda_1}^k & & & \\ & {\lambda_2}^k & & \\ & & \ddots & \\ & & & {\lambda_n}^k \end{pmatrix} V^{-1}x \\ &= V \begin{pmatrix} {\lambda_1}^k & & & \\ & {\lambda_2}^k & & \\ & & \ddots & \\ & & & {\lambda_n}^k \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \ddots \\ \alpha_n \end{pmatrix} \end{aligned} \tag{0.2.3}$$

□

Notice that if A is not square, Av, λv have different sizes and it doesn't make sense to talk about eigenvalues.

Eigenvector: what could possibly go wrong?

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 we have 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 0.2.8. *Let $A \in M(n, \mathbb{R})$. If $|\lambda_i| < 1$ for all eigenvalues λ_i of A then* $\lim_{k\to\infty} A^k x = 0$.

Theorem 0.2.9. *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 0.2.10. *Let $A \in M(n, \mathbb{R})$ be a diagonalizable matrix and let*

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

Let us now consider a reordering of V's columns and apply the same permutation to the "diagonal vector" of Λ 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 $\hat{\Lambda}$ as $A = V\Lambda V^{-1} = \hat{V}\hat{\Lambda}\hat{V}^{-1}$.

Moreover, in the case of repeated eigenvalues

Fact 0.2.11. *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 0.2.12 (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.*

Notice that for symmetric matrices none of the "unlucky" situations enumerated above may happen (it is justified by the spectral theorem).

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.

Definition 0.2.4 (Quadratic form)**.** *Let $Q \in S(n, \mathbb{R})$ we define* ***quadratic form***

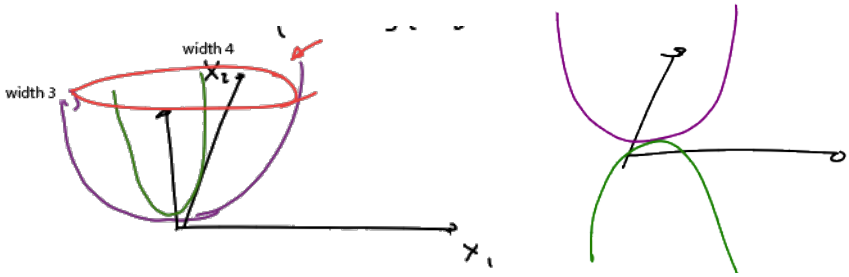
$$f(x) = x^T Q x$$

for each $x \in \mathbb{R}^n$.

Geometrically, a quadratic form defines a paraboloids.

Example 0.2.1. *Let* $f_1(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}) = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 3 & 2 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 3{x_1}^2 + 4x_1x_2 + 4{x_2}^2$ *and let* $f_2(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}) = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 3 & 2 \\ 2 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 3{x_1}^2 + 4x_1x_2 - 4{x_2}^2$

For a graphic hint see Figure 0.4.



(a) $f_1(x)$ (b) $f_2(x)$

FIGURE 0.4: The plot of functions.

Fact 0.2.13. *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 λ_{max} and λ_{min} are respectively the eigenvalue of maximum value and the eigenvalue of minimum value.

Proof. EASY CASE WITH Q DIAGONAL:

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

It is obvious that this sum is bounded by:

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

The following holds: $\lambda_{\min} \cdot ({x_1}^2 + {x_2}^2 + \cdots + {x_n}^2) = \lambda_{\min} \cdot x^T x = \lambda_{\min} \cdot \|x\|^2$
and, on the other hand, $\lambda_{\max} \cdot ({x_1}^2 + {x_2}^2 + \cdots + {x_n}^2) = \lambda_{\max} \cdot x^T x =$

$\lambda_{\max} \cdot \|x\|^2$ 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,

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

Now the point is that if we can replace $\left\|U^T x\right\|^2 = \|y\|^2$ with $\|x\|^2$ we have proved the theorem. In fact this is true, due to the orthogonality of matrix U and Proposition 0.1.4.

□

Corollary 0.2.14. *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 λ_{max} and λ_{min} are respectively the eigenvalue of maximum value and the eigenvalue of minimum value.*

Definition 0.2.5 (Positive semidefinite)**.** *Let $Q \in S(n, \mathbb{R})$. We say that Q is **positive semidefinite** (and we indicate $\succeq 0$) if*

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

Definition 0.2.6 (Positive definite)**.** *Let $Q \in S(n, \mathbb{R})$. We say that Q is **positive definite** (and we indicate $\succ 0$) if*

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

Fact 0.2.15. *Let $Q \in S(n, \mathbb{R})$. The following holds*

$$Q \text{ is positive semidefinite} \iff \forall \lambda \text{ eigenvalue of } Q \ \lambda \geq 0$$

And this holds with the > 0 in the positive definite case.

Proof.

($\Leftarrow$) $x^T Q x \geq \lambda_{\min} \geq 0$ since we are in the hypothesis that all the eigenvalues are ≥ 0

($\Rightarrow$) $\forall v_i$ eigenvector of Q $0 \leq {v_i}^T A v_i = {v_i}^T \cdot (\lambda_i v_i) = \lambda_i {v_i}^T v_i = \lambda_i \|v_i\|^2 \Rightarrow \lambda_i \geq 0$. □

Corollary 0.2.16. *Let $Q \in S(n, \mathbb{R})$ such that $Q \succeq 0$. The following holds:*

$$A \text{ is invertible} \iff Q \text{ is strictly positive definite}$$

Fact 0.2.17. *Let $B \in M(m,n,\mathbb{R})$ (possibly rectangular), $B^TB \in S(n,\mathbb{R})$ is a valid product and it is a square, symmetric, positive semidefinite matrix.*

Proof. SYMMETRY: $(B^TB)^T = B^T \cdot (B^T)^T = B^TB$.

POSITIVE DEFINITE: $x^TB^TBx = (Bx)^T(Bx) = \|Bx\|^2 \geq 0$

□

Corollary 0.2.18. *The same holds for $BB^T \in S(m,\mathbb{R})$ and it is easily proved taking $C = B^T \in M(n,m,\mathbb{R})$.*

Something on Matlab ...

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

Notice that in the **complex** case most of these properties work as well, but it is needed to replace A^T with $\overline{A^T}$* (transpose and entrywise conjugate). The norm of a complex vector is computed as

$$\|x\|_2^2 = x^*x = \overline{x_1}x_1 + \overline{x_2}x_2 + \cdots + \overline{x_n}x_n = |x_1|^2 + \cdots + |x_n|^2 \in \mathbb{R}^+ \cup \{0\}$$

Moreover, in the complex case, $UU^* = I$ is called **unitary matrix** (orthogonal + complex)

*Often denoted with A^* or A^H and called **Hermitian matrix**.

0.3 28th of September 2018 — F. Poloni

0.3.1 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 0.3.1 (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 0.3.2 (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 σ_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.*

$$\sigma_1 \geq |\lambda_1| \ \text{ and } \ |\lambda_m| \geq \sigma_m$$

λ_i is larger than the largest eigenvalue of a matrix A and λ_m is smaller than the smallest eigenvalue of a matrix A.

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

Definition 0.3.3 (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 Σ 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 0.3.4 (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, Σ, 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).

Properties of SVD

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

Definition 0.3.5 (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 0.3.1. *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 0.3.1, we can somehow talk about an "even thinner" SVD, where all the 0s in the bottom right part of the matrix Σ, 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 Σ can be found below.

$$\Sigma = \left(\begin{array}{ccc|ccc} \sigma_1 & & & & & \\ & \sigma_2 & & & & \\ & & \ddots & & & \\ & & & \sigma_r & & \\ \hline & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \\ \hline & & & \multicolumn{3}{c}{\mathbf{0}} \end{array}\right)$$

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 0.3.6 (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 0.1.9 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 0.3.2. *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 0.1.9:*

- $\|A\| \geq 0$ *(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 0.3.3. *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 0.3.7 (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 0.3.2 hold for the Frobenius norm as well.

Fact 0.3.4. *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 0.3.3, while the second is proved as follows:

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

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

$$= \sqrt{\sigma_1}^2 \cdot \frac{\sqrt{x_1{}^2 + x_2{}^2 + \cdots + x_n{}^2}}{\sqrt{x_1{}^2 + x_2{}^2 + \cdots + x_n{}^2}} = \sigma_1 \tag{0.3.1}$$

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 0.3.5 (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 0.3.6. *Let $A \in M(n, \mathbb{R})$ and let A be invertible. The following holds:* $\left\|A^{-1}\right\| = \frac{1}{\sigma_n}$

Proof. Since A is invertible, none of the σ_i is 0, hence the smaller (namely σ_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:

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

□

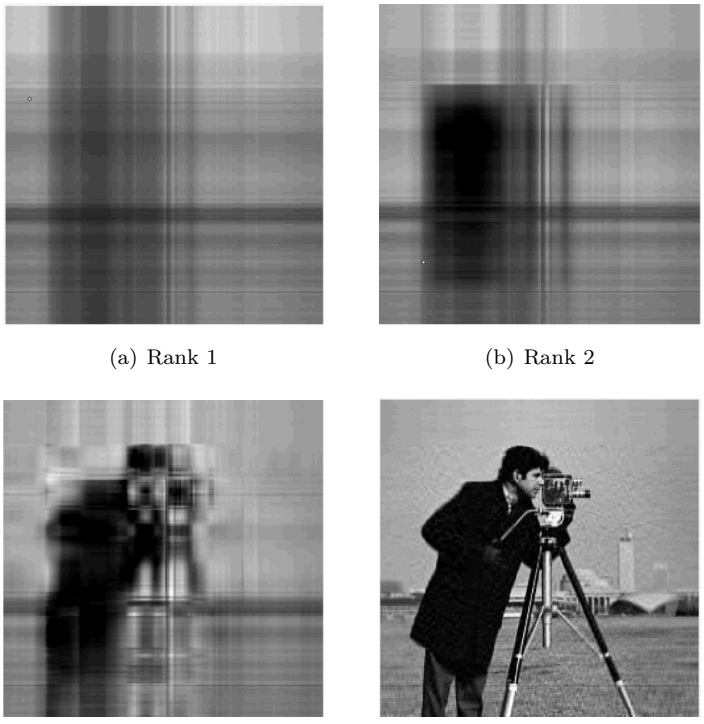
0.4 4th of October 2018 — F. Poloni

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 0.5. 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 0.5: How the approximation of a matrix changes with respect to the different ranks.

Definition 0.4.1 (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.

0.5 10th of October 2018 — F. Poloni

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

Definition 0.5.1 (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 0.5.1. *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 0.6.

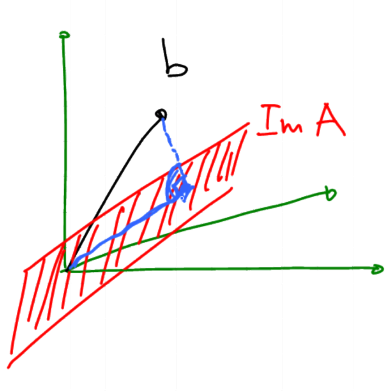


FIGURE 0.6: In this case the image of the matrix A (in red) does not contain b and the best one can do is to obtain a projection of b in the plane $Im(A)$ (drawn in blue).

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 0.5.2 (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 0.5.3 (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 0.5.2. *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 0.5.3. *Let $A \in M(n,m,\mathbb{R})$. A has full column rank iff A^TA is positive definite.*

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

0.6 18th of October 2018 — F. Poloni

0.6.1 Least squares problem

Fact 0.6.1. *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 0.6.2. *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 $f(x+h) = f(x) + (\nabla f(x))^T\dot{h} + o(\|h\|)$

$$\begin{aligned} f(x+h) &= (x+h)^TA^TA(x+h) - 2b^TA(x+h) + b^Tb \\ &= \mathbf{x^TA^TAx} + \mathbf{x^TA^TAh} + \mathbf{h^TA^TAx} + \mathbf{h^TA^TAh} \mathbf{-2b^TAx} - 2b^TAh + \mathbf{b^Tb} \\ &= \mathbf{f(x)} + (\mathbf{2x^TA^TAh} - 2b^TAh) + \mathbf{o(\|h\|)} \\ &= f(x) + \mathbf{(A^TAx - 2A^Tb)^Th} + o(\|h\|) \end{aligned} \tag{0.6.1}$$

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 $m > n$;

- $A^T b \rightarrow 2mn$;
- Solving $A^T Ax = A^T b$ with gaussian elimination has a computational complexity of $\frac{2}{3}n^3$;
- Cholesky factorization $A^T A = R^T R$ which has a cost of $\frac{1}{3}n^3$

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 $A^T A$

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?

$v^T(A^T Ax - A^T b) = 0$, see Figure 0.7.

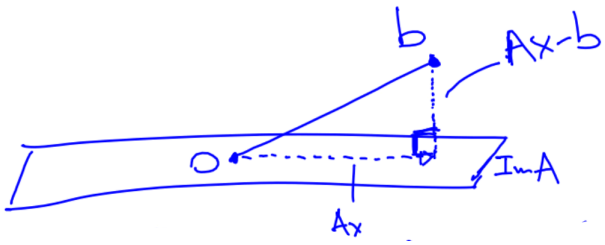


FIGURE 0.7: Geometric idea.

It's possible to find a close formula for solving this problem: $\min\|Ax - b\|$ is given by $x = (A^T A)^{-1} A^T b$.

Definition 0.6.1 (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 $\mathbf{A}^\dagger := (A^T A)^{-1} A^T$*

So we can write $x = A^\dagger b$ 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: $AA^\dagger = A(A^T A)^{-1} A^T = AA^{-1} A^{T^{-1}} A^T = I_{n\times m}$, but this doesn't hold for $A^\dagger A = (A^T A)^{-1}(A^T A)$, which is not the identity matrix.

0.6.2 QR factorization

Theorem 0.6.3. *$\forall A \in \mathcal{M}(m, m, \mathbb{R})$, $\exists Q \in \mathcal{O}(m, m, \mathbb{R})$ (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 0.6.1. *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 $\begin{pmatrix} s \\ 0 \\ . \\ . \\ . \\ 0 \end{pmatrix} = se_1$, where $s = \pm\|x\|$.

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

Definition 0.6.2 (Housholder reflector)**.** *Let U be a vector in $\mathbb{R}^m$. An **Householder reflector** is a matrix H such that $H = I - \frac{2}{U^T U} \cdot UU^T$.*

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 0.6.4. *Matrices of this form are orthogonal.*

Proof.

$$\begin{aligned} HH^T &= (I - \frac{2}{\|U\|^2} \cdot UU^T) \cdot (I - \frac{2}{\|U\|^2} \cdot UU^T) \\ &= I \cdot I - \frac{2}{\|U\|^2} \cdot UU^T I - I \cdot \frac{2}{\|U\|^2} \cdot UU^T + \frac{2}{\|U\|^2} \cdot UU^T \cdot \frac{2}{\|U\|^2} \cdot UU^T \\ &= I - \frac{2}{\|U\|^2} \cdot UU^T - \frac{2}{\|U\|^2} \cdot UU^T + \frac{4}{\|U\|^4} \cdot UU^T UU^T \\ &= I - \frac{4}{\|U\|^2} UU^T + \frac{4}{\|U\|^4} U\|U\|^2 U^T \\ &= I \end{aligned} \tag{0.6.2}$$

□

Example 0.6.2. $Hx = (I - \frac{2}{\|U\|^2} UU^T)x = x - \frac{2}{\|U\|^2} U(U^T U)$

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

$a = U' * x$

$b = U' * U$

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

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

A geometric idea is given by the following

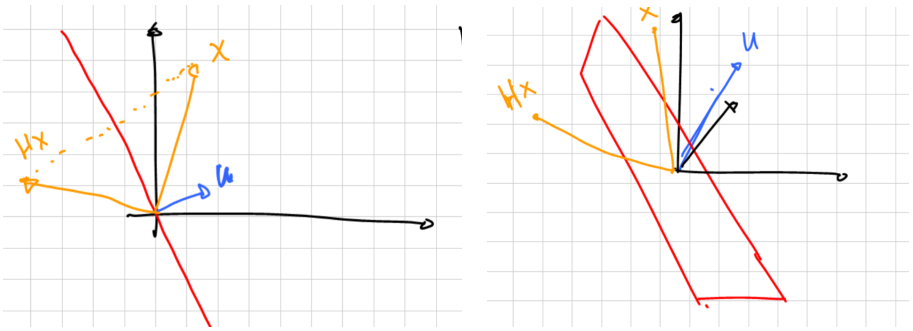


FIGURE 0.8: On the left in $\mathbb{R}^2$, on the right in $\mathbb{R}^3$.

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 $s = \|x\|$

Matlab syntax for functions

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

0.7 26th of October 2018 — F. Poloni

0.7.1 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 0.7.1 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)`.

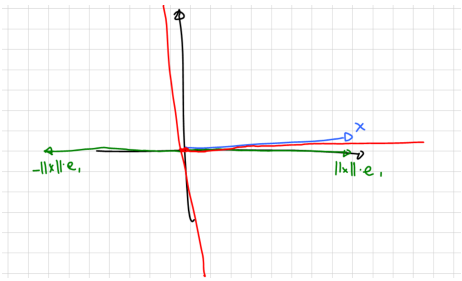


FIGURE 0.9: If x is oriented as in the plot it's better if we choose $-\|x\|e_1$ verse, since it's opposite to x.

Example 0.7.1. *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:*

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & & & & \\ & & & & \\ 0 & & & H_2 & \\ 0 & & & & \\ 0 & & & & \end{pmatrix} \cdot H_1A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & & & & \\ & & & & \\ 0 & & & H_2 & \\ 0 & & & & \\ 0 & & & & \end{pmatrix} \cdot \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix}$$
$$= \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix} \tag{0.7.1}$$

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:*

$$Q_3 \cdot (Q_2Q_1A) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & & & \\ & & & H_3 & \\ 0 & 0 & & & \\ 0 & 0 & & & \end{pmatrix} \cdot \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix}$$
$$= \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix} \tag{0.7.2}$$

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:*

$$Q_4 \cdot (Q_3 Q_2 Q_1 A) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & & \\ 0 & 0 & & H_4 & \\ 0 & 0 & & & \end{pmatrix} \cdot \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix} \tag{0.7.3}$$
$$= \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{pmatrix}$$

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,* $Q_1 \cdot Q_2 \cdot Q_3 \cdot Q_4 A = T$*, which is an upper triangular matrix.*

Theorem 0.7.1 (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.*

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

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

Matlab implementation

ALGORITHM 0.7.2 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 0.7.2. *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 0.7.3 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: $A = \begin{pmatrix} A_1 & A_2 \end{pmatrix} = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \cdot \begin{pmatrix} R_{1,1} & R_{1,2} \\ 0 & R_{2,2} \end{pmatrix}$ Then we can recover the factorization of A_1 from the factorization of A, since $A_1 = Q \cdot R_{11}$.

Fact 0.7.3 (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. $A_1 \in \mathcal{M}(m, n, \mathbb{R})$, $A = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \cdot \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = Q \cdot R = \begin{pmatrix} Q_1 \end{pmatrix} \cdot \begin{pmatrix} R_1 \end{pmatrix} + \begin{pmatrix} Q_2 \end{pmatrix} \begin{pmatrix} 0 \end{pmatrix} = Q_1 \cdot R_1$ □

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

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

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

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.

0.7.2 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 $m > n$ (a.k.a 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,

$$\begin{aligned} \|A\mathbf{x} - \mathbf{b}\| &= \left\|Q^T(A\mathbf{x} - \mathbf{b})\right\| = \left\|Q^TQR\mathbf{x} - Q^T\mathbf{b}\right\| \\ &= \left\|R\mathbf{x} - Q^T\mathbf{b}\right\| = \left\|\begin{pmatrix} R_1 \\ 0 \end{pmatrix}\mathbf{x} - \begin{pmatrix} {Q_1}^T \\ {Q_2}^T \end{pmatrix}\mathbf{b}\right\| \\ &= \left\|\begin{pmatrix} R_1\mathbf{x} - {Q_1}^T\mathbf{b} \\ {Q_2}^T\mathbf{b} \end{pmatrix}\right\| \end{aligned} \tag{0.7.4}$$

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 0.7.4. *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 $QR^TQR = R^TQ^TQR = R^TR = \begin{pmatrix} {R_1}^T & 0 \end{pmatrix} \cdot \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = {R_1}^T \cdot R_1$.

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

Note

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

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

0.8 7th of November 2018 — F.Poloni

0.8.1 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:

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

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: $A = U_1 \cdot \Sigma \cdot V^T$

.

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

$$
\begin{aligned}
\|Ax-b\| &= \left\|USV^T x - b\right\| && \leftarrow \text{ Def. of SVD} \\
&= \left\|U^T(USV^T x - b)\right\| && \leftarrow U^T \text{is orthogonal} \\
&= \left\|SV^T x - U^T b\right\| && \leftarrow \text{Distributivity + orthogonality} \\
&= \left\|Sy - U^T b\right\| && \leftarrow y = V^T x \\
&= \left\| \begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & & & & \\ & & & & \sigma_n \\ & & 0 & & \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_n \end{pmatrix} - \begin{pmatrix} {U^1}^T b \\ {U^2}^T b \\ \vdots \\ {U^n}^T b \\ {U^{n+1}}^T b \\ \vdots \\ {U^m}^T b \end{pmatrix} \right\| \\
&= \left\| \begin{pmatrix} \sigma_1 y_1 - {U^1}^T b \\ \sigma_2 y_2 - {U^2}^T b \\ \vdots \\ \sigma_n y_n - {U^n}^T b \\ \sigma_{n+1} y_{n+1} - {U^{n+1}}^T b \\ \vdots \\ \sigma_m y_m - {U^m}^T b \end{pmatrix} \right\|
\end{aligned}
\tag{0.8.1}
$$

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

$$
\begin{aligned}
x &= Vy && \leftarrow \text{Orthogonality of } V \\
&= V^1 y_1 + V^2 y_2 + \cdots + V^n y_n \\
&= V^1 \frac{1}{\sigma_1} {U^1}^T b + V^2 \frac{1}{\sigma_2} {U^2}^T b + \cdots + V^n \frac{1}{\sigma_n} {U^n}^T b \\
&= V \cdot \begin{pmatrix} \frac{1}{\sigma_1} & & & & \\ & \frac{1}{\sigma_2} & & & \\ & & \ddots & & \\ & & & \frac{1}{\sigma_n} & \end{pmatrix} \cdot U^T \cdot b \\
&= V \cdot \begin{pmatrix} \frac{1}{\sigma_1} & & & & \\ & \frac{1}{\sigma_2} & & & \\ & & \ddots & & \\ & & & \frac{1}{\sigma_n} & \end{pmatrix} \cdot {U_1}^T \cdot b
\end{aligned}
\tag{0.8.2}
$$

Which depends only on the tall, thin SVD.

Fact 0.8.1. *The σ_i are different from 0 iff A has full column rank.*

Proof. A has full column rank

$\Updownarrow$

$A^T A$ is invertible

$\Updownarrow$

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

invertible

$\Updownarrow$

$\forall i \ \sigma_i \neq 0$

□

Observation 0.8.1. *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 $O(15 \cdot n^3)$ for square matrices, while it's $O(m \cdot n^2)$ in the tall, thin case.

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

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 $y = \begin{pmatrix} -\frac{{U^1}^T b}{\sigma_1} \\ -\frac{{U^2}^T b}{\sigma_2} \\ \vdots \\ -\frac{{U^r}^T b}{\sigma_r} \\ * \\ \\ \end{pmatrix}$

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

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

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.

0.8.2 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 σ_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.

0.8.3 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 0.8.2. *The Tikhonov regularization is equivalent to*

$$x_{Tik} = \arg\min_{x \in \mathbb{R}^n} \left\| \begin{pmatrix} A \\ \alpha I \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|^2 \tag{0.8.3}$$

We show that the two objective functions coincide.

Proof.

$$\begin{aligned}\left\|\begin{pmatrix}A\\ \alpha I\end{pmatrix}x-\begin{pmatrix}b\\ 0\end{pmatrix}\right\|^2 &= \left\|\begin{pmatrix}Ax\\ \alpha x\end{pmatrix}-\begin{pmatrix}b\\ 0\end{pmatrix}\right\|^2\\ &= \left\|\begin{pmatrix}Ax-b\\ \alpha x\end{pmatrix}\right\|^2\\ &= \|Ax-b\|^2+\|\alpha x\|^2\\ &= \|Ax-b\|^2+\alpha\cdot\|x\|^2\end{aligned} \tag{0.8.4}$$

□

Fact 0.8.3. *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 (0.8.3) using the pseudoinverse

$$\begin{aligned}\begin{pmatrix}A\\ \alpha I\end{pmatrix}^{+}\begin{pmatrix}b\\ 0\end{pmatrix} &= \left(\begin{pmatrix}A\\ \alpha I\end{pmatrix}^{T}\begin{pmatrix}A\\ \alpha I\end{pmatrix}\right)^{-1}\begin{pmatrix}A\\ \alpha I\end{pmatrix}^{T}\begin{pmatrix}b\\ 0\end{pmatrix}\\ &= \left(\begin{pmatrix}A^T & \alpha I\end{pmatrix}\begin{pmatrix}A\\ \alpha I\end{pmatrix}\right)^{-1}\begin{pmatrix}A^T & \alpha I\end{pmatrix}\begin{pmatrix}b\\ 0\end{pmatrix}\\ &= (A^TA+\alpha^2 I)^{-1}A^Tb.\end{aligned} \tag{0.8.5}$$

□

Fact 0.8.4. *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 $z^TA^TA \geq 0$, since A^TA is positive semidefinite. □

Exercise 0.8.1. *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.*

Per casa: fare esercizio.

How can we choose k? We don't know.

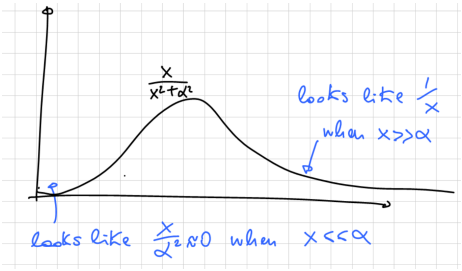


FIGURE 0.10: Here is the shape of the formula for the singular values.

0.9 9th of November 2018 — F. Poloni

0.9.1 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 0.9.1. *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 0.9.1 (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 0.11.

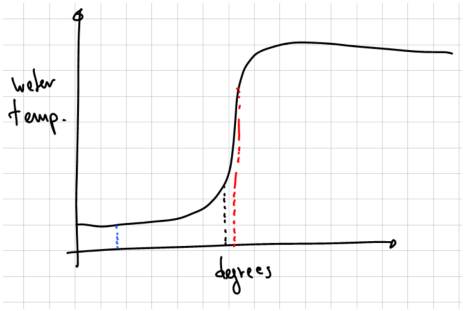


FIGURE 0.11: Geometric idea of temperature of the water in the shower

Definition 0.9.2 (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 0.12.

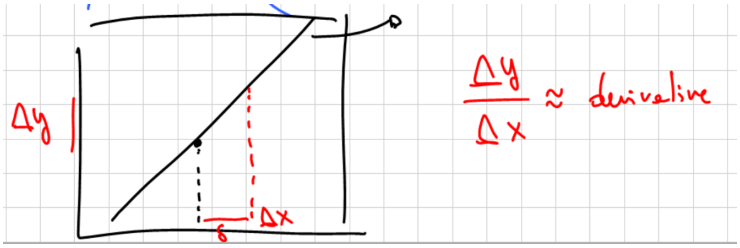


FIGURE 0.12: Geometric idea behind the derivative, as "zoom" of the function in a certain point.

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

Example 0.9.2. *Let $f(x) = x^2$.*

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

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

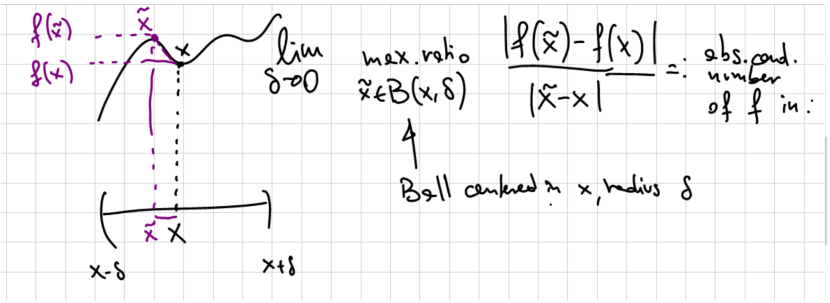


FIGURE 0.13: Geometric idea behind absolute condition number.

If we denote ε the radius of the ball, we obtain the following

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

then

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

Example 0.9.3. *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 $\mathbb{R}^3$.

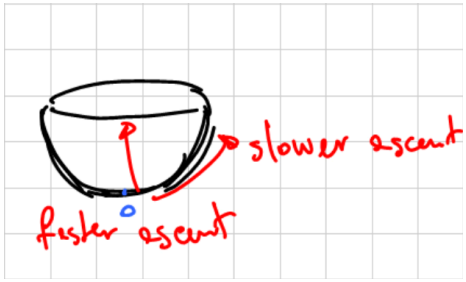


FIGURE 0.14: Paraboloid

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 $\lim_{\varepsilon\to 0} \max_{\widetilde{x}\in B(x,\varepsilon)} \frac{\left\|f(\widetilde{x})-f(x)\right\|}{\left\|\widetilde{x}-x\right\|}$, 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 $B(x,\varepsilon)$.

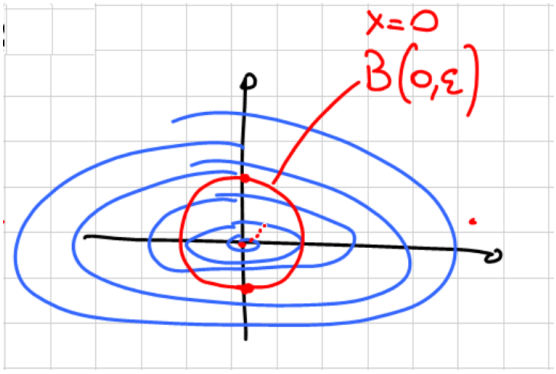


FIGURE 0.15: Level curves of a quadratic function ("seen from above").

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

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

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

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

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.
$$\begin{aligned}\|\widetilde{x} - x\| &= \left\|A^{-1}\widetilde{b} - A^{-1}b\right\| \\ &= \left\|A^{-1}(\widetilde{b} - b)\right\| \\ &\leq \left\|A^{-1}\right\| \cdot \left\|\widetilde{b} - b\right\|\end{aligned} \tag{0.9.1}$$

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

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

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

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

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

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

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

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

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

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

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.

0.9.2 Condition number, SVD, and distance to singularity

Fact 0.9.1. *$\kappa(A) = \frac{\sigma_1}{\sigma_n}$, 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 $\|A\| = \left\|U \cdot S \cdot V^T\right\| = \|S\| = \sigma_1$, since

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

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

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

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

Fact 0.9.2. *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

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

Proof.

bau (0.9.2)

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

$$= \left\| U \cdot \begin{pmatrix} 0 & & & & \\ & \cdot & & & \\ & & \cdot & & \\ & & & 0 & \\ & & & & \sigma_n \end{pmatrix} \cdot V^T \right\|$$

$$= \sigma_n \tag{0.9.3}$$

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.

0.9.3 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* $\kappa(A) = \frac{\sigma_1}{\sigma_n}$. We take this as a definition of $\kappa(A)$. Note that we cannot use the other definition $\kappa(A) = \|A\| \left\|A^{-1}\right\|$, since A^{-1} does not exist for a non-square A. However, one can verify that $\|A\| \cdot \left\|A^{\dagger}\right\| = \frac{\sigma_1}{\sigma_n} = \kappa(A)$, where $A^{\dagger}$ is the pseudoinverse.

Observation 0.9.1. *Note that $\frac{1}{\kappa(A)}$ is the relative distance to the closest $\hat{A}$ without full column rank.*

✯ θ The second quantity needed is the angle between Ax and b, see Figure 0.16. $\theta = \arccos \frac{\|Ax\|}{\|b\|}$

Now we can express the theorem:

Theorem 0.9.3 (Trefethen, Bau, Theorem 18.1)**.** *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*

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

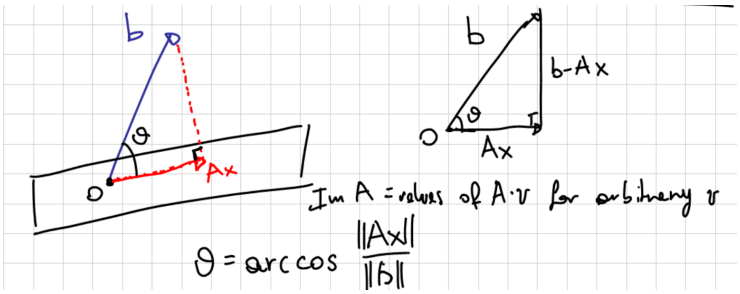


FIGURE 0.16: The triangle in the figure (the one whose cathets are Ax and $b - Ax$) is a square triangle.

and with respect to A it is

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

where θ 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 θ.

Observation 0.9.2.

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 0.17.*

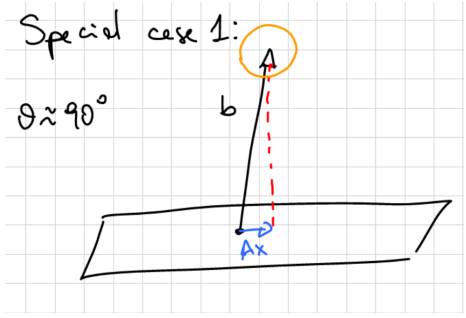


FIGURE 0.17: Special case 1

SPECIAL CASE 2: $\theta \approx 0°$ *When b is almost in plane with $Im(x)$. In this case cond $\approx \kappa(A)$, see Figure 0.18.*

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

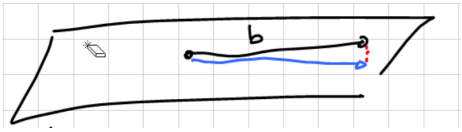


FIGURE 0.18: Special case 2.

0.10 15th of November 2018 — F. Poloni

0.10.1 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 0.19.

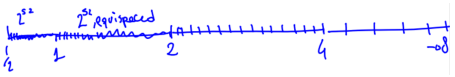


FIGURE 0.19: We have 2^{52} equispaced numbers between $\frac{1}{2}$ and 1 and between 1 and 2, and also 2^{52} between 2 and 4 and so on and so forth, so we have the same number of integers, although the space is enlarging.

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 0.10.1 (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 $f(a,b) = a^T b$.

$$a = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}, b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, \text{ so} f(a,b) = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

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

$$\begin{aligned}\text{computer result} &= a_1 \odot b_1 \oplus a_2 \odot b_2 \oplus a_3 \odot b_3 \\ &= \left[\left[a_1 b_1 \cdot (1+\delta) + a_2 b_2 \cdot (1+\delta_2)\right] \cdot (1+\delta_3) + a_3 b_3 \cdot (1+\delta_4)\right] \cdot (1+\delta_5) \\ &= a_1 b_1 \cdot (1+\delta_1) \cdot (1+\delta_3) \cdot (1+\delta_5) + a_2 b_2 \cdot (1+\delta_2) \cdot (1+\delta_3) \cdot (1+\delta_5) \\ &+ a_3 b_3 \cdot (1+\delta_4) \cdot (1+\delta_5) \\ &= a_1 b_1 (1 + \delta_1 + \delta_3 + \delta_5 + O(u^2)) \\ &+ a_2 b_2 \cdot (1 + \delta_1 + \delta_3 + \delta_5 + O(u^2)) \\ &+ a_3 b_3 \cdot (1 + \delta_4 + \delta_5 + O(u^2)) \\ &\approx a_1 b_1 (1 + \delta_1 + \delta_3 + \delta_5) + a_2 b_2 \cdot (1 + \delta_1 + \delta_3 + \delta_5) + a_3 b_3 \cdot (1 + \delta_4 + \delta_5)\end{aligned} \tag{0.10.1}$$

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

$$\begin{aligned}\mathrm{Err}_a &= \left|a_1b_1(\cancel{1}+\delta_1+\delta_3+\delta_5)+a_2b_2\cdot(\cancel{1}+\delta_1+\delta_3+\delta_5)+a_3b_3\cdot(\cancel{1}+\delta_4+\delta_5)+\cancel{a_1b_1}+\cancel{a_2b_2}+\cancel{a_3b_3}\right| \\ &\overset{(1)}{\leq} |a_1b_1|\cdot 3u+|a_2b_2|\cdot 3u+|a_3b_3|\cdot 2u \\ &\leq (|a_1b_1|+|a_2b_2|+|a_3b_3|)\cdot 3u\end{aligned} \tag{0.10.2}$$

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

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

$|$*computer result*$|\leq u\cdot(i+|1\cdot(1+\varepsilon)|+|-1\cdot 1|+|0\cdot 1|)=(2+\varepsilon)\cdot u$, *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.

$$\begin{aligned}\tilde{y} &= \dots \\ &= ((a_1b_1(1+\delta_1)+a_2b_2(1+\delta_2))(1+\delta_4)+a_3b_3(1+\delta_3))(1+\delta_5) \\ &= a_1\widetilde{b_1}+a_2\widetilde{b_2}+a_3\widetilde{b_3}\end{aligned} \tag{0.10.3}$$

where

$$\begin{aligned}
\widetilde{b_1} &= b_1(1+\delta_1)(1+\delta_4)(1+\delta_5) = b_1 + 3ub_1 + o(u),\\
\widetilde{b_2} &= b_2(1+\delta_2)(1+\delta_4)(1+\delta_5) = b_2 + 3ub_2 + o(u),\\
\widetilde{b_3} &= b_3(1+\delta_2)(1+\delta_5) = b_3 + 2ub_3 + o(u),
\end{aligned} \tag{0.10.4}$$

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

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

Hence,

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

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 δs.

Definition 0.10.2 (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 0.10.1. *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 0.10.1. *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.

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

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 $\widetilde{M} = \widetilde{x}^T \cdot \widetilde{y}$

Example 0.10.2. *Example (with exaggerated errors):*

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

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.

0.10.2 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

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

$$\Updownarrow$$

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

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

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

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

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

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

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

Mantra

Orthogonal transformations are the key for stability.

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 0.10.3. *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 $b = A \cdot \begin{pmatrix} 3 & 4 & 5 \end{pmatrix}$, so b lies in the image of A ($Im(A)$). The geometric idea in Figure 0.20.

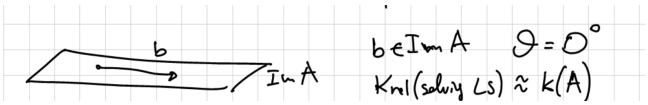


FIGURE 0.20: b lies (at least almost, because of numerical error) on the plane of $Im(A)$. In this case κ_{rel}(solving LS) $\approx \kappa(A)$.

A brief recap may be found in Table 1.

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 0.10.4. *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 0.10.3 (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 0.10.2. *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.

0.10.3 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 0.21.

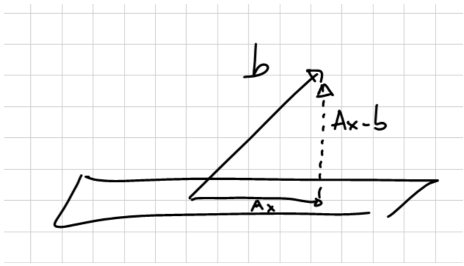


FIGURE 0.21: A can be as large as b if b is perfectly orthogonal.

Observation 0.10.3. *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 0.21, 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: $\min \|Ax - b\|^2 = \min x^T A^T A x - 2b^T A x + b^T b$.

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

Theorem 0.10.3. $\frac{\left\|\widetilde{x} - x\right\|}{x} \leq (\kappa(A))^2 \cdot \frac{\left\|A^T A x - A^T b\right\|}{\|A^T b\|}$. *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 0.10.4. $\frac{\left\|\widetilde{x} - x\right\|}{\|x\|} \leq \kappa_{rel,LS} \cdot \frac{\left\|\widetilde{b} - b\right\|}{\|b\|} = \kappa_{rel,LS} \cdot \frac{\|r\|}{\|b\|}$.

Theorem 0.10.5. *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 $\widetilde{x}$. □

0.11 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 0.22;

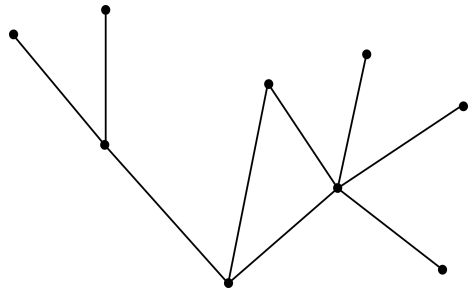


FIGURE 0.22: A local function on graph.

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 $T(A)_{ij} = \frac{1}{9} \cdot (A_{i-1j} + A_{i-1j-1} + A_{i-1j+1} + A_{ij} + A_{ij-1} + A_{ij+1} + A_{i+1j} + A_{i+1j-1} + A_{i+1j+1})$. 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.

0.11.1 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.

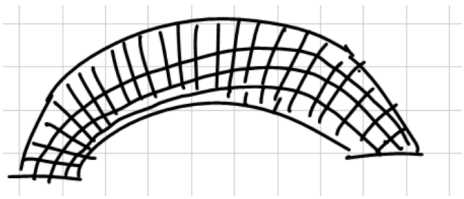


FIGURE 0.23: Graphic idea of a bridge partitioned into small blocks

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 0.11.1. *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 0.11.1 (Stroke of luck)**.** *The product of the ${L_i}^{-1}$'s (denoted L) can be computed for free, since the following holds:*

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

ALGORITHM 0.11.1 LU factorization, Matlab implementation.

```
function [L, U] = lu_factorization(A)
  m = size(A, 1);
  L = eye(m);
  U = A;
  for k = 1 : m - 1
    % compute "multipliers"
    L(k+1:end, k) = U(k+1:end, k) / U(k, k);
    % update U
    U(k+1:end, k) = 0;
    U(k+1:end, k+1:end) = U(k+1:end, k+1:end) ...
        - L(k+1:end, k) * U(k, k+1:end);
  end
```

The idea behind the implementation of Algorithm 0.11.1 is shown in Figure 0.24.

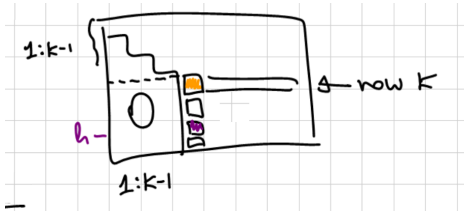


FIGURE 0.24: Assuming that we are at step k, we have that the h-th multiplier is expressed as $\frac{A_{hk}}{A_{kk}}$ and this muliplier goes to the right position in L.

Observation 0.11.2. *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^TA, which is too costly.
Obs: LU without pivoting isn't stable, as shown in Section 0.11.1, so Matlab uses pivoting.

missing

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 0.11.1.

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

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_is 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 0.11.3. *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 0.11.2. *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.

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.

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

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

Observation 0.11.4. *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.

0.12 23rd of November 2018 — F. Poloni

0.12.1 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:

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

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 0.12.1 (Stroke of luck)**.** *Notice that the stroke of luck of Observation 0.11.1 holds in this case too, hence we pay nothing to compute matrix L.*

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

Theorem 0.12.1 (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 0.12.1.

ALGORITHM 0.12.1 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 $P^T AP = LDL^T$, where D may have 2×2 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 $\forall z \neq 0 \in \mathbb{R}^m \; z^T Az > 0$.

Lemma 0.12.2. *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 0.12.3. *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 0.12.2 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.

0.12.2 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 0.12.2. *We will not discuss stability further, but Cholesky is always backward stable even without pivoting ($\|C\| = \|A\|^{1/2}$).*

Observation 0.12.3. *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.*

0.12.3 Krylov subspace methods

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

Example 0.12.1. *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 0.12.4. *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 0.12.1 (Krylof subspace)**.** *Let $A \in M(m, \mathbb{R})$ and let $b \in \mathbb{R}^m$. The **Krilov subspace** of index n is $\mathbf{K_n}(\mathbf{A}, \mathbf{b}) = span(b, Ab, A^2b, \dots, A^{n-1}b)$.*

Equivalently, $k \in K_n(A, b) \iff \exists \alpha_1, \dots, \alpha_{n-1} \in \mathbb{R}^m$ 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 0.12.5 (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 $b, Ab, A^2b, \dots, A^{n-1}b$ 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 $Av not \in K_n(A, b)$.*
5. *We may observe that $\dim(K_1(A, b)) < \dim(K_2(A, b)) < \cdots < \dim(K_{n_{max}}(A, b)) = \dim(K_{n_{max}+1}(A, b)) = \cdots$.*

0.13 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.

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.

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

0.13.1 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 0.13.1.

Algorithm 0.13.1 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 $b \in M(m, n)$.

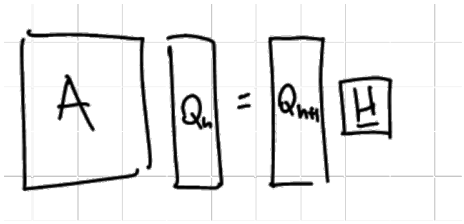


FIGURE 0.25: We started form a matirx A which has many entries and it gets factorized by the product of two smaller matrices

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.

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

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

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

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

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

$AQ_m = Q_mH$ 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:

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

Definition 0.13.1. *Let $H \in M(m, \mathbb{R})$ such that $H_{ij} = 0 \forall i > j + 1$. H is called **Hessemberg matrix**.*

Fact 0.13.1. *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 $K_{n+1}(A, b) = K_n(A, b)$?

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

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

In the implementation Algorithm 0.13.1 if we have a breakdown at step $n+1$ this means that $w = A_q n$ has already enough q_is, 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

$$AQ_m = H_mQ_m,$$

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

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,

$$\begin{aligned} x = A^{-1}b = Q_m H^{-1} {Q_m}^T b = Q_m H^{-1} \begin{bmatrix} {q_1}^T \\ {q_2}^T \\ \vdots \\ {q_m}^T \end{bmatrix} b \overset{(1)}{=} Q_m H^{-1} \begin{bmatrix} \|b\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ = \begin{bmatrix} Q_n & \widehat{Q} \end{bmatrix} \begin{bmatrix} {H_n}^{-1} & -{H_n}^{-1} L M^{-1} \\ 0 & M^{-1} \end{bmatrix} \begin{bmatrix} \|b\| \, \mathbf{e}_1 \\ 0 \end{bmatrix} \\ = \begin{bmatrix} Q_n & \widehat{Q} \end{bmatrix} \begin{bmatrix} \|b\| \, H_n^{-1} \mathbf{e}_1 \\ 0 \end{bmatrix} = Q_n \, \|b\| \, H_n^{-1} \mathbf{e}_1. \end{aligned} \tag{0.13.1}$$

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, λ such that $H_n v = \lambda v$ then $Q_n v$ is an eigenvector of A with eigenvalue λ
- 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 0.13.2. ***'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. $Q_n v$, λ 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 $Ax = b$

0.14 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.

0.14.1 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.

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

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, $\left\|\widetilde{H}_m - H_m\right\| = h_{n+1,n}$.

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

$$\begin{aligned} A^k b &= (V\Lambda V^{-1})\cdots(V\Lambda V^{-1})b \\ &= V\Lambda^k V^{-1} b \\ &= \begin{pmatrix} V^1 & V^2 & \cdots & V^m \end{pmatrix} \cdot \begin{pmatrix} {\lambda_1}^n & & & \\ & {\lambda_2}^n & & \\ & & \ddots & \\ & & & {\lambda_m}^n \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix} \qquad (0.14.1) \\ &= V^1\lambda_1^k c_1 + V^2\lambda_2^k c_2 + \cdots + V^m\lambda_m^k c_m \end{aligned}$$

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

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

Example 0.14.1. *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 0.14.1 (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 λ-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 0.14.1. *Let $A \in M(m, \mathbb{R})$ and let $(\lambda_1, \mathbf{v}_1), \ldots, (\lambda_k, \mathbf{v}_k)$ 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 λA 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 (λ, 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 λ and μ are close.

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

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 λ-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.

0.15 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 $Q_1y_1 + Q_2y_2 + \cdots + Q_ny_n = Qy$, 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:

$$\begin{aligned} \|AQ_n y - b\| &\overset{(1)}{=} \|Q_{n+1}\underline{H}_n y - b\| \\ &\overset{(2)}{=} \|Q_{n+1}\underline{H}_n y - Q_{n+1}\|b\| e_1\| \\ &= \|Q_{n+1} \cdot (\underline{H}_n y - \|b\| e_1)\| \\ &\overset{(3)}{=} \|\underline{H}_n y - \|b\| e_1\|. \end{aligned} \tag{0.15.1}$$

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 $z^Tz = z^T{Q_{n+1}}^TQ_{n+1}z$.

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

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

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 $\underline{H}_1, \underline{H}_2, \dots$ 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:

$$\begin{aligned}
\min_{x \in K_n(A,b)} \|Ax - b\| &= \min_{\substack{q(x) = xp(x) - 1 \\ \text{of degree} \le n}} \|Ap(A)b - b\| \\
&= \min_{\substack{q(x) = xp(x) - 1 \\ \text{of degree} \le n}} \|q(A)b\| \\
&\le (\min_{\cdots} \|q(A)\|) \cdot \|b\| \\
&= \min_{\cdots} \left\| V \begin{bmatrix} q(\lambda_1) & & \\ & \ddots & \\ & & q(\lambda_m) \end{bmatrix} V^{-1} \right\| \\
&\le \min_{\cdots} \|V\| \cdot \left\| \begin{bmatrix} q(\lambda_1) & & \\ & \ddots & \\ & & q(\lambda_m) \end{bmatrix} \right\| \cdot \|V^{-1}\| \\
&\le K(V) \cdot \left\| \min_{\cdots} \begin{bmatrix} q(\lambda_1) & & \\ & \ddots & \\ & & q(\lambda_m) \end{bmatrix} \right\|
\end{aligned} \tag{0.15.2}$$

If A has very few distinct eigenvalues ($k \leq n$ 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.

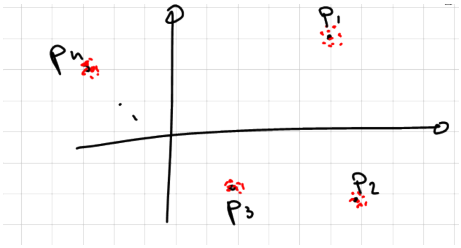


FIGURE 0.26: In this picture the eigenvalues are clustered around P_1, P_2, P_3 and P_4. We can find a polynomial q such that $q(\text{red points}) \approx 0$.

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.

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.

0.16 13th of December 2018 — F. Poloni

0.16.1 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 0.16.1, 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 0.16.1 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 β_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 0.16.1. $K_k(A,b) = span(x_1, x_2, \dots, x_k) = span(d_0, d_1, \dots, d_{k-1}) = span(r_0, r_1, \dots, r_{k-1})$.

Theorem 0.16.2. *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 $\alpha_k = \frac{r_{k-1}^T r_{k-1}}{r_{k-1}^T A d_{k-1}} = \alpha_k = \frac{r_{k-1}^T r_{k-1}}{d_{k-1}^T A d_{k-1}}$.

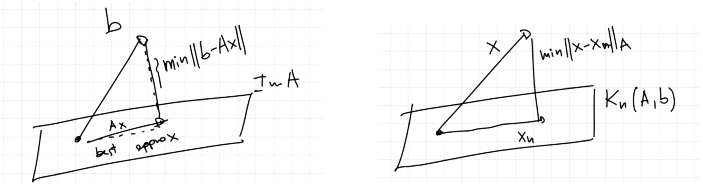
  This is true, since $d_{k-1} = r_{k-1} + \beta_{k-1} d_{k-2}$, so ${d_{k-1}}^T A d_{k-1} = (r_{k-1} + \beta_{k-1} d_{k-2})^T A d_{k-1} = {r_{k-1}}^T A d_{k-1} + \beta_{k-1} d_{k-2} {A d_{k-1}}^T$ and the second part is equal to 0 by induction.

□

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 0.27 we can see a comparison between Arnoldi algorithm and the conjugate gradient.



(a) Arnoldi (b) Conjugate gradient

FIGURE 0.27: 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 0.16.3. *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 0.16.4. *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 λ then fast convergence is implied.

Lecture 1

19th of September 2018

A. Frangioni

This course will deal with the optimization and numerical analysis of machine learning problems. We are not going to discuss difficult problems (e.g. NP-hard problems), besides we will present an efficient solution for simple ones (often **convex** ones), since we are dealing with huge amount of data.

Let us start with a warm up on machine learning problems.

1.1 Introduction to machine learning problems

Machine learning techniques are not as "young" as it might seem, the intuition has been there for ages, but we did not have enough calculus power. Machine learning algorithms have started working well recently, thanks to the many improvements in computer performances; for this reason, it is becoming a more and more popular subject to study.

The main idea behind machine learning is to take a huge amount of data (e.g. frames of a video for object-recognition) and squeeze them, in order to process them. This intuitive concept is translated into mathematical terms as "building a **model**" that fits our data. As in practical engineering problems, people want to construct a model (a small sized representation of the large thing we want to produce in the end) and try to understand its behaviour, before actually build such an object. Take as an example the problem of designing a jet. It is not clever to start building the plane before designing a cheap prototype to better study its behaviour in the atmosphere.

The kind of models we want to build are cheap to construct and as close as possible to the real problem we are studying. In physics, people try to find the best mathematical model to describe a real world phenomenon. The main issue is computation, since the more accurate the model, the more costly the prediction phase. Hence, a good model is a trade-off between accuracy and simplicity, namely it provides good prediction without incurring in slow computations.

The model, though, has to be parametric: we do not have only one model, we have a "shape" of a model, which is fit to our problem through the tuning of some parameters.

Example 1.1.1. *As an example, we are given three couples:* $f(x_1) = y_1$, $f(x_2) = y_2$, $f(x_3) = y_3$, *as shown in Figure 1.1.*

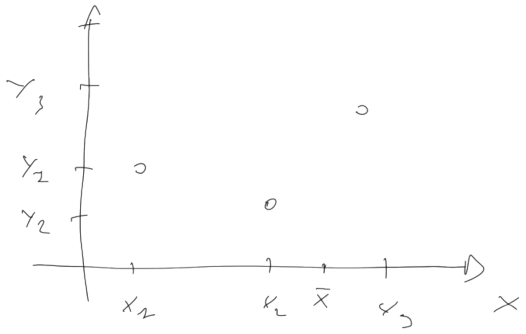


FIGURE 1.1: Geometric representation of the input. We are interested in finding a model that fits the input data and allows to predict $\bar{y}$ out of $\bar{x}$.

We need to make some choices: first, we need to decide the kind of model we believe is a good approximation of the objective function, say a linear model $f(x) = ax + b$. *After doing that, we are left with choosing its parameters (in order to pick a line among the whole family of linear functions), namely* a *and* b.

The aim is to build a model that fits the data we are given and then tune the parameters in order to achieve a good accuracy for a given application (recall that the model is parametric, hence the right values for the parameters have to be learned).

Another important characteristic of a good model is that it should not take too long to be built.

In this course we do not concentrate on the problem of finding the model that best fits our data, but we are already given a problem and a model and we only study its behaviour through its parameters. In other words, within the family of models with a given outline, we want to find the one that better represents the phenomena observed. This is called **fitting** and it is clearly some sort of optimization problem, where the fitting task is typically the computational bottleneck.

However, machine learning is more than simply fitting: fitting minimizes training error (or empirical risk), but ML aims at minimizing test error (i.e. generalization error).

A machine learning solution first builds a model that fits the observed data and then performs an hyper-parameter tuning in order to achieve a good "predicting power" on unseen inputs. In machine learning, a model that fits the

data while achieving good performances on new examples is said to be "non **overfitting**".
For machine learning purposes, a mathematical model should be:

- *accurate* (describing well the process under consideration)
- *computationally inexpensive* (providing answers rapidly)
- *general* (it can be applied to many different processes)

It goes without saying that achieving all these goals is practically impossible.

1.2 Optimization

In the rest of this lecture we are going to better understand what an optimization problem is, through some intuitive real world examples.

Terminology

In this course we will refer to vectors using the bold notation $\mathbf{v} \in \mathbb{R}^n$. The i-th entry of vector $\mathbf{v}$ is denoted as $v_i \in \mathbb{R}$, while the j-th pattern in the training set is a couple of vectors $(\mathbf{x^j}, \mathbf{y^j})$.
For more details on vectors and vector spaces see Section 2.4.

1.2.1 Linear estimation

Let us consider a phenomenon measured by one number $y \in \mathbb{R}$ that is believed to depend on a vector $\mathbf{x} = [x_1, \dots, x_n]$. We are provided a set of observations: $\{(y^1, \mathbf{x^1}), \dots, (y^m, \mathbf{x^m})\}$.

Definition 1.2.1 (Linear model)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be the objective function. We call $\widetilde{f}(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i + w_0 = \mathbf{w_+}^T \mathbf{x} + w_0$ the **linear model** of f for a given set of parameters, which is a vector $\mathbf{w} = (w_0, \mathbf{w_+}) = (w_0, w_1, \dots, w_n) \in \mathbb{R}^{n+1}$.*

How can we evaluate the "similarity" between our model and the objective function? Through computing the "error" or difference between the objective function value and the model prediction on each input. Under this assumption, the error function may be used to find the best parameters for our model, through solving a minimum problem:

Definition 1.2.2 (Least squares problem)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be the objective function, such that $f(\mathbf{x}) = \mathbf{y}$ and let $X\mathbf{w}$ be our linear model. Then we can find the best values for vector $\mathbf{w} \in \mathbb{R}^{n+1}$ by solving*

$$\min_{\mathbf{w}} \|\mathbf{y} - X\mathbf{w}\|$$

where $Y = \begin{pmatrix} \mathbf{y^1} \\ \vdots \\ \mathbf{y^m} \end{pmatrix}$ *and* $X = \begin{pmatrix} 1 & \mathbf{x^1} \\ \vdots & \vdots \\ 1 & \mathbf{x^m} \end{pmatrix}$.

If the matrix X is invertible then the simple solution is $\mathbf{w} = X^{-1}\mathbf{y}$. The point is that this operation is very costly when dealing with a huge number of entries (in the next paragraph we will see a way to manage it).

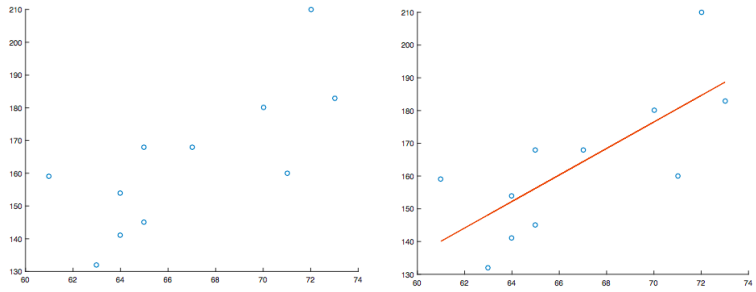


FIGURE 1.2: A linear estimation fitting example

1.2.2 Low-rank approximation

A (large, sparse) matrix $M \in M(n, m, \mathbb{R})$ describes a phenomenon depending on pairs (e.g., objects bought by customers) and we may want to approximate that matrix as the product between two smaller matrices (find a few features that describe most of users' choices): a "tall thin" matrix $A \in M(n, k, \mathbb{R})$ and a "fat large" $B \in M(k, m, \mathbb{R})$ (where $k \ll n, m$).

$$\boxed{M} \approx \boxed{A} \cdot \boxed{B}$$

This problem can be translated into a numerical analysis problem of the following shape

$$\min_{A,B} \|M - AB\|$$

The matrices A and B can be obtained from eigenvectors of $M^T M$ and MM^T, but that's a huge, possibly dense matrix. So a more efficient way should be used, which also avoids the explicit formation of $M^T M$ and MM^T because that would need a lot of memory.

Efficiently solving this problem requires:

- low-complexity computation
- avoiding the explicit forming of $M^T M$ and MM^T
- exploiting the structure of M (sparsity, similar columns, ...)
- ensuring that the solution is *numerically stable*

1.2.3 Support vector machines

Let us consider the so-called "decision problem": given a set of values of many parameters (variables) "label" a person i as ill or healthy, $y^i \in \{1, -1\}$.

The geometric intuition in two dimensions is given by Figure 1.3. We would like to find the line that better splits the plane into two regions, because this could help to diagnose the next patient. The rationale here is to maximize the space between the line and the nearest points (called **margin**), in order to have a better accuracy.

The distance between the two hyperplanes $(w_0, \mathbf{w}_+)$ and $(w_0', \mathbf{w}_+)$ in Figure 1.3 is computed as $\frac{2}{\|\mathbf{w}_+\|}$, so in order to maximize the distance between the planes we aim at minimizing $\|\mathbf{w}_+\|$. For each hyperplane that lies between $(w_0, \mathbf{w}_+)$ and $(w_0', \mathbf{w}_+)$ the following holds

$$\begin{cases} \mathbf{w}_+\mathbf{x}^i + w_0 \geq 1 & \text{if } \ y^i = 1 \\ \mathbf{w}_+\mathbf{x}^i + w_0 \leq -1 & \text{if } \ y^i = -1 \end{cases}$$

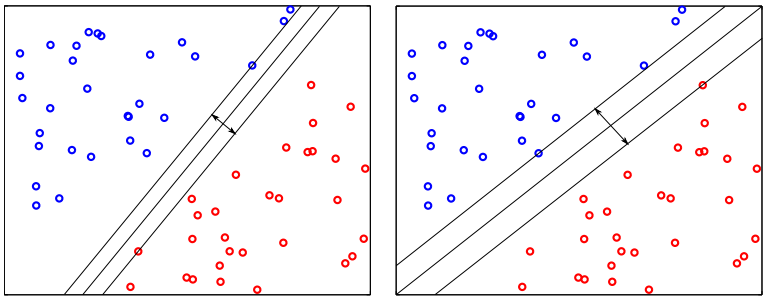


FIGURE 1.3: There are many possible boundaries that can be chosen as a model using many angular coefficients. Our best guess is the one that maximizes the distance between the line and the nearest points.

Under the hypotheses of this optimization problem, the maximum-margin separating hyperplane (assuming that any exists) is the solution of

$$\min_{\mathbf{w}} \{\|\mathbf{w}_+\|^2 \ : \ y^i(\mathbf{w}_+\mathbf{x}^\mathbf{i} + w_0) \geq 1, \ i = 1, \ldots, m\}$$

In practice, most of the times there is no such line. To overcome this issue we introduce the concept of "penalty" that accounts for the number of points that are misclassified, through the following

Definition 1.2.3 (SVM Primal problem)**.** *We term Support Vector Machine primal problem (SVM-P) a convex constrained problem with complex constraints that is formalized as*

$$\begin{cases} \min\limits_{\mathbf{w},\xi} \|\mathbf{w}_+\|^2 + C \sum\limits_{i=1}^{m} \xi_i \\ y^i(\mathbf{w}_+\mathbf{x}^i + w_0) \geq 1 - \xi_i \, , \ \xi_i \geq 0 \, , \ \forall i = 1, \ldots, m \end{cases} \tag{SVM-P}$$

*Where C is an **hyper-parameter** that weights the violation of the separating margin.*

This definition formalizes the intuition that the approximated function may have a greater norm and lead to a very small misclassification error, or it could be the other way round. Both these solutions are acceptable and their performances depend only on the problem.

Whenever we are able to solve a multi-objective optimization problem we are also able to solve what is called the **dual problem**, which in our case has the following shape.

Definition 1.2.4 (SVM Dual problem)**.** *We call Support Vector Machine dual problem (SVM-D) a convex constrained quadratic problem defined as*

$$\begin{cases} \max\limits_{\alpha} \sum\limits_{i=1}^{m} \alpha_i - \frac{1}{2} \sum\limits_{i=1}^{m} \sum_{j=1}^{m} \alpha_i < \mathbf{x^i}, \mathbf{x^j} > \alpha_j \\ \sum\limits_{i=1}^{m} y^i \alpha_i = 0 \\ 0 \leq \alpha_i \leq C \, , \, \forall i = 1, \ldots, m \end{cases} \tag{SVM-D}$$

The idea behind the theory of duality is to obtain the solution of a problem by solving an (apparently) different one.

Lemma 1.2.1. *Given an optimal solution α^* for the dual problem (SVM-D), then ${\mathbf{w}^*}_+ = \sum_{i=1}^{m} \alpha_i^* y^i \mathbf{x^i}$ is optimal for the primal problem (SVM-P).*

A reader who has a deeper background in the field of machine learning knows that the scalar product in the dual form allows the usage of the so-called "kernel trick". This mathematical transformation allows the mapping of the input space into a larger feature space where the points are more likely to be linearly separable.

Theoretically, the feature space can be infinite-dimensional, provided that the scalar product can be (efficiently) computed.

This whole course has the aim of presenting some techniques for solving efficiently **convex quadratic problems**, as the ones presented above.

Lecture 2

21st of September 2018

A. Frangioni

2.1 Mathematical background for optimization problems

Definition 2.1.1 (Minimum problem)**.** *Let S be a set, called **feasible region** and let $f : S \to \mathbb{R}$ be* any *function, called **objective function**. We term **minimum problem** the problem of finding the minimum value of such f. Formally,*

$$f_* = \min_x \{f(x) \ : x \in S\} \tag{P}$$

Definition 2.1.2 (Feasible solution)**.** *Let $x \in F$ be a solution of the minimum problem in which the domain is a superset of $S \subset F$. We say that x is a **feasible solution** if $x \in S$. Conversely, x is **unfeasible** if $x \in F \setminus S$.*

Definition 2.1.3 (Optimal solution)**.** *Under the same hypothesis of the above definition, we call x_* that realizes $f(x_*) = f_*$ an **optimal solution**, where $f_* \leq f(x) \, \forall x \in S$, $\forall v > f_* \, \exists \, x \in S$ s.t. $f(x) < v$.*

It is possible to find problems where there is no optimal solution at all.

Observation 2.1.1. *There are two cases in which it is not possible to find an optimal solution:*

1. *The domain is empty, which may be not trivial to prove, since it is an NP-hard problem sometimes;*
2. *We want to find the minimum of the objective function but the function is unbounded below ($\forall M \, \exists x_M \in S$ s.t. $f(x_M) \leq M$). Conversely, we aim to maximize a function, which is unbounded above.*

Example 2.1.1 (Bad optimization problems)**.** *The following practical cases are examples of problems that do not admit an optimal solution:*

- *empty case ($S = \emptyset$):* $\min\{f(x) = x \ : \ x \in \mathbb{R} \wedge x \leq -1 \wedge x \geq 1\}$
- *unbounded (below):* $\min\{f(x) = x \ : \ x \in \mathbb{R} \wedge x \leq 0\}$
- *bad f and S:* $\min\{f(x) = x \ : \ x \in \mathbb{R} \wedge x > 0\}$
- *bad f: let us consider an iterative algorithm that moves towards the optimum. It may happen that the function decreases and increases along a certain direction, such function is not continuous so it is impossible to reach the optimum:*

$$\min \left\{ f(x) = \begin{cases} x & \text{if } x > 0 \\ 1 & \text{if } x = 0 \end{cases} \quad x \in [0, 1] \right\}$$

Solving an optimization problem can be one of the following procedures:

1. Finding x_* and proving it is optimal
2. Proving $S = \emptyset$
3. Constructively proving that the function is unbounded ($\forall M \, \exists x_M \in S$ s.t. $f(x_M) \leq M$).

Typically "$x \in \mathbb{R}$" actually means "$x \in \mathbb{Q}$" with up to k digits precision and most of the times we consider optimal a solution which is close to the true optimal value, modulo some error (we call $\bar{x}$, the approximated optimal).

Definition 2.1.4 (Absolute error)**.** *We call **absolute error** the gap between the real value and the one we obtained. Formally,*

$$f(\bar{x}) - f_* \leq \varepsilon$$

Definition 2.1.5 (Relative error)**.** *We term **relative error** the absolute error, normalized by the true value of the function*

$$(\, f(\bar{x}) - f_* \,)/|\, f_* \,| \leq \varepsilon$$

2.1.1 Multi-objective Optimization

It may happen that there are more than one function that need to be minimized (maximized) and they could be constrasting or have incomparable units (apples vs oranges).

In multi-objective optimization, typically there is not a feasible solution that minimizes all objective functions simultaneously. It is in this context that *Pareto optimal solutions* are introduced: such solutions cannot be improved in any of the objectives without degrading at least one of the other objectives.

Definition 2.1.6 (Pareto frontier)**.** *Let (P) be the minimum multi-objective optimization problem formalized as*

$$\min_x \{[f_1(x), \ldots, f_k(x)] : x \in S\}$$

A feasible solution $x^1 \in S$ is said to ***(Pareto) dominate*** *another solution $\mathbf{x^2} \in S$, if*

$$\begin{cases} f_i(x^1) \leq f_i(x^2) & \forall i \in \{1, 2, \ldots, k\} \\ f_j(x^1) \leq f_j(x^2) & \forall j \in \{1, 2, \ldots, k\} \end{cases}$$

A solution $x^* \in S$ (and the corresponding outcome $f(x^*)$) is called ***Pareto optimal****, if there does not exist another solution that dominates it. The set of Pareto optimal outcomes is often called the* ***Pareto frontier****.*

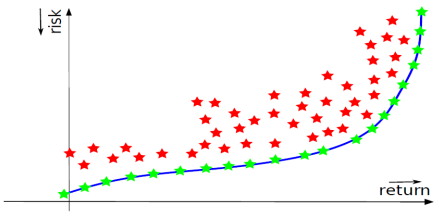


FIGURE 2.1: An example of Pareto frontier

Example 2.1.2. *Let us take two functions f_1 and f_2. We want to solve the minimization problem*

$$\min_x \{[f_1(x), f_2(x)] \ : x \in S\} \tag{P}$$

and to do so we present two different approaches:

SCALARIZATION. *Using a linear combination of the two functions, formally, $f(x) = \alpha f_1(x) + \beta f_2(x)$. An example, where $\alpha = 1$ is shown in Figure 2.2.*

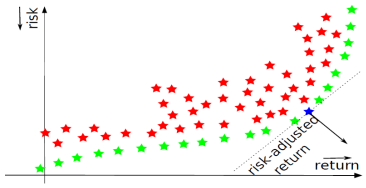
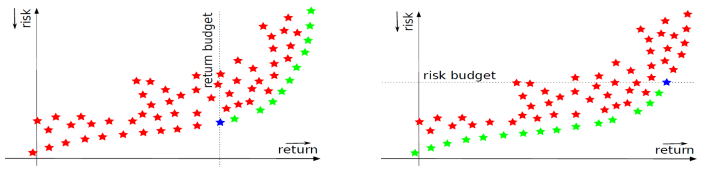


FIGURE 2.2: Maximize risk-adjusted return

BUDGETING. *Intuitively corresponds to taking into account only one objective function and add the others as constraints, provided that the values of the other functions are not too high. In formal terms,* $f(x) = f_1(x)$*, where* $S := S \cup \{x \in S \ : \ f_2(x) \leq b\}$*. As an example see Figure 2.3.*



(a) Maximize return with budget on maximum risk, $\min\{f_1(x) \ : \ f_2(x) \leq \beta \ : x \in S\}$

(b) Minimize risk with budget on minimum return,, $\min\{f_2(x) \ : \ f_1(x) \leq \beta \ : x \in S\}$

FIGURE 2.3: Budgeting

2.2 Infima, suprema and extended reals

Let us introduce some mathematical background on minimization (maximization).

Definition 2.2.1 (Totally ordered set)**.** *We say that a set* S *is* ***totally ordered*** *if* $\forall\, x, y \in S$*, either* $f(x) \leq f(y)$ *or* $f(y) \leq f(x)$*.*

Definition 2.2.2 (Infima and suprema)**.** *Let* R *be a totally ordered set and let* s *be one of its subsets* ($S \subseteq R$)*:*

$\underline{s}$ *is the* **infimum** *of* $S(\underline{s} = \inf S)$ *iff* $\underline{s} \leq s \ \ \forall s \in S \ \ \wedge \ \ \forall t > \underline{s} \ \exists\, s \in S \ s.t. \ s \leq t$

$\bar{s}$ *is the* **supremum** *of* $S(\bar{s} = \sup S)$ *iff* $\bar{s} \geq s \ \ \forall s \in S \ \ \wedge \ \ \forall t < \bar{s} \ \exists\, s \in S \ s.t. \ s \geq t$

An attentive reader may notice that it is possible that infima (suprema) do not exist in $\mathbb{R}$.

Definition 2.2.3 (Extended real)**.** *In the case of unbounded functions the value of infima or suprema are* ∞*, and we call* ***extended reals*** $\overline{\mathbb{R}} = -\infty \cup \mathbb{R} \cup +\infty$*.*

Property 2.2.1. *The following holds for the extended reals:*

- *For all* $S \subseteq \mathbb{R}, \sup/\inf S \in \overline{\mathbb{R}}$
- $\inf S = -\infty$ *indicates that there is no (finite) infimum*
- $\inf \emptyset = \infty$, $\sup \emptyset = -\infty$

2.3 (Monotone) Sequences in $\mathbb{R}$ and optimization

We are interested in studying sequences, because iterative methods start from a certain point and move towards the optimal.

Terminology

We denote a sequence of iterates as $\{x_i\} \subset S$ and we denote the plugging a function f into such a sequence as $v_i = f(x_i)$.

Definition 2.3.1 (Limit)**.** *Given a sequence $\{x_i\}$ the **limit** for $i \to \infty$ is defined as*

$$\lim_{i\to\infty} v_i = v \iff \forall \varepsilon > 0 \ \exists \ h \ s.t. \ |v_i - v| \leq \varepsilon \ \forall i \geq h$$

It may happen that a sequence has or does not have a limit. For example $\{\frac{1}{n}\}$ has limit 0 for $n \to +\infty$, while $\{(-1)^n\}$ does not have any.

Fact 2.3.1. *Let us be given a monotone sequence, then the sequence **does** have a limit.*

Notice that a sequence either is monotone or it can be "split" into two monotone sequences.

Example 2.3.1. *Let us consider the sequence $\{(-1)^n\}$. It does not converge to any value, but it can be split into $\{(-1)^{2n}\}$ and $\{(-1)^{2n+1}\}$ and these two subsequences are both monotone.*

Forcing monotonicity on sequences in $\mathbb{R}$ can be performed splitting the sequence into a non-decreasing sequence and a non-increasing sequence:

$$\underline{v}_i^* = \inf\{v_h : h \leq i\} \text{ and } \bar{v}_i^* = \sup\{v_h : h \leq i\}$$

This way we get $\underline{v}_1 \leq \underline{v}_2 \leq \dots$ and $\bar{v}_1 \geq \bar{v}_2 \geq \dots$ and they have a limit: $\liminf_{i\to\infty} v_i \lim_{i\to\infty} \underline{v_i}^* = v_\infty^* = f_*$ and $\limsup_{i\to\infty} v_i \lim_{i\to\infty} \bar{v}_i^* = v_\infty^* = f_*$

Fact 2.3.2. *The following holds:*

- $\bar{v}_i \geq \underline{v}_i \Rightarrow \limsup_{i\to\infty} v_i \geq \liminf_{i\to\infty} v_i$
- $\lim_{i\to\infty} v_i = v \iff \limsup_{i\to\infty} v_i = v = \liminf_{i\to\infty} v_i$

2.4 Vector spaces and topology

For the problems we are going to study, real numbers are used to describe each feature of a dataset and this has the mathematical equivalent in the term "vector".

Definition 2.4.1 (Euclidean vector space)**.** *We call* ***Euclidean space***

$$\mathbb{R}^n := \left\{ \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \ : \ x_i \in \mathbb{R}, \ i = 1, \ldots, n \right\}$$

Equivalently, we can characterize the Euclidean space as Cartesian product of $\mathbb{R}$ *n times:* $\mathbb{R}^n = \mathbb{R} \times \mathbb{R} \times \ldots \mathbb{R}$*. Every vector space and in particular Euclidean spaces are closed under sum and scalar multiplication.*

The main operations on elements of the Euclidean space (vectors) are:

$$\text{Sum: } \mathbf{x} + \mathbf{y} := \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{pmatrix}$$

$$\text{Scalar multiplication: } \alpha\mathbf{x} = \begin{pmatrix} \alpha x_1 \\ \alpha x_2 \\ \vdots \\ \alpha x_n \end{pmatrix}$$

In linear algebra, $\mathbf{x} \in \mathbb{R}^n$ is a column vector $\in \mathbb{R}^n$, whenever a row vector is needed for dimensionality issues it is denoted as x^T.

Definition 2.4.2 (Basis)**.** *Any vector* $\mathbf{x} \in \mathbb{R}^n$ *can be obtained from a linear combination of a set of vectors that is called "basis". In Euclidean spaces there exists a special basis that is called "canonical", where each vector* $\mathbf{u^i}$ *has a* 1 *in position* i *and a* 0 *elsewhere.*

Definition 2.4.3 (Finite vector space)**.** *Let* $V(K, n, m)$ *be a vector space. It is said to be* ***finite*** *if its basis have a finite cardinality.*

At this point, it is crucial to observe that not all vector spaces are finite nor they are a totally ordered set.
In this course we will deal with the concept of limit over a vector space very often and for this purpose we are going to introduce a topology on vector spaces (i.e. norm, scalar product, distance).

Definition 2.4.4 (Euclidean norm)**.** *Let* $\mathbf{x} \in \mathbb{R}^n$*, we define* ***euclidean norm*** *of* $\mathbf{x}$*:*

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} {x_i}^2} = \sqrt{< \mathbf{x}, \mathbf{x} >}$$

Fact 2.4.1. *Any norm on a vector space has the following properties:*

1. $\|\mathbf{x}\| \geq 0$ *and* $\forall \mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\| = 0 \iff \mathbf{x} = \mathbf{0}$;

2. $\|\alpha \mathbf{x}\| = |\alpha| \|\mathbf{x}\|, \ \forall \mathbf{x} \in \mathbb{R}^n, \ \alpha \in \mathbb{R}$;

3. $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|, \ \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ *(triangle inequality).*

Fact 2.4.2 (Cauchy-Schwartz inequality)**.** *Let* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$*. The following holds:*

$$< \mathbf{x}, \mathbf{y} >^2 \leq < \mathbf{x}, \mathbf{x} >< \mathbf{y}, \mathbf{y} > \equiv |< \mathbf{x}, \mathbf{y} >| \leq \|\mathbf{x}\| \|\mathbf{y}\|, \ \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$$

Fact 2.4.3 (Parallelogram Law)**.**

$$2 \|\mathbf{x}\|^2 + 2 \|\mathbf{y}\|^2 = \|\mathbf{x} + \mathbf{y}\|^2 + \|\mathbf{x} - \mathbf{y}\|^2$$

Fact 2.4.4.

$$\|\mathbf{x} + \mathbf{y}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 + 2 < \mathbf{x}, \mathbf{y} >$$

The 2-norm is a special case of the more general p-norm.

Definition 2.4.5 (p-norm)**.** *Let* $p \geq 1$ *be a real number, the p-norm of a vector* $\mathbf{x} = (x_1, \ldots, x_n)$ *is defined as follow:*

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

We require $p \geq 1$ *for the general definition of the p-norm because the triangle inequality fails to hold if* $p < 1$*. The p-norm is convex for* $p \geq 1$ *and non convex for* $p < 1$*.*

The most important p-norms are the following:

- $\|\mathbf{x}\|_1 := \sum_{i=1}^{n} |x_i|$
- $\|\mathbf{x}\|_\infty := \max\{|x_i| \ : \ i = 1, \ldots, n\}$
- $\|\mathbf{x}\|_i := |\{i \ : \ |x_i| > 0\}|$

Fact 2.4.5. *For any given finite-dimensional vector space* $V(K, n)$ *(e.g.* $\mathbb{R}^n$*), all norms on* V *are* equivalent *(therefore, convergence in one norm implies convergence in any other norm). Formally,* $\forall \|\cdot\|_A, \|\cdot\|_B$*:*

$$\exists \, 0 < \alpha < \beta \ \ s.t \ \ \alpha \|x\|_A \leq \|x\|_B \leq \beta \|x\|_A \qquad \forall \mathbf{x} \in V$$

This rule may not apply in infinite-dimensional vector spaces such as function spaces. For such cases we define the

Fact 2.4.6 (Holder's inequality)**.** *Let* $V(K, n)$ *be a (possibly infinite) vector space, for any two norms* $\forall \|\cdot\|_A, \|\cdot\|_B$ *the* ***Holder's inequality*** *holds*

$$< x, y >^2 \leq \|x\|_A \|y\|_B \ \ 1/A + 1/B = 1$$

Definition 2.4.6 (Ball)**.** *Let $\bar{\mathbf{x}} \in \mathbb{R}^n$. We term **ball** centered in $\bar{\mathbf{x}}$ and having ε as radius as the set of points that are close enough to $\mathbf{x}$: $B(\bar{\mathbf{x}}, \varepsilon) = \{\mathbf{x} \in \mathbb{R}^n \ : \ \|\mathbf{x} - \bar{\mathbf{x}}\| \leq \varepsilon\}$.*

Let us take a unit ball, if the center of the unit-ball is in the origin $(0, 0)$, then each point on the unit-ball will have the same p-norm (i.e. 1). The unit ball therefore describes all points that have "distance" 1 from the origin, where "distance" is measured by the p-norm. In Figure 2.4 we may observe the different shapes of the same ball varying the value of p in the p-norm.

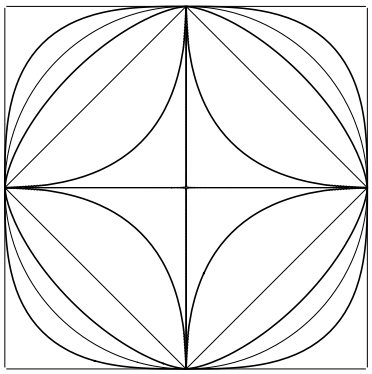


FIGURE 2.4: The shapes of balls centered in the origin of radius 1 varying the value of p-norm.

Definition 2.4.7 (Standard scalar product)**.** *Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ we define the **standard scalar product** between these two vectors as*

$$< \mathbf{x}, \mathbf{y} > := \ \mathbf{y}^T \mathbf{x} = \sum_{i=1}^{n} x_i y_i = x_1 y_1 + \cdots + x_n y_n$$

Fact 2.4.7. *A scalar product has the following properties:*

1. $< \mathbf{x}, \mathbf{y} >=< \mathbf{y}, \mathbf{x} > \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ *(symmetry)*
2. $< \mathbf{x}, \mathbf{x} > \geq 0, \ \forall \mathbf{x} \in \mathbb{R}^n, < \mathbf{x}, \mathbf{x} >= 0 \iff \mathbf{x} = \mathbf{0}$;
3. $< \alpha \mathbf{x}, \mathbf{y} >= \alpha < \mathbf{x}, \mathbf{y} >, \ \forall \mathbf{x} \in \mathbb{R}^n, \alpha \in \mathbb{R}$;
4. $< \mathbf{x} + \mathbf{y}, \mathbf{z} >=< \mathbf{x}, \mathbf{z} > + < \mathbf{y}, \mathbf{z} >, \ \forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^n$.

An important geometric characterization of the scalar product is the one that uses angles:

$$< \mathbf{x}, \mathbf{y} >= \|\mathbf{x}\| \, \|\mathbf{y}\| \cos\theta$$

According to this new definition, we can observe that $\mathbf{x}$ and $\mathbf{y}$ are orthogonal ($\mathbf{x} \perp \mathbf{y}$) iff $< \mathbf{x}, \mathbf{y} >= 0$ and in all the other cases they somehow point in the same direction.

We could rewrite the scalar product between $\mathbf{x}$ and $\mathbf{y}$ as $\mathbf{y}^T I \mathbf{x}$, where $I \in D(\mathbb{R}, n)$ is the identity matrix. Thanks to this observation. we can state the more general form of the scalar product:

Definition 2.4.8 (Scalar product)**.** *Let $\mathbf{x}, \mathbf{y} \in$ 'R^n and let $M \in M(\mathbb{R}, n)$ be a positive definite matrix (for further details see). We call **scalar product** the following*

$$< \mathbf{x}, \mathbf{y} >_M := \mathbf{y}^T M \mathbf{x}$$

Aggiungere reference al punto in cui si definisce una matrice definita positiva

Definition 2.4.9 (Euclidean distance)**.** *The **Euclidean distance** between points $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^n$ is the length of the line segment connecting them. In Cartesian coordinates, if $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ are two points in the n-dimensional Euclidean space, then the distance (d) from $\mathbf{x}$ to $\mathbf{y}$, or from $\mathbf{y}$ to $\mathbf{x}$ is given by*

$$d(\mathbf{x}, \mathbf{y}) := \|\mathbf{x} - \mathbf{y}\| = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}$$

Fact 2.4.8. *The Euclidean distance has the following properties:*

1. $d(\mathbf{x}, \mathbf{y}) \geq 0 \; \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n \; , \; d(\mathbf{x}, \mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y}$
2. $d(\alpha \mathbf{x}, 0) = |\alpha| d(\mathbf{x}, 0) \; \forall \mathbf{x} \in \mathbb{R}^n, \; \alpha \in \mathbb{R}$
3. $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y}) \; \forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^n$ *(triangle inequality)*

2.5 Limit of a sequence in $\mathbb{R}^n$

We have now all the tools to define the notion of limit of a sequence in $\mathbb{R}^n$.

Definition 2.5.1 (Limit of a sequence in the Euclidean space)**.** *Let $\{\mathbf{x_i}\} \subset \mathbb{R}^n$ be a sequence in $\mathbb{R}^n$. We say that $\{\mathbf{x_i}\}$ converges to a **limit** $\mathbf{x}$ for $i \to +\infty$ and denote $\lim_{i \to \infty} x_i = x$ or $\{x_i\} \to x$ if $\forall \varepsilon > 0 \; \exists h$ s.t. $d(\mathbf{x_i}, \mathbf{x}) \leq \varepsilon \; \forall i \geq h$*
or, equivalently, $\forall \varepsilon > 0 \; \exists h$ s.t. $\mathbf{x_i} \in \mathcal{B}(\mathbf{x}, \varepsilon) \; \forall i \geq h$
or, even $\lim_{i \to \infty} d(\mathbf{x_i}, \mathbf{x}) = 0$.

In the definition of limit there is the idea that the points of $\{\mathbf{x_i}\}$ eventually all come arbitrarily close to $\mathbf{x}$. Moreover, since $\mathbb{R}^n$ is not totally ordered, there is no obvious lim inf nor lim sup.

Lecture 3

27th of September 2018

A. Frangioni

Definition 3.0.1 (Minimizing sequence)**.** *Let $\{x_i\} \in S$ be a sequence and let $f : S \to \mathbb{R}^n$. We say that $\{x_i\}$ is a **minimizing sequence** if the sequence of function values $\{f(x_i)\}$ tends to the* greatest lower bound *of f ($f_* = \min\{f(x) \ : \ x \in S\}$).*

Example 3.0.1. *As an example, consider the following minimum problems and sequences:*

$$\min\{f(x) = x \ : \ x \in \mathbb{R} \wedge x > 0\} \text{ and the sequence } \{x_i = 1/i\}$$

$$\min\{1/x \ : \ x \in \mathbb{R} \wedge x > 0\} \text{ and the sequence } \{x_i = i\}$$

In both cases, $\{f(x_i)\} \to 0$, but it is not an optimum.

We want *conditions* that ensure $\{f(x_i)\} \to f_* \Rightarrow \{x_i\} \to x_* \in S \quad$ *optimal solution.*

Definition 3.0.2 (Interior and border of a set)**.** *Given $S \subseteq \mathbb{R}^n$, we say that $\mathbf{x} \in int(S)$ ($\mathbf{x}$ is an **interior point**) if it lies inside a ball contained in S. Formally, $\exists r > 0$ s.t. $\mathcal{B}(\mathbf{x}, \ r) \subseteq S$.*

*Moreover, we term **border points** those $\mathbf{x} \in \partial(S)$ such that all the points in the ball centered in x lie for a half inside the set S and for the other half outside. Formally, $\forall r > 0 \ \exists \mathbf{y}, \mathbf{z} \in \mathcal{B}(\mathbf{x}, \ r)$, where $\mathbf{y} \in S \wedge \mathbf{z} \notin S$. Intuitively, a point x lies on the border if, .*

Notice that a point on the boundary is not necessarily inside the ball, in that case we talk about **open set** (a set which is identical to its interior: $S = int(S)$).

Definition 3.0.3 (Closure of a set)**.** *Let $S \subseteq \mathbb{R}^n$ we term **closure** of S the set $cl(S) = int(S) \cup \partial S$.*

Definition 3.0.4 (Closed set)**.** *We say that a set $S \subseteq \mathbb{R}^n$ is **closed** if it coincides with his closure: $S = cl(S)$.*

*Equivalently, a set S is termed **closed** if its complementary ($\mathbb{R}^n \backslash S$) is open.*

It is interesting to notice that all the functions that lead to minimizing sequences and do not converge to an optimum are all defined in open sets.

Notice that there are sets that are both open and closed, for example $\mathbb{R}^n$.

Definition 3.0.5 (Bounded set)**.** *Let $S \subseteq \mathbb{R}^n$. We say that S is **bounded** if $\exists r > 0$ such that $S \subseteq \mathcal{B}(0, r)$.*

Intuitively, a bounded set does not go to ∞.

Definition 3.0.6 (Compact set)**.** *Let $S \subseteq \mathbb{R}^n$. We term S **compact** if it is both closed and bounded.*

Definition 3.0.7 (Accumulation point)**.** *Let $\{x_i\} \subseteq S$ be a sequence. We say that x is an **accumulation point** if $\exists\ \{x_{n_i}\}$ subsequence of $\{x_i\}$ converging to x. Formally, $\{x_{n_i}\} \to x$ or $\liminf_{i \to \infty} d(x_i,\ x) = 0$.*

Theorem 3.0.1 (Bolzano-Weierstrass)**.** *Let $\{x_i\} \subseteq S$ be a* bounded, real *sequence. Then it has a converging subsequence.*

Fact 3.0.2. *Let S be a compact set and let $\{x_i\} \subseteq S$ be a minimizing sequence for the objective function f. The limit of the sequence is a feasible solution.*

Proof. According to Bolzano-Weierstrass theorem $\{x_i\}$ has an accumulation point $x \in S$. Since it is minimizing $f(x)$ is feasible. □

Why did we say *feasible* but not *optimal*? If the function is not continuous (cfr. Figure 3.1) it may happen that the sequence is minimizing, but the limit is not the optimum.

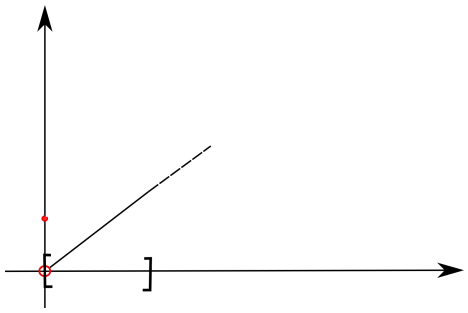


FIGURE 3.1: Case of non-continuity of the objective function in the border point $(0, 0)$.

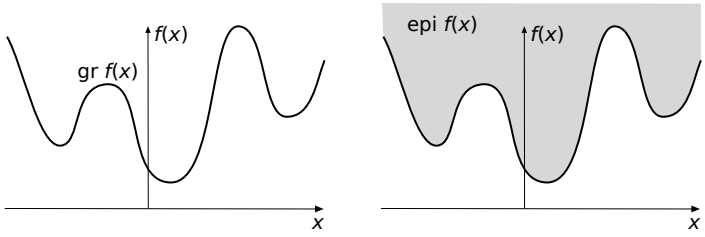
Definition 3.0.8 (Domain)**.** *Let $f : D \to \mathbb{R}$. We term D **domain** of f and denote $D = dom(f)$.*

In this course we will not take into account the domain of functions, because we can force all functions to be defined in the whole space as follows:

$f : \mathbb{R}^n \to \bar{\mathbb{R}}$, where $f(x) = \infty$ for $x \notin D$.

Definition 3.0.9 (Graph and epigraph)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$. We term **graph** $gr(f) = \{(f(\mathbf{x}),\ \mathbf{x}) \ :\ \mathbf{x} \in dom(f)\}$.*

*Conversely, we term **epigraph** $epi(f) = \{(v,\ \mathbf{x}) \ :\ \mathbf{x} \in dom(f) \wedge v \geq f(\mathbf{x})\}$, see **??**.*



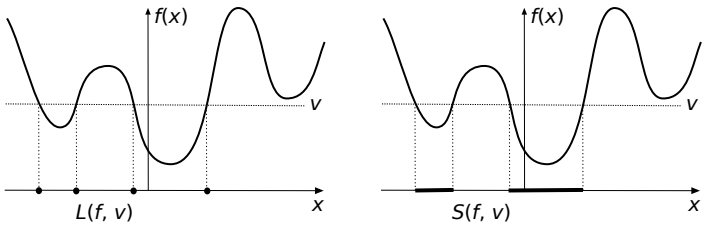
(a) Graph (b) Epigraph

FIGURE 3.2: Graph and epigraph

When dealing with multidimensional inputs spaces it is crucial to have mathematical tools that make possible to somehow "see" the function's behaviour in a lower-dimensional space.

Definition 3.0.10 (Level and sublevel set)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$. We term **level set** the set of all the inputs that have the same output value. Formally, $L(f, v) = \{\mathbf{x} \in dom(f) \ :\ f(\mathbf{x}) = v\}$.*

*Conversely, we term **sublevel set** the set of all the inputs which image is smaller than a fixed value v: $S(f, v) = \{\mathbf{x} \in dom(f) \ :\ f(\mathbf{x}) \leq v\}$, see Figure 3.3.*



(a) Level set (b) Sub-level set

FIGURE 3.3: Level and sub-level sets

3.1 Continuity

Definition 3.1.1 (Continuity)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$. We say that f is **continuous** in $\mathbf{x}$ if $\forall \varepsilon > 0\ \exists\, \delta > 0$ such that $\forall y \in \mathcal{B}(x,\ \delta)\ |f(y) - f(x)| < \varepsilon$.*

Property 3.1.1. *Let $f, g : \mathbb{R}^n \to \mathbb{R}$. The following holds:*

1. *$f + g$, $f \cdot g$ continuous at $\mathbf{x}$*
2. *$\max\{f,\ g\}$ and $\min\{f,\ g\}$ continuous at $\mathbf{x}$*
3. *$f \circ g \equiv f(g(\cdot))$ continuous at $\mathbf{x}$*

Theorem 3.1.2 (Intermediate value)**.** *Let $f : \mathbb{R} \to \mathbb{R}$. f is continuous on $[a,\ b]$ if $\forall v \in \mathbb{R}$ s.t. $\min\{f(a),\ f(b)\} \leq v \leq \max\{f(a),\ f(b)\}\ \exists\ c \in [a,\ b]$: $f(c) = v$.*

Theorem 3.1.3 (Weierstrass extreme value theorem)**.** *Let $S \subseteq \mathbb{R}^n$ be a compact set and let f be continuous on S. Then (P) has an optimal solution.*

Equivalently, let $S \subseteq \mathbb{R}^n$ compact and let f continuous on S. Then all accumulation points of any minimizing sequence are optima and there is at least one.

Definition 3.1.2 (Lipshitz continuity)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$. We term f **Lipschitz continuous** (L.c.) on $S \subseteq \mathbb{R}^n$ if $\exists L > 0$ such that*

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq L \left\|\mathbf{x} - \mathbf{y}\right\| \ \ \forall \mathbf{x}, \mathbf{y} \in S$$

*More generally, f is **globally Lipshitz continuous** if $S = \mathbb{R}^n$ and it is **locally Lipshitz continuous** at $\mathbf{x}$ if $\exists\ \varepsilon > 0\ S = \mathcal{B}(\mathbf{x},\ \varepsilon)$.*

Notice that Lipshitz continuity offers a stronger form of continuity and that the L constant value depends on S. The wider the set the smaller L.

Fact 3.1.4. *Let $f : \mathbb{R}^n \to \mathbb{R}$. On a compact set $S \subseteq \mathbb{R}^n$ if f is continuous then it is Lipshitz continuous.*

Definition 3.1.3 (Lower(upper) semi-continuity)**.** *Let $\{\mathbf{x_i}\} \subseteq \mathbb{R}^n$ be a sequence with accumulation point in $\mathbf{x}$ and let $f : \mathbb{R}^n \to \mathbb{R}$. f is **lower (upper) semi-continuous** (l.(u.)s.c.) at $\mathbf{x}$ if $f(\mathbf{x}) \leq \liminf\limits_{i \to \infty} f(\mathbf{x_i})$ ($f(\mathbf{x}) \geq \limsup\limits_{i \to \infty} f(\mathbf{x_i})$).*

Equivalently, $\liminf \mathbf{y} \to \mathbf{x} f(\mathbf{y}) \geq f(\mathbf{x})$, $\limsup\limits_{\mathbf{y} \to \mathbf{x}} f(\mathbf{y}) \leq f(\mathbf{x})$.

3.2 Derivatives

In this section we address the problem of inferring information on a complicated function around a certain value $\bar{x}$ using very simple functions, that are able to provide reliable information around that $\bar{x}$. Those functions are called "derivatives".
Let us start with a brief recap on single-dimension differentiability.

Definition 3.2.1 (Left and right derivative)**.** *Let $f : \mathbb{R} \to \mathbb{R}$. We term* ***left derivative*** $f'_-(x) = \lim_{t \to 0_-} [f(x+t) - f(x)]/t$.

Conversely, ***right derivative*** $f'_+(x) = \lim_{t \to 0_+} [f(x+t) - f(x)]/t$.

Definition 3.2.2 (Differentiable)**.** *Let $f : \mathbb{R} \to \mathbb{R}$. We say that f is* ***differentiable*** *at $x \in dom(f)$ if both left and right derivatives exist and coincide. Formally, $\exists f'_-, f'_+$ and $f'_-(x) = f'_+(x)$.*

Fact 3.2.1. *Let $f : \mathbb{R} \to \mathbb{R}$ and differentiable in $x \in dom(f)$ then f is continuous in x.*

Theorem 3.2.2 (Mean value theorem)**.** *Let $f : \mathbb{R} \to \mathbb{R}$ continuous on $[a, b]$ and differentiable on (a, b) then $\exists\, c \in (a, b)$ s.t. $f'(c) = (\, f(b) - f(a)\,)/(b - a)$.*

Theorem 3.2.3 (Rolle's theorem)**.** *Let $f : \mathbb{R} \to \mathbb{R}$. If $f(a) = f(b)$ then $\exists\ c \in (a, b)$ s.t. $f'(c) = 0$*

Corollary 3.2.4. *In the same hypothesis of Rolle's theorem, let a and b consecutive roots of f. Then $f'(a)$ and $f'(b)$ have opposite sign.*

3.2.1 Multivariate differentiability

Definition 3.2.3 (Partial derivative)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$. We term* ***partial derivative*** *of f w.r.t. x_i at $\mathbf{x} \in \mathbb{R}^n$ as*

$$\frac{\partial f}{\partial x_i}(\mathbf{x}) = \lim_{t \to 0} \frac{f(x_1,\ ,\ldots,\ x_{i-1},\ x_i + t,\ x_{i+1},\ \ldots,\ x_n) - f(\mathbf{x})}{t}$$

In other words, it is just $f'(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_n)$, considering each component x_j where $j \neq i$ as constant.

Definition 3.2.4 (Gradient)**.** *Given $f : \mathbb{R}^n \to \mathbb{R}$. We term* ***gradient*** *of f the vector of all the partial derivatives and we denote $\mathbb{R}^n \ni \nabla f = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})^T$.*

The gradient is the direction on which the function increases more rapidly, while the opposite of the gradient suggests the direction on which the function decreases most.

Definition 3.2.5 (Directional derivative)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$. We term* ***directional derivative*** *of f in point $\mathbf{x} \in dom(f)$ along direction $\mathbf{d} \in \mathbb{R}^n$*

$$\frac{\partial f}{\partial \mathbf{d}}(x) := \lim_{t \to 0} \frac{f(x + td) - f(x)}{t}$$

Notice that in a multivariate space it is not possible to assure differentiability checking if the directional derivatives are equal, because there is an infinite number of different directions.

Let us now introduce the notion of multivariate differentiability.

Definition 3.2.6 (Differentiable)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$. We say that f is **differentiable** at $\mathbf{x} \in \mathbb{R}^n$ if $\exists$ a linear function $\phi(h) =< c, h > +f(x)$ s.t.*

$$\lim_{\|h\| \to 0} \frac{\|f(x+h) - \phi(h)\|}{\|h\|} = 0$$

The intuition here is that the linear function should approximate f pretty well around x.

Fact 3.2.5. *Let $f : \mathbb{R}^n \to \mathbb{R}$ differentiable at $\mathbf{x}$. Then f is locally Lipschitz continuous at $\mathbf{x}$.*

Corollary 3.2.6. *Let $f : \mathbb{R}^n \to \mathbb{R}$ differentiable at $\mathbf{x}$. Then f is continuous at $\mathbf{x}$.*

Notice that the converse does not hold.

Fact 3.2.7. *Let $f : \mathbb{R}^n \to \mathbb{R}$, let $\mathbf{x} \in \mathbb{R}^n$ and let us assume $\exists \delta > 0$ s.t. $\forall i$ $\frac{\partial f}{\partial x_i}(\mathbf{y})$ is continuous $\forall \mathbf{y} \in \mathcal{B}(\mathbf{x}, \delta)$.*

Then f differentiable at point $\mathbf{x}$.

Notice that the converse of Proposition 3.2.7 does not hold either.

We are now ready to introduce a class of functions that allows good results for optimization: $\mathcal{C}^1$ class.

Definition 3.2.7 ($\mathcal{C}^1$ class)**.** *We call **$\mathcal{C}^1$-class** the class of functions with continuous gradient.*

Fact 3.2.8. *Let $f \in \mathcal{C}^1$, then f is differentiable everywhere and also continuous everywhere.*

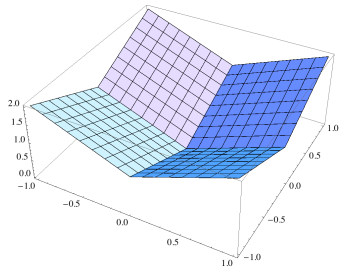
Definition 3.2.8 (First order model)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ and let $\mathbf{x} \in dom(f)$ We term **first order model** of f at point x*

$$L_x(y) = \nabla f(\mathbf{x})(y - \mathbf{x}) + f(\mathbf{x})$$

Fact 3.2.9. *Let $f : \mathbb{R}^n \to \mathbb{R}$ and let $x \in dom(f)$ such that f is differentiable at $\mathbf{x}$. Then $(L_x, f(\mathbf{x})) \perp S(f, f(\mathbf{x})) \perp \nabla f(\mathbf{x})$.*

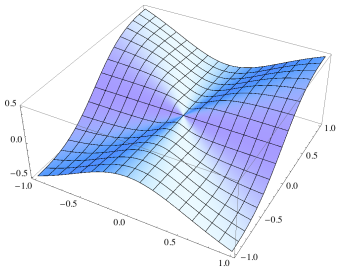
Geometrically speaking, we can observe that a function is smooth whenever its levelsets are smooth.

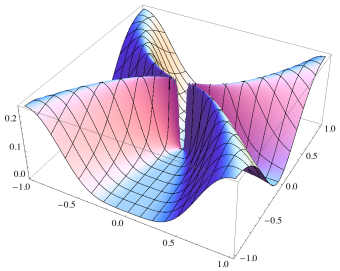
Example 3.2.1. *Let us consider some functions that are not differentiable in the minimum.*



The function $f(x_1, x_2) = \|[x_1, x_2]\| = |x_1| + |x_2|$ has some kinks.

The function $f(x_1, x_2) = \frac{{x_1}^2 x_2}{{x_1}^2 + {x_2}^2}$ *may be put to* 0 *in* $(0, 0)$ *for continuity, but it is still not differentiable in* $(0, 0)$ *although the function admits directional derivatives in* $(0, 0)$.





The function $f(x_1, x_2) = \left(\frac{{x_1}^2 x_2}{{x_1}^4 + {x_2}^2}\right)^2$ *is not continuous in* 0. *There are some directions that lead to the limit* 0, *while there are some other directions (the parabolas above) that do not lead to value* 0.

Lecture 4

3rd of October 2018

A. Frangioni

Example 4.0.1 (On derivatives)**.** *Let $f : \mathbb{R}^2 \to \mathbb{R}$ such that $f(x, y) = x^2 e^y$. Compute the partial derivatives and the directional derivatives in the two directions $\mathbf{d_1} = (0, 1)^T$ and $\mathbf{d_2} = (1, 0)^T$.*

- $\frac{\partial f}{\partial x} = 2xe^y$
- $\frac{\partial f}{\partial y} = x^2 e^y$
- $\frac{\partial f}{\partial \mathbf{d_1}} = \lim_{t \to 0} \frac{f(x+t\cdot 0, y+t\cdot 1)}{t} = \lim_{t \to 0} \frac{f(x,y+t)}{t}$. *An attentive reader may notice that the directional derivative in direction $\mathbf{d_1}$ is equivalent to the derivative on the second component.*
- $\frac{\partial f}{\partial \mathbf{d_2}} = \lim_{t \to 0} \frac{f(x+t\cdot 1, y+t\cdot 0)}{t} = \lim_{t \to 0} \frac{f(x+t,y)}{t}$. *Conversely with respect to what stated before, the directional derivative of f along the direction $\mathbf{d_2}$ is equivalent to the partial derivative w.r.t the first component.*

Let us compute the scalar product between the gradient and the direction $\mathbf{d_1}$:

$$\frac{\partial f}{\partial \mathbf{d_1}} = < \begin{pmatrix} 2xe^y \\ x^2 e^y \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} > = \begin{pmatrix} 2xe^y \cdot 0 \\ x^2 e^y \cdot 1 \end{pmatrix} = \begin{pmatrix} 0 \\ x^2 e^y \end{pmatrix}$$

The intuition of this example is formalized in the following

Fact 4.0.1. *Let $f : \mathbb{R}^n \to \mathbb{R}$. The partial derivative on a certain direction $\mathbf{d} \in \mathbb{R}^n$ is the scalar product between the gradient of the function and the direction, formally*

$$\frac{\partial f}{\partial d} = < \nabla f, \mathbf{d} >$$

Rivedere prima e seconda componente + parallelo con $e_1, \ldots, e_n$

Definition 4.0.1 (Vector-valued function)**.** *A function which codomain is multi-dimensional is called* ***vector-valued function****. Formally,* $f : \mathbb{R}^n \to \mathbb{R}^m$*, where* $f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_m(x) \end{pmatrix} \in \mathbb{R}^m$*.*

For such functions the computation of the derivative requires to specify not only the component with respect the one the derivation should be performed, but also the index of the function.

Definition 4.0.2 (Partial derivative for vector-valued functions)**.** *Let* $f : \mathbb{R}^n \to \mathbb{R}^m$*, the partial derivative of the* j*-th function with respect to the* i*-th component is*

$$\frac{\partial f_j}{\partial x_i}(\mathbf{x}) = \lim_{t \to 0} \frac{f_j(x_1, x_2, \ldots, x_{i-1}, \ldots, x_i + t, x_{i+1}, \ldots, x_n) - f_j(\mathbf{x})}{t}$$

Definition 4.0.3 (Jacobian)**.** *Let* $f : \mathbb{R}^n \to \mathbb{R}^m$ *we call* ***Jacobian*** *the matrix of all its first-order partial derivatives.*

$$Jf(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \nabla f_1(x) \\ \nabla f_2(x) \\ \vdots \\ \nabla f_m(x) \end{pmatrix}$$

Notice that when the number of variables increases the first derivative is a vector of numbers, the second derivative contains n^2 numbers.

Example 4.0.2. *Let* $f : \mathbb{R}^2 \to \mathbb{R}$ *such that* $f(x, y) = x^2 e^y$*. The gradient was computed above, resulting in*

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = \begin{pmatrix} 2xe^y \\ x^2 e^y \end{pmatrix}$$

Let us compute the second derivative of this function:

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x \partial x} & \frac{\partial^2 f}{\partial y \partial x} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y \partial y} \end{pmatrix} = \begin{pmatrix} 2e^y & 2xe^y \\ 2xe^y & x^2 e^y \end{pmatrix}$$

Definition 4.0.4 (Hessian)**.** *Let* $f : \mathbb{R}^n \to \mathbb{R}^m$*, we call* ***Hessian*** *of* f

$$\nabla^2 f(x) := J\nabla f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2}(x) & \frac{\partial^2 f}{\partial x_2 \partial x_1}(x) & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_1}(x) \\ \frac{\partial^2 f}{\partial x_1 \partial x_2}(x) & \frac{\partial^2 f}{\partial x_2^2}(x) & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_2}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) & \frac{\partial^2 f}{\partial x_2 \partial x_n}(x) & \cdots & \frac{\partial^2 f}{\partial x_n^2}(x) \end{pmatrix}$$

The size of the matrix grows very rapidly with the number of derivatives that we make.

In optimization, we like to work with Hessians, but they need to be handled, because they are very large, hence a trade off is needed.

In most cases, the Hessian is symmetric, meaning that the order in which we derive is not relevant and this happens when $\exists \delta > 0$ s.t. $\forall y \in \mathcal{B}(x, \delta)$ $\frac{\partial^2 f}{\partial x_j \partial x_i}(y)$ and $\frac{\partial^2 f}{\partial x_i \partial x_j}(y)$ exist and $\frac{\partial^2 f}{\partial x_j \partial x_i}(y)$ and $\frac{\partial^2 f}{\partial x_i \partial x_j}(y)$ are continuous at x.

Definition 4.0.5 ($\mathcal{C}^2$ functions)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}^m$. We say that f* ***belongs to $\mathcal{C}^2$ class*** *iff $\nabla^2 f(x)$ is continuous.*

Property 4.0.2. *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ such that $f \in \mathcal{C}^2$, then*

- $\nabla^2 f(x)$ *symmetric*
- $\nabla f(x)$ *continuous*

Definition 4.0.6 (First order Taylor model)**.** *Let $f : \mathbb{R}^n \to \mathbb{R} \in C^1$ in some $\mathcal{B}(x, \delta)$, we define the* ***first-order Taylor's formula*** *$\forall y \in \mathcal{B}(x, \delta)$ $\exists$ $\alpha \in (0, 1)$ s.t.*

$$f(y) =< \nabla f(\alpha x + (1 - \alpha)y,) > y - x + f(x)$$

Intuitively, it is a linear approximation of the function f in a neighbourhood of x.

Equivalently, we can write the so-called ***remainder version of first-order Taylor formula*** *as*

$$f(y) =< \nabla f(x), y - x > + f(x) + R(y - x)$$

where $\lim_{h \to 0} \frac{R(h)}{\|h\|} = 0$, in other words the error that we make is at most quadratic.

Notice that this is a completely local thing, the furthest we get from x the mor distant the function and the model are.

Example 4.0.3. *Let $f : \mathbb{R}^2 \to \mathbb{R}$ such that $f(x, y) = x^2 e^y$. The gradient was computed above and now we have $f(1, 0) = 1$, $\nabla f(1, 0) = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$.*

The linear model has the following shape $L_{(1,0)}(x, y) = 1 + < \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \begin{pmatrix} x - 1 \\ y - 0 \end{pmatrix} >=$ $-1 + 2x + y$.

And the quadratic model is

$$\begin{aligned} Q_{[1,0]}(x, y) &= -1 + 2x + y + \frac{1}{2} \cdot \begin{pmatrix} x - 1 & y - 0 \end{pmatrix} \cdot \begin{pmatrix} 2 & 2 \\ 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} x - 1 \\ y - 0 \end{pmatrix} \\ &= -1 + 2x + y + \frac{1}{2} \cdot \begin{pmatrix} 2x - 2 + 2y, & 2x - 2 + y \end{pmatrix} \cdot \begin{pmatrix} x - 1 \\ y - 0 \end{pmatrix} \\ &= -1 + 2x + y + \frac{1}{2} \cdot (2x - 2 + 2y) \cdot (x - 1) + (2x - x + y) \cdot y \end{aligned} \tag{4.0.1}$$

Definition 4.0.7 (Second order Taylor model)**.** *Let $f : \mathbb{R}^n \to \mathbb{R} \in C^1$ in some $\mathcal{B}(x, \delta)$, we define the* ***second-order Taylor's formula***

$$f(y) = L_x(y) + \frac{1}{2}(y-x)^T \nabla^2 f(x)(y-x) + R(y-x)$$

with $\lim_{h\to 0} R(h) \|h\|^2 = 0 \equiv$ the error is $O(\|y-x\|^3) \equiv$ the remainder vanishes at least cubically.

Notice that the k-th order Taylor expansion with k-th order derivatives, but $\nabla^k f(x)$ tensor of order $k \equiv n^k$ numbers. Often $k > 2$ is unfeasible, so this approach cannot be followed.

Fact 4.0.3. *Let $f : \mathbb{R}^n \to \mathbb{R} \in C^1$. If f Lipschitz continuous on S, then $\sup\{\|\nabla f(x)\| \ : \ x \in S\} \leq L$ ($\iff$, and $= L$ if S convex).*

Moreover, we can prove

Fact 4.0.4. *Let $f : \mathbb{R}^n \to \mathbb{R} \in C^1$. f is Lipschitz continuous on S iff $\nabla^2 f$ is bounded on S.*

Equivalently, iff $\lambda_1(\nabla^2 f)$ is bounded.

Fact 4.0.5. *Let $f : \mathbb{R}^n \to \mathbb{R} \in C^1$. If ∇f is Lipschitz continuous, then $f(y) \leq L_x(y) + \frac{L}{2}\|y-x\|^2$.*

The approximations are good because they give an hint about where the function is decreasing. The limitation is that we cannot take too be steps, because the derivative information is accurate only locally.

Moreover, we need to take into account the fact that if the model is too simple (but efficient) the approximation is not very good; on the other hand, complex and accurate approximations are computationally heavy.

4.1 Simple functions

4.1.1 Linear functions

In this scenario, f has the following shape: $f(x) = cx$, for a fixed $c \in \mathbb{R}^n$.

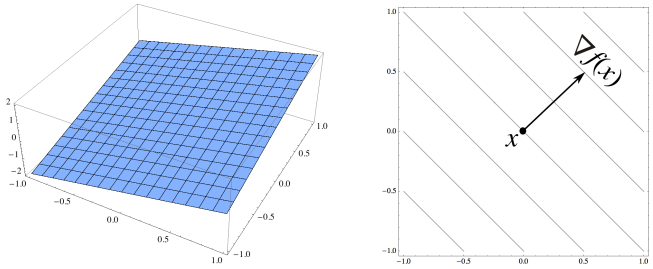
It holds that $\nabla f(x) = c$, $\nabla^2 f(x) = 0$ and that level sets are parallel hyperplanes orthogonal to c ($= [\,1\,,\,1\,]$ here)

4.1.2 Quadratic functions

Let us move to more interesting objects, aka quadratic functions: $f(x) = \frac{1}{2}x^T Q x + qx$, where $Q \in \mathbb{R}^{n\times n}$, $q \in \mathbb{R}^n$.

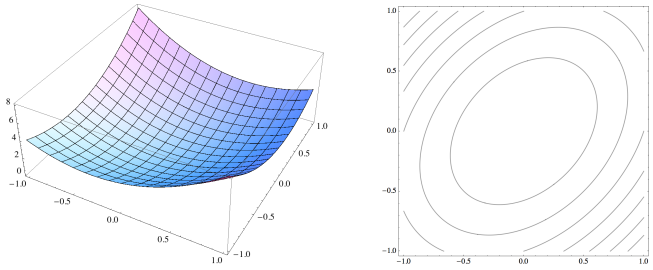
In Figure 4.2 we can see the plot of the quadratic function where

$$Q = \begin{bmatrix} 6 & -2 \\ -2 & 6 \end{bmatrix} q = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



(a) Linear function (b) Level sets

FIGURE 4.1: Graphical example of linear function.



(a) Quadratic function (b) Level sets

FIGURE 4.2: Graphical example of a quadratic function.

In quadratic functions the gradient is a linear function on x ($\nabla f(x) = Qx+q$), while the Hessian is just Q ($\nabla^2 f(x) = Q$) and the level sets are ellipsoids. Sometimes such ellipses can degenerate to lines, in the case that one of the axis has become $+\infty$.

Fact 4.1.1. *We can always assume that if Q is symmetric, then it has spectral decomposition.*

Proof. $x^T Qx = \frac{[(x^T Qx)+(x^T Qx)^T]}{2} = x^T[\frac{(Q+Q^T)}{2}]x = H\Lambda H^T$ □

We will almost always assume that Q is symmetric.

Let us consider the "easy case", where Q is non singular (aka $\lambda_i \neq 0 \forall i$).

In this case, $\bar{x} = -Q^{-1}q$, where $\bar{x}$ is called the center of the ellipsoid. Let us assume that we moved the origin in such $\bar{x}$, so $y = x - \bar{x}$ and $f_{\bar{x}}(y) = \frac{1}{2}y^T Qy$.

Fact 4.1.2. *Along H_i: $f(\alpha) = f_{\bar{x}}(\alpha H_i) = \alpha^2 \lambda_i$*

Moreover, the size of the axes of the level curves is proportional to $\sqrt{1/\lambda_i}$. If the eigenvalues are 0 then the axes get very long (but we would be in the case of Q singular), on the other hand, if $\lambda_i < 0$, we no longer have axes.

Fact 4.1.3. *We can state the following:*

- $\forall i\, \lambda_i > 0 \equiv Q \succ 0 \Longrightarrow \bar{x}$ *minimum of* f*;*
- $\exists i \lambda_i < 0 \Longrightarrow f$ *unbounded below.*

4.2 5th of October 2018 — A. Frangioni

4.2.1 Unconstrained optimization

Until now we stated that the best conditions are encountered when the domain is a compact set and we have many derivatives.

Now we need to consider when we can stop our algorithm.

Definition 4.2.1 (Unconstrained optimization problem)**.** *We want to solve* (P) $f_* = \min\{f(x) \ : \ x \in X\}$*, where* $X = \mathbb{R}^n$*.*

If $\mathbb{R}^n$ is not bounded, Weierstrass theorem does not apply, hence even if a (global) minimum x_* exists, finding it is a NP problem.

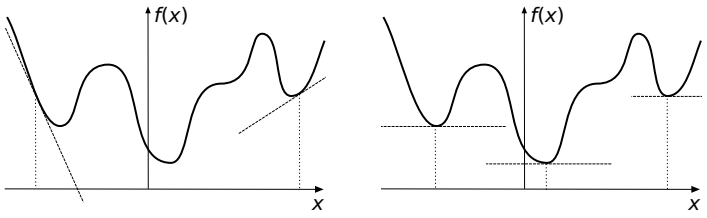
Let us use a weaker condition to ease things a little: x_* is a **local minimum** if it solves

$$\min\{f(x) \ : \ x \in \mathcal{B}(x_*, \varepsilon)\} \text{for some } \varepsilon > 0$$

aka, the minimum we found is a global minimum in a ball around x^*.

Also, x^* is a **strict local minimum** if $f(x) < f(y) \ \ \forall\, y \in \mathcal{B}(\, x_* \,,\, \varepsilon\,)$

To test these conditions derivatives help, as an example see Figure 4.3



(a) Not minima (b) Minima

FIGURE 4.3: In the leftmost plot, we can see that if the derivatives are non zero the point is not a minimum. Such a condition is satisfied in the right handed plot.

If $f'(x) < 0$ or $f'(x) > 0$, x clearly cannot be a local minimum.

Hence, $f'(x) = 0$ in all local minima, so this holds in the global one as well.

First order model

Do you recall?

The first order model of f is $L_x(y) = f(x) + \nabla f(x)(y - x)$, such that $f(y) = f(x) + \nabla f(x)(y - x) + R(y - x)$.
We already stated last lecture that if the norm of the argument of the residual is going to 0, then the residual is going to 0 faster (quadratically), formally $\lim\limits_{\|h\| \to 0} \frac{R(h)}{\|h\|} = 0$.

Fact 4.2.1. *Let f be differentiable, if x is a local minimum, then $\nabla f(x) = 0$.*

In which direction shall we move in order to get closer to the minimum, provided that we are sitting in x? $x(\alpha) = x - \alpha\nabla f(x)$, hence we should take a step along the anti-gradient $-\nabla f(x)$.

Proof by contraddiction. Let us assume that x is a local minimum but $\nabla f(x) \neq 0$.

In our case, $y = x - \alpha\nabla f(x)$, so we get $f(x - \alpha\nabla f(x)) = f(x) - \alpha \left\|\nabla f(x)\right\|^2 + R(-\alpha\nabla f(x))$.

Hence, in our case the direction is fixed, but we can choose the step size α, so it can be proved that $\lim_{\alpha\to 0} \frac{R(-\alpha\nabla f(x))}{\|\alpha\nabla f(x)\|} = 0$, that is equivalent by definition to $\forall\varepsilon > 0\, \exists\bar{\alpha} > 0$ s.t. $\frac{R(-\alpha\nabla f(x))}{\alpha\|\nabla f(x)\|} \leq \varepsilon\ \forall\ 0 \leq \alpha < \bar{\alpha}$.

Take $\varepsilon < \|nabla f(x)\|$ to get $R(-\alpha\nabla f(x)) < \alpha \left\|\nabla f(x)\right\|^2$, then

$$f(x(\alpha)) = f(x) - \alpha \left\|\nabla f(x)\right\|^2 + R(-\alpha\nabla f(x)) < f(x)$$

$\forall\alpha < \bar{\alpha}$ x cannot be a local minimum. □

Notice that the optimality condition also tells us how to move to get closer to the minimum.

An attentive reader may notice that the gradient is 0 in minima, maxima and saddle points (aka stationary point), hence how to discriminate among those?

We need to take infot account second derivatives, namely such second derivative should be positive for a minimum point.

Second order model

Fact 4.2.2. *Let $f \in C^2$. If x is a local minimum then $\nabla^2 f(x) \succeq 0$, in words the gradient is positive semidefinite.*

Proof by contraddiction. Our contraddictory hypothesis is that we are in a local minimum, but the Hessian is not positive semidefinite (formally, $\exists d$ s.t. $d^T\nabla^2 f(x)d < 0$ or equivalently, $\exists\lambda_i < 0$, noticing that $\bar{f}(\alpha) = tr(\alpha H_i)\nabla f(x)(\alpha H_i) = \alpha^2\lambda_i < 0$).

Obs: saying that Hessian is not positive semidefinite means saying that there is a direction of negative curvature.

Just like in previous case, we take the direction d normalized ($\|d\| = 1$).

Let us consider a step $x(\alpha) = x + \alpha d$ and then take the second-order Taylor formula (since $\nabla f(x) = 0$ there is no linear term involved)

$$f(x(\alpha)) = f(x) + \frac{1}{2}\alpha^2 d^T\nabla^2 f(x)d + R(\alpha d)$$

with $\lim_{\|h\|\to 0} \frac{R(h)}{\|h\|^2} = 0$, which means that the residual should go to 0 at least cubically.

Since $h = x - x(\alpha)$ we get that $\lim_{\alpha \to 0} \frac{R(\alpha d)}{\alpha^2} = 0$ or equivalently $\forall \varepsilon > 0 \, \exists \bar{\alpha} > 0$ s.t. $R(\alpha d) \leq \varepsilon \alpha^2 \forall 0 \leq \alpha < \bar{\alpha}$.

At this point, since this condition holds for each ϵ we are allowed to take the most convenient: $\varepsilon < -\frac{1}{2} d^T \nabla^2 f(x) d$, so that we obtain this condition on the residual $R(\alpha d) < -\frac{1}{2} \alpha^2 d^T \nabla^2 f(x) d$, hence

$$f(x(\alpha)) = f(x) + \frac{1}{2} \alpha^2 d^T \nabla^2 f(x) d + R(\alpha d) < f(x) \forall \, 0 \leq \alpha < \bar{\alpha}$$

Hence x cannot be a local minimum. □

In a local minimum, there cannot be directions of negative curvature "when the first derivative is 0, second-order effects prevail".

As far as sufficient conditions are concerned, we can prove the following

Fact 4.2.3. *Let $f \in C^2$ and let the Hessian be symmetric (hence real eigenvalues). If $\nabla f(x) = 0$ and the Hessian is strictly positive definite ($\nabla^2 f(x) \succ 0$) then x is a local minimum.*

Proof. Since the gradient is 0 we get the following second order Taylor approximation

$$f(x + d) = f(x) + \frac{1}{2} d^T \nabla^2 f(x) d + R(d) \text{ with } \lim_{h \to 0} \frac{R(d)}{\|d\|^2} = 0$$

Hence, by definition of limit $\forall \varepsilon > 0 \; \exists \; \delta > 0$ s.t. $R(d) \leq \varepsilon \|d\|^2 \;\; \forall d$ s.t. $\|d\| < \delta$.

Since the Hessian is strictly positive definite $\lambda_{\min} > 0$ minimum eigenvalue of $\nabla^2 f(x)$, hence the variational caracterization of eigenvalues $d^T \nabla^2 f(x) d \geq \lambda_{\min} \|d\|^2$.

We are now ready to pick the ε we prefer ($\varepsilon < \lambda_{\min}$) to get $\forall d$ s.t. $\|d\| < \delta$

$$f(x + d) = f(x) + \frac{1}{2} d^T \nabla^2 f(x) d + R(d) \geq f(x) + (\lambda_{\min} - \varepsilon) \|d\|^2 > f(x)$$

The term $\lambda_{\min} - \varepsilon$ is strictly positive □

In the remaining part of this lecture we will look for conditions that ensure that one a local minimum is found, it is also a global minimum.

Until now, we said that the local minima are those points where the fradient is 0 and the Hessian is positive semidefinite. An easy way to ensure that the Hessian is positive semidefinite in a ball around x is to have that the Hessian is positive semidefinite everywhere ($\forall x \in \mathbb{R}^n$) aka f is a convex function.

4.2.2 Convexity

Let us introduce some preliminaries to the hypothesis of convex functions.

Definition 4.2.2 (Convex hull)**.** *Let $x, y \in \mathbb{R}^n$ we term **convex hull** and denote $conv(x, y) = \{z = \alpha x + (1 - \alpha) y \;\; : \;\; \alpha \in [0, 1]\}$ the segment joining x and y.*

Definition 4.2.3 (Convex set)**.** *We term* ***convex set*** *if for each couple in the set, the line linksing such points belongs to the set.*

Formally, $C \subset \mathbb{R}^n$ *is a* ***convex set*** *if* $\forall\ x, y \in C\ conv(x, y) \subseteq C$*.*

Notice that "disconnected sets" cannot be convex sets.

Definition 4.2.4 (Convex hull of a set)**.** *Given a set* S*, we can "complete" it to a convex set:*

$$\begin{aligned} conv(S) &= \bigcup\{\, conv(x, y) \,:\, x, y \in S \,\} \\ &= \bigcap\{\, C \,:\, C \text{ is convex } \wedge C \supseteq S \,\} \end{aligned}$$

Equivalently, the convex hull of S *= iterated convex hull of all* $x, y \in S$ *or the smallest convex set containing* S

Our goal is to find the nicest possible convex set that approximates our set.

Fact 4.2.4. *A convex set is equal to its convex hull, formally* C *is convex* $\iff C = conv(C)$*.*

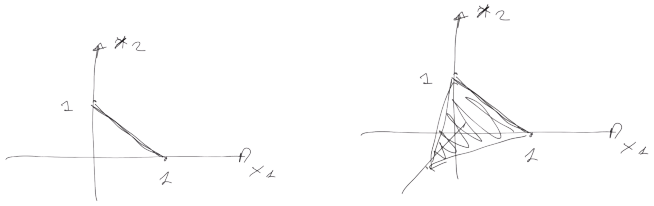
Note

A more general definition of a convex hull is the following: $\text{conv}(\{x_1, \ldots, x_k\}) = \{x = \sum_{i=1}^{k} \alpha_i x_i \ :\ \sum_{i=1}^{k} \alpha_i = 1,\ \alpha_i \geq 0\ \forall i\}$

Definition 4.2.5 (Unitary simplex)**.** *We term* ***unitary simplex*** *the set of* k *non-negative numbers summing to* 1*, formally*

$$\Theta^k = \{\alpha_i \in \mathbb{R}^k \ :\ \sum_{i=1}^{k} \alpha_i = 1, \alpha_i \geq 0\ \forall i\}$$

A few graphical examples are displayed in Figure 4.4.



(a) In $\mathbb{R}^2$ (b) In $\mathbb{R}^3$

FIGURE 4.4: Unitary simplexes.

We are interested in sufficient conditions for convexity.

Definition 4.2.6 (Cone)**.** *We term* ***cone*** *the set* $\mathcal{C} = \{x \ :\ \alpha x \in \mathcal{C} \forall \alpha \geq 0\}$*.*

An attentive reader may notice that the definition of cone is a relaxation of the unitary simplex, where we do not require the unitary sum.

The following sets are convex:

- Convex polytope $conv(\{x_1, \ldots, x_k\})$, unitary simplex Θ
- Affine hyperplane: $\mathcal{H} := \{x \in \mathbb{R}^n \ : \ ax = b\}$
- Affine subspace: $\mathcal{S} := \{x \in \mathbb{R}^n \ : \ ax \leq b\}$
- Ball in p-norm, $p \geq 1$: $\mathcal{B}_p(x, r) = \{ y \in \mathbb{R}^n \ : \ \| y - x \|_p \leq r \}$
- Ellipsoid: $\mathcal{E}(Q, x, r) := \{y \in \mathbb{R}^n \ : \ (y-x)^T Q (y-x) \leq r\}$ with $Q \succeq 0$. Notice that ellipsoids are levelsets of quadratic functions.
- Open versions by substituting "$<$" to "$\leq$"
- Cones
- Conical hull of a finite set of directions: $cone(\{d_1, \ldots, d_k\}) = \left\{ d = \sum_{i=1}^{k} \mu_i d_i \ : \ \mu_i \geq 0 \ \forall i \right\}$
- Lorentz (ice-cream) cone: $\mathbb{L} = \left\{ x \in \mathbb{R}^n \ : \ x_n \geq \sqrt{\sum_{i=1}^{n-1} x_i^2} \right\}$
- Cone of positive semidefinite matrices: $\mathbb{S}_+ = \{ A \in \mathbb{R}^{n \times n} \ : \ A \succeq 0\}$

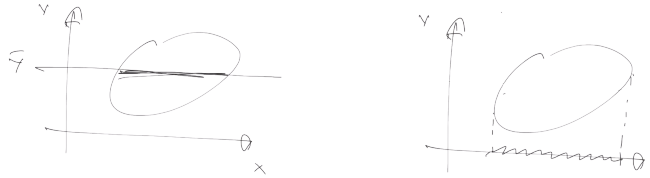
Fact 4.2.5. *The following operations preserve convexity.*

1. *Given a possibly infinite family of convex sets $(\{C_i\}_{i \in I})$, the intersection $(\bigcap_{i \in I} C_i)$ convex;*
2. *If we have convex sets in different subspaces, their cartesian product is a convex set ($C_1, \ldots, C_k$ convex $\iff$ $C_1 \times \cdots \times C_k$ convex);*
3. *Given a convex set, its image under a linear mapping (aka scaling, translation, rotation) is a convex set. Formally, C convex $\implies A(C) := \{x = Ay + b \ : \ y \in C\}$ convex;*
4. *C convex $\implies A^{-1}(C) := \{x \ : \ Ax + b \in C\}$ convex (inverse image under a linear mapping);*
5. *Let C_1 and C_2 convex and let $\alpha_1, \alpha_2 \in \mathbb{R}$. Then $\alpha_1 C_1 + \alpha_2 C_2 := \{x = \alpha_1 x_1 + \alpha_2 x_2 \ : \ x_1 \in C_1, x_2 \in C_2\}$ convex;*
6. *$C \subseteq \mathbb{R}^n = \mathbb{R}^{n_1} \times \mathbb{R}^{n_2}$ convex $\implies$*

   Slice: *$C(y) := \{x \in \mathbb{R}^{n_1} \ : \ (x, y) \in C\}$ conve;*

   Projection: *$C^1 := \{x \in \mathbb{R}^{n_1} \ : \ \exists y \text{ s.t. } (x, y) \in C\}$ convex*

   A pictorial example in Figure 4.5;



(a) Slice on $\bar{y}$ (b) Project on the x component

FIGURE 4.5: Pictorial examples of slicing and projecting.

7. *C convex $\Longrightarrow$ $int(C)$ and $cl(C)$ convex*

Theorem 4.2.6. *$\mathcal{P}$ is a polyhedron iff $\exists\{x_1, \ldots, x_k\}$ and $\{d_1, \ldots, d_h\}$ s.t. $\mathcal{P} = conv(\{x_1, \ldots, x_k\}) + cone(\{d_1, \ldots, d_h\})$.*

Notice that if we are interested in proving that a set with a certain shape is convex, we chould try to derive it from an object that we know is convex through the operations we enumerated above.

Definition 4.2.7 (Convex function)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a function. We say that f is **convex** if $\forall x, y \in \mathbb{R}^n$, the segment that joins $f(x)$ and $f(y)$ lies above the function.*

*In other words, f is **convex** iff $epi(f)$ is convex, where epi denotes the epigraph of the function, graphically speaking, the region which is above the function line (in the plot).*

*Equivalently, we say that f is **convex** if $\forall x, y \in dom(f)$ for any $\alpha \in [0, 1]$, $\alpha f(x) + (1 - \alpha)f(y) \geq f(\alpha x + (1 - \alpha)y)$.*

Equivalently, $\forall x^1, \ldots, x^k$, $\alpha \in \Theta^k$

$$f\left(\sum_{i=1}^{k} \alpha_i x^i\right) \leq \sum_{i=1}^{k} \alpha_i f(x^i)$$

Definition 4.2.8 (Sublevel graph)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a function. We term **sublevel graph** of $f(x)$ the projection on the x axis of the portions of the epigraph which lie below the constant $y = \bar{x}$.*

Fact 4.2.7. *The following holds:*

- *Let f convex. Then $S(f, v)$ convex $\forall v \in \mathbb{R}$;*
- *f is concave if $-f$ is convex ("convex analysis is a one-sided world").*

The second statement of Proposition 4.2.7 is useful to make a comparison between minimizing and maximizing. In particular, if our aim is to maximize the function, we can be sure to have found a global maximum if the function is concave.

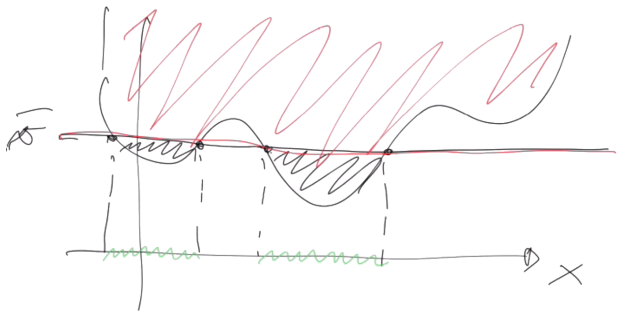


FIGURE 4.6: Pictorial example of sublevel graph. Such a graph is drawn in green in the figure.

Definition 4.2.9 (Strict convexity)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}^m$. We term f* ***strictly convex*** *iff $\alpha f(x) + (1-\alpha)f(y) > f(\alpha x + (1-\alpha)y)$.*

Definition 4.2.10 (Strong convexity)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}^m$. We term f* ***strongly convex modulus*** *$\tau > 0$ iff $f(x) - \frac{\tau}{2}\|x\|^2$ is convex.*

Formally,

$$\alpha f(x) + (1-\alpha)f(y) \geq f(\alpha x + (1-\alpha)y) + \frac{\tau}{2}\alpha(1-\alpha)\|(y-x)\|^2$$

Next lecture we will talk about how we can check tat a function is convex, operationally.

4.3 11th of October 2018 — A. Frangioni

Last lecture we left with the task of understanding how to check if a function is convex.

As we already stated for convex sets, we can prove that a function is convex deriving from convex functions, though "convex friendly" operations.

Note

There is a softare called CVX, designed to model convex object. A pretty easy way to check if an object is convex is to try to write it in CVX. If such an operation is possible, then the object is convex.

The following functions are convex:

1. $f(x) = wx$: linear functions are both convex and concave;
2. $f(x) = \frac{1}{2}xQx + qx$ if convex iff $Q \succeq 0$;
3. $f(x) = e^{ax}$ for any $a \in \mathbb{R}$ and $x \in \mathbb{R}$
4. $f(x) = -\log(x)$ for $x > 0$
5. $f(x) = x^a$ for $a \geq 1$ or $a \leq 0$ on $x \geq 0$;
6. $f(x) = \|x\|_p$ for $p \geq 1$;
7. $f(x) = \max\{x_1, \ldots, x_n\}$;
8. for any convex set C, its indicator function

$$\imath_C(x) = \begin{cases} 0 & \text{if } x \in C \\ +\infty & \text{if } x \notin C \end{cases} \quad (\text{l.s.c.} \iff C \text{ closed})$$

9. $A \in \mathbb{R}^{n \times n}$ symmetric, eigenvalues customarily ordered $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_n$: $f_m(A) = \sum_{i=1}^{m} \lambda_i$ (sum of m largest eigenvalues)

Fact 4.3.1. *The following operations preserve convexity:*

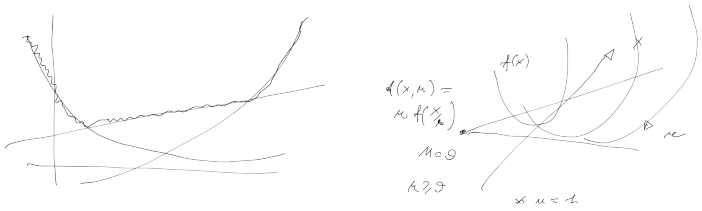
1. *f, g convex, α, $\beta \in \mathbb{R}_+ \implies \alpha f + \beta g$ convex (non-negative combination);*
2. *$\{f_i\}_{i \in I}$ (infinitely many) convex functions $\implies f(x) = \sup_{i \in I} f_i(x)$ convex, see Figure 4.7(a);*
3. *Pre-composition with linear function is convex, formally f convex $\implies f(Ax + b)$ convex;*
4. *Post-composition with increasing convex function is convex. Formally, $f : \mathbb{R}^n \to \mathbb{R}$ convex, $g : \mathbb{R} \to \mathbb{R}$ convex increasing $\implies g \circ f = g(f(x))$ is convex;*

5. *f_1, f_2 convex $\Longrightarrow$ $f(x) = \inf\{ f_1(x_1) + f_2(x_2) \ : \ x_1 + x_2 = x\}$ convex (infimal convolution);*

6. *g convex $\Longrightarrow$ $f(x) = \inf\{g(y) \ : \ Ay = x\}$ convex (image under a linear mapping, aka value function of convex constrained problem);*

7. *$g(x, y) : \mathbb{R}^{n+m} \to \mathbb{R}$ convex $\Longrightarrow$ $f(x) = \inf\{g(x, y) \ : \ y \in \mathbb{R}^m\}$ convex (partial minimization);*

8. *$f(x)$ convex $\Longrightarrow$ $\tilde{f}(x, u) = uf(x/u)$ when $u > 0$, $\tilde{f}(x, u) = \infty$ otherwise, convex (perspective or dilation function of f), see Figure 4.7(b).*



(a) Suprema — second property

(b) Perspective — eightth property

FIGURE 4.7: Graphic hints.

Fact 4.3.2. *Let $f : \mathbb{R}^n \to \mathbb{R} \cup \infty$ a convex function. If $\exists \bar{x} \in dom(f)$ such that $f(\bar{x}) = -\infty$, then $f \equiv -\infty$.*

From now on we will solve the issue of functions with non convex domain, sayting that in those points where the function is not defined, we value it $+\infty$.

Fact 4.3.3. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then f is Lipschitz continuous $\forall$ bounded convex set $S \subseteq int(dom(f))$ but on $\partial dom(f)$ (the border of the domain) anything can happen.*

Moreover, a function f, which is continuous but not Lipschitz continuous is not convex.

A couple of examples of Proposition 4.3.3 can be found in Figure 4.8.

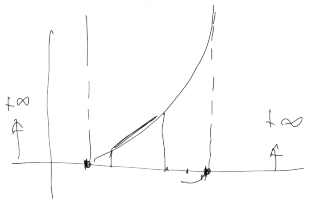
Fact 4.3.4. *Let convex $f : \mathbb{R}^n \to \mathbb{R}$. Then it is Lipschitz continuous on any bounded set and continuous everywhere.*

It happens often that the set of points in which a function is non differentiable have measure 0.

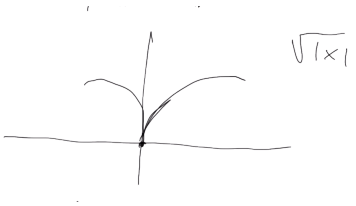
Theorem 4.3.5 (Convexity characterization)**.** *Let $f \in C^1$. It is convex on C convex iff*

$$f(y) \geq f(x) + \nabla f(x)(\, y - x\,) \forall\, x, y \in C$$

In other words, given a point x, we compute the derivative and the value fo the function is above the derivative in that point.



(a) Take a compact set in the interior of the domain (far from the boundaries) there the function is Lipschitz continuous.



(b) If a function is not Lipschitz on a compact subset it is not convex.

FIGURE 4.8

Proof. $\Rightarrow$) $\alpha(f(y) - f(x)) \geq f(\alpha(y-x)+x) - f(x)$, send $\alpha \to 0$

$\Leftarrow$) TODO

□

Theorem 4.3.6. *Let $f \in C^1$ convex. x is a stationary point iff x is a global minimum.*

Fact 4.3.7. *Let f be twice differentiable (aka has Hessian). f is convex iff the Hessian is positive semidefinite.*

Formally, let $f \in C^2$. f is convex on the open set S iff $\nabla^2 f(x) \succeq 0 \ \forall x \in S$.

This proposition gives us an algorithm to check if a function is convex or not: we only need to compute the eigenvalues of the Hessian and check if they are positive.

There are some functions which do not have differentiability property.

A way to work with functions which are not defined on all $\mathbb{R}$ is to solve the following problem:

$$(P) \equiv \inf\{f_X(x) = f(x) + ß_X(x) \ : \ x \in \mathbb{R}^n\}$$

thanks to

Theorem 4.3.8 (Essential objective)**.** *x_* optimal for (P) $\iff$ x_* local minimum of f_X.*

Subgradients and subdifferentials

Definition 4.3.1 (Subgradient)**.** *For each $s \in \mathbb{R}$ we term **s-subgradient** of f at x as*

$$f(y) \geq f(x) + s(y-x) \qquad \forall\, y \in \mathbb{R}^n$$

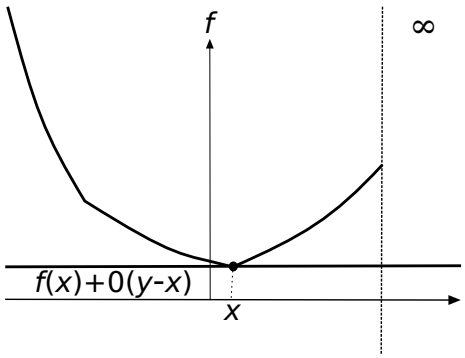


FIGURE 4.9: Pictorial example of subgradients of a non diffrentaible function.

Let us assume that the minimum of the non differentiable function resides in one of its kiky points, then for $s = 0$ we have a subgradient which is flat and this is a sufficient condition for minimality, for a pictorial example see Figure 4.10.

The issue here is that it is unfeasible to check if the subgradient with $s = 0$ is a subgradient for f.

Definition 4.3.2 (Subdifferential)**.** *We term* ***subdifferential*** *the set of all possible subgradients.*

Formally,

$$\partial f(x) := \{s \in \mathbb{R}^n \ : \ s \text{ is a subgradient at } x\}$$

Theorem 4.3.9. *x global minimum $\iff 0 \in \partial f(x)$.*

Notice that in general, when we are not in proximity of a border (where f is unbounded above) we get that the subdifferential is a compact interval.

Formally,$\partial f(x)$ closed and convex, compact $\forall\, x \in int\, dom(f)$.

Moreover, we can prove the following

Fact 4.3.10. *$\partial f(x) = \{\nabla f(x)\} \iff f$ differentiable at x.*

An attentive reader may have noticed that in the case of non differentiable functions it is not possible to derive the directional derivative from the gradient ($\frac{\partial f}{\partial d}(x) =< \nabla f(x), d >$), but we can prove the following

Fact 4.3.11. *$\frac{\partial f}{\partial d}(x) = \sup\{< s, d > : \ s \in \partial f(x)\} \Longrightarrow d$ is a descent direction $\iff < s, d >< 0 \ \forall \ s \in \partial f(x)$.*

As in the differentiable case, we are interested in moving in the steepest descent direction, formally $s_* = -\text{argmin}\{\|s\| \ : \ s \in \partial f(x)\}$.

Example 4.3.1. *Le us assume we are in x and we want to move towards x^* knowing only the subdifferentials. $(-)\partial f(x)$ is convex and compact and All $(-)g \in \partial f(x)$ "point towards x_*": $< g, x - x_* >< 0$.*

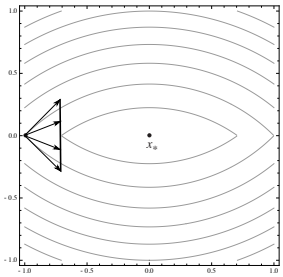


FIGURE 4.10: There are many different subgradients in x. We pick the one with minimum norm among the ones which have a negative scalar product with $x - x^*$.

Not all of them are descent directions, but the (opposite to the) minimum-norm one is a descent direction.

Notice that in $\mathbb{R}^2$ if we take a function and compute the subdifferential, we can scale both the function and the subdifferential by any positive constant (negative contants would lead to concave functions). Formally,

Fact 4.3.12. *Let $f, g : \mathbb{R}^n \to \mathbb{R}$ and take $\alpha, \beta \in \mathbb{R}_+$, then $\partial[\alpha f + \beta g](x) = \alpha \partial f(x) + \beta \partial g(x)$.*

Fact 4.3.13 (Chain rule)**.**
- *Let $f : \mathbb{R}^n \to \mathbb{R}$ and let $A \in M(n, \mathbb{R})$ and $b \in R^n$ then $\partial[f(Ax + b)] = A^T[\partial f](Ax + b)$;*
- *$g : \mathbb{R} \to \mathbb{R}$ increasing, then $\partial[g(f(x))] = [\partial g](f(x))[\partial f](x)$.*

Definition 4.3.3 (ε-subgradient)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$. s is ε-**subgradient** at x if*

$$f(y) \geq f(x) + s(y - x) - \varepsilon \ \forall y \in \mathbb{R}^n$$

support hyperplane passing ε below epi(f)

Fact 4.3.14. *Given a point x, the value of the finction in x cannot be further from ε the minimum value fo the function. Formally, $0 \in \partial_\varepsilon f(x) \iff x$ is ε-optimal.*

We are now allowed to compute $s_* = \text{argmin}\{\|s\| \ : \ s \in \partial_\varepsilon f(x)\}$.

If $s_* = 0$ then x is ε optimal. Otherwise, $\exists \alpha > 0$ s.t. $f(x - \alpha s_*) \leq f(x) - \varepsilon$ ($-s_*$ is of ε-descent).

The ε-subgradient is very powerful, but the issue is that is even more expensive to compute than the subgradient.

4.4 17th of October 2018 — A. Frangioni

4.4.1 Optimization algorithms

Do you recall?

We are interested in finding the minimum of a function through an iterative procedure, such that we start form an initial guess x_0 and go on $x^i \rightsquigarrow x^{i+1}$. We want to move towards the otimum.

How to be sure we are in an optimum?

- (strong) $\{x^i\} \to x_*$: the whole sequence converges to an optimal solution;
- (weaker) all accumulation points of $\{x^i\}$ are optimal solutions;
- (weakest) at least one accumulation point of $\{x^i\}$ is optimal.

Such iterative process, can be held in two different forms:

LINE SEARCH**:** first choose $d^i \in \mathbb{R}^n$ (direction), then choose $\alpha^i \in \mathbb{R}$ (that we term stepsize or equivalently "learning rate") s.t. $x^{i+1} \leftarrow x^i + \alpha^i d^i$;

TRUST REGION**:** first choose α^i (trust radius), then choose d^i.

In both these alternatives, it is crucial to choose a proper model to approximate function f.

Gradient method for quadratic functions

The simplest model we can build is a linear one, namely $L^i(x) = L_{x^i}(x) = f(x^i) + \nabla f(x^i)(x - x^i)$ and find the direction according to the model.

We should not move too far from x^i, so we want $\alpha_i \to 0$ and then $d_i = argmin\{\lim_{t\to 0} \frac{f(x+td)}{t}\} = -\nabla f(x^i)$, aka the steepest descent direction.

Notice that a too short step is bad either, because the gain in the value of the function is very little.

At each step we want $\alpha^i \in \text{argmin}\{f(x^i + \alpha d^i) \ : \ \alpha \geq 0\}$.

Linear functions are unbounded below, so we would like to use a different family of functions. The easiest family of functions which are still more complex than linear ones are quadratic functions.

$$f(x) = \frac{1}{2} x^T Q x + qx$$

where $Q \succeq 0$ otherwise f is unbounded below.

The minimum here is the point in which the gradient ($\nabla f(x) = Qx + q$) is 0.

ALGORITHM 4.4.1 Pseudocode for quadratic functions local minimum detection.

```
1: procedure SDQ(f, x, ε)
2:     while (‖∇f(x)‖ > ε) do
3:         d ← −∇f(x);
4:         α ← ‖d‖² / (dᵀQd);
5:         x ← x + αd;
6:     end while
7: end procedure
```

For the time being we do not go into detail of how to chose the ε such that we can stop when the norm of the gradient is smallert than such a constant.

Let us see how to obtain the formula for α.

We are interested in computing the minimum of $\{f(x^i + \alpha d^i) : \alpha \geq 0\}$.

Let us do come algebra to describe better such an f:

$$\begin{aligned} f(x^i + \alpha d^i) &= \frac{1}{2}(x^i + \alpha d^i)Q(x^i + \alpha d^i) + q(x^i + \alpha d^i) \\ &= \frac{1}{2}x^{iT}\cancel{Qx^i} + (x^i)^T Q(\alpha d^i) + \frac{1}{2}(d^i)^T Q d^i + \cancel{qx^i} + \alpha(qd^i) \\ &= [\frac{1}{2}(d^i)^T Q d^i]\alpha^2 + \alpha[(x^i Q + q)d^i] \\ &\overset{(1)}{=} [\frac{1}{2}(d^i)^T Q d^i]\alpha - \left\|d^i\right\| \end{aligned} \tag{4.4.1}$$

What if $(d^i)^T Q d^i = 0$? If Q is strictly positive definite this cannot happen, so the algorithm never breaks down.

Can we prove that the sequence of iterates is moving towards the optimum?

For this proof let us assume that $\varepsilon = 0$, hence the procedure will neve stop. We want to prove that the sequence $\{x^i\}$ is (or contains) a minimizing sequence.

First of all we can state that the sequence is monotone, so it has a limit for sure.

What we can prove is that the the point where the sequence is converging is a stationary point. $\lim_{i\to\infty} < \nabla f(x^i), \nabla f(x^{i+1}) >= 0 \overset{(*)}{=} < \nabla f(x), \nabla f(x) >$ and this holds because of the fundamental relationship $< \nabla f(x^i), \nabla f(x^{i+1}) >= 0$.

Notice that $\overset{(*)}{=}$ follows from the fact that the gradient is continuous.

How fast is the convergence?

In general, showing how fast $\| x^i - x_* \|$ decreases is more involved that showing how fast $f(x^i) - f_*$ decreases, but we do not know f_*.

We concentrate on computing $\lim_{i\to\infty} \frac{f(x^{i+1}) - f_*}{f(x^i) - f_*{}^p} = R$.

According to the values of p and R we get the following alternatives:

SUBLINEAR: $p = 1, R = 1$;

LINEAR: $p = 1, R < 1$;

SUPERLINEAR: $p = 1, R = 0$;

QUADRATIC: $p = 2, R > 0$.

Since the optimum is in $x^* = -Q^{-1}q$, we get that $f(x^*) = \frac{1}{2}q^TQ^{-1}QQ^{-1}q - q^TQ^{-1}q = \frac{1}{2}q^TQ^{-1}q - q^TQ^{-1}q = -\frac{1}{2}q^TQ^{-1}q$.

Let us use a nifty trick: let us define

$$\begin{aligned}
\bar{f}(x) &= \frac{1}{2}(x - x_*)^TQ(x - x_*) \\
&= \frac{1}{2}x^TQx + \frac{1}{2}{x_*}^TQx_* - x^T(Qx) \\
&\overset{(2)}{=} \frac{1}{2}x^TQx + \frac{1}{2}Q^{-1}{q}^TQ(Q^{-1}q) + qx \\
&= \frac{1}{2}x^TQx + \frac{1}{2}q^T\cancel{Q^{-1}Q}Q^{-1}q + qx \\
&= \frac{1}{2}x^TQx + \frac{1}{2}q^TQ^{-1}q + qx \\
&= f(x) - f_*
\end{aligned} \tag{4.4.2}$$

4.5 19th of October 2018 — A. Frangioni

4.5.1 Gradient method for quadratic functions

This is the simplest possible family of functions where a minumum exists.

Do you recall?

A quadratic function is defined as: $f(x) = \frac{1}{2}x^T Qx + qx$, so its gradient is the following $\nabla f(x) = Qx + q$

We are interested in finding local minima of the function f.

Fact 4.5.1. *A quadratic function $f : \mathbb{R}^n \to \mathbb{R}^n$, s.t. $f(x) = \frac{1}{2}x^T Qx + qx$ admits a minimum iff $Q \succeq 0$ (Q is positive semidefinite).*

The gradient method generates points that move along orthognal directions. More formally, $x^{i+1} = x^i + \alpha^i d^i$, where $d^i = -\nabla f(x^i) = -Qx^i - q$

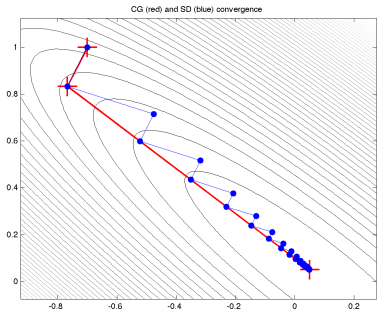


FIGURE 4.11: Some iterations of the gradient method

Fact 4.5.2. $\forall i \ < d^i, d^{i+1} > = 0$.

Proof. TODO □

$i\alpha^i = \frac{\|d^i\|^2}{d^{i\,T} Q d^i}$, $x^i \to \overline{x}$

Our aim is to estimate how fast this converges.

$f(x^i) - f_* = -\frac{1}{2}q^T Q^{-1} q$ we know everything for this function, from last lecture ($x^* = -Q^{-1}q$).

What can we say about $f(x^{i+1}) - f_* = \frac{1}{2}(x^{i+1} - x_*)^T A(x^{i+1} - x_*)$?

I want it in terms of $x^{i+1} = x^i - \frac{\|Qx^i - q\|^2}{(Qx^i - q)^T Q(Qx^i - q)}$

From this fomula we can say that the error at the current step $(i+1)$ is equal to the error of the step before (i) divided by something. More precisely, $f_*(x) = \frac{1}{2}(x - x_*)^T Q(x - x_*) = f(x) + \frac{1}{2}x_*{}^T Q x_* = f(x) - f_*$

If the quantity after the minus is positive but less than 1 the error at the next step will be less than the error at the previous step. This means not only **linear convergence**, but a bit more, because linear convergence only takes into consideration steps in proximity to the limit, while this formula holds also at the beginning.

Fact 4.5.3. *If Q is positive semidefinite $d^TQd = \|d\|_Q^2$*

Fact 4.5.4. *If Q is positive definite we can say that the error goes like $1 - \left(\frac{\|d_i\|^2}{(d^{iT}Qd^i)(d^{iT}Q^{-1}d^i)}\right)$ or, equivalently, $1 - \frac{\|d_i\|_I{}^2}{\|d_i\|_Q{}^2} \cdot \frac{\|d_i\|_I{}^2}{\|d_i\|_{Q^{-1}}{}^2}$.*

We are measuring a vector with three different norms.

We would like to estimate $\frac{<d_i,d_i>}{d_i{}^TQd_i}$.

Given λ_i the eigenvalues of Q, $\frac{1}{\lambda_i}$ are the eigenvalues of the matrix Q^{-1}.

We can say that $\lambda^n\|x\|^2 \leq x^TQx \leq \lambda^1\|x\|^2$, where λ^n is the smallest eigenvalue, while λ^1 is the biggest.

We are looking for a close formula for calculating the convergence rate, since it depends reursivley by the steps done. So we want to do a worst case analysis in order to find a faster way to calculate convergence rate.

We want to prove that R is smaller than 1 so we are looking for an upperbound. A coarse upperbound is $(1 - \frac{\lambda^n}{\lambda^1})$, but we can prove more:

$$\forall\, x \in \mathbb{R}^n \quad \frac{\|x\|^4}{(x^TQx)(x^TQ^{-1}x)} \geq \frac{4\lambda^1\lambda^n}{(\lambda^1 + \lambda^n)^2}$$

We won't see the proof of this fact.

R close to 0 means that the algorithm is converging fast, so when the larger eigenvalue (λ^1) and the smallest eigenvalue (λ^n) are very close to eachother the algorithm is converging fast. We can say that, since the eigenvalues are the axis of the ellipsoid, the algorithm is converging fast when the ellipsoid has a round shape.

We can provide an even better estimate, which is $\left(\frac{\lambda^1-\lambda^n}{\lambda^1+\lambda^n}\right)^2$

How can we understand if the number of iterations needed to converge is a good result? Well, it depends on how that number is obtained. If it's dimensional independent it's very good, because it scales well (when the size of the space — number of variables — increases). It only depends on the conditioning of the matrix Q.

Of course as n grows Q changes, so in practice it may happen that the conditioning of the problem is worsening as n grows.

If the balls are very rounded the zig zags needed to start converging are very few.

Obs: We found a bound for the convergence speed of the algorithm when Q is positive definite. What can we say when Q is positive semidefinite? The algorithm works, but we can't provide an upperbound for the convergence rate. We are even more restrictive when dealing with machine precision, since if there

is an eigenvalue which is bigger than zero, but very close to zero, becomes 0 on the machine, so we can't give an upperbound.

We will see how to deal with this case.

4.5.2 MatLab implementation

Let's talk about the implementation. First of all we need to set a proper value for ϵ. A good idea would be the norm of the gradient. We need to give also some performance bound, like maximum number of iterations or the maximum amount of time.

MatLab call:

```
1  function[x, status] = SDQ(Q, q, x, fStar, eps, MaxIter)
```

where fStar is the optimal value, which should give us an idea of the convergence.

Note

- Always check the coherence of input values (check if the user passed allowed parameters);
- Always give all possible information about your result (for example if the algorithm stopped because the maximum number of iterations was reached or because the epsilon value was reached);
- Always check in your code the accepted values of variables during calculations: for example if you need to divide by a quantity that may be smaller than the precision check it before computing the ratio;
- Design a good log, in order to understand what's happening at each step.

In the code of that function we also kept track of the actual ratio between the error at one step and the error at the next one. This information tells us how many orders of magnitude the error decrease. We can find starting points where the ratio is exaclty R. We can observe that when the conditioning is quite good the error decrease faster than the R limitation, but as soon as we change the values for Q and q things may be worse.

We saw from some examples that if the conditioning grows the number of iterations needed to find the minumum increases as well.

Trying the algorithm on some examples shows that the theoric results are reflected well in the practical case.

Error

When we are running an algorithm, starting from a point that will not lead to the optimum we would like to stop anyway, at a certain point, when we are close

enough to the solution. $\varepsilon_A = f(x^i) - f_* \leq \varepsilon$ (absolute error)

In this context, we introduce the consept of **relative error**, since the error may be big or small if compared with the value of the function.

This relative error is invriant for scaling transformations. Notice that if f^* might be zero the forumla should be changed.

$$\varepsilon_R = (\, f(x^i) - f_* \,) \,/\, |\, f_* \,| = \varepsilon_A \,/\, |\, f_* \,| \leq \varepsilon \qquad \text{(relative error)}$$

The problem is that often we don't know f^*, so it's impossible to compute this kind of error.

In this very common case a good lower bound $\underline{f} \leq f^*$for f^*. In this course we won't focus on finding $\underline{f}$.

Another stopping conditions may be the following:

- $\| \nabla f(x^i) \| \leq \varepsilon$ ("absolute version")
- $\| \nabla f(x^i) \| \,/\, \| \nabla f(x^0) \| \leq \varepsilon$ ("relative version")

The sencond stopping condition is expressed in relation to the first value for the norm of the gradient.

We usually chose the norm of the gradient as a threshold for precision, but we don't know how this quantity relates to ϵ_A or ϵ_B.

Example 4.5.1. *for $X = \mathcal{B}(0, r)$ and f convex, estimate ε_A when $\| \nabla f(x^i) \| \leq \varepsilon$*

I've got a convex function and I know the the minimum is in a ball. We can minimize the linear function in the range of the ball and that minumum is surely a lower bound.

4.6 24th of October 2018 — A. Frangioni

4.6.1 Gradient method for non quadratic functions

We want to move from quadratic functions to wider families of functions.

Do you recall?

The step size α in the quadratic case is defined as follows: $\alpha = \frac{\|d\|^2}{d^T Q d}$

We would like to find a more general form for the step size, which doesn't depend on the fact that the function is quadratic.

The algorithm for finding local minima of non quadratic functions has the same structure of the one used for quadratic ones, i.e. first compute the direction of the step and then compute its size.

We will see that, differently from the quadratic case (where the gradient was $\nabla f(x) = Qx + q$) computing the gradient in this more general case isn't easy at all.

We may recall from last lecture that the proof of the orthogonality of the gradient doesn't depend on the quadratic nature of our functions, so it works in this case too.

How fast does it converge?

Given Algorithm 4.6.1 for finding minima of quadratic functions (that differs from the one provided for quadratic ones for the step size) we would like to understand how fast the convergence is.

ALGORITHM 4.6.1 Pseudocode for non quadratic functions local minimum detection.

```
1: procedure SDQ(f, x, ε)
2:     while (‖∇f(x)‖ > ε) do
3:         d ← −∇f(x);
4:         α ← ‖d‖² / (dᵀQd);
5:         x ← x + αd;
6:     end while
7: end procedure
```

Theorem 4.6.1. *Let f be a function in C^2 and let x_* be a local minimum for f s.t. $\nabla^2 f(x_*) \succ 0$ (which means that the Hessian matrix is strictly positive definite):*

$$\{ x^i \} \to x_* \Longrightarrow \{ f(x^i) \} \to f(x_*)$$

linearly, with same R as the quadratic case, depending on λ_1 and λ_n of $\nabla^2 f(x_*)$

This theorem means that if the function is differentiable and the Hessian is strictly positive definite then when getting closer and closer to the minimum, the function is more and more similar to a quadratic function.

This similarity is a good new, since we can use the same methods of the quadratic case, but, as we recall from the previous lecture, we must pay attention to **conditioning**.

At this point we need to work on finding the local minimum of the one dimensional function φ^i, s.t.

$$\varphi^i(\alpha) = f(x^i + \alpha d^i),$$

where $d^i = -\nabla f(x^i)$.

Let's omit the i, since we are concentrating on a single iteration.

We are interested in finding a α^* such that $\varphi'(\alpha^*) = 0$

Example 4.6.1. *Let's suppose we are in* $\mathbb{R}^2$ *and* $f(x,y) = x^2e^y$. *We can differentiate* F *and* $\nabla f(x,y) = (2xe^y, x^2e^y)$.

At the i-th iteration $(x,y) = (1,0)$, *so* $\nabla f(1,0) = (2,1)$. *Now* $x(\alpha) = (1,0) - \alpha(2,1) = (1-2\alpha, 0-\alpha)$.

At this point we obtain $\varphi(\alpha) = f(x(\alpha)) = (1-2\alpha)^2e^{-\alpha}$.

Chi è $x(\alpha)$?

It's hard to find the roots of this function. The points we can find are not $\varphi'(\alpha) = 0$, but instead $|\varphi'(\alpha) \leq \varepsilon'|$, where the meaning of ε is bounding the directional derivative to be small.

Fact 4.6.2. *Let* $\varphi : \mathbb{R} \to \mathbb{R}$, *such that* $\varphi(\alpha) = f(x^i + \alpha d^i)$, $\varphi'(\alpha) =< \nabla f(x^i + \alpha d^i), d >$.

Proof. TODO: using the **chain rule**: $f : \mathbb{R}^m \to \mathbb{R}^k$, $g : \mathbb{R}^n \to \mathbb{R}^m$, $h : \mathbb{R}^n \to \mathbb{R}^k$ such that $h(x) = f(g(x)) \Rightarrow \mathbf{Jh(x)} = \mathbf{Jf(g(x))} \cdot \mathbf{Jg(x)}$

Obs: $Jf \in \mathbb{R}^{k\times m}$, $Jg \in \mathbb{R}^{m\times n}$ and $Jh \in \mathbb{R}^{k\times m} \cdot \mathbb{R}^{m\times n} = \mathbb{R}^{k\times n}$. □

Fact 4.6.3. *We claim that* $\varepsilon' = \varepsilon$.

Proof. We want to find the relationship between the two parameters ε and ε'. Assuming that we've got a black box that finds α, given ε', we are interested in computing ε'.

Key idea: Normalization of the direction.

We may normalize the direction of movement d^i without perturbing the behaviour of the algorithm: $d^i = -\frac{\nabla f(x^i)}{\|\nabla f(x^i)\|}$. Note that we're not worried of dividing by the norm of the gradient, since if it gets 0 we have already stopped the procedure.

In this new context $\left\|d^i\right\| = 1$ and

$$\begin{aligned}
\varphi'(0) &= \frac{\partial f}{\partial d}(x) \\
\text{(From Proposition 4.6.2)} &=< \nabla f(x), d > \\
&=< \nabla f(x), \frac{-\nabla f(x)}{\|\nabla f(x)\|} > \\
&= -\frac{< \nabla f(x), \nabla f(x) >}{\|\nabla f(x)\|} \\
&= -\frac{\|\nabla f(x)\|^2}{\|\nabla f(x)\|} \\
&= -\left\|\nabla f(x^i)\right\|
\end{aligned} \tag{4.6.1}$$

$\left|\varphi'(\alpha^i)\right| = |< \nabla f(x^{i+1}), d^i >| = \left|< \nabla f(x^{i+1}), -\frac{\nabla f(x^i)}{\|\nabla f(x^i)\|} >\right|$

Hence, if $\{x^i\} \to x$

$$\lim_{i\to\infty}\left|< \frac{\nabla f(x^i)}{\|\nabla f(x^i)\|}, \nabla f(x^{i+1}) >\right| =< \frac{\nabla f(x)}{\|\nabla f(x)\|}, \nabla f(x) >= \|\nabla f(x)\| \leq \varepsilon \tag{4.6.2}$$

Since $\left\|\nabla f(x^i)\right\| > \varepsilon$ we have the thesis. □

Per each phase the new epsilon is obtained $\varepsilon \leftarrow \varepsilon \left\|\nabla f(x^i)\right\|$.

If we can prove that the algorithm is converging we know when to stop.

This convergence isn't the perfect mathematical convergence, since $\varepsilon \neq 0$, because the line search will never terminate.

Exact line search, first orderd approach

We want to find the minimum points of φ, which corresponds to points where the first order derivative is zero and it goes from negative to positive. Since we are talking about numerical algorithms we are going to stop a little before the minimum is reached.

Key idea: We would like to reduce the range in which performing the search, at each step.

How can we be sure that in a given range there is a point where the derivative is 0? Rolle's theorem, as shown in Figure 4.12.

Since the gradient is continuous the directional derivative is continue, so φ is continuous (the scalar product is continuous).

Actually, we only need to find where the derivative is positive, because the 0 of the derivative is between the previous value and this point.

The algorithm is the following:

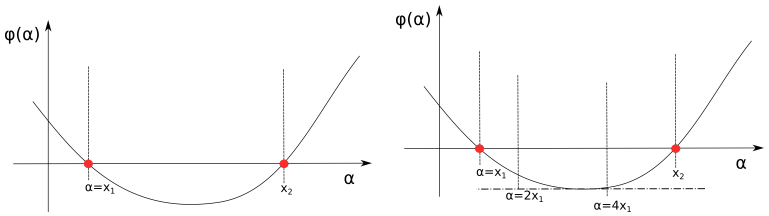


FIGURE 4.12: First, we restrict from $\mathbb{R}$ to $[x_1, x_2]$, then double α until the derivative is greater than 0

ALGORITHM 4.6.2 First algorithm for exact line search

```
1: ᾱ ← x_1; #or whatever value > 0
2: while (φ'(ᾱ) < 0) do
3:     ᾱ ← 2ᾱ; #or whatever factor > 1
4: end while
```

We'll stop when $\alpha < -10^{308}$, which is the smallest value for a double, since we will get a numerical error and stop.

Works if φ **coercive**: $\lim_{\alpha\to\infty} \varphi(\,\alpha\,) = \infty$ (ex. f strongly convex)

Exercise 4.6.1. *Build an example where $\bar{\alpha}$ exists but it is not found by this algorithm.*

Solution: The function changes its derivative in a range between α and 2α.

An alternative to the algorithm presented above may be the bisection algorithm, which follows.

ALGORITHM 4.6.3 Bisection algorithm

```
1: procedure LSBM((φ'ᾱε))
2:     α_- ← 0;
3:     α ← α_+;
4:     α_+ ← ᾱ;
5:     while (|φ'(α)| > ε) do
6:         α ← (α_+ + α_-)2;
7:         if (φ'(α) < 0) then
8:             α_- ← α;
9:         else
10:            α_+ ← α;
11:        end if
12:    end while
13: end procedure
```

We would like to improve this algorithm too, finding a better point in the middle than the middle point.

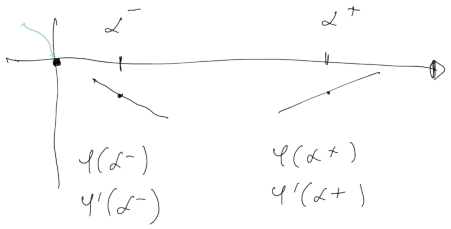


FIGURE 4.13: The information we have about function φ

We may use the information we have about the function, since we know $\varphi(\alpha^-)$, $\varphi'(\alpha^-)$, $\varphi(\alpha^-)$ and $\varphi'(\alpha^-)$.

At this point we can write a model ($m(\alpha) = a\alpha^2 + b\alpha + c$) and specialize it with the information we have, via computing a linear system:

$$\begin{cases} a(\alpha^-)^2 + b(\alpha^-) + c = \varphi(\alpha) \\ a(\alpha^-)^2 + b(\alpha^-) + c = \varphi(\alpha) \\ 2a\alpha^- + b = \varphi'(\alpha^-) \\ 2a\alpha^+ + b = \varphi'(\alpha^+) \end{cases}$$

Then we look for the stationary point of this function and that's the ball where I'm likely to find the root of the derivative.

We can say something more, since the following fact holds:

Fact 4.6.4. *Let $\varphi \in C^3$, then quadratic interpolation has convergence of order $1 < p < 2$ (superlinear)*

In Figure 4.14 we can observe a situation in which the hypothesis of Proposition 4.6.4 aren't satisfied.

FIGURE 4.14: If the function isn't C^3 and one derivative is very big, then the range doesn't shrink much.

We would like to modify the formula to have at least linear convergence.

We can ensure to move not to close to one of the extremes, for example more than 10%.

Another idea is, since we have four equations, to build a cubic function, although the operation is very long. In this case the convergence get quadratic, which is better than linear.

4.7 25th of October 2018 — A. Frangioni

We need to choose a descending direction, we have more choices than the opposite of the gradient.

We want the derivative to be almost zero, given a tolerance value.

Let's see another variant of line search.

Line search: second order approaches

Theorem 4.7.1. *Given $f \in C^2$, $\exists \varphi''(\alpha) = d^T \nabla^2 f(x+\alpha d) d$ and it's continuous.*

Proof. Via chain rule. □

Since we are looking for a point where the derivative $\varphi'(\alpha) = 0$, we may use the second derivative to write a model and, assuming to trust the model, it can be studied.

Definition 4.7.1 (Model−Newton tangent method)**.** *Our model, in this case is*

$$\varphi'(\alpha) \approx \varphi'(\alpha^k) + \varphi''(\alpha^k)(\alpha - \alpha^k)$$

In this context, solving $\varphi'(\alpha) = 0$ implies finding those α such that $\alpha = \alpha^k - \varphi'(\alpha^k)\varphi''(\alpha^k)$

ALGORITHM 4.7.1 Pseudocode for Newton method.

```
1: procedure LSNM(φ', φ'', α, ε)
2:     while (|φ'(α)|> ε) do
3:         α ← α − φ'(α)/φ''(α);
4:     end while
5: end procedure
```

We need to understand when and why $\varphi''(\alpha) \neq 0$ and when and why this method converges.

Theorem 4.7.2. *Let $\varphi \in C^3$ such that $\varphi'(\alpha_*) = 0$ and $\varphi''(\alpha_*) \neq 0$. $\exists\ \delta > 0$ s.t. if $\alpha^0 \in [\alpha_* - \delta, \alpha_* + \delta]$ then $\{\alpha^k\} \to \alpha_*$, with $p = 2$.*

Proof. We are in the hypothesis that the function φ is three times differentiable and we would like to prove that $\alpha^{k+1} - \alpha_* \to 0$

We want to compute how much the error is, if compared to the error at the previous iteration. Since $\varphi(\alpha_*) = 0$ we can use a dirty trick.

1. Since $\alpha^{k+1} \overset{(1)}{=} \alpha^k - \frac{\varphi'(\alpha_*)}{\varphi''(a^k)}$ and $\varphi'(\alpha_*) \overset{(2)}{=} 0$, we obtain:

$$\begin{aligned} \alpha^{k+1} - \alpha_* &\overset{(1)}{=} \alpha^k - \alpha_* - \frac{(\varphi'(\alpha^k))}{\varphi''(\alpha^k)} \\ &\overset{(2)}{=} \alpha^k - \alpha_* - \frac{(\varphi'(\alpha^k) - \varphi'(\alpha_*))}{\varphi''(\alpha^k)} \\ &= \frac{[\varphi'(\alpha^k) - \varphi'(\alpha_*) + \varphi''(\alpha^k) \cdot (\alpha^k - \alpha_*)]}{\varphi''(\alpha^k)} \end{aligned} \tag{4.7.1}$$

Where the term inside the square parenthesis is the first order model, centered in α_* computed in α^k.

2. Now we can use the first form of Taylor's formula, which says$\exists\ \beta \in [\alpha^k, \alpha^*]$ s.t. $\varphi'(\alpha_*) \overset{(3)}{=} \varphi'(\alpha^k) + \varphi''(\alpha^k)(\alpha^k - \alpha_*) + \varphi'''(\beta)\frac{(\alpha^k - \alpha_*)^2}{2}$. Let's see what happens to $\alpha^{k+1} - \alpha_*$:

$$\begin{aligned} \alpha^{k+1} - \alpha_* &\overset{(5)}{=} \frac{[\varphi'(\alpha^k) - \varphi'(\alpha_*) + \varphi''(\alpha^k) \cdot (\alpha^k - \alpha_*)]}{\varphi''(\alpha^k)} \\ &\overset{(3)}{=} \frac{-\varphi'''(\beta)}{2\varphi''(\alpha^k)} \cdot (\alpha^k - \alpha_*)^2 \end{aligned} \tag{4.7.2}$$

3. We can say that the quantity $2\varphi''(\alpha^k)$ doesn't become too small and that the numerator $\varphi'''(\beta)$ doesn't become too big. This is proved since $\exists \delta > 0$ s.t. $\varphi''(\alpha) \geq k_2 > 0$ and also $|\varphi'''(\beta)| \leq k_1 < \infty$. We can go on bounding the difference between α^{k+1} and α_* as follows: for $\alpha,\ \beta \in [\,\alpha_* - \delta\,,\ \alpha_* + \delta$

$$\begin{aligned} |\alpha^{k+1} - \alpha_*| &\overset{(6)}{=} \frac{-\varphi'''(\beta)}{2\varphi''(\alpha^k)} \cdot (\alpha^k - \alpha_*)^2 \\ &= |\alpha^{k+1} - \alpha_*| \leq [\frac{k_1}{2k_2}](\alpha^k - \alpha_*)^2 \end{aligned} \tag{4.7.3}$$

We may notice that $[\frac{k_1}{2k_2}]$ may be very large, but it's multiplied for $(\alpha^k - \alpha_*)^2$, which means that if we start close enough to α^* it's ok.

$$\begin{aligned} |\alpha^{k+1} - \alpha_*| = |\alpha^{k+1} - \alpha_*| &\leq [\frac{k_1}{2k_2}](\alpha^k - \alpha_*)^2 \\ &= |\alpha^{k+1} - \alpha_*| \leq [\frac{k_1}{2k_2}](\alpha^k - \alpha_*) \cdot (\alpha^k - \alpha_*) \end{aligned} \tag{4.7.4}$$

Where $[\frac{k_1}{2k_2}](\alpha^k - \alpha_*) < 1$, so $\frac{k_1(\alpha^k - \alpha_*)}{2k_2} \leq 1 \Longrightarrow \left|\alpha^{k+1} - \alpha_*\right| < \left|\alpha^k - \alpha_*\right|$ At this point, if we start from a point a^0 close enough to α^* (according to this formula) then $\{\,\alpha^k\,\} \to \alpha_*$ and the convergence is quadratic.

□

In the end, we may conclude that if we start from the right point we converge with a quadratic speed.

Problem: This solution makes us compute all the derivatives until the third one. We will now see a solution to this issue.

Exact line search: zeroth-order approaches

Can we do line search without computing derivatives at all? Following this approach we can circumvent the problem of the existence of derivatives. In the case of derivatives definite, it's better if we don't have to compute them.

Key idea: The more derivatives we have, the smallest number of points we need (second derivative → two points, third derivative → zero points). The opposite holds as well.

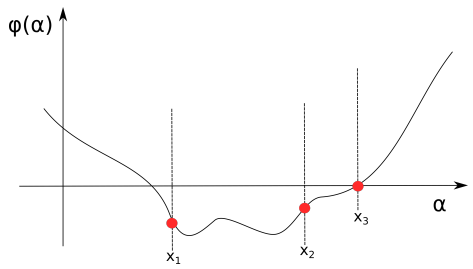


FIGURE 4.15: The interesting interval is $[x_1, x_2]$, since $\varphi(x_2) > \varphi(x_1)$ and we are allowed to exclude the interval $[x_3, +\infty)$ since the value in x_3 is bigger than $\varphi(x_2)$.

Obs: We have to minimize a function we know nothing about, except for its value in a point where we compute it. We would like to reduce to the interval which has as extremes the smallest point.

How can we choose these points? The idea is to choose the points that imply that the interval shrinks as fast as possible.

Obs: We have no guarantee that the interval that we are discarding doesn't contain a very deep minimum.

Elegant solution via golden ratio:

$$r = (\sqrt{5} - 1)/2 \ (\approx 0.618), \ r : 1 = (1 - r) : r$$

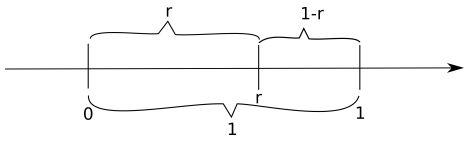


FIGURE 4.16: The relationship between r and $1 - r$ is $r \ : \ 1 = (1 - r) \ : \ r$.

ALGORITHM 4.7.2 Pseudocode for non differentiable functions for local minimum detection.

```
1: procedure LSGRM(φ, α, ε)
2:     α_{i−} ← 0; α_+ ← α;
3:     α'_− ← (1 − r) α;
4:     α'_+ = r α;
5:     while (α_+ − α_− ≤ ε) do // note: not the same ε
6:         if φ(α'_−) > φ(α'_+) then
7:             α_− ← α'_−;
8:             α'_− ← α ← α'_+;
9:             α'_+ ← r(α_+ − α_−);
10:        else
11:            α_+ ← α'_+;
12:            α'_+ ← α ← α'_−;
13:            α'_− ← (1 − r)(α_+ − α_−);
14:        end if
15:    end while
16: end procedure
```

Inexact line search: Armijo-Wolfe

Key idea: Take your favourite line search (it also suggest a simpler one), and run it, but you don't have to wait for the derivative to become zero.

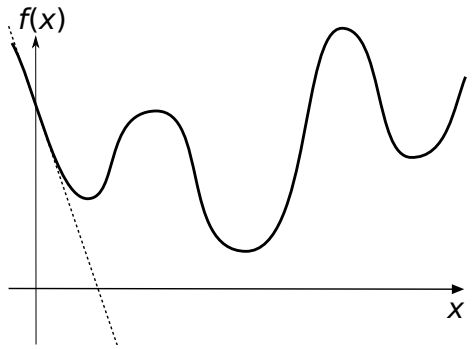


FIGURE 4.17: The dotted line represents the line which has the derivative as slope.

Definition 4.7.2 (Armijo condition)**.** *Let* $0 < m_1 < (\ll)1$

$$\varphi(\alpha) \leq \varphi(0) + m_1 \alpha \varphi'(0) \ (A)$$

Problem of Armijo condition: small steps satisfy the armijo condition, but make convergence very slow.

We need another condition, in order to have a lower bound for the step size:

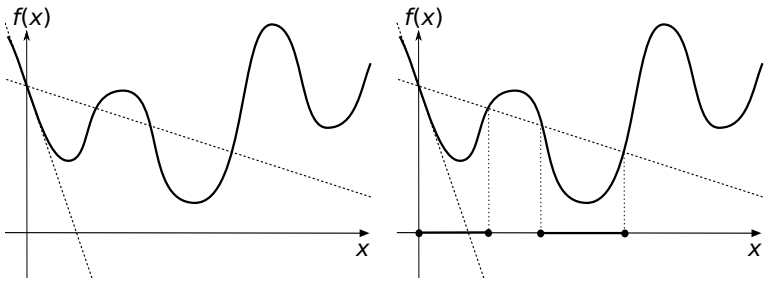


FIGURE 4.18: In the left picture Armijo condition chooses a new line which slope is still negative, but less stip than the original one. In the right one, the ranges where to search are highlighted.

Definition 4.7.3 (Goldstein condition)**.** *Let* $m_1 < m_2 < 1$

$$\varphi(\alpha) \geq \varphi(0) + m_2\alpha\varphi'(0) \ (G)$$

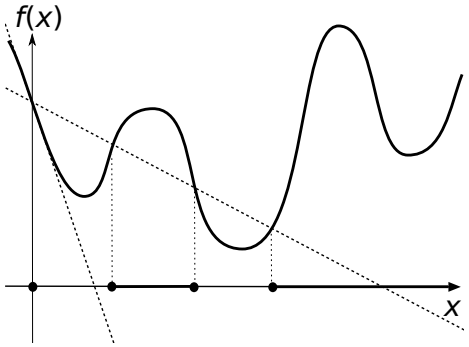


FIGURE 4.19: Goldstein condition chooses a new line which slope is still negative, less stip than the original one, but stipper than the one obtained by Armijo.

Problem: the point that satisfies both Goldstein and Armijo may not contain a local minimum.

To circumvent this problem another condition comes to help us.

Definition 4.7.4 (Wolfe condition)**.** *Let* $m_1 < m_3 < 1$

$$\varphi'(\alpha) \geq m_3\varphi'(0) \ (W)$$

Another issue of this conditions is that the derivative in the interval is quite big on the right side.

Definition 4.7.5 (Strong Wolfe condition)**.**

$$|\varphi'(\alpha)| \leq m_3|\varphi'(0)| = -m_3\varphi'(0)$$

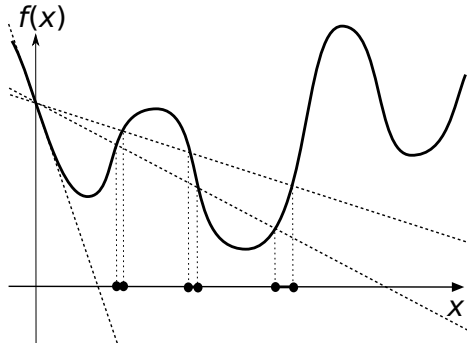


FIGURE 4.20: Here are the intervals that satisfy both Armijo and Goldstein conditions.

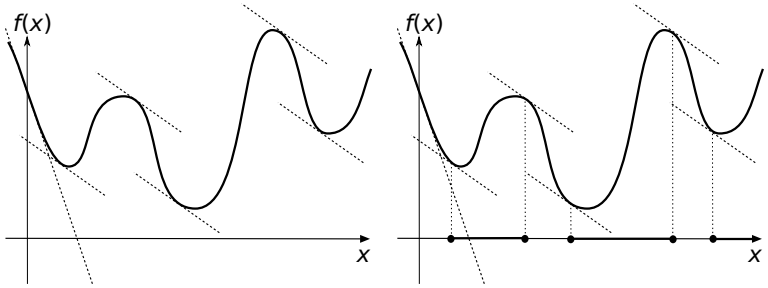


FIGURE 4.21: On the left Wolfe condition (which chooses derivatives that are substantially zero). On the right part the intervals selected by Wolfe.

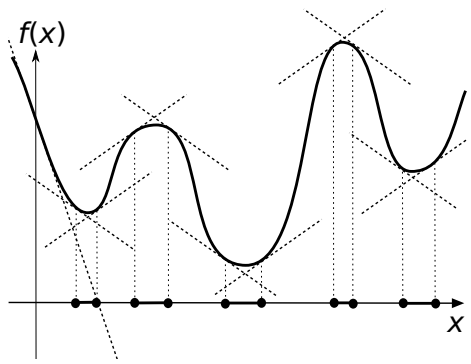


FIGURE 4.22: Strong Wolfe condition.

Fact 4.7.3. *If $\varphi'(\alpha) \not\gg 0$ and (A) $\cap$ (W) / (W') then all local minima (& maxima) are captured unless m_1 too close to* 1 *(that's why usually $m_1 \approx 0.0001$)*

The m_i are like the hyperparameters of machine learning. Less formally, if

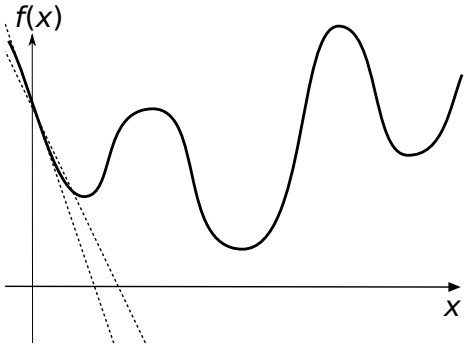


FIGURE 4.23: If the line is too close to the original one only a few points will satisfy Armijo condition which means that the intersection between Armijo and Wolfe will almost be empty.

we choose an m_1 far enough from 1 everything works fine.

Theorem 4.7.4. *Let $\varphi \in C^1$ and $\varphi(\alpha)$ bounded below for $\alpha \geq 0$ then $\exists\ \alpha$ s.t. (A) $\cap$ (W') holds.*

Proof. $l(\alpha) = \varphi(0) + m_1\alpha\varphi'(0)$, $d(\alpha) = l(\alpha) - \varphi(\alpha) \Longrightarrow$
$d(0) = 0$, $d'(0) = (m_1 - 1)\varphi'(0) > 0$ $(m_1 < 1)$ □

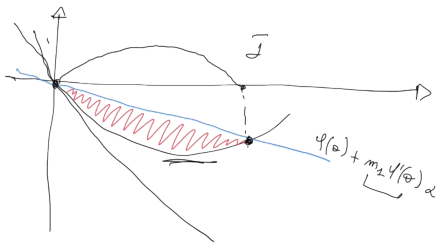


FIGURE 4.24: If the function isn't going to $-\infty$ the blue line and the function will meet again and we denote the value along the x axis $\bar{\alpha}$.

0 and $\bar{\alpha}$ are the two roots of the function d, so we can use Rolle's theorem, in order to prove that the function d has a stationary point in the interval $[0, \bar{\alpha}]$.

$$d(\alpha) = \varphi(0) + \alpha(m_1\varphi'(0)) - \varphi(\alpha)$$

$$d'(\alpha) = m_1\varphi'(0) + \varphi'(\alpha)$$

So $d'(\alpha^*)$ iif $\varphi'(\alpha^*) = m_1\varphi'(0)$. Then strong Wolfe requests that $|\varphi'(\alpha^*)| \leq m_1|\varphi'(0)|$.

How can we find such a point?

ALGORITHM 4.7.3 Pseudocode for backtracking line search.

```
1: procedure BLS(φ, φ', α, m_1, τ)
2:     while (φ(α) > φ(0) + m_1 α φ'(0)) do
3:         α ← τ α; τ < 1;
4:     end while
5: end procedure
```

- Fundamental assumption: ∇f Lipschitz $\implies$ φ' Lipschitz and L does not depend on x^i (**check**)
- Recall: $\exists \bar{\alpha}$ s.t. (A) holds $\forall \alpha \in]0, \bar{\alpha}]$ and $\varphi'(\bar{\alpha}) > m_1 \varphi'(0) > \varphi'(0)$;
- φ' Lipschitz $\implies \bar{\alpha}$ is "large" if $\left\|\nabla f(x^i)\right\|$ is:

  $$L(\bar{\alpha} - 0) \geq \varphi'(\bar{\alpha}) - \varphi'(0) > (1 - m_1)(-\varphi'(0)) \implies \bar{\alpha} > (1 - m_1) \frac{\left\|\nabla f(x^i)\right\| \,|}{L}$$

  (recall $-\varphi'(0) = \left\|\nabla f(x^i)\right\|$);
- Fundamental trick: $\bar{\alpha}$ can $\searrow 0$, but only as fast as $\left\|\nabla f(x^i)\right\|$ does;
- Enough to prove that $\alpha^i \geq \bar{\alpha}$, or "not too smaller".

Now we can prove the following

Theorem 4.7.5. *If (A) $\cap$ (W) holds $\forall i$ then either $\{f(x^i)\} \to -\infty$ or $\{\left\|\nabla f(x^i)\right\|\} \to 0$.*

Proof. **By contraddiction**, we assume $-\varphi'(0) = \left\|\nabla f(x^i)\right\| \geq \varepsilon > 0 \forall i$. Then

1. (W) $\implies \alpha^i \geq \bar{\alpha} > (1 - m_1) \frac{\left\|\nabla f(x^i)\right\|}{L} \implies \alpha^i \geq \delta > 0$;
2. (A) $\implies f(x^{i+1}) \leq f(x^i) - m_1 \alpha^i \left\|\nabla f(x^i)\right\| \leq f(x^i) - m_1 \delta \varepsilon$;
3. So $\{f(x^i)\} \to -\infty$ (or $\{\left\|\nabla f(x^i)\right\|\} \to 0$).

□

Backtracking is similar: for simplicity, $\alpha = 1$ (input)
$\left\|\nabla f(x^i)\right\| > \varepsilon \forall i \implies \bar{\alpha} > \delta > 0 \forall i$
$h = \min\{k : \tau^{-k} \leq \delta\} \implies \alpha^i \geq \tau^{-h} > 0 \forall i \implies f(x^{i+1}) \leq f(x^i) - m_1 \tau^{-h} \varepsilon \implies \{f(x^i)\} \to -\infty$ or ↯.

4.8 8th of November 2018 — A. Frangioni

4.8.1 Good practice of designing a project

In this lecture we focused on how to implement the gradient method inn Matlab and some good hints were underlined:

- When we have to implement a function the first question we need to ask ourselves is: "how many parameters should the function take?";
- The interface needs to be designed, i.e. in the case of the gradient method we would like the user to be able to run it altough he might not know what a good starting point could be;
- There might be some parameters with the same sematic of hyperparameters (using machine learning terminology), so they need to be adjusted in order to specify the algorithm;
- Another thing that should be taken into consideration is that sometimes, when measuring the number of function calls we need to count also the calls to functions inside a single step;
- It's important to check the conditions that should be satisfied by the input data for the theortical coherence. In case of not valid input an error message should be returned;
- Another important thing to be done is building a set of test functions, possibly limit case for the problem, to see how the algorithm works;
- `diff(f, x)` calculates the gradient of the function f in the variable x. If we want to get a simplified version, we may run `simplify(diff(f,x))`;
- A very nice tool to compute derivatives for arbitrary functions is **Adigator** (click here).

4.9 14th of November 2018 — A. Frangioni

Mantra

If you want better convergence, use a better model.

So far we chose the direction for the step as $d^i = -\nabla f(x^i) \cdot \left\|\nabla f(x^i)\right\|$, in particular this is the direction where the decrease of the function is maximum.

We introduced the **convergence argument** which says that in proximity of the stationary point of the linear model the norm of the gradient goes to 0. Formally, $\varphi'(0) = \frac{\partial f}{\partial d^i}(x^i) =< d^i, \nabla f(x^i) >= - < \nabla f(x^i), \nabla f(x^i) >= -\left\|\nabla f(x^i)\right\|$, which implies that $\|\nabla f(x)\| \to 0$ when $\varphi'(0) \to 0$.

Can we take another direction which isn't the opposite of the gradient and have that the same argument holds? Yes, for example if we choose as direction a rotation of the opposite of the gradient, the value of φ' is then the cosine of the angle of the rotation times the opposite of the norm fo the gradient. It's trivial to observe that there are infinite angles that could be chosen, so we have a lot of flexibility.

Note that the angle between d^i and the gradient shouldn't be too close to 90ř, otherwise the cosine would get approximately 0..

Theorem 4.9.1 (Zoutendijk)**.** *Let $f \in C^1$, ∇f Lipschitz and f bounded below. If (A) ∩ (W') then $\sum_{i=1}^{\infty} \cos^2(\theta^i)\left\|\nabla f(x^i)\right\|^2 < \infty$.*

If we have a positive infinite sequence and we have that the corresponding series converges to a number, than the limit of the sequence is going to 0 reasonably fast.

If we choose an angle which is bounded below then the norm of the gradient has to converge very fast. More formally,

Fact 4.9.2. $\cos(\theta^i) \geq \varepsilon > 0 \Longrightarrow \left\|\nabla f(x^i)\right\| \to 0$

Proof. From the observation above, we may recall that the n-th term of the Zoutendijk sum should be approximately 0 $\forall n > \widetilde{n}$, which means that one between $\cos^(\theta^i)$ and $\left\|\nabla f(x^i)\right\|^2$ should be zero.

The proof is obtained from the fact that it can't be the cosine, because it's bounded below. □

4.9.1 Taylor method

At this point we assume that the function is perfectly convex (Hessian strictly positive definite).

Theorem 4.9.3. *Let f be a function such that $\nabla^2 f(x^i) \succ 0$. Then the second order model $Q_{x^i}(y)$ admits a minimum.*

Proof. For simplicity of notation let us forget about the index ^{i} of x^i. The Taylor expansion is the following:

$$\begin{aligned} Q_x(y) &= f(y) + < \nabla f(x), y - x > + \frac{1}{2}(y-x)^T \nabla^2 f(x)(y-x) \\ &= f(x) + < \nabla f(x), y > - < \nabla f(x), x > + \\ &+ \frac{1}{2}\left(y^T \nabla^2 f(x) y - 2x^T \nabla^2 f(x) y + x^T \nabla^2 f(x) y \right) \\ &\overset{*}{=} < \nabla f(x) + x^T \nabla^2 f(x), y > + \frac{1}{2} y^T \nabla^2 f(x) y \end{aligned} \tag{4.9.1}$$

Where $\overset{*}{=}$ is given by the fact that there are some contant terms.
$qy + \frac{1}{2} y^T Q y \Leftrightarrow q + Qy = 0$ so

$$\nabla f(x) - x^T \nabla^2 f(x) + \nabla^2 f(x) y = 0 \nabla^2 f(x) y = \nabla f^2(x) x - \nabla f(x) y = x - [\nabla^2 f(x)]^{-1} \nabla f(x)$$

□

Scrivere meglio la dim

Corollary 4.9.4. *Newton's direction is* $d^i \leftarrow -[\nabla^2 f(x^i)]^{-1} \nabla f(x^i)$ *(just* $\mathbb{R}^n$ *version).*

The direction d is obtained taking the opposite of the inverse of the Hessian times the gradient of the function

Observation 4.9.1. *We need the Hessian to be invertible, which isn't true in general.*

We need to solve a non linear equation, namely a system of equations. A way to circumvent this problem is putting the gradient to 0, so we can write the Taylor form of the gradient and solve a linear equation which is $\nabla f(x) \approx \nabla f(x^i) + \nabla^2 f(x^i)(x - x^i)$.

Theorem 4.9.5. *Let* f *be a function s.t.* $f \in C^3$, $\nabla f(x_*) = 0$ *and* $\nabla^2 f(x_*) \succ 0$. *Then* $\exists \mathcal{B}(x_*, r)$ *s.t.* $x^1 \in \mathcal{B}$, *which implies* $\{x^i\} \to x_*$ *quadratically.*

We may observe that the direction of the Newton method is good not only when we are close to the minimum, but also when we are far.

The scalar product should be negative and it is so, but we also need to ensure that the scalar product isn't too close to 0. When is it that this condition is satisfied? When the function is "reasonable".

Let us now introduce what we mean by "reasonable": a function f such that $uI \preceq \nabla^2 f \preceq LI$, which implies that the function is strongly convex, in other words that the eigenvalues of the Hessian don't get too close to zero (numerically speaking) .

Theorem 4.9.6. *Let* f *be a function that satisfies* $uI \preceq \nabla^2 f \preceq LI$ *and* $\cos(\theta^i) \leq -\lambda^n / \lambda^1 \leq -u/L$. *Then the method converges.*

Proof.

STEP 1 From the definition of d^i we have that

$$d^i = -[\nabla^2 f(x^i)]^{-1} \nabla f(x^i)$$

or, equivalently,

$$\nabla^2 f(x^i) d^i = -\nabla f(x^i)$$

which implies

$$d^i \nabla f(x^i) = -(d^i)^T \nabla^2 f(x^i) d^i \leq -\lambda^n \left\| d^i \right\|^2$$

STEP 2 Since we have that $\cos(\theta^i) = d^i \nabla f(x^i) \cdot (\left\| x^i \right\| \cdot \left\| \nabla f(x^i) \right\|) \leq \delta < 0$, we want to bound the norm of the gradient:

$$\left\| \nabla f(x^i) \right\| = \left\| \nabla^2 f(x^i) d^i \right\| \leq \left\| \nabla^2 f(x^i) \right\| \cdot \left\| d^i \right\| = \lambda^1 \left\| d^i \right\|$$

which implies

$$\cos(\theta^i) \leq -\lambda^n / \lambda^1 \leq -u/L$$

□

Theorem 4.9.7. *Let f be a function that satisfies $uI \preceq \nabla^2 f \preceq LI$ and $\cos(\theta^i) \leq -\lambda^n/\lambda^1 \leq -u/L$. Then the method not only converges, but we also have that for some iteration onwards, $\alpha^i = 1$ always satisfy (A).*

Proof.

$$\begin{aligned} f(x^i + d^i) &= f(x^i) + d^i \nabla f(x^i) + \frac{1}{2} \cdot (d^i)[\nabla^2 f(x^i)] d^i + R(\left\| d^i \right\|) \\ &= f(x^i) - \nabla f(x^i)^T [\nabla^2 f(x^i)]^{-1} \nabla f(x^i) + \\ &+ \frac{1}{2} \nabla f(x^i)^T [\nabla^2 f(x^i)]^{-1} \nabla f(x^i) + R(\left\| d^i \right\|) \\ &= f(x^i) - \frac{1}{2} \cdot \nabla f(x^i)^T \cdot [\nabla^2 f(x^i)]^{-1} \nabla f(x^i) + R(\left\| d^i \right\|) \\ &= f(x^i) + \frac{1}{2} < \nabla f(x^i), d^i > + R(\left\| d^i \right\|) \end{aligned} \tag{4.9.2}$$

□

It can be proved that the convergence is superlinear.

If we start with a step size of 1, we end up in a situation in which the line search isn't computed when we are close to the minimum.

This works under the assumption that the eigenvalues are bounded both above and below (deriving from the bounds on the Hessian).

Interpretation of Newton method

Let us consider the Newton method from a different point of view. We can construct a different space where the gradient method coincides with the Newton method. Given R which is not singular, we make a variable change ($y = Rx$) — which is possible given that R is non singular — and we get $f_y(y) = \frac{1}{2}y^T Iy + qR^{-1}y$, which has as an Hessian the identity metrix, which is the optimal matrix for convergence in Newton method.

Formally, the descending direction d_y is computed as follows: $d_y = -\nabla f_y(y) = -y - R^{-1}q$. Since we chose 1 as step size we obtain that $\nabla f_y(y + d_y) = \nabla f_y(y - y - R^{-1}q) = \nabla f_y(-R^{-1}q) = 0$.

It takes only one iteration, because all the eigenvalues are 1 so the the ratio between the greatest and the smallest is 1 and the subtraction is 0.

If we do the inverse operation ($x = R^{-1}y$) to the direction we get the direction in x variable.

Problem:

We made a lot of assumptions on the Hessian, without the ones there's no guarantee that we are moving on a descending direction. How can we relax these constraints?

Non convex case

If the Hessian isn't positive definite we may take something which is close to the Hessian, which has the bounding property we used before (eigenvalues bounded below and above). In truth, there is no reason to choose exactly the opposite of the inverse of the Hessian times the gradient for the direction, we may use another matrix which is striclty positive definite ad has more or less th same properties of the Hessian.

In particular, given an Hessian that has some negative eigenvalues we can sum to it a multiple of the identity matrix: $H^i = \nabla^2 f(x^i) + \varepsilon^i I \succ 0$. We iterate this procedure until the resulting matrix is strictly positive definite.

How to compute ε numerically? How much larger should ε be? We also want the smallest eigenvalue to be not too close to 0.

$\varepsilon = \max\{0,\ \delta - \lambda^n\}$ for "appropriately chosen smallish δ", in other words we want all the eigenvalues to be at least δ.

The reasons behind the choice of δ (not too small) are bothnumerical (any double $\leq$ `1e-16` "is 0") and algorithmical (if $\lambda^n(\nabla^2 f(x^i) + \varepsilon I)$ "very small" then the axes of $S(Q_{x^i}, \cdot)$ are "very elongated").

It can be proved that the ε we chose is the solution to an optimization problem: $\min\{\left\|H - \nabla^2 f(x^i)\right\| \ H \succeq \delta I\}$. The choice for δ in the MatLab code is 10^{-6}.

Observation 4.9.2. *Note that these contraints are important and we will get back to them later on in the course.*

Could we use a different norm instead of the 2-norm? Yes, for example we can use Frobenius norm, changing a bit the algorithm, but we still get convergence. We would need to solve $\min\{\left\|H - \nabla^2 f(x^i)\right\|_F \; H \succeq \delta I\}$, which is performed in two steps:

1. compute spectral decomposition $\nabla^2 f(x^i) = H\Lambda H^T$
2. $H^i = H\bar{\Lambda}H^T$ with $\bar{\gamma}^i = \max\{\lambda^i, \delta\}$

In both cases, $\{x^i\} \to x_*$ with $\nabla^2 f(x_*) \succeq \delta I$, which implies $varepsilon^i = 0$ and $H^i = \nabla^2 f(x^i)$ eventually. It holds also that we have superlinear convergence if "H^i looks like $\nabla^2 f(x^i)$ along d^i", formally $\lim_{i\to\infty} \left\|(H^i - \nabla^2 f(x^i))d^i\right\| \left\|d^i\right\| = 0$.

Computational complexity

We still need to compute eigenvalues, which takes $O(n^3)$, which is too much if we are in multidimensional spaces.

As a closing observation we may notice that Newton method is very fast to go to a local minimum and this may represent a problem, because it misses global minima.

4.10 16th of November 2018 — A. Frangioni

What happens when the Hessian isn't strictly positive definite? If there are some negative eigenvaues, I can desume that there are some directions in which the function goes to $-\infty$. The model has no minimum, unless we restrict to a **compact set**.

In particular, we may decide to restrict to a part of the space where we can trust our model, which is called **trust region**.

Finding such a region is a NP-hard problem, if we don't restrict to a ball.

Definition 4.10.1 (Karush-Khun-Tucker conditions)**.** *Any optimal solution of the problem x^{i+1} must satisfy that $\equiv \exists\, \lambda \geq 0$ s.t.*

KARUSH: $[H^i + \lambda I]x^{i+1} = -\nabla f(x^i)$ *[linear];*

KUHN: $H^i + \lambda I \succeq 0$ *[semidefinite];*

TUCKER: $\lambda(r - \left\|x^{i+1}\right\|) = 0$ *[nonlinear].*

Where the last condition means that λ has to be 0, unless the solution we get is exaclty on the border of the ball.

What's the difference between this approach and what we used to do before? We have two different cases:

- $\left\|x^{i+1}\right\| < r \implies \lambda = 0 \implies$ normal Newton step ($\mathcal{T}$ has no effect);
- $\lambda > 0$(= small radius r) $\implies$ like in line search with $\varepsilon^i = \lambda$.

The problem is computing these systems of equations taking less than $O(n^3)$.

Key idea

We don't need to compute the Hessian, we can use the first order information to infer things on the second order matrix, although we don't really need to compute the Hessian.

4.10.1 Quasi-newton methods

At a step we have: $m^i(x) = \nabla f(x^i)(x - x^i) + \frac{1}{2}(x - x^i)^T H^i(x - x^i)$, $x^{i+1} = x^i + \alpha^i d^i$

At the next step we recompute the gradient in x^{i+1} and the matrix H^{i+1}: $m^{i+1}(x) = \nabla f(x^{i+1})(x - x^{i+1}) + \frac{1}{2}(x - x^{i+1})^T H^{i+1}(x - x^{i+1})$

How should H^i be chosen? It should satisfy the following:

1. positive definite ($H^{i+1} \succ 0$);
2. we know the gradient in the previous point, we know the gradient in the current point, we construct H^{i+1} such that the model works: $\nabla m^{i+1}(x^i) = \nabla f(x^i)$;

3. $\left\|H^{i+1} - H^i\right\|$ is "small".

This new model isn't too different from the previous one, because of the third preerty. Or equivalently $H^{i+1}(\, x^{i+1} - x^i\,) = \nabla f(x^{i+1}) - \nabla f(x^i)$, which we call **secant equation** and we denote (S).

To ease notation we define s^i such that: $s^i = x^{i+1} - x^i = \alpha^i d^i$ and $y^i = \nabla f(x^{i+1}) - \nabla f(x^i)$.

s^i is chosen, while y^i is decided by the function.

In order to have a matrix H^i that satifies the first two condition we could check that $H^{i+1}s^i = y^i$, because this implies $s^i y^i = (s^i)^T H^{i+1} s^i$ and this implies 1. and 2., hence we obtain the **curvature condition** $s^i y^i > 0$.

Theorem 4.10.1. *Wolf condition implies $s^i y^i > 0$, using the notation we inroduced: (W) $\Longrightarrow$ (C).*

Proof.

$$\varphi'(\alpha^i) = \nabla f(x^{i+1})d^i \geq m_3\varphi'(0) = m_3\nabla f(x^i)d^i$$

$$\Downarrow$$

$$(\nabla f(x^{i+1}) - \nabla f(x^i))d^i \geq (m_3 - 1)\varphi'(0) > 0$$

□

Observation 4.10.1. *We may observe that this theorem implies that if we perform Armijo Wolf exact line search condition (C) can always be satisfied.*

Davidson-Fletcher-Powell

How can we choose a H^i that satisfies the three conditions enumerated above? Taking $H^{i+1} = argmin\{\left\|H - H^i\right\| \ : \ H \in$(S) and $H \succeq 0\}$ is a good idea and for this minimum problem holds the following:

Theorem 4.10.2 (Davidson-Feltcher-Powell)**.** *The new matrix is obtained at each step constructing a rank two matrix, obtained from H^i as a rank to correction, as follows: $H^{i+1} = (I - \rho^i y^i (s^i)^T)H^i(I - \rho^i s^i (y^i)^T) + \rho^i y^i (y^i)^T$*

Let us denote $B^i = {H^i}^{-1}$. At any step we need to compute $B^{i+1} = (H^{i+1})^{-1}$, because we need to solve the system. We have some fomulas that give us a way to compute $(H^{i+1})^{-1}$ from ${H^i}^{-1}$.

Theorem 4.10.3 (Sherman-Morrison-Woodbury)**.** *The inverse of a matrix of the form $A + ab^T$ has the following shape: $[A + ab^T]^{-1} = \frac{A^{-1} - A^{-1}ab^T A^{-1}}{1 - b^T A^{-1} a}$.*

Observation 4.10.2. *From Theorem 4.10.3 we can conclude that $B^{i+1} = \frac{B^i + \rho^i s^i (s^i)^T - B^i y^i (y^i)^T B^i}{(y^i)^T B^i y^i}$.*

It's important to notice that this operation has a cost of $O(n^2)$.

We can do better, in terms of computational complexity.

Broyden-Fletcher-Goldfarb-Shanno

We can use directly B^i, the inverse of H^i. Write (S) for B^{i+1}: $s^i = B^{i+1}y^i \implies B^{i+1} = \text{argmin}\ \{\|B - B^i\| \ : \ \dots\}$.

$B^{i+1} = B^i + \rho^i[(1 + \rho^i(y^i)^T B^i y^i)s^i(s^i)^T - (B^i y^i(s^i)^T + s^i(y^i)^T B^i)]$

This formula proves to be more stable than the other one.

This method takes $O(n^2)$.

The two B^is, obtained from DFP and BFGS, are different although both sensible, but we can use a convex combination of the two.

Observation 4.10.3. *How can we choose H^1? The value we choose will make a differencein the results, at least for the first steps.*

Let us see a couple of choices for B^1:

- Scalar multiples of identity, but how to choose the scalar?
- Compute the gradient in every direction and approximate H. This will cost $O(n^3)$, but it should be done only once.

Let us compute the space needed to store the B^is: order of n^2 is still a lot. What happens if we restrict to working with information of the last k operations?

Poorman's approach

At each step we only consider B^{i-k} and k rank one operations. This operations cost n each, and we have k lines. The problem is that B^{i-k} takes $O(n^2)$ space. We can optimize if we choose B^{i-k} to be simpler, say a multiple of the identity, or finite difference of the gradient. Then the space complexity is $O(kn)$.

I need to tune the algorithm to find the right k which gives me enough precision and also keeps the computational cost low.

Final observation of quasi Newton methods

We may notice that this variation of Newton method doesn't get trapped in local minima, as Newton method did. In the end, the fact that quasi Newton isn't that precise at the beginning may be a good feature.

4.10.2 Conjugate gradient method

Do you recall?

In the gradint method, the angle between two consecutive directions is exactly 90°, as can be seen in Figure 4.25.

We would like to take into account not only the subspace spanned by d^{i+1} but we would like to optimize over larger and larger subspaces (spanned by d^i and d^{i+1}).

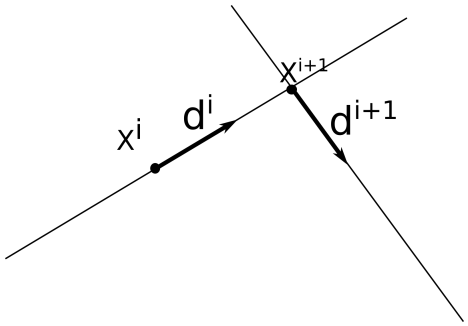


FIGURE 4.25: Geometric idea on how the new direction is chosen.

Definition 4.10.2 (Q-conjugate)**.** *Let v and w be vectors in $\mathbb{R}$. We say that v ad w are **Q-conjugate** if $(v)^T Qw = 0$.*

We would like pick a direction to be Q-conjugate with all the previous iterations. The point is that we can't take into account all the previous directions, but we will see that we only need the previous direction to obtain all the information we need.

ALGORITHM 4.10.1 Pseudocode for conjugate gradient method for quadratic functions.

```
1:  procedure CGQ(Q, q, x, ε)
2:      d⁻ ← 0;
3:      while (‖∇f(x)‖ > ε) do
4:          if (d⁻ = 0) then
5:              d ← −∇f(x);
6:          else
7:              β = (∇f(x)^T Q d⁻)/((d⁻)^T Q d⁻);
8:              d ← −∇f(x) + βd⁻;
9:          end if
10:         α ← (∇f(x)^T d)/(d^T Q d);
11:         x ← x + αd;
12:         d⁻ ← d;
13:     end while
14: end procedure
```

The number of iterations needed to converge is proportional to the clusterization of the eigenvalues of the matrix Q.

The algorithm that was presented is for quadratic functions, but the same algorithm works for non quadratic function as well, as long as we change the fomula for β.

The pseudocode of Algorithm 4.10.2 is referred to Fletcher-Reeves definition of β^i $\beta^i = \left\|\nabla f(x^i)\right\| / \left\|\nabla f(x^{i-1})\right\|^2$.

This algorithm converges in at most n iterations.

ALGORITHM 4.10.2 Pseudocode for conjugate gradient method for arbitrary functions

```
1: procedure CGA(Q, q, x, ε)
2:     ∇f⁻ = 0;
3:     while (‖∇f(x)‖ > ε) do
4:         if (∇f⁻ = 0) then
5:             d ← −∇f(x);
6:         else
7:             β = ‖∇f(x^i)‖² / ‖∇f⁻‖²;
8:             d ← −∇f(x) + βd⁻;
9:         end if
10:        α ← AWLS(f(x + αd));
11:        x ← x + αd;
12:        d⁻ ← d;
13:        ∇f⁻ ← ∇f(x);
14:    end while
15: end procedure
```

We have three different formulas for β_i, which coincide in the quadratic case.

1. Polak-Ribière: $\beta^i = \frac{[\nabla f(x^i)^T(\nabla f(x^i)-\nabla f(x^{i-1}))]}{\|\nabla f(x^{i-1})\|^2}$
2. Hestenes-Stiefel: $\beta^i = \frac{\nabla f(x^i)^T(\nabla f(x^i)-\nabla f(x^{i-1}))}{(\nabla f(x^i)-\nabla f(x^{i-1}))^T d^{i-1}}$
3. Dai-Yuan: $\beta^i = \frac{\left\|\nabla f(x^i)\right\|^2}{(\nabla f(x^i)-\nabla f(x^{i-1}))^T d^{i-1}}$

Some of these algorithms require some hypothesis on the function in order for the conjugate method to converge.

1. Fletcher-Reeves requires $m_1 < m_2 < \frac{1}{2}$ for (A) $\cap$ (W') to work;
2. (A) $\cap$ (W') $\not\Longrightarrow d^i$ of Polak-Ribière is of descent, unless $\beta^i_{PR} = \max\{\beta^i, 0\}$.

Sometimes it's important to restart from scratches if the algorithm isn't converging, because many bad choices may lead to a bad result.

The idea of taking the gradient and modify it instead of multiplying by a factor, adding the previous direction.

It's possibile to design hybrids between quasi-Newton and conjugate method.

4.11 22nd of November 2018 — A. Frangioni

4.11.1 Deflected gradient methods

The idea behind this family of algorithms, is to determine the next position x^{i+1} using the gradient and summing to it something else that gives us more information.

This kind of algorithms work also in cases in which the gradient isn't continuous.

This methods use the information about the prevous iterations without exploiting properties about the second order derivative.

Heavy ball gradient method

The intuition behind this algorithm may be expressed through the following metaphore: an object is moving in the space and it's subject to a force. We can observe that the heavier the object, the stronger should be the force imposed in order to make it describe a certain trajectory.

In this interpretation, we may define the $(i+1)$-th iteration as

$$x^{i+1} \leftarrow x^i - \alpha^i \nabla f(x^i) + \beta^i (x^i - x^{i-1}),$$

where $\boldsymbol{\beta^i}$ is called **momentum**, $\mathbf{x^i}$ **heavy** and $\nabla \mathbf{f}(\mathbf{x^i})$ **force**.

This isn't a descent algorithm: we are not choosing a direction and doing line search. We have no guarantee that the value of the function after one iteration will we smaller than the prevous one.

First thing to do is to choose α^i and β^i properly.

We can prove for some cases that these methods are better than gradient method, although they aren't as good as Newton or quasi Newton. Their strength resides in their simplicity though.

Notice that if the smaller eigenvalue isn't zero (i.e. quadratic case) we have a close formula to choose α and β independently from the iteration:

$$\alpha = \frac{4}{\left(\sqrt{\lambda^1} + \sqrt{\lambda^n}\right)^2}, \qquad \beta = \max\left\{\left|1 - \sqrt{\alpha\lambda^n}\right|, \left|1 - \sqrt{\alpha\lambda^1}\right|\right\}^2$$

We may observe that the step we take is something that goes like $\frac{1}{L}$, where L is the Lipshitz constant, since λ_n is very small. With these choices the rate is the following. We observe that in the gradient the rate is the same, although there aren't the square roots

$$\left\| x^{i+1} - x_* \right\} \leq \left(\frac{\sqrt{\lambda^1} - \sqrt{\lambda^n}}{\sqrt{\lambda^1} + \sqrt{\lambda^n}} \right) \cdot \left\| x^i - x_* \right\|$$

An alternative idea could be choosing β^i and finding α^i using line search. A possible issue is that we don not know if we are moving along a descending

direction, but in this method it is perfectly acceptable not to make any movement at a single step (notice that in gradient method if one step has size 0 then we will not move anymore).

β^i is seen as an hyperparameter, hence its value is tuned rerunning the alogirthm several times.

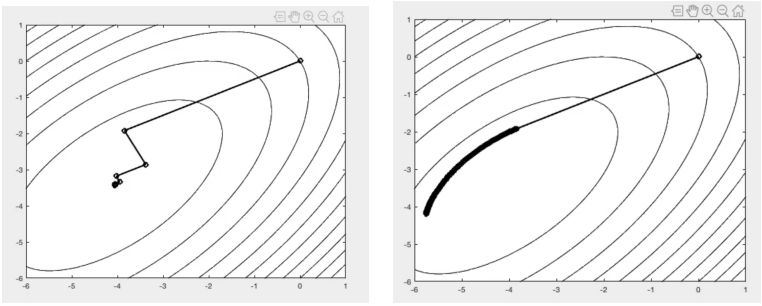


FIGURE 4.26: On the left side Newton method and on the right side the heavy ball method.

The plot of the convergenc of the heavy ball method gives a graphical idea of the fact that the direction isn't orthogonal to the one at the previous iteration, since we have the gradient plus some quantity.

In particular, the bigger β^i the "less orthogonal" the steps are. This feature allows the algorithm to have good performances on elongated functions.

Accelerated gradient

This method works only on convex functions, it has some similarities with heavy ball, but it's slightly different.

ALGORITHM 4.11.1 Pseudocode for accelerated gradient method.

1: **procedure ACCG**$(f, \nabla f, x, \varepsilon)$
2: $x_- \leftarrow x$;
3: $\gamma \leftarrow 1$;
4: **repeat**
5: $\gamma_- \leftarrow \gamma$;
6: $\gamma \leftarrow (\sqrt{4\gamma^2 + \gamma^4} - \gamma^2)/2$;
7: $\beta \leftarrow \gamma(1 / \gamma_- - 1)$;
8: $y \leftarrow x + \beta(x - x_-)$;
9: $g \leftarrow \nabla f(y)$;
10: $x_- \leftarrow x$;
11: $x \leftarrow y - (1 / L)g$;
12: **until** $(\|g\| > \varepsilon)$
13: **end procedure**

The rational behind this algorithm is the following: When we are in a certain point at a certain iteration, we go on a little bit β^i and we end up in a point y. The gradient is computed in that point and used to choose the next point.

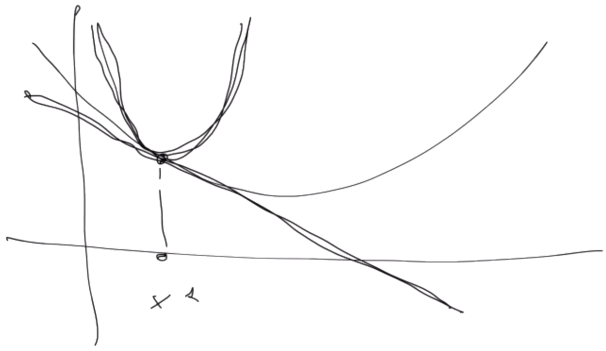


FIGURE 4.27: We build a linear model, which is a lowerbound for the function, then we build a quadratic model, which is above our function.

If we choose γ optimally then the quadratic approximation is very close to the function. We want to find the value for γs that gives best results in the worst case.

We can prove that the error $\left\|f(x^i) - f_*\right\| \leq \sigma^i(f(x^i) - f_*) \searrow 0$ is multiplied by this factor δ_i, that goes like the inverse of i^2.

Notice that if we choose α small, it will always be small.

The convergence is sublinear.

Theorem 4.11.1 (Optimality of accelerated gradient)**.** *If the function isn't strongly convex no algorithm has better convergence than* $\left|f(x^i) - f_*\right| = 3L\frac{\left\|x^1 - x_*\right\|^2}{32(i+1)^2}$.

Observation 4.11.1. *This theorem tells us that this algorithm never gets worse than* $\left|f(x^i) - f_*\right| = 3L\frac{\left\|x^1 - x_*\right\|^2}{32(i+1)^2}$*, but this doesn't imply that this method is fast on average. The state of the art provides a lot of different formulas for* β*, which of the ones leads to some theoretical results.*

From now on we will move towards a different family of functions, that aren't even differentiable, hence we can't compute the gradient.

4.11.2 Incremental gradient methods

This method has good performances in real world machine learning cases, where the function is differentiable but we do not want to compute the gradient.

Let $I = \{1, \ldots, m\}$ be the set of observations, $X = [X^i \subset \mathcal{X}]_{i \in I}$ the set of inputs and $y = [y^i]_{i \in I}$ the set of outputs. Our goal is to explain y from X.

Since these vectors are uniformly distributed over the space (at least this is our hypotesis) when they get summed we expect some of them to cancel out.

The idea is to rewrite the problem as learning a linear function in the feature space, formally $\min\{\sum_{i\in I} l(y^i, <\Phi(X^i), w>) \ : \ w \in \mathbb{R}^n\}$, where $l(\cdot, \cdot) =$ is called **loss function**.

We are now interested in computing the gradient, which is not hard to compute since the function is linear: $\nabla f(w) = \sum_{i\in I} \nabla f^i(w) = \sum_{i\in I} -A^i(y^i - A^i w)$.

The issue here is that computing the gradient, although it's the gradient of a very simple function, takes too long to be computed (since there are too many vectors in machine learning datasets).

To overcome this problem we choose to restrict to only a subset of observations. How can we choose such set? Randomly, of course.

At this point the algorithm isn't deterministic anymore, but it's completely **stochastic**.

The intuition behind this algorithm is to take only one observation, compute what is needed on this observation and make a small step.

An online aplication may be a sensor that produces hundreds of outputs per second and it's not possible to store each of them. They should be used to infer some information and then thrown away.

We study the converge of this kind of algorithms from a stochastic point of view.

In machine learning we always need some regularization, because the tuning of hyperparameters clearly takes into account only the error in the samples that have been seen. Let us regularize the model as follows

$$\min\{\sum_{i\in I} l(y^i, <\Phi(X^i), w>) + \mu\Omega(w) : w \in \mathbb{R}^n\}$$

The usage of regularization may be useful, since we want to keep close to the minimum, but not reach it. It's enough to change slightly the problem and then solve it.

The regularization hyperparameter $\Omega(w)$ may be chosen as follows:

1. Lasso regularizer (best known): $\Omega(w) = \|w\|_1$;
2. In order to increase sparsity: $\Omega(w) = \|w\|^{2^2}$;
3. Leading to feature selection: many $w_j = 0$ as possible.

Ω function is not differentiable, so the function gets non differentiable, as an example look at Figure 4.28, which represents the plot of $f(w_1, w_2) = (3w_1 + 2w_2 - 2)^2 + 10(|w_1| + |w_2|)$.

4.11.3 Subgradient methods

These methods are thought for convex functions that are not differentiable.

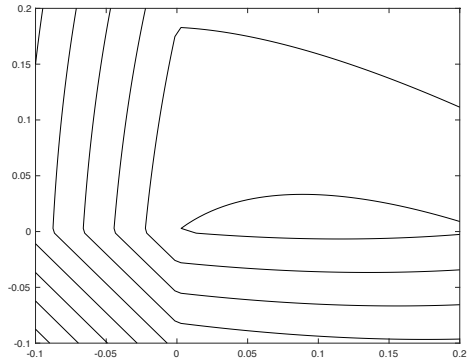


FIGURE 4.28: This function has a lot of kinky points.

Let us consider a kinky point: how can we choose between all the subgradients of that point? We assume to be able to compute some subgradients; since the function is convex we may recall

Property 4.11.2. *Let f be a convex function, $\forall g \in \partial f(x), \forall\ y \in \mathbb{R}^n g(y) \equiv f(y) \geq f(x) + g(y - x)$.*

Let x^* be the optimum, $< x^* - x^i, g >$ is smaller than 0. This means that the angle between x and $x*$ is acute. If we knew the exact direction $x - x^*$ the line search would land on x^*. For a pictorial representation see Figure 4.29

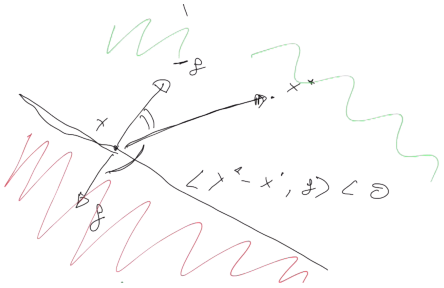


FIGURE 4.29: Since we know that $< g, x^* >$ is negative we get that the angle between g and x^* is larger than 90°. Moreover, we know where the optimum is not (red region), hence we restrict to the green region. At this point we will perform a line search on g direction. Since the function is not smooth, the line search may not succeed since the value of the function along the half line from x in g direction may remain the same, but there may be some points there that are closer to x^*.

The intuition behind this algorithm is to move in the direction of $-g$, but with a small α^i, because if the step is too large we may end up in a point which is actually further from x^* than the previous step. In that point we may

find a point where perform line search, because that point isn't kinky. In this context we won't try to minimize $\|f(x) - f(x^*)\|$, but we will minimize $\|x - x^*\|$, because the function may zigzag near that point. It goes without saying that choosing a too small value for α leads to a too slow convergence speed.

4.12 28th of November 2018 — A. Frangioni

4.12.1 Subgradient methods

We are still in the hypothesis of convex objective functions that are not differentiable in the whole domain.

Do you recall?

Property 4.11.2 of last lecture: for f convex function, $\forall g \in \partial f(x), \forall\ y \in \mathbb{R}^n g(y) \equiv f(y) \geq f(x) + g(y-x)$. This is a characterization of the function with respect to the model.

Let us assume we know the minimum point x_*. We observe that the scalar product between the subgradient and the direction we should choose is negative. Formally, $f(x_*) \geq f(x) + < g, x_* - x >$, hence $< g, x_* - x > \leq f(x_*) - f(x) \leq 0$.

We want to bound the distance between the point at the "next step" and the optimum:

$$\begin{aligned} \left\|x^{i+1} - x_*\right\|^2 &\overset{(1)}{=} \left\|x^i - \alpha^i d^i - x_*\right\|^2 \\ &= \left\|x^i - x_*\right\|^2 + 2\alpha^i g^i \frac{(x_* - x^i)}{\|g^i\|} + \left(\alpha^i\right)^2 \\ &\overset{(2)}{\leq} \left\|x^i - x_*\right\|^2 - 2\alpha^i \frac{(f(x^i) - f(x_*))}{\|g^i\|} + \left(\alpha^i\right)^2 \end{aligned} \tag{4.12.1}$$

Where $\overset{(1)}{=}$ follows from the definition of a normalized step: $x^{i+1} = \frac{x^i - \alpha^i g^i}{\|g^i\|}$ and $\overset{(2)}{\leq}$ follows from the inequality we stated above.

Observation 4.12.1. *The distance from the optimum is bounded by a square function in α_i (the step size), where the linear part is negative and the quadratic part is positive.*

Hence, if the steps are short enough we get close to the optimum fast enough, because the linear part dominates the quadratic one.

Formally, $-2\frac{(f(x^i) - f(x_*))}{\|g^i\|} < 0 + \alpha^i \searrow$, hence $\left(\alpha^i\right)^2 \searrow 0_+$.

An attentive reader may notice that from Equation (4.12.1) follows $\left\|x^{i+1} - x_*\right\|^2 < \left\|x^i - x_*\right\|^2 + \left(\alpha_i\right)^2$ and $\left\|x^{i+1} - x_*\right\|^2 \leq \left\|x^1 - x_*\right\|^2 + \sum_{k=1}^{i} \left(\alpha^k\right)^2$.

The point here is that we cannot choose a too small α, recall Armijo conditions.

If the series of the squares of step sizes does not diverge, then the sequence does not diverge as well, hence it converges somewhere, say $\bar{x}$.

The convergence of the series of the squares may be obtained using the following

Definition 4.12.1 (Diminishing-Square Summable)**.** *We term a series that diverges, while the series of the squares of the terms diverges, as* ***diminischingsquare summable****.*

Formally,

(DSS) $\sum_{i=1}^{\infty} \alpha^i = \infty \ \wedge \ \sum_{i=1}^{\infty} (\alpha^i)^2 < \infty.$

Assumptions: function convex, definite in all its domain, hence the norm of the gradient will never become very large.

Fact 4.12.1. *We claim that the sequence of the stepsizes is a diminishing-square sequence.*

Proof by contraddiction. Let us assume $f(x^i) - f_* \geq \varepsilon > 0, \ \forall i$. Then,

$$\left\|x^{i+1} - x_*\right\|^2 \leq \left\|x^1 - x_*\right\|^2 - \delta \cdot \sum_{k=1}^{i} \alpha^k + \sum_{k=1}^{i} (\alpha^k)^2$$

The contraddiction is due to the fact that as $i \to \infty$ the right-hand side goes to $-\infty$. □

This family of algorithms is clearly incredibly robust in theory, but in practice it does not work very well.

Let us make an experiment: let us suppose we know the minimum of the function f.

Definition 4.12.2 (Plyak stepsize)**.** *Let f be our convex objective function and let x_* be the optimum for f. We term* ***Polyak stepsize****:*

$$\textit{(PSS)} \ \alpha^i = \beta^i \frac{(f(x^i) - f(x_*))}{\|g^i\|}$$

where $\beta \in (0, 2)$.

If we pick a Polyak stepsize, then $\left\|x^{i+1} - x_*\right\|^2 < \left\|x^i - x_*\right\|^2$, so $\{x^i\} \to x_*$. The best value for β_i is 1. In this case, we obtain what follows by sybstituting $\beta_i = 1$ in Equation (4.12.1)

$$\frac{(f(x^i) - f(x_*))^2}{\|g^i\|^2} \leq \left\|x^i - x_*\right\|^2 - \left\|x^{i+1} - x_*\right\|^2$$

The problem is that we don't know the optimum.

Let us assume that we know it and compute the efficiency:

Since we know that the sequence is bounded, we know that the objective function is globally Lipshitz (the norm of the gradient is bounded above).

The point is that the sequence $\{x_1\}$ is not necessarily monotone, so we pick the so-called record value of best vaue ($\underline{f}^i = \min\{f(x^h) \ : \ h = 1, \ldots, i\}$)

$$\frac{(\underline{f}^i - f(x_*))^2}{L^2} \leq \frac{(f^i - f(x_*))^2}{\|g_i\|^2} \leq \left\|x^i - x_*\right\|^2 - \left\|x^{i+1} - x_*\right\|^2$$

Summing for $i = 1, \dots, k$ we obtain a telescopic series:

$$\left\|x^1 - x_*\right\| - \cancel{\left\|x^2 - x_*\right\|} + \cancel{\left\|x^2 - x_*\right\|} - \cancel{\left\|x^3 - x_*\right\|} + \cdots + \cancel{\left\|x^k - x_*\right\|} - \left\|x^{k+1} - x_*\right\|$$

Hence resulting in

$$k \cdot \frac{\left(\underline{f}^k - f(x_*)\right)^2}{L^2} \leq \left\|x^1 - x_*\right\|^2 - \left\|x^{k+1} - x_*\right\|^2 \leq \left\|x^1 - x_*\right\|^2 = R$$

which is equivalent to $\left(\underline{f}^k - f(x^*)\right)^2 \leq \frac{R^2 L^2}{k}$, which is again equivalent to $\underline{f}^k - f(x^*) \leq \sqrt{\frac{RL}{k}}$, where L is the Lipshitz constant.

The issue here is that the convergence is sublinear: $k \geq \frac{1}{\varepsilon^2}$.

Theorem 4.12.2. *Take an algorithm that uses only the subgradient. It's possible to construct a function that makes the algorithm converge with sublinear speed. Hence, we cannot do better.*

It comes without saying that although this algorithm is not very good it is the "less bad" it can be.

There are some lucky cases in which we do know the optimal value, but this is not the case usually.

Target level stepsize

Let us assume $f(x_*)$ is unknown. The only information available is that this value is below any value in any iteration.

The rationale behind this algorithm is to assume to know the optimal value and as soon as we realize it is not correct we change it.

Let us first give an informal description of the algorithm:

- δ is the displacement: how much below the function is with respect to the best value obtained so far;
- reference value $f_{rec} = \underline{f}$. At the beginning is the value at the first iterate and then we define the target value as the difference between the reference value and some δ (at the beginning δ_0).

ALGORITHM 4.12.1 Pseudocode for target level stepsize.

```
1:  procedure SGPTL(f, g, x, i_max, β, δ_0, R, ρ)
2:      r ← 0;
3:      δ ← δ_0;
4:      f_ref ← f_rec ← f(x);
5:      i ← 1;
6:      while (i < i_max) do
7:          g = g(x);
8:          α = β(f(x) − (f_ref − δ))/‖g‖^2;
9:          x ← x − αg;
10:         if (f(x) ≤ f_ref − δ/2) then
11:             f_ref ← f_rec;
12:             r ← 0;
13:         else
14:             if ( r > R ) then
15:                 δ ← δρ;
16:                 r ← 0;
17:             else
18:                 r ← r + α‖ g ‖;
19:             end if
20:         end if
21:     end while
22:     f_rec ← min{ f_rec , f(x) };
23:     i ← i + 1;
24: end procedure
```

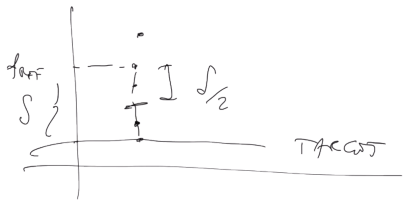


FIGURE 4.30: There are two cases: either the function value is significanlty below the reference (for example $\frac{\delta}{2}$ below the reference) or it's not . If we are in the "happy case" (we just move the reference to the best value). For what conserns the "unhappy case", if we are unhapy for 1 iteration, no problem. Two iterations? no prolem. Many iterations? Problem: after some iterations in which we are not improving it means that we have to decreas the reference values. How? r is updated at each bad step and reset when a good step occurs. When r gets too large we decrease δ and reset everything.

At this point we defined the algorithm and we are ready to implement it,

except for the fact that we need to choose of a lot of parameters. A way to choose them is to use the ML approach: try many possible values.

Two big issues of this algorithm are that it does not provide a good stopping criterion and it is very sensitive to many parameters.

4.12.2 Deflected subgradient

The idea is to use the same trick of ball-step (also called primal-dual).

Let us assume that our function was differentiable. The subgradient method collapses to the gradient method and we know that the gradient method does not provide a good convergence. Yet, deflection is possible: $d^i = \gamma^i g^i + (1 - \gamma^i) d^{i-1}$, $x^{i+1} = x^i - \alpha^i d^i$. We can prove that d^i approximates the subgradient. We can also prove that the algorithm converges in the end. The prameters of this algorithm are two: β (stepsize) and γ (deflection). In order to choose them we have two different approaches:

STEPSIZE-RESCRICTED $\equiv$ deflection-first. We first choose β and when choosing α we need to take into account β: $\alpha^i = \frac{\beta^i (f(x^i) - f_*)}{\|d^i\|} \ \wedge \ \beta^i \leq \gamma^i$ "as deflection $\nearrow$, stepsize has to $\searrow$";

DEFLECTION-RESTRICTED $\equiv$ stepsize-first. We first choose γ, then we pick a step size that depends on γ:

$$\text{(DSS)} \ \wedge \ \frac{\alpha^{i-1} \left\| d^{i-1} \right\|}{(f(x^i) - f_*) + \alpha^{i-1}} \left\| d^{i-1} \right\| \leq \gamma^i$$

"as $f(x^i) \to f_*$, deflection $\searrow$"

This algorithm gets the optimal $O(1/\varepsilon^2)$ on average, sadly not worst case.

4.12.3 Smoothed gradient methods

Let us assume that the target function is a **Lagrangian function**.

Definition 4.12.3 (Lagrangian function)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function of the following shape:*

$$f(x) = \max\{x^T A z \ : \ z \in Z\}$$

where Z is convex and bounded.

Let us assume that Z is also compact.

A graphical example in the case of $f(x) = |x| = \max\{x, -x\} = \max\{zx : z \in [-1, 1]\}$ is shown in Figure 4.31.

In the case of the absolute value, the nasty trick is "to make it have only one optimal solution" in the point 0.

This result is obtained adding a small quadratic term: $f(x) = max\{x^T A z - \mu \|z\|^2 \ : \ z \ inG\}$ that is shown in Figure 4.32. At this point the new function f_μ is not the original function anymore, but it is very close to it whenever the μ is small.

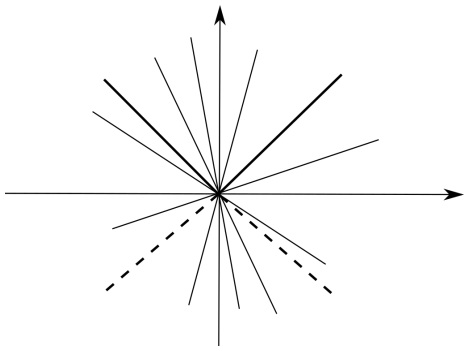


FIGURE 4.31: Let us take the absolute value function $f(x) = |x|$. This function would be differentiable, if it wasn't for some nasty points (in this case 0, were the optimization problem has many optimal solutions). In 0 there are many subgradients, so it has many optima.

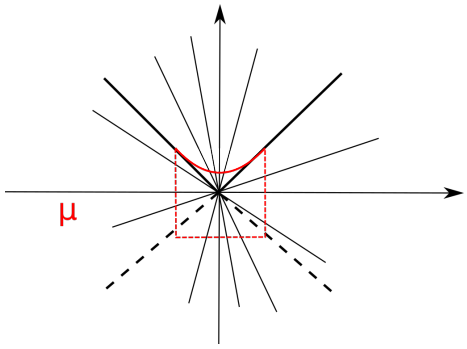


FIGURE 4.32: Geometric intuition of the usage of variable μ.

Notice that this new function is smooth (differentiable).

It might be not ery easy to compute once chosen the value for μ.

We are solving a problem which is different from the original one and would be exactly the same if μ where 0.

On the other hand for $\mu = 0$ we have the problem which is not differentiable once again, so we need to keep close to 0 but not too close.

At this point we are in the situation of $f_\mu(x) \leq f(x) \leq f_\mu(x) + \mu R$, such that as $\mu \searrow 0$, "argmin $\{ f_\mu(x) \} \to x_*$".

The new function is not only convex, but it is also Lipshitz continuous: ∇f_μ Lipschitz with $L = \|A\|^2/\mu$, but it is "less and less Lipschitz" as $\mu \searrow 0$.

Fact 4.12.3. *If $f_* > -\infty$ and picking a very special value of μ ($\mu = \varepsilon/(2R)$), then an appropriate ACCG obtains $f(x^i) - f_* \leq \varepsilon$ for $i \geq 4 \cdot \|A\| \cdot \|x_*\| \cdot \sqrt{R}/\varepsilon$.*

We observe that the convergence is much better, because it depends on

$O(1/\varepsilon)$ instead of $O(1/\varepsilon^2)$.

FIGURE 4.33: At the beginning we eill make a lot of bad steps (the upper gray line). We can improve (pick the black line) changing the ε value and we obtain something that looks more stable. The more precision we want, the smaller the step we make. At the ending it pays, but at the beginning it is not so.

4.12.4 Cutting-plane algorithm

We cheated to get first order information, we want to do more. We want to cheat and have also the second order information.

We want to use the same idea of limited memory quasi Newton methods. Using some limited memory of the Hessian, in order to understand th curvature.

The point is that the directional derivatives are defined and (if computed massively) give me a hint of the curvature of the function (cfr. Figure 4.34). Notice that it is not possible to build a matrix, because it is not defined.

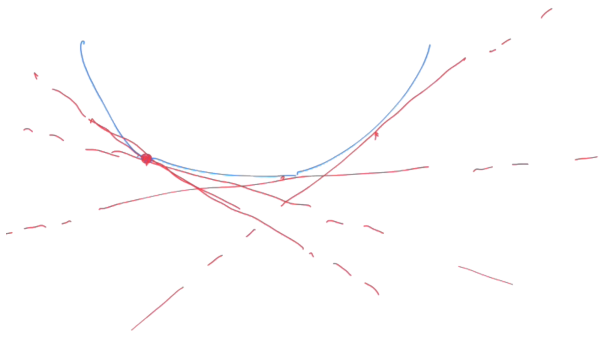


FIGURE 4.34: The idea is that we do not compute a subgradient and discard it. We store it, thus resulting in a model

Let us say we have performed i iterates, then we have collected i subgradients ($\mathscr{B} = \{(x^i,\ f^i = f(x^i),\ g^i \in \partial f(x^i))\} \equiv$ bundle of first-order information) and function values.

We can now define a piece wise linear function defined as the maximum of

the first order model:

$$f_{\mathscr{B}}(x) = \max\{f^i + g^i(x - x^i) \ : \ (x^i, f^i, g^i) \in \mathscr{B}\}$$

At this point we can apply Newton method: first we minimize the model and the use the minimum as next point. We collect information in that specific point and then we repeat.

Notice that the model is always below the objective function ($f_{\mathscr{B}}(x) \leq f(x) \ \forall x$), hence $\min\{f_{\mathscr{B}}(x)\} \leq f_*$, so $x^*_{\mathscr{B}} \in \operatorname{argmin}\{f_{\mathscr{B}}(x)\} \approx \ x_*$.

This function has many kinky points, so how to minimize it? Dirty trick from Ricerca Operativa:

$$\min\{\, f_{\mathscr{B}}(x)\,\} = \min\{\, v \ : \ v \geq f^i + g^i(x - x^i), \text{ where } (x^i, f^i, g^i) \in \mathscr{B}\}$$

And on this problem, we can use the simplex method.

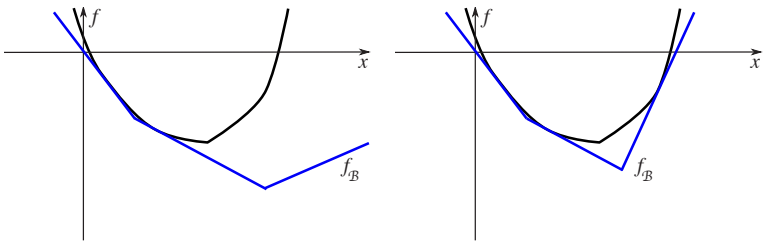


FIGURE 4.35: A geometric representation of how the model (blue line) changes after one iteration.

At this point we have obtained both an upperbound and a lower bound, which go one towards the other.

We may decide to stop iterating when they are "close enough".

The problem is that at each step we need to solve a minimization problem and the convergence is very slow.

It's also possibile that the blue function (the model) does not have a minimum (unbounded below).

How to overcome this problem? Use so-called bundle methods.

4.12.5 Bundle methods

The intuition behind this is that we want to keep quite close to the point where the model corresponds to the function, because the further we get the more the difference from the function.

A way to express this is to add a quadratic quantity to the function that grows when we move outside the current point.

Definition 4.12.4 (Stabilized master problem)**.** *We term* ***stabilized master problem*** *the following*

$$\min\{f_{\mathscr{B}}(x) + \mu\|x - \bar{x}\|^2/2\}$$

This improved model cannot be undounded below (quadratic function), but we need to choose μ and $\bar{x}$ wisely.

A possible way is to move the **stability center** whenever the current value is better the the best encountered so far.

ALGORITHM 4.12.2 Pseudocode for boundle method.

```
1: procedure PBM(f, g, x̄, m_1, ε)
2:     choose μ;
3:     B ← {(x̄, f(x̄), g(x̄))};
4:     while ( true ) do
5:         x* ← argmin {f_B(x) + μ‖x − x̄‖²/2};
6:         if (μ‖ x* − x̄ ‖_2 ≤ ε) then
7:             break;
8:         end if
9:         if (f(x*) ≤ f(x̄) + m_1(f_B(x*) − f(x̄))) then
10:            x̄ ← x*;
11:            possibly decrease μ;
12:        else
13:            possibly increase μ;
14:        end if
15:        B ← B ∪ ( x* , f(x*) , g(x*) );
16:    end while
17: end procedure
```

This algorithm may never move (without cycling, luckily), but at least we gained some information.

The bundle method converges in few steps, although each step is quite costly.

We reached a point where to solve an unconstrained problem we need to solve a contrained one, so from next lecture we will start dealing with constrained optimization problems.

4.13 30th of November 2018 — A. Frangioni

4.13.1 Constrained optimization

In this lecture we address the problem of finding the **optimum** of a function in a subset of its domain, called X. The term optimum differs from the minimum, because the optimum in that subset may not be a minimum of the whole function.

$f_* = \min\{ f(x) \ : \ x \in X \}$

Definition 4.13.1 (Local optimum)**.** *Given a function f and a constraint set X, we denote **local optimum** the point where the function assumes the minimum value inside the set X. Formally,* $\min\{f(x) \ : \ x \in \mathcal{B}(x_*, \varepsilon) \ \cap \ X\}$ *for some* $\varepsilon > 0$.

Notice that the only points in which the constraint adds some informations are the ones on the boundary, as shown in Figure 4.36

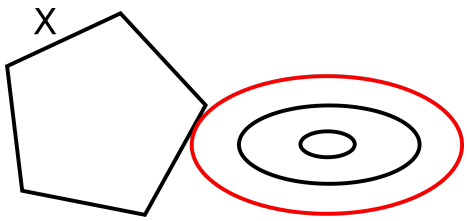


FIGURE 4.36: The red line is level set of the function corresponding to the smallest value that touches the set X. The point in the intersection is not a saddle point of the function f, altough it is the minimum.

There are two kinds of contraints:

FAKE ONES: this first kind is such that the minimum of the function lies inside the set X, hence there is no need to use the constraints at all;

REAL ONES: when the optimal is on the boundary. This is the case of linear functions, because the gradient is constant $\nabla f(x) = c$.

At this point we want to decide if a point on the boundary is an optimum. In this context it is important how the boundary is defined.

Linear equality constraints

A constraint of this kind is very simple: it is a subspace, as shown in Figure 4.37.

$\min\{f(x) \ : \ Ax = b\}$, where the rank of A counts the number of linearly independent rows.

Let us assume that there are not linearly independent rows in A, then or this behaviour is reflected in B or the system does not have any solution.

In the case of presence of linearly dependent columns such columns may be eliminated to ease the computation without loss of information.

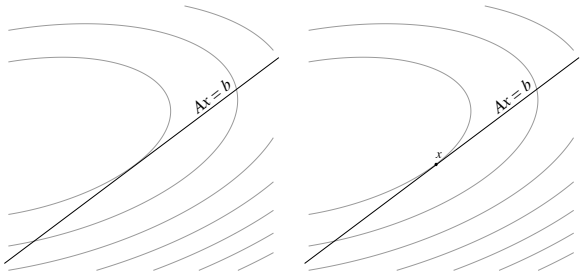


FIGURE 4.37: Linear constraint and a point on the boundary.

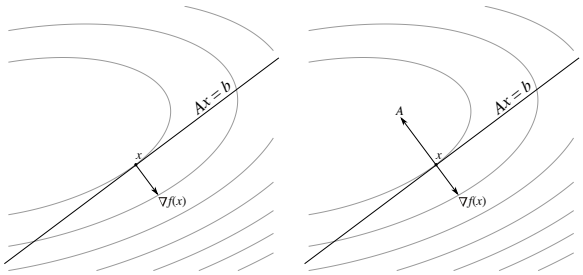


FIGURE 4.38: The gradient is orthogonal to the level set in that point, when the function is smooth. The same holds for matrix A.

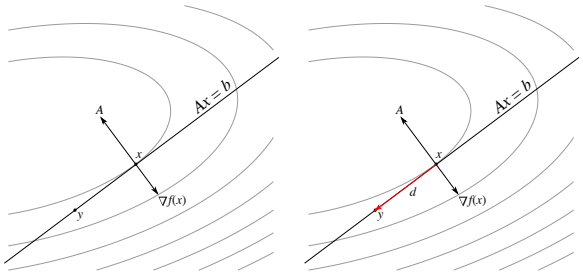


FIGURE 4.39: If we take any other point in the space it has to be orthogonal to A.

The intuition behind what follows is that each linear constraint kills one degree of freedom, formally $det(A_B) \neq 0 \Rightarrow Ax = b \equiv x_B = A_B^{-1}(b - A_N x_N) \Rightarrow$.

We want to extract a submatrix A_B from $A \in M(m, n, \mathbb{R})$, such that $A_B \in M(m, \mathbb{R})$ and then the system induces a partitioning in the variables as well.

$A = [A_B, A_N]$, $x = [x_B, x_N]$, so the system becomes $A_B x_B + A_N x_N = b$ and (since A_B is non singular) $X_B + {A_B}^{-1} x_N = {A_B}^{-1} b$ in other words, given the indepedent variables we can compute the values of the dependent ones and this is a linear operation.

The original optimization problem becomes an optimization problem on a reduced space, formally $\min\{r(w) = f(Dw + d) : w \in \mathbb{R}^{n-m}\}$, where $D =$

$\begin{bmatrix} -A_B^{-1}A_N \\ I \end{bmatrix}$ and $d = \begin{bmatrix} A_B^{-1}b \\ 0 \end{bmatrix}$. For each point in the smaller space we can compute the function in the larger space.

How can we compute the gradient of $r(w)$? The gradient is $\nabla\ r(w) = D^T \nabla\ f(Dw + d)$. The fact that w^* is an optimum implies that $\nabla\ r(w^*) = 0$.

$D = \begin{bmatrix} -A_B^{-1}A_N \\ I \end{bmatrix}$ then $AD = [A_B, A_N] \cdot \begin{bmatrix} -A_B^{-1}A_N \\ I \end{bmatrix} = -A_B \cdot {A_B}^{-1} \cdot A_N + A_N = 0$.

Now the point is taking a multiple of matrix A, finding a feasible x and corresponding w (because there is a bijection), then finding the value μ that allows the equality.

Theorem 4.13.1. *Let $Ax = b$ be a linear system and w such that $x = \left[{A_B}^{-1}(b - A_N w), w\right]$. If $\exists\ \mu\ \in\ \mathbb{R}^m$ s.t. $\mu A = \nabla\ f(x)$ then $r(w) = D^T \nabla\ f(x) = 0$, see Figure 4.40. In other words, it is equivalent to find a stationary point x for the original problem (P) or finding the stationary point w for (R).*

Definition 4.13.2 (Poorman's Karush Kuhn-Tucker conditions)**.** *A point is a good candidate for being a minimum of the constrained problem if and only if it satisfy **Poorman's KKT conditions**, namely the problem is feasible and that $\exists \mu \in \mathbb{R}^m$ s.t. $\mu A = \nabla f(x)$.*

Theorem 4.13.2. *Let f be a convex function, then KKT conditions are enough for optimality.*

A very naive explanation of the theorem is that if the function is convex also the restriction is convex and a tationary point of a convex function is a minimum.

Our idea is to characterize the directions we can move along in order to find new points that satisy the constraint. Formally, $Ax = b$ is our contstraint and we want to move towards $x + d$ and stay in the feasible region. How? $A(w + d) = b \Leftrightarrow Ax + Ad = b \Leftrightarrow Ad = 0$, since $Ax = b$. The only way to move along the constraint is choosing a direction which scalar product with A is 0, hence 0 scalar product with the gradient.

From now on we would like to study the behaviour on contrained problems where the constraints are equalities, but inequalities.

In order to do this we need some mathematical background.

Background for linear inequality contstraints

Definition 4.13.3 (Tangent cone)**.** *We call **tangent cone** of X at x $\mathbf{T_X(x)}$ =t.*

$$\{d \in \mathbb{R}^n\ :\ \exists\ \{z_i \in X\} \to x\ \wedge\ \{t_i \geq 0\} \to 0\ s.t.\ d = \lim_{i \to \infty} \frac{z_i - x}{t_i}\}$$

Theorem 4.13.3. *Let $\mathcal{C}$ be a cone, $\forall x \in \mathcal{C}\ \alpha x \in \mathcal{C},\ \forall \alpha > 0$.*

Theorem 4.13.4. *Given a function f, where x is a **local** optimum $< \nabla f(x), d > \geq 0\ \forall d \in T_X(x)$.*

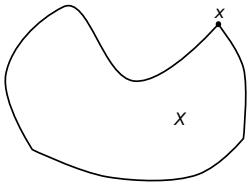


FIGURE 4.40: When the boundary has this shape we can move along directions that point inside the constraints. Tangent directions are not allowed.

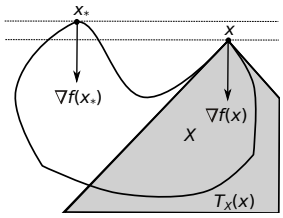


FIGURE 4.41: Geometric representation of tangent cone, which is the region of the space where we can pick the directions. The intuition is to zoom in x and the result of this zooming is a cone.

Proof. **Proof by contraddiction:** Assume $\exists d \in T_X(x)$ such that $< \nabla f(x), d ><0$, but x is a local optimum.

By definition $\exists X \supset \{z_i\} \to x$ and $\{t_i\} \to 0$ such that $d \overset{*}{=} \lim_{i\to\infty} \frac{z_i - x}{t_i}$.

First order Taylor $f(z_i) - f(x) =< \nabla f(x), (z_i - x) > + R(z_i - x)$.

$$\begin{aligned} \lim_{i\to\infty} \frac{f(z_i - x)}{t_i} &= \lim_{i\to\infty} < \nabla f(x), \frac{z_i - x}{t_i} > + \frac{R(z_i - x)}{t_i} \\ &\overset{*}{=} < \nabla f(x), d > + \lim_{i\to\infty} \frac{R(z_i - x)}{t_i} \\ &\overset{(1)}{=} < \nabla f(x), d > \\ &< 0 \end{aligned} \tag{4.13.1}$$

Where, $\overset{(1)}{=}$ follows from $\lim_{i\to\infty} \frac{R(z_i - x)}{t_i} = 0$ by Taylor. □

Observation 4.13.1. *The optimum of Theorem 4.13.4 is global when the function is convex, because in that case $X \subseteq x + T_X(x)$. For a geometric idea see Figure 4.42.*

Observation 4.13.2. *Notice that the rule $< \nabla f(x), d >\geq 0 \ \forall d \in T_X(x) \Rightarrow x$ local optimum does not hold. Let us see a counter example:* $\min\{\, x_2 : x_2 \geq x_1^3 \,\}$, *displayed in Figure 4.44.*

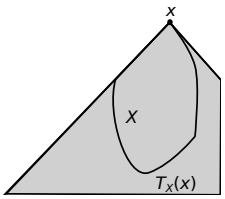


FIGURE 4.42: Convex function.

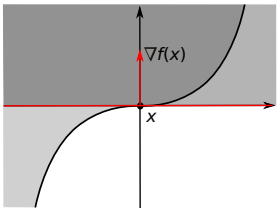


FIGURE 4.43: Let us suppose we pick the direction towards the left part of the function in the saddle point. That direction is promising and actually the value of the function decreases. In this case the problem is that the contraint is not convex.

Now we need a more manageable object for $T_X(x)$.

Definition 4.13.4 (Cone of feasible directions)**.**
Intuitively, we are in x and we want to find al directions ***feasible cone*** *such that there exist small but not* 0 *steps such that all the points on this direction are feasible. Formally, a* ***feasible cone*** *is $F_X(x) = \{d \in \mathbb{R}^n \ : \ \exists \bar{\varepsilon} > 0$ such that $x + \varepsilon d \in X, \ \forall \varepsilon \in [0, \bar{\varepsilon}]\}$.*

Fact 4.13.5. *The properties of such cone are:*

1. *T_X closed, F_X in general not (hence the cone of feasible directions is the tangent cone minus the tangent directions);*
2. *$cl(F_X) \subseteq T_X$, where $cl(F_X)$ is the closure of the cone of feasible directions;*
3. *If X convex then the cones coincide: T_X and F_X convex and $cl\, F_X = T_X$.*

We are now interested in finding a better characterization of the cone fo feasible directions, hence we introduce a new characterization fo the set of constraints X.

FIRST REPRESENTATION:

$$X = \{x \in \mathbb{R}^n \ : \ g_i(x) \leq 0 \ \ i \in \mathcal{I}, \ h_j(x) = 0 \ \ j \in \mathcal{J}\}$$

where $\mathcal{I}$ is the set of inequality constraints and $\mathcal{J}$ is the set of equality constraints;

Second representation:

$$X = \{\, x \in \mathbb{R}^n \,:\, G(x) \leq 0\,,\, H(x) = 0\}$$

where $G = [g_i(x)]_{i\in\mathcal{I}} : \mathbb{R}^n \to \mathbb{R}^{|\mathcal{I}|}$ and $H = [h_i(x)]_{i\in\mathcal{J}} : \mathbb{R}^n \to \mathbb{R}^{|\mathcal{J}|}$;

Third representation: (hiding equalities)

$$X = \{x \in \mathbb{R}^n : g_i(x) \leq 0 \; i \in \mathcal{I},\; h_j(x) \leq 0 \;\wedge\; h_j(x) \geq 0 \; j \in \mathcal{J}\}$$

Fourth representation: (hiding inequalities into a single function)

$$X = \{x \in \mathbb{R}^n \,:\, g(x) = \max\{\, g_i(x) \,:\, i \in \mathcal{I}\,\} \leq 0 \; i \in \mathcal{I},\; h_j(x) = 0 \;\in \mathcal{J}\}$$

Definition 4.13.5 (Active constraints)**.** *We term* ***active constraints*** *at $x \in X$ the following set*

$$\mathcal{A}(x) = \{i \in \mathcal{I} \;:\; g_i(x) = 0\} \subseteq \mathcal{I}$$

Let us introduce some useful notation on the subject: let $\mathcal{B} \subseteq \mathcal{I}$ a subset of indices.

We denote $G_{\mathcal{B}} = [g_i(x)]_{i\in\mathcal{B}} : \mathbb{R}^n \to \mathbb{R}^{|\mathcal{B}|}$ the corresponding set of inequalities.

Definition 4.13.6 (First-order feasible direction cone)**.** *We term* ***First-order feasible direction cone at*** *$x \in X$:*

$$D_X(x) = \{d \in \mathbb{R}^n \;:< \nabla g_i(x), d >\leq 0 \; i \in \mathcal{A}(x) \;< \nabla h_j(x), d >= 0 \; j \in \mathcal{J}\} = \{d \in \mathbb{R}^n : (JG_{\mathcal{A}(x)}(x))d \leq 0$$

Intuitively, the fact that we are looking at the active set means that we are zooming very close to 0*.*

In this cone we require that all the directions inside it have a negative scalar product with the gradient of the contraints.

A visual example of a first order feasible direction cone is displayed in Figure 4.44.

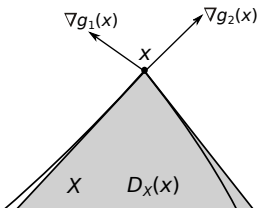


Figure 4.44: The first-order feasible direction cone is made of those directions that are orthogonal to the gradient of the constraints $g_1(x)$ and $g_2(x)$ in x.

Fact 4.13.6. *The tangent cone is a subset of the first-order feasible direction cone. Formally,* $T_X(x) \subseteq D_X(x)$.

We would like the first-order feasible direction cone to be exactly equal to the tangent cone and this is almost always true, except for some pathological cases.

Example 4.13.1. *Let our minimum problem be* $\min\{\dots : {x_1}^2 + (x_2 - 1)^2 - 1 \leq 0,\ x_2 \leq 0\}$.

The plot of such functions is shown in Figure 4.45.

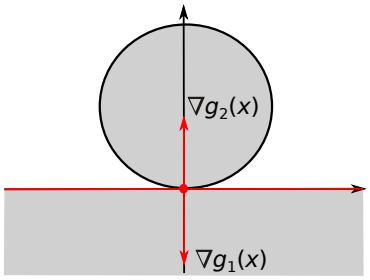


FIGURE 4.45: The circle represents the quadrati constraint, while the semi-plane represents the first degree constraint.

Our claim is that the only feasible point is $X = \{x = [0, 0]\}$.

The only feasible direction is 0*, hence cone of feasible direction and the tangent cone are the singleton* $\{[0, 0]\}$.

On the other hand, the set of directions that have non-negative scalar product with both g_1 *and* g_2 *are all the* x *axis.*

We would like to ensure we are not in one of these pathological cases and to do this we introduce some conditions.

Fact 4.13.7. *The following holds:*

AFFINE CONSTRAINTS (AFFC): *Let* g_i *and* h_j *be affine constraints. Then,* $\forall\ i \in \mathcal{I}$ *and* $j \in \mathcal{J}$ $T_X(x) = D_X(x)\ \forall x \in X$.

SLATER'S CONDITION (SLAC): *Let* g_i *convex* $\forall\ i \in \mathcal{I}$ *and let* h_j *affine* $\forall\ j \in \mathcal{J}$ $\exists\ \bar{x} \in X$ *s.t.* $g_i(\bar{x}) < 0\ \forall\ i \in \mathcal{I}$. *Then* $T_X(x) = D_X(x)\ \forall\, x \in X$*;*

LINEAR INDEPENDENCE (LINI): $\bar{x} \in X \wedge$ *the vectors* $\{\nabla g_i(\bar{x}) : i \in \mathcal{A}(\bar{x})\} \cup \{\nabla h_j(\bar{x}) : j \in \mathcal{J}\}$ *linearly independent* $\implies T_X(\bar{x}) = D_X(\bar{x})$. *Among all these conditions this is the only local one.*

It goes without saying that we cannot check all the directions in order to exclude the nasty pathological cases.

Definition 4.13.7 (Dual cone)**.** *Let D_X be a* ***polyhedral cone*** $\mathcal{C} = \{d \in \mathbb{R}^n \ : \ Ad \leq 0\}$*, for some* $A \in \mathbb{R}^{k \times n}$*.*

We term ***dual cone*** $\mathcal{C}^* = \{c = \sum_{i=1}^{k} \lambda_i A_i \ : \ \lambda \geq 0 \}$*.*

Lemma 4.13.8 (Farka's lemma)**.** *Intuitively, this lemma says that pick a vector: either it belongs to the dual cone or there exists a vector in the polyhedral cone which has a negative scalar product with it.*

Equivalently, either $c \in \mathcal{C}^*$ *or* $c \notin \mathcal{C}^*$*.*

More formally, either $\exists \lambda \geq 0$ *s.t.* $c = \sum_{i=1}^{k} \lambda_i A_i$ *or* $\exists d$ *s.t.* $Ad \leq 0 \wedge < c, d >> 0$*.*

Theorem 4.13.9 (Karush-Kuhn-Tucker conditions)**.** *Let us assume that we found an optimal solution* x_* *and the constraints qualification holds.*

Then $\exists \lambda \in \mathbb{R}_+^{|\mathcal{I}|}$ *and* $\mu \in \mathbb{R}^{|\mathcal{I}|}$ *such that*

$$\nabla f(x) + \sum_{i \in \mathcal{A}(x)} \lambda_i \nabla g_i(x) + \sum_{j \in \mathcal{J}} \mu_j \nabla h_j(x) = 0$$

It is interesting to notice that we did not impose $\mu \geq 0$. Let us take an equality constraint $h_j(x) = 0$. This is equivalent to write $h_j(x) \leq 0 \ \wedge h_j(x) \geq 0$, thus leading to two different multipliers, say λ_j^+ and λ_j^-.

The term of the sum concerning h_j looks like this $\lambda_j^+ \nabla h_j(x_*) - \lambda_j^- \nabla h_j(x_*) = (\lambda_j^+ - \lambda_j^-) \cdot \nabla h_j(x_*)$, where both λ_j^+ and λ_j^- are ≥ 0, hence their difference (denoted by μ_j) may be either positive or negative.

Fact 4.13.10. *The Karush-Kuhn-Tucker conditions are also written as:*

FEASIBILITY: $x \in X \equiv g_i(x) \leq 0 \ \ i \in \mathcal{I}, \ h_j(x) = 0 \ \ j \in \mathcal{J}$

KKT-G: $\nabla f(x) + \sum_{i \in \mathcal{I}} \lambda_i \nabla g_i(x) + \sum_{j \in \mathcal{J}} \mu_j \nabla h_j(x) = 0$

COMPLEMENTARITY SLACKNESS: $\sum_{i \in \mathcal{I}} \lambda_i g_i(x) = 0$

where the second and the third equations formalize the definition of KKT, where the terms of the first sum should be summed only if their contraints are active.

Fact 4.13.11. *Let (P) be a convex problem. In this case, if the Karush-Kuhn-Tucker conditions hold then x is a global optimum.*

4.14 6th of December 2018 — A. Frangioni

4.14.1 Duality

Do you recall?

The KKT-G condition:

$$\nabla f(x) + \sum_{i \in \mathcal{I}} \lambda_i \nabla g_i(x) + \sum_{j \in \mathcal{J}} \mu_j \nabla h_j(x) = 0$$

We would like to understand what is the function of which this is the gradient.

Definition 4.14.1 (Lagrangian function)**.** *Let (P) be our minimum problem over $f : \mathbb{R}^n \to \mathbb{R}$. We term **Lagrangian function** the following:*

$$L(x\ \lambda,\ \mu) = f(x) + \sum_{i \in \mathcal{I}} \lambda_i g_i(x) + \sum_{j \in \mathcal{I}} \mu_j h_j(x)$$

Fact 4.14.1. *A necessary condition for optimality is that the gradient of the Lagrangian function is 0. Formally, $\nabla L(x; \lambda, \mu) = 0$.*

A first approach to use the Lagrangian function would be taking λ and μ such that the Langrangian is positive semidefinite.

But this is a too strict requirement, in fact we impose:

Definition 4.14.2 (Critical cone)**.** *Let us assume (x, λ, μ) satisfies (KKT). We define the **critical cone** as*

$$C(x,\ \lambda,\ \mu) = \left\{ d \in \mathbb{R}^n\ :\ \begin{array}{ll} < \nabla g_i(x), d >= 0 & i \in \mathcal{A}(x)\ s.t.\ \lambda_i^* > 0 \\ < \nabla g_i(x), d >\leq 0 & i \in \mathcal{A}(x)\ s.t.\ \lambda_i^* = 0 \\ < \nabla h_j(x), d >= 0 & i \in \mathcal{J} \end{array} \right\}$$

Theorem 4.14.2. *Let us assume we have a point $(x,\ \lambda,\ \mu)$ that satisfies the Karush-Khun-Tucker conditions and satisfies the linear independence of the contraints. If x is local optimum then $d^T \nabla^2_{xx} L(x,\ \lambda,\ \mu) d \geq 0\ \forall\ d \in C(x,\ \lambda,\ \mu)$.*

Informally, if the hypothesis holds, then the Hessian of the Lagrangian function is $\succeq 0$ on the critical cone.

Observation 4.14.1. *$(x,\ \lambda,\ \mu)$ satisfies (KKT) $\wedge$ $\nabla^2_{xx} L(x,\ \lambda,\ \mu) \succ 0$ on $C(x,\ \lambda,\ \mu)$ then x local optimum.*

We would like to say something more about λ and μ. Until now we considered the Lagrangian as a function of x, but what if we consider the Lagrangian in terms of λ and μ?

4.14.2 Lagrangian duality

Definition 4.14.3 (Lagrangian relaxation)**.** *Let us consider the Lagrangian function. We term* ***Lagrangian relaxation*** *the function where we fixed λ and μ and minimize on x:*

$$\psi(\lambda,\ \mu) = \min\{L(x,\ \lambda,\ \mu)\ :\ x \in \mathbb{R}^n\}$$

The relaxation leads to an unconstrained problem and we learnt how to solve one of those.

This Lagrangian relaxation leads to the definition of **lagrangian dual** ψ.

Property 4.14.3. *The dual function has the following properties:*

- *ψ is concave, but $\psi(\lambda, \mu) = -\infty$;*
- *ψ is non differentiable, although f, g_i and h_j are;*
- *Let $\bar{x}$ be optimal in $(R_{\lambda,\mu})$, then $[\sum_{i\in\mathcal{I}} \nabla g_i(\bar{x}) + \sum_{j\in\mathcal{J}} \nabla h_i(\bar{x})] \in \partial\psi(\lambda, \mu)$*
- *$\forall$ fixed $\lambda \geq 0, \mu, \bar{x} \in X$ $\psi(\lambda, \mu) = \min_x L(x, \lambda, \mu) \leq L(\bar{x}, \lambda, \mu) \leq f(\bar{x})$*

$$\psi(\lambda,\mu) = \begin{pmatrix} \lambda_1 g_1(\bar{x}) \\ \vdots \\ \lambda_k g_k(\bar{x}) \\ \lambda_1 \mu_1(\bar{x}) \\ \vdots \\ \lambda_p \mu_p(\bar{x}) \end{pmatrix} \text{ is such that } \begin{pmatrix} g_1(\bar{x}) \\ \vdots \\ g_k(\bar{x}) \\ \mu_1(\bar{x}) \\ \vdots \\ \mu_p(\bar{x}) \end{pmatrix} \text{ belongs to the supergradient.}$$

Theorem 4.14.4 (Weak duality)**.** *$\forall$ fixed $\lambda \in \mathbb{R}^+ \cup \{0\}, \mu \in \mathbb{R}$ such that $\psi(\lambda,\ \mu) \leq v(P)$ and let us take any feasible $\bar{x} \in X$ and $g(\bar{x}) \leq 0$, $h(\bar{x}) = 0$.*

It can be proved that $\psi(\lambda, \mu) = \min_x L(x, \lambda, \mu) \leq L(\bar{x}, \lambda, \mu) \leq f(\bar{x})$.

Proof. $L(x, \lambda, \mu) = f(x) + \sum_i \lambda_i g_i(x) + \sum_j \mu_j h_j(x)$. Let us assume we have some $\bar{\lambda}, \bar{\mu}, \bar{x}$, where $\bar{x}$ is feasible (aka $g_i(\bar{x}) \leq 0$ and $h_i(\bar{x}) = 0$ and $\bar{\lambda} \geq 0$).

The value of the dual function is

$\psi(\bar{\lambda}, \bar{\mu}) = \min_x\{f(x)+\sum_i \lambda_i g_i(x)+\sum_j \mu_j h_j(x) \leq f(\bar{x})+\sum_i \bar{\lambda}_i g_i(\bar{x})+\cancel{\sum_j \bar{\mu}_j h_j(\bar{x})}$,

where the last term is cancelled, since $h_i(\bar{x}) = 0$.

On the other hand, $\sum_i \bar{\lambda}_i g_i(x) \leq 0$, which implies that $f(x) \leq 0$. □

Observation 4.14.2. *This theorem gives us the way to prove that a point is an optimum. Let us assume that we found $\lambda_* \leq 0$, x_* and μ_* such that an equality holds: $\psi(\lambda_*, \mu_*) = f(x_*)$. The value of ψ gives us the information about how far we are from the optimum. Let us assume we have $\bar{\lambda}$, $\bar{\mu}$ and $\bar{x}$ then $\psi(\bar{\lambda}, \bar{\mu}) \leq f(\bar{x}) \leq \psi(\bar{\lambda}, \bar{\mu}) + \varepsilon$ and this tells that we are far from the optimum of a factor ε.*

If everything in the original function was convex then the Lagrangian is convex and the computations are easy.

At this point we want to find the biggest lower bound and it may be computed since the function is concave.

$\max\{ \psi(\lambda , \mu) \ : \ \lambda \in \mathbb{R}_+^{|\mathcal{I}|} \ , \ \mu \in \mathbb{R}^{|\mathcal{J}|} \}$

Notice that the constraints on λ are very easy, so the problem may be considered somehow uncontrained.

We can use local methods to compute the maximum of this function.

If we are able to compute the gradient of ψ then we have the supergradients of the fucntion f.

Is (P) equal to (D)? Yes, provided that everything is convex. In general the Lagrangian gives a lowerbound.

$v(D) \leq v(P)$

Example 4.14.1. *Let us take a concave objective function* $\min\{-x^2 \ : \ 0 \leq x \leq 1\}$*, such that its Lagrangian function is* $L(x, \lambda) = -\mathbf{x}^2 + \lambda_1(x-1) - \lambda_2 x$*. It goes without saying that the Lagrangian function is unbounded below (upside-down parabola).*

Formally, $\psi(\lambda) = \min_{x \in \mathbb{R}} L(x, \lambda) = -\infty, \ \forall \lambda \in \mathbb{R}^2$*, which means that the Lagrangian dual is* $v(D) = -\infty < v(P) = -1$*.*

Theorem 4.14.5. *Let us assume that* f*,* g *and* h *are convex, and constraints qualification holds. If the problem has an optimum then the value of the primal and the value of the dual are the same.*

Formally, $T_X(x_*)D_X(x_*) \Rightarrow v(D) = v(P)$*.*

Proof. We are assuming x_* is an optimum, so the necessary conditions hold, hence we have the Karush Khun Tucker conditions, then we can find (λ^*, μ^*) that satisfy the KKT with x.

Then our claim is that (λ^*, μ^*) is an optimal solution of the dual (D) and the value of the dual is equal to the value of the primal.

x_* is a stationary point for the Lagrangian function, since it satifies the KKT conditions then x_* is exactly the minimum, since the Lagrangian function is convex.

Hence, the value of the dual $v(D) \geq \psi(\lambda - *, \mu_*) = L(x_*, \lambda_*, \mu_*) = f(x_*) = v(P) \geq v(D)$. □

How many solution may $\psi(\lambda, \mu)$ have? In principle many, but what if f is strongly convex, than everything in the Lagrangian is strongly convex, then he solution of ψ is unique and it is also differentiable, bu typically not two times differentiable. At this point the Lagrangian dual has a single solution and the optimum of the Lagrangian dual corresponds to an optimum for f.

We only translated a minimum problem on convex functions to another minimum problem on other convex functions. We will see soon that in some cases this approach is advantageous in others it is not.

4.14.3 Specialized dual

Linead programs

(P) $\min\{\, cx \,:\, Ax \geq b \,\}$

Lagrangian function: $L(x, \lambda) = cx + \lambda(b - Ax) = \lambda b + (c - \lambda A)x$

Dual function:

$$\psi(\,\lambda\,) = \min_{x \in \mathbb{R}^n} L(\,x\,,\,\lambda\,) = \begin{cases} -\infty & \text{if } c - \lambda A \neq 0 \\ \lambda b & \text{if } c - \lambda A = 0 \end{cases}$$

Since we can find a minimum only on a function of the second case we have that:

(D) $\max\,\{\,\psi(\,\lambda\,) \,:\, \lambda \geq 0\,\} \equiv \max\,\{\,\lambda b \,:\, \lambda A = c\,,\ \lambda \geq 0\,\}$

Since the lagrangian is far from having an unique optimal solution the dual does not have a unique maximum so the do not match usually.

Quadratic programs

Notice that thwe quadratic case is simpler than the linear case.

(P) $\min\,\{\,\frac{1}{2}\|\,x\,\|_2^2 \colon\, Ax = b\,\}$ (linear least-norm solution)

$L(\,x\,,\,\mu\,) = \frac{1}{2}\|\,x\,\|_2^2 + \mu(\,Ax - b\,)$, $\nabla_x L = x + \mu A = 0 \iff x = -\mu A$, which is stricltly convex since x^2 is striclty convex.

$\psi(\mu) = \min_{x \in \mathbb{R}^n} L(\,x\,,\,\mu\,) = L(\,-\mu A\,,\,\mu\,) = -\frac{1}{2}\mu^T(AA^T)\mu - \mu b$

Then the Lagrangian dual (D) $\max\{-\frac{1}{2}\mu^T(AA^T)\mu - \mu b \,:\, \mu \in \mathbb{R}^m\}$

We can generalize with quadratic functions for general problems:

Strictly convex QP: (P)

$\min\,\{\,\frac{1}{2}x^TQx + qx \,:\, Ax \geq b\,\}$, $Q \succ 0 \Longrightarrow$ (D) $\max\,\{\,\lambda b - \frac{1}{2}v^TQ^{-1}v \,:\, \lambda A - v = q\,,\ \lambda \geq 0\,\}$ strong duality $\equiv v(P) = v(D)$ (almost) always holds

Conic program

We can do duals of things that are not quadratic programs. Sometimes we want to have non linear things to be able to draw different shapes.

Definition 4.14.4 (Conic program)**.** *We term* ***conic program*** $\min\{cx \,:\, Ax \,\geq_K\, b\}$*, where* $x \geq_K y \equiv x - y \in K$*, where* K *is a convex cone.*

Example 4.14.2. *It is easy to see that the* $\geq$ *constraints we saw before are a special case of conic program, where the conic is* $\mathbb{R}^n_+$*.*

$-x_1 - 2x_2 \leq 2$

$x_1 + x_2 \geq 1$

$2x_1 - x_2 \geq 0$

Let us write $-x_1 - 2x_2 - 2 = s_0$*,* $x_1 + x_2 - 1 = s_1$ *and* $2x_1 - x_2 = s_2$*, such that* $s_0, s_1, s_2 \geq 0$*.*

We have a mapping from R^n *(where the variables live) to* $\mathbb{R}^m$ *(where the constraints live) and* $Ax - b \in K$*.*

Rather than writing $s_0, s_1, s_2 \geq 0$ *we could write* $s_0 \geq \sqrt{s_1^2 + s_2^2}$ *and this would still be a conic program.*

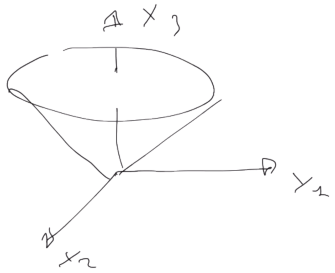


FIGURE 4.46: $x^3 \geq \sqrt{x_1^2 + x_2^2}$ is a convex cone.

The idea is to hide the non linear part in the cone, in particular in $\geq_K$. Let us see what happens to cones depending on the function.

There are thrww interesting cones:

- $K = \mathbb{R}^n_+ \equiv$ sign constraints $\equiv$ Linear Program;
- $K = \mathbb{L} = \{ x \in \mathbb{R}^n \, : \, x_n \geq \sqrt{\sum_{i=1}^{n-1} x_i^2} \} \equiv$ Second-Order Cone (or Lorentz cone) Program, that generalizes linear programs;
- $K = \mathbb{S}_+ = \{ A \succeq 0 \} \equiv$ "$\succeq$" constraints $\equiv$ SemiDefinite Program.

Given the problem that looks like linear except for the cone.

Definition 4.14.5 (Conic dual)**.** *Conic dual: (D)* $\max\{ yb \, : \, yA = c \, , \, y \geq_{K^D} 0 \}$*, where* $K^D = \{ z \, :< z, x >\geq 0 \ \ \forall \, x \in K \}$ *is a dual cone.*

Sometimes constraints qualification does not hold, since the conic program is not linear sometimes.

Explicit form of second order cone program (SOCP) ("explicit data" D_i, d_i, p_i, q_i) $\min\{cx \ : \ \|D_i x - d_i\|_2 \leq p_i x - q_i \ i = 1, \ldots, m\}$.

It collapses to a linear program for $D_i = 0$ and $d_i = 0$.

It turns out that the dual has just this explicit form:

$$\max\{\sum_{i=1}^{m} \lambda_i d_i + \nu_i q_i \ : \ \sum_{i=1}^{m} \lambda_i D_i + \nu_i p_i = c, \ \|\lambda_i\|_2 \leq \nu_i, \ i = 1, \ldots, m\}$$

where the nu_i are the dual variables of the rightmost part, while the λ_i are the dual variables of the leftmost part.

We can write the explicit form of semidefinit problems as $\min\{cx \ : \ \sum_{i=1}^{n} x_i A^i \succeq B\}$, where $A^i, B \in M(k, \mathbb{R})$.

There is an even mroe general form of duality, that we are going to introduce next.

4.14.4 Fenchel's duality

Definition 4.14.6 (Fenchel's conjugate)**.** *We denote* ***Fenchel's conjugate*** *of* f $f^*(z) = sup_x\{zx - f(x)\}$.

We can observe that f^* is always convex, even if f is not and it is closed if f is.

The following functions are such that f^* can be computed easily:

1. $f(x) = \frac{1}{2}\|x\|_2^2 \Longrightarrow f^*(z) = \frac{1}{2}\|z\|_2^2$ (only function s.t. $f^* = f$);
2. $(\|\cdot\|_1)^*(z) = ß_{\mathcal{B}_\infty(0,1)}(z)$, $(\|\cdot\|_\infty)^*(z) = ß_{\mathcal{B}_1(0,1)}(z)$;
3. $f(x) = \max\{g_i x - \alpha_i \ i \in I\} \Longrightarrow f^*(z) = \min\{\sum_{i\in I}\alpha_i\theta_i \ : \ \sum_{i\in I} g_i\theta_i = z,\ \sum_{i\in I}\theta_i = 1\ ,\ \theta_i \geq 0 \ i \in I\}$.

Fact 4.14.6. *If we are minimizing the sum of two convex functions (P)* $\min\{f(x)+g(x)\}$*, this problem is the same of maximizing the sum of "almost" the two conjugates: (D)* $-\min\{f^*(z) + g^*(-z)\}$.

4.15 12th of December 2018 — A. Frangioni

4.15.1 Quadratic problem with linear equality constraints

Let A be a matrix in $M(m, n, \mathbb{R})$, where $m < n$ (otherwise the system is either fully determined or impossible) and $rk(A) = m$. The constrained quadratic problem may be written as

$$\min \left\{ \frac{1}{2} x^T Q x + qx \ : \ Ax = b \right\}$$

where $Q \succeq 0$.

A way to solve this problem is through Karusch Kuhn Tucker system:

$$\begin{matrix} \text{(a)} \\ \text{(b)} \end{matrix} \begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \mu \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix} \tag{4.15.1}$$

where the first row (a) says that the gradient is a linear combination of the normals of the gradient of the constraints and the second one is just feasibility.

Everything is linear here. This system is symmetric, although indefinite, because it contains many 0s.

We are left with solving the KKT system:

Reduced KKT: in this method we first add the hypothesis of non-singularity to the matrix Q so we get the following

$$\begin{cases} Qx + A^T \mu = -q & \text{(a)} \\ Ax = b & \text{(b)} \end{cases}$$

Multiply (a) by AQ^{-1}

$$\begin{cases} \cancel{AQ^{-1}}\cancel{Q}x + AQ^{-1}A^T \mu = -AQ^{-1}q & \text{(a)} \\ Ax = b & \text{(b)} \end{cases}$$

Multiply (b) by -1 and add to it (a)

$$\begin{cases} Ax + AQ^{-1}A^T \mu = -AQ^{-1}q & \text{(a)} \\ \cancel{Ax} + AQ^{-1}A^T \mu - \cancel{Ax} = -AQ^{-1}q - b & \text{(b)} \end{cases}$$

Multiply (a) by A^{-1}

$$\begin{cases} x + Q^{-1}A^T \mu = -Q^{-1}q & \text{(a)} \\ AQ^{-1}A^T \mu = -AQ^{-1}q - b & \text{(b)} \end{cases}$$

Isolate x

$$\begin{cases} x = -Q^{-1}(A^T \mu + q) & \text{(a)} \\ AQ^{-1}A^T \mu = -AQ^{-1}q - b & \text{(b)} \end{cases}$$

Notice that $0 \preceq AQ^{-1}A^T = M \in M(m, \mathbb{R})$ and may be much smaller than the original one, since its size depends on the number of constraints. The issue here is that the matrix M is less sparse than both A and Q.

NULL SPACE METHOD: In this method we need no assumption on Q.

First of all we rearrange the matrix A in order to have a small square matrix A_B and then the rest of the columns (A_N): $A = [A_B,\ A_N]$, $x = [x_B, x_N]$, $det(A_B) \neq 0$.

Replacing A in (b) we get $A_B x_B + A_N x_N = b \iff x_B = {A_B}^{-1} \cdot (b - A_N x_N)$, hence resulting in $x = Dx_N + d$, where

$$d = \begin{bmatrix} b \\ 0 \end{bmatrix},\ D = \begin{bmatrix} -A_B^{-1} A_N \\ I \end{bmatrix} \in M(m, n-m)$$

Notice that D is a basis of the **null space** (or kernel) of A and it is not mandatory to build it like we did above, it is only important to obtain a basis of the kernel.

Let us multiply (a) by D^T and obtain

$$\begin{aligned} D^T(Qx + A^T\mu) &= -D^T q \\ D^T Qx + D^T A^T \mu &= -D^T q \\ D^T Qx + (\cancel{AD})^T \mu &= -D^T q \\ D^T Q(Dx_N + d) &= -D^T q \end{aligned} \tag{4.15.2}$$

Where in the last step we applied the definition $x = Dx_N + d$ and hence, $(D^T QD)x_N = -D^T(Qd + q)$.

We term $H = D^T QD \in M(n-m, \mathbb{R})$ the reduced Hessian of the problem and notice that whenever the number of constraints is close to the number of variables this matrix is very small.

It is important to note that in order to solve an equality contrained problem we can choose either the reduced KKT method or the null space method, depending on the structure of our problem.

4.15.2 Inequality constrained problems

The problem, in this case, is written as follows

$$(P)\, \min\{f(x)\ :\ Ax \leq b\}$$

Projected gradient method

The intuition behind this kind of algorithm is that we are interested in finding as direction for the line search not the opposite of the gradient (because then the step size should be put to 0 if we lie on the boundaries, see Figure 4.47), but it is enough to pick a direction which has a negative scalar product with the gradient.

The rationale is to pick the direction that minimizes the norm of the difference with the gradient. Formally

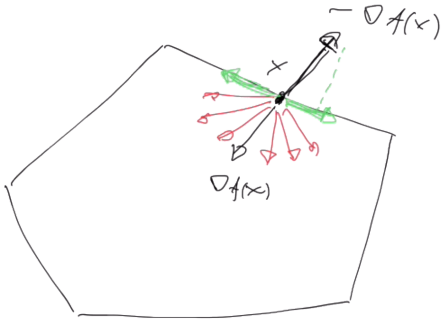


FIGURE 4.47: Geometric idea of how to choose the direction in case of lying on the boundary.

$$d = argmin\{\|\nabla f(x) - d\|^2 : d \in \mathcal{D}_X(x)\}$$

where $\mathcal{D}_X(x)$ is the feasible cone. Notice that if the difference between the gradient and the direction is 0 it means that we can stop, since the descent direction would bring us outside the feasible set.

A more formal definition of the feasible cone $\mathcal{D}_X(x) = \{d \in \mathbb{R}^n \ : \ A_{\mathcal{A}(x)}d \leq 0\}$, where $\mathcal{A}(x)$ is the set of all the active constraints.

For sake of clarity we denote $\bar{A} = A_{\mathcal{A}(x)}$.

At this point we are ready to project the problem in such a way that inequality contraints become equality constraints:

$$\min \left\{ \frac{1}{2}\|\nabla f(x) + d\|^2 = \frac{1}{2}d^T I d + \nabla f(x) d \ : \ \bar{A}d = 0 \right\}$$

Note that in this case the Hessian is the identity matrix $I \succ 0$, hence the KKT conditions become simpler:

$$\begin{cases} Qx + A^T\mu = -q \iff d + A^T\mu = -\nabla f(x) \\ Ad = 0 \end{cases}$$

since $x = d, A = I, b = \nabla f(x)d$, so we get

$$\begin{aligned} d &= -\nabla f(x) - A^T\mu \\ \cancel{Ad} &= -A\nabla f(x) - AA^T\mu \\ AA^T\mu &= -A\nabla f(x) \end{aligned} \tag{4.15.3}$$

resulting in

$$\begin{cases} Qx + A^T\mu = -q \iff d + A^T\mu = -\nabla f(x) \\ Ad = 0 \end{cases}$$

Therefore, we need to solve a system in μ and then we have the direction. If the number of active contraints is small, solving the system is very fast.

Fact 4.15.1. *We can restrict to solve*

$$\begin{cases} \mu = -(\bar{A}\bar{A}^T)^{-1}\bar{A}\nabla f(x) \\ d = (I - \bar{A}^T(\bar{A}\bar{A}^T)^{-1}\bar{A})(-\nabla f(x)) \end{cases}$$

Proof. $\bar{A}$ has full row rank, then is non singular and may be inverted.

$$\begin{cases} \mu = -(\bar{A}\bar{A}^T)^{-1}\bar{A}\nabla f(x) \\ d = -\nabla f(x) + \bar{A}^T(\bar{A}\bar{A}^T)^{-1}\bar{A}\nabla f(x) \end{cases} \iff \begin{cases} \mu = -(\bar{A}\bar{A}^T)^{-1}\bar{A}\nabla f(x) \\ d = (I - \bar{A}^T(\bar{A}\bar{A}^T)^{-1}\bar{A})(-\nabla f(x)) \end{cases}$$

□

If the set of active contraints contains some linear dependent ones, we take the maximal subset of our constraints such that the matrix $\bar{A}$ has full rank.

This procedure is formalized in Algorithm 4.15.1, where at each stage we remove one of the contraints that lead to a negative μ, keeping a set of linearly independent constraints.

In the 14-th line of the algorithm we find that our step size is takes as the minimum step size among the ones that satisfy some subsets of the constraints.

ALGORITHM 4.15.1 Pseudocode for projected gradient method for quadratic functions.

```
1: procedure PGM(f, A, b, x, ε)
2:     for (; ;) do
3:         B ← maximal ⊆ A(x) s.t. rank(A_B) = |B|;
4:         for (; ;) do
5:             d ← (I − A_B^T (A_B A_B^T)^{-1} A_B)(−∇f(x));
6:             if < ∇f(x), d >≤ ε then
7:                 μ_B ← −(A_B A_B^T)^{-1} A_B ∇f(x);
8:                 μ_i ← 0 ∀ i ∉ B;
9:                 if μ_B ≥ 0 then return
10:                end if
11:                h ← min{i ∈ B : μ_i < 0};
12:            end if
13:            ᾱ ← min{α_i = (b_i − A_i x)/A_i d : A_i d > 0, i ∉ B};
14:            x ← x + αd;
15:            if ᾱ > 0 then
16:                break;
17:            end if
18:            k ← min{ i ∉ B : A_i d > 0 : α_i = 0 };
19:            B ← B ∪ { k };
20:        end for
21:        α ← Line_Search(f, x, d, ᾱ);
22:        x ← x + αd;
23:    end for
24: end procedure
```

Fact 4.15.2. *The following holds:*

1. $d = 0 \wedge \mu \geq 0 \implies x$ *optimal (from KKT);*
2. $d = 0 \wedge \exists h \in B$ *s.t.* $\mu_h < 0 \implies \exists x' \in \{ x \in \mathbb{R}^n : A_{B\setminus\{h\}} x = b_{B\setminus\{h\}}, A_h x \leq b_h \}$ *s.t.* $f(x') < f(x)$*;*
3. $d \neq 0$ *descent direction:* $H = I - A_B{}^T [A_B A_B{}^T]^{-1} A_B$ *symmetric and idempotent* $HH = H^T H = H \implies < d, \nabla f(x) >< 0$*.*

Proof.

trivial;

- The intuition is that if we remove h from B next time we will have a descent direction.

  $\exists d$ s.t. $A_{B\setminus\{h\}} d = 0 \wedge A_h d < 0 \implies < \nabla f(x), d >=< -\mu A_B, d >= -\mu_h A_h d < 0$;

- $< d, \nabla f(x) >=< -H\nabla f(x), \nabla f(x) >= -\nabla f(x)^T H \nabla f(x) = -(H\nabla f(x))^T H \nabla f(x)$.

□

Once we found the optimal face we move inside that face and we are dealing with a steepest descent, which is slow.

At a certain point of the execution the set B of the active contstraints will stabilize and become the one of the optimal solution.

At this point the smart thing to do is to use the KKT conditions, since we know the set of active constraints. This idea is pursued in Section 4.15.2.

Projected gradient method with box constraints

Given the non linear minimum problem of the following shape

$$\text{(P)} \ \min \left\{ f(x) \ : \ l \leq x \leq u \right\}$$

a slightly different version of the projected gradient method forces the algorithm to put to 0 the component of the direction which would bring us to go outside the feasible region. This intuition is formalized in Algorithm 4.15.2.

ALGORITHM 4.15.2 Pseudocode for projected gradient method for quadratic functions in the case of box constraints.

```
1: procedure PGMBC(f, l, u, x, ε)
2:     d = −∇f(x);
3:     ᾱ = ∞;
4:     for (i = 1 . . . n s.t. d_i ≠ 0) do
5:         if (d_i < 0) then
6:             if (x_i = l_i) then
7:                 d_i = 0;
8:             else
9:                 ᾱ ← min{ᾱ(x_i − l_i)/d_i};
10:            end if
11:        else
12:            if (x_i = u_i) then
13:                d_i = 0;
14:            else
15:                ᾱ ← min{ᾱ(u_i − x_i)/d_i};
16:            end if
17:        end if
18:        if (< ∇f(x), d >≤ ε) then return
19:        end if
20:        α ← Line_Search(f, x, d, ᾱ);
21:        x ← x + αd;
22:    end for
23: end procedure
```

Notice that we can assume that $l_i < u_i \ \forall i$, because otherwise that component would be fixed.

Active-set method for quadratic programs

Let us be given the following minimum problem

$$\min \left\{ \frac{1}{2} x^T Q x + qx \ : \ Ax \leq b \right\}$$

where an important hypothesis is that the problem is quadratic, once we know $\mathcal{A}(x_*)$.

We start from a certain point, such that the constraints are satisfied as equalties. According to KKT conditions, a solution $\bar{x}$ is optimal if $\bar{x}$ is feasible and $\mu \geq 0$. Otherwise, if μ is not positive we eliminate something from the active set and start again. In the case of x unfeasible, we know the descent direction, we only need to revise the step size.

All this machinery is formalized in Algorithm 4.15.3.

ALGORITHM 4.15.3 Pseudocode for active set method for quadratic functions.

```
1: procedure ASMQP(Q, q, A, b, x, ε)
2:     for (B ← A(x); ;) do
3:         solve (P_B) min{½ x^T Q x + qx : A_B x = b_B} for (x̄, μ̄_B);
4:         if (A_i x̄ ≤ b_i ∀ i ∉ B) then
5:             if (μ_B ≥ 0) then return
6:             end if
7:             h ← min{i ∈ B : μ_i < 0};
8:             B ← B\{h};
9:             continue;
10:        end if
11:        d ← x̄ − x;
12:        ᾱ ← min{α_i = (b_i − A_i x)/A_i d : A_i d > 0, i ∉ B};
13:        x ← x + ᾱ d;
14:        B ← A(x);
15:    end for
16: end procedure
```

Notice that this algorithm allows easy solving of the same problem, under box constraints, because we end up having two sets: L, indexes of variables fixed to the lower bound, and U, indexes of variables fixed to the upper bound.

$\{1, \ldots, n\} \backslash (L \cup U)$ is the set of the indexes of the free variables.

Thanks to this consideration the problem to solve becomes

$$\min \left\{ \frac{1}{2} {x_F}^T Q_{FF} x_F + (q_F + {u_U}^T Q_{UF}) x_F \right\} \ [+ \frac{1}{2} {x_U}^T Q_{UU} x_U + q_U u_U]$$

Frank-Wolfe method

Let us take a non-linear function and the following problem

Supposing f is convex we can make a first order model and minimize it over our constraints.

In this case we use the right constraints and an approximation of the function.

ALGORITHM 4.15.4 Pseudocode for Frank-Wolf method for non linear functions.

```
1: procedure FWM(f, A, b, x, ε)
2:     while (‖∇f(x)‖ > ε) do
3:         x̄ ← argmin {< ∇f(x), y > :  Ay ≤ b};
4:         d ← x̄ − x;
5:         α ← Line_Search(f,  x,  d,  1);
6:         x ← x + αd;
7:     end while
8: end procedure
```

Provided that the computation at line 3 of Algorithm 4.15.4 is performed efficiently, then either $d = 0$ and we are in the optimum or $d > 0$ and we are moving toward the optimum.

The linear model is below the function in the quadratic case, hence we get a lower bound.

4.16 14th of December 2018 — A. Frangioni

In the previous lecture we addressed the problem of linear constrained optimization. Our first approach was to deal very little with constraints (projected gradient method), after a few improvements we took all of them and modify the function (Frank-Wolfe method).

4.16.1 Dual methods for linear constrained optimization

In this class of methods constraints are first class citizens.

Let us be given the following optimization problem:

$$\min\left\{\frac{1}{2}x^TQx + qx \ : \ Ax \leq b\right\}$$

We would like to work with dual feasibility.

$\forall$ fixed $\lambda \geq 0$, this is the lagrangian problem $\psi(\lambda) = \min_x\{\frac{1}{2}x^TQx + qx + \lambda(b - Ax)\} \leq v$ and the Lagrangian dual is the following

$$(D)\ \max\{\psi(\lambda) \ : \ \lambda \geq 0\}$$

which is equivalent to the primal problem.

The solution of the Lagrangian problem is **unique** and this is due to the fact that ψ is concave and $Q \succ 0$. The optimal solution is then $x(\lambda) = Q^{-1}(\lambda A - q)$.

The function is differentiable in the solution, because we have only one sub (or super) gradient which is $\nabla\psi(\lambda) = b - Ax(\lambda)$.

At this point we only need to solve the dual problem, which has the shape of a box constrained optimization, where the box has only one boundary and this can be solved via quasi-Newton methods.

Notice that $\psi \notin C^2$ so the Hessian is not defined.

This dual approach is advantageous in the case of a small number of constraints, because the size of the problem decreases. For example, in the case of one constraint, the Lagrangian dual becomes a problem in one variable, hence solvable through a line search.

In this method the degenerate case (more than one constraint active at a time) is not an issue.

If the quadratic function is convex we may use quasi Newton method. Otherwise the global optimality is not guaranteed and may be used if we accept not to be able to solve the original problem, but only to find a lower bound.

Separable problems and partial dual

Let us assume that our constraints are separable, which means that it is not mandatory to work with all of them, but they can be splitted into constraints of groups of variables.

$$\min\{f(x) \ : \ Ax \leq b,\ Ex \leq d\}$$

We can decide to use a **partial dual**, writing the Lagrangian problem picking only some constraints that we chose:

$$\psi(\lambda) = \min_x \{f(x) + \lambda(b - Ax) \ : \ Ex \leq d\}$$

The Lagrangian dual method may be better than projected gradient or worse and it depends on the instance.

In the dual approaches we can't move inside the feasible solution. We find an optimum for the dual, which surely breaks feasibility. Then, if the variable is above the upperbound it gets decreased to the upper bound, otherwise if it is below the lower bound it takes the value of the lower bound.

This way we get an upperbound for the function to be minimized.

4.16.2 Primal/dual methods or barrier methods

This kind of methods are designed to overcome the cons of dual approaches, namely the fact that ψ does not have the Hessian (and this creates problems to quasi Newton method) and the fact that x is not feasible until the end.

At the same time this methods keep the unconstrained property of the Lagrangian dual.

Barrier function and central path

The rationale behind this algorithm is to minimize a function which penalizes the value of the original function when the solution is getting closer and closer to the boundaries of the feasible set.

$$(P_\mu) \ \min\{f_\mu(x) = f(x) - \mu \sum_{i=1}^{m} \log(b_i - A_i x)\}$$

The parameter μ is there to weight the proximity to the boundary.

Property 4.16.1.

- *If f is convex, f_μ is strictly convex;*
- *If $f \in \mathcal{C}^2$ then $f_\mu \in \mathcal{C}^2$, since $\log \in \mathcal{C}^\infty$;*
- *$\forall \mu \ \exists! \ x_\mu$ optimal of (P_μ), since $\mu \sum_{i=1}^{m} \log(b_i - A_i x)$ is striclty convex;*
- *As $\mu \to 0$ x_μ converges to the analytic center of the optimal face. An example of this behaviour may be seen in Figure 4.48.*

Another interesting property is that, since the barrier function is **self concordant** x^i gets "close" to $x(\mu^i)$ in very few Newton's steps.

In one step x^{i+1} is "much closer" to $x(\mu^i)$ than x^i was and x^{i+1} is "close" to $x(\mu^{i+1})$, with $\mu^{i+1} \ll \mu^i$ (more formally, $\mu^{i+1} = \tau\mu^i$, $\tau < 1$).

This behaviour may be observed in Figure 4.49

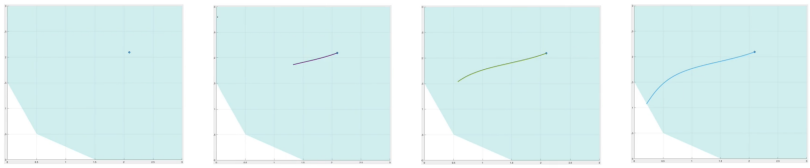


FIGURE 4.48: The trajectory converges to the optimal solution of the problem, when $\mu \to 0$.

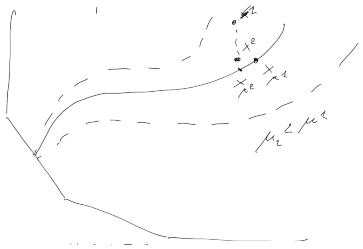


FIGURE 4.49: The dotted line represents a region where the Newton method is very efficient. We are starting from a point x_1, which belongs to that region and we want to move towards x_{μ^1}. At next iterate x_2 is closer to x_{μ^2} than the current iterate.

The convergence is exponential if τ is very small, but these iterations are very costly, because the Hessian changes at each step, hence it needs to be recomputed.

Let us focus on computing the **Newton's step**.

First, we write the Karusch-Kuhn-Tucker conditions:

PRIMAL FEASIBILITY: $Ax + s = b,\ s \geq 0$;

DUAL FEASIBILITY: $Qx + \lambda A = -q,\ \lambda \geq 0$;

COMPLEMENTARY SLACKNESS: $\lambda_i s_i = 0,\ i = 1, \ldots, m$.

We can write a slackened version of KKT, imposing $\lambda_i s_i = \mu,\ i = 1, \ldots, m$, where $\mu \in \mathbb{R}^n$ and should be decreased over iterations until it gets closer engough to 0.

Let us construct $\Lambda,\ S \in \text{Diag}(m, \mathbb{R})$ such that the diagonal is made of λ_i and s_i respectively.

At this point, we rewrite the problem in terms of the displacement from the fixed current point we are in:

- $x \to x + \Delta x$
- $s \to s + \Delta s$
- $\lambda \to \lambda + \Delta \lambda$

The KKT system becomes:

PRIMAL FEASIBILITY: $Ax + A\Delta x + s + \Delta s = b,\ s \geq 0$;

DUAL FEASIBILITY: $Qx + Q\Delta x + \lambda A + \Delta\lambda A = -q,\ \lambda \geq 0$;

COMPLEMENTARY SLACKNESS: $\lambda_i s_i + \lambda_i \Delta s_i + s_i \Delta\lambda_i + \Delta\lambda_i \Delta s_i = \mu,\ i = 1, \ldots, m$.

In this new system of coordinates the first two KKT remain linear, while the third one is no longer linear ($\Delta\lambda_i \Delta s_i$).

$$\begin{bmatrix} Q & A^T & 0 \\ A & 0 & I \\ 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta s \end{bmatrix} \overset{(1)}{=} \begin{bmatrix} -(Qx+q) - \lambda A \\ b - Ax - s \\ \mu u - \Lambda S u - \Delta\Lambda\Delta S u \end{bmatrix} \approx \begin{bmatrix} 0 \\ 0 \\ \mu u - \Lambda S u \end{bmatrix} \tag{4.16.1}$$

Where (1) holds since $\Delta\lambda A = A^T \Delta\lambda^T$, although $\Delta\lambda$ is written without the "transpose" sintax to ease notation.

Notice that $u = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^m$ and has the purpose of adjusting dimension:

$$S\Delta\lambda = \begin{bmatrix} s_1\Delta\lambda_1 \\ \vdots \\ s_m\Delta\lambda_m \end{bmatrix} \in \mathbb{R}^m;\ \Lambda\Delta s = \begin{bmatrix} \lambda_1\Delta s_1 \\ \vdots \\ \lambda_m\Delta s_m \in \mathbb{R}^m \end{bmatrix} \in \mathbb{R}^m;\ \mu u = \begin{bmatrix} \mu \\ \vdots \\ \mu \end{bmatrix} \in \mathbb{R}^m;$$

$$\Lambda S u = \begin{bmatrix} -s_1\lambda_1 \\ \vdots \\ -s_m\lambda_m \end{bmatrix} \in \mathbb{R}^m;\ \Delta\Lambda\Delta s u = \begin{bmatrix} -\Delta\lambda_1\Delta s_1 \\ \vdots \\ -\Delta\lambda_m\Delta s_m \in \mathbb{R}^m \end{bmatrix} \in \mathbb{R}^m;$$

The Newton method can be applied if we discard the non-linear part (written in red), pretending it does not exist. Notice that vector u is the vector of all 1s.

4.16.3 Primal-dual interior point method

This method is based on the observation that we can solve the dual problem

$$(D) \quad \max\{-\lambda b - \frac{1}{2}x^T Q x \ :\ Qx + \lambda A = -q,\ \lambda \geq 0\}$$

thus obtaining both a lower and upper bound for the solution x.

We term **complementarity gap** $(\frac{1}{2}x^T Q x + qx) - (-\lambda b - \frac{1}{2}x^T Q x) = \lambda s = \mu$.

Once we found a solution for Equation (4.16.1), we perform a step and compute a new couple of primal and dual solutions and reduce the gap μ.

We are left with solving Equation (4.16.1). The trick is to express one between Δs and $\Delta\lambda$ as a linear combination of the other.

For example, let us take the third line of Equation (4.16.1):

$$\begin{aligned} 0\Delta x + S\Delta\lambda + \Lambda\Delta s &= \mu u - \Lambda S u \\ \Lambda\Delta s &= \mu u - \Lambda S u - S\Delta\lambda \\ \Delta s &= \Lambda^{-1}\mu u - \cancel{\Lambda^{-1}\Lambda} S u - \Lambda^{-1} S\Delta\lambda \\ \Delta s &= \Lambda^{-1}\mu u - S u - \Lambda^{-1} S\Delta\lambda \\ \Delta s &= \Lambda^{-1}\cdot(\mu u - S\Delta\lambda) - S u \\ \Delta s &= \Lambda^{-1}\cdot(\mu u - S\Delta\lambda) - s \end{aligned} \tag{4.16.2}$$

We obtain the modified normal equations (or KKT system)

$$\begin{bmatrix} Q & A^T \\ A & -\Lambda^{-1}S \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} 0 \\ s - \mu\Lambda^{-1}u \end{bmatrix} \tag{4.16.3}$$

where the last row is derived working on the second row of Equation (4.16.1) ($A\Delta x + 0\Delta\lambda + I\Delta s = 0 \Leftrightarrow A\Delta x = -\Delta s$), substituting Equation (4.16.2):

$$\begin{aligned} A\Delta x - [\Lambda^{-1}S]\Delta\lambda &= -\Delta s - [\Lambda^{-1}S]\Delta\lambda \\ &= -\Lambda^{-1}(\mu u - S\Delta\lambda) + s - \Lambda^{-1}S\Delta\lambda \\ &= -\Lambda^{-1}\mu u + \cancel{\Lambda^{-1}S\Delta\lambda} + s - \cancel{\Lambda^{-1}S\Delta\lambda} \\ &= s - \mu\Lambda^{-1}u \end{aligned} \tag{4.16.4}$$

With respect to normal equations of **??**, we have in position $(2,2)$ a quantity $(-\Lambda^{-1}S)$, which is not 0, but it is the opposite of a striclty positive definite matrix.

A possible approach for solving this system is making some calculations and obtain something of the shape of reduced KKT (see Section 4.15.1).

$$\begin{cases} Q\Delta x + A^T\Delta\lambda = 0 \\ (Q + A^T\Lambda S^{-1}A)\Delta x = A^T(\lambda - \mu S^{-1}u) \end{cases} \tag{4.16.5}$$

where the first set of equations follows from the expansion of the first row of the KKT system (Equation (4.16.3)) and the second one is obtained taking the same row of the same system and substituting the value of $\Delta\lambda$ as follows:

$$\begin{aligned} A\Delta x - \Lambda^{-1}S\Delta\lambda &= s - \mu\Lambda^{-1}u \\ \Lambda^{-1}S\Delta\lambda &= A\Delta x - s + \mu\Lambda^{-1}u \\ \Delta\lambda &= (\Lambda^{-1}S)^{-1}A\Delta x - (\Lambda^{-1}S)^{-1}s + (\Lambda^{-1}S)^{-1}\mu\Lambda^{-1}u \\ \Delta\lambda &= S^{-1}\Lambda A\Delta x - S^{-1}\Lambda s + \mu S^{-1}\cancel{\Lambda\Lambda^{-1}}u \\ \Delta\lambda &= S^{-1}\Lambda A\Delta x - S^{-1}\Lambda s + \mu S^{-1}u \\ \Delta\lambda &= \mu S^{-1}u + \Lambda S^{-1}A\Delta x - \Lambda S^{-1}s \\ \Delta\lambda &= \mu S^{-1}u + \Lambda S^{-1}A\Delta x - \Lambda u \\ \Delta\lambda &= \mu S^{-1}u + \Lambda S^{-1}A\Delta x - \lambda \end{aligned} \tag{4.16.6}$$

Hence,

$$\begin{aligned} Q\Delta x + A^T\Delta\lambda &= 0 \\ Q\Delta x + A^T \cdot (\mu S^{-1}u + \Lambda S^{-1}A\Delta x - \lambda) &= 0 \\ Q\Delta x + A^T\mu S^{-1}u + A^T\Lambda S^{-1}A\Delta x - A^T\lambda &= 0 \\ Q\Delta x + A^T\Lambda S^{-1}A\Delta x &= -A^T\mu S^{-1}u + A^T\lambda \\ (Q + A^T\Lambda S^{-1}A)\Delta x &= A^T(\lambda - \mu S^{-1}u) \end{aligned} \tag{4.16.7}$$

We term $M = Q + A^T\Lambda S^{-1}A$ and the following holds.

Fact 4.16.2. *If A has full column rank (aka it is invertible), $M \succ 0$.*

At this point we need to factorize the matrix M, that changes at each iteration (since ΛS^{-1} does) and this is the bottleneck.

Cholesky factorization may be used, although its complexity is cube. Another downside of this approach is that the matrix M is much denser than A, Λ and S^{-1}.

An orthogonal approach to the reduced KKT is called **predictor-corrector** and it works computing a solution without taking into account the non linear term $\Delta\Lambda\Delta Su$, then computing it according to the approximated solution and repeat until convergence.

The bottlneck again is solving the system in Equation (4.16.1).

For what concerns implementation, we should start from a triplet (x, λ, s), that could be not feasible and then compute the residuals and iterate until feasibility is reached.

$r^D = -(Qx + q) - \lambda A$, $r^P = b - Ax - s$.

When dealing with the step size we need to highlight the fact that $\lambda + \Delta\lambda \geq 0$, $s + \Delta s \geq 0$ should hold.

In order to achieve this we find the maximum α that satisfies the equality and then multiply it by a constant $\bar{\alpha} = 0.995$ (or 0.9995), in order to get closer.

Let us assume that we also have a bunch of box contraints, hence our problem becomes

$$(P) \qquad \min\left\{\frac{1}{2}x^TQx + qx \ : \ Ax = b,\ 0 \leq x \leq u\right\}$$

In this special case, things simplify a lot.