CM4L&DA

Mathematical Optimization

Authors:
Those who must not be named

Date: somewhere during the end of the second decade of the twenty-first century...

Contents

Chapter 1

Introduction

These notes are entirely based on the material of the course Computational Mathematics for Learning and Data Analysis held by Professor Antonio Frangioni and Professor Federico Poloni at University of Pisa. The course is part of the M.Sc. degree in Computer Science.

1.1 Mathematical Background

1.1.1 Notations and Nomenclatures

Let's start with introducing some notations and nomenclatures that we are going to use throughout this book. Given a set $\mathcal{X}$, that we call **feasible region**, and a function $f : \mathcal{X} \to \mathbb{R}$, that we call **objective function**, we want to find the solution to the following problem:

$$f_* = \min_{x \in \mathcal{X}} \{f(x)\} \tag{1.1}$$

where f_* is called **optimal value**. The argument x that *minimises* the objective function f is called **optimal solution**. It is generally denoted with x_*.

Note that: f_* lives in the output space while x lives in the input space. This is going to be very important where we speak about convergence of the various algorithms that we will see during this brief journey.

Obviously, from the moment that we are minimising the function f, we have the following property:

$$\forall\ x \in \mathcal{X}\ f(x) \geq f(x_*)$$

Here again, note that we do not state any ordering relation in the input space; x may be larger, equal or even smaller than x_*.

As we will see in the next chapters, we often employ *iterative methods* for finding the optimal value/optimal solution. Applying an iterative method implies producing a certain sequence of the form $x_1, x_2, ...$ that will bring us to the optimum solution; and thus to the optimal value. Typically we cannot hope to get exactly to f_*, what we hope instead is to get *as close as possible* to it. Accordingly, limits are very important to us. We say that:

$$\lim_{i\to\infty} x_i = x \iff \forall\epsilon > 0\ \exists h : |x_i - x| \leq \epsilon\ \forall i \geq h \tag{1.2}$$

In other words, the sequence $\{x_i\}$ converges to x, written as $\{x_i\} \to x$, if and only if... the rest of the formula :). Just kidding. So we were saying, the sequence $\{x_i\}$ converges to x if and only if for *any* tiny number ϵ that we can choose (as long as it is greater than 0), at some point in our sequence, starting from the number x_h, all the numbers in it will be at distance at most ϵ from x.

Not all sequences have the limit. For instance think of $(-1)^i$. This sequence does not converge to neither of -1, 0 or 1. This is the reason why we introduce the concept of **infima** and **suprema**, or respectively $\liminf$ and $\limsup$. We define the following two quantities:

$$\{x_i\} \to \underline{x}_i = \inf\{x_h : h \geq i\}$$

and

$$\{x_i\} \to \bar{x}_i = \sup\{x_h : h \geq i\}$$

Now if we take the sequence of $\{\underline{x}_i\}$ we are sure that $\underline{x}_1 \leq \underline{x}_2 \leq \underline{x}_3 \leq ...$ and the sequence of $\{\bar{x}_i\}$ we are sure that $\bar{x}_1 \geq \bar{x}_2 \geq \bar{x}_3 \geq ....$ Accordingly, they have a limit! So we can speak about inferior and superior limit of a certain sequence $\{x_i\}$. We have the following equality:

$$\liminf_{i\to\infty} x_i = \lim_{i\to\infty} \inf\{x_h : h \geq i\} = \lim_{i\to\infty} \underline{x}_i \tag{1.3}$$

$$\limsup_{i\to\infty} x_i = \lim_{i\to\infty} \sup\{x_h : h \geq i\} = \lim_{i\to\infty} \bar{x}_i \tag{1.4}$$

Now obviously $\bar{x}_i \geq \underline{x}_i$. But if these two quantities are equal, then we are sure that the limit of the sequence exists:

$$\lim_{i\to\infty} x_i = v \iff \liminf_{i\to\infty} x_i = v = \limsup_{i\to\infty} x_i$$

Now the life would be pretty straightforward and we would not need to study optimisation if we worked just with single numbers, i.e. in $\mathbb{R}$. We need to scale.

1.1.2 Euclidean Space $\mathbb{R}^n$

We will mostly use function that operate on vectors not numbers. We need to define the space in which these vectors have some meaning. We thus define the **Euclidean Space** as the Cartesian product of n one dimensional spaces $\mathbb{R}$:

$$\mathbb{R}^n = \underbrace{\mathbb{R} \times ... \times \mathbb{R}}_{n \text{ times}} \tag{1.5}$$

The elements of this space are vectors of n components, i.e. $x \in \mathbb{R}^n = [x_1, ..., x_n]$.

Note: during this book we will always assume to work with **column vectors**. When we use **row vectors** we will use the transposition sign.

The Euclidean space is **closed** under summation and scalar multiplication. This means that whenever we take two vectors and we sum them, or we take a constant α and we multiply all the components of a vector with α, we will always get a new vector that is for sure still in our Euclidean space.

The Euclidean space is a **finite** vector space: this means that each $x \in \mathbb{R}^n$ can be obtained from a **finite basis**. A basis is a set of vectors such that for whatever vector x in the space, we can somehow linearly combine the vectors from the basis to obtain x. Linearly combine means summations and scalar multiplications.

The Euclidean space is *not a totally ordered set.* We do not have a natural concept of which one comes first.

Since we will work a lot with distances we also need to define the concept of distance in this space. Before talking about the distance we need to introduce a very important concept that we will use constantly. **The norm**. We define Euclidean norm as:

$$\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{1.6}$$

Some of the properties of the Euclidean norm are:

- $\|x\| \geq 0 \ \forall x \in \mathbb{R}^n$ with 0 only in case that x is the zero vector
- $\|\alpha x\| = |\alpha| \|x\| \ \forall x \in \mathbb{R}^n \ \forall \alpha \in \mathbb{R}$
- $\|x + y\| \leq \|x\| + \|y\| \ \forall x, y \in \mathbb{R}^n$. This property is called **triangular inequality**.

The norm function is nothing more than a simple function that maps vectors to numbers (also matrices to numbers). Apart from the norm we have defined in the equation 1.6, we have also:

- 0-norm $\|.\|_0 = |i : |x_i| > 0|$, i.e. the cardinality of the set of all non zero components in the vector x
- 1-norm $\|.\|_1 = \sum_{i=1}^{n} |x_i|$
- ∞-norm $\|.\|_\infty = \max_{i \in I}\{|x_i|\}$

All these norms (including 2-norm) are the special cases of the unique definition of the p-norm:

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} \tag{1.7}$$

In figure 1.1 we show how the function of the norm changes as we augment p. Note that for $p \geq 1$ we have a convex function while for $p < 1$ we have a concave function.

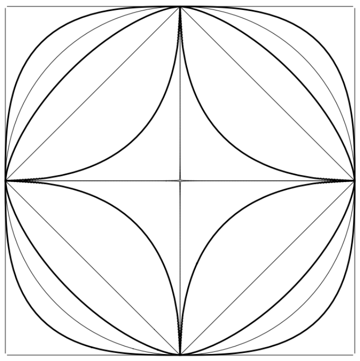


Figure 1.1: The various types of norms

It is not very important which norm we are using since all of them are topologically equivalent. This means that whatever two norms we choose, there is a simple linear relation between them:

$$\forall \ \|.\|_x, \|.\|_y \ \exists \ 0 < \alpha < \beta : \alpha\|.\|_y \leq \|.\|_x \leq \beta\|.\|_y \tag{1.8}$$

Accordingly, we are free to work with whatever norm. We will generally use the two norm, unless stated otherwise.

Another concept we have to introduce before defining the distance in the Euclidean space is the **scalar product** between two vectors. We define the scalar product as:

$$\langle x, y\rangle = y^T x = \sum_{i=1}^{n} = x_i y_i \tag{1.9}$$

Like the norm function, the scalar product is a simple function that takes two vectors and returns the value, i.e. $\langle . \rangle : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$. It satisfies the following properties ($\forall \ x, y, z \in \mathbb{R}^n, \forall \ \alpha \in \mathbb{R}$):

- $\langle x, y\rangle = \langle y, x\rangle$ (symmetry)
- $\langle x, y\rangle \geq 0$
- $\langle x, x\rangle = 0 \iff x = 0$
- $\langle \alpha x, y\rangle = \alpha\langle x, y\rangle$
- $\langle x + y, z\rangle = \langle x, z\rangle + \langle y, z\rangle$
- $\langle x, x\rangle = \|x\|^2$ (only for $\|.\|_2$)
- $\langle x, y\rangle^2 \leq \langle x, x\rangle\langle y, y\rangle = \|x\|^2\|y\|^2$. This is a very important property and it is called **Cauchy-Schwarz inequality**

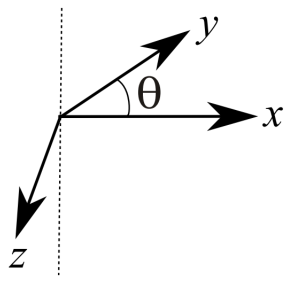


Figure 1.2: Angle θ between x and y

The scalar product have another definition, the one that uses the cosine function:

$$\langle x, y \rangle = \|x\| \|y\| \cos \theta \tag{1.10}$$

From this definition it is clear that when two vectors are perpendicular to each other then the scalar product is 0 (figure 1.2). When the two vectors point to the same direction then the scalar product is > 0. Cosine of an angle that is between 0 and $\frac{\pi}{2}$ is positive, thus the scalar product is positive (the norms are always ≥ 0). This will be very useful when we will speak about various descent methods for unconstrained optimisation.

We are now ready to introduce the **Euclidean distance**. The Euclidean distance between $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$ is defined as:

$$d(x, y) = \|x - y\| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{1.11}$$

The euclidean distance satisfies the following properties ($\forall\ x, y, z \in \mathbb{R}^n$ and $\forall\ \alpha \in \mathbb{R}$):

- The distance between two vectors is always ≥ 0: $d(x, y) \geq 0$
- $d(\alpha x, y) = |\alpha| d(x, y)$
- $d(x, y) \leq d(x, z) + d(z, y)$ (triangular inequality)

So the distance between two vectors is given by the norm of the vector difference between them.

We concept of Euclidean distance is useful for defining a very important mathematical object in optimisation.

1.1.3 Ball, Interior, Boundary and Bounded sets

Definition 1.1.1. *Given a centre $x \in \mathbb{R}^n$ and a radius $r > 0 \in \mathbb{R}$, we call **Ball**:*

$$\mathcal{B}(x, r) = \{y \in \mathbb{R}^n : d(x, y) \leq r\}$$

Having these new mathematical objects, we can restate the concepts of converging sequences as:

$$\{x_i\}_{i\to\infty} \to x \iff \forall\epsilon > 0\ \exists h : d(x_i, x) \leq \epsilon\ \forall i \geq h$$

or

$$\{x_i\}_{i\to\infty} \to x \iff \forall\epsilon > 0\ \exists h : x_i \in \mathcal{B}(x, \epsilon)\ \forall i \geq h$$

or even

$$\lim_{i\to\infty} d(x_i, x) = 0$$

Why we are so interested in these sequences of points? Because if we consider the optimisation problem again:

$$f_* = \min_{x\in\mathcal{X}} \{f(x)\}$$

we want to construct the so called *minimising sequence* $\{x_i\}_{i\to\infty}$ such that $\{f(x_i)_{i\to\infty}\} \to f_*$.
Note again (again, yes) that $\{f(x_i)_{i\to\infty}\} \to f_* \not\Rightarrow \{x_i\}_{i\to\infty} \to x_*$. These are two different problems.

It is not sufficient to just solve analytically a problem. Consider for instance the following problem:

$$\min\{x : x \in \mathbb{R}^n \wedge x > 0\}$$

The optimal solution f_* is infinitely close to 0 but cannot be reached by the sequence $\{x_i = \frac{1}{i}\}_{i\to\infty}$ ($\{f(x_i)_{i\to\infty}\} \to 0$). What can we do in order to avoid this situation. Generally we cannot. What we can do, as we will see during this book, is to require certain properties on the set or functions on which we optimise in order to guarantee that certain situations can never happen.

Definition 1.1.2. *Given $\mathcal{S} \subseteq \mathbb{R}^n$ we define:*

$$\text{interior of } \mathcal{S} \equiv int(\mathcal{S}) = \{x : x \in \mathcal{S} \wedge \exists r > 0 : \mathcal{B}(x, r) \subseteq \mathcal{S}\} \tag{1.12}$$

$$\text{boundary of } \mathcal{S} \equiv \partial(\mathcal{S}) = \{x : \forall r \exists y, z \in \mathcal{B}(x, r) : y \in \mathcal{S} \wedge z \notin \mathcal{S}\} \tag{1.13}$$

Definition 1.1.3. *The set $\mathcal{S}$ is said to be open $\iff$ $\mathcal{S} = int(\mathcal{S})$.*

Definition 1.1.4. *The set $\mathcal{S}$ is said to be closed $\iff$ $\mathcal{S} = int(\mathcal{S}) \cup \partial\mathcal{S}$. Or equivalently if $\mathbb{R}^n \setminus \mathcal{S}$ is open set.*

The union of $int(\mathcal{S}) \cup \partial\mathcal{S}$ is called *closure* of the set $\mathcal{S}$.

Exercise: Prove that both $\mathbb{R}^n$ and $\emptyset$ are both open and closed.
Let us start from the easiest one: empty set. In order to be open its interior should be equal to itself. The interior of an empty set is empty, thus they are equal. In order to be a closed set, it should be equal to the union of its interior and its boundary. But both of these are empty sets, thus the union is an empty set again.
Now consider the case of $\mathbb{R}^n$. This set does not have a boundary, $\partial(\mathbb{R}^n) = \emptyset$.

Thus it is equal to its interior. Thus it is open. The union of its interior and an empty set is equal to its interior. Thus it is also closed.

Exercise: Exhibit a set that is neither closed nor open.
TODO

The concept of the minimising sequences and closed sets can be connected by the following property:

Theorem 1.1.1. *S is closed* $\iff \forall\{x_i\} \subset S \to x \Rightarrow x \in S$

This basically means that S is closed if and only if all accumulation points of sequences in S are in S.

Suppose we have an infinite sequence of sets $\{S_i\}_{i\to\infty}$. If the sets are all open, then their union is also an open set.

$$\cup_{i\in I} S_i \text{ is open} \tag{1.14}$$

On the other hand, if the sets are all closed, then their intersection is also closed.

$$\cap_{i\in I} S_i \text{ is closed} \tag{1.15}$$

Note that the union of an infinite family of closed sets is not necessarily a closed set and that the intersection of an infinite family of open sets is not necessarily an open set. As a counterexample, suppose we define a family of open sets in the following manner:

$$D_i = \mathcal{B}(x_0, \frac{1}{i})\ i \in \mathbb{N} - \{0\} \tag{1.16}$$

Then the intersection of these sets tends to a set with only one point (0), which is obviously not an open set. The same reasoning can be applied to the union of a family of closed sets.

Another important concept when talking about balls and sets, is the concept of a **Bounded set**.

Definition 1.1.5. *$S \subseteq \mathbb{R}^n$ is called Bounded if $\exists r > 0 : S \subseteq \mathcal{B}(0, r)$.*

This translates into: S is a Bounded set if we can choose a radius r strictly greater than 0 such that we can create a ball with centre in 0 so that it fully contains the whole S.

Definition 1.1.6. *If a set S is both closed and bounded, it is called* ***Compact****.*

1.1.4 Bolzano-Weierstrass Theorem

TODO BETTER

Before announcing this important theorem, we need to introduce the concept of subsequence.

Definition 1.1.7. *Given a sequence of points $\{x_i\} \in \mathbb{R}^n$ and a sequence of indexes $\{n_i\} \in \mathbb{N}$, the sequence $\{x_{n_i}\} \subseteq \{x_i\}$ is called a subsequence.*

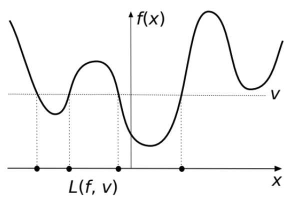


Figure 1.3: Level set

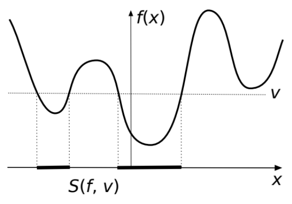


Figure 1.4: Sublevel set

The **Bolzano–Weierstrass theorem**, named after Bernard Bolzano and Karl Weierstrass, is a fundamental result about convergence in a finite-dimensional Euclidean space $\mathbb{R}^n$. The theorem states that each bounded sequence in $\mathbb{R}^n$ has a convergent subsequence. An equivalent formulation is that a subset of $\mathbb{R}^n$ is sequentially compact if and only if it is closed and bounded. The theorem is sometimes called the sequential compactness theorem.

1.1.5 Functions

Remember that our problem is to minimise functions (equation 1.1). As we can see, we have two mathematical objects in this equation: the set $\mathcal{X}$ and the function f. Up until now we have discussed about sets and the properties we would like to have on them in order to make our life simpler. Now we need to do the same thing with the functions.

Note that the functions live in a different space ($\mathbb{R}^{n+1}$) w.r.t. their inputs ($\mathbb{R}^n$). Nevertheless, sometimes we'd like to see how sequences of the functional values behave in the input space. To this end there is a concept of **level sets** and **sublevel sets**.

Definition 1.1.8. *The set $L(f, v) = \{x \in dom(f) : f(x) = v\}$ is called level set of the function f and level v (figure 1.3).*

Definition 1.1.9. *The set $S(f, v) = \{x \in dom(f) : f(x) \leq v\}$ is called sublevel set of the function f and level v (figure 1.4).*

In case that f is unbounded below, we cannot actually talk about sublevel sets. We can reverse the concept and talk about superlevel sets.

The first property that we'd like that our functions that we minimise have is continuity.

Definition 1.1.10. *$f : \mathbb{R}^n \to \mathbb{R}$ is continuous in $x \in \mathbb{R}^n$ if*

- $\{x_i\} \to x \Rightarrow \{f(x_i) \to f(x)\}$
- $\forall \epsilon > 0 \ \exists \delta > 0 : |f(y) - f(x)| < \epsilon \ \forall y \in \mathcal{B}(x, \delta)$

We say that f is continuous on a set $S \subseteq \mathbb{R}^n$ if it is continuous on each $x \in S$.

Theorem 1.1.2 (Intermediate Value Theorem)**.** *Suppose we have function $f : \mathbb{R} \to \mathbb{R}$ continuous on a closed interval $[a, b]$. Then:*

$$\forall v : \min\{f(a), f(b)\} \leq v \leq \max\{f(a), f(b)\} \ \exists c \in [a, b] : f(c) = v$$

All the considerations pointed out so far bring us to the Weistrass extreme value theorem. This theorem is one of the most important in optimisation.

Theorem 1.1.3 (Weistrass Extreme Value Theorem)**.** *Suppose we have a set $X \subseteq \mathbb{R}^n$ compact and a function f that is continuous, then the minimising problem has an optimal solution.*

From this statement, this theorem doesn't seem to be so meaningful. But, if we exploit a little bit some of the considerations explained so far, we can say that this statement is equivalent to say that: if our set is closed and our function continuous, then all the accumulation points of any minimising sequence are optimal solutions and there is at least one of them. Up until now, we have understand that having a continuous function is a really great thing and for this reason, in a lot of the methods we'll consider, the basic assumptions will be: a compact set and a continuous function. But, if we are really lucky, we can ask for more. More we have, simpler it is to gain better results. Talking about continuity, a more strict version of it is Lipschitz continuity.

Theorem 1.1.4 (Lipschitz continuity)**.**

$$\exists L > 0 \quad \forall x, y \in X : |f(x) - f(y)| \leq L\|x - y\|$$

Since this is a stronger definition of continuity, all the considerations said so far for the continuity case are still true. Moreover, with L continuity, we know that our function doesn't jump wildly. This is a very good result, but unfortunately, a lot of functions are not L continuous. If this is the case, we could see if our function is lower semi continuous (or upper).

Now, we have to remember our main task: find the minimum of an iterative sequence. For simplicity, we focus our attention on functions in two dimensions. We know that a derivative gives us a linear approximation of the function in that point. So, if I compute the derivative in x, in which way of the linear approximation ($f'(x)$) should I go? Of course, we have to go where the line decrease: if the slope of my line is greater than zero, I have to go left, otherwise right. I can stop to iterate when I find $f'(x) = 0$ (since I know that in this case we are on a saddle point or a max or min).

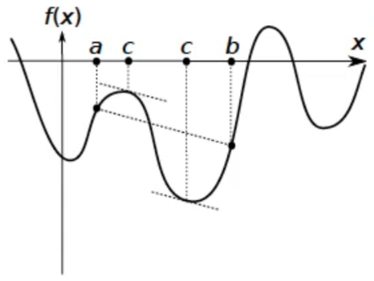


Figure 1.5: A differentiable function $f(x)$ and two points a and b.

The saddest part of it is that not all the function have a first derivative and that even if they have it, it may be the case that it isn't continuous. In order to understand when a function is differentiable or not, the definition of derivative is necessary: TODO

1.1.6 Useful results about derivatives

Suppose you have a differentiable function like in figure 1.5, with two points and a line connecting those two points. Then according to the **mean value theorem** we have the following:

Theorem 1.1.5. *Given a function $f : \mathbb{R} \to \mathbb{R}$ continuous on $[a, b]$ and differentiable on (a, b) then there exists a point $c \in (a, b)$ such that the slope of the derivative in point c is the same of the slope of the line connecting a and b. In other words:*

$$f'(c) = \frac{f(b) - f(a)}{b - a} = m$$

where m is the usual slope of a line.

Why do we care about this simple theorem? Because there is a special case of it, namely when $f(a) = f(b)$. This takes the name of **Rolle's theorem** (see figure 1.6) and it says:

Theorem 1.1.6. *Given a function $f : \mathbb{R} \to \mathbb{R}$ continuous on $[a, b]$ and differentiable on (a, b), if $f(a) = f(b)$ then there exists a point $c \in (a, b)$ such that the slope of the derivative in point c equals 0.*

This theorem assures us that if $f(a) = f(b)$ in the middle there must me at least one local minimum or maximum, or both.

Actually, we can even state that between two consecutive roots of f there is an odd number of roots of the derivative f' in the range of (a, b). If a and b are two consecutive roots of f then the sign of their derivatives are opposite to each other.

Exercise. Prove that between two consecutive roots of f' there is at most one root of f.

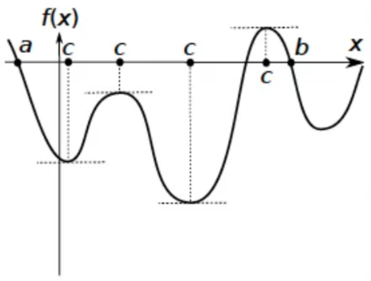


Figure 1.6: Rolle's theorem.

Suppose a and b are two consecutive roots of f'. Suppose now by contradiction that there are two different points $c, d \in (a, b)$ that are also the roots of f. According to Rolle's theorem, there must exist at least one point $e \in (c, d)$ such that $f'(e) = 0$. This contradicts the fact that a and b are two consecutive roots of f'.

1.1.7 A different interpretation of the gradient

Remember that the function is a mathematical object that lives in $\mathbb{R}^{n+1}$. The gradient is the linear function approximating f in a point. Thus, the gradient it self lives again in $\mathbb{R}^{n+1}$. It is important also to look at it from the input space, i.e. in $\mathbb{R}^n$. We can do that via level sets again.

The gradient turns out to be *normal* to the level sets. In other words, it is tangent to the level set.

1.1.8 Vector valued function

Chapter 2

Unconstrained Optimisation

2.1 Local minimums

Now we want to start studying the unconstrained optimisation problems. The optimisation problems are always conducted over a set of values, and the most simple form of optimisation problems are those whose sets are not constrained at all. Thus the name of *unconstrained* optimisation.

The problem that we deal with is typically described by the following syntax:

$$f_* = \min\{f(x) : x \in \mathbb{R}^n\} \tag{2.1}$$

In these kind of problems only the objective function accounts. Since the set is unbounded, the Weierstrass theorem cannot be applied. Many problems in Machine Learning are of this very kind. Thus, these problems are quite important to study. Moreover, the things that we are going to develop will also be very important for the more difficult version of optimisation problems, that is the constrained version. You need to be able to do the unconstrained optimisation in order to do the constrained optimisation.

Note that in unconstrained optimisation, the set over which the function is defined is the whole space $\mathbb{R}^n$. This implies that we cannot use Weierstrass theorem.

If we aim to find a global minimum of a problem that does not have any special property, is a very hard to solve. Even when we know that a function admits a global minimum, it may be very hard to recognise it. Actually this is usually the major problem. Detecting *a* minimum may be easy, recognising (proving that it is actually a global minimum is what is difficult (see figure 2.1). We won't touch this argument in this course, but for not fainted heart look into globalisation techniques.

For this reason, we usually start simple and small: find a local minimum.

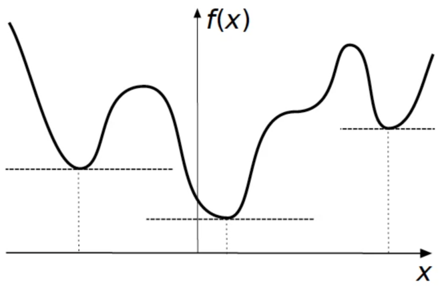


Figure 2.1: There are different minimums, which one is local and which one is global? It depends where we start looking for it.

We say that x_* is a local minimum if it solves the problem:

$$\min\{f(x) : x \in \mathcal{B}(x_*, \epsilon)\} \tag{2.2}$$

for some $\epsilon > 0$. We say it is a strict local minimum if: $f(x) < f(y)\ \forall y \in \mathcal{B}(x_*, \epsilon)$.

If the objective function f is differentiable, then we already know how to compute it. If x_* is a local minimum then its first order derivative (the gradient ∇) is zero:

$$x^* \textit{ is local minimum} \Rightarrow \nabla f(x^*) = 0 \tag{2.3}$$

This is called **first order (necessary, local) optimality condition**.

Proof. By contradiction. Suppose that x^* is a local minimum and that $\nabla f(x^*) \neq 0$. Since the gradient is not 0, we can perform a step in the opposite direction in order to lower a little bit the objective function:

$$f(x - \alpha\nabla f(x)) \tag{2.4}$$

Let us now consider the Taylor extension (first order) of f in the point $x - \alpha\nabla f(x)$:

$$\begin{aligned} f(x - \alpha\nabla f(x)) &= f(x) + ((x - \alpha\nabla f(x)) - x)\nabla f(x) + R(-\alpha\nabla f(x)) = \\ &= f(x) - \alpha\|\nabla f(x)\|^2 + R(-\alpha\nabla f(x)) \end{aligned} \tag{2.5}$$

where:

$$\lim_{h\to 0} \frac{R(h)}{\|h\|} = 0 \tag{2.6}$$

The point here is that residual goes to 0 quicker than linear. In other words, if we consider the limit in α (because we are moving a little bit further from x^*), the limit must obey to the following relation:

$$\lim_{\alpha\to 0} \frac{R(-\alpha\nabla f(x))}{\| -\alpha\nabla f(x)\|} = \frac{R(-\alpha\nabla f(x))}{\alpha\|\nabla f(x)\|} \tag{2.7}$$

But what is the definition of the good old limit?

$$\forall \epsilon \ \exists \bar{\alpha} \ s.t. \ \frac{R(-\alpha \nabla f(x))}{\alpha \|\nabla f(x)\|} \leq \epsilon \ \forall \alpha \ 0 < \alpha < \bar{\alpha} \tag{2.8}$$

We can choose ϵ however small, as long as it is greater than 0. So let's take ϵ strictly less than $\nabla f(x)$. This is possible since we assumed it is $\neq 0$.

$$\epsilon < \|\nabla f(x)\| \tag{2.9}$$

But then we have that:

$$\forall \epsilon \ \exists \bar{\alpha} \ s.t. \ \frac{R(-\alpha \nabla f(x))}{\alpha \|\nabla f(x)\|} \leq \epsilon < \|\nabla f(x)\| \ \forall \alpha \ 0 < \alpha < \bar{\alpha} \tag{2.10}$$

This leads us to:

$$\forall \epsilon \ \exists \bar{\alpha} \ s.t. \ R(-\alpha \nabla f(x)) \leq \epsilon < \alpha \|\nabla f(x)\|^2 \ \forall \alpha \ 0 < \alpha < \bar{\alpha} \tag{2.11}$$

Finally we get the contradiction:

$$\begin{aligned} f(x - \alpha \nabla f(x)) &= f(x) - \alpha \|\nabla f(x)\|^2 + R(-\alpha \nabla f(x)) = \\ &= f(x) - \alpha \|\nabla f(x)\|^2 + R(-\alpha \nabla f(x)) < f(x) \end{aligned} \tag{2.12}$$

which holds $\forall \alpha < \bar{\alpha}$. □

Note that in order to be able to to the first order Taylor extension, we need the function to be continuous, in other words $f \in C^1$.

If we are in a point whose first order derivative is not 0, then we can consider a ball around it, however small, we can make a step into anti-gradient direction in order to lower a bit the objective function value.

This is actually only a necessary condition and not also the sufficient. If we have a point x where the first order derivative is 0 then we know only that x is a stationary point. A stationary point is a point on the graph where the function stops increasing or decreasing. Basically, it could also be a simple inflection (saddle) point or a maximum (local or global). They all look the same when you consider the first order information. That's why we say that the problem is not finding but rather recognising the global minimum. The techniques that are used for distinguish between local and global minimums are called globalisation techniques. We won't touch them in this course.

Take away is: whenever we are dealing with nasty functions, we will not require a global minimum. Only under certain conditions, when it is easy, we will deal with global minimums.

To distinguish between saddle points, maximums and minimums, we need to look into the Hessian (second order derivative, i.e. derivative of the gradient). Positive definite Hessian identifies a strict minimum point, a positive negative Hessian identifies a strict maximum point, a indefinite Hessian identifies a saddle point.

$$x^* \ \textit{is a local minimum} \Rightarrow \nabla^2 f(x^*) \succeq 0 \tag{2.13}$$

This is called **Second order (necessary, local) optimality condition**. Let us now prove this.

Proof. By contradiction. Assume that x^* is a local minimum and that its Hessian is not positive semi-definite, i.e. $\nabla^2 f(x^*) \prec 0$. If the Hessian is negative, this means that there is a vector d with $\|d\| = 1$ such that:

$$d^T \nabla^2 f(x^*) d \leq 0 \tag{2.14}$$

At the same time, since we are talking about a stationary point, we know that the gradient in x^* is 0, i.e. $\nabla f(x^*) = 0$. Let us now consider the Taylor expansion up to the second level:

$$\begin{aligned} f(x^* + \alpha d) &= f(x^*) + ((x^* + \alpha d) - x^*)\nabla f(x^*) + \\ &+ \frac{1}{2}((x^* + \alpha d) - x^*)^2 \nabla^2 f(x^*) + R(\alpha d) = \\ &= f(x^*) + \frac{1}{2}\alpha^2 d^T \nabla^2 f(x^*) d + R(\alpha d) \end{aligned} \tag{2.15}$$

where:

$$\lim_{\alpha \to 0} \frac{R(\alpha d)}{\|\alpha d\|^2} = \lim_{\alpha \to 0} \frac{R(\alpha d)}{\alpha^2 \|d\|^2} = \lim_{\alpha \to 0} \frac{R(\alpha d)}{\alpha^2} = 0 \tag{2.16}$$

That is, the remainder goes to 0 quicker that the quadratic function in α.

By applying now the definition of the limit we get:

$$\forall \epsilon > 0 \ \exists \bar{\alpha} > 0 \ s.t. \ R(\alpha d) \leq \epsilon \alpha^2 \ \forall \alpha < \bar{\alpha} \tag{2.17}$$

Let us now take $0 < \epsilon < -\frac{1}{2} d^T \nabla^2 f(x^*) d$. Remember that our Hessian is negative definite and that we assumed that d is mapped to a negative number. If we choose such an ϵ, we get the following:

$$R(\alpha d) < -\alpha^2 \frac{1}{2} d^T \nabla^2 f(x^*) d \tag{2.18}$$

and so:

$$f(x^* + \alpha d) = f(x^*) + \frac{1}{2}\alpha^2 d^T \nabla^2 f(x^*) d + R(\alpha d) < f(x^*) \ \forall \alpha < \bar{\alpha} \tag{2.19}$$

□

Note that in order to be able to to the second order Taylor extension, we need the function to be continuously differentiable, in other words $f \in C^2$.

If we want to be sure that we are in a local minimum, we need to require the Hessian to be strictly positive definite, not positive semi-definite. That is, we have the **Second order (sufficient, local) optimality condition**:

$$f \in C^2 \wedge \nabla f(x^*) = 0 \wedge \nabla^2 f(x^*) \succ 0 \Rightarrow x^* \ \textit{is a local minimum} \tag{2.20}$$

Proof. Let us consider again the second order Taylor expansion:

$$f(x^* + d) = f(x^*) + \frac{1}{2} d^T \nabla^2 f(x^*) d + R(d) \tag{2.21}$$

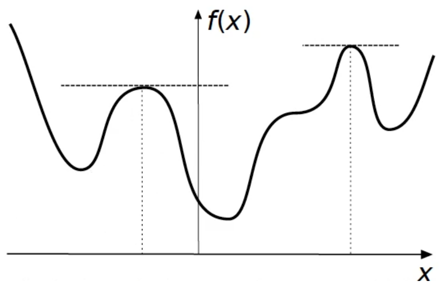


Figure 2.2: Between a local minimum and a global minimum, there must be a local/global maximum.

again with:

$$\lim_{d\to 0} \frac{R(d)}{\|d\|^2} = 0 \tag{2.22}$$

Since the Hessian is positive definite, we know that all its eigenvalues are strictly positive and that every vector d is mapped to another space with the following relation:

$$\lambda_{\min}\|d\|^2 \le d^T\nabla^2 f(x^*)d \le \lambda_{\max}\|d\|^2 \tag{2.23}$$

Consider again the definition of the limit:

$$\forall\epsilon > 0 \ \exists\delta > 0 \ s.t. \ R(d) \le \epsilon\|d\|^2 \ \forall d \ \|d\| < \delta \tag{2.24}$$

Let us now take $\epsilon < \lambda_{\min}$. We then get:

$$f(x^* + d) = f(x^*) + \frac{1}{2}d^T\nabla^2 f(x^*)d + R(d) \ge f(x^*) + (\lambda_{\min} - \epsilon)\|d\|^2 > f(x^*) \tag{2.25}$$

This means that in each direction we take from x^* we end up in a place where the value of the objective function is strictly greater than in x^*. This means that we have a strictly quadratic function that locally (in a ball $\mathcal{B}(x^*, \epsilon)$) approximates our function. □

Unfortunately, these conditions are also locally defined. They cannot be applied to the global case. Some properties need to be valid on our problem in order to be able to find global minimums.

2.2 Towards global minimum

Let us start with an intuition: if we are in the local minimum that is not the global minimum, then we necessarily have to grow from that local minimum and then decrease towards a global minimum (figure 2.2). In other words, there has to be a local maximum before the global minimum (remember Rolle's theorem).

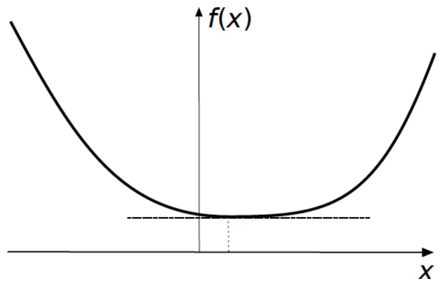


Figure 2.3: An example of a convex function.

We want our function to have only local minimums in the stationary points. That is each time we are in a stationary point, we have a positive semi-definite matrix. In formula:

$$\nabla f(x) = 0 \Rightarrow \nabla^2 f(x) \succeq 0 \tag{2.26}$$

The sufficient condition for having 2.26 is to have the positive semi-definite Hessian on the whole space over which our function is defined. That is:

$$\forall x \in \mathbb{R}^n \dot{\nabla}^2 f(x) \succeq 0 \tag{2.27}$$

The functions satisfying this property are called **Convex Functions** (figure 2.3).

Optimising convex functions is usually an easy problem (not always). Optimising non convex functions is instead a much more difficult problem. Infact in ML usually one always tries to build a convex model because every local minimum is also a global minimum, mathematically true. Whenever this is not possible, one usually does not require a global minimum and gets satisfied with a local one.

2.3 Convexity

2.3.1 Introduction

Given two points x, y in a space $\mathbb{R}^n$, we define the segment (set of points) joining those two points with the following equation:

$$conv(x, y) = \{z = \alpha x + (1 - \alpha)y \ : \ \alpha \in [0, 1]\} \tag{2.28}$$

Given a set, we say that it is a **convex set**, if any time we take two points inside the set, the segment joining these two points is entirely contained in the set (see figure 2.4). Similarly, if the set does not contain the whole segment, the set is not a convex set (see figure 2.5).

The nice thing is that every non convex set can be made a convex set. Basically, you perform a closure operation until there are some points to be added (see figure 2.6). There are actually in infinite number of sets that "complete"

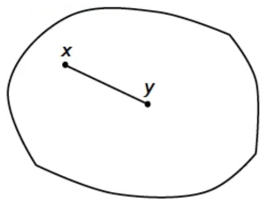


Figure 2.4: An example of the convex set with the segment joining two points entirely contained in the set.

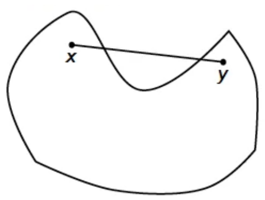


Figure 2.5: An example of the non convex set.

the non convex set, but the smallest one is of our interest and it is called **convex hull**.

Clearly, we can generalise the concept from above. Instead of using two points, we can use an arbitrarily large number of points.

$$conv(\{x_1, ..., x_k\}) = \{x = \sum_{i=1}^{k} \alpha_i x_i \ : \ \sum_{i=1}^{k} \alpha_i = 1 \wedge \forall \alpha_i \geq 0\} \tag{2.29}$$

We then have the following relation:

$$C \ is \ convex \iff conv(\{x_1, ..., x_k\}) \subseteq C \ \forall x_1, ..., x_k \in C \tag{2.30}$$

In other words, a set C is convex if every time we take k elements and we make a convex hull of them, that convex hull is completely inside the C.

Exercise. Close and convex properties are not the same. Provide an example of a set that is closed but not convex and viceversa. Moreover, prove that:

$$C \ closed \not\Rightarrow conv(C) \ closed \tag{2.31}$$

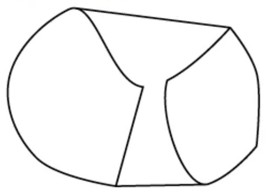


Figure 2.6: Any non convex set can be completed into a convex set.

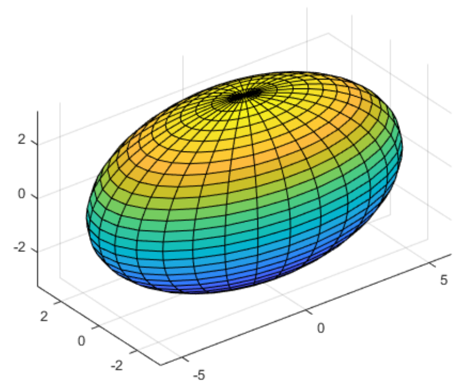


Figure 2.7: An example of an Ellipsoid.

Ultimately, explain under which condition the logical equation 2.31 is true. Actually, quite always this is the case.

Solution. One example of a closed but not convex set is the set of natural numbers. An example of convex but not closed set is $(0, 1)$. In order to prove 2.31, we need an example of the closed set whose convex hull is not closed. TO FINISH.

2.3.2 Examples of convex sets

The convex sets are not rare. There are plenty of them, and we use some operations that preserve convexity in order to create other convex sets. Let us see some of the obvious convex sets:

- Convex hull. Of course.
- Affine hyperplane: $\mathcal{H} = \{ax = b \ : \ x \in \mathbb{R}^n\}$. A hyperplane is a subspace whose dimension is one less than that of its ambient space. If a space is 3-dimensional then its hyperplanes are the 2-dimensional planes, while if the space is 2-dimensional, its hyperplanes are the 1-dimensional lines.
- Affine subspaces: $\mathcal{H} = \{ax \leq b \ : \ x \in \mathbb{R}^n\}$.
- Ball in p-norm, $p \geq 1$: $\mathcal{B}_p(x, r) = \{y \in \mathbb{R}^n \ : \ \|x - y\|_p \leq r\}$.
- Ellipsoid (figure 2.7): $\mathcal{E}(Q, x, r) = \{y \in \mathbb{R}^n \ : \ (y - x)^T Q(y - x) \leq r\}$, and Q positive semidefinite.
- All of the previous sets are closed, but also the open version is convex.
- Convex cones. There are different kinds of cones, for instances:

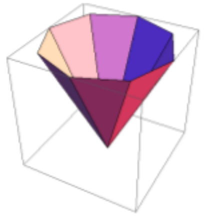


Figure 2.8: An example of a polyhedral cone.

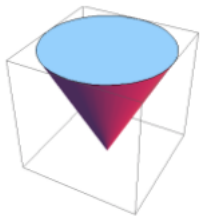


Figure 2.9: An example of a Lorentz cone.

– Conical hull (polyhedral cone, see figure 2.8):

$$cone(\{d_1, ..., d_k\}) = \{d = \sum_{i=1}^{k} \mu_i d_i \ : \ \mu_i \geq 0 \ \forall i\} \tag{2.32}$$

– Lorentz (ice-cream) cone (see figure 2.9):

$$\mathbb{L} = \left\{ x \in \mathbb{R}^n \ : \ x_n \geq \sqrt{\sum_{i=1}^{n-1} x_i^2} \right\} \tag{2.33}$$

– Cone of the positive semidefinite matrices (see figure 2.10):

$$\mathbb{S}_+ = \{A \in \mathbb{R}^{n \times n} \ : \ A \succeq 0\} \tag{2.34}$$

The last two cases of cones, are examples of non-polyhedral cones.

Exercise. Prove that the above sets are convex. Provide an example of a non convex cone.

2.3.3 Operations that preserve convexity

Let us now see some operations that preserve the convexity. These operations are frequently used to extend the set of convex sets.

- The intersection of convex sets $\{C_i\}_{i \in I} = \bigcap_{i \in I} C_i$ is a convex set.

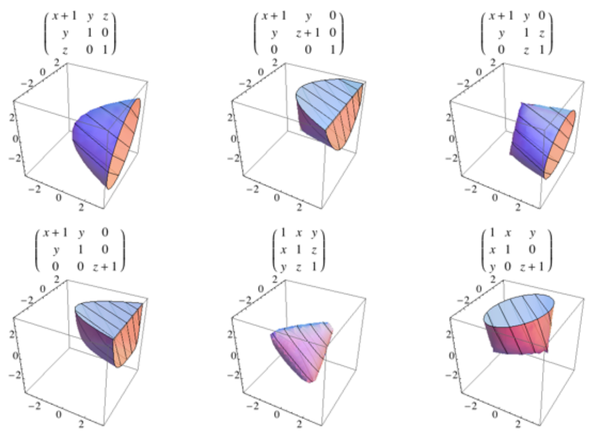


Figure 2.10: Some examples of positive semidefinite matrices that generate cones.

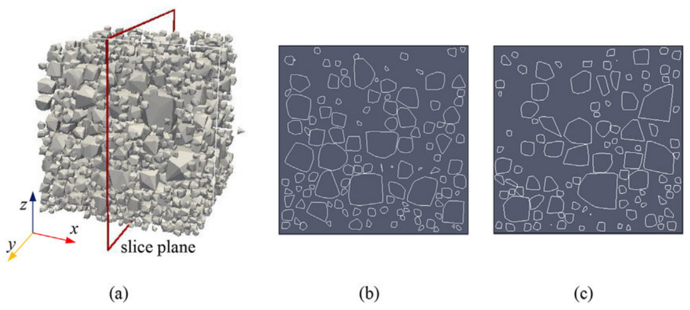


Figure 2.11: An example of a slice object.

- The Cartesian product of convex sets $\{C_i\}_{i\in I} = C_1 \times ... \times C_{|I|}$ is a convex set.
- Given a convex set C, then the image of its linear transformation (scaling, rotating, translating) $A(C) = \{x = Ay + b \;:\; y \in C\}$ is a convex set.
- Given a convex set C, then the inverse image of its linear transformation (scaling, rotating, translating) $A^{-1}(C) = \{x \;:\; Ax + b \in C\}$ is a convex set.
- Given two convex sets C_1 and C_2, then its linear combination $\alpha_1 C_1 + \alpha_2 C_2 = \{x = \alpha_1 x_1 + \alpha_2 x_2 \;:\; x_1 \in C_1, x_2 \in C_2\}$ with any $\alpha_1, \alpha \in \mathbb{R}$ is a convex set.
- Slice is a convex set: given that $C \subseteq \mathbb{R}^n = \mathbb{R}^{n_1} \times \mathbb{R}^{n_1}$, then we define slice $C(y) = \{x \in \mathbb{R}^{n_1} \;:\; (x, y) \in C\}$ (see figure 2.11).

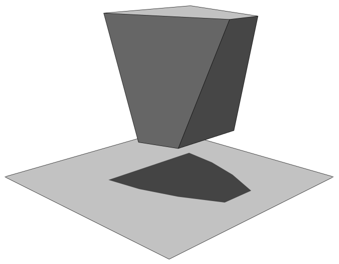


Figure 2.12: An example of a shadow object.

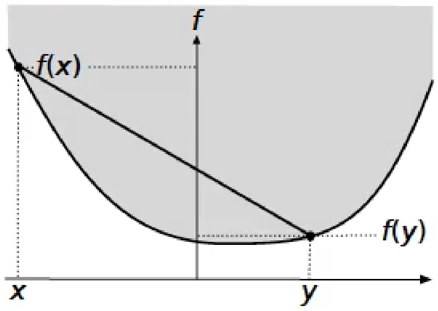


Figure 2.13: An example of a convex function.

- Shadow is a convex set: given that $C \subseteq \mathbb{R}^n = \mathbb{R}^{n_1} \times \mathbb{R}^{n_1}$, then we define shadow $C^1 = \{x \in \mathbb{R}^{n_1} \ : \ \exists y \ s.t. \ (x, y) \in C\}$ (see figure 2.12).

Exercise. Is the union of two convex set convex? Given a matrix $A \in \mathbb{R}^{m \times n}$, prove that the polyhedron of type $\mathcal{P} = \{x \in \mathbb{R}^n \ : \ Ax \leq b\}$ is convex.

Exercise. Given a polyhedron $\mathcal{P}$, prove that $\{d \ : \ x + \alpha d \in \mathcal{P}, \forall x \in \mathcal{P}, \alpha \geq 0\}$ (its recession cone) is a cone. (hint: look at $\mathcal{C} = \{x \in \mathbb{R}^n \ : \ Ax \leq 0\}$)

2.3.4 Convex functions

Let us define a convex function informally. A function is said to be convex if for any two points of its domain, the line connecting those two points is completely above the function itself (see figure 2.13).

Formally (see figure 2.14), for any two points $x, y \in \text{dom}(f) \subseteq \mathcal{R}^n, \alpha \in [0, 1]$ we have that:

$$\alpha f(x) + (1 - \alpha) f(y) \geq f(\alpha x + (1 - \alpha) y) \tag{2.35}$$

What we are saying is that given two points in the domain of the function, the line connecting them is the approximation model of the real function, given that we know just those two points. This line is at least the very same piece of the function f or it must be that it is above. This must happen for every two points in the domain of f in order to define f convex.

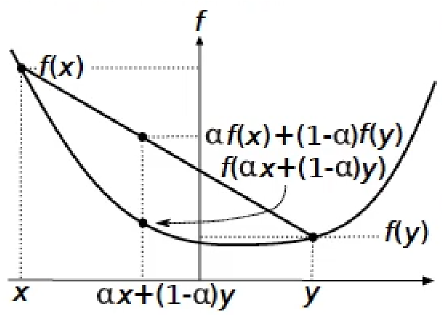


Figure 2.14: The definition of the convex function.

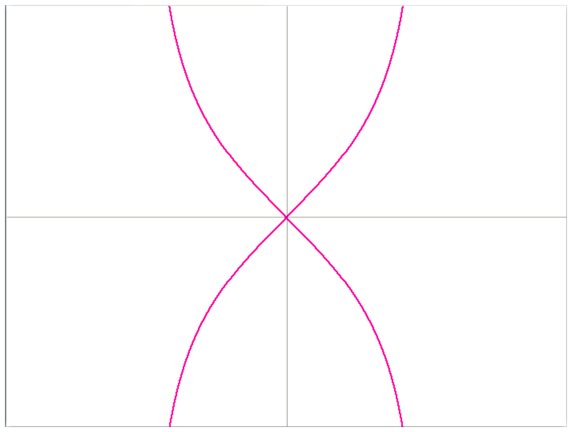


Figure 2.15: Bullet-nose curve. Also called sandglass by humans.

Let us now be a little bit more Einstein and generalise the above convex definition to more than two points.

$$\forall x_1, ..., x_k, \forall \alpha_1, ..., \alpha_k \in [0,1] \ s.t. \ \sum_{i=1}^{k} \alpha_i = 1 \ . \ f\Big(\sum_{i=1}^{k} \alpha_i x_i\Big) \leq \sum_{i=1}^{k} \alpha_i f(x_i) \tag{2.36}$$

Property. If a function f is convex, then any level set $S(f, v)$ is convex $\forall v \in \mathbb{R}$. TODO proof.

Nevertheless, it is not true the reverse. It is not true that if $\forall v \in \mathbb{R}$ level set $S(f, v)$ is convex then f is itself convex. Counterexample? Imagine Bullet-nose curve in some dimension. Yeah, you said: err.. imagine what?? See the figure 2.15. Clearly whatever level set has to be convex. But, this is not a convex function.

There is also a strict version of the convexity, where the signs $\leq, \geq$ are exchanged with their strict alternative, i.e. $<, >$.

One particularly important version of convexity is **strong convexity**. f is said to be strongly convex with modulus $\tau > 0$ if the following holds:

$$f(x) - \frac{\tau}{2}\|x\|^2 \tag{2.37}$$

or in more obvious terms:

$$\alpha f(x) + (1-\alpha)f(y) \geq f(\alpha x + (1-\alpha)y) + \frac{\tau}{2}\alpha(1-\alpha)\|x-y\|^2 \tag{2.38}$$

This basically means that the functions is bellow quadratically much. In other words, given a point x our function f in that point is steeper than the quadratic model in that point.

Convex functions cannot be unbounded bellow. The only case of a convex function that is unbounded below is the linear case, in any dimension.

Last but not least, f is called concave if $-f$ is convex. Clearly, we have also strictly and strongly concave functions.

2.3.5 Prototypical convex functions

As for the convex sets, we have some prototypical functions that are simple and we know that are convex. These, and many other, functions can be composed to get more complicated convex functions.

Let us illustrate some of the simplest convex functions:

- $f(x) = wx$, i.e. the linear object. This is the only case that is both convex and concave.
- $f(x) = \frac{1}{2}x^TQx + qx$ quadratic function, convex if and only if the Hessian matrix is positive semidefinite.
- $f(x) = e^{\alpha x}, \forall \alpha \in \mathbb{R}, x \in \mathbb{R}^n$ the exponential function.
- $f(x) = -\log(x), x > 0$ the complement logarithm function.
- $f(x) = x^\alpha, \forall \alpha \in \mathbb{R} \setminus (0,1), x \in \mathbb{R}^n$ the power function.
- $f(x) = \|x\|_p, p \geq 1$ the norm function.
- $f(x) = \max\{x_1, ..., x_n\}$ the maximum function.
- For any convex set C, its indicator function is convex:

$$I_C(x) = \begin{cases} 0, & \text{if } x \in C \\ +\infty, & \text{if } x \notin C \end{cases} \tag{2.39}$$

  The indicator function simply maps to 0 all the elements that belong to the set and to $+\infty$ those that don't. Since the underlying set is convex, it is obvious that the mathematical object that we obtain when we apply the indicator function transformation is indeed convex.

Exercise. Prove that the above functions are convex.
Exercise. Is the function $f(x) = \min\{x_1, ..., x_n\}$ convex?

2.3.6 Functional operations that preserve convexity

Let us now show some of the composition operators that can be used over convex functions in order to get other, more complex, convex functions. There are basically two ways of checking if a given function f is convex. Either we compute the Hessian and then verify that it is positive semidefinite everywhere on the function domain, or we try to get f as composition of some basic convex functions using the some operators that preserve convexity. Some of these operators are:

- Given f and g convex, $\alpha, \beta \in \mathbb{R}_+$, then the linear non negative combination $\alpha f + \beta g$ is also convex.
- Given a set of infinitely many convex functions $\{f_i\}_{i \in I}$, the function $f(x) = \sup_{i \in I} f_i(x)$ is convex.
- Given a convex function f, then the linear precomposition $f(Ax + b)$ is also convex.
- Given a convex multivariable function $f : \mathbb{R}^n \to \mathbb{R}$ and a convex increasing function $g : \mathbb{R} \to \mathbb{R}$, the composition $g \circ f = g(f(x))$ is a convex function.
- FINISH THE REMAINING

Exercise. Prove that the above composition operators preserve the convexity.

2.3.7 First order conditions over convex functions

The question is: why the heck we like the convex functions so much? Are they that pretty? Yes. They are like Sweden girls.

Given a function f that is convex, then its gradient exists almost everywhere. Let us prove the following property:

$$f \in C^1 \text{convex over convex set C} \iff f(y) \geq f(x) + (y - x)\nabla f(x), \forall x, y \in C \tag{2.40}$$

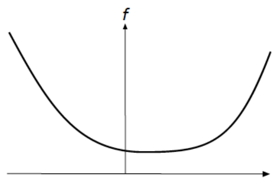
In other words, the linear model of our function f tangent in the point x is always under the function in whatever other point y (see figure 2.16).

What happens when we have the gradient 0? That is, what happens if we are in a stationary point? Well, the following happens (see figure 2.17):

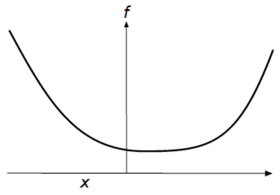
$$\nabla f(x) = 0 \Rightarrow f(y) \geq f(x), \forall y \in \mathbb{R}^n \tag{2.41}$$

What?! Yes, exactly! If we are in a stationary point x, pick any other y from the function domain, the value of f in y will be $\geq$ than the value in the point x. Wait, but that means that...? Yes! That means that x is the global minimum solution and that $f(x)$ is the global minimum value. That's!, why the heck we like the convex functions!

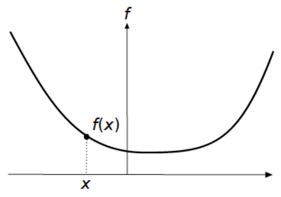
Theorem 2.3.1. *Given f convex and continuous, then if x is a stationary point of f then x is the global minimum.*



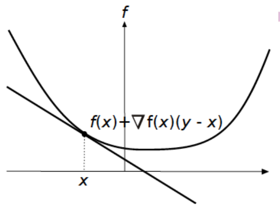
(a) Suppose you have a convex function



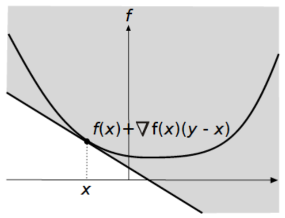
(b) Take a point x



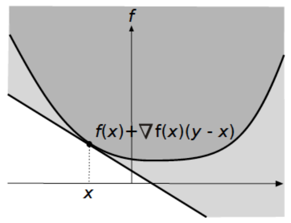
(c) Project on f and take its value



(d) Build the linear model in point x. Call it L_x

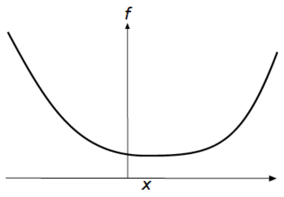


(e) The epigraph of L_x

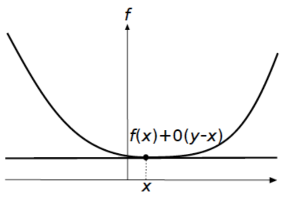


(f) The epigraph of f, subset of the epigraph of L_x

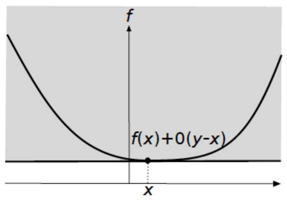
Figure 2.16: Visual explanation of the equation 2.40.



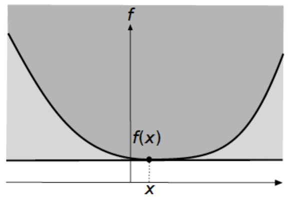
(a) Point x is a stationary point



(b) $\nabla f(x) = 0$



(c) Epigraph of L_x



(d) Epigraph of $f(x)$

Figure 2.17: Visual explanation of the equation 2.41.

2.3.8 Second order conditions over convex functions

We said that proving that the function is convex entirely on its domain requires computing its Hessian. In order to be able to do that we need the function to be continuously differentiable, that is it needs to have continuous first derivative. Usually, this is the simplest way to check convexity of a function.

Theorem 2.3.2. *A function $f \in C^2$, i.e. continuously differentiable, is convex over an open domain S if and only if its Hessian is positive semidefinite, in formulae:*

$$f \text{ is convex} \iff \nabla^2 f(x) \succeq 0 \tag{2.42}$$

TODO add exercises from the slide

2.3.9 Subgradients and subdifferentiales

Convexity is nice. But we spoke only about differentiable stuff, otherwise no gradient and no Hessian. Do we really need differentiability? Well, no. With convex functions we can even non differentiable things. But it get complicated.

As a side note, consider the function f and the set over which is defined to be convex. We could actually define the minimisation problem in the following way:

$$(P) \equiv \inf\{f_X(x) = f(x) + I_X(x) : x \in \mathbb{R}^n\} \tag{2.43}$$

Where $I_X(x)$ is the indicator function, which is non differential by design. f_X is called **essential objective**. Now look at this magic:

$$x_* \text{ is a global optimum} \iff x_* \text{ is local minimum for } f_X \tag{2.44}$$

Proof. By contradiction. Assume $\exists y \in \mathbb{R}^n : f(y) < f(x_*)$. Then:

$$f(x_*) \leq f(\alpha x_* + (1-\alpha)y) \leq \alpha f(x_*) + (1-\alpha) f(y) < f(x_*)$$

Which is a contradiction. □

Cool. But the question is still: how do I identify the local minimum without the gradient $\nabla f(x)$? We don't have the gradient. We need something that is similar to the gradient but not the gradient :). What was the main property of the gradient when we looked at first order conditions? The epigraph. Consider again something like in figure 2.18. The important thing was that the epigraph of the linear model was under the epigraph of the actual function it was approximating in the stationary point. In other point, it is not important that it is the gradient but that it is completely under the function.

So what is the subgradient s of f, the country cousin of the gradient ∇:

$$s \text{ is the subgradient of } f \text{ in } x \iff f(y) \geq f(x) + s(y-x), \forall y \in \mathbb{R}^n \tag{2.45}$$

Consider the function in figure 2.19a. As you can see, there are some kinks. One of the kinks is also our nice global minimum (figure 2.19b). In those kinks

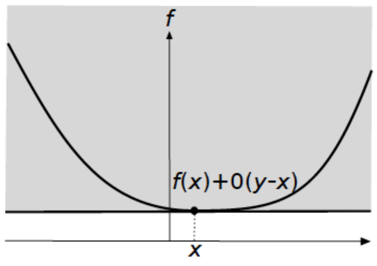
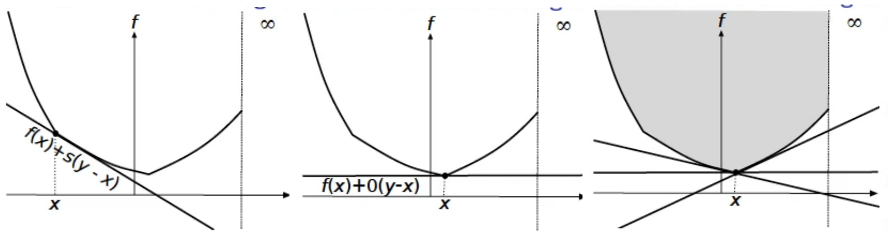


Figure 2.18: Caption



(a) Caption (b) Caption (c) Caption

there are infinitely many linear models that satisfy the equation 2.45 (see figure 2.19c). And that is fine. Because in that point we can have a (sub)gradient that is 0.

What is the problem? The problem is that in the point that is actually a stationary point, and so the gradient is 0, and so it is a local minimum, and so due to the convexity it is also a global minimum; there are infinitely many subgradients. A sub set of these tell you go on the right to find the minimum, another set tells you go on the left, and a singleton set tells you that you are at the right place. Solutions? Keep a set of subgradients.

Let us now define the **subdifferential**:

$$\partial f(x) = \{s \in \mathbb{R}^n : s \text{ is a subgradient at } x\} \tag{2.46}$$

Clearly, if f is differentiable in x then it must be the case that the subdifferential is a singleton set containing just the regular gradient.

You remember that the gradient characterise all the directional derivatives in a point x (scalar product). If you want to know if a direction is a descent direction, you just need to compute the scalar product with the directional derivative and the gradient. It turns out that with subgradient it is a bit more complicated. Taking just one gradient does not tells you the whole story. If you want to be sure that you are going in the descent direction, you need to have a subdifferential full of directions that in scalar product with the gradient give a negative value.

The steepest descent direction for such non differentiable functions in a point of a kink, the computation requires solving an optimisation problem:

$$s_* = -\text{argmin}\{\|s\| : s \in \partial f(x)\} \tag{2.47}$$

Note that the set over which this problem is operating, i.e. subdifferential, is closed and convex. Thus it is again a convex optimisation. Note also that if the solution is 0, it means that we are in the global minimum.

Let us now consider the following example. In figure 2.20a you can see the level sets of the multivariable function:

$$f(x, y) = \max\{x^2 + (y-1)^2, x^2 + (y+1)^2\} \tag{2.48}$$

This function is convex, non differentiable, and the minimum value is located in the point $x_* = [0, 0]$.

In the differentiable case the things are easy (figure 2.20b), we have our nice gradient that is normal (orthogonal) to the level sets and the opposite of it goes straight to the minimum (figure 2.20c), the steepest descent direction. Actually, whatever direction that is opposite to the gradient is a descent direction (figure 2.20d).

This does not occur in the non differentiable points. Suppose you are in the point shown on figure 2.20e. In that case, we have infinitely many subgradients. In particular, there are two directions (one is shown in figure 2.20f), that are not even pointing in the direction of the descent (they are pointing out of the level set). In other words, in those two directions the function is increasing. Nevertheless, the two direction form a closed and convex set that you can see in figure 2.20g. Each of the direction in that set is obtainable from the convex combination of these two extreme directions. The point is that each of these directions points inside the level set (figure 2.20h) and the one with the smallest norm is actually the steepest descent direction (figure 2.20i).

2.3.10 Subdifferential calculus

Like in the case of convex sets and convex functions composition operators that preserve convexity, we have similarly the composition operations over subdifferentials that preserve subdifferentiability.

Let us now illustrate some of these composition operators:

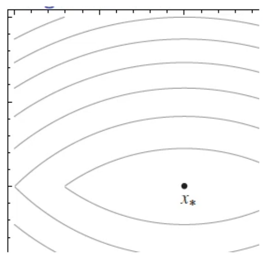
- TODO

2.3.11 Approximate subgradients

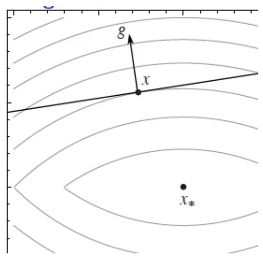
TODO

2.4 Descent Methods

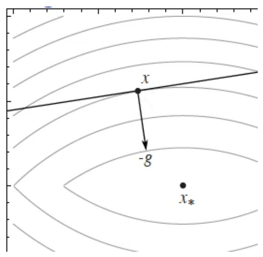
Let us now ask our self a very basic question: what is so special with the gradient? Can we use something different? The answer is yes. The only thing



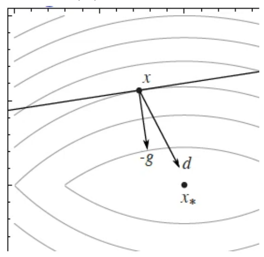
(a) Caption



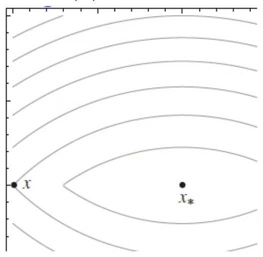
(b) Caption



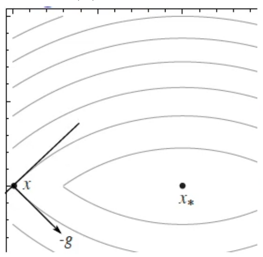
(c) Caption



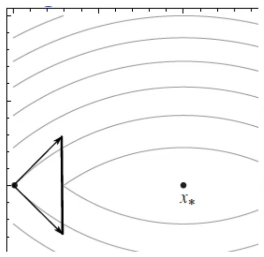
(d) Caption



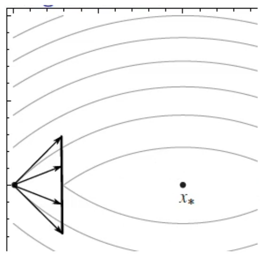
(e) Caption



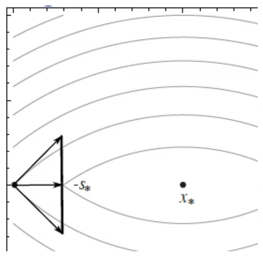
(f) Caption



(g) Caption



(h) Caption



(i) Caption

that we really care about is to go in a *descent* direction. The crucial arguments in our previous proofs about convergence was that the derivative was promising a significant descent along the direction $d = -\nabla f(x)$. But what if our direction is, let's say, rotated 45 degrees? We would still go in a descent direction. So we would expect the various proofs to carry on even in these cases, provided that the angle with the opposite of the gradient and the descent direction is not too big.

Let us now define mathematically a descent direction. A **descent direction** is a direction d such that:

$$d = \frac{\partial f}{\partial d^i}(x^i) \equiv d \cdot \nabla f(x) < 0 \equiv \cos(\theta^i) \geq 0 \tag{2.49}$$

This means that the direction d points roughly in the same direction of $-\nabla f(x^i)$.

There are infinitely many descent direction of course. The whole half space of descent directions. There is a nice result called *Zoutendijk condition* that states the following:

Theorem 2.4.1. *Suppose we have a bounded function whose gradient is Lipschitz continuous. Then under Armijo and strong Wolfe we have that:*

$$\sum_{i=1}^{\infty} \cos^2(\theta^i)\|\nabla f(x^i)\|^2 < \infty \tag{2.50}$$

Basically, this means that provided that the angle θ^i between $-\nabla f(x^i)$ and d^i is not too big, our sequence is converging to the minimum, i.e. $\nabla f(x)$ converges to 0. Think of the sequence that instead of moving straight to the minimum (when $d^i = -\nabla f(x^i)$), it circulate around in a spiral manner, until it reaches the minimum (centre). This happens provided that the angle is not too big. Actually Zoutendijk says that the sequence converges also in case θ^i converges to 90 degrees with the gradient ($\cos(\theta^i)$ converges to 0) provided that it does not converge too fast.

Note that if $\cos^2(\theta^i) = 1$ the formula above is the usual gradient method.

Note also that usually a non negative sequence has to converge to 0 quite fast in order for the summation to be convergent. Here, on the contrary, we require the sequence to not converge to 0 that fast, in order to have the overall summation convergent.

2.5 Newton Method

Let us now talk about one of the most important methods for unconstrained optimisation, the **Newton method**. You want a better direction? Use a better model. So instead of using just linear approximation, use the quadratic one.

Let us assume that the Hessian is positive definite:

$$\nabla^2 f(x) \succ 0 \tag{2.51}$$

So our second order model becomes:

$$f(x^i) + \nabla f(x^i)(x - x^i) + \frac{1}{2}\nabla^2 f(x^i)(x - x^i) \tag{2.52}$$

A side note: it is easy to say "let us use the Hessian". But suppose you have a function of thousands of variables. Since Hessian is a square matrix, its dimensions are order of millions. Those are huge matrices to deal with.

The Newton direction d^i is defined as:

$$d^i = -[\nabla^2 f(x^i)]^{-1}\nabla f(x^i) \tag{2.53}$$

Remember that in order to Newton to work, the quadratic model has to have a minimum. This basically means that the Hessian needs to be positive semi definite. But since in the direction computation we also need it invertible (non singular) we must require that the matrix is positive definite.

Consider the following nonlinear equation interpretation of Newton method. What we want to do is to solve the nonlinear set of equations of the form $\nabla f(x) = 0$. What we can do is to "linearise" this nonlinear system by applying Taylor to the gradient:

$$\nabla f(x) \approx \nabla f(x^i) + \nabla^2 f(x^i)(x - x^i) \tag{2.54}$$

Since we are looking for $\nabla f(x) = 0$ the model becomes:

$$(x - x^i) \approx -[\nabla^2 f(x^i]^{-1}\nabla f(x) \tag{2.55}$$

And what is $x - x^i$? It is the direction from the point x^i to the point x, in other words it is the direction d.

We have seen a proof that Newton method converges quadratically in a ball around local optimum. It is not surprising that the very same proof holds also for this more general case:

$$f \in C^3 \wedge x^* \text{ is a stationary point } \wedge \nabla^2 f(x^*) \succ 0 \Rightarrow \tag{2.56}$$

$$\exists \mathcal{B}(x^*, r) : x^0 \in \mathcal{B}(x^*, r) \Rightarrow \{x^i\} \to x^* \text{ quadratically} \tag{2.57}$$

In a more human language: if we have a function with continuous third derivative, a point x^* that is a stationary point on that function, and the Hessian is positive definite, then if we start in a point within a ball around the local optimum, then the converging sequence will converge at quadratic rate. Moreover, we also know that in that ball we can use the maximum step size of 1.

But what if we don't start in the ball around the local optimum? Line search. This is why we have studied techniques like backtracking. We start with the step size of 1 and if it does not work, then we backtrack and we redo with the line search.

Let us now interpret Newton method from another point of view. The gradient method is a good method if the Hessian is well conditioned, meaning that the largest and smallest eigenvalue are close together. Newton method is nothing else than the gradient method in a different space. What you actually do is to treat Hessian as space transformer. Wait, what?

Remember that the matrices map vectors to other vectors. So $y = Ax$ means that the x vector is mapped to y through A. Nice. What is our direction? $d = -[\nabla^2 f(x)]\nabla f(x)$ the gradient in x is mapped to $-d$ through the Hessian in x. Let us develop this mathematically.

Suppose we define the matrix $R = Q^{\frac{1}{2}}$. The matrix Q is our Hessian and we suppose that it is $Q \succeq 0$. This means that we can write our Q as $Q = RR$. Cool. Are we sure that R exists? Yeah man! Remember the eigenvalue decomposition:

$$Q = H\Lambda H^T \tag{2.58}$$

Since $Q \succeq 0$, no eigenvalue is negative, so we can take the square roots:

$$R = H\sqrt{\Lambda}H^T \tag{2.59}$$

We can now check what is RR:

$$RR = H\sqrt{\Lambda}H^T H\sqrt{\Lambda}H^T = H\sqrt{\Lambda}I\sqrt{\Lambda}H^T = H\Lambda H^T = Q \tag{2.60}$$

Since H is a unitary matrix ($HH^T = I$).

So let us consider a quadratic problem:

$$f(x) = \frac{1}{2}x^T Qx + qx \tag{2.61}$$

the gradient is:

$$\nabla f(x) = Qx + q \tag{2.62}$$

and our cool Newton direction is:

$$d = -[\nabla^2 f(x)]^{-1}\nabla f(x) = -[Q]^{-1}(Qx + q) = -Q^{-1}Qx - Q^{-1}q = -Q^{-1}q \tag{2.63}$$

What happens now if we use R to project all the vectors into another space?

$$y = Rx \tag{2.64}$$

We get a function in another variable, i.e. y. Let's see how it looks:

$$f(y) = \frac{1}{2}(R^{-1}y)^T Q(R^{-1}y) + q(R^{-1}y) = \tag{2.65}$$

$$= \frac{1}{2}y^T(R^{-1})^T QR^{-1}y + qR^{-1}y = \tag{2.66}$$

$$= \frac{1}{2}y^T Iy + qR^{-1}y \tag{2.67}$$

whose gradient is:

$$\nabla f(y) = Iy + qR^{-1} \tag{2.68}$$

and whose Hessian is I. The largest and smallest eigenvalue are the same. Basically, the gradient method here along a direction d finishes in one iteration.

Well, for the quadratic functions :). But it also works pretty well in case of non quadratic functions.

What is the direction? Let's see:

$$d = -[I]^{-1}\nabla f(y) = -\nabla f(y) = -qR^{-1} \tag{2.69}$$

What is this direction in the original space? Exactly the Newton direction.

Geometrically, given a function in the original space, the level sets may be quite elongated. Thus, the gradient method performs quite poor. Then we apply an intelligent trick: we use the Hessian to change the space where the level set of the functions are quite round. In this way, the gradient method performs extremely well. And that's what we do.

So in the end, Newton method means space transformation plus gradient method.

2.5.1 Apropos convergence

What about the convergence of this method? We have the global convergence provided that the angle is not to close to 0. There are some technicalities that need to be true, such as that the gradient needs to be Lipschitz, so that the Hessian is strictly positive definite and bounded:

$$uI \preceq \nabla^2 f \preceq LI \tag{2.70}$$

In order to be convergent, we need to prove that the angle does not go to 0. What follows from the Hessian being positive definite is that:

$$\nabla^2 f(x^i)d^i = -\nabla f(x^i) \tag{2.71}$$
$$\Rightarrow \tag{2.72}$$
$$\text{(multiply both sides with } -(d^i)^T) \tag{2.73}$$
$$-(d^i)^T\nabla^2 f(x^i)d^i = (d^i)^T\nabla f(x^i) \tag{2.74}$$

Now from Poloni world, we know that:

$$\lambda^n\|d\|^2 \leq d^T M d \leq \lambda^1\|d\|^2 \tag{2.75}$$

So we have that the previous equality is:

$$-(d^i)^T\nabla^2 f(x^i)d^i = (d^i)^T\nabla f(x^i) \leq -\lambda^n\|d^i\|^2 \tag{2.76}$$

Let us not apply norm operator to both sides of the equation 2.71. We get the following:

$$\|\nabla^2 f(x^i)d^i\| = \|\nabla f(x^i)\| \tag{2.77}$$

By Cauchy Schwarz inequality we have:

$$\|\nabla f(x^i)\| = \|\nabla^2 f(x^i)d^i\| \leq \|\nabla^2 f(x^i)\|\|d^i\| \leq \lambda^1\|d^i\| \tag{2.78}$$

From the scalar product between Newton direction and the gradient we get the following:

$$\cos(\theta^i) = \frac{d^i \cdot \nabla f(x^i)}{\|d^i\|\|\nabla f(x^i)\|} = \frac{-\nabla f(x^i)^T[\nabla^2 f(x^i)]^{-1}\nabla f(x^i)}{\|d^i\|\|\nabla f(x^i)\|} \leq \tag{2.79}$$

$$\leq \frac{-\lambda^n\|d\|^2}{\|d^i\|\|\nabla f(x^i)\|} = -\frac{1}{\|\nabla f(x^i)\|}\lambda^n\|d^i\| \leq -\frac{\lambda^n\|d^i\|}{\lambda^1\|d^i\|} = \tag{2.80}$$

$$= -\frac{\lambda^n}{\lambda^1} \leq -\frac{u}{L} \tag{2.81}$$

Thus, since both eigenvalues and the pair (u, L) are all positive numbers, we have proven that the angle stays always below $\frac{u}{L}$, and so away from 0. Thus, we have the global convergence.

So now we know that our Newton method converges but we don't know how fast. We will not prove it but the Newton method has a superlinear convergence except from when we enter into the ball around the local optimum where the convergence gets quadratic.

Now, all of this seems to work only for convex functions. Except it does not :). It also work with non convex functions. Why, how? Look at the previous mathematics, we have used Hessian everywhere but what really matters is that the matrix is positive definite. So as long as you can provide a matrix H that satisfies:

$$uI \preceq H \preceq LI \tag{2.82}$$

we can proceed with the very same mathematics that we have employed with the Hessian. So the descent method does not depend on the fact that we have used the Hessian but only that we have used a positive definite matrix with the bounded eigenvalues.

But given a non convex function, its Hessian will obviously not be positive definite, not either positive semi definite. It will probably be indefinite. So what matrix can we use? Well let's hack its whatever definite Hessian into a positive definite. How? Take the Hessian of the function, take its minimum eigenvalue λ^n (which will be ≤ 0. otherwise the matrix would be positive definite), take an $\epsilon > -\lambda^n$. Consequently, take $H^i = \nabla^2 f(x^i) + \epsilon^i I$ as the positive definite matrix to use in Newton method. This hack actually has a name and it is called **Hessian modification**. But beware! The larger is ϵ the more H^i will be different from the true Hessian and thus you are getting far from real things. But the more ϵ is near to $-\lambda^n$ the more you get numerical issues of machine precision stuff. Moreover, there are some algorithmic issues.

Usually what we do is to set $\epsilon = \max\{0, \delta - \lambda^n\}$ for some δ. In this way when, hopefully, we are close to the local optimum, the Hessian will be positive definite. In such case, $\lambda^n > 0$ thus the quantity $\delta - \lambda^n > 0$, and so we do not change the Hessian. This will allow the Newton method to run quadratically in the ball around the optimum solution. If on the other hand the minimum eigenvalue is negative, then $\delta + \lambda^n > 0$ and so we will perturb slightly the original Hessian. The problem still remains: how much is δ?

Note that what we are trying to do here is to solve a constrained problem that is quite frequent in linear algebra and optimisation. The problem is:

$$\min\{\|H - \nabla^2 f(x^i)\| \ : H \succeq \delta I\} \tag{2.83}$$

that is: find the closest matrix to the Hessian that is at least positive definite as the matrix δI.

The problem with Newton method is clear. At each point I need to have a Hessian, or a modification of the Hessian, which means that I have to compute it! Moreover, and probably even worse news, we need to invert it. This takes $\mathcal{O}(n^3)$. Newton is okay for problems with hundred or even thousands of variables. But take a problem with millions of variables and you will kill Newton instantly. Suggestions for better stuff? Continue to read...

2.6 Quasi-Newton methods

We have already said that if we are able to provide the positive definite matrix instead of the Hessian we can get the superlinear convergence and quadratic in the proximity of the optimal value. Let us then forget about the Hessian. So let us compute a matrix H that is near the Hessian and we want to compute it without ever touching the Hessian itself. We want to compute an approximation of the Hessian using only the gradients. The Hessian is the derivative of the gradient. So one approximation of the Hessian could be that once we have a gradient in a point, we move a little bit, take the gradient in that point, make the difference, divide by the step and that is an approximation of the Hessian. You can always approximate the derivative if you know the function. In this case the function is the derivative and we are approximating its derivative, i.e. the Hessian. But we want to do something more clever.

Suppose we have the following model:

$$m^i(x) = \nabla f(x^i)(x - x^i) + \frac{1}{2}(x - x^i)^T H^i (x - x^i) \tag{2.84}$$

$$x^{i+1} = x^i + \alpha^i d^i \tag{2.85}$$

Now, suppose we have computed $\nabla f(x^{x+1})$. We want to update the model in the following manner:

$$m^{i+1}(x) = \nabla f(x^{i+1})(x - x^{i+1}) + \frac{1}{2}(x - x^{i+1})^T H^{i+1}(x - x^{i+1}) \tag{2.86}$$

$$x^{i+2} = x^{i+1} + \alpha^{i+1} d^{i+1} \tag{2.87}$$

How can we choose H^{i+1}? Let us reason about what we properties we want our matrix to have:

- We want the matrix H^{i+1} to be positive definite because we need it to solve the system.

- We want that in the new model the gradient in the previous point is exactly the true gradient of the function, i.e.:

$$\nabla m^{i+1}(x^i) = \nabla f(x^i) \tag{2.88}$$

  This is also called Secant equation since it can be also written as:

$$H^{i+1}(x^{i+1} - x^i) = \nabla f(x^{i+1}) - \nabla f(x^i) \tag{2.89}$$

- We want that the two consecutive Hessians are not that different, i.e. $\|H^{i+1} - H^i\|$ is "small".

How if we want to develop something like this, it is useful to work in the space of movements. Let us define some quantities:

$$s^i = x^{i+1} - x^i = \alpha^i d^i \tag{2.90}$$
$$y^i = \nabla f(x^{i+1}) - \nabla f(x^i) \tag{2.91}$$

From these two, our secant equation becomes:

$$(S)\ H^{i+1}s^i = y^i \tag{2.92}$$

Let us now multiply both sides by $(s^i)^T$, and employ the first property that we want H^{i+1} to have:

$$(C)\ (s^i)^T H^{i+1}s^i = (s^i)^T y^i = \|s^i\|^2 > 0 \tag{2.93}$$

Note that once that we do a step, s^i and y^i are fixed and they are not something that we can choose. But in order to have solvable the secant equation, you need to assure that the scalar product from equation 2.93 is positive. This is because our H is positive definite, and it maps positive (resp. negative) vectors to positive (resp. negative) vectors. The equation 2.93 is called **curvature condition**.

If we have chosen the direction d and the step size satisfies the strong Wolfe condition, then the curvature condition holds. Let us prove it:

Proof. We want to prove that $(W') \Rightarrow (C)$. Strong Wolfe means that:

$$\phi'(\alpha^i) = \nabla f(x^{i+1})d^i \geq m_3\phi'(0) = m_3\nabla f(x^i)d^i \tag{2.94}$$
$$\Rightarrow \text{ subtract on both sides } \nabla f(x^i)d^i \tag{2.95}$$
$$\nabla f(x^{i+1})d^i - \nabla f(x^i)d^i \geq m_3\nabla f(x^i)d^i - \nabla f(x^i)d^i \tag{2.96}$$
$$\iff \tag{2.97}$$
$$(\nabla f(x^{i+1}) - \nabla f(x^i))d^i \geq (m_3 - 1)\nabla f(x^i)d^i = (m_3 - 1)\phi'(0) > 0 \tag{2.98}$$
$$\iff \tag{2.99}$$
$$y^i d^i \geq (m_3 - 1)\phi'(0) > 0 \tag{2.100}$$

□

So everything holds if you do the step reasonably. And in this case reasonably means that it must respect the strong Wolfe condition.

There are many matrices that satisfy (S). Infact it is a linear system with n^2 unknowns. Clearly due to the third condition, we want to solve the following minimisation constrained problem:

$$\min\{\|H - H^i\| : Hs^i = y^i, H \succeq 0\} \tag{2.101}$$

This problem highly depends on the type of the employed norm. Different norms bring different solutions. But if we chose the norm in a good way, reasonable norms, it turns out that there is a closed formula that comes in our help. One of these formulas is **DFP formula** due to Davidon-Fletcher-Powell. By setting $\rho^i = \frac{1}{y^i s^i}$ which is > 0, we have the following formula:

$$H^{i+1} = (I - \rho^i y^i (s^i)^T) H^i (I - \rho^i s^i (y^i)^T) + \rho^i y^i (y^i)^T \tag{2.102}$$

This is a clever way to describe what the approximation of the Hessian should be. But there is still one thing. We will still have to invert the matrix H^{i+1} at each step, which is something we don't want to do because we pay $\mathcal{O}(n^3)$. And here comes the real elegant fact. What we really want is to work with the inverses. So assume we have just computed the inverse of H^i, i.e. $(H^i)^{-1}$. What we want to get is $(H^{i+1})^{-1}$. Now, if we look better at our DFP formula from above, we can see that it is a so called *rank-two correction.* A *rank-n correction* of a matrix M is the addition to M of a matrix N or rank n. In our case y and s form a matrix of rank 1 and in the formula above we add twice a rank 1 matrix, so we get a rank 2 correction.

Rank two correction of H^i can use the formula called **Sherman-Morrison-Woodbury** formula to get the inverse of rank 1 correction:

$$[A + ab^T]^{-1} = A^{-1} - \frac{A^{-1}ab^T A^{-1}}{1 - b^T A^{-1} a} \tag{2.103}$$

Let us call $B^{i+1} = (H^{i+1})^{-1}$. Then by injecting SMW formula in DFP from above, we get:

$$B^{i+1} = B^i + \rho^i s^i (s^i)^T - \frac{B^i y^i (y^i)^T B^i}{(y^i)^T B^i y^i} \tag{2.104}$$

It is the inverse of H^i plus rank one correction, plus another rank one correction. Computing B^{i+1} is $\mathcal{O}(n^2)$ and solving the Newton system is also $\mathcal{O}(n^2)$ because the matrix B^{i+1} is already the inverse. You just multiply this matrix with the gradient.

Another formula instead of DFP is called BFGS due to Broyden-Fletcher-Goldfarb-Shanno. Turns out that if you use BFGS the matrices B that you get along the various iterations are better matrices than those obtained with DFP formula.

Write $H^{i+1}s^i = y^i$ in terms of B^{i+1}:

$$s^i = B^{i+1} y^i \tag{2.105}$$

Substitute H with B and y with s and invert everything:

$$H^{i+1} = H^i + \rho^i y^i (y^i)^T - \frac{H^i s^i (s^i)^T H^i}{(s^i)^T H^i s^i} \tag{2.106}$$

$$B^{i+1} = (I - \rho^i s^i (y^i)^T) B^i (I - \rho^i y^i (s^i)^T) + \rho^i s^i (s^i)^T = \tag{2.107}$$

$$= B^i + \rho^i [(1 + \rho^i (y^i)^T B^i y^i) s^i (s^i)^T - (B^i y^i (s^i)^T + s^i (y^i)^T B^i)] \tag{2.108}$$

Actually, people are crazy so they have been making this funny stuff of combining the two approaches. This gives rise to the so called **Broyden family**:

$$\beta H^{i+1}_{\text{DFP}} + (1-\beta) H^{i+1}_{\text{BFGS}} :\succeq \text{ if } \beta \in [0,1] \wedge (S) \text{ is satisfied} \tag{2.109}$$

There is one last thing to say. How do we choose the first matrix? Remember that we don't want to compute the Hessian. So we could do the finite differences method.

Let us now recap. We started with Newton. It is a good method but computing the Hessian is not cheap. Then we moved to Quasi-Newton methods whose aim is to approximate at best the Hessian. We got $\mathcal{O}(n^2)$ both in time and space. This is a good achievement. But still, if we deal with problems that have millions of variables, even Quasi-Newton methods won't work. Note that there are plenty of problems with so many variables, especially in ML. So can we do better? Yes.

2.7 Limited memory BFGS

Limited-memory BFGS (L-BFGS or LM-BFGS) is an optimisation algorithm in the family of quasi-Newton methods that approximates the Broyden Fletcher Goldfarb Shanno algorithm (BFGS) using a limited amount of computer memory. It is quite popular algorithm for parameter estimation in machine learning. The algorithm's target problem is to minimise $f(x)$ over unconstrained values of the real-vector x where f is a differentiable scalar function.

Like the original BFGS, L-BFGS uses an estimate of the inverse Hessian matrix to steer its search through variable space, but where BFGS stores a dense $n \times n$ approximation to the inverse Hessian (n being the number of variables in the problem), L-BFGS stores only a few vectors that represent the approximation implicitly. Due to its resulting linear memory requirement, the L-BFGS method is particularly well suited for optimisation problems with many variables. Instead of the inverse Hessian H_k, L-BFGS maintains a history of the past m updates of the position x and gradient $\nabla f(x)$, where generally the history size m can be small (often $m < 10$). These updates are used to implicitly do operations requiring the Hessian vector product.

Memory and time worsen proportionally to the increase of k. But the converges improves with the increase of k. So the more k is small, the more L-BFGS performs similar to the gradient method. On the contrary, the more it approaches to n the more it resembles the Quasi-Newton method. So a trade off is necessary here.

2.8 Conjugate Gradient Method

Conjugate gradient method is yet another algorithm for dealing with very large scale problems. Conjugate gradient is actually a Krylov method.

Let us start with quadratic functions. Remember what the gradient does. At each iteration it take a step of size α^{i+1} in the direction d^{i+1} that is perfectly orthogonal to the previous direction d^i. This is because it has already performed the optimal step in that direction. In other words, it optimises in one of n dimensions of the space at each iteration. The problem is that, each time you go orthogonal to the previous direction, you loose all that you have done up until that previous direction. Mathematically speaking $d^{i+1} \not\perp d^{i-1}$. So people asked them self, could we construct a chain of direction such that:

$$d^1 \perp d^2 \perp ... \perp d^i = 0 \tag{2.110}$$

Actually what we want is to have direction that are conjugate between them when passed through Q, $(d^i)^T Q d^{i-1} = 0$.

To achieve this, we cannot use the old nice gradient as the next direction. We need to modify it slightly so that we can get what we want. Turns out that if we modify the gradient in the following manner, we get what we want:

$$d^0 = 0 \tag{2.111}$$

$$d^i = \nabla f(x^i) + \beta^i d^{i-1} \tag{2.112}$$

We say that the gradient is **deflected** using d^{i-1}. The "only" complicated term in the formula is β^i (a scalar value). Fortunately, for the quadratic functions there is a closed formula:

$$\beta^i = \frac{(\nabla f(x^i))^T Q d^{i-1}}{(d^{i-1})^T Q d^{i-1}} = \frac{\|\nabla f(x^i)\|^2}{\|\nabla f(x^{i-1})\|^2} \tag{2.113}$$

Also for the step size there is a closed formula which is:

$$\alpha^i = -\frac{(\nabla f(x^i))^T d^i}{(d^i) Q d^i} = \frac{\|\nabla f(x^i)\|^2}{(d^i)^T Q d^i} \tag{2.114}$$

In figure 2.21 you can find the algorithm for Conjugate Gradient method. The algorithm starts with the first direction just the opposite of the gradient. In all the other iterations the algorithm computes the deflected gradient and the next point, exploiting the closed formula of both α and β. End of story.

A powerful property of this algorithm is that it ends in at most n iterations. This is because at each iteration we are minimising over a subset of $k \leq n$ dimensions. Actually, the algorithm can even finish in strictly less than n iterations. This has to do with how are the eigenvalues of Q clustered. If the matrix has t eigenvectors that have equal or almost equal eigenvalues, those t vectors will be killed in just one iteration. In other words, we will minimise along all t directions in just one step. So if we have matrices of 10 thousands variables, but only 10 different eigenvalues, then the algorithm will find the optimum in just

```
procedure $x = CGQ(Q, q, x, \varepsilon)$ {
  $d^- \leftarrow 0$;
  while( $\|\nabla f(x)\| > \varepsilon$ ) do {
    if( $d^- = 0$ ) then $d \leftarrow -\nabla f(x)$;
      else { $\beta = (\nabla f(x)^T Q d^-) / ((d^-)^T Q d^-)$; $d \leftarrow -\nabla f(x) + \beta d^-$; }
    $\alpha \leftarrow (\nabla f(x)^T d)/(d^T Q d)$; $x \leftarrow x + \alpha d$; $d^- \leftarrow d$;
  }
}
```

Figure 2.21: Conjugate Gradient method.

```
procedure $x = CGQ(Q, q, x, \varepsilon)$ {
  $d^- \leftarrow 0$; $\nabla f^- = 0$;
  while( $\|\nabla f(x)\| > \varepsilon$ ) do {
    if( $d^- = 0$ ) then $d \leftarrow -\nabla f(x)$;
      else { $\beta = \| \nabla f(x^i) \|^2 / \| \nabla f^- \|^2$; $d \leftarrow -\nabla f(x) + \beta d^-$; }
    $\alpha \leftarrow$ AWLS( $f(x + \alpha d)$ ); $x \leftarrow x + \alpha d$; $d^- \leftarrow d$; $\nabla f^- \leftarrow \nabla f(x)$;
  }
}
```

Figure 2.22: Conjugate Gradient method for non linear objective function.

10 iterations. This is very powerful property because what we could do is to do some preconditioning on the initial matrix Q. For instance multiply it with some matrix that cluster the eigenvalues together.

The interesting idea is that all this idea can be used also in case of non linear objective function (see figure 2.22). The only difference is that we don't have a closed formula for α. No, it is not an error. Look closely at β and you will see that the second expression of β does not have anything to do with Q. Unfortunately we can't say the same for α. So we need the saint ways of the line search. This non linear version of Conjugate Gradient with this particular β closed formula is called **Fletcher Reeves method**.

There are more ways to compute β. And of course each one has its own positive and negative sides. In particular there are some formulas that are identical to the one that we have seen in quadratic case, while very different in the non linear case.

2.8.1 Convergence and efficiency

The convergence of conjugate gradient method is not trivial and it highly depends on β. Turns our that for some formulas, it may even not be sufficient to assure Armijo-Wolfe in order to have a descent direction.

Sometimes it is also suggested to "restart" the direction. Meaning that at a certain point, do not employ the deflected gradient, but just take a plain gradient and deflected gradient from the next iteration.

The convergence of conjugate gradient is rather good. It has an n *step quadratic* convergence. Meaning that each n steps the convergence is approximately quadratic. If you think about it, conjugate gradient method minimises optimally a quadratic function in n steps.

2.9 Heavy Ball Gradient

It all starts with the following point update:

$$x^{i+1} = x^i - \alpha \nabla f(x^i) + \beta^i (x^i - x^{i-1}) \tag{2.115}$$

This method is called Heavy Ball because it is as if x_i was a heavy ball and the gradient is something like a force that wants to steer it away. But since the ball is heavy, it cannot change immediately the direction to 90 degree but it takes time. This is why β is called momentum.

This is not a descent algorithm. For this reason no one is guaranteeing that each iteration the function is monotonically decreasing. For this reason, the convergence analysis is quite complicated and we won't go into the details.

In case of strongly convex function, i.e. $(\lambda^n = u)I \preceq \nabla^2 f(x) \preceq (\lambda^1 = L)I$, actually α and β can be estimated:

$$\alpha = \frac{4}{(\sqrt{\lambda^1} + \sqrt{\lambda^n})^2} \tag{2.116}$$

$$\beta = \max\{|1 - \sqrt{\alpha\lambda^n}|, |1 - \sqrt{\alpha\lambda^1}|\}^2 \tag{2.117}$$

With this choices that we do not prove because it takes a lot of algebra and math, we get the following result:

$$\|x^{i+1} - x^*\| \leq (\frac{\sqrt{\lambda^1} - \sqrt{\lambda^1}}{\sqrt{\lambda^1} + \sqrt{\lambda^1}})\|x^i - x^*\| \tag{2.118}$$

Note that in case of the gradient, the two differences in the norm are in the output space, while here we are getting the difference in the input space. This is a stronger result. This is not a surprising result because strongly convex means just one minimum and the level sets are compact, the best possible situation.

The other important difference with the gradient is that we have the square roots. It may seem not a big deal but, square root of the biggest eigenvalue lowers the value of the difference a lot. This means that the expression is nearer to 1, which translates into better convergence. This is the reason why many use Heavy Ball gradient instead of the simple plain gradient.

2.10 Accelerated Gradient

There is a version of Heavy Ball gradient that is actually better and it is called **accelerated gradient** (see figure **??**). It is only for convex functions. This algorithm is obtained from a sophisticated theoretical analysis so will just go through it very quickly.

2.11 $< \nabla$ methods

Up until now we have assumed to have the gradient. We can use the gradient method, if we want something faster we can use more than gradient methods. But what if we don't have gradient? What if the function is non differentiable or it is just too difficult to compute the gradient?

2.11.1 Machine Learning applications

Let us first motivate why we need less than gradient methods. One of the direct application is Machine Learning. In machine learning one has a set of inputs $X = [X^i \subset \mathcal{X}]_{i \in I}$ (it is a matrix) for a given $I = \{1, ..., m\}$ and a corresponding set of outputs $Y = [y_i]_{i \in I}$ (a vector). Then one usually wants to devise a mapping function $\Phi : \mathcal{X} \to \mathcal{F}$ from the space of inputs to the space of features, such that:

$$\min\{\sum_{i \in I} L(y^i, \Phi(X^i) \cdot w) : w \in \mathbb{R}^n\} \tag{2.119}$$

where $L(\cdot, \cdot)$ is a **loss function**. In other words, one wants to minimise the loss function, that is a function that computes the distance of the observed outputs from the true outputs.

If the loss function is simply the two norm, it becomes the least squares problem:

$$D \tag{2.120}$$

Note that in ML one has m observations and n weights and usually $m \gg n$. So in case of least squares, we are summing a lot of stuff. So what is that people usually do? Usually the inputs are drawn randomly from some distribution and hopefully they are independently distributed. This means that a small subset of the whole set of observation could be as well as representative. So this is what actually people do. Take a small subset, compute the gradient for that small subset and treat it as it was the gradient of the whole function. So two important questions arise now. How do I choose the subset, and how many do I need? This method takes name of **stochastic gradient descent** or **incremental gradient**.

Typically, this is not end of story. Convergence may be very hard to achieve. It is for this reason that one usually **regularise** the objective function:

$$\min\{\sum_{i \in I} L(y^i, \Phi(X^i) \cdot w) + \mu\Omega(w) : w \in \mathbb{R}^n\} \tag{2.121}$$

μ is a hyper parameter to decide empirically, $\Omega(w)$ is the regulariser, which is usually $\Omega(w) = \|w\|_1$ called **lasso**. Actually, people would like to have 0 norm instead of 1 norm. This is because 0 norm would count how many features are used in the objective function. The problem is that 0 norm is not convex thus it would make the overall problem not convex. So the best convex approximation of the 0 norm is the 1 norm. Unfortunately, 1 norm is not differentiable. This is why of this topic now :).

2.11.2 Subgradient methods

Remember what a subgradients are:

$$g \in \partial f(x) \equiv f(y) \geq f(x) + g(y - x), \forall y \in \mathbb{R}^n \tag{2.122}$$

Basically all the functions tangent to the kinks whose epigraph contains entirely the epigraph of the objective function. Each opposite of the subgradient points towards the half space where the optimal solution is located. So going along the direction of one of the opposite of the subgradient should bring us closer to the optimal solution, provided that we do the right step. So the question is: what is the right step?

2.12 $> \nabla$ methods

In this section we are going to examine the so called "more than Gradient methods". The name comes from the fact that these approaches does not necessarily require to go *strictly* in the opposite direction of the gradient. As we will see, as long as we are going in a descent direction, we are going well.

Let us recap briefly what does it mean descent direction and why we selected the opposite of the gradient as the king of such directions. A direction is a descent direction if:

$$\langle \nabla f(x), d \rangle < 0 \tag{2.123}$$

but what does it mean actually? Remember what is the definition of the scalar product:

$$\langle \nabla f(x), d \rangle = \|\nabla f(x)\| \|d\| \cos(\theta) \tag{2.124}$$

where θ is the angle between d and $\nabla f(x)$. From the moment that both $\|d\|$ and $\|\nabla f(x)\|$ are positive (a norm is always ≤ 0), the only factor that influences the sign of the scalar product is the cosine $\cos(\theta)$. Accordingly, if we set gradient to be the direction of the x asix, as long as the direction d is pointing in the same direction of the x asix, the cosine will be positive. The opposite for the negative value. Thus, the maximum negative value is when the angle θ is π, i.e. $\cos(\pi) = -1$, i.i.e. $d = -\nabla f(x)$.

Nevertheless, as long as $\cos(\theta) < 0$ we are going down. It is just the matter of a factor $\in [-1, 0)$. Provided that the angle is not too far from π (or equivalently too small), we can expect that the methods that we are going to present converge. And indeed they do. There is a general result regarding these kind of methods, we announce it without proving it:

Theorem 2.12.1. *Suppose we have a non unbounded function $f \in C^1$ with ∇f Lipschitz. Then:*

$$(A) \cap (W') \Rightarrow \sum_{i=1}^{\infty} \cos^2(\theta_i) \|\nabla f(x_i)\|^2 < \infty$$

This result basically says that, as long as our angle θ_i is bounded away from 0, we will converge. Note that: if an infinite sequence converges then $\{a_i\} \to 0$, but not conversely, i.e. a converging sequence to 0 does not imply a converging infinite sum. In our case, actually, it is also possible to converge even if $\cos(\theta)$ is not bounded away from 0, i.e. $cos(\theta) \approx 0$, provided that this does not happen to fast in the series.

2.12.1 Newton's Method

Instead of using just the first order model for f, we the second order, i.e. quadratic model. Assume for now that the Hessian is positive definite:

$$\nabla^2 f(x_i) \succ 0$$

then exists the minimum of the second order model:

$$f(x_i) + \nabla f(x_i)(x - x_i) + \frac{1}{2}(x - x_i)^T \nabla^2 f(x_i)(x - x_i)$$

We can then compute the Newton's direction by:

$$\nabla f(x) \approx \nabla f(x_i)(x - x_i) + \frac{1}{2}(x - x_i)^T \nabla^2 f(x_i)(x - x_i) \tag{2.125}$$

$$\nabla f(x) = 0 \tag{2.126}$$

$$\nabla f(x_i)(x - x_i) + \frac{1}{2}(x - x_i)^T \nabla^2 f(x_i)(x - x_i) = 0 \tag{2.127}$$

$$x = x_i - [\nabla^2 f(x_i)]^{-1} \nabla f(x_i) \tag{2.128}$$

Thus the direction is given by $-[\nabla^2 f(x_i)]^{-1} \nabla f(x_i)$ with $\alpha_i = 1$.

Thus the Newton's method is just take a unitary step size along the direction $-[\nabla^2 f(x_i)]^{-1} \nabla f(x_i)$.

We have seen that Newton converges superlinearly and quadratically in the tail of the convergence:

$$\begin{aligned} &f \in C^3 \wedge x_\star \text{ saddle point } \wedge \nabla^2 f(x_\star) \succ 0 \Rightarrow \\ &\exists \mathcal{B}(x_\star, r) : (x_1 \in \mathcal{B}(x_\star, r) \Rightarrow \{x_i\} \to x_\star \text{ quadratically}) \end{aligned}$$

This means that if the algorithm starts in a ball around the saddle point, it will converge quadratically to the optimum point. The issue is that we cannot know whether or not we are starting in the proximity of the optimal point. Not a big deal: we could try to run Netwon's method and in case we do not get the quadratic convergence we restart by using the standard Armijo-Wolfe line search.

The function must have a positive definite Hessian since we need to invert it to get the direction. The direction obtained is surely a descent direction as we can verify it by the following:

$$d_i \nabla f(x_i) = -\nabla f(x_i)^T [\nabla^2 f(x_i)]^{-1} \nabla f(x_i) < 0$$

where the last inequality follows from the fact that the Hessian is a symmetric positive definite matrix.

We can interpret Newton's method from another point of view (**Space Dilatation**). Suppose Hessian Q is symmetric and positive definite. Then we can find a matrix R such that $Q = R^T R$ and $R = \sqrt{Q}$. We are sure that such an R exists and that it is symmetric. Thus:

$$Q = H\Lambda H^T \Rightarrow R = H\sqrt{\Lambda}H^T$$

infact:

$$R^T R = (H\sqrt{\Lambda}H^T)^T H\sqrt{\Lambda}H^T = H\sqrt{\Lambda}H^T H\sqrt{\Lambda}H^T = H\Lambda H^T$$

Suppose now:

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

the gradient of this function is:

$$\nabla f(x) = Qx + q$$

and the direction is:

$$d = -Q^{-1} q$$

Since we have $Q = R^T R$, we can rewrite the function as:

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

We then set $y = Rx$ and proceed like:

$$f(y) = \frac{1}{2}y^T I y + q^T R^{-1} y$$

with the gradient:

$$\nabla f(y) = y + R^{-1} q$$

and the direction:

$$d = -R^{-1} q$$

As we can see, the Hessian of this new function is surely symmetric positive definite (identity matrix). Moreover, $\lambda_n(I) = \lambda_1(I)$ which means that the Newton's method *finishes in one iteration.*

Let us now speak about Newton's convergence. We suppose our function is convex with gradient Lipschitz continuous. This implies the following:

$$uI \leq \nabla^2 f \leq LI$$

We can now prove that it is convergent provided that $\cos(\theta)$ does not go to 0 too fast, otherwise we will not go in the descent direction.

Proof. We want to prove the following:

$$\cos(\theta_i) \leq 0$$

Since the dot product between $\nabla f(x_i)$ and d_i must be negative. Thus we need to show that:

$$\frac{\langle \nabla f(x_i), d_i \rangle}{\|\nabla f(x_i)\| \|d_i\|} \leq 0$$

Let us start with the numerator:

$$d_i = -[\nabla^2 f(x_i)]^{-1} \nabla f(x_i) \Rightarrow [\nabla^2 f(x_i)] d_i = -\nabla f(x_i)$$

we now multiply both sides by $-d_i^T$:

$$-d_i^T [\nabla^2 f(x_i)] d_i = d_i^T \nabla f(x_i)$$

Since the Hessian is symmetric we can use the Spectra decomposition:

$$\lambda_n \|d_i\|^2 \leq d_i^T [\nabla^2 f(x_i)] d_i \leq \lambda_1 \|d_i\|^2$$

But be careful, we have a minus up there, thus:

$$-\lambda_n \|d_i\|^2 \geq -d_i^T [\nabla^2 f(x_i)] d_i \geq -\lambda_1 \|d_i\|^2$$

So we then obtain:

$$d_i^T \nabla f(x_i) = -d_i^T [\nabla^2 f(x_i)] d_i \leq -\lambda_n \|d_i\|^2$$

As for the denominator, we get:

$$\|\nabla f(x_i)\| = \|\nabla^2 f(x_i) d_i\|$$

from the definition of the Newton's direction. Now we use the Cauchy-Schwartz inequality:

$$\|\nabla f(x_i)\| = \|\nabla^2 f(x_i) d_i\| \leq \|\nabla^2 f(x_i)\| \|d_i\| = \lambda_1 \|d_i\|$$

where the last equality follows from the fact that the norm of the Hessian, by spectral decomposition, is the norm of the diagonal matrix Λ, i.e. λ_1.

We can now give an upper bound for $\cos(\theta_i)$:

$$\cos(\theta_i) = \frac{\langle \nabla f(x_i), d_i \rangle}{\|\nabla f(x_i)\| \|d_i\|} = \frac{\nabla d_i^T f(x_i)}{\|\nabla f(x_i)\| \|d_i\|} \leq \frac{-\lambda_n \|d_i\|^2}{\lambda_1 \|d_i\| \|d_i\|} = -\frac{\lambda_n}{\lambda_1} \leq -\frac{u}{L}$$

Since u and L are strictly positive, we have that the value of cosine is strictly negative. $\square$

Newton works very well on convex and strictly convex function. But does it also work with non convex ones? Yes.

When we spoke about the convergence above, we did not use the fact that the function is convex really. What we used is the fact that Hessian is positive definite. Thus if our function does not have a positive definite matrix we can perturb it slightly in order to get a close positive definite one. This method is called **Hessian modification**. We choose the smallest ϵ_i such that:

$$H^i = \nabla^2 f(x_i) + \epsilon_i I \succ 0$$

Nevertheless, we have a minor numerical issue here. Every $\epsilon_i > -\lambda_n$ will work fine (remember: since $\nabla^2 f \succ 0$ we know that the smallest eigenvalue is strictly positive). But if λ_n is of machine precision order, we could risk to set $\epsilon_i = 0$. We suppose here that this does not occur.

Accordingly, we take $\epsilon_i = \max\{0, \delta - \lambda_n\}$ and we solve the following constraint optimisation problem:

$$\min_{H_i}\{\|H_i - \nabla^2 f(x_i)\|_F : H \geq \delta I\}$$

How can we solve such a problem. Using spectral decomposition of the Hessian:

$$\nabla^2 f(x_i) = H \Lambda H^T$$

We then take:

$$H_i = H \bar{\Lambda} H^T$$

where $\bar{\Lambda}$ has on diagonal $\bar{\gamma}_i = \max\{\lambda_i, \delta\}$.

The problem with this approach is that at each iteration we need to invert and factorise. This costs $\mathcal{O}(n^3)$, cubic complexity. That's the reason why we move now to the Quasi-Newton methods.

2.12.2 Quasi-Newton Methods

Quasi-Newton methods can be used if the Hessian is unavailable or is too expensive to compute at every iteration. We do not need actually the exact Hessian in the quadratic model of the function, we just need a positive definite matrix. The model m of our function f then becomes:

$$m_i(x) \approx f(x_i) + \nabla f(x_i)(x - x_i) + \frac{1}{2}(x - x_i)^T H_i (x - x_i)$$

After computing $\nabla f(x_{i+1})$ where $x_{i+1} = x_i + \alpha_i d_i$, we update our model m of f by:

$$m_{i+1}(x) \approx f(x_{i+1}) + \nabla f(x_{i+1})(x - x_{i+1}) + \frac{1}{2}(x - x_{i+1})^T H_i (x - x_{i+1})$$

The problem is: how can we compute H^{i+1} efficiently. Let us reason about what we want that H^{i+1} satisfy:

1. $H^{i+1} \succ 0$
2. $\nabla m^{i+1}(x_i) = \nabla f(x_i)$, i.e. the gradient in the previous point x_i of the new model must be the same of the previous model in the same point. This is called **Secant equation**
3. $\|H^{i+1} - H^i\|$ is small

Note how the second point can be also written as:

$$\begin{aligned}\nabla m^{i+1}(x_{i+1} - x_i) &= f(x_{i+1}) + \nabla f(x_{i+1})(x_{i+1} - x_i - x_{i+1}) + \\ &+ \frac{1}{2}(x_{i+1} - x_i - x_{i+1})^T H_i (x_{i+1} - x_i - x_{i+1}) = \\ &= f(x_{i+1}) - \nabla f(x_{i+1})(x_i) + \frac{1}{2}(-x_i)^T H_i(-x_i) TOFINISH\end{aligned}$$

Let us now see how can we compute such a matrix H^{i+1}. We define the following:

$$s_i = x_{i+1} - x_i$$

and

$$y_i = f(x_{i+1}) - f(x_i)$$

Thus we can write:

$$H^{i+1} s_i = y_i$$

We then multiply both sides by s_i^T:

2.12.3 Conjugate Gradient Method

2.12.4 Deflected Gradient Methods

Chapter 3

Constrained Optimisation

Now is the time to talk real problems. Constrained optimisation problems are the problems that have constraints. Whenever a problem is restricted to a particular part of the space, we are talking about the constraint optimisation. Remember that the unconstrained and constrained optimisation problems are quite connected. Each constrained problem can be transformed into unconstrained by simply bringing the constraints into the objective function. This can be done with the indicator function: whenever a point does not belong to the admissible region the value of the function goes to ∞. No one is that crazy to do it though. The indicator function is a bad function, it brings non differentiability into a problem that may be perfectly differentiable everywhere (both the objective function and the constraints).

In general this is a complicated problem unless everything is convex. Most of the times it is but sometimes it is not. Generally the global optimum is very difficult to find. We usually stick to a weaker condition: **the local optimum** x_*:

$$\min\{f(x) : x \in \mathcal{B}(x_*, \epsilon) \cap X\} \tag{3.1}$$

for some $\epsilon > 0$. Note that we are talking about local optimum, not local minimum. This is not casual. Local minimum is the point where you truly have a minimum of the function. This happens when you are working on the entire space $\mathbb{R}^n$. If you are constrained to a subspace X then you are talking about *minimum in that subspace*. Hence we are talking about the local optimum constrained to that subspace. Clearly, with this definition, if the local optimum is located in the interior of the set X, i.e. if $x_* \in intX$ then the local optimum is also the local minimum. This is because in the interior of the set the constraints are not touching the optimal solution. So the optimum is independent from the constraints. So in order to have local minimum different from local optimum, it must be located on the boundary of the set, i.e. ∂X. So the boundary of the feasible set is quite important.

3.1 A simple case: feasible sets with no interior

Let us start with the simplest constrained problem: quadratic problem. The problems of this kind are those that have quadratic objective function but linear constraints. Moreover, let us consider a problem with just the linear equality constraints:

$$\min\{f(x) : Ax = b\} \tag{3.2}$$

where $A \in \mathbb{R}^{m\times n}$ with $\text{rank}(A) = m < n$ and rows linearly independent. We need to have more variables than equations because if $m = n$, then it is a square system and thus we could just apply the closed formula and solve it without any fancy optimisation technique. Linear independence is needed because otherwise if there is a row that is linearly dependent from the others then either the system is impossible or that dependant row can be eliminated (linear algebra). Think of a plane that is parallel to another one. In that case, the two planes will never intersect thus there is no solutions.

Actually we could transform this particular constrained problem into the unconstrained one. Since A is short and fat, we can split it in two parts: A_B and A_N, respectively $m \times m$ matrix and $m \times (n - m)$ matrix. We do the same for the input vector x, i.e. x_B of m entries and x_N of $n-m$ entries. So we have:

$$A_B x_B + A_N x_N = b \tag{3.3}$$

Now if A_B is invertible, that is if $\det(A_B) \neq 0$ then we can write the above equation as:

$$x_B = A_B^{-1}(b - A_N x_N) \tag{3.4}$$

This means that we have transformed the variables of x_B into dependant variables from independent variables x_N. In other words, now we con concentrate on optimising just x_N, while x_B can be subsequently obtained from x_N. How? Like this:

$$x_B = -A_B^{-1}A_N x_N + A_B^{-1}b = Dw + d \tag{3.5}$$

where:

$$D = \begin{bmatrix} -A_B^{-1}A_N \\ I \end{bmatrix}, d = \begin{bmatrix} A_B^{-1}b \\ 0 \end{bmatrix}$$

and $w \in \mathbb{R}^{n-m}$. The problem then becomes:

$$w_* = \min_{w\in\mathbb{R}^{n-m}} \{r(w) = f(Dw + d)\} \tag{3.6}$$

which is clearly an unconstrained optimisation problem in w. m linear constraints kill m degrees of freedom.

We can thus construct a *reduced gradient* of the form:

$$\nabla r(w) = D^T \nabla f(Dw + d) \tag{3.7}$$

which is obtained by applying the usual derivative of the composition operator (chain rule). How do we solve an unconstrained optimisation problem. The good old gradient: we need to solve:

$$\nabla r(w_*) = D^T \nabla f(Dw + d) = 0 \tag{3.8}$$

Let us now explain something that is quite important. We know how to characterise the optimality in the space of w. But what happens when we want to write the optimality in the original space of x? Note that:

$$D^T A^T = AD = 0 \tag{3.9}$$

Having said that, we can also say that whatever scalar of the matrix A will also produce 0 when multiplied with D (obviously), that is:

$$\mu AD = \mu D^T A^T = 0 \tag{3.10}$$

Let us now call μA with z:

$$D^T z = 0 \tag{3.11}$$

Now, look at the gradient in 3.8. We want to find a point $x = Dw + d$ such that the gradient of that point x multiplied with D^T will give us 0. But that would mean:

$$\mu A = \nabla f(x), x = Dw + d \tag{3.12}$$

So the gradient must be a scalar multiple of A. So if we manage to find x such that the gradient of f in that point is a multiple of A, then in the space of w that point is a stationary point, and x is a local optimum.

Let us now state it in more formal terms:

$$\begin{aligned}&\text{if}\\&Ax = b \text{ and } \exists\mu : \mu A = \nabla f(x)\\&\text{then}\\&x \text{ is a stationary point for } (P) \equiv \min\{f(x) : Ax = b\}\end{aligned} \tag{3.13}$$

This is a special case of **Poorman's KKT conditions**, which are THE optimality conditions for constraint optimisation problems. Basically, the optimal point must be a feasible point and the gradient in that point must be written as a linear combination of the gradients of the constraints. So in order to show that x is optimal, we need to compute μ.

Of course if f is convex, these conditions are also sufficient for the optimality.

3.1.1 Geometric interpretation

What are these μ geometrically speaking? Suppose you have a quadratic function whose level sets are shown in figure 3.1. In the same figure you can also see a line that correspond to the feasible region described by the equality constraints $Ax = b$. For the sake of simplicity, suppose we have just one equality, as in figure. Remember that we are in $\mathbb{R}^n$ space, not $\mathbb{R}^{n+1}$ where the value of the function lives.

The minimum is clearly located in the point in the middle of the inner most level set. But since we must restrict the admissible points to the admissible region indicated by $Ax = b$, the local optimum is the point that is the intersection

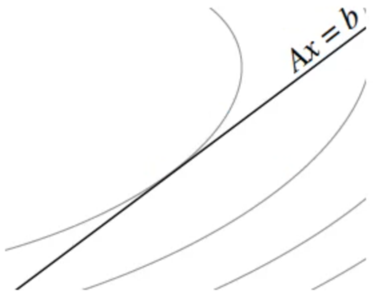


Figure 3.1: Caption

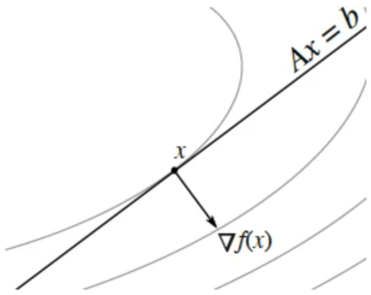


Figure 3.2: Caption

between the line and the inner most level set. Why inner most? Because every other outer level set has a greater function value.

Now, we know that the gradient is always normal (orthogonal) to the level sets. Consequently, it is also normal to the admissible region in the point of the intersection, i.e. $\nabla f(x) \perp \partial X$ (see figure 3.2). But since by definition $A \perp \partial X$ (all the points in X have the scalar product 0 with A), it must be that the gradient is also co-linear with A, i.e. $A \parallel \nabla f(x)$ (see figure 3.3).

The matrix A sends every vector somewhere in the space, you remember, matrices are used for transformations :). The vectors that when multiplied with A produce b are in the admissible region. But who said A likes only the admissible region? A sends vectors wherever: and so it can send also in the opposite direction. Exactly where our gradient is located.

It is for this reason that we define the directions in the feasible region:

$$F = \{d \in \mathbb{R}^n : A \cdot x = 0\} \tag{3.14}$$

In constrained optimisation it is no more necessary to have the gradient zero in the optimal point. Think of taking a minimum of a concave quadratic function whose admissible region is from a certain level set above. Clearly the minimum is at the edge of the admissible region. In that point the gradient is for sure negative.

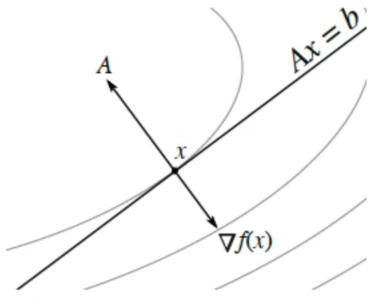


Figure 3.3: Caption

What we have in the local optimum point is that all the directions in the feasible region have the scalar product with the gradient ≥ 0:

$$\nabla f(x) \cdot d \geq 0, \forall d \in F \tag{3.15}$$

Unfortunately constraints are way more complicated than lines and affine spaces. So we need to dive into it, right now.

3.2 The theory of ice creams: cones

The crucial mathematical object that we are going to use to define the optimality is the **Tangent Cone**:

$$T_X(x) = \left\{ d \in \mathbb{R}^n : \exists \{z_i \in X\} \to x \wedge \{t_i \geq 0\} \to 0 : d = \lim_{i \to \infty} \frac{z_i - x}{t_i} \right\} \tag{3.16}$$

Err... what? Yes, it is not nice as a definition. But it is easier than you think. Basically, this definition states that in order for d to be a direction along which we can move from the current point x, it must be a direction that comes from a sequence of points belonging to the feasible set and that converge to x. But also there must be a sequence of corresponding step sizes that converge to 0. The overall sequence may be a straight line, may be of sinusoidal form, it may even be on the border. What matters is that it tends to x in the limit 3.4.

Why is this cone important? Because it provides a necessary condition for x to be local optimum (and global optimum in case X is convex).

Theorem 3.2.1.

$$x \text{ local optimum } \Rightarrow \nabla f(x) \cdot d \geq 0, \forall d \in T_X(x)$$

Proof. Let us assume by contradiction that there exists a direction in the local optimum x that has negative scalar product with the gradient of x. Let us call that direction d. Since $d \in T_X(x)$, this means that there exists a sequence

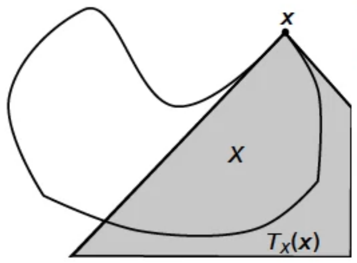


Figure 3.4: An example of the tangent cone

of points $\{z_i\}$ in X that converge to x with a sequence of step sizes $\{t_i\}$ that converge to 0. Consider now the first order Taylor expansion in z_i:

$$\begin{aligned} f(z_i) &= f(x) + \nabla f(x)(z_i - x) + R(z_i - x) = \\ &= f(x) + \nabla f(x) \cdot (z_i - x) + R(z_i - x) \end{aligned} \tag{3.17}$$

Let us now add $-f(x)$ to both sides, divide them by t_i and send them to limit:

$$\begin{aligned} \lim_{i\to\infty} \frac{f(z_i) - f(x)}{t_i} &= \nabla f(x) \cdot \lim_{i\to\infty} \frac{z_i - x}{t_i} + \lim_{i\to\infty} \frac{R(z_i - x)}{t_i} = \\ &= \nabla f(x) \cdot d + \lim_{i\to\infty} \frac{R(z_i - x)}{t_i} < 0 \end{aligned} \tag{3.18}$$

Which is a contradiction. The scalar product is negative by assumption and the limit on the right is converging to 0. So the whole right part is strictly negative. This would mean that z_i is a point in the neighbour of x that has the value of the function strictly smaller than the one in x. But we said that x was a local optimum. □

Something that you could say: yes but if x is local optimum, nobody is telling me that it is also a global optimum. Thus, for sufficiently large ball around x, I could find a point where the function value is actually smaller than the one in x. True. But we are talking about cones, not balls. There must not be a point in the cone of x that has a smaller value.

Tangent cone does not give us also the sufficient condition. In other words, there may be a point y where all the directions in the tangent cone of y give positive scalar product with the gradient $\nabla f(y)$, but still y may not be a local optimum.

Consider the example in figure 3.5. This function is clearly not convex. The grey part is the admissible region X. Consider the point x in the origin as our current point. The set of directions that are part of the tangent cone in x are all those directions that belong to the positive part of the y axe (see figure 3.6). All those directions give ≥ 0 scalar product with the gradient in x. Clearly, the point x is not a local optimum. x is just a stationary point.

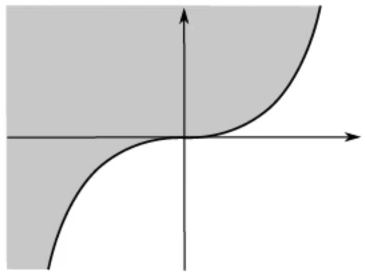


Figure 3.5: A counterexample for the sufficient condition of the tangent cone.

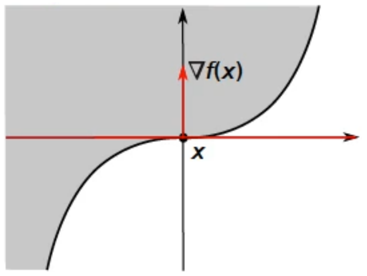


Figure 3.6: $T_X(x)$ is the x axes. All directions are in 0 scalar product with the gradient $\nabla f(x)$.

What we really want to have is the set of feasible directions.

$$F_X(x) = \{d \in \mathbb{R}^n : \exists\bar{\epsilon} > 0 \text{ s.t. } x + \epsilon d \in X, \forall \epsilon \in [0, \bar{\epsilon}]\} \tag{3.19}$$

If X is convex we can omit the last part of the definition and just use $\bar{\epsilon}$ instead of ϵ.

Clearly, this cone looks very similar to the tangent cone. Generally, the feasible directions cone is not a closed set (T_X is). The closure of F_X is always a subset of T_X and if everything is convex then the closure of F_X is the very same T_X. In this case we could actually conclude that:

$$x \text{ is global optimum } \iff \nabla f(x) \cdot d \geq 0, \forall d \in T_X(x) \tag{3.20}$$

How do we characterise T_X? It depends on how you characterise the admissible region X. Usually, one can do that in many different ways. A classical way is by explicitly describing the constraints:

$$X = \{x \in \mathbb{R}^n : g_i(x) \leq 0, i \in \mathcal{I}, h_j(x) = 0, j \in \mathcal{J}\} = \{x \in \mathbb{R}^n : G(x) \leq 0, H(x) = 0\} \tag{3.21}$$

where $\mathcal{I}$ is the set of inequalities and $\mathcal{J}$ is the set of equalities. Note that G and H are vector valued functions. G sends vectors from space $\mathbb{R}^n$ to vectors in space $\mathbb{R}^{|\mathcal{I}|}$ and H sends vectors from space $\mathbb{R}^n$ to vectors in space $\mathcal{R}^{|\mathcal{J}|}$.

There is a nice *theoretical* way to represent all the constraints with just one inequality constraint. To do that, we need to passages:

- Each equality constraint can be expressed as two inequality constraints:
$$a = 0 \iff a \le 0 \wedge -a \le 0$$
- The set of inequality constraints follows the relation:
$$G(x) \le 0 \iff \max\{g_i(x) : i \in \mathcal{I}\} \le 0$$

One concept that we will use a lot is the set of **active constraints**. A constraint is active if the point is on the border described by that constraint. By definition all the equality constraint are always active, otherwise the point would not be in the feasible region. The active inequality constraints are basically all those constraints that result in $= 0$. In formulae:

$$\mathcal{A}(x) = \{i \in \mathcal{I} : g_i(x) = 0\} \tag{3.22}$$

It is thus convenient to define G restricted to that particular set of indices whose corresponding inequalities are active:

$$G_{\mathcal{A}(x)} : \mathbb{R}^n \to \mathbb{R}^{|\mathcal{A}(x)|} \tag{3.23}$$

So now we can define a cone that can be used in practical way. The cone is called **first order feasible direction cone**.

$$D_X(x) = \{d \in \mathbb{R}^n : \nabla g_i(x) \cdot d \le 0 \; i \in A(x), \nabla h_j(x) \cdot d = 0 \; j \in \mathcal{J}\} \tag{3.24}$$

Note that we are now looking into the gradients not functions. It is for this reason that we can rewrite the above definition with:

$$D_X(x) = \{d \in \mathbb{R}^n : JG_{A(x)}(x) \cdot d \le 0, JH(x) \cdot d = 0\} \tag{3.25}$$

where J stands for Jacobian matrix.

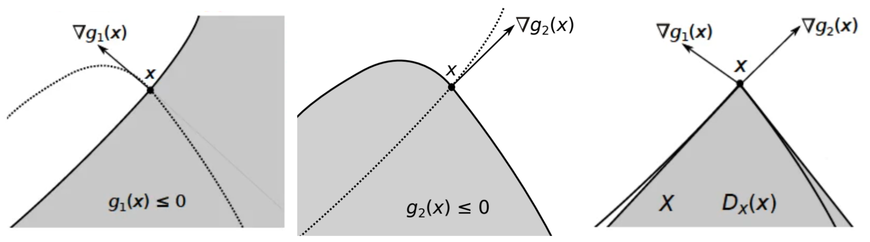
Each active inequality i defines one particular curve that passes in the point x. The gradient in x with respect to that curve will be normal to the curve it self (see figure 3.7 for an example).

It is intuitive that the tangent cone is contained in the first order feasible directions cone, i.e.:

$$T_X(x) \subseteq D_X(x) \tag{3.26}$$

So what we really want is the set of feasible directions F_X, but due to the closure issue we moved to the tangent cone directions T_X, and finally due to its complexity of characterisation we have constructed the first order feasible directions cone D_X. It is clear that all of these cones are very similar to each other. We would like to have the tangent cone same as the first order feasible directions cone. Unfortunately, in some pathological cases this may not be true. In other words, we could have $T_X \subset D_X$ (see figure 3.8). Note that here everything is convex. So the convexity are not helping us to make the two cones the same stuff.

We need to to introduce something that will guarantee us that we are not in some of these pathological cases. This something is called *constraint qualifications.* There are plenty of them, but let us consider the three most important for our purposes:



(a) Caption (b) Caption (c) Caption

Figure 3.7: Assume we have only two active inequalities in the point x. Take the two corresponding gradients and then construct the cone in the opposite direction.

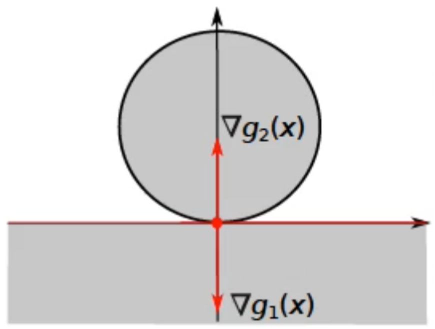


Figure 3.8: Suppose you have to convex constraints, one is the all negative subspace of the y axes and the other one is the ball you see in the figure. The first order feasible directions cone is the whole x axes while the tangent cone consist just of one point, the origin. This is because the feasibility region is just the origin (the intersection of the two constraints). $D_X(x) = \{[x_1, x_2] : x_2 = 0\}, T_X(x) = F_X(x) = [0, 0]$.

- Affine constraints: everything is linear, except the objective function. It this constraint holds then $\forall x \in X \; T_X(x) = D_X(x)$.
- Slater's conditions: inequalities are convex and equalities are affine. If there exists at least one point $x \in X$ that is in the interior, $g_i(x) < 0, \forall i \in \mathcal{I}$, then $T_X(x) = D_X(x), \forall x \in X$. Informally, Slater's condition states that the feasible region must have an interior point.
- Linear independence: given a point $x \in X$, if the set of:

  $$\{\nabla g_i(x) : i \in \mathcal{I}\} \cup \{h_j(x) : j \in \mathcal{J}\}$$

  is linearly independent, then $T_X(x) = D_X(x)$.

But excluding these pathological cases, or by using some of the constraint qualifications, we have our necessary condition (also sufficient in convex case):

$$\nabla f(x) \cdot d \geq 0, \forall x \in D_X(x) \tag{3.27}$$

The problem is: there are infinitely many directions to check.

3.2.1 Farkas' lemma

Let us now see how we can avoid checking all the infinite direction of the first order feasible directions cone D_X but still be able to say whether we are in the local optimum or not. We know that D_X is a polyhedral cone of type: $\{d \in \mathbb{R}^n : Ad \leq 0\}$. Now consider the figure 3.9. Consider a generic vector $c \in \mathcal{R}^n$. This vector can either belong to the dual cone or not. In case it belongs then we can write c as a linear combination of the gradients (see figure 3.10) . Otherwise there must exist a direction d in the original cone that has a positive scalar product with c (see figure 3.11).

Let us formally define Farkas' lemma:

Theorem 3.2.2.

$$\forall c \in \mathbb{R}^n . \exists \lambda \geq 0 : c = \lambda A \vee \exists d : Ad \leq 0 \wedge c \cdot d > 0$$

Note that we use strictly greater than 0 and not ≥ 0. This is because if we were using $\geq$ then our vector c could have been A_2 for instance.

Why do we care about Farkas' lemma? Get ready!

We want to use Farkas' lemma to get a nice and (more than everything) useful necessary condition. So we want something like:

$$\begin{aligned} &x^* \text{ is local optimum } \Rightarrow \nabla f(x^*) \cdot d \geq 0, \forall d \text{ that satisfy:} \\ &\nabla g_i(x^*) \cdot d \leq 0, \forall i \in \mathcal{A}(x^*) \wedge \nabla h_i(x^*) \cdot d = 0, \forall j \in \mathcal{J} \end{aligned} \tag{3.28}$$

Turns out that this problem is equivalent to:

$$\begin{aligned} &x^* \text{ is local optimum } \Rightarrow \exists \lambda \in \mathbb{R}_+^{|\mathcal{A}(x^*)|} \wedge \mu \in \mathbb{R}^{|\mathcal{J}|} \text{ s.t.} \\ &f(x^*) + \sum_{i \in \mathcal{A}(x^*)} \lambda_i \nabla g_i(x^*) + \sum_{j \in \mathcal{J}} \mu_i \nabla h_j(x^*) = 0 \end{aligned} \tag{3.29}$$

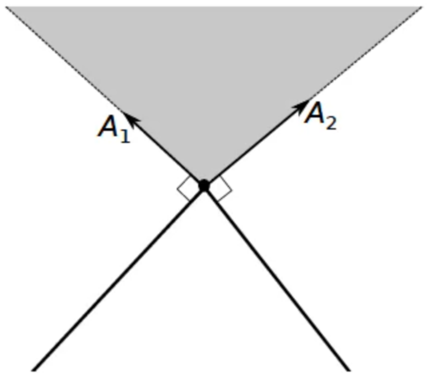


Figure 3.9: The dual cone $C^* = \{c = \sum_{i=1}^{k} \lambda_i A_i : \lambda_i \geq 0\}$ is the one defined by the two gradients and all the positive linear combinations of the two.

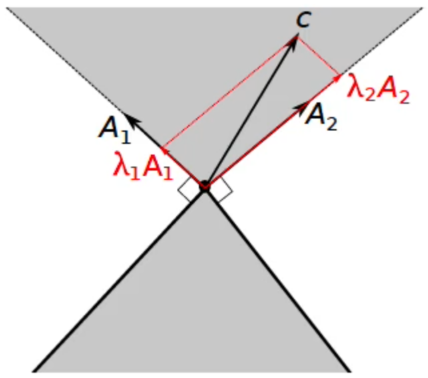


Figure 3.10: If $c \in C^*$ then there must be a set of non negative λ_i multipliers for the linear combination of gradients to produce c.

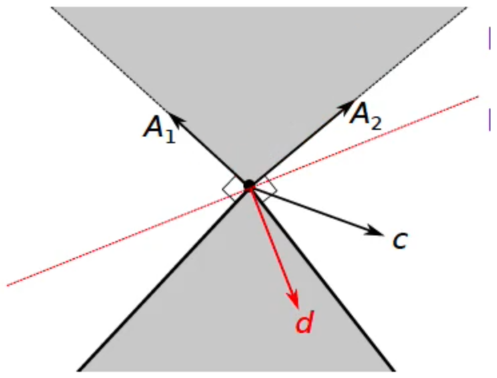


Figure 3.11: If $c \notin C^*$ then it must be the case that there is $d \in C$ such that $c \cdot d > 0$.

Note that we are saying that the opposite of the gradient must be written as the linear combination of the two sums. Moreover, note that we leave μ to be of any sign. This is not a random stuff. As we know, each equality constraint can be written as two inequality constraints:

$$\nabla h(x) = 0 \iff \nabla h(x) \leq 0 \wedge -\nabla h(x) \leq 0 \tag{3.30}$$

so there exist two positive scalars μ^+, μ^- such that:

$$\mu^+ \nabla h(x) \leq 0 \wedge \mu^- \nabla h(x) \leq 0 \tag{3.31}$$

thus we can write:

$$(\mu^+ - \mu^-)\nabla h(x) = 0 \tag{3.32}$$

This difference can be of any sign.

So if x^* is a local optimum then there must exist the multipliers λ and μ such that 3.29 holds. This is a necessary condition (under constraint qualifications), and also sufficient in case of convex optimisation.

3.2.2 From Farkas' lemma to KKT saint graal

So the equivalent problem from above takes name of **Karush-Kuhn-Tucker conditions**. The complete characterisation of KKT conditions is the following:

$$g_i(x) \leq 0, i \in \mathcal{I} \text{ and } h_j(x) = 0, j \in \mathcal{J} \qquad \text{(KKT-F)} \tag{3.33}$$

$$\nabla f(x) + \sum_{i \in \mathcal{I}} \lambda_i \nabla g_i(x) + \sum_{j \in \mathcal{J}} \mu_i \nabla h_j(x) = 0 \qquad \text{(KKT-G)} \tag{3.34}$$

$$\sum_{i \in \mathcal{I}} \lambda_i g_i(x) = 0 \qquad \text{(KKT-CS)} \tag{3.35}$$

The first condition requires that the point x is feasible. The second is the Farkas' lemma stuff, so $-\nabla f(x)$ has to be written as the linear combination of the gradients of the constraints. Note how we are using the whole set of inequalities not just the active ones. This is why we need the third constraint called **complementary slackness**. Think about it: with complementary slackness constraint, an inequality constraint i is either active, and then $g_i(x) = 0$ so λ can be any number; or the constraint i is not active, i.e. $g_i(x) < 0$, but due to the complementary slackness requirement, λ_i must be 0.

λ and μ are called **Lagrangian multipliers** or **Lagrangian** duals.

Let us now announce the saint graal of the constraint optimisation:

Theorem 3.2.3. *If the constraint qualification hold, that is $T_X(x) = D_X(x)$ and x is local optimum, then KKT conditions must hold, that is there must be some λ and μ such that the KKT constraints hold all together.*

Moreover, under convexity, this theorem becomes if and only if. So in the end, KKT conditions are the ones that recognise the local optimum.

3.2.3 The nasty critter: Lagrangian function

Let us briefly touch the bad guy: non convex case. In these cases, the first order information is not enough and we have to look also into the second order information. In the non convex case, the first order information is just telling us that we are in a stationary point, but it can be a saddle point, a maximum or a minimum. We need to look into the Hessian. What we do is to construct a mathematical object, whose second order derivative, the Hessian, will give us more information about the things we are looking for, that is the local optimum. This mathematical object is called **Lagrangian function**:

$$L(x; \lambda, \mu) = f(x) + \sum_{i \in \mathcal{I}} \lambda_i g_i(x) + \sum_{j \in \mathcal{J}} \mu_j h_j(x) \tag{3.36}$$

where x are the variables and λ and μ are the constant parameters. Look at this as a way to construct an infinite family of functions in x. Once that you have fixed the two parameters, you get a particular function in the variable x.

The gradient of this guy is exactly the second condition of the KKT system. You don't see it? Remember these two properties and then retry:

$$\nabla(a + b) = \nabla(a) + \nabla(b) \tag{3.37}$$

$$\nabla(\alpha a) = \alpha \nabla(a) \tag{3.38}$$

The points that satisfy KKT conditions are all the stationary points of the Lagrangian function. So as we did in the case of the unconstrained optimisation of non convex cases, we need to compute the second order derivative. We may think that it is sufficient to deal with the Hessian of the Lagrangian function. But the reality is a bit more complicated. In order to speak about the second order derivative, we need to define the concept of the **critical cone**:

$$C(x, \lambda, \mu) = \left\{ d \in \mathrm{R}^n : \begin{cases} \nabla g_i(x) \cdot d = 0, & i \in \mathcal{A}(x) \text{ s.t. } \lambda_i^* > 0 \\ \nabla g_i(x) \cdot d \leq 0, & i \in \mathcal{A}(x) \text{ s.t. } \lambda_i^* = 0 \\ \nabla h_j(x) \cdot d = 0, & j \in \mathcal{J} \end{cases} \right\} \tag{3.39}$$

The first condition says that the direction in the critical cone must be orthogonal to the gradient of the active inequality constraint whose λ coefficient is strictly positive; the second says it must be in the opposite direction of the gradient of the active inequalities whose λ is 0 (yeah man, it must go in the minimum direction); and the third one obviously says that it must remain orthogonal to the equality constraint.

Why are we interested in the critical cone? Because, if x^* is the local optimum of the constrained problem, then it turns out that the Hessian of the Lagrangian function in that point must be positive semidefinite not in all directions, but only in those of the critical cone.

Theorem 3.2.4. *If under constraint qualifications, the triple (x, λ, μ) satisfy KKT conditions, and x is local optimum, then:*

$$d^T \nabla^2 L(x; \lambda, \mu) d \geq 0, \forall d \in C(x, \lambda, \mu)$$

This is also sufficient condition if the Hessian is positive definite.

Until now, we have spoke about a function in x, without bothering about the other two parameters: λ and μ. But actually the Lagrangian dual is a function in three variables. So let us now consider the Lagrangian function as a function in only two variables, λ and μ, for a given x.

3.2.4 Lagrangian relaxation

Let us define the following function:

$$\psi(\lambda, \mu) = \min_{x \in \mathbb{R}^n} \{L(x, \lambda, \mu)\} \tag{3.40}$$

This function defines the **dual problem** or **dual function** of the original objective function.

As you an see, this is an unconstrained optimisation problem that is kind of a "wrapper" of the original Lagrangian function. Actually what we have just done is called **Lagrangian** relaxation. I don't know how to treat constraints efficiently. Cool, let us relax this fact and bring it into the objective function. If the feasibility is not respected on some inequality for example, then g_i and λ will be positive and so $L(x, \lambda, \mu)$ will be larger than $f(x)$. Think about a car on the highway. You should not go above the speed limit (constraints). No one is actually preventing you, so you can go (relaxation). But if police catches you doing that, you will pay a lot of money (violate constraints in the objective function).

This is an example of penalty method. Penalty methods are a certain class of algorithms for solving constrained optimisation problems. A penalty method replaces a constrained optimisation problem by a series of unconstrained problems whose solutions ideally converge to the solution of the original constrained problem. The unconstrained problems are formed by adding a term, called a penalty function, to the objective function that consists of a penalty parameter multiplied by a measure of violation of the constraints. The measure of violation is nonzero when the constraints are violated and is zero in the region where constraints are not violated.

Our question is now: how is the value of this optimisation problem related to the value of the original optimisation problem? Turns out that we have the following relation:

Theorem 3.2.5. $\psi(\lambda, \mu) \leq f(x)$

Proof. Suppose you have a $\bar{x}$ that is in the feasible region (not necessarily the optimum solution). Since it is in the feasible reason we have that:

$$g_i(\bar{x}) \leq 0 \qquad i \in \mathcal{I} \tag{3.41}$$

$$h_j(\bar{x}) = 0 \qquad j \in \mathcal{J} \tag{3.42}$$

Which means that the Lagrangian function becomes:

$$L(\bar{x}, \lambda, \mu) = f(\bar{x}) + \lambda G(\bar{x}) + \mu H(\bar{x}) = f(\bar{x}) + \lambda G(\bar{x}) \leq f(\bar{x})$$

Since ψ function is a minimisation problem over the Lagrangian function L, we have that:

$$\psi(\lambda, \mu) \leq f(\bar{x}), \forall \bar{x} \in \text{feasible region}$$

□

But $\bar{x}$ may also be the optimal solution x^*. We would like to have $\psi(\lambda, \mu)$ as close to $f(x^*)$ as possible. So what we need to do is to maximise the Lagrangian relaxation problem:

$$\max\{\psi(\lambda, \mu) : \lambda \in \mathbb{R}_+^{|\mathcal{I}|}, \mu \in \mathbb{R}^{|\mathcal{J}|}\} \tag{3.43}$$

This problem is called the **Lagrangian dual** of the original problem. ψ is a concave function, so local optima is global optima. Why concave? Well imagine that you fix L with a given x. Now look at the Lagrangian function defined in 3.36. Do you see it :)? That nasty critter is now a simple linear function in two variables λ and μ. Linear functions generate planes in three dimensional space or lines in two dimensional space. There are infinitely many lines (hyperplanes), one per each x. Consider the figure 3.12. Each line represents a particular choice of x. We are interested in the function ψ which is identified by the bold line. Clearly, this function is non differentiable and it is concave.

But now bad things: it may happen that the value is $-\infty$ and usually it is not differential, even if all the other stuff is differentiable. But most of all, each

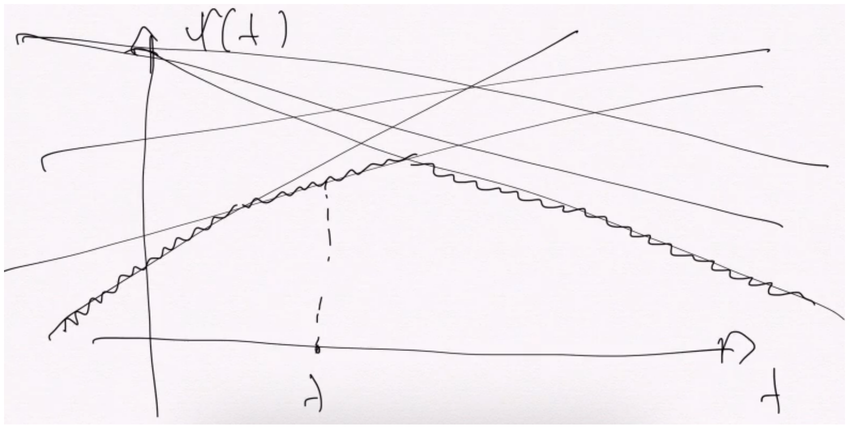


Figure 3.12: Caption

time we want to compute the value of the Lagrangian dual, we need to solve a minimisation problem inside. For each λ and μ I clearly have to minimise the Lagrangian relaxed problem.

The property $\psi(\lambda, \mu) \leq f(x)$ is called **weak duality**. We are interested in those cases where we have the equality, i.e. $\psi(\lambda, \mu) = f(x)$, which is called **strong duality**. Is it always the case that we have the strong duality? Unfortunately no. A counter-example may be:

$$\min\{-x^2 : 0 \leq x \leq 1\} \tag{3.44}$$

$$L(x, \lambda, \mu) = -x^2 + \lambda_1(x-1) + \lambda_2 x \tag{3.45}$$

$$\psi(\lambda, \mu) = \min\{L(x, \lambda, \mu\} = -\infty, \forall \lambda \in \mathbb{R}^2 \tag{3.46}$$

$$\psi(\lambda, \mu) = -\infty \leq f(x^*) = -1 \tag{3.47}$$

This is due to the fact that you are minimising a concave function, or maximising a convex function. But under convexity (minimisation) and constraint qualifications, the things work:

Theorem 3.2.6. *If the problem P is convex, x^* is local optimum and $T_X(x^*) = D_X(x^*)$ (constraint qualifications or regularity), then there exist optimal λ^*, μ^* for the dual problem such that $\psi(\lambda^*, \mu^*) = v(P)$.*

Proof. Since there are optimal λ^* and μ^* for some optimal solution for the problem P, x^*, we know that the gradient of the Lagrangian function is 0, that is:

$$\nabla_X L(x^*, \lambda^*, \mu^*) = 0$$

It is a saddle point. But since our function is convex, we know that:

$$v(D) \geq \psi(\lambda^*, \mu^*) = \min_{x \in \mathbb{R}^n} \{L(x, \lambda^*, \mu^*)\} = L(x^*, \lambda^*, \mu^*) = f(x^*) = v(P)$$

The first and the second passages are due to the definition. Since we have assumed that the optimal solution is in x^* we get the third passage. Now the forth passage is slightly more involved. Since everything is optimal, KKT conditions are satisfied. So the two large sums in the Lagrangian function become 0 (complementary slackness).

But we also know that the dual optimal value is a lower bound of the optimal value of the primal:

$$v(P) \geq v(D)$$

So it must be the case that the two values are the same:

$$v(P) = v(D)$$

□

3.2.5 Specialised duals

Linear programs

Dealing with duals and Lagrangian relaxation is not easy. It is a powerful tool, but we need to solve a max min problem.

There are cases where we can simplify the things a little bit. One of those cases are Linear programs.

$$\min\{cx : Ax \geq b\} \tag{3.48}$$

Let us compute the Lagrangian function of this problem:

$$L(x, \lambda) = cx + \lambda(b - Ax) = cx + \lambda b - \lambda Ax = \lambda b + x(c - \lambda A) \tag{3.49}$$

We have obtained a linear function in x. The linear function cannot be minimised, it is always $-\infty$. Except one case. The case when the slope of the linear function is 0. That is when we have a constant function. So we get the following:

$$\psi(\lambda) = \min_{x \in \mathbb{R}^n} L(x, \lambda) = \begin{cases} -\infty & \text{if } c - \lambda A \neq 0 \\ \lambda b & \text{if } c - \lambda A = 0 \end{cases} \tag{3.50}$$

So our dual problem becomes:

$$(D) \max\{\psi(\lambda) : \lambda \geq 0\} \equiv \max\{\lambda b : \lambda A = c, \lambda \geq 0\} \tag{3.51}$$

This is the famous dual of the linear program from operational research.

Quadratic programs

Another interesting special case is the quadratic program. Let us first start with the following one:

$$\min\{\frac{1}{2}\|x\|^2 : Ax = b\} \tag{3.52}$$

The Lagrangian relaxation becomes:

$$L(x,\mu) = \frac{1}{2}\|x\|^2 + \mu(Ax - b) \tag{3.53}$$

The gradient of the Lagrangian function is then:

$$\nabla L(x,\mu) = x + \mu A \tag{3.54}$$

Since the gradient of the Lagrangian function must be 0 in the optimum solution, we have that:

$$\nabla L(x,\mu) = x + \mu A = 0 \to x = -\mu A \tag{3.55}$$

The dual problem them becomes:

$$\begin{aligned} \psi(\mu) = \min_{x\in\mathbb{R}^n} L(x,\mu) = L(-\mu A,\mu) = \frac{1}{2}\|-\mu A\|^2 + \mu(A(-\mu A) - b) = \\ = \frac{1}{2}\mu^T AA^T\mu - \mu^T AA^T\mu - b\mu = -\frac{1}{2}\mu^T AA^T\mu - b\mu \end{aligned} \tag{3.56}$$

$$\max_{\mu\in\mathbb{R}^n} \Big\{ -\frac{1}{2}\mu^T AA^T\mu - b\mu \Big\} \tag{3.57}$$

This is a real unconstrained problem.

Actually, more generally we could have:

$$\min\{\frac{1}{2}x^T Qx : Ax \geq b\} \tag{3.58}$$

Under the condition that A is not singular, and thus invertible, we have the following dual problem:

$$\max\{\lambda b - \frac{1}{2}v^T Q^{-1}v : \lambda A - v = q, \lambda \geq 0\} \tag{3.59}$$

All of this is very nice. Imagine that you have a lot of variables, say ten thousands, and very few constraints, say less than 10. Dealing with the dual problem means dealing with a much smaller amount of variables :).

Conic programs

TODO

3.3 Algorithms

We are now going to study some algorithms for solving constrained optimisation programs. In particular we will see three algorithms for solving linearly constrained quadratic programs and two for linearly constrained non linear programs.

Thus we will deal with only linear constraints and the difficult part will eventually be the objective function. This is because it is much easier and because this is what we typically do in the machine learning. Sometimes when possible we will do some references to the more general non linear but convex constraints $G(x) \leq 0$. We will exclude non linear and non convex cases. Mainly because of the time limit, but also because they are not that useful in the machine learning world. Typically if you have something that is not linear what you usually do is to approximate the non linear function with something quadratic, e.g. you use Newton. So quadratic programs are quite central.

We will not take into account equality constraints because we can always convert them into inequality constraints. Moreover, the problems with only equality constraints we know how to deal with them, since it is almost unconstrained optimisation.

3.3.1 Equality constraints for quadratic programs

Let us start small. We have already seen quadratic programs with equality constraints. Let us look at it again, this time from the constraint optimisation point of view.

Our problem is of the following form:

$$\min \left\{ \frac{1}{2} x^T Q x + qx : Ax = b \right\} \tag{3.60}$$

where $A \in \mathbb{R}^{m \times n}$ with $\operatorname{rank}(A) = m < n$ and rows of A linearly independent. Note that $Q \succeq 0$ otherwise $v(P) = -\infty$ almost always.

Note that since we have just equality constraints, there are no λs and thus no complementary slackness condition. The KKT system then becomes:

$$Ax = b \tag{3.61}$$

$$\nabla f(x) + \mu \nabla H = Qx + q + \mu A = 0 \tag{3.62}$$

Which becomes:

$$Ax = b \tag{3.63}$$

$$Qx + \mu A = -q \tag{3.64}$$

Which can be also written as:

$$\begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \mu \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix} \tag{3.65}$$

This system is symmetric but indefinite. A lot of eigenvalues are 0. This is where Bolzano guy from Pisa comes into play.

We could solve this system directly, plain linear algebra. But we can also exploit the particularity of this system in order to craft some efficient methods to solve it. We need to be very efficient since this problem frequently appears as subprogram of a much larger program.

One of the possibility is to go by Reduced KKT. If we know that the $Q \succ 0$ then it means that our objective function is strictly convex. We can then do the following. Let us multiply both sides of the main KKT condition by AQ^{-1}:

$$AQ^{-1}Qx + AQ^{-1}A^T\mu = -AQ^{-1}q \tag{3.66}$$
$$Ax + AQ^{-1}A^T\mu = -AQ^{-1}q \tag{3.67}$$
$$b + AQ^{-1}A^T\mu = -AQ^{-1}q \tag{3.68}$$
$$AQ^{-1}A^T\mu = -b - AQ^{-1}q \tag{3.69}$$

Let us call M the matrix $AQ^{-1}A^T$. We get the following linear system:

$$M\mu = -(b + AQ^{-1}q) \tag{3.70}$$

The solution of this system (μ) can be then used to find x:

$$Qx + \mu A^T = -q \tag{3.71}$$
$$x = Q^{-1}(-A^T\mu - q) \tag{3.72}$$

The matrix M is positive semi definite and it is a square matrix ($M \in \mathbb{R}^{m\times m} \wedge M \succeq 0$). This is particularly useful when we have very few constraints (m).

Another way to solve this problem is the one that we have already examined, Null Space Method. Let us divide A in $[A_B, A_N]$ and x in $[x_B, x_N]$ in a way that $\det(A_B) \neq 0$. In this way A_B is non singular and so we can invert it:

$$x_B = A_B^{-1}(b - A_N x_N) = A_B^{-1}b - A_B^{-1}A_N x_N \tag{3.73}$$

Let us now call $D = -A_B^{-1}A_N$ and $d = A_B^{-1}b$. We get then:

$$x_B = Dx_N + d \tag{3.74}$$

where:

$$d = \begin{bmatrix} b \\ 0 \end{bmatrix} D = \begin{bmatrix} -A_B^{-1}A_N \\ I \end{bmatrix} \in \mathbb{R}^{m\times n-m} \tag{3.75}$$

The matrix D is called the basis of null space of A since we have that:

$$AD = \begin{bmatrix} A_B & A_N \end{bmatrix} \begin{bmatrix} -A_B^{-1}A_N \\ I \end{bmatrix} = -A_B A_B^{-1}A_N + A_N I = -A_N + A_N = 0 \tag{3.76}$$

Let us no multiply the KKT main condition by D^T:

$$D^TQx + D^TA^T\mu = -D^Tq \tag{3.77}$$

But since $x = Dx_N + d$, the above equality becomes:

$$D^TQ(Dx_N + d) = -D^Tq \Rightarrow= (D^TQD)x_N = -D^T(Qq + d) \tag{3.78}$$

We call $H = D^TQD \in \mathbb{R}^{n-m\times n-m}$ reduced Hessian and it is positive semi definite.

So which method is the best? It depends. Look at the matrices dimensions. If the number of constraints is almost the same as the number of variables then the second one is probably the right choice. If the two numbers are quite different and the number of overall constraints is small, then probably the first one is the one to go.

3.4 Projected gradient method

Since the first algorithm that we study for unconstrained optimisation, we will also do the same for the constrained case.

This is a method that can be applied to the general non linear objective function with linear constraints:

$$\min_{x\in\mathbb{R}^n}\{f(x) : Ax \le b\} \tag{3.79}$$

What we typically do is the line search. So suppose we are in a feasible solution x. We want to move in a direction that is feasible. Since everything is linear here (in the sense of the constraints) all the three cones that we have studied are the same. So we need to pick a direction in the first order feasible direction cone $D_X(x)$ restricted to the active constraints:

$$D_X(x) = \{d \in \mathbb{R}^n : A_{\mathcal{A}(x)}d \le 0\} \tag{3.80}$$

We know that the best direction to take is the one of the opposite of the gradient in x. So if $\nabla f(x) \in D_X(x)$ then we can just line search on the direction $d = -\nabla f(x)$. Note that if x is in the interior, i.e. $\mathcal{A}_X = \emptyset$ then $D_X(x) = \mathbb{R}^n$, in other words I can go wherever I want.

If the gradient does not belong to the set of first order feasible direction cone, then let us find the direction that is as close as possible to the one of the opposite of the gradient. If you think a little bit, the closest direction to $-\nabla f(x)$, which is located outside of the cone, is the direction d that goes along the boundary of the cone on the side of $-\nabla f(x)$.

Let us do now some math. We want to project the gradient $\nabla f(x)$ onto $\partial D_X(x)$. Let us describe mathematically the frontier:

$$\partial D_X(x) = \{d \in \mathbb{R}^n : A_{\mathcal{A}(x)}d = 0\} \tag{3.81}$$

Projection means that we want to minimise the following problem (the closest point, the norm, man):

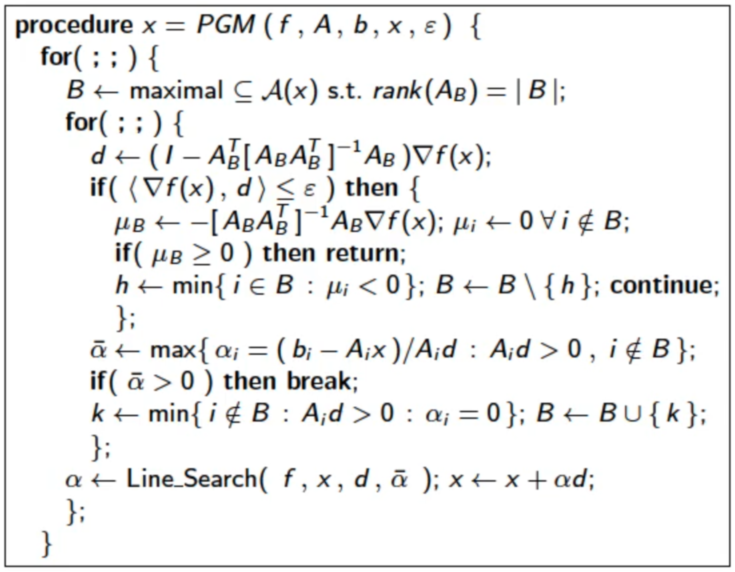
$$\min\left\{\frac{1}{2}\|\nabla f(x) - d\|^2 = \frac{1}{2}d^T I d - d^T \nabla f(x) : A_{\mathcal{A}(x)}d = 0\right\} \tag{3.82}$$

This is a linearly constrained quadratic problem with a very special Q, that is identity matrix. Note that the identity matrix is strictly positive, that is it is positive definite. So we can use Reduced KKT to solve it in the following manner. First we get μ by solving this system:

$$[A_{\mathcal{A}(x)}A^T_{\mathcal{A}(x)}]\mu = A_{\mathcal{A}(x)}\nabla f(x) \tag{3.83}$$

Then we use μ to find the projected direction:

$$d = -(A^T_{\mathcal{A}(x)}\mu - \nabla f(x)) \tag{3.84}$$



```
procedure $x = PGM\,(f, A, b, x, \varepsilon)$ {
  for( ; ; ) {
    $B \leftarrow$ maximal $\subseteq \mathcal{A}(x)$ s.t. $rank(A_B) = |\,B\,|$;
    for( ; ; ) {
      $d \leftarrow (I - A_B^T[A_B A_B^T]^{-1} A_B)\nabla f(x)$;
      if( $\langle \nabla f(x), d \rangle \leq \varepsilon$ ) then {
        $\mu_B \leftarrow -[A_B A_B^T]^{-1} A_B \nabla f(x)$; $\mu_i \leftarrow 0\ \forall\, i \notin B$;
        if( $\mu_B \geq 0$ ) then return;
        $h \leftarrow \min\{\, i \in B \,:\, \mu_i < 0 \,\}$; $B \leftarrow B \setminus \{\, h \,\}$; continue;
        };
      $\bar{\alpha} \leftarrow \max\{\, \alpha_i = (\, b_i - A_i x\,)/A_i d \,:\, A_i d > 0\,,\ i \notin B \,\}$;
      if( $\bar{\alpha} > 0$ ) then break;
      $k \leftarrow \min\{\, i \notin B \,:\, A_i d > 0 \,:\, \alpha_i = 0 \,\}$; $B \leftarrow B \cup \{\, k \,\}$;
      };
    $\alpha \leftarrow$ Line_Search( $f, x, d, \bar{\alpha}$ ); $x \leftarrow x + \alpha d$;
    };
}
```

Figure 3.13: Projected Gradient Method.

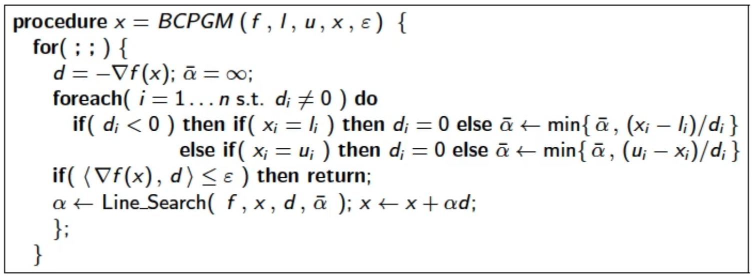
Actually, if the matrix of the active constraints in x, $A_{\mathcal{A}(x)}$, is non singular, we can also directly get d by doing the following computation:

$$d = (I - A_{\mathcal{A}(x)}^T (A_{\mathcal{A}(x)} A_{\mathcal{A}(x)}^T)^{-1} A_{\mathcal{A}(x)})\nabla f(x) \tag{3.85}$$

$I - A_{\mathcal{A}(x)}^T (A_{\mathcal{A}(x)} A_{\mathcal{A}(x)}^T)^{-1} A_{\mathcal{A}(x)}$ is called **projection operator**. This operator projects our nice gradient on the boundary of the feasible region. Remember good old Poloni: householder reflection stuff.

So once we have the direction we can do the line search. Almost. d can be also 0. If this is the case, and $\mu \geq 0$ this means that we have found the optimum. Why? Well, look at the original problem. We had $Ad \leq 0$. But since our $-\nabla f(x)$ is not a feasible direction, we had to project it by forcing the inequality constraint into the equality constraint. So if there are no μ_i negative, it means that all the equalities are satisfied. So we are in the optimum. But if μ is not all positive, then it means that there is a constraint that should not be in the active set $\mathcal{A}_(x)$. We will talk in a minute about that.

The algorithm is shown in figure 3.13. Let us now go through the algorithm. You start at a point x. The first thing that you do is to construct the set of the active constraints in x. We take the maximal set of active constraints such that the rank of that set is full rank. We must do this because it is the only way to force non singularity of the matrix $A_B A_B^T$. We annotate that maximal subset of $\mathcal{A}(x)$ with B in the algorithm. Note that we may clearly leave apart some of the active constraints in x, and we will have to deal with that. So now we compute the direction d. Such direction can either be 0 or not. So if it is 0, then we check if $\mu \geq 0$. We set to 0 all the entries of μ whose corresponding



```
procedure $x = BCPGM(f, l, u, x, \varepsilon)$ {
  for( ; ; ) {
    $d = -\nabla f(x)$; $\bar{\alpha} = \infty$;
    foreach( $i = 1 \dots n$ s.t. $d_i \neq 0$ ) do
      if( $d_i < 0$ ) then if( $x_i = l_i$ ) then $d_i = 0$ else $\bar{\alpha} \leftarrow \min\{\bar{\alpha}, (x_i - l_i)/d_i\}$
                      else if( $x_i = u_i$ ) then $d_i = 0$ else $\bar{\alpha} \leftarrow \min\{\bar{\alpha}, (u_i - x_i)/d_i\}$
    if( $\langle \nabla f(x), d \rangle \leq \varepsilon$ ) then return;
    $\alpha \leftarrow$ Line_Search( $f, x, d, \bar{\alpha}$ ); $x \leftarrow x + \alpha d$;
    };
  }
```

Figure 3.14: Projected Gradient Method with Box Constraints.

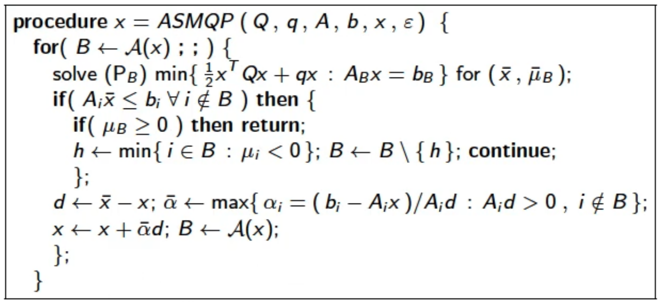
active constraint has not been included in B previously. In case $\mu \geq 0$, we are done. Otherwise we find the entry with negative value in μ that has the lowest index and we throw from B the corresponding constraint. Yes. We just ignore it. And we repeat. Two cases eventually happen. Either we end up again with $d = 0$ and the whole process repeats until we get out from the whole procedure because we hit $\mu \geq 0$ or we find a $d \neq 0$. In this last case we proceed in the algorithm. Since we left some of the constraints out of the set B we now must verify that the direction that we found does not violate some of the constraints out of B. We take the step size along each such constraint and then we take the minimum of those values. If the minimum is greater than 0, it means we can proceed along d. If $d \leq 0$ it means that some of the constraints is active but not in B. So we include that constraint and we repeat.

Let us now state without proving some of the properties about PGM. First of all, the addition of the constraints in B preserve the linear independence. That is we do not risk to make B singular. It is also not possible to have the add/remove process cyclical. That is the process of addition and removal of constraints eventually terminates. This is called **Bland's anti-cycle rule**. Maximal B is easy to get, just do the greedy algorithm. Each constraint either we put it or not in B depending on whether it is linearly independent from all the constraints already belonging to B. How? For each matrix that you form with a new vector, check that the determinant of that matrix is not 0.

Note that if f is a simple linear function, then PGM is just the **Primal Simplex Method**.

Sometimes in ML the constraints are much easier than the general constraints. One of the easiest types of constraints is called **box constraints**. Each variable belongs to an interval of values, $l \leq x \leq u$. If we have a non linear program with box constraints, the algorithm becomes much more simpler (see figure 3.14).

For each box constraint, if it is active then x is either l or u. So the set of active constraints is always linearly independent. How this algorithm works? We start with the direction opposite of the gradient. For each entry i of the direction vector d that is $\neq 0$ (remember, we want d to be 0 in order to terminate), we do the following: if $d_i < 0$ we need to lower a bit x_i. But if $x_i = l_i$ we just set



```
procedure $x = ASMQP(Q, q, A, b, x, \varepsilon)$ {
  for( $B \leftarrow \mathcal{A}(x)$ ; ; ) {
    solve $(P_B)$ $\min\{ \frac{1}{2}x^T Qx + qx \;:\; A_B x = b_B \}$ for $(\bar{x}, \bar{\mu}_B)$;
    if( $A_i\bar{x} \leq b_i \; \forall\, i \notin B$ ) then {
      if( $\mu_B \geq 0$ ) then return;
      $h \leftarrow \min\{ i \in B \;:\; \mu_i < 0 \}$; $B \leftarrow B \setminus \{ h \}$; continue;
      };
    $d \leftarrow \bar{x} - x$; $\bar{\alpha} \leftarrow \max\{ \alpha_i = (b_i - A_i x)/A_i d \;:\; A_i d > 0 \,,\, i \notin B \}$;
    $x \leftarrow x + \bar{\alpha} d$; $B \leftarrow \mathcal{A}(x)$;
    };
}
```

Figure 3.15: Active set method for quadratic programming

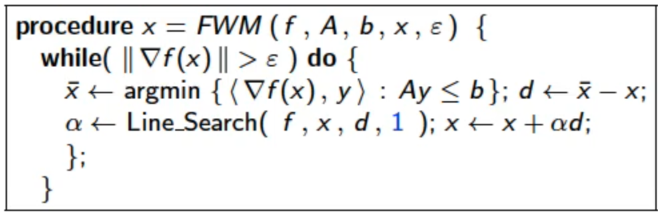
it to 0 since we cannot go anymore down. Otherwise we calculate the minimum step size for the new direction by taking the minimum between the previous minimum step sizes and the current one. The same process applies for $d_i > 0$.

3.5 Active-set method for Quadratic Programming with linear inequalities

Suppose now we have a quadratic program with linear inequalities. If we knew which were the active constraints in the optimal solution x^* we would just apply linear algebra and the world would be a better place. But unfortunately we don't know that. We like, especially in ML, to guess when we don't know exactly the things. We need also to be prepared to also revise our estimates.

The algorithm is shown in figure 3.15. Let us go through it. The first thing that we note is that we do not require that the matrix B contains just linearly independent set of active constraints. Note that this set can also be empty, i.e. $B = \emptyset$. What we do then is to solve the linear equality constrained quadratic program. Note that we are hiding the concept of B being full rank inside that subroutine. The rest is quite straightforward. If $\bar{x}$ is a feasible solution, then we have found the optimum if $\mu_B \geq 0$. Note that we know that it is optimal on A_B, so we need to check only the constraints not in B. If this is not the case we proceed as in the case of the projected gradient method. The only thing to say here is that as you can see we do not need to do any line search. This is because we are in the case of the quadratic program. We can just take the maximum step size in the descent direction.

This method can be also extended to f general, that is not necessarily quadratic. What changes is how we solve the subproblem. We could use for instance quasi-Newton. TODO box contraints



```
procedure $x = FWM(f, A, b, x, \varepsilon)$ {
  while( $\|\nabla f(x)\| > \varepsilon$ ) do {
    $\bar{x} \leftarrow$ argmin $\{ \langle \nabla f(x), y \rangle : Ay \leq b \}$; $d \leftarrow \bar{x} - x$;
    $\alpha \leftarrow$ Line_Search( $f, x, d, 1$ ); $x \leftarrow x + \alpha d$;
    };
  }
```

Figure 3.16: Frank Wolfe Method, also called Conditional Gradient Method.

3.6 Frank Wolfe method, aka Conditional Gradient

The main difficulty with projected gradient method and active set method is the active set. Frank Wolfe tries to do without it at all. We will deal here with non linear programs with linear constraints.

The idea of Frank Wolfe is quite simple (figure 3.16. It assumes there is someone else that can deal with the linear problem and let us concentrate only on the non linear part of the problem which is in the objective function. It iteratively approximate the our nasty function with the first order linear model, minimise this program in the admissible region and take that direction as the direction of descent.

Another thing worth mentioning is that we do not need to compute the maximum step size. We just set it to 1.

This algorithm is particularly easy to implement but unfortunately it is not that good when it comes to the convergence. This is because we trust the linear model on the whole admissible region. It is accurate locally to x but probably this is not the case when we look at the whole admissible region. What we can do to improve the convergence is the stabilisation. TODO

3.7 Dual Methods

Frank Wolfe completely hides Lagrangian multipliers. Suppose again we have a quadratic program with linear constraints.