Notes on Continuous Optimization

October 12, 2020

Abstract

Lecture notes on continuous optimization theory, covering part of BEMACS Computer Programming's course.

Contents

1 Introduction and notation

Optimization is an ubiquitous task in science and applications. A large number of important and different problems, such as trying to asses the correct behavior for a rational agent (decision theory), selecting the correct model explaining some observed data (machine learning), and finding the best allocation of finite resources for a company or a government, can be formally framed as a mathematical optimization problem. Quite surprisingly, most laws of physics as well can be derived from an optimization principle (the Principle of Least Action).

Given the special relevance of optimization for many seemingly unrelated disciplines, a huge effort has been done in the last century to produce increasingly more powerful and sophisticated tools to solve optimization problems. These tools involve general and problem specific algorithms, worst case complexity guarantees (bounds), mathematical proofs for convergence rate, and from the practical perspective highly optimized numerical libraries. In this introductory notes we will explore some of the basic mathematical concepts in optimization theory.

These are some (rough) notes on the continuous optimization theory part of the course, to be studied as a complement to the Nocedal and Wright's textbook, and whose aim is to make a gentler introduction to the topic for undergraduate students enrolled in scientific bachelors.

1.1 Readings for the course

The course Computer Programming in the BEMACS program at Bocconi consists in three parts: 1) Monte Carlo methods; 2) continuous optimization; 3) data structures and dynamic programming.

For part 1 it was not possible to find a concise reference textbook and we rely entirely on lecture notes. For part 3 we strictly follow some selected chapters of Cormen's textbook "Introduction to Algorithms". For part 2, which is the subject of this part of the notes, we follow Nocedal and Wright textbook (without delving too much into the mathematical details) along with these lecture notes.

Textbook:

- Jorge Nocedal and Stephen J. Wright, "Numerical Optimization. 2ed." (2006).

Mandatory readings:

- these notes

- Chapters 1, 2, 6.2

Suggested readings:

- Appendix A.1 (mathematical background): up to the paragraph “Determinant and Trace”
- Appendix A.2 (mathematical background): up to the paragraph “Mean value theorem”
- Chapter 3: 3.1, 3.2, 3.3
- Chapter 6: 6.1

Further (advanced) readings:

- Yurii, Nesterov, "Introductory Lectures on Convex Optimization: A basic course". This is a self-contained introductory textbook, mathematically heavier than Nocedal and not covering as many topics, but in some sense more inspiring.

1.2 Notation

- $\mathbb{R}$: the set of real numbers.
- $\forall x \in A$: for each element x in the set A. When the set is clear from the context (e.g. the set of real numbers), we just write $\forall x$.
- $\exists x$: there exist some x. It is usually followed by some proposition, "there exists an x such that ...."
- $\boldsymbol{x}$, $\vec{x}$: bold symbols and symbols with arrows represent vectors in some n-dimensional space, i.e. $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$. Sometimes we will omit the bold/arrow and just write x when the fact that x is a vector is clear from the context.
- $f : A \to B$: the function f has domain A and takes values in the set B. We will always consider real-valued functions, i.e. $B = \mathbb{R}$, on the one-dimensional or n-dimensional Euclidean space, i.e. $A = \mathbb{R}$ or $A = \mathbb{R}^n$.
- $f'(x)$: derivative of the one-dimensional function f at point x

- $\nabla f(\boldsymbol{x})$: gradient of the multi-dimensional function f at point $\boldsymbol{x}$. The gradient at point $\boldsymbol{x}$ is a vector with the partial derivatives as components.

- $g(x) = o(h(x))$ for $x \to a$: $g(x)$ becomes smaller than $h(x)$ when x goes to a, i.e. $\lim_{x\to a} \frac{g(x)}{h(x)} = 0$. This notation is very convenient for simplifying expressions and keeping only relevant terms when considering asymptotic behaviors. For example, we have:

$$x^2 + 2x^4 + x^5 = x^2 + o(x^3) \qquad \text{for } x \to 0, \tag{1.1}$$

$$\log(x) + 3x^5 + 4x^8 = 4x^8 + o(x^6) \qquad \text{for } x \to +\infty, \tag{1.2}$$

2 Continuous optimization in one dimension

We consider here one-dimensional unconstrained continuous optimization problems, where one has to find the *global minimizer* of an objective/cost/energy function $f : \mathbb{R} \to \mathbb{R}$, assumed to be bounded from below (i.e. $\exists c \in \mathbb{R}$ such that $f(x) > c \ \forall x$).

Therefore, we have to solve the optimization problem

$$\min_{x\in\mathbb{R}} \ f(x). \tag{2.1}$$

In other words we are looking for a point, x^*, such that any other point gives larger or equal value of f, i.e. $f(x) \geq f(x^*) \ \forall x$.

We will always assume that the function $f(x)$ is continuous, since in the majority of cases it is practically impossible to minimize more difficult functions such as the discontinuous ones.

We will also assume that the above problem is well posed, i.e. that the minimizer is not at $\pm\infty$ (this is not the case for e.g. $f(x) = \frac{1}{1+x^2}$).

If $f(x)$ is continuously differentiable (that is, the derivative $f'(x)$ exists and it is continuous in $\mathbb{R}$) the global minimum x^* is a *stationary point*, satisfying the condition

$$f'(x) = 0. \tag{2.2}$$

Global minima are not the only stationary points. In fact also local minima, maxima and saddles (see Fig. 2.1) all satisfy condition (2.2).

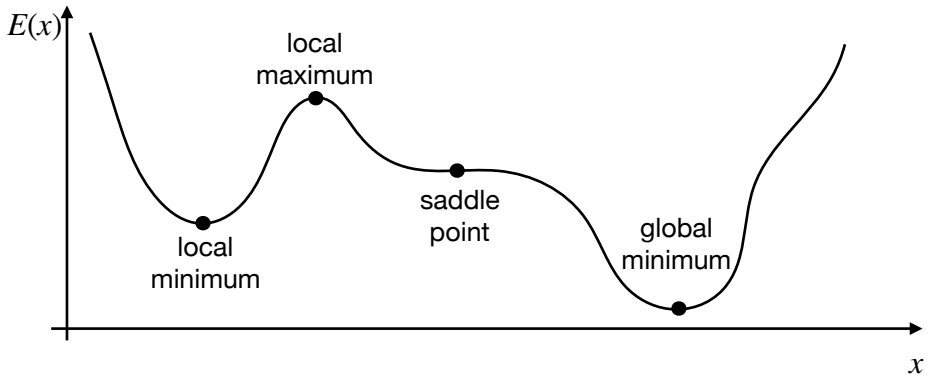

Figure 2.1: Stationary points of a cost function in 1D. In optimization problems we would like to find the global minimum, but often a local one is sufficient for practical purposes.

A local minimum is a point which is a (global) minimum only if the space is restricted to a neighborhood of the point itself.

If we were able to solve the equality (2.2), we would also solve the minimization problem (2.1) by selecting the stationary point with lowest cost. Unfortunately, Eq. (2.2) admits an analytic solution only in few cases, while generally one has to resort to numerical algorithms to perform the minimization. This is essentially the reason of existence of the huge field called numerical optimization or mathematical programming.

A very important family of functions that can be exactly minimized is the family of *quadratic functions:*

$$\min_{x \in \mathbb{R}} \; f(x) \coloneqq \frac{1}{2} a x^2 + b x + c, \tag{2.3}$$

where we suppose $a > 0$ (otherwise $f(x)$ would be unbounded below). In this case, $f(x)$ has a unique stationary point, the global minimum x^*, which we find applying condition (2.2) to $f'(x) = ax + b$:

$$a\, x^* + b = 0 \quad \Rightarrow \quad x^* = -\frac{b}{a}. \tag{2.4}$$

More generally, finding a global minimizer when there are many local minima is a very hard task, and one would be happy enough with an algorithm that is able

to approximately reach any local minimizer x^*. For a given tolerance $\epsilon > 0$, this amounts to finding an x^{appr} such that

$$|f(x^{appr}) - f(x^*)| = f(x^{appr}) - f(x^*) < \epsilon. \tag{2.5}$$

Notice that we don't need an absolute value in the right hand side (r.h.s.) since $f(x) \geq f(x^*) \ \ \forall x$ in a neighborhood of x^*, since x^* is a local minimum. From now on we will denote equivalently with x^* either a global or a local minimum.

We would like to implement an *iterative procedure,* that starts from some initial conditions, $x^{(0)}$, and that updates at iteration k the proposal to $x^{(k)}$ through some rule that uses the information collected during the previous updates. k has the role of a time index in our iterative procedure.

In the following, we drop for convenience the parenthesis in the superscript unless there is risk of ambiguity with the exponentiation operation.

So we are looking for a sequence

$$x^0 \to x^1 \to \cdots \to x^k \to \dots$$

that gets as closer and closer to a minimizer x^*.

A very important piece of information that we can collect during the iteration and that can help us deciding the next move is the derivative of the function at the given position. So we assume that if we find ourselves in a position $\bar{x}$, we can query for the value $f(\bar{x})$ of the function itself and for its derivative $f'(\bar{x})$.

Both $f(\bar{x})$ and $f'(\bar{x})$ are just *pointwise* pieces of information, but thanks to the regularity of $f(x)$ (e.g. continuity), they also tell us something on how the function $f(x)$ behaves in proximity of $\bar{x}$. In fact, by the very definition of differentiability, we know that

$$f(x) = f(\bar{x}) + f'(\bar{x})(x - \bar{x}) + o(x - \bar{x}). \tag{2.6}$$

This means that the remainder of the first order Taylor expansion goes to zero faster than $x - \bar{x}$ when $x \to \bar{x}$, and that $f(x)$ can be locally approximated by a linear function. We can generalize this argument using the *Taylor expansion* of order m for f around some point $\bar{x}$, given by:

$$f(x) \approx \sum_{i=0}^{m} \frac{f^{(i)}(\bar{x})}{i!}(x - \bar{x})^i$$

Here $f^{(i)}(\bar{x})$ denotes the i-th derivative in position $\bar{x}$. For a given x and in the limit $m \to +\infty$, the sum of the series may converge to the value $f(x)$ (e.g. this is

the case for the Taylor expansion of e^x), or to something else, or diverge (e.g. the series of $\log(1+x)$ for $x > 1$). Anyway it is fair to assume that increasing the order of the series we get better and better approximations for the first few orders as long as x is close enough to $\bar{x}$.

The first order expansion approximates f with a line that is tangent to f in the point $(\bar{x}, f(\bar{x}))$. The second order expansion approximates f with a parabola and so forth so on.

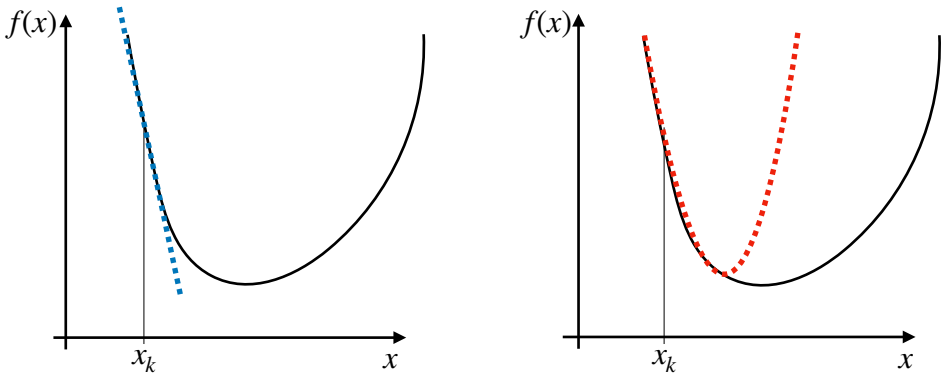

Figure 2.2: Approximation of $f(x)$ by first (*left*) and second (*right*) order Taylor expansion.

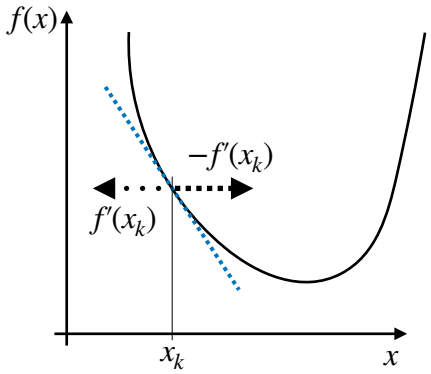

Figure 2.3: Representation of the 1-d gradient descent.

Back to our iterative algorithm, we know that at iteration k, the derivative $f'(x^k)$ tells us the direction and the rate of the *local increase* of $f(x)$ in a neighborhood

of x^k. Therefore, during our procedure that seeks a minimizer, it seems convenient to choose the next candidate in the form $x^{k+1} = x^k + \alpha p^k$, where the step p^k is in the opposite direction to the one pointed by the derivative. The parameter $\alpha > 0$ is called *step size* and is also known in the machine learning community as learning rate. Formally, assuming we are not already in a local minimum (where $f'(x^k) = 0$), we want to take a step p^k such that the following condition, called *descent condition,* holds*:*

$$f'(x^k)p^k < 0. \tag{2.7}$$

This guarantees that if α is small enough, the cost will be reduced in the next iteration, as can be seen combining last equation with Eq. (2.6):

$$f(x^{k+1}) = f(x^k) + \alpha f'(x^k)p^k + o(\alpha p^k). \tag{2.8}$$

In fact, by definition of the small o notation, for α small enough $o(\alpha p^k)$ will be small compared to $\alpha f'(x^k)p^k$, and we will have $f(x^{k+1}) < f(x^k)$.

Notice that (2.7) is automatically satisfied if we choose $p^k = -f'(x^k)$.

In 1D Eq. (2.7) just tells us we have to select the right sign for p^k, but the same condition can be generalized to higher dimension and becomes less trivial.

In Algorithm 1 we implement the simplest iterative procedure for approximately solving 1D optimization problems, the Steepest Descent procedure.

In this procedure, we fix the step size α during the iterations. It has to be chosen small enough so that the iterations are able to converge to a local minimum without exploding, but also large enough so that the minimizing procedure does not take too much time to converge. More refined and adaptive rules for choosing α will be briefly discussed later.

We note that in Algorithm 1 we imposed a maximum number of iterations k_{MAX} for simplicity. It is also common to have an additional criterion, typically in the form $|x^{k+1} - x^k| < \epsilon$ for some small ϵ, that stops the iterative procedure when convergence is achieved.

Algorithm 1 Steepest Descent 1D

Require: cost function f, step size $\alpha > 0$, init. conf. x^0, max iters k_{MAX}
Ensure: candidate minimun of f
  $x \leftarrow x^0$
  for $k \leftarrow 1, ..., k_{MAX}$ **do**
    $x \leftarrow x - \alpha f'(x)$
  end for
  return x

When available, *second order information*, namely $f''(x^k)$, provides information about the local curvature, a refinement of the local characterization that we have with just the gradient. In this case, we have

$$f(x) = f(x^k) + f'(x^k)(x - x^k) + \frac{1}{2}f''(x^k)(x - x^k)^2 + o((x - x^k)^2), \tag{2.9}$$

$$f'(x) = f'(x^k) + f''(x^k)(x - x^k) + o(x - x^k). \tag{2.10}$$

Here we obtain a more precise approximation of the function f using a parabola instead of a line (see Fig. (2.2)). We will see how this leads to an optimization algorithm, called Newton method, that is much faster than gradient descent.

Now we move on to the general optimization problem over n-dimensional space. Fortunately, many of the concepts we have developed for 1d case covered in this section readily extend to the multi-dimensional case.

3 Continuous optimization in many dimensions

3.1 Problem setting and first order necessary condition

Let us now consider multi-dimensional unconstrained continuous optimization problems, where for a given cost function $f : \mathbb{R}^n \to \mathbb{R}$ one wants to solve:

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \; f(\boldsymbol{x}). \tag{3.1}$$

In the following, we will denote with a plain x a variable in a $n-$dimensional space, i.e. a vector $x = (x_1, x_2, \ldots, x_n)$. Common notations used to distinguish vectors from scalars are $\boldsymbol{x}$ and $\vec{x}$. Since we will mostly deal with high dimensional spaces, sometimes we won't remark such difference, which should be clear from the context. Many of the considerations from previous chapters generalize to this multi-dimensional setting.

Gradients. The high-dimensional analogue of the derivative is the gradient, $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$, a function that associates to each point $\boldsymbol{x}$ a vector that contains the partial derivative of f:

$$\nabla f(\boldsymbol{x}) = \left(\left.\frac{\partial f}{\partial x_1}\right|_x, \left.\frac{\partial f}{\partial x_2}\right|_x, \ldots, \left.\frac{\partial f}{\partial x_n}\right|_x \right). \tag{3.2}$$

The partial derivative of f with respect to component i at point x is defined as the standard derivative of $f(\boldsymbol{x})$ when all the other component are fixed:

$$\left.\frac{\partial f}{\partial x_i}\right|_{\boldsymbol{x}} = \lim_{\epsilon \to 0} \frac{f(x_1, \ldots, x_i + \epsilon, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n)}{\epsilon} \tag{3.3}$$

When the gradient is defined and continuous everywhere, the function is said to be continuously differentiable.

Scalar product and norm. Let's now introduce the Euclidean inner product, $\langle \cdot, \cdot \rangle$. For any $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ it is defined by

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle \coloneqq \sum_{i=1}^{n} u_i v_i. \tag{3.4}$$

It can be also written as $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \boldsymbol{u}^T \boldsymbol{v}$ or $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \boldsymbol{u} \cdot \boldsymbol{v}$. Notice that for each non-zero vector, $\boldsymbol{u} \neq \boldsymbol{0}$, we have that the scalar product of the vector with itself is strictly positive: $\langle \boldsymbol{u}, \boldsymbol{u} \rangle > 0$. Scalar products are symmetric, $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \langle \boldsymbol{v}, \boldsymbol{u} \rangle$, and enjoy the bilinearity property, i.e. linearity with respect to the second argument ($\langle \boldsymbol{u}, \lambda \boldsymbol{v} \rangle = \lambda \langle \boldsymbol{u}, \boldsymbol{v} \rangle$; $\langle \boldsymbol{u}, \boldsymbol{v}^1 + \boldsymbol{v}^2 \rangle = \langle \boldsymbol{u}, \boldsymbol{v}^1 \rangle + \langle \boldsymbol{u}, \boldsymbol{v}^2 \rangle$), and linearity also with respect to the first argument. Notice also that linearity implies $\langle \boldsymbol{u}, \boldsymbol{0} \rangle = 0$.

An inner product always defines a *norm* $\|\bullet\|$ (in this case it is called the Euclidean or L_2 norm) by $\|u\| \coloneqq \sqrt{\langle \boldsymbol{u}, \boldsymbol{u} \rangle} = \sqrt{\sum_{i=1}^{n} u_i^2}$. In turn a norm defines a metric (i.e. a notion of distance) by $d(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \|x - y\|$. It turns out that the inner product can be expressed as follows:

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \|\boldsymbol{u}\| \|\boldsymbol{v}\| \cos\theta, \tag{3.5}$$

where θ is the angle between the two vectors.

Taylor expansion. If the function f is differentiable at some point $\overline{\boldsymbol{x}}$, we have the multi-dimensional generalization of Eq. (2.6):

$$f(\boldsymbol{x}) = f(\overline{\boldsymbol{x}}) + \sum_{i=1}^{n} \left.\frac{\partial f}{\partial x_i}\right|_{\overline{\boldsymbol{x}}} (x_i - \overline{x}_i) + o(\|\boldsymbol{x} - \overline{\boldsymbol{x}}\|) \tag{3.6}$$

$$= f(\overline{\boldsymbol{x}}) + \langle \nabla f(\overline{\boldsymbol{x}}),\, \boldsymbol{x} - \overline{\boldsymbol{x}} \rangle + o(\|\boldsymbol{x} - \overline{\boldsymbol{x}}\|) \tag{3.7}$$

Last equation corresponds to a first order *Taylor expansion* of $f(\boldsymbol{x})$ around $\overline{\boldsymbol{x}}$.

Local minima. A *local minimize*r is a point $\boldsymbol{x}^*$ such that there is no other point with a lower cost within a ball with a certain strictly positive radius centered on it. Formally, a point $\boldsymbol{x}^*$ is a local minimizer if $\exists \epsilon > 0$ such that for any x with $\|\boldsymbol{x} - \boldsymbol{x}^*\| < \epsilon$ we have:

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*). \tag{3.8}$$

In general, a function $f(\boldsymbol{x})$ can have many local minimizers, and the lowest lying of them are called global minimizers. A very important family of functions, the family of convex functions that we shall introduce later in more details, enjoys the very nice property of having only global minimizers. Local minimizers of a differentiable functions have zero gradient, as the following important theorem states.

Theorem 1. *First Order Necessary Condition*

Given a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, any local minimizer $\boldsymbol{x}^*$ of f satisfies the "stationarity condition"

$$\nabla f(\boldsymbol{x}^*) = 0 \tag{3.9}$$

Proof. If by contradiction we set $\nabla f(\boldsymbol{x}^*) \neq 0$, then $\langle \nabla f(\boldsymbol{x}^*), \nabla f(\boldsymbol{x}^*) \rangle > 0$. Fix $\epsilon > 0$ and set $\boldsymbol{p} = -\nabla f(\boldsymbol{x}^*)$, then Eq. (3.7) gives

$$f(\boldsymbol{x}^* + \epsilon \boldsymbol{p}) = f(\boldsymbol{x}^*) + \epsilon \langle \nabla f(\boldsymbol{x}^*), \boldsymbol{p} \rangle + o(\epsilon) \tag{3.10}$$

with $\langle \nabla f(\boldsymbol{x}^*), \boldsymbol{p} \rangle < 0$. As a consequence, $f(\boldsymbol{x}^* + \epsilon \boldsymbol{p}) < f(\boldsymbol{x}^*)$ for ϵ small enough. In order to see this, consider the following. By definition of the small$-o$ notation, there is a $\bar{\epsilon} > 0$ such that for any ϵ, $0 < \epsilon < \bar{\epsilon}$, we have:

$$|o(\epsilon)| \leq -\frac{1}{2} \epsilon \langle \nabla f(\boldsymbol{x}^*), \boldsymbol{p} \rangle. \tag{3.11}$$

Therefore

$$\begin{aligned} f(\boldsymbol{x}^* + \epsilon \boldsymbol{p}) &= f(\boldsymbol{x}^*) + \epsilon \langle \nabla f(\boldsymbol{x}^*), \boldsymbol{p} \rangle + o(\epsilon) && (3.12) \\ &\leq f(\boldsymbol{x}^*) + \epsilon \langle \nabla f(\boldsymbol{x}^*), \boldsymbol{p} \rangle - \frac{1}{2} \epsilon \langle \nabla f(\boldsymbol{x}^*), \boldsymbol{p} \rangle && (3.13) \\ &= f(\boldsymbol{x}^*) - \frac{1}{2} \epsilon \langle \nabla f(\boldsymbol{x}^*), \boldsymbol{p} \rangle && (3.14) \end{aligned}$$

which leads to $f(\boldsymbol{x}^* + \epsilon \boldsymbol{p}) < f(\boldsymbol{x}^*)$. As a consequence $\boldsymbol{x}^*$ cannot be a local minimizer of f. □

Descent condition. The proof of the above theorem relies on the fact that, when we have a non zero gradient $\nabla f(\boldsymbol{x})$, it is always possible to choose a direction $\boldsymbol{p}$ such that we are sure to be able to decrease the cost if the step in that direction is small enough. Therefore, at any position $\boldsymbol{x}$, we call *descent direction* a vector $\boldsymbol{p}$ that satisfies

$$\langle \nabla f(\boldsymbol{x}), \boldsymbol{p} \rangle < \mathbf{0}. \tag{3.15}$$

3.2 Contour plots

In this paragraph we introduce the notion of *contour plot*.

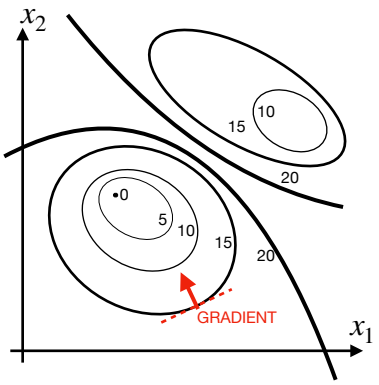

Figure 3.1: An example of a contour plot.

A contour plot is a graphical technique for representing surfaces in 3d space defined by a function $f : \mathbb{R}^2 \to \mathbb{R}$, $z = f(x_1, x_2)$. This is done by plotting constant z slices, called contours, on a 2-dimensional plane (Fig.3.1). This means that, given a certain z_0, the set of points $(x_1, x_2) \in \mathbb{R}^2$ such that $f(x_1, x_2) = z_0$ are drawn. For continuous functions, this corresponds to drawing one or multiple lines for the contour level z_0. The same can be done within the same plot for different levels z_1, $z_2, \ldots$

Using the Taylor expansion of f it is easy to see that **the gradient has to be orthogonal to a contour line**. In fact if $\boldsymbol{x} = (x_1, x_2)$ is on a contour line, and $\boldsymbol{x}+\boldsymbol{p}$ is another point on it, by definition of the contour line we have $f(\boldsymbol{x}) = f(\boldsymbol{x} + \boldsymbol{p})$. But the Taylor expansion also gives

$$f(\boldsymbol{x} + \boldsymbol{p}) = f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{p} \rangle + o(\|\boldsymbol{p}\|). \tag{3.16}$$

Last equation, along with $f(\boldsymbol{x}) = f(\boldsymbol{x} + \boldsymbol{p})$, implies that if we decrease $\|p\|$ while staying on the contour line (this is feasible since the contour line is continuous) we have $\langle \nabla f(\boldsymbol{x}), \boldsymbol{p} \rangle = o(\|\boldsymbol{p}\|)$, which can be rewritten as

$$\langle \nabla f(\boldsymbol{x}), \frac{\boldsymbol{p}}{\|\boldsymbol{p}\|} \rangle = o(1). \tag{3.17}$$

Therefore, since $p/\|p\|$ becomes the unit vector tangent to the contour line, we proved that $\nabla f(x)$ has to be orthogonal to the contour line. This argument can be easily generalized to n dimensional spaces, where equal $f : \mathbb{R}^n \to \mathbb{R}$ contours are hypersurfaces of dimension $n - 1$ and the gradient computed in any point on an hypersurface is orthogonal to the hypersurface itself.

3.3 Quadratic functions

For convenience, from now on we drop the bold notation for vectors. Consider a quadratic function in n dimensions:

$$f(x) = \frac{1}{2} \sum_{i,j=1}^{n} x_i A_{ij} x_j + \sum_{i=1}^{n} b_i x_i + c \tag{3.18}$$

We will set $c = 0$, since a constant term does not change the result of minimization. Also, we assume the matrix A to be positive definite. A square symmetric matrix A is called ***positive definite*** if $p^T A p > 0$ for any $p \neq 0$, and *positive semi-definite* if $p^T A p \geq 0$ for any $p \neq 0$. Being positive definite corresponds to have only strictly positive eigenvalues. Therefore, the matrix A can be safely inverted, i.e. we can compute A^{-1} without the fear of zero eigenvalues. Also, if A is positive definite thequadratic function is convex and has a unique stationary point, the (global) minimum. We write more compactly the quadratic form using vector notation as

$$f(x) = \frac{1}{2} x^T A x + b^T x \tag{3.19}$$

It is easy to check, by making the summation in previous equation explicit and computing each partial derivative (do it as an exercise!), that the gradient reads

$$\nabla f(x) = Ax + b \tag{3.20}$$

We can find the minimum by imposing the stationarity condition $\nabla f(x^*) = 0$:

$$Ax^* + b = 0 \;\Rightarrow\; x^* = -A^{-1} b \tag{3.21}$$

We thus have an analytic expression for the unique minimum.

We make some further considerations that will become useful for the understanding of Newton's method in later paragraphs. First, we need to introduce the concept of the **Hessian matrix**. The Hessian is the generalization to many dimension of the second derivative, in the same way as the gradient is the generalization of the first derivative. If a function f can be differentiated twice, we can define the $n \times n$ *Hessian matrix* $\nabla^2 f(x)$, as the matrix whose components are given by:

$$\left(\nabla^2 f(x)\right)_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}(x). \tag{3.22}$$

The Hessian is a symmetric matrix due to Schwarz's theorem (under some weak assumptions). In our quadratic case, we can compute the partial derivatives of each component of the gradient (3.20) and obtain

$$\nabla^2 f(x) = A. \tag{3.23}$$

We see that in the quadratic case the Hessian does not depend on the position: it is a constant matrix. Now consider the following problem: we start from some arbitrary position x, and consider the step p that would lead to the minimum $x^* = -A^{-1}b$. This step can be written as

$$p = -A^{-1}b - x \tag{3.24}$$
$$= -A^{-1}(b + Ax). \tag{3.25}$$

We find that in the quadratic case the optimal displacement, i.e. the one that leads to the minimum in one step, can be expressed in terms of the gradient and of the Hessian:

$$p = -\left(\nabla^2 f(x)\right)^{-1} \nabla f(x) \tag{3.26}$$
$$x^* = x + p. \tag{3.27}$$

Therefore, starting from any position we can reach the minimum in one step, if we choose the step as the right combination of the Hessian and the gradient. This is not true in general, but this rule (called Newton method, as we shall see later) in practice leads to much faster converge rate than the gradient only method. This fact is justified by the Taylor expansion, which tells us that any smooth function is locally quadratic, as we will see in the next paragraph.

3.4 Second order Taylor expansion and necessary condition

If a function f can be continuously differentiated twice, the *second order Taylor expansion* around a point $\overline{x}$ reads

$$f(x) = f(\overline{x}) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(\overline{x})\,(x_i - \overline{x}_i) + \frac{1}{2}\sum_{i,j=1}^{n} \frac{\partial^2 f}{\partial x_i \partial x_j}(\overline{x})\,(x_i - \overline{x}_i)(x_j - \overline{x}_j) + o(\|x - \overline{x}\|^2) \tag{3.28}$$

$$= f(\overline{x}) + \langle \nabla f(\overline{x}),\, x - \overline{x}\rangle + \frac{1}{2}\langle \nabla^2 f(\overline{x})\,(x - \overline{x}),\, x - \overline{x}\rangle + o(\|x - \overline{x}\|^2). \tag{3.29}$$

Using vector notation, we can write equivalently

$$f(x) = f(\overline{x}) + \nabla f(\overline{x})^T (x - \overline{x}) + \frac{1}{2}(x - \overline{x})^T\, \nabla^2 f(\overline{x})\,(x - \overline{x}) + o(\|x - \overline{x}\|^2) \tag{3.30}$$

The Hessian contains the information about the local curvature of the function $f(x)$. Last equation shows that any function smooth enough can be locally approximated by a quadratic function. Sometimes, it may be convenient to express Eq. (3.30) in a slightly different form, which makes more clear that we are analyzing the function in terms of a small displacement $p = x - \overline{x}$:

$$f(\overline{x} + p) = f(\overline{x}) + \nabla f(\overline{x})^T p + \frac{1}{2} p^T\, \nabla^2 f(\overline{x})\, p + o(\|p\|^2). \tag{3.31}$$

The minima of twice differentiable functions are characterized by having a positive semi-definite Hessian, as the following theorem states.

Theorem 2. *Second Order Necessary Condition*

Given a twice continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, for any local minimizer x^* of f, we have that $\nabla f(x^*) = 0$ and that $\nabla^2 f(x^*)$ is positive semi-definite.

Proof. By Theorem 1 we have $\nabla f(x^*) = 0$. If by contradiction we assume $\nabla^2 f(x^*)$ not positive semi-definite, then there is a $p \neq \mathbf{0}$ such that $p^T \nabla^2 f(x^*)\, p < 0$. Fix $\epsilon > 0$, then Eq. (3.30) gives

$$f(x^* + \epsilon p) = f(x^*) + \epsilon^2 \frac{1}{2} p^T \nabla^2 f(x^*)\, p + o(\epsilon^2). \tag{3.32}$$

As a consequence, $f(x^* + \epsilon p) < f(x^*)$ for ϵ small enough, therefore x^* cannot be a local minimizer of f. □

3.5 Line search procedures

Back to our minimization problem: given a cost/objective/energy function $f : \mathbb{R}^n \to \mathbb{R}$, find

$$\min_{x \in \mathbb{R}^n} \; f(x). \tag{3.33}$$

We want to devise a numerical algorithm to perform the minimization when the problem cannot be solved analytically (which is almost always the case, the quadratic cost being an exception). The first order Taylor expansion around a certain x,

$$f(x+p) = f(x) + \langle \nabla f(x), p \rangle + o(\|p\|), \tag{3.34}$$

tells us that as long as the gradient is non-zero and we select a displacement p *small enough* and *partially anti-aligned* to the gradient, we can decrease the value of our objective function going from x to $x+p$. Therefore it is convenient to introduce the notion of descent direction. A ***descent direction*** p at position x is a direction which is partially anti-aligned to the gradient, that is

$$\langle p, \nabla f(x) \rangle < 0. \tag{3.35}$$

We will discuss many different iterative procedures that produce a sequence of estimates $x^0, x^1, \ldots, x^k, \ldots$, possibly converging to a local minimizer x^*.

At each step k, in order to produce the next iterate x^{k+1}, we will make use of the zeroth, first and second order information (when available) collected at x^k, which means exploiting the function value, its gradient and its Hessian at x^k. In some methods (momentum and quasi-newton), we will use the same type of information but also keeping some internal state that is computed integrating the information over the past trajectory.

We will use the shorthands $f_k \coloneqq f(x^k)$, $\nabla f_k \coloneqq \nabla f(x^k)$ and $\nabla^2 f_k \coloneqq \nabla^2 f(x^k)$. It is conceptually useful to decompose the displacement vector $\Delta^k = x^{k+1} - x^k$ in two parts, $\Delta^k = \alpha^k p^k$, where the vector p^k is a descent direction (and not necesseraly a unit vector) and $\alpha^k > 0$ is a scalar called **step-size** or learning rate. In this way, once we choose a descent direction p^k, we can consider the following *one-dimensional problem*, which is the restriction of our original problem to a one-dimensional slice (i.e. *a line*):

$$\min_{\alpha \geq 0} \; f(x^k + \alpha p^k). \tag{3.36}$$

Solving a one-dimensional problem is much easier than solving a high-dimensional one, nonetheless it could be not worthy to perform exact minimization over α, since

the true minimum in the ambient n-dimensional space may well be very far from the one-dimensional submanifold (the line) we are considering at the moment.

If $\nabla f_k \neq 0$, we have the guarantee that if we choose α small enough we can go down in cost. Therefore, even if we don't want to (and don't need to) solve the one-dimensional problem exactly, we may look for some "safe" range where we can choose α. This is the approach of *line search methods* to optimization problems. We include the ample class of line search procedures in the following generic algorithm:

Algorithm 2 General Line Search Procedure

$x \leftarrow x^0$
for $k \leftarrow 1, ..., k_{MAX}$ **do**
  Choose descent direction p^k
  Choose step-size α^k
  $x \leftarrow x + \alpha^k p^k$
end for
return x

Choosing the direction p^k and choosing the step size α^k can be considered as partially decoupled problems, although in order to have convergence guarantees and to achieve optimal speed, the rule for choosing the step-size should be carefully tuned based on the method used for choosing the direction p^k. In any case, many possible options exist for the choice of p^k and α^k, with different trade-offs in terms of computational complexity and convergence rate. We will explore some of these options in the following paragraphs.

3.6 Selecting the descent direction p^k

Many commonly used schemes for choosing the descent direction p^k can be written in the form

$$p^k = -B_k^{-1} \nabla f_k, \tag{3.37}$$

for some matrix B_k. This form is general enough to include the following three popular algorithms.

Steepest Descent. Also known as Gradient Descent, this is the simplest algorithm and corresponds to setting $B_k = I$ (identity matrix). With this choice we have

$$p^k = -\nabla f_k \tag{3.38}$$

and the update rule is given by

$$x^{k+1} = x^k - \alpha^k \nabla f_k$$

The algorithm is very easy to implement, and just requires the evaluation of the gradient. It has the slowest convergest rate among the 3 though.

Newton Method. The Newton method is provably (in strictly convex settings) faster in terms of number of iterations than gradient descent, since it uses the second order information contained in the Hessian. For this method we have $B_k = \nabla^2 f_k$, therefore

$$p^k = -\left(\nabla^2 f_k\right)^{-1} \nabla f_k. \tag{3.39}$$

The Newton method is motivated by the study of the quadratic function case and the fact that any smooth function is locally quadratic.

The Newton method is typically computationally expensive, since we need to compute n^2 second derivatives at each step, therefore it is rarely used in large scale applications. Nonetheless, it has the fastest convergence rate among the 3 algorithms.

Exercise: show that p^k given by Newton's method is a descent direction if the Hessian is positive definite.

Quasi-Newton methods. In Quasi-Newton methods, one tries to compute an approximation B_k as close as possible to the true Hessian $\nabla^2 f_k$, without computing second order derivatives but using first order information from the past trajectory.

According to these methods, at each step k of the algorithm there is a **quadratic model** that is supposed to be a good approximation of the function $f(x)$ in a neighborhood of x^k. Of course, the best possible local quadratic approximation is the second Taylor expansion of $f(x)$ at x^k, but since we don't want to compute the Hessian, we will have some matrix B_k ideally approximating it. In this way our local model, that we call m_k, is roughly close to the Taylor expansion. Therefore, we assume that at step k the following approximation of f holds for some (possibly small) displacement p:

$$f(x^k + p) \approx m_k(p) = f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p.$$

We see that the only term that we have to fix in our local model is the "quasi-Hessian" B_k. This is done by starting with some guess B_0 at time 0 (typically $B_0 = I$,

the identity matrix) and then iteratively updating B_k according to some algorithmic specific rules. Once we have the model $m_k(p)$, we select the descent direction by doing exact minimization on the model (we can do it since it is quadratic). This leads to a new position $x^{k+1} = x^k + \alpha^k p^k$ once also the step size is considered (usually we set $\alpha^k < 1$ since we are not fully confident in our local model). Once we have x^{k+1}, we have to determine the new local model m_{k+1}, again in the form:

$$f(x^{k+1} + p) \approx m_{k+1}(p) = f_{k+1} + \nabla f_{k+1}^T p + \frac{1}{2} p^T B_{k+1} p. \tag{3.40}$$

According to some criterion the Hessian approximation at the new position is computed as an update to the old Hessian approximation:

$$B_{k+1} = B_k + U_k. \tag{3.41}$$

The update matrix U_k is often chosen as a **low rank matrix**: rank 1 for the symmetric update we shall discuss in Section 3.9, and rank 2 for the BFGS method, one of the most robust and most commonly used quasi-Newton methods.

3.7 Selecting the step size α^k

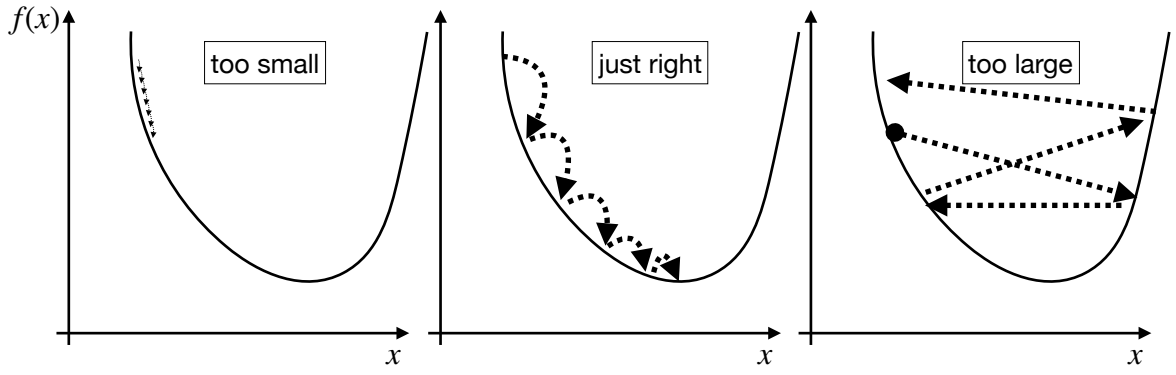

Figure 3.2: Choosing the learning rate in 1D. A learning rate too small would be too slow to find the solution, while a learning rate too large would cause big fluctuations and never converge to the solution.

Once at step k a descent direction p^k is selected, the ideal step size α^k would be the one that gives the solution of the reduced one-dimensional problem in one step, that

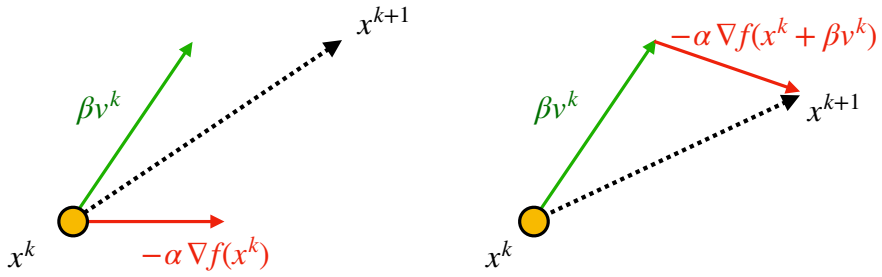

Figure 3.3: Standard Momentum compared to Nesterov Momentum.

is

$$\alpha^k = \arg\min_{\alpha \geq 0} f(x^k + \alpha p^k). \tag{3.42}$$

As we already argued, solving this 1D optimization problem is typically uselessly expensive, therefore in practice one of the following rules are commonly used:

- **Fixed step size.** $\alpha^k = \alpha \quad \forall\ k$ (see Fig.3.2).
- **Decreasing step size.** E.g. $\alpha^k = \frac{\alpha^0}{k}$ or some other decreasing function of the step k. Common choice are power law decay, $\alpha^k = \frac{\alpha^0}{k^\gamma}$ with $\gamma > 0$, and exponential decay, $\alpha^k = \rho^k \alpha^0$ with $\rho < 1$.
- **Dependent on some conditions.** There are some (complicated) adaptive rules for choosing α^k that come with convergence guarantees (see Wolfe conditions on Nocedal's textbook). In practice, these rules are not commonly used in large scale machine learning problems since they come with a large additional complexity cost.

3.8 Momentum methods

The general strategy used in momentum methods is to add an inertial term, called ***momentum*** (in physics momentum is defined as mass × velocity). This momentum term makes it possible to mantain some inertia in almost flat regions, where the gradient is very small but the minimum is still far away. Small gradient would cause a slowdown of e.g. gradient descent. Momentum methods instead keep making some progress due to the previously acquired velocity. This is a little departure from the

general line search departure, since the direction taken in momentum methods is not necessarily a descent one. Momentum methods come in many flavors.

- **Standard Momentum.** In classical mechanics, the motion of a body of mass m subject to a force $\boldsymbol{F}$ is given by Newton's second law:

$$\begin{cases} m\frac{d\boldsymbol{v}}{dt} = \boldsymbol{F} \\ \frac{d\boldsymbol{x}}{dt} = \boldsymbol{v} \end{cases}$$

  That is, a force produces a change in velocity and a velocity produces a change in position. In the optimization context, the force $\boldsymbol{F}$ is the gradient of our cost function. We can adapt the natural law of motion to our discrete time iterative procedure as follows. Let's call $\beta \in [0, 1]$ a parameter which is associated to inertia. A popular choice is $\beta = 0.9$. We start with a velocity $v^k = \mathbf{0}$, and in the following steps we apply the recursion rule

$$\begin{cases} v^{k+1} = \beta v^k - \alpha^k \nabla f_k \\ x^{k+1} = x^k + v^k \end{cases}$$

- **Nesterov Momentum.** Nesterov Momentum is just a small variation of the standard momentum method. It often proves to be more convenient in terms of performance. In Nesterov momentum, the velocity is calculated based on the gradient computed at the current position shifted to the approximated direction towards which we are headed (see Fig. (3.3)). The update rule reads:

$$\begin{cases} v^{k+1} = \beta v^k - \eta \nabla f(x^k + \beta v^k), \\ x^{k+1} = x^k + v^k. \end{cases}$$

  Nesterov's update rule can take different forms. Also, in the optimization literature, it is known as accelerated gradient method or fast gradient method.

- **Adam, AdaGrad, RmsProp, ...** There are many heuristic optimization methods that, while incorporating the momentum idea, try also to compute a local approximation of the local curvature, as in quasi-Newton methods, and to average out noisy fluctuations of the gradient. In these methods, the approximation to the Hessian always takes a diagonal form. See `http://ruder.io/optimizing-gradient-descent/` for a comparison of these methods.

3.9 Symmetric Rank-1 update

This is just a rewrite of Chap. 6.2 of Nocedal's textbook.

Among the quasi-Newton methods, one of the simplest is the Symmetric Rank-1 update rule. It constructs an approximation B_k of the true Hessian $\nabla^2 f_k$ using only first order information as follows:

1. Choose some $\boldsymbol{v} \in \mathbb{R}^n$ and $\sigma \in \{-1, 1\}$ optimal for the problem ;
2. $B_0 = I$;
3. At each step, update B with a rank-1 perturbation: $B_{k+1} = B_k + \sigma \boldsymbol{v} \cdot \boldsymbol{v}^T$. Note that $\boldsymbol{v} \cdot \boldsymbol{v}^T$ is a $n \times n$ matrix with rank 1.

In particular, suppose to construct a local model m_{k+1} such that

$$f(\boldsymbol{x}^{k+1} + \boldsymbol{p}) \approx m_{k+1}(\boldsymbol{p}) = f(\boldsymbol{x}^{k+1}) + (\nabla f(\boldsymbol{x}^{k+1}))^T \boldsymbol{p} + \frac{1}{2} \boldsymbol{p}^T B_{k+1} \boldsymbol{p} \tag{3.43}$$

For $m_{k+1}(\boldsymbol{p})$ to be a good approximation in the neighborhood of $\boldsymbol{x}^{k+1}$, we impose the following requirements:

1. $f(\boldsymbol{x}^{k+1}) = m_{k+1}(\boldsymbol{0})$
2. $\nabla f(\boldsymbol{x}^{k+1}) = \nabla m_{k+1}(\boldsymbol{0})$
3. $\nabla m_{k+1}(-\alpha_k \boldsymbol{p}^k) = \nabla f(\boldsymbol{x}^k)$

While the first two conditions are met by construction by the class of models in Eq. (3.43), the last one constrains B_{k+1} as follows:

$$\begin{aligned} \nabla m_{k+1}(-\alpha_k \boldsymbol{p}^k) = \nabla f(\boldsymbol{x}^{k+1}) + B_{k+1}(-\alpha_k \boldsymbol{p}^k) = \nabla f(\boldsymbol{x}^k) \\ \Rightarrow \nabla f(\boldsymbol{x}^{k+1}) - \nabla f(\boldsymbol{x}^k) = B_{k+1}(\alpha_k \boldsymbol{p}^k) \end{aligned}$$

Let $\nabla f(\boldsymbol{x}^{k+1}) - \nabla f(\boldsymbol{x}^k) = \boldsymbol{y}^k$ and $\alpha_k \boldsymbol{p}^k = \boldsymbol{s}^k$, then

$$\begin{aligned} \boldsymbol{y}^k = B_{k+1} \boldsymbol{s}^k = (B_k + \sigma \boldsymbol{v} \boldsymbol{v}^T) \boldsymbol{s}^k = B_k \boldsymbol{s}^k + \sigma \boldsymbol{v}^T \boldsymbol{s}^k \boldsymbol{v} \\ \Rightarrow \boldsymbol{y}^k - B_k \boldsymbol{s}^k = \sigma \boldsymbol{v}^T \boldsymbol{s}^k \boldsymbol{v} \end{aligned}$$

Let $\boldsymbol{v} = \delta(\boldsymbol{y}^k - B_k \boldsymbol{s}^k)$, then

$$\begin{aligned} \boldsymbol{y}^k - B_k \boldsymbol{s}^k = \sigma \delta^2 (\boldsymbol{y}^k - B_k \boldsymbol{s}^k)^T \boldsymbol{s}^k (\boldsymbol{y}^k - B_k \boldsymbol{s}^k) \\ \Rightarrow \sigma \delta^2 (\boldsymbol{y}^k - B_k \boldsymbol{s}^k)^T \boldsymbol{s}^k = 1 \end{aligned}$$

which leads to

$$\Rightarrow \begin{cases} \delta = |(\boldsymbol{y}^k - B_k \boldsymbol{s}^k)^T \boldsymbol{s}^k|^{-\frac{1}{2}} \\ \sigma = \text{sign}((\boldsymbol{y}^k - B_k \boldsymbol{s}^k)^T \boldsymbol{s}^k) \\ B_{k+1} = B_k + \sigma \boldsymbol{v}\boldsymbol{v}^T \end{cases}$$

and finally to

$$B_{k+1} = B_k + \sigma \frac{(\boldsymbol{y}^k - B_k \boldsymbol{s}^k)(\boldsymbol{y}^k - B_k \boldsymbol{s}^k)^T}{(\boldsymbol{y}^k - B_k \boldsymbol{s}^k)^T \boldsymbol{s}^k}$$

A Convex functions

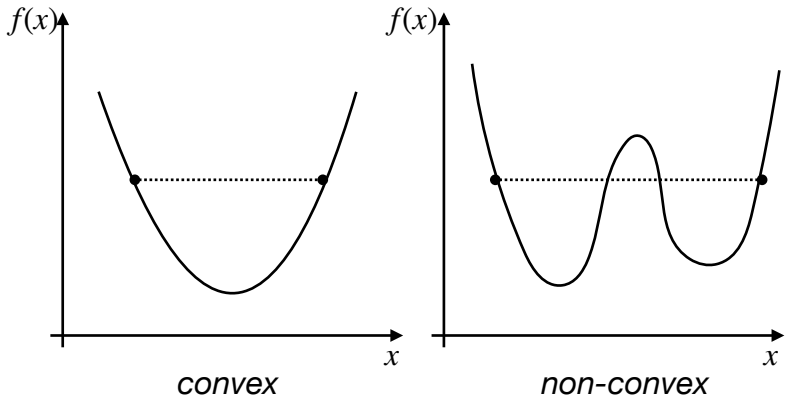

Figure A.1: Convex and non-convex functions.

Here we explore a family of functions, the convex functions, which are object of investigation of a large branch of optimization theory, called convex optimization. These functions are very import because of two very desirable properties from the view point of optimization: 1) each minimizer is a global minimizer; 2) along any direction, the slope (i.e. the directional derivative) is a monotonous function, either increasing or decreasing. Thanks to this structure, essentially it is sufficient to take small enough (but not too small) steps in the direction of the gradient to provably get close to a global minimum. Powerful and general algorithms, such as the interior point method, are available to solve the constrained convex optimization case. To define the family of convex functions, we first have to introduce the concept of convex set.

A ***convex set*** $S \subseteq \mathbb{R}^n$ is a set such that for any two points $x, y \in S$ the whole segment with extremities in x and y is contained in S, that is for any $c \in [0, 1]$ we have

$$cx + (1 - c)y \in S. \tag{A.1}$$

A ***convex function*** f is a function defined on a convex set S and such that for any two points $x, y \in S$ the segment connecting the points $(x, f(x))$ and the point $(y, f(y))$ stays above the graph of the function, that is for any $c \in [0, 1]$ we have

$$cf(x) + (1 - c)f(y) \geq f(cx + (1 - c)y). \tag{A.2}$$

If the above inequality holds strictly, i.e. if we can replace $\geq$ with $>$, we say that the function f is *strictly convex*. It is easy to see that for convex functions, any minimizer is a global minimizer (prove it by contradiction!). This is very important from the view point of optimization, since we have the guarantee that using an iterative procedure we don't get stuck in a local minimum with high cost. Moreover, for strictly convex function, there is a unique minimum. For convex, but non-strictly convex function, it may be the case that an infinity of points lying on a *flat* region are all global minima. An important class of convex functions is the one of the form $\frac{1}{2}x^T Ax + b^T x + c$ with positive semi-definite A. These are called quadratic functions (for obvious reasons).

B Convergence rate

The performance of a linear search implementation is defined in terms of convergence rate. Given a succession $\{x^k\}_k$ with limit x^*, we say that the convergence of the sequence is q-linear, for some $q \geq 1$, if for some $c \in (0, 1)$ and for some $k_0 > 0$ we have

$$||x^{k+1} - x^k|| \leq c||x^* - x^k||^q \qquad \text{for } k \geq k_0. \tag{B.1}$$

In particular, for different choices of descent direction p^k and for some appropriate step sizes α^k, there are different convergence rates for the algorithm:

- Gradient descent: $q = 1 \ \rightarrow$ linear rate
- Quasi-Newton methods: $1 \leq q \leq 2 \ \rightarrow$ super-linear rate
- Newton method: $q = 2 \ \rightarrow$ quadratic rate

It is easy to convince yourself, by iterative application of Eq. (B.1), that for the linear rate ($q = 1$) we have

$$\|x^* - x^k\| \leq \ Ac^k \tag{B.2}$$

so that the distance from the minimum is exponentially shrinking in time. Super linear rates have even faster convergence.