A Refresher on Numerical Linear Algebra and Optimization

Franck Iutzeler and Jérôme Malick

September 21, 2020

Contents

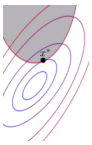
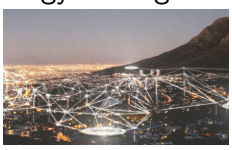
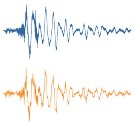
Introduction

These are basic "covid"-notes for the refresher course of the Master of Applied Mathematics at Univ. Grenoble Alpes.

Why matrix analysis and optimization ?

Matrix and optimization are at the **heart of computational mathematics**, With applications everywhere, e.g.

Machine Learning | Energy Management | Signal/Image Processing

Mix between	**theory** meaning of a problem; existence/uniqueness of solution; math properties	and	**practice** Computability, speed, and use of standard libraries to solve numerically these problems.

This short course focuses on *matrix analysis and optimization in action* with exercises inspired from:

- Google PageRank, Image processing
- Machine learning applications (regression, classification)

This course is not a standard course on linear algebra or optimization

- not a math course
  basic knowledge is assumed (take a look to your undergraduate courses)
- not an algorithmic course
  basic programming skills are expected (check-out the Python tutorial in Chapter 1 at https://github.com/iutzeler/refresher-course))

This course is

- a review of basics of matrix analysis – from numerical perspective
- a short overview of numerical optimization
- includes quick recalls from matrix calculus and differential calculus

Contents

- A classification: cvx/non-cvx, smooth/non-smooth, stochastic/deterministic

2. Convexity and optimization
   - Convex sets and functions, exemples
   - Convex optimization problems : global solutions, convex set of solutions
   - Recognizing convexity: definition, convexity-preserving operations, Hessian
   - In practice : modeling, interface, algorithms, experience
3. Simple algorithm for a simple problem : gradient algorithm
   - Unconstrained convex differentiable problems, optimality conditions
   - Gradient algorithm with 4 ingredients of all algorithms
   - Study : convergence theorem vs numerical experiments
   - Beyond gradient : acceleration, 2nd order, Newton

\+ Notes:

- Recalls on derivatives : gradient, Hessian, chain rule, examples
- Application to classification : geometrical/statistical problems, optim. models

Main References:

- Horn, R. & Johnson, C.: *Matrix analysis*.
- Boyd, S. & Vandenberghe, L.: *Convex optimization*.
- Ciarlet, Ph.: *Introduction à l'analyse numérique et l'optimisation*.
- Hiriart-Urruty J.-B., Lemaréchal C.: *Fundamentals of convex analysis*.

CHAPTER I

Linear Algebra

LINEAR ALGEBRA is at the core of most numerical methods.

1 Matrices

1.a Matrix operations

A matrix is an m(lines/rows)$\times n$(columns) array of real[1] numbers. The set of all $m \times n$ real matrices is denoted by $\mathbb{R}^{m\times n}$ (or sometimes $\mathcal{M}_{m\times n}(\mathbb{R})$). If $n = m$, the matrix is said to be square.

[1] in this course

$$A = (A_{ij})_{i=1,..,m;j=1,..,n}$$

Operations between matrices:

- Addition
- Product with a scalar
- Product between matrices

Special cases:

- Matrix/vector product
- Matrix powers

Operations on matrices:

- Transposition
- Trace
- Determinant

1.b Special matrices

- Identity

  $$I \in \mathbb{R}^{n\times n} : I_{ii} = 1 \text{ and } I_{ij} = 0 \text{ for } i \neq j$$

  $AI = A$ for any A; $\det(I) = 1$; $\mathrm{trace}(I) = n$

- Diagonal

  $$D \in \mathbb{R}^{n\times n} : D_{ij} = 0 \text{ for } i \neq j$$

  $\det(D) = \prod_i D_{ii}$

- Triangular

  $$T \in \mathbb{R}^{n\times n} : T_{ij} = 0 \text{ for } i > j$$

  $\det(T) = \prod_i T_{ii}$

- Symmetric

  $$S \in \mathbb{R}^{n\times n} : S = S^\top$$

  e.g. $A^\top A$ is symmetric.

- Orthogonal

  $$Q \in \mathbb{R}^{n\times n} : Q^\top Q = I$$

  the column of Q are unitary and orthogonal, e.g. permutation matrices.

- Invertible (a.k.a. nonsingular)

  $$A \in \mathbb{R}^{n\times n} : \exists B \text{ s.t. } AB = BA = I$$

  if B exists, it is unique and called A^{-1} the inverse of A.
  A invertible $\Leftrightarrow \det(A) \neq 0$.
  A, B invertible $\Rightarrow AB$ invertible with $(AB)^{-1} = B^{-1}A^{-1}$.

- positive (semi-)definite

  $$A \in \mathbb{R}^{n\times n} : \text{symmetric } + x^\top Ax > 0 \forall x$$

  semi-definite if the inequality is not strict. If A is positive definite, then it is invertible.

1.c Factorizations

- LU

  For A invertible, $A = PLU$

  with P a permutation, L lower-triangular, U (upper-)triangular.

- Cholesky

  For A positive definite, $A = LL^\top$

  with L lower-triangular.

- QR

  For any A, $A = QR$

  with Q orthogonal, R (upper-)triangular.

2 Linear Systems

2.a Linear equations

Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, the problem of finding $x \in \mathbb{R}^n$ such that $Ax = b$ appears as a subproblem in virtually all branches of numerical mathematics (ODE, optimization, statistics, etc.).

This problem can be seen as finding the coefficients x of a linear combinations of the columns of A giving b.

Vocabulary reminder:

- The *rank* of A is the number of linearly independent columns
- The set $\{Ax : x \in \mathbb{R}^n\}$ is called the *image* of A
- The set $\{x : Ax = 0\}$ is called the *kernel* of A
- If $n = m$ and ($\operatorname{rank}(A) = n$ or $\operatorname{image}(A) = \mathbb{R}^n$ or $\ker(A) = \{0\}$), A is *invertible* i.e. there is an inverse matrix A^{-1} such that $AA^{-1} = A^{-1}A = I$

Different situations can arise:

- There might might be *no* solutions (if the image is not the full space) $\Rightarrow$ change the problem to least-squares $\min_x \|Ax - b\|_2$

- There might might be *multiple* solutions (if the kernel is not reduced to $\{0\}$) $\Rightarrow$ change the problem to least norm $\min_x \|x\|$ s.t. $\|Ax - b\|_2$
- If A is invertible, $x = A^{-1}b$ is the unique solution

How to *compute* a solution to one of the above problems?

2.b "Easy" cases

The problem of finding x such that $Ax = b$ is easy for certain matrices:

- If you *have the inverse* A^{-1}; however computing A^{-1} is roughly n times harder than solving $Ax = b$ so this should never be done numerically[2].
- Diagonal
- Triangular
- Orthogonal

[2] Use `solve(A,b)` and not `Ainv = inv(A); Ainv*b`.

If we are not in an easy case, we need to be a bit more inventive.

2.c 3 types of algorithms for solving linear systems

1. *using pre-computed factorizations*
   If $A = A_1 A_2$ eg. LU or QR, then

$$\begin{aligned} & Ax = b \\ \Leftrightarrow & A_1 \underbrace{A_2\ x}_{:=y} = b \\ \Leftrightarrow & \begin{cases} A_1\ y & = b \\ A_2\ x & = y \end{cases} \end{aligned}$$

2. *iterative methods* (fixed-point algorithm)
   Choose M, N such that $A = M - N$
   eg. Jacobi , Gauss-Siedel , then

$$\begin{aligned} & Ax = b \\ \Leftrightarrow & (M - N)x = b \\ \Leftrightarrow & Mx = Nx + b \end{aligned}$$

   Initialize x_0 and iteratively set x_{k+1} as a solution of $Mx = Nx_k + b$ (M should be easy)

3. *optimization methods*

$$\begin{aligned} & Ax = b \\ \Leftrightarrow & Ax - b = 0 \end{aligned}$$

   Find x that minimizes $\|Ax - b\|$ (see in the second part).

So what to do in practice?

- In most cases, trust your solver !
- Preconditionning might help: solve $MAx = Mb$ instead with M "reducing the difference between columns/lines" (eg. standard scaler in ML)
- Otherwise, use optimization/iterative methods and use the structure of your matrix

3 Spectral Decompositions

3.a Eigenvalues

For a *squared* matrix $A \in \mathbb{R}^{n \times n}$, a scalar $\lambda \in \mathbb{R}$ or $\mathbb{C}$[3] is an eigenvalue of A if

$$\exists x \neq 0 \quad Ax = \lambda x$$

and a vector x satisfying the above relation is called an *eigenvector* (associated with λ). It is not unique, and globally means that in the direction of x, A is simply a dilatation. The set of all x such that $Ax = \lambda x$ (0 included) is called the *eigenspace* associated with λ.

There are between 1 and n *distinct* eigenvalues; an important quantity being the *spectral radius* defined as $\rho(A) = \max\{|\lambda| : \lambda \text{ is an eigenvalue of } A\}$.

- There are many applications (dynamical systems, graphs, image processing, etc.) and lot of theory surrounding eigenvalues
- $Ax = \lambda x \Leftrightarrow (A - \lambda I)x = 0 \Leftrightarrow x \in \ker(A - \lambda I)$

[3] eigenvalues are complex in general even if A is real

3.b Eigenvalue decomposition for symmetric matrices

For a *symmetric squared* matrix $A \in \mathbb{R}^{n\times n}$, one can say a bit more.

For each λ, such that $\exists x \neq 0 \quad Ax = \lambda x \Leftrightarrow x \in \ker(A - \lambda I)$ the associated eigenspace has dimension $m = n - \operatorname{rank}(A - \lambda I)$, called the (geometric) multiplicity of λ, and thus one can extract m independent eigenvectors from it[4].

[4] This is true in the general squared case

In the symmetric case, there are *exactly* n eigenvalues counting multiplicities[5] (that take between 1 and n *distinct* values) and A admits an *eigendecomposition*, that is

[5] If A is not symmetric, the sum of all (geometric) multiplicities may not be n and such a decomposition is invalid

$$A = PDP^\top = \sum_{i=1}^{n} \lambda_i p_i p_i^\top$$

where $P \in \mathbb{R}^{n\times n}$ is orthogonal and $D \in \mathbb{R}^{n\times n}$ is diagonal. D contains the n eigenvalues of A and P contains n associated eigenvectors (orthonormalized, with vector 1 corresponding to eigenvalue 1 etc., without any particular order for the eigenvalues).

- The eigenvalues of a symmetric matrix are real

  Proof. $\lambda\langle x; x\rangle = \langle \lambda x; x\rangle = \langle Ax; x\rangle = \langle x; Ax\rangle = \langle x; \lambda x\rangle = \overline{\langle \lambda x; x\rangle} = \overline{\lambda}\langle x; x\rangle$ □

- If A is furthermore positive (semi-)definite, the eigenvalues are positive (non-negative)

  Proof. $0 < \langle Ax; x\rangle = \lambda \|x\|^2$ □

- in the non-symmetric case, the *singular value decomposition* is often used, it is linked to the eigendecomposition of the symmetric matrix $A^\top A$

3.c Singular Value Decomposition (SVD)

A real matrix $A \in \mathbb{R}^{m\times n}$ admits a singular value decomposition

$$A = UDV^\top = \sum_{i=1}^{\min\{m,n\}} \lambda_i u_i v_i^\top$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal and $D\mathbb{R}^{m\times n}$ is "diagonal" with non-negative entries (σ_i).

These (σ_i) are the *singular values* of A.

- The rank of A is equal to the number r of non-zero singular values of A, i.e. $r = \operatorname{rank}(A) \leq \min\{m, n\}$. In practice, we do not need information (vectors) corresponding to null singular values. It is common to use[6] the *reduced/compact* SVD

$$A = \sum_{i=1}^{\operatorname{rank}(A)} \lambda_i u_i v_i^\top = \tilde{U}\tilde{D}\tilde{V}^\top$$

where $\tilde{U} \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{r\times n}$ have orthonormal columns and $D\mathbb{R}^{r\times r}$ is diagonal with positive entries (σ_i).

[6] In Python, this is done with the option `full_matrices=False`

- The singular values are related to the eigenvalues of $A^\top A$:

$$\begin{aligned} A^\top A &= VD^\top U^\top UDV^\top \\ &= VD^\top DV^\top \end{aligned}$$

thus $\sigma_i(A)^2 = \lambda_i(A^\top A)$ (≥ 0 and real since $A^\top A$ is positive semi-definite)

4 Matrix Norms

- Norms in $\mathbb{R}^n$
  - $\|\alpha x\| = |\alpha|\|x\|$ (homogeneous)
  - $\|x + y\| \leq \|x\| + \|y\|$ (triangle inequality or sub-additivity)
  - $\|x\| \geq 0$ and $\|x\| = 0 \Leftrightarrow x = 0$ (definition)
- *Matrix Norms* have to verify the additional property[7]
  - $\|AB\| \leq \|A\|\|B\|$ (sub-multiplicativity for squared matrices)
- Typical matrix norms include:
  - the Frobenius norm

$$\|A\|_F = \sqrt{\sum_{i,j=1}^{m,n} |A_{i,j}|^2} = \sqrt{\operatorname{trace} A^\top A} = \sqrt{\sum_{i=1}^{\min\{m,n\}} \sigma_i^2(A)}$$

  - Induced/Operator norms

$$\|A\|_{\alpha,\beta} = \sup\left\{ \frac{\|Ax\|_\beta}{\|x\|_\alpha} : x \neq 0 \right\}$$

where for $A \in \mathbb{R}^{m\times n}$, $\|\cdot\|_\alpha$ is a norm on $\mathbb{R}^n$, $\|\cdot\|_\beta$ is a norm on $\mathbb{R}^m$. For such a norm, we have $\|Ax\|_\beta \leq \|A\|_{\alpha,\beta}\|x\|_\alpha$. The most common case is $\|A\|_{2,2} = \sigma_{\max}(A)$.

[7] One can also get norms on matrices by taking the norm of its entries viewed as a vector, but it may not be sub-mutliplicative eg. $\|x\|_\infty = \max_i |x_i|$.

– Schatten norms

$$\|A\|_{*p} = \left(\sum_{i=1}^{\min\{m,n\}} \sigma_i^p(A) \right)^{\frac{1}{p}}$$

They are vector norms on the singular values of A. $\|A\|_{*\infty} = \|A\|_{2,2}$ and $\|A\|_{*1}$ (also called nuclear norm) are the most frequently used.

- Gelfand's formula: For any matrix norm, $\rho(A) = \lim_{k\to\infty} \|A^k\|^{1/k}$

CHAPTER II

Optimization

OPTIMIZATION is everywhere in science and industry. It is at the core of the most recent advances in Machine Learning

0 Recalls on derivatives

0.a Gradient

Let us take a function

$$\begin{array}{rccc} f: & \mathbb{R}^n & \longrightarrow & \mathbb{R} \\ & x=(x_1,..,x_n) & \longmapsto & f(x) \end{array}$$

- At x, the i-th *partial function* is defined as

$$\begin{array}{rccc} \phi_{i,x}: & \mathbb{R} & \longrightarrow & \mathbb{R} \\ & x & \longmapsto & f(x_1,..,x_{i-1},t,x_{i+1},..,x_n) \end{array}$$

- The i *partial derivative* of f at x is defined (if it exists) as

$$\frac{\partial f}{\partial x_i}(x) = \phi'_{i,x}(x_i) = \lim_{h\to 0} \frac{\phi_{i,x}(x_i+h) - \phi_{i,x}(x_i)}{h}$$

- If all partial derivatives exist, f is differentiable at x and the *gradient* of x is defined as the vector of partial derivatives

$$\nabla f(x) = \left[\frac{\partial f}{\partial x_i}(x)\right]_{i=1,..,n}$$

and it is the unique vector such that

$$f(x+h) = f(x) + \langle \nabla f(x); h \rangle + o(\|h\|)$$

0.b Jacobian

Now, let us take a multi-valued function

$$\begin{array}{rccc} g: & \mathbb{R}^m & \longrightarrow & \mathbb{R}^n \\ & x = (x_1, .., x_m) & \longmapsto & g(x) = [g_i(x)]_{i=1,..,n} \end{array}$$

- The *Jacobian* of g is defined (if it exists) as the matrix of the gradients of the component functions g_i

$$Jg(x) = \left[\nabla g_i(x)^\top\right]_{i=1,..,n} = \left[\frac{\partial g_i}{\partial x_j}(x)\right]_{i=1,..,n;j=1,..,m}$$

0.c Calculus rules

- The *Hessian* of f is its second-order derivative defined as the Jacobian of the gradient

$$Hf(x) = \nabla^2 f(x) = J\nabla f(x) = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}(x)\right]_{i=1,..,n;j=1,..,n}$$

  it is an $n \times n$ symmetric matrix.

- The *chain rule* gives a way to differentiate $f \circ g$ (ie. $x \mapsto f(g(x))$)

$$\nabla(f \circ g)(x) = Jg(x)^\top \nabla f(g(x))$$

- As a side note, in terms of differential $Df(x)[h] = \langle \nabla f(x); h \rangle$; $Dg(x)[h] = Jg(x)h$; $D(f \circ g)(x)[h] = Df(g(x))[Dg(x)[h]] = \langle \nabla f(g(x)); Jg(x)h \rangle$

1 What is optimization?

1.a Optimization problems

Given $f : \mathbb{R}^n \to \mathbb{R}$, the *objective function*, and $C \subset \mathbb{R}^n$, the *constraint set*, our problem is to find $f^\star$ and $x^\star \in C$ such that $f^\star = f(x^\star) \leq f(x)$ for all $x \in C$.

We write this problem

$$\left\{ \begin{array}{l} \min f(x) \\ x \in C \end{array} \right. \qquad (\mathcal{P})$$

How to solve optimization problems? First, what does solving means? Finding $f^\star$, $x^\star$, or an approximation of one/both? It is extremely rare to have explicit/exact solutions so most of the time numerical methods are used to approximate them.

In general, optimization is a (NP-)hard problem but there are situations that are more favorable than others eg. linear programs, convex problems.

1.b Convexity

A central notion in optimization is convexity.

Convex sets A set $C \subset \mathbb{R}^n$ is convex if and only if for any $x, y \in C$ and $\alpha \in [0, 1]$, we have $\alpha x + (1 - \alpha)y \in C$.

Convex functions A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if and only its epigraph[8] is convex, or equivalently if and only if for any $x, y \in \mathbb{R}^n$ and $\alpha \in [0, 1]$, we have $f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$.

[8] $\mathrm{epi} f = \{(x, t) \in \mathbb{R}^n \times \mathbb{R} : f(x) \leq t\}$

Impact of Convexity In a convex optimization problem ($\mathcal{P}$), i.e. if f and C are convex,

- all local solutions are global solutions
- the set of all solutions is convex
- there is a strong theory of duality and a plethora of algorithms

How to recognize convexity?

- Use the definition
- Decompose the function into a sequence of convexity-preserving operations
- When f is twice differentiable, look at the positive definiteness of the Hessian

1.c A classification of optimization problems

- Convex problems
  - smooth unconstrained problems *(least-squares, logistic regression)*
  - smooth constrained *(non-negative least-squares)*
  - non-smooth *(SVM, Lasso)*
- Non-Convex problems
  - continuous problems *(neural networks, Nash equilibrium finding)*
  - discrete optimization *(bandits)*
- Other special cases
  - stochastic optimization *(signal processing, finite-sums)*
  - infinite dimension *(optimal control, calculus of variations)*

So in practice?

- There is an important work of *modeling*
- Rely on *optimization software* for medium-scale problems
- Use at most the *structure* of the problem

2 Gradient descent for smooth optimization

Let us consider the following problem

$$\begin{cases} \min f(x) \\ x \in \mathbb{R}^n \end{cases} \qquad (\mathcal{SP})$$

where f is convex and differentiable.

How to solve ($\mathcal{SP}$)?

- Explicit solution by the optimality conditions

$$x^\star \text{ is optimal } \Leftrightarrow \nabla f(x^\star) = 0$$

- Approximate solution by a numerical method
  We have a (first-order) *oracle* $x \rightsquigarrow (f(x); \nabla f(x))$ and want to construct iteratively a sequence (x_k) such that $x_k \to x^\star$ (in some sense)

2.a Gradient descent

The opposite of the gradient is a *descent direction* for f, indeed

$$\begin{aligned} f(x - t\nabla f(x)) &= f(x) + \langle \nabla f(x); -t\nabla f(x) \rangle + o(\|t\nabla f(x)\|) \\ &= f(x) - t\|\nabla f(x)\|^2 + o(t\|\nabla f(x)\|) \end{aligned}$$

thus for t sufficiently small, $f(x - t\nabla f(x)) \leq f(x)$.

It is thus natural to consider the following algorithm

$$x_{k+1} = x_k - t_k \nabla f(x_k)$$

where we have to set

- x_0 an initialization *(random)*
- one oracle call at each iteration *(usually the costly part)*
- the stepsize t_k at each iteration *(fixed, decreasing, line-search)*
- a stopping criterion *($\|\nabla f(x_k)\| \leq \varepsilon$, $f(x_k) - f^\star \leq \delta$, max. number of iterations)*

2.b Theoretical analysis

A typical convergence result for gradient descent is

Theorem. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex differentiable function with at least a minimizer and further assume that ∇f is Lipchitz-continuous[9] with constant L, i.e.*

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2 \quad \forall x, y.$$

Then, the fixed-stepsize gradient algorithm

$$x_{k+1} = x_k - \gamma \nabla f(x_k)$$

with $0 < \gamma < 2/L$ converges to a minimizer.

[9] *the term L-*smooth *is often used*

Remarks:

- The gradient method with *constant stepsize* works for smooth functions
- The smoothness parameter has te be estimated beforehand. For instance, if f is twice differentiable by the mean value theorem

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq \left(\sup_u \|Hf(u)\|_{2,2}\right) \|x - y\|_2 = \left(\sup_u \sigma_{\max}(Hf(u))\right) \|x - y\|_2.$$

- Such a fixed stepsize is maybe not optimal in practice

Beyond gradient descent The gradient method, as the archetype of first-order methods, has several advantages but also many drawbacks (slowness, global parameters), to get over them:

- More advanced methods
  - Nesterov's accelerated gradient
  - Conjugate gradient for linear systems
- Second order methods
  - Newton's method
  - Quasi-Newton (BFGS, etc.)

2.c Examples

Least squares

Classification