Computational Methods for Learning and Data Analysis*

— Exercise sessions — Seminars —

Leonardo Robol
<leonardo.robol@isti.cnr.it>

Introduction

The purpose of these lecture notes is to follow as closely as possible the material that is presented in the seminars. You will find a (brief!) recap of many things that you see during the lectures, as well as some more exercises to see if you have understood completely the covered topics.

Simple exercises (without solutions) are spread out in the text, and some others with detailed solutions and explanations are at the end of every section.

> *Please notice that these notes are a work-in-progress. I am writing them while I teach the exercise sesssions. They are likely to change and (hopefully) improve. The content might be reorganized several times based on time and students' feedback.*

Some generic hints:

- Concerning linear algebra, MATLAB is a very powerful tool not only for computations, but also for learning. Try out to compute inverses, to perform matrix multiplications, and so on and so forth, and verify what you think should be true.
- Some exercises here are relatively theoretical, but the focus of this course is quite hands-on. So feel free to ignore them, if you wish – but sometimes trying to prove things can reveal if you did (or did not) really understand how things work "under the hood".

Found a typo or an error? Let me know at <leonardo.robol@isti.cnr.it>

*Notes updated at November 14, 2017.

Contents

1 Vector spaces and basic linear algebra

1.1 The Euclidean space $\mathbb{R}^n$ and elements of linear algebra

Most of the time, we will work with a very special space: the Euclidean space $\mathbb{R}^n$. It might happen that we swap $\mathbb{R}$ with the complex numbers $\mathbb{C}$, but this will not change much from our perspective.

The set $\mathbb{R}^n$ is the set of *vectors* with n components, that is of the tuples of n numbers:

$$x \in \mathbb{R}^n \iff x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \qquad x_i \in \mathbb{R}.$$

Vectors in $\mathbb{R}^n$ are *column vectors*, i.e., the numbers are stacked one above the other (opposed to *row vectors*). You might be tempted to say "who cares?", but please think of them this way and everything will be much more clear in the following.

What can we do with vectors? We can sum two of them, and we can multiply them by scalars:

$$x + y = \begin{bmatrix} x_1 + y_1 \\ \vdots \\ x_n + y_n \end{bmatrix}, \qquad \lambda x = \begin{bmatrix} \lambda x_1 \\ \vdots \\ \lambda x_n \end{bmatrix}.$$

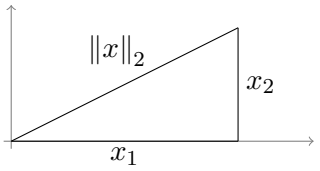
Generically, a *vector space* is any set that can be endowed with such operations: the sum of two vectors, and the multiplication by a scalar. $\mathbb{R}^n$ is a very representative example. In fact, all the n-dimensional vector spaces of dimension[1] n is a copy of $\mathbb{R}^n$, in some sense.

Given a vector we can measure its "length", which mathematically is what we call the *Euclidean norm*, denoted by $\|\cdot\|_2$:

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

If you consider $n = 2$, this is just Pythagorean theorem:

$$\left\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right\|_2 = \sqrt{x_1^2 + x_2^2}$$



In more dimensions, it still is the length of the vector, or its distance from the origin (the vector with all the entries equal to 0). In fact, using the norm we can define the distance between any two vectors in $\mathbb{R}^n$ by $d(x, y) := \|x - y\|_2$.

There are many more possible choices for a distance, and also more choices for norms. In fact, a norm is just a function from $\mathbb{R}^n \to \mathbb{R}$ that satisfies the following constraints:

[1] We have not defined what the dimension is, as of now, but think of the degrees of freedom.

- $\|x\| \geq 0$ and $\|x\| = 0$ if and only if $x = 0$.
- $\|\lambda x\| = |\lambda|\|x\|$ for any scalar λ.
- $\|x + y\| \leq \|x\| + \|y\|$ (triangular inequality).

Exercise 1.1. Show that the Euclidean norm $\|\cdot\|_2$ satisfies the above properties.

Exercise 1.2. Show that the following functions from $\mathbb{R}^n$ to $\mathbb{R}$ are norms:

$$\|x\|_p := \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}, \ p \in \mathbb{N}_+ \qquad \|x\|_\infty := \max_{i=1,\ldots,n} |x_i|.$$

In the above exercises proving the third property ("triangular inequality") is the most difficult part. We will look at it again when we see what convexity is. Don't worry if you cannot do it right now.

1.2 Bases, dimension, and invertibility

Every finite-dimensional vector space (such as $\mathbb{R}^n$, which is the only kind of vector spaces we consider) has a basis (in fact, it has many of them).

A basis is a collection of vector $x^{(1)}, \ldots, x^{(n)}$ such that:

- Every vector $x \in \mathbb{R}^n$ can be obtained as a linear combination of $x^{(1)}, \ldots, x^{(n)}$ (these vectors *generate* $\mathbb{R}^n$).
- There is only one way to combine $x^{(1)}, \ldots, x^{(n)}$ to obtain x or, which is the same, there is no (non-trivial) combination of $x^{(1)}, \ldots, x^{(n)}$ that sums to zero (these vectors are *linearly independent*).

A basis must have cardinality n, which is called the *dimension* of the vector space.

Exercise 1.3. Show that, if you have two different combinations of $x^{(1)}, \ldots, x^{(n)}$ which are equal to the same vector x, i.e.,

$$x = \alpha_1 x^{(1)} + \ldots + \alpha_n x^{(n)} = \beta_1 x^{(1)} + \ldots + \beta_n x^{(n)}$$

with $\alpha \neq \beta$, then there must exists a non-trivial combination of $x^{(1)}, \ldots, x^{(n)}$ that sums to zero.

Assume that we have a basis $\mathcal{B} = \{b^{(1)}, \ldots, b^{(n)}\}$. The vector with the numbers $\alpha_1, \ldots, \alpha_n$ such that $x = \alpha_1 b^{(1)} + \ldots + \alpha_n b^{(n)}$ is called the *coordinate vector* of x with respect to the basis $\mathcal{B}$.

In $\mathbb{R}^n$ we have a *standard* choice for the basis, which is what we call the *canonical basis*, and is tipically denoted with the letter e_i:

$$\mathcal{E} := \{e_1, \ldots, e_n\}, \qquad e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad e_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad e_n = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

Writing a vector x in coordinates can be expressed even more conveniently in matrix form. What is a matrix? One possible definition, not very insightful, is that is just a table of numbers; we use the following notation:

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}.$$

The matrix A above is $m \times n$, it has m rows and n columns. Notice that column vectors can be seen as $n \times 1$ matrices, and that will be useful several times. As a motivating example, that's exactly how MATLAB represents vectors.

What makes matrices useful is that they have an *algebraic* structure: we can sum them (just summing element-wise, so we need them to have the same number of rows and columns), and multiply them, if the *inner* dimensions are compatible. Assume that we have $A \in \mathbb{R}^{m \times p}, B \in \mathbb{R}^{p \times n}$, then the matrix $C \in \mathbb{R}^{m \times n}$ can be defined as

$$AB = C = \begin{bmatrix} c_{11} & \dots & c_{1n} \\ \vdots & & \vdots \\ c_{m1} & \dots & c_{nn} \end{bmatrix}, \qquad c_{ij} := \sum_{k=1}^{p} a_{ip} b_{pj}.$$

We sum on the inner index p (and therefore the number of columns of A must be equal to the number of rows of B!), and we can compute all the elements of the product.

Let's look at the simpler case where B is a vector (so an $n \times 1$ matrix), and A is $n \times n$:

$$Ax = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \iff y_i = \sum_{j=1}^{n} a_{ij} x_j.$$

In fact, grouping all the indices i in vector form, we can rewrite the above expression as:

$$y = x_1 \begin{bmatrix} a_{11} \\ \vdots \\ a_{n1} \end{bmatrix} + \ldots + x_n \begin{bmatrix} a_{1n} \\ \vdots \\ a_{nn} \end{bmatrix},$$

which means that y is obtained as a linear combination of the columns of A. Remember when we defined writing x in coordinates with respect to a basis $\mathcal{B}$? This can be rewritten in matrix form as follows:

$$x = B\alpha \qquad \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b^{(1)} & \dots & b^{(n)} \\ & & \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}.$$

For a generic matrix product $C = AB$, the columns of C are obtained as a linear combination of the columns A, according to the coefficients in the corresponding column of B.

Remark 1.4. The product of two matrices does not *commute*, in fact it might not even make sense dimension-wise! So please always remember that there is no reason (unless we are in a very very special case) to assume that $AB = BA$.

An important matrix is the *identity*, which we denote by I, which is zero everywhere except on the diagonal (the entries with $i = j$), where it has 1s. One immediately verifies that $IA = A$.

Definition 1.5. A (square) matrix A is said to be *invertible* if there exists another matrix B such that $AB = I = BA$.

Exercise 1.6. Prove that a matrix which has as columns the elements of a basis is invertible. Prove that this is an if and only if (every invertible matrix has as columns the elements of a basis).

Another important operation on matrices is the *transposition*. Given a matrix with elements a_{ij}, the transposed matrix is the one with elements a_{ji}. In practice, we are "flipping" the matrix around its diagonal. This implies that if A is $m \times n$ then the transpose of A, denoted by A^T, is $n \times m$.

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \implies A^T = \begin{bmatrix} a_{11} & \dots & a_{m1} \\ \vdots & & \vdots \\ a_{1n} & \dots & a_{mn} \end{bmatrix}$$

With the newly defined matrix product and transposition we can introduce a new operation, called *scalar product* of two vectors:

$$x, y \in \mathbb{R}^n \implies x^T y = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i.$$

Notice that the scalar product is symmetric in x and y, so $x^T y$ is equal to $y^T x$. Moreover, we can now rephrase the Euclidean norm as $\|x\|_2 = \sqrt{x^T x}$.

If A is invertible, then also A^T is invertible, and in fact the inversion *commutes* with the transposition: it does not matter in which order we apply them. For this reason, we sometimes shorten $(A^{-1})^T = (A^T)^{-1}$ as A^{-T}.

Exercise 1.7. Prove that for A, B matrices with compatible sizes we have $(AB)^T = B^T A^T$. Use that result to show that, if A is invertible, then also A^T is invertible and that its inverse if $(A^{-1})^T$.

1.3 The Vandermonde matrix

Let us consider the following matrix, usually called the *Vandermonde* matrix associated with the nodes $x_1, \dots, x_n$:

$$V := \begin{bmatrix} 1 & x_1 & \dots & x_1^{n-1} \\ \vdots & & & \vdots \\ 1 & x_n & \dots & x_n^{n-1} \end{bmatrix}$$

This is a very important matrix, so let's try to prove some basic facts about it. First of all, we would like to prove that, if $x_1, \dots, x_n$ are pairwise distinct points, then V is invertible. Let's try to recap a bit the characterization of invertibility that we have until now:

1. A matrix is invertible if it has an inverse, that is V invertible if and only if there exists another matrix V^{-1} such that $VV^{-1} = V^{-1}V = I$.
2. A matrix is invertible if its columns form a basis, that is they generate the space $\mathbb{R}^n$ and they are linearly independent.

For now we will focus on point 2., since it's much easier. In fact, trying to design an inverse off the top of our head is a much more intimidating task!

Remark 1.8. Originally, we have defined some vectors to be a basis if they *generate* and they are *linearly independent*. We have mentioned that if these conditions are true then they must be n vectors. In fact, consider the following conditions:

- A set of vectors generate the space $\mathbb{R}^n$.
- A set of vectors are linearly independent in $\mathbb{R}^n$.
- A set of vectors has cardinality n.

If two of the above conditions are true, then the third is automatically satisfied. In particular, we can check just two conditions to show that a set of vectors form a basis.

In our case the columns of V are n, so we can for example check that the columns are linearly independent. This is true if and only if $Vx = 0 \implies x = 0$, that is the only combination of the columns summing to zero is the trivial one.

Consider the product of V for a generic vector of components a_i, for $i = 0, \dots, n-1$:

$$\begin{bmatrix} 1 & x_1 & \dots & x_1^{n-1} \\ \vdots & & & \vdots \\ 1 & x_n & \dots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} a_0 + a_1 x_1 + \dots + a_{n-1} x_1^{n-1} \\ \vdots \\ a_0 + a_1 x_n + \dots + a_{n-1} x_n^{n-1} \end{bmatrix}.$$

Let us call $a(x) = a_0 + a_1 + \dots + a_{n-1}x^{n-1}$ the degree $n-1$ polynomial in x, defined by the coefficients a_i. It is an immediate check that

$$Va = \begin{bmatrix} a(x_1) \\ \vdots \\ a(x_n) \end{bmatrix}.$$

Let us recall a very powerful theorem.

Theorem 1.9 (Fundamental theorem of algebra)**.** *A degree n polynomial different from zero can have at most[2] n roots, that is:*

$$\#\{p(x) = 0\} \leq \deg p(x).$$

[2]It has exactly n roots in $\mathbb{C}$, if you count them with multiplicity, but this is not so important for now.

Notice that $a(x)$ has degree $n-1$, so it can have at most $n-1$ roots. However, if we ask that $Va = 0$, and $x_1, \ldots, x_n$ are *distinct*, then $a(x)$ vanishes in n points, which is *strictly* larger than $n-1$. Therefore, the only possibility is that $a(x) \equiv 0$, so $a_i = 0$ for every i, and we have proved that V has an inverse.

However, you might be disappointed, since we do not *know* what the inverse is. We have just proved that it must exists. The good news is that, for most of the important cases, we do not care what the inverse is.

In fact, we will tipically need to compute $y = V^{-1}x$, for x a certain vector. In order to do this, as we will see later on (cfr. the section on linear systems), we do not need to compute the inverse of V, but just its effect on the vector x.

This is a quite recurrent situation in numerical analysis: for most of the time we do not care about computing inverses explicitly, but we just want to know the inverse times a vector, and the turns out to be a somewhat easier and much more well-behaved problem.

So, as a rule of thumb, if you're using MATLAB and you are writing `inv(A)`, you're probably doing something wrong (as a matter of fact, MATLAB even *tells you* that you should not do that in recent versions).

Let us now show something else about this matrix.

Theorem 1.10. *Assume that $y_1, \ldots, y_n$ are the evaluations of a function $f(x)$ at pairwise distinct points $x_1, \ldots, x_n$. Let V be the Vandermonde matrix associated with $x_1, \ldots, x_n$, and $p = V^{-1}y$. Then, the degree $n-1$ polynomial with coefficients $p(x) = p_0 + p_1 x + \ldots + p_n x^n$* interpolates *$f(x)$ at x_i, that is: $p(x_i) = f(x_i)$.*

Proof. Trying to show this is a good (not too difficult) exercise; the proof is not included, but you can try to write it down as an exercise. □

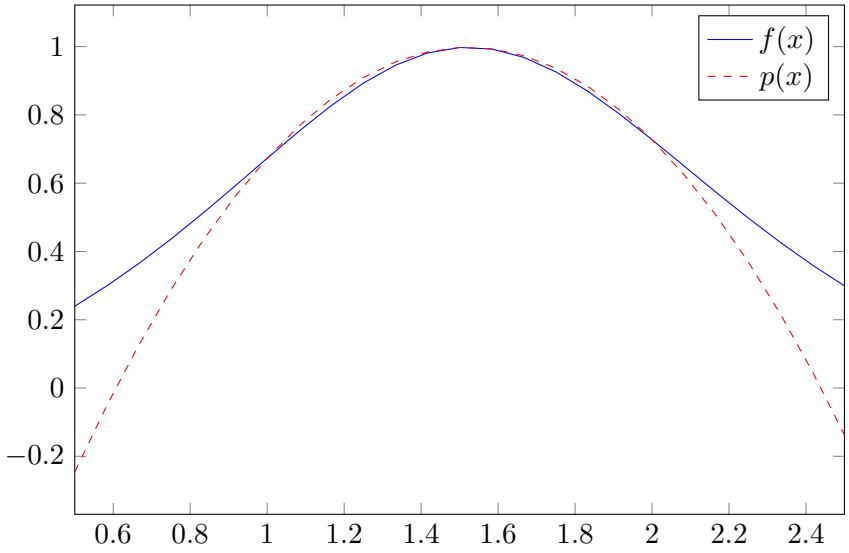
But let's try to make an example in practice. Consider the function $f(x)$, and the vector x as follows:

$$f(x) = \frac{\sin(x)}{1 + (x - 1.5)^2}, \qquad x = \begin{bmatrix} 1 \\ 1.5 \\ 2 \end{bmatrix}.$$

We can easily test this example in MATLAB (notice the syntax for inline definition of functions).

```
>> V = [ 1 1 1 ; 1 1.5 1.5^2 ; 1 2 2^2 ];
>> f = @(x) sin(x) ./ (1 + (x - 1.5).^2);
>> p = inv(V) * f(x) % Equivalent: use p = V \ y
>> t = linspace(.5,2.5,100); plot(t,f(t),t,p(1) + p(2)*t + p(3)*t.^2);
```

The result of the plot should look more or less like this:



You see that the polynomial (the red dashed line) matches the function exactly in the prescribed points. In fact it is a very good approximation in the interval $[1, 2]$, but degrades quickly ouf of that interval.

Using MATLAB you might want to checkout the functions `vander` to generate Vandermonde matrices, and `polyval`, to evaluate polynomials. But beware that MATLAB uses the opposite ordering for the coefficients of polynomials.

1.4 Complexity of matrix multiplication

We are interested in playing with computers, so we will often multiply two matrices A and B. How long does it take to do this on a CPU? To have a rough idea, tipically we count the number of operations that we need to do. Modern CPUs can perform roughly $3 \cdot 10^9$ operations per second (it's slightly more complex than this), so 1 billion of operation will take a fraction of a second.

Let's try to check how many operations we need to multiply A and B. Assume that $A \in \mathbb{R}^{m\times p}$, and that $B \in \mathbb{R}^{p\times n}$. Then, if $C = AB$, to compute each entry in C we need to perform p multiplication and $p-1$ additions. Since C has mn elements, this accounts for $\mathcal{O}(mnp)$ operations.

We will see this notation (called "big-oh" notation) very often. If some algorithm requires $\mathcal{O}(n^\alpha m^\beta)$ it means that the total number of operations, for n, m sufficiently large, is dominated by a term of the form $Kn^\alpha m^\beta$. Notice that the constant K is omitted in this computation! This can often make a huge difference.

1.5 Range, kernel, and rank

We shall define some basic elements related to the matrices. In order to do this, we need another (very useful) interpreation of a matrix as a linear operator. Let A be an $m \times n$

matrix, and consider the mapping between $\mathbb{R}^n$ and $\mathbb{R}^m$ defined as follows:

$$T_A : \mathbb{R}^m \longrightarrow \mathbb{R}^n, \qquad T_A(x) = Ax.$$

Consider the following set, called the *range* of the matrix A, which is the image of the map T_A:

$$\text{Range}(A) := \{y \mid \exists\ x \in \mathbb{R}^n\ y = Ax\}.$$

Exercise 1.11. Prove that the above set is a vector space. More precisely, show that it is a subspace of the ambient vector space $\mathbb{R}^m$.

The dimension of the subspace Range(A) (think of the number of degrees of freedom), is the *column-rank* of A. Let us make an example. Consider the matrix A defined by

$$A = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

The column rank of the above matrix is 2. How can we check it? Let x be any vector in $\mathbb{R}^4$, and consider $y = Ax$. We have

$$y = Ax = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_1 + 2x_2 + x_4 \\ 0 \\ x_3 + x_4 \end{bmatrix}.$$

It's rather clear that, with proper choices of the components of x, we can obtain all the vectors of the form

$$y = \begin{bmatrix} \gamma \\ 0 \\ \delta \end{bmatrix},$$

for some parameters γ and δ. However, there is no way that we can make the second component nonzero. In particular, we have 2 degrees of freedom and so the dimension of the range of A is 2.

In a similar way, we can define the *kernel* of A. Consider the set defined as

$$\text{Ker}(A) := \{x \in \mathbb{R}^n \mid Ax = 0\} \subseteq \mathbb{R}^n.$$

Exercise 1.12. Prove that Ker(A) is a subspace of $\mathbb{R}^n$.

First of all, let us remark an important difference: the kernel is a subspace of the domain of T_A, whereas Range(A) is a subspace of the codomain of T_A. The dimension of Ker(A) is called the row rank of A. Let us go back to our usual matrix A, and let's try to determine what the row rank is.

In order to have $Ax = 0$ we need to have $x_3 + x_4 = 0$, so $x_3 = -x_4$, and then $x_1 + 2x_2 + x_4 = 0$. Working out these relations one easily see that all the vectors that

satisfy these equations needs to be of the form

$$x = \begin{bmatrix} -2\delta - \gamma \\ \delta \\ \gamma \\ -\gamma \end{bmatrix}.$$

Again, x has two degrees of freedom, so the dimension of the $\mathrm{Ker}(A)$ is 2. In particular, the row and column rank coincides! In fact, this is not a coincidence: these two ranks always are the same, and therefore we just call this number *rank*.

1.6 Rank 1 matrices

Consider two vectors u, v, with m and n elements, respectively. Then the matrix $A = uv^T$ is $m \times n$ (the inner dimension of u and v^T is 1). Matrices of this form are called rank 1 matrices. They will be a very important tool in the following.

1.7 Orthogonal basis

One could say that not all the bases are equal. In fact, some are "numerically" better than others. In order to describe these nice bases we need the concept of *orthogonality*.

We say that two vectors x, y in the space $\mathbb{R}^n$ are *orthogonal* if $x^T y = y^T x = 0$. We say that a set of vectors $x_1, \ldots, x_k$ are *orthonormal* if they are orthogonal and $x_i^T x_i = \|x_i\|_2^2 = 1$ for every $i = 1, \ldots, k$.

The operation $x^T y$ is what we typically call the (canonical) scalar product on $\mathbb{R}^n$, and is somtimes denoted in other ways, such as $\langle x, y \rangle$. However, since we have agreed that the vectors are *column* vectors, for us it's much easier to write $x^T y$, which is a special case of matrix multiplication.

We say that a matrix $U \in \mathbb{R}^{n \times k}$ is *orthogonal* if its columns are orthonormal. We can write this condition as $U^T U = I_k$.

Exercise 1.13. Show that an $n \times n$ orthogonal matrix U is invertible.

Orthogonal matrices have several nice properties.

- They preserve the Euclidean norm. That is, if U is orthogonal and $y = Ux$ then $\|y\| = \|x\|$. Notice that x and y might have a different number of entries here (if U is not square).
- When they are square, they are very easy to invert. In fact, matrix multiplication and inversion are "essentially the same" for a unitary matrix.

When dealing with matrices with complex entries (that is, numbers of the form $a + bi$ where $i = \sqrt{-1}$), the scalar product is denoted as $x^H y$, where $x^H := \overline{x}^T$ is the conjugate-transpose operator. Sometimes, the small start $x^\star$ is used in place of the ^{H}. Orthogonal matrices on the complex field are called *unitary* matrices.

Exercise 1.14. Consider the *Fourier matrix*, with complex entries, defined as follows:

$$F = \frac{1}{\sqrt{n}} \begin{bmatrix} 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(n-1)} \\ \vdots & & & \vdots & \\ 1 & \omega^{(n-1)} & \omega^{2(n-1)} & \dots & \omega^{(n-1)^2} \end{bmatrix}, \qquad \omega^n = 1, \qquad \omega^j \neq 1, \ \forall j < n,$$

that is the (scaled) Vandermonde matrix with the roots of the unity as nodes. Prove that F is unitary. The multiplication $y = Fx$, or $x = F^H y$ are usually called the *discrete Fourier transform*, or the *inverse discrete Fourier transform*.

2 Elements of topology and analysis

2.1 Open and closed sets

With a distance at hand, we can define open (and closed) sets in $\mathbb{R}^n$. A set $X \subseteq \mathbb{R}^n$ is said to be *open* if, for every $x_0 \in X$, all the points which are sufficiently close to x_0 are contained in X. More formally:

Definition 2.1. A set X is *open* if for every point x_0 there exists a positive $\epsilon > 0$ such that $B(x_0, \epsilon) \subseteq X$, with

$$B(x_0, \epsilon) := \{y \mid d(y, x_0) = \|x_0 - y\|_2 < \epsilon\}.$$

The set $B(x_0, \epsilon)$ is called the *ball* of center x_0 and radius ϵ, and is itself an open set (can you prove it?).

Definition 2.2. A set X is closed if $X^C := \mathbb{R}^n \setminus X$, the complement of X, is open.

Exercise 2.3. Try to answer the following questions:

- Is it true that a set is either open or closed? (yes/no/why). If not, can you provide examples of sets which are neither open nor closed, or which are both open *and* closed?
- Are the sets $\emptyset$ (:= the empty set, with no elements), $\mathbb{R}^n$ open? Are they closed?

For any set we call *internal points* the ones where the conditions of being open is satisfied, and the *boundary* (or border, sometimes) point the ones where it fails, that is the ones where, for any $\epsilon > 0$, there points outside the set which are closer than ϵ.

Exercise 2.4. Try to prove the following facts about open / closed sets.

- Prove that $[0, 1]$ is closed, and that $(0, 1)$ is open.
- Prove that the internal part of $(0, 1]$ is $(0, 1)$, and that the boundary of is $\{1\}$.

- Prove that the union and interesction of two open (resp. closed) sets is open (resp. closed). Show that the union of a family of open sets is open and the interection of a family of closed sets is closed. Show that since these properties hold for two sets, they need to hold for an arbitrary (finite) number of sets.
- Show that the interection of a family[3] of open sets does not need to be open, and that the union of a family of closed sets does not need to be closed.

2.2 Continuity and Compactness

Let's fix some terminology. Given a function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ the set X is called the *domain* of f. This is the set of points where f is defined, that is the ones where we can compute $f(x)$.

The set of points that can be reached through $f(x)$, i.e,

$$\Im(f) := \{y \mid \exists x, f(x) = y\}$$

is called the *image* of $f(x)$. We are interested in some form of *regularity* for $f(x)$, because otherwise working with "just any function" would be too difficult.

Definition 2.5. We say that $f(x)$ is continuous if, for every point $x \in X$ and for any $\epsilon > 0$ there exists a $\delta > 0$ such that

$$\|x - y\| \leq \delta \implies |f(x) - f(y)| \leq \epsilon.$$

Warning: δ depends on ϵ *and* x. If, given ϵ, we can find a δ that fits all x then we say that $f(x)$ is *uniformly continuous*.

Exercise 2.6. Show that the functions $f_1(x) = \sin(\frac{1}{x})$, $f_2(x) = \frac{1}{x}$ are continuous on $\{x > 0\}$ but not uniformly continuous.

Intuitively, continuous functions are well-behaved, because they do not "jump unexpectedly". There are many more forms of continuity:

Definition 2.7. We say that $f(x)$ is Lipschitz continuous with constant L if

$$|f(x) - f(y)| \leq L|x - y|.$$

Exercise 2.8. Show that Lipschitz continuous $\implies$ uniformly continuous $\implies$ continuous.

These inclusions are strict, in the sense that there exists at least a function that is continuous but not Lipschitz continuous, and a function that is uniformly continuous but not Lipschitz.

Exercise 2.9. Show that the function $f(x) = x \sin(\frac{1}{x})$, extended with 0 in 0, is uniformly continuous on $[-1, 1]$ but not Lipschitz.

[3] Here family means a not necessarily finite collection. In fact, given the previous point, any example where these properties fail will need to be done with an infinite collection of sets.

Continuity is preserved by many operations:

- If $f(x)$ and $g(x)$ are continuous, then $f+g$ and fg are also continuous.
- If f and g are continuous then $f \circ g$ (the composition) is continuous.

Many problems in the Real World$^{\text{TM}}$ can be rephrased as "find the minimum point (and possibly the value there) of a function $f(x)$ on a given set X". In more mathematical terms we want to determine a point x^* such that $x^* := \min_{x \in X} f(x)$.

Unfortunately, there are many things that can go wrong:

- Such a minimum point may not exist.
- If it exists, it might not be unique.
- In practice, it is much easier to find *local* minima (points which are a minimum amongst the closeby points), and much more difficult to prove that a point is a *global* minimizer.

Can we guarantee to find this minimum from a theoretical perspective? Trying to answer this question in the affirmative relies on two important assumptions: the *continuity* of $f(x)$ and the *compactness* of the set X. As we will see, with these two hypotheses we can predict that a minimum (and a maximum) point exist in the given set.

Remark 2.10. Most of the time, our set X will not be completely general. In practice, we will consider either $X \subseteq \mathbb{R}$ or $X \subseteq \mathbb{R}^n$, or the analogous complex versions $X \subseteq \mathbb{C}$ and $X \subseteq \mathbb{C}^n$.

A set $X \subseteq \mathbb{R}^n$ is *compact* if it's closed and bounded. Example: the interval $[0,1]$. The most important property of compact sets[4] is that every sequence of elements in X has at least one convergent subsequence.

This allows to prove the following.

Theorem 2.11 (Weierstrass' extreme value)**.** *If $f(x)$ is continuous on a compact set X, then $f(x)$ has maximum and minimum in X.*

If some of the above hypotheses (continuity and/or compactness) are not satisfied, the minimum / maximum might not exist. For example, consider the function $f(x) = x$ (which is continuous!) defined on $\mathbb{R}$. This function has neither maximum nor minimum.

On the other hand, consider the function $f(x) = x$ if $x > 0$ and $f(0) = 1$, defined on the compact set $[0,1]$. This set is compact, but the function is not continuous, and there is no minimum (but there is a maximum).

Sometimes we can ask even more regularity than continutity! We can compute the well-known derivative of $f(x)$ (in one variale), or the gradient $\nabla f(x)$ (in more variables). If $f(x)$ is a function from $\mathbb{R} \to \mathbb{R}$,

$$f'(x) := \lim_{t \to 0} \frac{f(x+t) - f(x)}{t}.$$

[4] In fact, this can be the definition in more general contexts.

This limit might not exist, and in that case the function is not differentiable. If it exists, and the derivative is continuous, we say that the function $f(x)$ is $\mathcal{C}^1$. We can define partial and directional derivatives the usual way for functions in more variables:

$$\frac{\partial f(x)}{\partial v} := \lim_{t\to 0} \frac{f(x+tv)-f(x)}{t}, \qquad \frac{\partial f(x)}{\partial x_i} := \frac{\partial f(x)}{\partial e_i},$$

where e_i are the vectors of the canonical basis. We call the vector with the partial derivatives the *gradient of f at x*:

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} & \dots & \frac{\partial f(x)}{\partial x_n} \end{bmatrix}.$$

The gradient is a *row vector*. We say that $f(x)$ is differentiable (in more variables) if the partial derivatives are continuous[5]. If that's the case, we can recover the directional derivatives as

$$\frac{\partial f(x)}{\partial v} = \nabla f(x) \cdot v = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} & \dots & \frac{\partial f(x)}{\partial x_n} \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}.$$

We can use derivatives to find minima and maxima of functions. Derivatives are *local* operators, they only depend on the behavior of the function close to a point, so they cannot give us information on *global* minima (or maxima), but only on local ones.

Definition 2.12. We say that x is a local minimum (resp. maximum) of $f(x)$ if there exists a ball $B(x,\epsilon)$, for some $\epsilon > 0$, such that $f(y) \geq f(x)$ (resp $f(y) < f(x)$) for every $y \in B(x,\epsilon)$.

This is when derivatives come into play: if a function is $\mathcal{C}^1$ and has a local minimum (resp. maximum) at an interal point x, then we must have $\nabla f(x) = 0$.

Let's try to understand why this happens. The gradient gives us an approximation of the function as

$$f(y) = \underbrace{f(x) + \nabla f(x) \cdot (y-x)}_{f_1(y)} + h(y),$$

where $h(x) = 0$, and $\nabla h(x) = 0$. Intuitively, close to x, the linear function $f_1(y)$ is the best possible linear approximation of $f(y)$. In particular, if $\nabla f(x) \neq 0$, we can choose $y = x - \gamma \nabla f(x)$, for a *very small* γ, so that

$$f(y) \approx f(x) + \nabla f(x) \cdot (y-x) = f(x) - \gamma \|\nabla f(x)\|^2 < f(x),$$

which is not possible since $f(x)$ was a local minimum. We have ignored the term $h(y)$ since we have supposed that γ is very small. This concept can be made more precise using Taylor series, which we will see later on.

[5]The definition of differentiability is slightly more general, but it does not matter at the moment for us.

Remark 2.13. In general, $\nabla f(x) = 0$ is not a sufficient condition for x to be a minimum, but only a necessary one. However, it already helps a lot in finding some good candidates, that then we can check one at a time. Recall that this characterization does not tell us anything about the boundaries, so we need to check these carefully!

Exercise 2.14. Find the (local and global) minima and maxima of the following functions on the given sets. Give all the details on why the points you have found are the right ones.

- $f(x) = x$, on the set $[0, 1]$.
- $f(x) = x^2 - x + 1$, on $[0, 1]$.
- $f(x_1, x_2) = x_1^2 + x_2^1 + x_1 x_2$ on $\mathbb{R}^2$.

3 Optimization algorithms

The design of the optimization algorithms that we consider is based on *descent directions*. We construct an iterative process

$$x_{k+1} = x_k + \alpha_k d_k, \qquad k > 0,$$

that we initialize with a proper choice x_0, and that we hope will converge to a minimum (resp. a maximum) of the objective function f. The direction in which we move is denoted by d_k, which we call *descent direction*, whereas the "length" of the step is controlled by the scalar number α_k, usually referred to as the *step length*.

Choosing these values (α_k and d_k) is the main difference in most gradient-based optimization algorithms. Based on the theory that we have seen, we know a good choice for d_k can be obtained in a simple way: if we set $d_k = -\nabla f(x_k)$ we obtain the direction along which the descent of $f(x)$ is *locally* maximum.

This leads to the simplest method: the *steepest descent iteration*. This can be defined as follows

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

However, there are still lots of open problems in this setting:

- Even though $-\nabla f(x_k)$ is the best choice that one can make to minimize the function taking arbitrarily small steps, there is really no guarantee that this will continue to hold as we go farther from x_k.
- How to choose α_k? This is an optimization problem by itself, and we will see that there are several strategies to make this choice that work for different optimization methods.

3.1 Exact line search

We shall notice that having chosen d_k is already a huge step forward in making the step: we have reduced the dimensionality of the problem considerably. In fact, independently of the number of variables of x, once the descent direction d_k is chosen, determining α_k is a 1D optimization problem.

For this reason, it makes sense to discuss how to optimize a 1D function in more detail. Notice that, at this stage, we are not considering derivatives of order higher than 1, therefore we can only try to extract informations from the first derivatives. In particular, our goal will be to find stationary points, that is points where our objective function has a vanishing derivative.

Let us make the problem more precise. If d_k if fixed, we want to find

$$\min_{\alpha \geq 0} \phi(\alpha), \qquad \phi(\alpha) := f(x_k + \alpha d_k).$$

The chain rule for the derivative tells us immediately that $\phi'(\alpha) = \nabla f(x_k) d_k$. In particular, the minimization is essentially equivalent to the zerofinding problem to determine the points α for which $\phi'(\alpha) = 0$.

3.1.1 The bisection method

One of the simplest (and oldest) methods to compute roots of a continuous function is the *bisection* method, which finds one zero of $\phi'(\alpha) \in [a, b]$ such that $\phi'(a)\phi'(b) < 0$ up to an accuracy $\epsilon > 0$ following the procedure described here:

1. We initialize $a_0 = a, b_0 = b$.
2. if $b_k - a_k < 2\epsilon$ we set $x^* = (a_k + b_k)/2$ and exit.
3. We compute $m_k = (a_k + b_k)/2$, and we evaluate $\phi'(m)$. If $\phi'(m_k)\phi'(a_k) < 0$ we set $a_{k+1} = a_k, b_{k+1} = m_k$, otherwise we choose $a_{k+1} = m_k, b_{k+1} = b_k$ and we go back to 2.

Exercise 3.1. Show that, assuming $\epsilon < b - a$, the bisection method reaches convergence with at most $\lceil \log_2(\epsilon^{-1}(b-a)) - 1 \rceil$ steps.

In order to compare different methods, we need to introduce the concept of convergence rate. Assume that the sequence $\alpha^{(k)}$ that converges to α^* is generated by some zerofinding or optimization method. We say that the method has *linear convergence* with rate ρ if

$$\lim_{k \to \infty} \frac{|\alpha^* - \alpha^{(k+1)}|}{|\alpha^* - \alpha^{(k)}|} = \rho.$$

It is a rather immediate check to show that the bisection method has indeed a linear convergence behavior, with rate $\rho = \frac{1}{2}$. We shall emphasize that the above definition only describes the convergence of the method "in the tail"; generically we say that the

closer the rate ρ to 0 the better, but this is only true if we start from an approximation $\alpha^{(0)}$ close enough to the solution.

In general, it is difficult to say something about the behavior of the convergence of a method when it's far from the solution. This is not the case with the bisection: we know quite precisely that the error (or at least a worse case estimate) is reduced by a factor 2 at each step.

Better methods can have a higher order convergence. We say that a method has convergence of order p if the following equality holds

$$\lim_{k\to\infty} \frac{|\alpha^* - \alpha^{(k+1)}|}{|\alpha^* - \alpha^{(k)}|^p} = \rho.$$

for some $\rho \neq 0$. If instead the previous limit holds with $\rho = 0$ we simply say that the method has *superlinear* convergence.

3.1.2 Newton's method

Another method that deserved to be name is Newton's method, which is in fact not necessarily restricted to the 1D case. However, for simplicity, we now introduce it for 1D functions, and we generalized it later.

Newton's iteration can be defined as follows:

$$\alpha^{(k+1)} = \alpha^{(k)} - \frac{\phi'(\alpha^{(k)})}{\phi''(\alpha^{(k)})}.$$

On immediately notices that this method requires the evaluation of the derivative of $\phi(\alpha)$. This in turns implies that we need to evaluate the Hessian of the objective function. In fact, again by a straightfoward use of the chain rule we have:

$$\phi''(\alpha) = \alpha^2 d_k^T \nabla^2 f(x_k) d_k.$$

It is rather immediate that the evaluation of a $\phi'(\alpha)$ and $\phi'(\alpha)$ required to perform a single Netwon step is more expensive that just doing a bisection one. However, there is much to gain in the process. Newton's method has quadratic convergence, that is convergence of order 2.

Exercise 3.2. Prove that Newton's method has order 2 if $\phi(\alpha)$ is at least $\mathcal{C}^3$ and $\phi^{(3)}(\alpha^*) \neq 0$. To show that, consider the error equation at step k by subtracting the limit α^* by the Newton's iteration

$$\alpha^{(k+1)} - \alpha^* = \alpha^{(k)} - \alpha^* - \frac{\phi'(\alpha^{(k)})}{\phi''(\alpha^{(k)})}.$$

Then, use the fact that $\phi'(\alpha^*) = 0$ to derive the Taylor expansion

$$\phi'(\alpha^*) = \phi'(\alpha^{(k)}) + (\alpha^* - \alpha^{(k)})\phi''(\alpha^{(k)}) + \frac{1}{2}(\alpha^* - \alpha^{(k)})^2 \phi^{(3)}(\xi),$$

for some ξ that is between α^* and $\alpha^{(k)}$. Conclude by plugging the above expansion in the previous equation.

Exercise 3.3. Show that if $\phi^{(3)}(\alpha^*) > 0$ then the convergence of Newton is monotone (that is, if $\alpha^{(k)}$ is on the right of α^*, so is $\alpha^{(k+1)}$). Prove that if instead $\phi^{(3)}(\alpha^*)$ is negative the convergent sequence jumps right and left of the limit.

3.2 Inexact line search

Quite often, it is appealing to perform a rather inexact line search instead of relying on an accurate method for the solution of the 1D problem. In fact, it is not particularly useful to solve the line search problem very accurately if we are far from the convergence: the restricted problem might be quite meaningless in practice, and we are just trying to improve our estimate.

For this reason, one typically tries to impose loose conditions. However, we do not want these approximation to affect the convergence behavior of the algorithm. As we will see, if careful choices are made, they do not.

3.2.1 Armijo condition

One quite natural condition to be asked is that, when moving along the descent direction, the functions decreases at least a fraction of what "promised" by the gradient.

The first order approximation of our function is given by $f(x_k + \alpha_k d_k) = f(x_k) + \alpha_k \nabla f(x_k) d_k$. If $\nabla f(x_k) d_k < 0$ and so this is a descent direction, we can ask

$$f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k) d_k, \qquad 0 < c_1 < 1. \tag{1}$$

Unfortunately, a problem with the above condition (if not complemented with other strategies) is that arbitrarily close point to x_k satisfy it. Therefore, there is the risk of accepting a step that does not improve the objective function enough.

In order to avoid these problems, we need to introduce other conditions.

3.2.2 The Goldstein condition

A first possibility to complement the Armijo condition is the so-called *Goldstein* condition. We ask that the new x_{k+1} delivers at least c_1 of the promised descent, but we also ask that the approximation is not "too good". In fact, if that's the case, it is likely that the computed gradient is still a good approximation of the current descent speed, and we can proceed further.

In formulas, one can formulate the Armijo-Goldstein conditions as follows:

$$\begin{cases} f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k) d_k, \\ f(x_k + \alpha_k d_k) \geq f(x_k) + (1 - c_1) \alpha_k \nabla f(x_k) d_k, \end{cases} , \qquad 0 < c_1 < \frac{1}{2} \tag{2}$$

This already improves the behavior, but there is the risk that actual local minima are excluded. Consider the example in Figure 1.

This shows that the choice of the parameter c_1 can be quite challenging. For this reason, we introduce the Wolfe conditions, that will give us more guarantees.

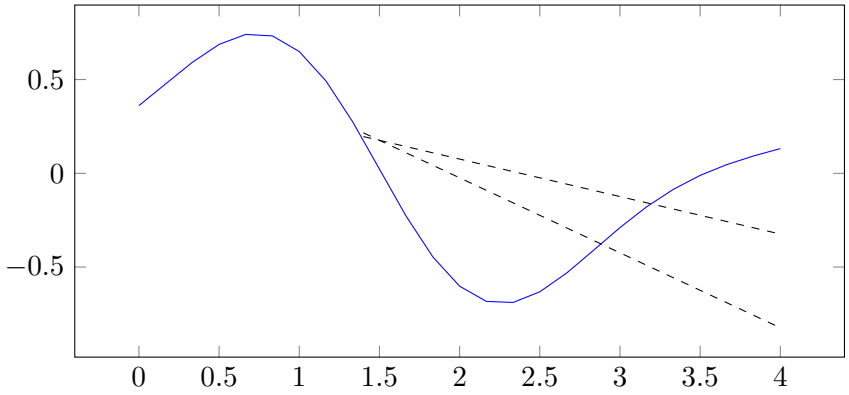


Figure 1: Example of a Goldstein condition where the local minimum is excluded from the feasible points.

3.2.3 Wolfe conditions

Wolfe conditions involve the computation of the gradient at the new point $x_{k+1} = x_k + \alpha_k d_k$. The idea is that, if the gradient there is still quite negative, not far from what we had in x_k, then we can continue to follow the descent direction.

In practice, this can be transformed in the following conditions.

$$\begin{cases} f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k) d_k, \\ \nabla f(x_k + \alpha_k d_k) \geq c_2 \nabla f(x_k) d_k, \end{cases} \quad , \qquad 0 < c_1 < c_2 < 1. \tag{3}$$

In order to avoid the opposite problem, when we go too far and we select a point where the function has began to grow too rapidly again, we can impose the so-called strong Wolfe conditions, where the second inequality is enforced with an absolute value:

$$\begin{cases} f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k) d_k, \\ |\nabla f(x_k + \alpha_k d_k)| \leq c_2 |\nabla f(x_k) d_k|, \end{cases} \quad , \qquad 0 < c_1 < c_2 < 1. \tag{4}$$

Obviously, every point that satisfies (4) also satisfies (3) (recall that, being d_k a descent direction, we have that $\nabla f(x_k) d_k$ is negative).

Let us concentrate on Armijo-Wolfe conditions. Assume that we have a generic $f(x_k)$. Can we always find another point x_{k+1} such that the Armijo-Wolfe condition hold?

Lemma 3.4. *Let $f(x)$ be a $\mathcal{C}^1$ function that is bounded below. Then, for every point x_k, we can always find another point x_{k+1} such that Armijo-Wolfe conditions hold.*

Proof. Let x_k the generic point, and d_k the descent direction. We can define

$$\phi(\alpha) = f(x_k) + \alpha d_k, \qquad \alpha \geq 0$$

which is bounded below since $f(x)$ is bounded below. In particular, we can always find another point $\hat{\alpha}$ such that

$$\phi(\hat{\alpha}) = \phi(0) + c_1 \hat{\alpha} \nabla f(x_k) d_k.$$

To show this, consider $g(\alpha) = \phi(\alpha) - \phi(0) - c_1\alpha\nabla f(x_k)d_k$. For values of α arbitrarily close to 0, we have that $g(\alpha)$ is negative. However, as $\alpha \to \infty$ $g(\alpha) \to \infty$, so there must be a 0 in the middle. Notice that every point in $[0, \hat{\alpha}]$ satisfies the Armijo condition.

As a direct consequence, we can write

$$\phi(\hat{\alpha}) - \phi(0) = c_1\hat{\alpha}\nabla f(x_k)d_k = \hat{\alpha}\phi'(\xi) = \hat{\alpha}\nabla f(x_k + \xi d_k)d_k,$$

thanks to the mean value theorem. Since $c_2 > c_1$ and $\nabla f(x_k)d_k < 0$ we have

$$\nabla f(x_k + \xi d_k)d_k = c_1\hat{\alpha}\nabla f(x_k)d_k > c_2\hat{\alpha}\nabla f(x_k)d_k,$$

and so ξ satisfies the Armijo-Wolfe conditions. □

We can prove that, if we run the steepest descent with them, then we have guaranteed convergence to the stationary point. In fact, the result that we can prove is slightly more general, and it is usually referred to as the Zoutendijk condition.

Theorem 3.5 (Zoutendijk)**.** *Let $f(x)$ be a $\mathcal{C}^1$ function that is bounded below, with a Lipschitz gradient, and such that*

$$x_{k+1} = x_k + \alpha_k d_k$$

is a sequence generated by a method satisfying the Armijo-Wolfe optimality conditions. Let θ_k be the angle beween d_k and $-\nabla f(x_k)$, that is $\cos\theta_k := -\|d_k\|^{-1}\|\nabla f(x_k)\|^{-1}\nabla f(x_k)d_k$. Then,

$$\sum_{j=0}^{\infty} \cos^2\theta_k \|\nabla f(x_k)\|^2 < \infty$$

We will soon prove this result, but before doing it we shall make some comments. The so-called Zoutendijk condition, that is the summation involving the cosines of the angles and the gradients, could look not so meaningful at a first glance. However, let us consider some practical examples that will definitely shed some light on it.

Consider the steepest descent method, where $d_k = -\nabla f(x_k)$. Then, we clearly have that $\cos\theta_k = 1$, and so the Zoutendijk condition becomes that the summation of the square of the gradients is convergent. This implies that they go to zero, so the steepest descent method converges to a critical point.

This is already a strong consequence, but opens the door to many more generalizations. If, for example, one is certain that the cosines are bounded below by $\delta > 0$, which in practice means that the descent directions that we choose are not "too orthogonal" to the descent direction $-\nabla f(x_k)$, then the same result holds.

Let us now draft a proof of the above result (the proof here is taken more or less verbatim from the book Numerical optimization by Nocedal and Wright).

Proof. Notice that the Wolfe condition implies that

$$\nabla f(x_{k+1})d_k - \nabla f(x_k)d_k \geq (c_2 - 1)\nabla f(x_k)d_k,$$

and the hypothesis that the gradient is Lipschitz implies the existence of a constant L such that

$$\nabla f(x_{k+1})d_k - \nabla f(x_k)d_k \leq L\|x_{k+1} - x_k\|\|d_k\| \leq L\alpha_k\|d_k\|^2.$$

Combining these two inequalities yields

$$\alpha_k \geq \frac{c_2 - 1}{L\|d_k\|^2}\nabla f(x_k)d_k.$$

Using the Armijo condition we get

$$f(x_{k+1}) \leq f(x_k) + c_1\alpha_k\nabla f(x_k)d_k \leq f(x_k) + c_1\frac{c_2 - 1}{L\|d_k\|^2}(\nabla f(x_k)d_k)^2 \leq f(x_k) - c\cos^2\theta_k\|\nabla f(x_k)\|^2,$$

where c is an appropriate positive constant (notice that $c_2 - 1$ is negative). Therefore, we can expand the above several times (or by induction, if you prefer), to obtain

$$\frac{f(x_0) - f(x_{k+1})}{c} \geq \sum_{j=0}^{\infty}\cos^2\theta_k\|\nabla f(x_k)\|^2,$$

and since the left hand side is bounded (recall that $f(x)$ is bounded below), so is the right hand side, and we have the thesis. □

Exercise 3.6. Let $x_{k+1} = x_k + \alpha_k d_k$ be an optimization method for a function that satisfies the hypotheses in Theorem 3.5, and assume that the descent directions are chosen by $d_k = -B_k^{-1}\nabla f(x_k)$, with B_k positive definite and with $\|B_k\|\|B_k^{-1}\| \leq M$. Prove that this method is convergent (In practice, prove that $\cos\theta_k \geq \delta$ for some strictly positive value δ).