Optimization and Approximation

Elisa Riccietti[1]

[1]Reference book: Numerical Optimization, Nocedal and Wright, Springer

Contents

Part I

I part: nonlinear optimization

Chapter 1

Introduction

The main aim in numerical optimization is to develop reliable algorithms to optimize (that is minimize or maximize) a function. A general problem in optimization can be stated as follows.

Let A be a subset of some space and let

$$f: \quad A \quad \longrightarrow \quad \mathbb{R}^t$$

with $t \in \mathbb{N}$, we want to solve

$$\min_{x \in A} f(x),$$

where f is called the *objective function*. Generally $t = 1$, if $t > 1$ the problem is a *multi-objective problem*, that is we want to find a point that provides the best compromise among the different functions.

In this course we will focus on the following setting, that is the most common one.

Let A be an open set of $\mathbb{R}^n$ and let

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

be a nonlinear and differentiable function.

We consider first *unconstrained optimization problems*, that is problems of the form:

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}),$$

and then constrained problems, in which $\mathbf{x}$ is not free to vary in $\mathbb{R}^n$ but needs to satisfy some (possibly nonlinear) relations called constrains:

$$\min_{\mathbf{x} \in \mathbb{R}^n, h(\mathbf{x}) \geq 0, g(\mathbf{x}) = 0} f(\mathbf{x}),$$

with $h : \mathbb{R}^n \to \mathbb{R}^m$, $g : \mathbb{R} \to \mathbb{R}^p$, with $p, m \in \mathbb{N}$.

Examples Optimization problems are ubiquitous in real-life:

- Production: determine how many products of each type to assemble from certain parts to maximize profits while not exceeding available parts inventory;

  allocate production of a product to different machines, with different capacities, startup cost and operating cost, to meet production target at minimum cost;

determine which raw materials from different sources to blend to produce a substance with certain desired qualities at minimum cost

- Distribution: determine how many products to ship from each factory to each warehouse, or from each factory to each warehouse and direct to each end customer, to minimize shipping cost while meeting warehouse demands and not exceeding factory supplies
- Loading: Decide which sizes or types of products to load into a vehicle, given its size limits, to best meet demand or to minimize wasted space
- Flights: determine which aircraft to fly on each route, and the sequence of segments flown by each aircraft;

  assign crews to different airline flight segments to minimize total cost while ensuring that a crew "rotation" begins and ends in the same city or to maximize employees happiness;

  for different classes of tickets, determine how many seats to sell or hold back as flight date approaches
- Gas: with forecasted but uncertain demand for gas or electricity, determine which contracts to buy, and how much gas or electricity to store at different times while not exceeding the capacity
- Agriculture: given forecasted crop prices and growing conditions, determine how much of each crop to plant;
- Diet: given the nutritional requirements for feed animals or humans and the price of available feeds, find the blend of feed ingredients that will minimize total cost
- Electric Power: given forecasted demand by period and operating cost for each generator, determine which generators should be run in each time interval
- Machine learning: training of neural networks

We will see more detailed examples in the following lessons.

1.0.1 What do we look for?

Ideally, we would like to obtain the *global minimizer* of the function, that is a point in which the function reaches the lowest possible value (assuming f is bounded from below). In practice it is really hard to reach such a point, and the usual methods used in optimization rather reach a *stationary point*, a point in which the gradient is zero. In most cases this stationary point will be a *local minimizer*, a point such that in its neighbourhood the function does not increase, but it is usually not possible to exclude it to be a *saddle point*.

Definition 1.0.1. *Let* $\mathbf{x}^* \in A$.

- $\mathbf{x}^*$ *is a local minimizer or a local minimum point for* f *if it exists a neighbourhood* Ω *of* $\mathbf{x}^*$ *such that*

  $$f(\mathbf{x}^*) \leq f(\mathbf{x}) \qquad \forall \mathbf{x} \in \Omega.$$

  $f(x^*)$ *is a local minimum of* f.
- $\mathbf{x}^*$ *is a global minimizer or a global minimum point for* f *if*

  $$f(\mathbf{x}^*) \leq f(\mathbf{x}) \qquad \forall \mathbf{x} \in A.$$

  $f(x^*)$ *is a global minimum of* f.

- $\mathbf{x}^*$ *is a stationary point for* f *if*

$$\nabla f(\mathbf{x}^*) = 0.$$

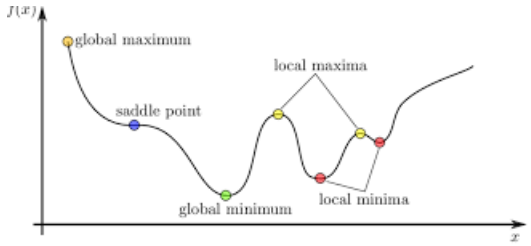


Figure 1.1: Stationary points

1.1 Essential information for the optimization: derivatives

Most optimization methods require the access to derivative information to reach a minimum:

1. If f is differentiable in $\mathbf{x}$ (i.e. if there exist all the partial derivatives of f in $\mathbf{x}$), the *gradient* of f in $\mathbf{x}$ is $\nabla f(\mathbf{x}) \in \mathbb{R}^n$:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \dfrac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \dfrac{\partial f(\mathbf{x})}{\partial x_n} \end{pmatrix}.$$

2. If f is two times differentiable in $\mathbf{x}$, the *Hessian* matrix of f in x is $H(\mathbf{x}) \in \mathbb{R}^{n \times n}$

$$H(\mathbf{x}) = H_f(\mathbf{x}) = \begin{pmatrix} \dfrac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_1} & \cdots & \dfrac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \dfrac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \cdots & \dfrac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_n} \end{pmatrix} = \begin{pmatrix} \left(\nabla \dfrac{\partial f(\mathbf{x})}{\partial x_1}\right)^T \\ \vdots \\ \left(\nabla \dfrac{\partial f(\mathbf{x})}{\partial x_n}\right)^T \end{pmatrix}.$$

   If $f \in C^2(\mathbf{x})$ then $H(\mathbf{x})$ is a symmetric matrix.

Based on the information they use, the optimization methods can be divided into two large classes:

1. *first order methods*: they use only information coming from first order derivatives, i.e. the gradient. They are usually cheap, but slow in reaching a stationary point

2. *second order methods*: they use the gradient information and also the second-order informations contained in the Hessian matrix. They are rather expensive and memory demanding, but can reach a stationary point much faster.

Remark 1.1.1. *It exists also another class of optimization methods, called derivative free methods, because they do not use any derivative information. Not having any information of the decrease or decrease of the objective function, these are methods that explore the space along random directions, evaluate the objective function and move towards the point in which the function decreases. Examples are direct-search methods, particle swarm, genetic algorithms. They need however a lot of function evaluations and it is not suggested to use them if derivatives are available. Sometimes this is however not the case, for example when the function is the result of a black-box computation. In this case derivative-free optimization is the only possibility. We will not study these methods in this course.*

1.1.1 Approximate the function using derivatives: Taylor's theorem

Most optimization methods employs approximations of the nonlinear objective functions built with polynomials. The theoretical foundation comes from the Taylor's theorem. Let's consider first the simple case of a scalar function $f : \mathbb{R} \to \mathbb{R}$.

Theorem 1.1.1. *Taylor's theorem (scalar version) Let $k \geq 1$ be an integer and let the function $f : \mathbb{R} \to \mathbb{R}$ be k times differentiable at the point $a \in \mathbb{R}$. Then there exists a function $h_k : \mathbb{R} \to \mathbb{R}$ such that*

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k + h_k(x)(x-a)^k,$$

and

$$\lim_{x \to a} h_k(x) = 0.$$

This is called the Peano form of the remainder.

The polynomial appearing in Taylor's theorem is the k-th order Taylor polynomial:

$$P_k(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k = \sum_{i=0}^{k} \frac{f^{(i)}(a)}{i!}(x-a)^i$$

of the function f at the point a. The Taylor polynomial is the unique "asymptotic best fit" polynomial in the sense that if there exists a function h_k and a k-th order polynomial p such that

$$f(x) = p(x) + h_k(x)(x-a)^k, \quad \lim_{x \to a} h_k(x) = 0,$$

then $p = P_k$.

Taylor's theorem describes the asymptotic behavior of the remainder term

$$R_k(x) = f(x) - P_k(x),$$

which is the approximation error when approximating f with its Taylor polynomial. Using the little-o notation, the statement in Taylor's theorem reads as

$$R_k(x) = o(|x-a|^k), \quad x \to a.$$

Lemma 1.1.1. *Lagrange form of the remainder Let $f : \mathbb{R} \to \mathbb{R}$ be $k+1$ times differentiable on the open interval with $f(k)$ continuous on the closed interval between a and x. Then*

$$R_k(x) = \frac{f^{(k+1)}(\xi)}{(k+1)!}(x-a)^{k+1}$$

for some real number ξ between a and x.

If f is $(k+1)$-times continuously differentiable on an interval $I = (a-r, a+r)$ with some $r > 0$ and $|f^{(k+1)}(x)| \le M$ in I, it holds

$$|R_k(x)| \le M \frac{|x-a|^{k+1}}{(k+1)!} \le M \frac{r^{k+1}}{(k+1)!}$$

Exercise

- Find an approximation of the function $f(x) = e^x$ on the interval $[-1, 1]$ ensuring that the error in the approximation is no more than 10^{-5}.
- Plot the function in the interval and the Taylor polynomials of order 1 to 5. Observe how the approximation improves (cf. `https://en.wikipedia.org/wiki/Taylor_series#/media/File:Exp_series.gif` or also `https://en.wikipedia.org/wiki/Taylor_series#/media/File:Logarithm_GIF.gif`).

Solution: The k-th order Taylor polynomial of f at 0 and its remainder term in the Lagrange form are given by

$$P_k(x) = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^k}{k!}, \qquad R_k(x) = \frac{e^\xi}{(k+1)!} x^{k+1},$$

where ξ is some number between 0 and x. Since e^x is an increasing function, we can simply use $e^x \le e$ for $x \in [-1, 1]$ to estimate the remainder.

We then see that

$$|R_k(x)| \le \frac{e|x|^{k+1}}{(k+1)!} \le \frac{e}{(k+1)!}, \qquad -1 \le x \le 1,$$

so the required precision is certainly reached, when

$$\frac{e}{(k+1)!} < 10^{-5} \quad \Longleftrightarrow \quad e \cdot 10^5 < (k+1)! \quad \Longleftrightarrow \quad k \ge 9.$$

(It holds $9! = 362880$ and $10! = 3628800$.) As a conclusion, Taylor's theorem leads to the approximation

$$e^x = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^9}{9!} + R_9(x), \qquad |R_9(x)| < 10^{-5}, \qquad -1 \le x \le 1.$$

For instance, this approximation provides a decimal expression $e \sim 2.71828$, correct up to five decimal places.

1.1.2 Taylor's theorem in n dimensions

This can be generalized to functions defined in $\mathbb{R}^n$. Assume $f : \mathbb{R}^n \to \mathbb{R}$.

Taylor polynomials in $\mathbb{R}^n$:

1. *First-order Taylor polynomial*: the best linear approximation of f in a neighbourhood of x. Let $f \in C^1(A)$ and let $\mathbf{x}, \mathbf{x}+\mathbf{h} \in A$ with $\mathbf{h} \neq \mathbf{0}$, we define

$$T_1(\mathbf{x}, \mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h}. \tag{1.1}$$

2. *Second-order Taylor polynomial* the best quadratic approximation of f in a neighbourhood of x. Let $f \in C^2(A)$ and $\mathbf{x}, \mathbf{x}+\mathbf{h} \in A$ with $\mathbf{h} \neq \mathbf{0}$, we define

$$T_2(\mathbf{x}, \mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T H(\mathbf{x}) \mathbf{h}. \tag{1.2}$$

Usually in $\mathbb{R}^n$ we stop at order two: higher order derivatives are difficult or expensive to evaluate and are not used in practice.

In some proofs we will use the following formula related to the Taylor polynomials:

1. *Zero-th-order Taylor formula* with Lagrange form of the remainder. Let $f \in C^1(A)$. Let $\mathbf{x}, \mathbf{x}+\mathbf{h} \in A$ with $\mathbf{h} \neq \mathbf{0}$ such that the segment $\{\mathbf{x}+t\mathbf{h} \mid t \in [0,1]\}$ whose endpoints are $\mathbf{x}$ and $\mathbf{x}+\mathbf{h}$ is contained in A. Then it exists $t \in (0,1)$, depending on $\mathbf{x}$ and $\mathbf{h}$, such that

$$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x}+t\mathbf{h})^T\mathbf{h}.$$

2. *First-order Taylor formula* with Lagrange form of the remainder. Let $f \in C^2(A)$. Let $\mathbf{x}, \mathbf{x}+\mathbf{h} \in A$ with $\mathbf{h} \neq \mathbf{0}$ such that the segment $\{\mathbf{x}+t\mathbf{h} \mid t \in [0,1]\}$ whose endpoints are $\mathbf{x}$ and $\mathbf{x}+\mathbf{h}$ is contained in A. Then it exists $t \in (0,1)$, depending on $\mathbf{x}$ and $\mathbf{h}$, such that

$$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T\mathbf{h} + \frac{1}{2}\mathbf{h}^T H(\mathbf{x}+t\mathbf{h})\mathbf{h}.$$

3. *Third-order Taylor formula* Let $f \in C^3(A)$. Let $\mathbf{x}, \mathbf{x}+\mathbf{h} \in A$ with $\mathbf{h} \neq \mathbf{0}$ such that the segment $\{\mathbf{x}+t\mathbf{h} \mid t \in [0,1]\}$ whose endpoints are $\mathbf{x}$ and $\mathbf{x}+\mathbf{h}$ is contained in A. Then it exists $t \in (0,1)$, depending on $\mathbf{x}$ and $\mathbf{h}$, such that

$$f(\mathbf{x}+\mathbf{h}) \sim f(\mathbf{x}) + \nabla f(\mathbf{x})^T\mathbf{h} + \frac{1}{2}\mathbf{h}^T H(\mathbf{x})\mathbf{h}$$

   the rest depends on the 3rd order tensor

1.2 How do we recognise a minimum?

In order to practically implement an algorithm to find a minimum, we need to have some conditions to be aware that we have reached the desired point. In this section we give necessary and sufficient conditions for a point to be a minimum point. These conditions can be first or second order, depending if they use only first order information (coming from first order derivatives, i.e. the gradient) or second order ones (coming from second order derivatives, i.e. the Hessian matrix).

1.2.1 Necessary conditions

Necessary conditions are conditions that need to be satisfied in order to have a minimum.

Theorem 1.2.1. *First order necessary condition*
Let $f \in C^1(\Omega)$ in a neighbourhood Ω of $\mathbf{x}^*$.

$$\mathbf{x}^* \text{ is a minimizer for } f \text{ (in } \Omega) \implies \nabla f(\mathbf{x}^*) = \mathbf{0}, \text{ i.e. } \mathbf{x}^* \text{ is a stationary point for } f.$$

Proof. We prove it by contradiction. Let us assume that $\nabla f(\mathbf{x}^*) \neq \mathbf{0}$. Let

$$\mathbf{p} = -\nabla f(\mathbf{x}^*)$$

the antigradient of f in $\mathbf{x}^*$, clearly $\mathbf{p} \neq \mathbf{0}$. The function

$$g(\mathbf{x}) = \nabla f(\mathbf{x})^T\mathbf{p}$$

is such that

$$g(\mathbf{x}^*) = \nabla f(\mathbf{x}^*)^T \mathbf{p} = -\nabla f(\mathbf{x}^*)^T \nabla f(\mathbf{x}^*) = -\|\nabla f(\mathbf{x}^*)\|^2 < 0.$$

Then, as $f \in C^1(\Omega)$ and so g is continuous in Ω, it will remain negative in a neighbourhood of x^*, i.e., it exists $T \in \mathbb{R}, T > 0$ such that $\forall t \in [0, T]$

$$0 > g(\mathbf{x}^* + t\mathbf{p}) = \nabla f(\mathbf{x}^* + t\mathbf{p})^T \mathbf{p}. \tag{1.3}$$

From (1.1), $\forall \tau \in (0, T)$, it exists $t \in (0, 1)$ such that

$$f(\mathbf{x}^* + \tau\mathbf{p}) = f(\mathbf{x}^*) + \nabla f(\mathbf{x}^* + \underbrace{t\tau}_{= t' \in (0,T)} \mathbf{p})^T \tau\mathbf{p} = f(\mathbf{x}^*) + \tau \underbrace{\nabla f(\mathbf{x}^* + t'\mathbf{p})^T \mathbf{p}}_{< 0 \text{ from (1.3)}} < f(\mathbf{x}^*).$$

Then $\mathbf{x}^*$ cannot be a minimum point for f, which leads us to a contradiction. □

This is just a necessary conditions, all minimizers are stationary points but not all stationary points are minimizers, they may be maximizers or saddle points.

Theorem 1.2.2. *Second order necessary condition*
Let $f \in C^2(\Omega)$ for a neighbourhood Ω of $\mathbf{x}^*$.

$$\mathbf{x}^* \textit{ is a minimum point for } f \textit{ (in } \Omega) \implies H(\mathbf{x}^*) \textit{ is positive semidefinite.}$$

Proof. We do the proof by contradiction. Let us assume that $H(\mathbf{x}^*)$ is not positive semidefinite, i.e. that it exists $\mathbf{p} \in \mathbb{R}^n, \mathbf{p} \neq \mathbf{0}$ such that $\mathbf{p}^T H(\mathbf{x}^*)\mathbf{p} < 0$. Let us define

$$g(\mathbf{x}) := \mathbf{p}^T H(\mathbf{x})\mathbf{p},$$

it holds $g(\mathbf{x}^*) < 0$. Then, as $f \in C^2(\Omega)$ and so g is continuous in Ω, it exists $T \in \mathbb{R}, T > 0$ such that $\forall t \in [0, T]$

$$0 > g(\mathbf{x}^* + t\mathbf{p}) = \mathbf{p}^T H(\mathbf{x}^* + t\mathbf{p})\mathbf{p}. \tag{1.4}$$

From (1.2), $\forall \tau \in (0, T)$, it exists $t \in (0, 1)$ such that

$$f(\mathbf{x}^* + \tau\mathbf{p}) = f(\mathbf{x}^*) + \underbrace{\nabla f(\mathbf{x}^*)^T}_{= 0 \text{ from Theorem 1.2.1}} \tau\mathbf{p} + \frac{1}{2}(\tau\mathbf{p})^T H(\mathbf{x}^* + \underbrace{t\tau}_{= t' \in (0,T)} \mathbf{p})\tau\mathbf{p} =$$

$$= f(\mathbf{x}^*) + \frac{1}{2}\tau^2 \underbrace{\mathbf{p}^T H(\mathbf{x}^* + t'\mathbf{p})\mathbf{p}}_{< 0 \text{ from (1.4)}} < f(\mathbf{x}^*).$$

Then $\mathbf{x}^*$ cannot be a minimum point for f, which leads us to a contradiction. □

1.2.2 Sufficient conditions

In the following theorem we show a sufficient condition: if this condition is satisfied, we are sure to have a minimum point. This is a second order condition: first order derivatives are not enough to establish a sufficient condition.

Theorem 1.2.3. *Sufficient second-order condition*
Let $f \in C^2(\Omega)$ for a neighbourhood Ω of $\mathbf{x}^*$.

$$\begin{cases} \nabla f(\mathbf{x}^*) = \mathbf{0} \\ H(\mathbf{x}^*) \textit{ is positive definite} \end{cases} \implies \mathbf{x}^* \textit{ is a minimum point for } f.$$

Proof. As $H(\mathbf{x}^*)$ is positive definite, it exists a neighbourhood $B = B_T(\mathbf{x}^*)$ of $\mathbf{x}^*$ such that $\forall \mathbf{x} \in B$ the matrix $H(\mathbf{x})$ remains positive definite. Then for every $\mathbf{p} \in \mathbb{R}^n$, $\forall \tau \in (0, T)$ it exists $t \in (0, 1)$ such that

$$f(\mathbf{x}^* + \tau\mathbf{p}) = f(\mathbf{x}^*) + \underbrace{\nabla f(\mathbf{x}^*)^T}_{=\mathbf{0}}\tau\mathbf{p} + \frac{1}{2}(\tau\mathbf{p})^T H(\mathbf{x}^* + \underbrace{t\tau}_{=t'\in(0,T)}\mathbf{p})\tau\mathbf{p} \quad =$$

$$= \quad f(\mathbf{x}^*) + \frac{1}{2}\tau^2 \underbrace{\mathbf{p}^T H(\underbrace{\mathbf{x}^* + t'\mathbf{p}}_{\in B})\mathbf{p}}_{>0} \quad > \quad f(\mathbf{x}^*),$$

i.e. $f(\mathbf{x}^*)$ is the minimum value taken by f in B, so $\mathbf{x}^*$ is a minimum point for f. □

Remark 1.2.1. *It is in general expensive to establish if $H(\mathbf{x}^*)$ is positive definite, as this requires the computation of the eigenvalues of the matrix. This condition is then usually not employed.*

1.3 How do we find the minimum

In unconstrained optimization *iterative algorithms* are usually used to find a minimizer of f, that is algorithms such that, starting from an initial guess $\mathbf{x}_0 \in A$, builds a sequence $\{\mathbf{x}_k\}_{k\in\mathbb{N}}$ of points of A converging to a stationary point $\mathbf{x}^* \in A$.

in the hope of reaching a minimum, the optimization methods employs a sequence of descent directions.

Definition 1.3.1. *Directional derivatives*

Let $\mathbf{p} \in \mathbb{R}^n$ and f differentiable in a neighbourhood of $\mathbf{x}$. The directional derivative of f in $\mathbf{x}$ with respect to the direction $\mathbf{p}$ is defined as

$$\frac{\partial f}{\partial \mathbf{p}}(\mathbf{x}) = \lim_{h\to 0} \frac{f(\mathbf{x} + h\mathbf{p}) - f(\mathbf{x})}{h}. \tag{1.5}$$

It holds

$$\frac{\partial f}{\partial \mathbf{p}}(\mathbf{x}) = \nabla f(\mathbf{x})^T \mathbf{p}.$$

Definition 1.3.2. *Descent directions*

A direction $\mathbf{p} \in \mathbb{R}^n$ is a descent direction for f in $\mathbf{x}$ if

$$\frac{\partial f}{\partial \mathbf{p}}(\mathbf{x}) = \nabla f(\mathbf{x})^T \mathbf{p} < 0,$$

i.e., if the angle ϑ between $\mathbf{p}$ and $\nabla f(\mathbf{x})$ is such that $\vartheta \in \left(\frac{\pi}{2}, \pi\right]$.

If $\nabla f(\mathbf{x}) \neq \mathbf{0}$, we can always find a descent direction: that of the antigradient $-\nabla f(\mathbf{x})$. From (1.5) it follows that if $\mathbf{p}$ is a descent direction, then it exists $\bar{h} > 0$ such that

$$f(\mathbf{x} + h\mathbf{p}) - f(\mathbf{x}) < 0 \quad \forall h \in (0, \bar{h}).$$

The sequence built will then satisfy the *simple decrease* property:

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k)$$

(or, as it happens in *nonmonotone* algorithms, that satisfies a condition such as $f(\mathbf{x}_{k+m}) < f(\mathbf{x}_k)$ for $m \geq 2$ fix). In this way $\mathbf{x}^*$ will surely not be a maximum point, but in unfortunate cases it may be a saddle point, there is usually no guarantee of convergence to a minimum, as the sufficient condition is usually not checked.

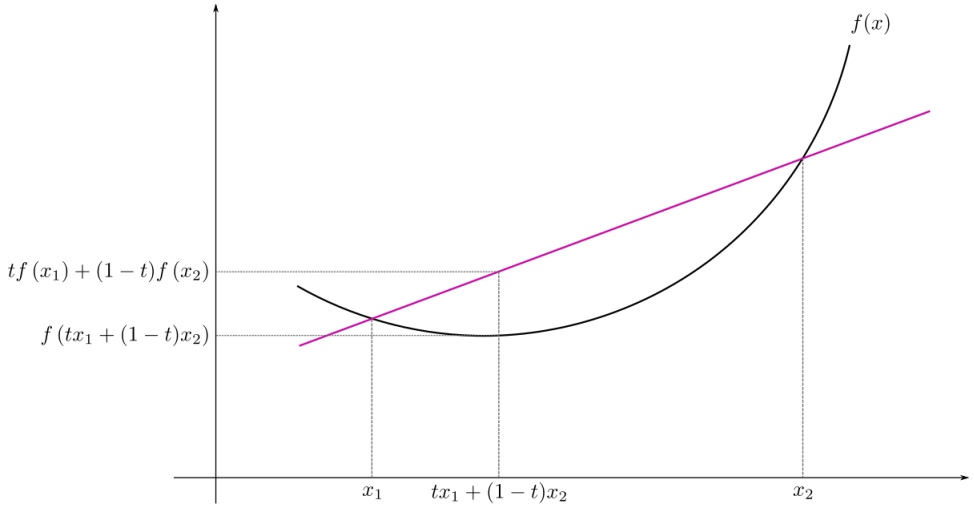


Figure 1.2: Convex function

1.4 A special class of functions: convex functions

Convex functions are function with very desirable properties, which make minimizing them easier. Let A be a convex subset of a real vector space and let $f : A \to \mathbb{R} f : X \to \mathbb{R}$ be a function.

- f is *convex* in A if $\forall \mathbf{x}, \mathbf{y} \in A$

$$f(t\mathbf{x} + (1-t)\mathbf{y}) \leq tf(\mathbf{x}) + (1-t)f(\mathbf{y}), \quad \forall t \in [0,1].$$

- f is *strictly convex* in A if $\forall \mathbf{x}, \mathbf{y} \in A$, $\mathbf{x} \neq \mathbf{y}$

$$f(t\mathbf{x} + (1-t)\mathbf{y}) < tf(\mathbf{x}) + (1-t)f(\mathbf{y}), \quad \forall t \in (0,1).$$

- f is *strongly convex* with parameter $m > 0$ if the following inequality holds for all points $\mathbf{x}, \mathbf{y} \in A$

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{m}{2}\|\mathbf{y} - \mathbf{x}\|^2.$$

The concept of strong convexity extends and parametrizes the notion of strict convexity. A strongly convex function is also strictly convex, but not vice versa. Intuitively speaking, strong convexity means that there exists a quadratic lower bound on the growth of the function.

Lemma 1.4.1. *Let A be a convex subset of $\mathbb{R}^n$. Suppose the function $f : A \longrightarrow \mathbb{R}$ is twice differentiable over A. Then, the followings are equivalent:*

(i) *f is convex.*

(ii) *$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x})$, for all $\mathbf{x}, \mathbf{y} \in A$.*

(iii) *$\nabla^2 f(\mathbf{x}) \succeq 0$, for all $\mathbf{x} \in A$.*

Lemma 1.4.2. *Let A be a convex subset of $\mathbb{R}^n$. Suppose the function $f : A \longrightarrow \mathbb{R}$ is twice differentiable over A. Then:*

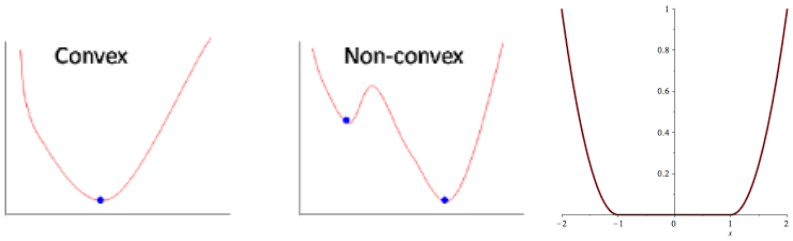


Figure 1.3: Minima of convex functions

(i) f *is strictly convex if and only if* $f(\mathbf{y}) > f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x})$*, for all* $\mathbf{x}, \mathbf{y} \in A$*,* $\mathbf{x} \neq \mathbf{y}$*.*

(ii) $\nabla^2 f(\mathbf{x}) \succ 0$*, for all* $\mathbf{x} \in A$ *implies* f *strictly convex.*

Remark 1.4.1. *The converse of condition (ii) in Lemma 1.4.2 is not true: for example, the function* $f : \mathbb{R} \to \mathbb{R}$ *given by* $f(x) = x^4$ *is strictly convex but has zero second derivative at* $x = 0$*.*

Theorem 1.4.1 (Convexity along lines)**.** *A function* $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ *is convex if and only if the function* $g : \mathbb{R} \longrightarrow \mathbb{R}$ *given by* $g(t) = f(\mathbf{x} + t\mathbf{y})$ *is convex (as a univariate function) for all* $\mathbf{x}$ *in the domain of* f *and all* $\mathbf{y} \in \mathbb{R}^n$*.*

Let's prove some properties of convex functions, that do not hold for generic non-convex functions.

Lemma 1.4.3. *Minima of convex functions. If* f *is convex, then every local minimum point for* f *is a global minimum point.*

Proof. Let $\mathbf{x}$ be a local minimum point for f and let us proceed by contradiction. Assume that $\mathbf{x}$ is not a global minimum, i.e. that it exists $\mathbf{y} \in A$ such that $f(\mathbf{y}) < f(\mathbf{x})$. Then, $\forall t \in [0, 1)$

$$f(t\mathbf{x} + (1-t)\mathbf{y}) \leq tf(\mathbf{x}) + (1-t)f(\mathbf{y}) < tf(\mathbf{x}) + (1-t)f(\mathbf{x}) = f(\mathbf{x}).$$

Then f has lower values in the points of the segment that connects $\mathbf{x}$ with $\mathbf{y}$ (except the point $\mathbf{x}$) than in $\mathbf{x}$, so $\mathbf{x}$ cannot be a local minimum point for f. □

Lemma 1.4.4. *Minima of strictly convex functions. If* f *is strictly convex it has just one minimum point.*

Proof. Let $\mathbf{x}$ be a minimum point for f and assume, by contradiction, that it exists another minimum point $\mathbf{y}$. From Lemma 1.4.3, all the minima points of f are global minima, so $f(\mathbf{x}) = f(\mathbf{y})$. Then $\forall t \in (0, 1)$

$$f(t\mathbf{x} + (1-t)\mathbf{y}) < tf(\mathbf{x}) + (1-t)f(\mathbf{y}) = tf(\mathbf{x}) + (1-t)f(\mathbf{x}) = f(\mathbf{x}).$$

Then f has lower values in the points of the segment that connects $\mathbf{x}$ with $\mathbf{y}$ (except for the endpoints) than in $\mathbf{x}$. Thus $\mathbf{x}$ cannot be a minimum point for f. □

Notice that if the function is convex but not strictly convex, it can have multiple minima, cf. Figure 1.3.

1.5 Quadratic functions

A quadratic function is a function of the form

$$\begin{array}{rccl} q: & \mathbb{R}^n & \longrightarrow & \mathbb{R} \\ & \mathbf{x} & \longmapsto & q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x} = \frac{1}{2}\sum_{i,j=1}^{n} x_i a_{ij} x_j - \sum_{i=1}^{n} b_i x_i, \end{array}$$

with $A \in \mathbf{R}^{n\times n}$ symmetric, $\mathbf{b} \in \mathbb{R}^n$.

Remark 1.5.1. *We can easily compute the gradient and Hessian for a quadratic function:*

- $\nabla q(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$.
- $H(\mathbf{x}) = A$.

Definition 1.5.1. *$q(\mathbf{x})$ is positive definite if A is positive definite.*

Theorem 1.5.1. *A quadratic function that is positive definite is a strictly convex function.*

Theorem 1.5.2. *A positive definite quadratic function $q(\mathbf{x})$ has a unique minimizer $\mathbf{x}^*$, that is the unique solution of the problem*

$$A\mathbf{x} = \mathbf{b}.$$

Proof. As $q(\mathbf{x})$ is strictly convex, $q(\mathbf{x})$ has at most one minimizer. The Hessain matrix of $q(\mathbf{x})$ is A and it is positive definite, so from Theorem 1.2.3 and Theorem 1.2.1 it holds

$$\mathbf{x}^* \text{ is a minimizer of } q(\mathbf{x}) \quad \Longleftrightarrow \quad \mathbf{x}^* \text{ is a solution of } \nabla q(\mathbf{x}) = \mathbf{0},$$

i.e.

$$\mathbf{x}^* \text{ is a minimizer of } q(\mathbf{x}) \quad \Longleftrightarrow \quad \mathbf{x}^* \text{ is a solution of } A\mathbf{x} = \mathbf{b}.$$

A is positive definite and therefore invertible, so the problem $A\mathbf{x} = \mathbf{b}$ has a unique solution. □

Practice time

Along all the course we will implement and the test the methods we study on practical problems. Our first test problem will be the Rosenbrock function. In mathematical optimization, the *Rosenbrock function* is a non-convex function, introduced by Howard H. Rosenbrock in 1960, which is used as a performance test problem for optimization algorithms. The global minimum is inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial. To converge to the global minimum, however, is difficult.

The n-dimensional function is defined by

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} [b(x_{i+1} - x_i^2)^2 + (a - x_i)^2],$$

which in 2 dimension becomes

$$f(x, y) = (a - x)^2 + b(y - x^2)^2.$$

Usually these parameters are set such that $a = 1$ and $b = 100$, in this case it has a global minimum at $\mathbf{x} = [1, \ldots, 1]$, where $f(\mathbf{x}) = 0$. Only in the trivial case where $a = 0$ the function is symmetric and the minimum is at the origin.

Let's get the code of this test problem ready for the next times:

a) For generic n, write some functions to compute the value of the function, the gradient and the Hessian matrix.

b) For $n = 2$, write a function to plot the function and its level curves.

Chapter 2

Iterative methods

In unconstrained optimization *iterative algorithms* are usually used to find a minimizer of f, that is algorithms such that, starting from an initial guess $\mathbf{x}_0 \in A$, builds a sequence $\{\mathbf{x}_k\}_{k\in\mathbb{N}}$ of points of A converging to a stationary point $\mathbf{x}^* \in A$. To reach a minimum, the optimization methods employs a sequence of descent directions $\mathbf{d}_k$ and define at each iteration k

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_h \mathbf{d}_k$$

where α_k is called the step length.

If α is small enough, we have that from Taylor's theorem that the *simple decrease* property is satisfied:

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k).$$

The simple decrease property ensures $\mathbf{x}^*$ not to be a maximum point, but in unfortunate cases it may be a saddle point, there is usually no guarantee of convergence to a minimum, as the sufficient condition is usually not checked.

2.0.1 Examples of descent directions

Direction of steepest descent

The steepest descent direction for f in $\mathbf{x}_k$ is

$$\mathbf{p}_k = -\nabla f(\mathbf{x}_k).$$

The direction $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$ is called of steepest descent for f in $\mathbf{x}_k$ because it is the direction in which, starting from $\mathbf{x}_k$, the values of f decrease the fastest. Indeed the direction of steepest descent is the one that minimizes the directional derivative of f in $\mathbf{x}_k$:

$$\frac{\partial f}{\partial \mathbf{p}}(\mathbf{x}_k) = \nabla f(\mathbf{x}_k)^T \mathbf{p} \underbrace{=}_{\substack{\vartheta \in [0,\pi] \text{ is the angle} \\ \text{between } \nabla f(\mathbf{x}_k) \text{ and } \mathbf{p}}} \|\nabla f(\mathbf{x}_k)\| \|\mathbf{p}\| \cos\vartheta,$$

If we assume, without loss of generality, that $\|\mathbf{p}\| = 1$, the direction that maximises the decrease is the one that minimizes $\cos\vartheta$, so it must be such that $\vartheta = \pi$ and so

$$p_k = \frac{-\nabla f(\mathbf{x}_k)}{\|\nabla f(\mathbf{x}_k)\|}.$$

An advantage of the steepest descent method (the one that uses this direction) is that it requires just the computation of the gradient of f at each iteration, and not that of the second order derivatives. However the convergence is generally really slow (it requires a large number of iterations to reach a stationary point).

Gradient or steepest descent method
Given $\mathbf{x}_0$, toll > 0, set $k = 0$.

While $\|\nabla f(\mathbf{x}_k)\| >$ toll

1. Compute the search direction $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$.
2. Choose α_k.
3. Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
4. Set $k = k + 1$

Newton's direction

Let us consider the quadratic model of f in $\mathbf{x}_k$, deriving from the second order Taylor's polynomial:

$$m_k(\mathbf{p}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \mathbf{p} + \frac{1}{2}\mathbf{p}^T H(\mathbf{x}_k)\mathbf{p},$$

and let us assume $H(\mathbf{x}_k)$ to be positive definite.
Newton's direction $\mathbf{p}_k^N$ is the minimizer of $m_k(\mathbf{p})$, i.e., being $m_k(\mathbf{p})$ a positive definite quadratic function, it is the solution of Newton's system

$$H(\mathbf{x}_k)\mathbf{p} = -\nabla f(\mathbf{x}_k). \tag{2.1}$$

The analytic expression of Newton's direction is then

$$\mathbf{p}_k^N = -H(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k).$$

Remark 2.0.1. *Note that in practice $H(\mathbf{x}_k)$ is never explicitly inverted to compute such a direction, Newton's system is rather solved, because linear equation solution is faster and produces less round-off error than inversion and multiplication. This is particularly evident when the dimension of the problem is large.*

Thanks to the fact that $H(\mathbf{x}_k)$ is positive definite, we have not only that $H(\mathbf{x}_k)$ is invertible, so Newton's system has one and just one solution, but it also holds that $\mathbf{p}_k^N$ is a descent direction, as

$$\nabla f(\mathbf{x}_k)^T \mathbf{p}_k^N = \nabla f(\mathbf{x}_k)^T \Big(- H(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)\Big) = -\nabla f(\mathbf{x}_k)^T H(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k) < 0,$$

because $H(\mathbf{x}_k)^{-1}$ is positive definite.
If $H(\mathbf{x}_k)$ is not positive definite, not only $\mathbf{p}_k^N$ may not be well-defined ($H(\mathbf{x}_k)$ may not be invertible and so (2.1) may not have a unique solution), but even if $\mathbf{p}_k^N$ is well-defined, $\mathbf{p}_k^N$ may not be a descent direction.
Methods based on Newton's direction are usually characterized by fast local convergence (they require few iterations to converge), but they are expensive as they require not only the computation of $H(\mathbf{x}_k)$ at each step, but also the solution of the linear system (2.1) that may be expensive if the size of the problem is large.

Newton's method
Given $\mathbf{x}_0$, toll > 0, set $k = 0$.

While $\|\nabla f(\mathbf{x}_k)\| >$ toll

1. Compute the search direction solving $H_k \mathbf{p}_k = -\nabla f(\mathbf{x}_k)$.
2. Choose α_k
3. Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
4. Set $k = k + 1$

Quasi-Newton directions

Let us consider a quadratic model for f:

$$m_k(\mathbf{p}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \mathbf{p} + \frac{1}{2} \mathbf{p}^T B_k \mathbf{p},$$

where $B_k \approx H(\mathbf{x}_k)$ is a SPD (symmetric positive definite) matrix.
Quasi-Newton direction $\mathbf{p}_k$ is the minimizer of $m_k(\mathbf{p})$, i.e. is the solution of quasi-Newton system

$$B_k \mathbf{p} = -\nabla f(\mathbf{x}_k);$$

the analytical expression of the quasi-Newton direction is

$$\mathbf{p}_k^{QN} = -B_k^{-1} \nabla f(\mathbf{x}_k).$$

Quasi Newton method
Given $\mathbf{x}_0$, toll > 0, inverse Hessian approximation $\tilde{B}_0$, set $k = 0$.

While $\|\nabla f(\mathbf{x}_k)\| >$ toll

1. Compute the search direction $\mathbf{p}_k = -\tilde{B}_k \nabla f(\mathbf{x}_k)$.
2. Choose α_k
3. Update the Hessian approximation: choose $\tilde{B}_{k+1}$
4. Set $k = k + 1$

2.0.2 What do we expect from an optimization method?

What we usually expect from an optimization method is its *convergence*:

$$\lim_{k \to \infty} \|\nabla f(\mathbf{x_k})\| = 0. \tag{2.2}$$

A critical choice for optimization algorithm is the starting guess: usually such algorithms have problems to converge when the starting point is far from the true solution and often there are not a-priori information available to choose a starting point. A very desirable property is then the *global convergence*. An algorithm is said to be *globally convergent*, if (2.2) is guaranteed for any initial guess $\mathbf{x}_0$, independently of the proximity of $\mathbf{x}_0$ to $\mathbf{x}^*$.

Remark 2.0.2. *Very important! Global convergence does not mean convergence to a global minimum!!*

Ideally we would also like to prove convergence of the generated sequence:

$$\lim_{k\to\infty} \mathbf{x_k} = \mathbf{x}^*,$$

this is usually difficult to prove and it is a result established only for initial guesses close enough to the true solution. Methods that are convergent just for initial guesses close enough to a minimum are said *locally convergent*.

2.0.3 Practical choices

The stopping criterion In practice the algorithm is stopped when a suitable stopping criterion is met. The stopping criterion is usually based on the norm of the gradient. As we want to reach a stationary point, we want (2.2) to be satisfied, so, given a positive tolerance $\epsilon > 0$, the method is stopped as soon as $\|\nabla f(\mathbf{x_k})\| < \epsilon$. The magnitude of the tolerance depends on the specific application. Usually in classical applications this can be $10^{-6}, 10^{-8}$, in machine learning applications higher tolerances are used $(10^{-2}, 10^{-3})$.

The step length The choice of the step length is critical for local optimization methods. If the step-length is too large the method may not be able to converge. On the other hand, if the step length is too short the method may take a large number of iterations to reach a solution. When the Newton method converges, it is desirable to choose $\alpha_k = 1$, usually lower step-length are necessary to ensure convergence of the gradient method. There exist some strategies to make a wise choice, which can for instance make a local method globally convergent: line-search and trust-region strategies. These guarantee a step length that is not fixed, but is dynamically adapted through the iterations. In the following we will focus on line-search strategies.

2.1 How fast can we reach the solution? Rates of convergence

Let $\{\mathbf{x}_k\}_{k\in\mathbb{N}}$ be a sequence of elements of $\mathbb{R}^n$ converging to $\mathbf{x}^*$. The speed at which a convergent sequence approaches its limit is represented by its order of convergence and by its rate of convergence. The sequence is said to have order of convergence $q \geq 1$ and rate of convergence μ if

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^q} = \mu.$$

- The sequence is said to converge *linearly* if it exists $r \in (0, 1)$ such that

$$\lim_{k\to+\infty} \frac{\|\mathbf{x}_{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}_k - \mathbf{x}^*\|} = r.$$

- The sequence is said to converge *superlinearly* (faster than linearly) if

$$\lim_{k\to+\infty} \frac{\|\mathbf{x}_{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}_k - \mathbf{x}^*\|} = 0.$$

- The sequence is said to converge *sublinearly* (slower than linearly) if

$$\lim_{k\to+\infty} \frac{\|\mathbf{x}_{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}_k - \mathbf{x}^*\|} = 1.$$

- The convergence is said to be *quadratic* if it exists $M > 0$ such that

$$\lim_{k \to +\infty} \frac{\|\mathbf{x}_{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}_k - \mathbf{x}^*\|^2} < M.$$

- In general, given $p > 1$, the sequence is said to converge with order p if it exists $M > 0$ such that

$$\lim_{k \to +\infty} \frac{\|\mathbf{x}_{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}_k - \mathbf{x}^*\|^p} < M.$$

  In particular, convergence with order

  - $p = 1$ is called linear convergence,
  - $p = 2$ is called quadratic convergence,
  - $p = 3$ is called cubic convergence.

If the convergence is linear it means that

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq r\|\mathbf{x}_k - \mathbf{x}^*\|$$

for all k large enough. This means that (asymptotically) the distance of the solution approximation from the solution at step $k+1$ ($\|\mathbf{x}_{k+1} - \mathbf{x}^*\|$) is lower than a fraction of the distance of the solution approximation from the solution at the previous step ($r\|\mathbf{x}_k - \mathbf{x}^*\|$): at each iteration this distance is decreased and the rate at which it is decreased depends on r, if r is close to one the decrease is really slow.

Usually the cost of a method is directly proportional to the speed of convergence: generally an expensive method (for which a single iteration is expensive to compute) has a higher rate of convergence and requires less iterations to converge. For example methods based on Newton's directions enjoy a quadratic local rate of convergence. Quasi-Newton methods are less expensive then Newton's method, but this is paid with a slower superlinear convergence.

2.2 Convergence of gradient method

Many results can be established about the convergence of gradient method, depending on the assumptions we make on the function. We report here a result for smooth-strongly convex functions.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be

1. *L-smooth* if ∇f is Lipschitz continuous with Lipschitz constant L:

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|$$

   for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$.

2. *μ-strongly convex* if

$$f(\mathbf{y}) \geq f(\mathbf{x}) - \nabla f(\mathbf{x})^T(\mathbf{x} - \mathbf{y}) + \frac{\mu}{2}\|\mathbf{x} - \mathbf{y}\|^2 \tag{2.3}$$

   $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$.

In many convergence results the function is assumed to have a Lipschitz gradient. This is a smoothness assumption: if f is Lipschitz continuous it means that the gradient is bounded from above, so the slope of the function cannot change abruptly. If the gradient is Lipschitz continuous the Hessian is bounded in norm from above by some L, which is typically a reasonable assumption to make.

Lemma 2.2.1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ a twice differentiable function. If f is L-smooth, then the two following relations hold for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$:*

$$f(\mathbf{x}) \leq f(\mathbf{y}) + \nabla f(\mathbf{y})^T(\mathbf{x}-\mathbf{y}) + \frac{L}{2}\|\mathbf{x}-\mathbf{y}\|^2, \tag{2.4}$$

$$f(\mathbf{x}^*) - f(\mathbf{x}) \leq -\frac{1}{2L}\|\nabla f(\mathbf{x})\|^2. \tag{2.5}$$

Proof. Using the first order Taylor expansion of $f(\mathbf{x})$ we have that

$$\begin{aligned}
f(\mathbf{x}) &= f(\mathbf{y}) + \int_0^1 \nabla f(\mathbf{y} + t(\mathbf{x}-\mathbf{y}))^T(\mathbf{x}-\mathbf{y})\, dt \\
&= f(\mathbf{y}) + \nabla f(\mathbf{y})^T(\mathbf{x}-\mathbf{y}) + \int_0^1 [\nabla f(\mathbf{y} + t(\mathbf{x}-\mathbf{y})) - \nabla f(\mathbf{y})]^T(\mathbf{x}-\mathbf{y})\, dt \\
&\leq f(\mathbf{y}) + \nabla f(\mathbf{y})^T(\mathbf{x}-\mathbf{y}) + \int_0^1 \|\nabla f(\mathbf{y} + t(\mathbf{x}-\mathbf{y})) - \nabla f(\mathbf{y})\|\|\mathbf{x}-\mathbf{y}\|\, dt \\
&\leq f(\mathbf{y}) + \nabla f(\mathbf{y})^T(\mathbf{x}-\mathbf{y}) + L\int_0^1 t\|\mathbf{x}-\mathbf{y}\|^2\, dt \\
&= f(\mathbf{y}) + \nabla f(\mathbf{y})^T(\mathbf{x}-\mathbf{y}) + \frac{L}{2}\|\mathbf{x}-\mathbf{y}\|^2.
\end{aligned}$$

To prove the second point we use (2.4) on $\mathbf{x} - \frac{1}{L}\nabla f(\mathbf{x})$, obtaining

$$\begin{aligned}
f\left(\mathbf{x} - \frac{1}{L}\nabla f(\mathbf{x})\right) &\leq f(\mathbf{x}) - \frac{1}{L}\nabla f(\mathbf{x})^T\nabla f(\mathbf{x}) + \frac{L}{2}\left\|\frac{1}{L}\nabla f(\mathbf{x})\right\|^2 \\
&= f(\mathbf{x}) - \frac{1}{2L}\|\nabla f(\mathbf{x})\|^2.
\end{aligned}$$

The result follows by considering that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for any $\mathbf{x}$.

□

Theorem 2.2.1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ a twice differentiable function, L-smooth and μ-strongly convex. Let $\mathbf{x}^*$ be the global minimum of f. Given $\mathbf{x}_0 \in \mathbb{R}^n$ and $\frac{1}{L} \geq \alpha > 0$, the iterates*

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\nabla f(\mathbf{x}_k) \tag{2.6}$$

converge according to

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2 \leq (1-\alpha\mu)^{k+1}\|\mathbf{x}_0 - \mathbf{x}^*\|^2.$$

Proof. From (2.6) we have that

$$\begin{aligned}
\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2 &= \|\mathbf{x}_k - \mathbf{x}^* - \alpha\nabla f(\mathbf{x}_k)\|^2 \\
&= \|\mathbf{x}_k - \mathbf{x}^*\|^2 - 2\alpha\nabla f(\mathbf{x}_k)^T(\mathbf{x}_k - \mathbf{x}^*) + \alpha^2\|\nabla f(\mathbf{x}_k)\|^2 \\
&\overbrace{\leq}^{(2.3)} (1-\alpha\mu)\|\mathbf{x}_k - \mathbf{x}^*\|^2 - 2\alpha(f(\mathbf{x}_k) - f(\mathbf{x}^*)) + \alpha^2\|\nabla f(\mathbf{x}_k)\|^2 \\
&\overbrace{\leq}^{(2.5)} (1-\alpha\mu)\|\mathbf{x}_k - \mathbf{x}^*\|^2 - 2\alpha(f(\mathbf{x}_k) - f(\mathbf{x}^*)) + 2\alpha^2 L(f(\mathbf{x}_k) - f(\mathbf{x}^*)) \\
&= (1-\alpha\mu)\|\mathbf{x}_k - \mathbf{x}^*\|^2 - 2\alpha(1-\alpha L)(f(\mathbf{x}_k) - f(\mathbf{x}^*)).
\end{aligned}$$

Since $\frac{1}{L} \geq \alpha$ we have that $-2\alpha(1-\alpha L)$ is negative, and thus can be dropped to give

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2 \leq (1-\alpha\mu)\|\mathbf{x}_k - \mathbf{x}^*\|^2.$$

The thesis can be obtained unrolling the recurrence.

□

2.3 Convergence of Newton's method

What is usually called the "pure Newton's method" is based on the following iterative scheme:

$$\left\{ \begin{array}{c} \text{Given } \mathbf{x}_0 \\ \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k^N \end{array} \right\}$$

As shown in the following theorem it is a locally convergent method, and the local rate of convergence is quadratic. This is not a globally convergent method and it can be made so by coupling it with a line-search strategy, as we will see in the next chapter.

Theorem 2.3.1. *Local convergence of Newton's method. Let $\mathbf{x}^*$ be a minimizer for f, Ω be a neighbourhood of $\mathbf{x}^*$, $f \in C^2(\Omega)$, $H(\mathbf{x}^*)$ positive definite and $H(\mathbf{x})$ Lipschitz continuous in Ω with Lipschitz constant L.*
It exists $\rho > 0$ such that if $\mathbf{x}_0 \in B_\rho(\mathbf{x}^*)$ then the sequence $\{\mathbf{x}_k\}$ built by Newton's method is well-defined [1], converges to $\mathbf{x}^*$ quadratically and $\|\nabla f(\mathbf{x}_k)\|$ converges to 0 quadratically.

Proof. By assumption it exists $r > 0$ such that $\forall \mathbf{x} \in B_r(\mathbf{x}^*)$ $H(\mathbf{x})$ is positive definite (so invertible) and it holds (see TD2)

$$\|H(\mathbf{x})^{-1}\| \leq 2\|H(\mathbf{x}^*)^{-1}\|. \tag{2.7}$$

We can assume $B_r(\mathbf{x}^*) \subseteq \Omega$.
If $\mathbf{x}_k \in B_r(\mathbf{x}^*)$ then

$$\begin{aligned} \mathbf{x}_{k+1} - \mathbf{x}^* &= \mathbf{x}_k + \mathbf{p}_k^N - \mathbf{x}^* = \mathbf{x}_k - \mathbf{x}^* - H(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k) = H(\mathbf{x}_k)^{-1}\Big(H(\mathbf{x}_k)(\mathbf{x}_k - \mathbf{x}^*) - \nabla f(\mathbf{x}_k)\Big) \\ &= H(\mathbf{x}_k)^{-1}\Big(H(\mathbf{x}_k)(\mathbf{x}_k - \mathbf{x}^*) + \overbrace{\nabla f(\mathbf{x}^*)}^{=\mathbf{0}} - \nabla f(\mathbf{x}_k)\Big). \end{aligned}$$

Because

$$\nabla f(\mathbf{x}^*) - \nabla f(\mathbf{x}_k) = \Big[\nabla f(\mathbf{x}_k + t(\mathbf{x}^* - \mathbf{x}_k))\Big]_{t=0}^{t=1} = \int_0^1 H(\mathbf{x}_k + t(\mathbf{x}^* - \mathbf{x}_k))(\mathbf{x}^* - \mathbf{x}_k)\, \mathrm{d}t,$$

we have

$$\begin{aligned} \mathbf{x}_{k+1} - \mathbf{x}^* &= H(\mathbf{x}_k)^{-1}\left(H(\mathbf{x}_k)(\mathbf{x}_k - \mathbf{x}^*) + \int_0^1 H(\mathbf{x}_k + t(\mathbf{x}^* - \mathbf{x}_k))(\mathbf{x}^* - \mathbf{x}_k)\, \mathrm{d}t\right) \\ &= H(\mathbf{x}_k)^{-1}\left(\underbrace{H(\mathbf{x}_k)(\mathbf{x}_k - \mathbf{x}^*)}_{\text{does not depend on } t} - \int_0^1 H(\mathbf{x}_k + t(\mathbf{x}^* - \mathbf{x}_k))(\mathbf{x}_k - \mathbf{x}^*)\, \mathrm{d}t\right) \\ &= H(\mathbf{x}_k)^{-1}\int_0^1 \Big(H(\mathbf{x}_k) - H(\mathbf{x}_k + t(\mathbf{x}^* - \mathbf{x}_k))\Big)(\mathbf{x}_k - \mathbf{x}^*)\, \mathrm{d}t. \end{aligned}$$

[1] We can build it because $H(\mathbf{x}_k)$ is positive definite for each $k \in \mathbb{N}$.

Passing to norms we obtain that

$$\begin{aligned}
\|\mathbf{x}_{k+1}-\mathbf{x}^*\| &\le \|H(\mathbf{x}_k)^{-1}\|\left\|\int_0^1 \Big(H(\mathbf{x}_k)-H(\mathbf{x}_k+t(\mathbf{x}^*-\mathbf{x}_k))\Big)(\mathbf{x}_k-\mathbf{x}^*)\,\mathrm{d}t\right\| \quad \underbrace{\le}_{(2.7)}\\
&\le 2\|H(\mathbf{x}^*)^{-1}\|\left\|\int_0^1 \Big(H(\mathbf{x}_k)-H(\mathbf{x}_k+t(\mathbf{x}^*-\mathbf{x}_k))\Big)(\mathbf{x}_k-\mathbf{x}^*)\,\mathrm{d}t\right\|\\
&\le 2\|H(\mathbf{x}^*)^{-1}\|\int_0^1 \Big\|H(\mathbf{x}_k)-H(\mathbf{x}_k+t(\mathbf{x}^*-\mathbf{x}_k))\Big\|\|\mathbf{x}_k-\mathbf{x}^*\|\,\mathrm{d}t\\
&= 2\|H(\mathbf{x}^*)^{-1}\|\|\mathbf{x}_k-\mathbf{x}^*\|\int_0^1 \Big\|H(\mathbf{x}_k)-H(\mathbf{x}_k+t(\mathbf{x}^*-\mathbf{x}_k))\Big\|\,\mathrm{d}t\\
&\le 2\|H(\mathbf{x}^*)^{-1}\|\|\mathbf{x}_k-\mathbf{x}^*\|\int_0^1 L\Big\|\mathbf{x}_k-(\mathbf{x}_k+t(\mathbf{x}^*-\mathbf{x}_k))\Big\|\,\mathrm{d}t\\
&= 2\underbrace{L\|H(\mathbf{x}^*)^{-1}\|}_{\text{we call this }\widetilde{L}}\|\mathbf{x}_k-\mathbf{x}^*\|\int_0^1 \|-t(\mathbf{x}^*-\mathbf{x}_k)\|\,\mathrm{d}t\\
&= 2\widetilde{L}\|\mathbf{x}_k-\mathbf{x}^*\|^2\underbrace{\int_0^1 t\,\mathrm{d}t}_{=\left[\frac{t^2}{2}\right]_0^1=\frac{1}{2}} = \widetilde{L}\|\mathbf{x}_k-\mathbf{x}^*\|^2.
\end{aligned}$$

We have proved that if $\mathbf{x}_k \in B_r(\mathbf{x}^*)$ then

$$\|\mathbf{x}_{k+1}-\mathbf{x}^*\| \le \widetilde{L}\|\mathbf{x}_k-\mathbf{x}^*\|^2. \tag{2.8}$$

Let $\rho = \min\left\{r, \dfrac{1}{2\widetilde{L}}\right\}$.
Let us assume that $\mathbf{x}_0 \in B_\rho(\mathbf{x}^*)$. Because $\mathbf{x}_0 \in B_\rho(\mathbf{x}^*) \underbrace{\subseteq}_{\rho\le r} B_r(\mathbf{x}^*)$, it follows

$$\|\mathbf{x}_1-\mathbf{x}^*\| \le \widetilde{L}\|\mathbf{x}_0-\mathbf{x}^*\|^2 \le \widetilde{L}\rho^2 \underbrace{\le}_{\rho\le\frac{1}{2\widetilde{L}}\implies\widetilde{L}\rho\le\frac{1}{2}} \frac{1}{2}\rho,$$

i.e., $\mathbf{x}_1 \in B_\rho(\mathbf{x}^*)$. Because $\mathbf{x}_1 \in B_\rho(\mathbf{x}^*) \underbrace{\subseteq}_{\rho\le r} B_r(\mathbf{x}^*)$, analogously it holds $\|\mathbf{x}_2-\mathbf{x}^*\| < \dfrac{1}{2}\rho$, i.e., $\mathbf{x}_2 \in B_\rho(\mathbf{x}^*)$, and so on. Then by induction we have that

$$\mathbf{x}_k \in B_\rho(\mathbf{x}^*) \qquad \forall k \in \mathbb{N}.$$

This ensures that the sequence $\{\mathbf{x}_k\}$ built by Newton's method is well-defined because the matrices $H(\mathbf{x}_k)$ are all positive definite from (2.7). Moreover, from (2.8)

$$\|\mathbf{x}_{k+1}-\mathbf{x}^*\| \le \widetilde{L}\|\mathbf{x}_k-\mathbf{x}^*\|^2 \qquad \forall k \in \mathbb{N}. \tag{2.9}$$

Then $\forall k \in \mathbb{N}$

$$\|\mathbf{x}_{k+1}-\mathbf{x}^*\| \le \widetilde{L}\|\mathbf{x}_k-\mathbf{x}^*\|^2 \le \widetilde{L}\rho\|\mathbf{x}_k-\mathbf{x}^*\| \underbrace{\le}_{\rho\le\frac{1}{2\widetilde{L}}\implies\widetilde{L}\rho\le\frac{1}{2}} \frac{1}{2}\|\mathbf{x}_k-\mathbf{x}^*\|,$$

i.e.

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \frac{1}{2}\|\mathbf{x}_k - \mathbf{x}^*\| \qquad \forall k \in \mathbb{N}$$

and so

$$\|\mathbf{x}_k - \mathbf{x}^*\| \leq \frac{1}{2}\|\mathbf{x}_{k-1} - \mathbf{x}^*\| \leq \left(\frac{1}{2}\right)^2 \|\mathbf{x}_{k-2} - \mathbf{x}^*\| \leq \ldots < \left(\frac{1}{2}\right)^k \|\mathbf{x}_0 - \mathbf{x}^*\| \quad \xrightarrow[k\to+\infty]{} \quad 0,$$

The sequence $\{\mathbf{x}_k\}$ converges then to $\mathbf{x}^*$ and from (2.9) we have that the convergence is quadratic. As x^* is a stationary point by assumption, the sequence $\{\|\nabla f(\mathbf{x}_k)\|\}$ converges to 0 by continuity. We remark that

$$\|\nabla f(\mathbf{x}_{k+1})\| = \|\nabla f(\mathbf{x}_{k+1}) - \overbrace{(H(\mathbf{x}_k)\mathbf{p}_k^N + \nabla f(\mathbf{x}_k))}^{=\mathbf{0}}\| = \|\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k) - H(\mathbf{x}_k)\mathbf{p}_k^N\|.$$

Because

$$\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k) = \Big[\nabla f(\mathbf{x}_k + t(\mathbf{x}_{k+1} - \mathbf{x}_k))\Big]_{t=0}^{t=1} = \Big[\nabla f(\mathbf{x}_k + t\mathbf{p}_k^N)\Big]_{t=0}^{t=1} = \int_0^1 H(\mathbf{x}_k + t\mathbf{p}_k^N)\mathbf{p}_k^N \, \mathrm{d}t,$$

we have

$$\begin{aligned}
\|\nabla f(\mathbf{x}_{k+1})\| &= \left\| \int_0^1 H(\mathbf{x}_k + t\mathbf{p}_k^N)\mathbf{p}_k^N \, \mathrm{d}t - \underbrace{H(\mathbf{x}_k)\mathbf{p}_k^N}_{\text{does not depend on } t} \right\| = \left\| \int_0^1 \Big(H(\mathbf{x}_k + t\mathbf{p}_k^N) - H(\mathbf{x}_k)\Big)\mathbf{p}_k^N \, \mathrm{d}t \right\| \\
&\leq \int_0^1 \|H(\mathbf{x}_k + t\mathbf{p}_k^N) - H(\mathbf{x}_k)\|\|\mathbf{p}_k^N\| \, \mathrm{d}t = \|\mathbf{p}_k^N\| \int_0^1 \|H(\mathbf{x}_k + t\mathbf{p}_k^N) - H(\mathbf{x}_k)\| \, \mathrm{d}t \\
&\leq \quad \|\mathbf{p}_k^N\| \int_0^1 L\|\mathbf{x}_k + t\mathbf{p}_k^N - \mathbf{x}_k\| \, \mathrm{d}t = L\|\mathbf{p}_k^N\| \int_0^1 \|t\mathbf{p}_k^N\| \, \mathrm{d}t \\
&= L\|\mathbf{p}_k^N\|^2 \underbrace{\int_0^1 t \, \mathrm{d}t}_{=\left[\frac{t^2}{2}\right]_0^1 = \frac{1}{2}} = \frac{1}{2}L\|\mathbf{p}_k^N\|^2 = \frac{1}{2}L\|H(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)\|^2 \\
&\leq \quad \frac{1}{2}L\|H(\mathbf{x}_k)^{-1}\|^2\|\nabla f(\mathbf{x}_k)\|^2 \quad \underbrace{\leq}_{(2.7)} \quad \frac{1}{2}L4\|H(\mathbf{x}^*)^{-1}\|^2\|\nabla f(\mathbf{x}_k)\|^2 \\
&= \quad 2L\|H(\mathbf{x}^*)^{-1}\|^2\|\nabla f(\mathbf{x}_k)\|^2 := M\|\nabla f(\mathbf{x}_k)\|^2.
\end{aligned}$$

Because M does not depend on k, we have proved that it exists $M > 0$ such that $\forall k \in \mathbb{N}$

$$\big| \, \|\nabla f(\mathbf{x}_{k+1})\| - 0 \, \big| \leq M \big| \, \|\nabla f(\mathbf{x}_k)\| - 0 \, \big|^2,$$

i.e. the convergence of $\{\|\nabla f(\mathbf{x}_k)\|\}$ to 0 is quadratic. □

Practice time

Implement the steepest descent and the Newton's method. Use them to minimize the Rosenbrock and the Rastrigin functions. First try the initial point $\mathbf{x}_0 = [1.2, 1.2]^T$ and then the more difficult point $\mathbf{x}_0 = [-1.2, 1]^T$.

Plot the relative error $\left(\frac{\|\mathbf{x}_c - \mathbf{x}^*\|}{\|\mathbf{x}^*\|} \text{ with } \mathbf{x}_c \text{ the computed solution and } \mathbf{x}^* \text{the true solution}\right)$, the objective function, the norm of the gradient vs the number of iterations.

a) How many iterations does Newton's method require if $\alpha_k = 1$? And if $\alpha_k = 0.5$? Can you explain this?

b) Does the gradient method converge?

c) Generalize now to the n-dimensional case, $n > 2$. Does Newton's method still converge? Why?

d) Compare the computational time for the two methods.

e) For Newton's method, compare the computational time for x=x-linalg.solve(H,g) and x=x-np.matmul(np.linalg.inv(H),g) (take $n > 1000$)

Chapter 3

Line-search strategies

In this chapter we will introduce and analyse the line-search methods for nonlinear optimization problems.

In line-search methods first of all a descent direction $\mathbf{p}_k$ is chosen, it may be the gradient, newton's or quasi-Newton direction, for instance. Then, the iterative scheme is as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k,$$

a step is taken in the selected direction from the current point $\mathbf{x}_k$ whose length is $\alpha_k > 0$. We call $\alpha_k \mathbf{p}_k$ the step, $\mathbf{p}_k$ the step direction and α_k the step length. The crucial operation in line-search methods is then the computation of the step-length, for which we have to face a tradeoff. We would like to choose α_k to have a substantial reduction of f, but at the same time we do not want to spend too much time making this choice. Ideally, one should choose $\alpha_k > 0$ that minimizes $\varphi(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{p}_k)$, but to do that a minimization problem in $\mathbb{R}$ has to be solved at each iteration (find the points α such that $\varphi'(\alpha) = 0$). This would make the algorithm too expensive, except in some exceptional cases, as the one introduced in the following section. Different strategies are then used to make this choice in the general case, as we will see later.

3.1 Steepest descent method for quadratic functions

When using line-search strategies it is in general too expensive to choose α_k solving the minimization problem

$$\min_{\alpha > 0} \varphi(\alpha) \tag{3.1}$$

with $\varphi(\alpha) = f(x_k + \alpha p_k)$, given the current iterate x_k and a descent direction p_k. In particular cases the solution of this minimization problem can be computed analytically, and so the optimal value can be employed in a cheap way. This is the case when the steepest descent direction is used for quadratic functions. In case the exact minimizer of (3.1) is used, the line-search is said to be *exact*, it is said to be *inexact* if an approximation is used instead.

Let us consider the positive definite quadratic function

$$\begin{array}{rccl} q: & \mathbb{R}^n & \longrightarrow & \mathbb{R} \\ & \mathbf{x} & \longmapsto & q(\mathbf{x}) = \dfrac{1}{2}\mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x}. \end{array} \tag{3.2}$$

We know that $\mathbf{x}^*$ is a minimizer for $q(\mathbf{x})$ if and only if $\nabla q(\mathbf{x}^*) = \mathbf{0}$.
We choose the steepest descent direction

$$\mathbf{p}_k = -\nabla q(\mathbf{x}_k) = -(A\mathbf{x}_k - \mathbf{b}) := -\mathbf{g}_k,$$

and we use the line-search method

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k = \mathbf{x}_k - \alpha_k \mathbf{g}_k.$$

In this particular case it is possible to choose $\alpha_k \in \mathbb{R}$ that exactly minimizes

$$\varphi(\alpha) = q(\mathbf{x}_k + \alpha \mathbf{p}_k) = q(\mathbf{x}_k - \alpha \mathbf{g}_k) = \frac{1}{2}(\mathbf{x}_k - \alpha \mathbf{g}_k)^T A(\mathbf{x}_k - \alpha \mathbf{g}_k) - (\mathbf{x}_k - \alpha \mathbf{g}_k)^T \mathbf{b}.$$

The minimizer can indeed be analytically computed. Performing the computations and by remarking that

$$\underbrace{\mathbf{x}_k^T A \mathbf{g}_k}_{\in \mathbb{R}} = (\mathbf{x}_k^T A \mathbf{g}_k)^T = \mathbf{g}_k^T A^T \mathbf{x}_k \underbrace{=}_{A \text{ is symmetric}} \mathbf{g}_k^T A \mathbf{x}_k,$$

we obtain that

$$\begin{aligned} \varphi(\alpha) &= \frac{1}{2}\mathbf{g}_k^T A \mathbf{g}_k \alpha^2 - \mathbf{x}_k^T A \mathbf{g}_k \alpha + \mathbf{b}^T \mathbf{g_k} \alpha + \frac{1}{2}\mathbf{x}_k^T A \mathbf{x}_k - \mathbf{x}_k^T \mathbf{b} = \\ &= \frac{1}{2}\mathbf{g}_k^T A \mathbf{g}_k \alpha^2 - \mathbf{g}_k^T \mathbf{g}_k \alpha + \frac{1}{2}\mathbf{x}_k^T A \mathbf{x}_k - \mathbf{x}_k^T \mathbf{b}, \end{aligned}$$

i.e., $\varphi(\alpha)$ is a parabola with branches pointing up, since $\frac{1}{2}\mathbf{g}_k^T A \mathbf{g}_k > 0$, as A is positive definite. Then the minimizer of $\varphi(\alpha)$ is α such that

$$0 = \varphi'(\alpha) = \mathbf{g}_k^T A \mathbf{g}_k \alpha - \mathbf{g}_k^T \mathbf{g}_k,$$

that is

$$\alpha_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_k^T A \mathbf{g}_k} = \frac{\|\nabla \mathbf{q}(\mathbf{x}_k)\|^2}{\|\nabla \mathbf{q}(\mathbf{x}_k)\|_A^2},$$

having defined $\forall \mathbf{y} \in \mathbb{R}^n$, $\forall A \in \mathbb{R}^{n \times n}$ SPD the energy norm $\|\mathbf{y}\|_A^2 = \mathbf{y}^T A \mathbf{y}$. Remark that $\alpha_k > 0$: that means that we will go along the direction $\mathbf{p}_k$, and not in the opposite one.

We derive then the following algorithm for the steepest descent method or gradient method.

Algorithm for gradient method (first version)

0. Given $\mathbf{x}_0, A, \mathbf{b}, \text{toll}$
1. Compute $\mathbf{g}_0 = A\mathbf{x}_0 - \mathbf{b}$
2. For $k = 0, 1, \ldots$
   1. Compute $\alpha_k = \dfrac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_k^T A \mathbf{g}_k}$
   2. Set $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k$
   3. Compute $\mathbf{g}_{k+1} = A\mathbf{x}_{k+1} - \mathbf{b}$
   4. If $\|\mathbf{g}_{k+1}\| \leq \text{toll}$ return $\mathbf{x}_{k+1}$ and stop.

At each iteration two matrix-vector products are performed: $A\mathbf{g}_k$ and $A\mathbf{x}_{k+1}$. The algorithm can be improved to require just one matrix vector product at each iteration, thanks to the fact that

$$\mathbf{g}_{k+1} = A\mathbf{x}_{k+1} - \mathbf{b} = A(\mathbf{x}_k - \alpha_k \mathbf{g}_k) - \mathbf{b} = A\mathbf{x}_k - \alpha_k A\mathbf{g}_k - \mathbf{b} = \mathbf{g}_k - \alpha_k A\mathbf{g}_k.$$

We derive then the following optimized version of the algorithm.

Algorithm for gradient method (optimized version)

0. Given $\mathbf{x}_0, A, \mathbf{b}, \text{toll}$
1. Compute $\mathbf{g}_0 = A\mathbf{x}_0 - \mathbf{b}$
2. For $k = 0, 1, \ldots$
   1. Compute $\mathbf{r}_k = A\mathbf{g}_k$
   2. Compute $\alpha_k = \dfrac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_k^T \mathbf{r}_k}$
   3. Set $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k$
   4. Compute $\mathbf{g}_{k+1} = \mathbf{g}_k - \alpha_k \mathbf{r}_k$
   5. If $\|\mathbf{g}_{k+1}\| \leq \text{toll}$ return $\mathbf{x}_{k+1}$ and stop

This algorithm has then a really low per-iteration cost. The memory consumption is also low: at each iteration it requires to memorize the vector $\mathbf{g}_k$, and no matrices.
The convergence is on the contrary slow: we can prove the following results, which state the linear convergence of the method.

Theorem 3.1.1. *When the steepest descent method with exact line searches is applied to the strongly convex quadratic function* (3.2)*, the error norm satisfies*

$$\frac{\|\mathbf{x}_{k+1} - \mathbf{x}^*\|_A}{\|\mathbf{x}_k - \mathbf{x}^*\|_A} \leq \frac{k_2(A) - 1}{k_2(A) + 1},$$

where $k_2(A)$ is the condition number of A in the 2-norm. [1]

Theorem 3.1.2. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable, and that the iterates generated by the steepest descent method with exact line searches converge to a point $\mathbf{x}^*$ where the Hessian matrix $H(\mathbf{x}^*)$ is positive definite. Then*

$$f(\mathbf{x}_{k+1}) - f(\mathbf{x}^*) \leq \left(\frac{k_2(H(\mathbf{x}^*)) - 1}{k_2(H(\mathbf{x}^*)) + 1}\right)^2 (f(\mathbf{x}_k) - f(\mathbf{x}^*)).$$

[1] $k_2(A) = \|A\|_2 \|A^{-1}\|_2$. Being A SPD, if $\sigma(A) = \{\lambda_1, \ldots, \lambda_n\}$ is the spectre of A with $0 < \lambda_1 < \lambda_2 < \ldots < \lambda_n$, it holds $k_2(A) = \dfrac{\lambda_n}{\lambda_1}$. Indeed

$$\|A\|_2 \underbrace{=}_{\text{definition}} \sqrt{\rho(A^T A)} \underbrace{=}_{A \text{ is symmetric}} \sqrt{\rho(A^2)} = \sqrt{\rho(A)^2} = \sqrt{\lambda_n^2} \underbrace{=}_{\lambda_n > 0} \lambda_n.$$

The 2-norm of a SPD matrix is equal to its largest eigenvalue, so $\|A^{-1}\|_2 = \dfrac{1}{\lambda_1}$, and then

$$k_2(A) = \|A\|_2 \|A^{-1}\|_2 = \frac{\lambda_n}{\lambda_1}.$$

This second result tell us that the rate-of-convergence behavior of the steepest descent method is essentially the same on general nonlinear objective functions. Note however that in the theorem it is assumed that the step length is the global minimizer along the search direction, which may be not easy to compute as in the case of quadratic functions.

If a method converges linearly it exists $r \in (0, 1)$ such that

$$\frac{\|\mathbf{x}_{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}_k - \mathbf{x}^*\|} \leq r$$

for each k sufficiently large. The more the constant r is close to 0 the faster the method converges. In this case, the closer $\dfrac{k_2(A) - 1}{k_2(A) + 1}$ is to 0, the faster the method converges, i.e., the closer $k_2(A)$ is to 1, i.e. if A is well-conditioned.

Methods with linear convergence are in general not well suited for problems in which a high accuracy (low toll) is required, because they will need a large number of iterations to find the desired solution approximation.

Figure 3.1 shows possible sequences of iterations generated by the steepest descent method (or gradient method) applied to an elliptic quadratic function $q(\mathbf{x}) = q(x_1, x_2)$ (a quadratic function that has ellipses as level curves). The convergence depends on the choice of the starting guess and of the step length. This is particularly evident when the ellipses have the two focal points that are far from each other. In general, as the condition number of A increases, the contours of the quadratic become more elongated, the zigzagging in Figure 3.1 becomes more pronounced, and the theorem's result implies that the convergence degrades. If on the contrary the level curves are circles, the minimizer $\mathbf{x}^*$ is easily obtained in just one iteration: $\mathbf{x}_1 = \mathbf{x}^*$.

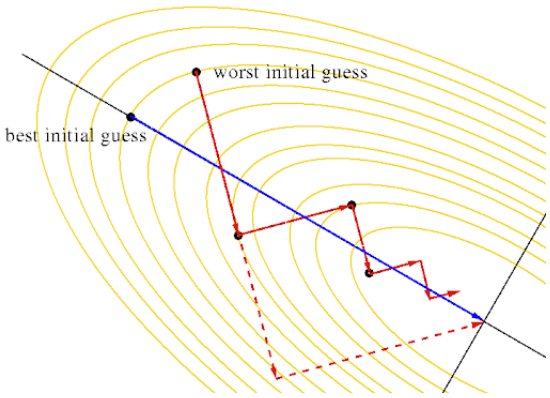


Figure 3.1: Sequence of iterations generated by the steepest descent method applied to a quadratic function. The convergence depends on the choice of the starting guess and of the step length.

3.2 Armijo and Wolfe conditions

Asking the simple decrease of f is not a sufficient condition to get a convergent method. We will require some other conditions on the step-length to avoid too small or too long steps. Let's see why these are needed by some examples.

Example 1 (Too long steps)

Let us consider $f(x) = x^2$, $x_0 = 2$, $p_k = (-1)^{k+1}$ (these are descent directions), $\alpha_k = 2 + \frac{3}{2^{k+1}}$. The sequence x_k built iteratively starting from x_0 setting $x_{k+1} = x_k + \alpha_k p_k$ is

$$x_k = (-1)^k \left(1 + \frac{1}{2^k}\right), \qquad k \in \mathbb{N}.$$

We can prove this by induction on k.

$$x_1 = x_0 + \alpha_0 p_0 = 2 + \left(2 + \frac{3}{2}\right)(-1) = -\frac{3}{2}.$$

If the thesis holds for k,

$$x_{k+1} = x_k + \alpha_k p_k = (-1)^k \left(1 + \frac{1}{2^k}\right) + \left(2 + \frac{3}{2^{k+1}}\right)(-1)^{k+1} =$$

$$= (-1)^{k+1} \left(-1 - \frac{1}{2^k} + 2 + \frac{3}{2^{k+1}}\right) = (-1)^{k+1} \left(1 - \frac{3-2}{2^{k+1}}\right) = (-1)^{k+1} \left(1 + \frac{1}{2^{k+1}}\right).$$

The simple decrease condition is satisfied because

$$f(x_{k+1}) = x_{k+1}^2 = \left(1 + \frac{1}{2^{k+1}}\right)^2 < \left(1 + \frac{1}{2^k}\right)^2 = x_k^2 = f(x_k),$$

but x_k does not converge to the minimizer of $f(x)$, $x^* = 0$, because

$$x_{2k} = 1 + \frac{1}{2^{2k}} \xrightarrow[k \to +\infty]{} 1, \qquad x_{2k+1} = -\left(1 + \frac{1}{2^{2k+1}}\right) \xrightarrow[k \to +\infty]{} -1.$$

The simple decrease condition is not sufficient to guarantee a convergence of $\{x_k\}$ to $x^* = 0$ because we have chosen a sequence of α_k with too large values.

Example 2 (Too short steps)

Let us consider $f(x) = x^2$, $x_0 = 2$, $p_k = -1$ (these are descent directions), $\alpha_k = \frac{1}{2^{k+1}}$. The sequence x_k built iteratively starting from x_0 setting $x_{k+1} = x_k + \alpha_k p_k$ is

$$x_k = 1 + \frac{1}{2^k}, \qquad k \in \mathbb{N}.$$

We can prove it by induction on k:

$$x_1 = x_0 + \alpha_0 p_0 = 2 + \frac{1}{2}(-1) = 1 + \frac{1}{2}.$$

If the thesis holds for k,

$$x_{k+1} = x_k + \alpha_k p_k = 1 + \frac{1}{2^k} + \frac{1}{2^{k+1}}(-1) = 1 + \frac{1}{2^k} - \frac{1}{2^{k+1}} = 1 + \frac{2-1}{2^{k+1}} = 1 + \frac{1}{2^{k+1}}.$$

The simple decrease condition is satisfied as

$$f(x_{k+1}) = x_{k+1}^2 = \left(1 + \frac{1}{2^{k+1}}\right)^2 < \left(1 + \frac{1}{2^k}\right)^2 = x_k^2 = f(x_k),$$

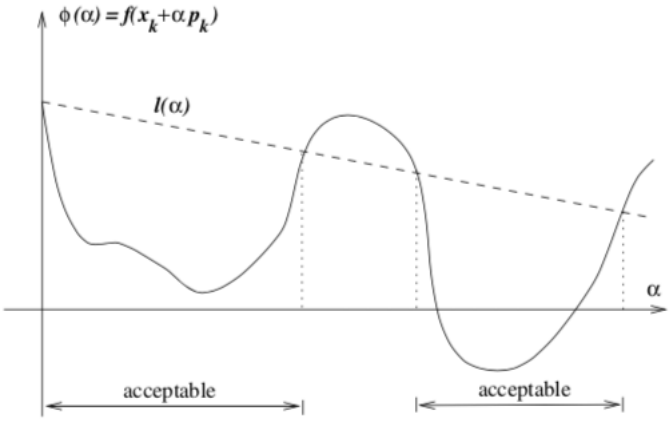


Figure 3.2: Parameters α that satisfy (A)

but x_k does not converge to $x^* = 0$ minimizer of $f(x)$ because

$$x_k = 1 + \frac{1}{2^k} \xrightarrow{k \to +\infty} 1.$$

The simple decrease condition is not sufficient to guarantee a convergence of $\{x_k\}$ to $x^* = 0$ because we have chosen a sequence of α_k with too small values.

We need then additional conditions.

Armijo rule

Armijo rule (A) requires that

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \leq f(\mathbf{x}_k) + \alpha_k c_1 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k, \quad c_1 \in (0,1), \tag{A}$$

usually $c_1 = 10^{-4}$. (A) is stronger than just asking the simple decrease $f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k)$ because $\nabla f(\mathbf{x}_k)^T \mathbf{p}_k < 0$.

Let

$$\varphi(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{p}_k),$$
$$\ell(\alpha) = f(\mathbf{x}_k) + \alpha c_1 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k,$$

Armijo rule requires $\varphi(\alpha)$ to be below the line $\ell(\alpha)$, i.e., that $\varphi(\alpha) \leq \ell(\alpha)$.

The slope of $\varphi(\alpha)$ is $\varphi'(\alpha) = \nabla f(\mathbf{x}_k + \alpha \mathbf{p}_k)^T \mathbf{p}_k$, for $\alpha_k = 0$ this is $\nabla f(\mathbf{x}_k)^T \mathbf{p}_k < 0$. The slope of $\ell(\alpha)$ is $c_1 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k = c_1 \varphi'(0) < 0$. Because $c_1 < 1$ and the two terms are negative, it follows

$$c_1 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k > \nabla f(\mathbf{x}_k)^T \mathbf{p}_k,$$

that is the line $\ell(\alpha)$ lies above the graph of φ for small positive values of (α).

Choosing α_k according to (A) avoids choosing α_k too large, as in Example 1. However, this condition alone is not sufficient to ensure the algorithm to make reasonable progress, because too small steps may be taken. We then introduce also the following condition, to rule out unacceptably small steps.

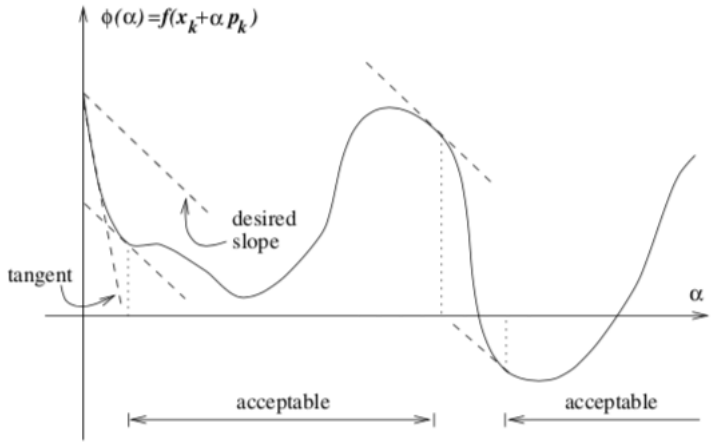


Figure 3.3: Parameters α that satisfy (W)

Wolfe rule

Wolfe rule requires that

$$\nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)^T \mathbf{p}_k \geq c_2 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k \quad c_2 \in (c_1, 1). \tag{W}$$

The first term $\nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)^T \mathbf{p}_k = \varphi'(\alpha_k)$ is the slope of $\varphi(\alpha)$. The condition requires this slope to be greater than the negative slope $c_2 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k$ that in Figure 3.3 is labelled as the desired slope. This ensures that the slope has been reduced sufficiently. Indeed, if the slope of $\varphi(\alpha_k)$ is strongly negative, it means that we can reduce f significantly by moving further along the chosen direction. On the other hand, if the slope is only slightly negative we cannot expect much more decrease in f in this direction and we can accept the α value. Usually $c_2 = 0.9$, for example when p_k is the Newton or quasi-Newton direction.

Because $c_1 < c_2 < 1$ and $\nabla f(\mathbf{x}_k)^T \mathbf{p}_k$ is negative it holds

$$c_1 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k > c_2 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k > \nabla f(\mathbf{x}_k)^T \mathbf{p}_k,$$

i.e. the desired slope is between those of $\ell(\alpha)$ and $\varphi(\alpha)$.

Choosing α_k satisfying (W) avoids choosing α_k too small, as it happens in Example 2.

Lemma 3.2.1. *Wolfe's Lemma*

Let $f : \mathbb{R}^n \to \mathbb{R}$ continuously differentiable and bounded below in $\{\mathbf{x}_k + \alpha \mathbf{p}_k \mid \alpha > 0\}$, with $\mathbf{p}_k$ a descent direction for f in $\mathbf{x}_k$, and let $c_1, c_2 : 0 < c_1 < c_2 < 1$.
It exists $I \subseteq (0, +\infty)$ non empty such that every $\alpha \in I$ satisfies (A) + (W).

Proof. Let $g(\alpha) = \varphi(\alpha) - \ell(\alpha)$. (A) requires that

$$g(\alpha) \leq 0.$$

Because

$$g(0) = \varphi(0) - \ell(0) = f(\mathbf{x}_k) - f(\mathbf{x}_k) = 0$$

and

$$g'(0) = \varphi'(0) - \ell'(0) = \nabla f(\mathbf{x}_k)^T \mathbf{p}_k - c_1 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k = \underbrace{(1 - c_1)}_{>0} \underbrace{\nabla f(\mathbf{x}_k)^T \mathbf{p}_k}_{<0} < 0,$$

it means that the value of g in zero is zero, and then g decreases. As $g \in C^0$ (because $f \in C^1$), it exists a right neighbourhood of 0 where $g(\alpha) < 0$. Let $\overline{\alpha}$ be the smallest positive zero of $g(\alpha)$.[2] It holds $g(\alpha) \leq 0$, $\forall \alpha \in [0, \overline{\alpha}]$, that is all the $\alpha \in [0, \overline{\alpha}]$ satisfy (A). In particular, in $\overline{\alpha}$ (A) is satisfied and it is an equality because $g(\overline{\alpha}) = 0$. From this it follows

$$f(\mathbf{x}_k + \overline{\alpha}\mathbf{p}_k) - f(\mathbf{x}_k) = c_1 \overline{\alpha} \nabla f(\mathbf{x}_k)^T \mathbf{p}_k. \tag{3.3}$$

For the mean value theorem applied to $\varphi(\alpha)$ in $[0, \overline{\alpha}]$, [3] it exists $\tilde{\alpha} \in (0, \overline{\alpha})$ such that

$$\varphi(\overline{\alpha}) - \varphi(0) = \overline{\alpha}\varphi'(\tilde{\alpha}),$$

that is

$$\overline{\alpha}\nabla f(\mathbf{x}_k + \tilde{\alpha}\mathbf{p}_k)^T \mathbf{p}_k = f(\mathbf{x}_k + \overline{\alpha}\mathbf{p}_k) - f(\mathbf{x}_k) \underbrace{=}_{(3.3)} c_1 \underbrace{\overline{\alpha}\nabla f(\mathbf{x}_k)^T \mathbf{p}_k}_{<0} \underbrace{>}_{0<c_1<c_2} c_2 \overline{\alpha}\nabla f(\mathbf{x}_k)^T \mathbf{p}_k.$$

Deleting $\overline{\alpha}$ we obtain

$$\nabla f(\mathbf{x}_k + \tilde{\alpha}\mathbf{p}_k)^T \mathbf{p}_k > c_2 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k,$$

that is $\tilde{\alpha}$ satisfies (W) without equality, so it exists a neighbourhood I_W of $\tilde{\alpha}$ where (W) is satisfied. Given that $\tilde{\alpha} < \overline{\alpha}$, in $I_W \cap [0, \overline{\alpha}] \neq \varnothing$ both criterion (A) e (W) are satisfied. □

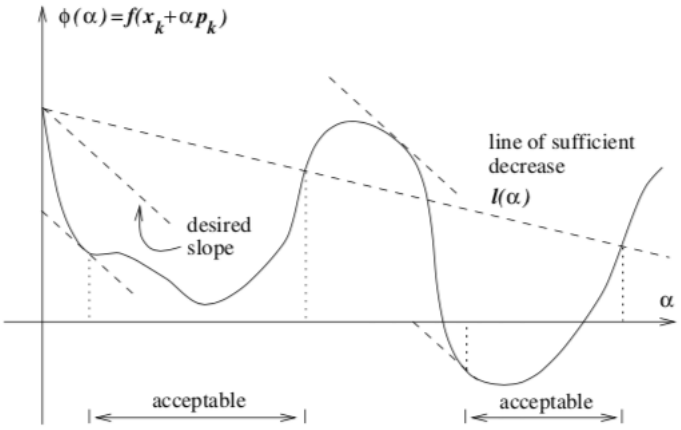


Figure 3.4: Parameters α that satisfy (A) e (W)

3.3 Convergence of line-search methods

To obtain global convergence, we must not only have well-chosen step lengths but also well- chosen search directions $\mathbf{p}_k$. We discuss requirements on the search direction in this section, focusing on one key property: the angle θ_k between $\mathbf{p}_k$ and the steepest descent direction $-\nabla f(\mathbf{x}_k)$.

[2] By assumption f is lower bounded in $\{\mathbf{x}_k + \alpha\mathbf{p}_k \mid \alpha \geq 0\}$, i.e., $\varphi(\alpha) = f(\mathbf{x}_k + \alpha\mathbf{p}_k)$ is lower bounded; then (see also Figure 3.3) it exists $\alpha > 0$, point in which $\varphi(\alpha)$ intersect the line $\ell(\alpha)$. So $g(\alpha)$ surely has a positive zero.

[3] By assumption $f \in C^1$ in $\{\mathbf{x}_k + \alpha\mathbf{p}_k \mid \alpha \geq 0\}$, i.e., $\varphi(\alpha) \in C^1([0, +\infty))$, so $\varphi'(\alpha)$ is continuous in $[0, +\infty)$, and we can apply the mean value theorem to $\varphi'(\alpha)$ in $[0, \overline{\alpha}]$.

Theorem 3.3.1. *Zoutendijk's theorem*

Let $\Omega = \{\mathbf{x} \in \mathbb{R}^n \mid f(\mathbf{x}) \leq f(\mathbf{x}_0)\}$, $f \in C^1(\Omega)$ and lower bounded on Ω, $\mathbf{p}_k$ a descent direction for f, and assume that α_k satisfies (A) and (W) and that $\nabla f(\mathbf{x})$ is Lipschitz continuous in Ω.
Let ϑ_k be the angle between $-\nabla f(\mathbf{x}_k)$ and $\mathbf{p}_k$, i.e. the angle such that

$$\cos(\vartheta_k) = -\frac{\nabla f(\mathbf{x}_k)^T \mathbf{p}_k}{\|\nabla f(\mathbf{x}_k)\| \|\mathbf{p}_k\|}.$$

The numerical series

$$\sum_{j=0}^{+\infty} \cos^2(\vartheta_j) \|\nabla f(\mathbf{x}_j)\|^2$$

is convergent.

Proof. Adding $-\nabla f(\mathbf{x}_k)^T \mathbf{p}_k$ to both members of (W) we obtain

$$\nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)^T \mathbf{p}_k - \nabla f(\mathbf{x}_k)^T \mathbf{p}_k \geq c_2 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k - \nabla f(\mathbf{x}_k)^T \mathbf{p}_k,$$

that is

$$\begin{aligned}(c_2 - 1)\nabla f(\mathbf{x}_k)^T \mathbf{p}_k &\leq \big(\nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) - \nabla f(\mathbf{x}_k)\big)^T \mathbf{p}_k \leq \|\nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) - \nabla f(\mathbf{x}_k)\| \|\mathbf{p}_k\| \\ &\leq L \|(\mathbf{x}_k + \alpha_k \mathbf{p}_k) - \mathbf{x}_k\| \|\mathbf{p}_k\| = L \alpha_k \|\mathbf{p}_k\|^2,\end{aligned}$$

which gives

$$\alpha_k \geq \frac{(c_2 - 1)\nabla f(\mathbf{x}_k)^T \mathbf{p}_k}{L \|\mathbf{p}_k\|^2}, \tag{3.4}$$

which is a positive quantity because $c_2 - 1 < 0$ and $\nabla f(\mathbf{x}_k)^T \mathbf{p}_k < 0$.

Note that

$$\begin{aligned}f(\mathbf{x}_{k+1}) &\underbrace{\leq}_{\text{(A)}} f(\mathbf{x}_k) + \alpha_k \underbrace{c_1 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k}_{<0} \underbrace{\leq}_{(3.4)} f(\mathbf{x}_k) + \frac{(c_2 - 1)c_1}{L} \frac{(\nabla f(\mathbf{x}_k)^T \mathbf{p}_k)^2}{\|\mathbf{p}_k\|^2} \\ &= f(\mathbf{x}_k) - c \frac{(\nabla f(\mathbf{x}_k)^T \mathbf{p}_k)^2}{\|\nabla f(\mathbf{x}_k)\|^2 \|\mathbf{p}_k\|^2} \|\nabla f(\mathbf{x}_k)\|^2 = f(\mathbf{x}_k) - c \cos^2(\vartheta_k) \|\nabla f(\mathbf{x}_k)\|^2,\end{aligned}$$

where $c = -\dfrac{(c_2 - 1)c_1}{L} > 0$. This holds for each α_j that satisfies the assumptions, so it holds $\forall j \leq k$:

$$f(\mathbf{x}_{j+1}) \leq f(\mathbf{x}_j) - c \cos^2(\vartheta_j) \|\nabla f(\mathbf{x}_j)\|^2.$$

We can then use it recursively

$$\begin{aligned}f(x_{k+1}) &\leq f(\mathbf{x}_k) - c \cos^2(\vartheta_k) \|\nabla f(\mathbf{x}_k)\|^2 \\ &\leq f(\mathbf{x}_{k-1}) - c \cos^2(\vartheta_{k-1}) \|\nabla f(\mathbf{x}_{k-1})\|^2 - c \cos^2(\vartheta_k) \|\nabla f(\mathbf{x}_k)\|^2 \leq \dots \\ &\leq f(\mathbf{x}_0) - c \sum_{j=0}^{k} \cos^2(\vartheta_j) \|\nabla f(\mathbf{x}_j)\|^2.\end{aligned}$$

We then have

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_0) - c \sum_{j=0}^{k} \cos^2(\vartheta_j) \|\nabla f(\mathbf{x}_j)\|^2,$$

that is

$$\sum_{j=0}^{k} \cos^2(\vartheta_j)\|\nabla f(\mathbf{x}_j)\|^2 \leq \frac{f(\mathbf{x}_0) - f(\mathbf{x}_{k+1})}{c}.$$

It holds

$$\begin{aligned}\sum_{j=0}^{+\infty} \cos^2(\vartheta_j)\|\nabla f(\mathbf{x}_j)\|^2 &= \lim_{k\to+\infty} \sum_{j=0}^{k} \cos^2(\vartheta_j)\|\nabla f(\mathbf{x}_j)\|^2 \leq \lim_{k\to+\infty} \frac{f(\mathbf{x}_0) - f(\mathbf{x}_{k+1})}{c} \\ &= \frac{f(\mathbf{x}_0)}{c} - \frac{1}{c}\lim_{k\to+\infty} f(\mathbf{x}_{k+1}) = (**).\end{aligned}$$

For the simple decrease condition (which is implied by (A)) and from the definition of Ω it holds $\mathbf{x}_k \in \Omega \quad \forall k \in \mathbb{N}$. This together with the assumption that f is lower bounded in Ω, implies that $\lim_{k\to+\infty} f(\mathbf{x}_{k+1}) \neq -\infty$, then

$$(**) \quad \neq +\infty.$$

This implies that $\sum_{j=0}^{+\infty} \cos^2(\vartheta_j)\|\nabla f(\mathbf{x}_j)\|^2$ is not divergent. As the series has positive terms it must converge. □

The fact that the series $\sum_{j=0}^{+\infty} \cos^2(\vartheta_j)\|\nabla f(\mathbf{x}_j)\|^2$ converges implies that

$$\lim_{k\to+\infty} \cos^2(\vartheta_k)\|\nabla f(\mathbf{x}_k)\|^2 = 0.$$

If the sequence $\{\cos(\vartheta_k)\}_k$ is bounded away from zero, we can deduce that

$$\lim_{k\to+\infty} \nabla f(\mathbf{x}_k) = \mathbf{0}.$$

In this case every accumulation point of $\{\mathbf{x}_k\}$ (if it exists) is a stationary point. Indeed, let $\widetilde{\mathbf{x}}$ be an accumulation point of $\{\mathbf{x}_k\}$, i.e., let $\widetilde{\mathbf{x}}$ be the limit point of a subsequence $\{\mathbf{x}_{k_j}\}$ of $\{\mathbf{x}_k\}$. Then

$$\nabla f(\widetilde{\mathbf{x}}) = \nabla f\left(\lim_{k_j\to+\infty} \mathbf{x}_{k_j}\right) = \lim_{k_j\to+\infty} \nabla f(\mathbf{x}_{k_j}) \underbrace{=}_{\lim_{k\to+\infty}\nabla f(\mathbf{x}_k)=\mathbf{0}} \mathbf{0}.$$

However it can happen that $\lim_{k\to+\infty} \cos(\vartheta_k) = 0$. This happens if

$$\lim_{k\to+\infty} \nabla f(\mathbf{x}_k)^T \mathbf{p}_k = 0,$$

that is if $\nabla f(\mathbf{x}_k)$ and $\mathbf{p}_k$ tend to be orthogonal. Formally $\nabla f(\mathbf{x}_k)^T \mathbf{p}_k$ is negative, and $\mathbf{p}_k$ remains a descent direction, but actually for k large $\nabla f(\mathbf{x}_k)^T \mathbf{p}_k$ is close to 0, i.e., $\mathbf{p}_k$ is a "poor" descent direction and along this direction the values of f are almost constant. This is a situation that can be avoided, choosing a descent direction $\mathbf{p}_k$ such that $\cos(\vartheta_k) > M$ for some $M > 0$ and for all k.

With the steepest descent method, as $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$, we have

$$\cos(\vartheta_k) = -\frac{\nabla f(\mathbf{x}_k)^T \mathbf{p}_k}{\|\nabla f(\mathbf{x}_k)\|\|\mathbf{p}_k\|} = \frac{\nabla f(\mathbf{x}_k)^T \nabla f(\mathbf{x}_k)}{\|\nabla f(\mathbf{x}_k)\|\|\nabla f(\mathbf{x}_k)\|} = 1.$$

Then, under the assumptions of Zoutendijk's Theorem, it holds $\lim_{k\to+\infty} \nabla f(\mathbf{x}_k) = \mathbf{0}$, as the possibility (ii) is excluded.
With Newton's and quasi-Newton methods, because $\mathbf{p}_k = -B_k^{-1}\nabla f(\mathbf{x}_k)$ (with $B_k = H(\mathbf{x}_k)$ in Newton's method or $B_k \approx H(\mathbf{x}_k)$ for quasi-Newton method) it holds

$$\cos(\vartheta_k) = -\frac{\nabla f(\mathbf{x}_k)^T \mathbf{p}_k}{\|\nabla f(\mathbf{x}_k)\|\|\mathbf{p}_k\|} = \frac{\nabla f(\mathbf{x}_k)^T B_k^{-1}\nabla f(\mathbf{x}_k)}{\|\nabla f(\mathbf{x}_k)\| \underbrace{\|B_k^{-1}\nabla f(\mathbf{x}_k)\|}_{\leq \|B_k^{-1}\|\|\nabla f(\mathbf{x}_k)\|}} \geq \frac{\nabla f(\mathbf{x}_k)^T B_k^{-1}\nabla f(\mathbf{x}_k)}{\|\nabla f(\mathbf{x}_k)\|^2\|B_k^{-1}\|}.$$

To bound this we need the following definition.

Definition 3.3.1. *The Rayleigh quotient. Given $A \in \mathbb{R}^{n\times n}$ a symmetric matrix and $\mathbf{v} \in \mathbb{R}^n$ we call Rayleigh quotient associated to them the scalar*

$$r_A(\mathbf{v}) = \frac{\mathbf{v}^T A\mathbf{v}}{\|\mathbf{v}\|^2}.$$

Notably, it holds that $\forall \mathbf{v} \in \mathbb{R}^n$

$$\lambda_{\min}(A) \leq r_A(\mathbf{v}) \leq \lambda_{\max}(A),$$

with $\lambda_{\min}(A)$ and $\lambda_{\max}(A)$ respectively the smallest and the largest eigenvalues of A.

Being B_k SPD we have that (cf. footnote at page 19) $\|B_k\| = \lambda_{\max}(B_k)$ and $\|B_k^{-1}\| = \lambda_{\max}(B_k^{-1}) = \dfrac{1}{\lambda_{\min}(B_k)}$. We then have that

$$\begin{aligned}\cos(\vartheta_k) &= r_{B_k^{-1}}(\nabla f(\mathbf{x}_k))\frac{1}{\|B_k^{-1}\|} = r_{B_k^{-1}}(\nabla f(\mathbf{x}_k))\frac{1}{\lambda_{\max}(B_k^{-1})} = r_{B_k^{-1}}(\nabla f(\mathbf{x}_k))\lambda_{\min}(B_k)\\ &\geq \lambda_{\min}(B_k^{-1})\lambda_{\min}(B_k) = \frac{\lambda_{\min}(B_k)}{\lambda_{\max}(B_k)} = \frac{1}{k_2(B_k)}.\end{aligned}$$

Then, if it exists M such that $k_2(B_k) < M$, we have $\cos(\vartheta_k) > 1/M$ and under the assumptions of Zoutendijk's Theorem, it holds $\lim_{k\to+\infty} \nabla f(\mathbf{x}_k) = \mathbf{0}$, being again excluded the situation (ii). This condition means that the approximations of the Hessian matrix have a uniformly bounded condition number (the bound is the same for any k).

3.4 Step-length selection algorithm: backtracking strategy

The algorithm for a generic line-search method can be sketched as follows:

0. Given $\mathbf{x}_0, f, \text{toll}$
1. For $k = 0, 1, \ldots$
   - 1.1 Choose $\mathbf{p}_k$ descent direction (may be the steepest descent direction, Newton's direction, quasi-Newton direction..)
   - 1.2 Find α_k that satisfies (A) and (W)
   - 1.3 Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
   - 1.4 If $\|\nabla f(\mathbf{x}_{k+1})\| \leq \text{toll}$ return $\mathbf{x}_{k+1}$ (approximation of $\mathbf{x}^*$) and stop

This algorithm is well-defined because the sequence $\nabla f(\mathbf{x}_k)$ converges to 0 for Zoutendijk's Theorem and because from Wolfe's Lemma it exists α_k that satisfies (A) and (W), as required at step 1.2. The question is how to compute such an α_k? We can use the backtracking technique. Even if we have seen that both (A) and (W) conditions are necessary for the convergence, this technique just checks the (A) condition, we will see later why.

The *backtracking strategy* is described in the following algorithm:

0. Given $\mathbf{x}_k, \alpha_0, \mathbf{p}_k, b_{\max}, c_1 \in (0,1), \gamma \in (0,1)$
1. $\alpha_k = \alpha_0$
2. For $b = 0, 1, \ldots, b_{\max}$
   1. If $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \leq f(\mathbf{x}_k) + \alpha_k c_1 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k$, i.e. if α_k satisfies (A), then set IND = 1 and stop,
      otherwise set $\alpha_k = \gamma \alpha_k$
3. Set IND = -1

This can be used at each iteration of the line-search algorithm (at step 1.2).

The backtracking algorithm works as follows: if $\alpha_k = \alpha_0$ does not satisfy (A), we reduce α_k by multiplication with γ and this is repeated until the new α_k satisfies (A) (the name backtracking is due to the fact that α_k if progressively reduced). In the proof of Wolfe's Lemma we have seen that it exists $\overline{\alpha} > 0$ such that each $\alpha \in [0, \overline{\alpha}]$ satisfies (A). Then we check if $\alpha_k = \alpha_0 \leq \overline{\alpha}$, if not we reduce α_k by multiplication with γ until the new $\alpha_k \leq \overline{\alpha}$. After a finite number of reductions we will obtain a α_k in $[0, \overline{\alpha}]$, so the backtracking technique never fails. However in the algorithm we decide to do at most $b_{\max}$ backtracking steps. If after $b_{\max}$ iterations $\alpha_k \leq \overline{\alpha}$ has not been found, it means that $\overline{\alpha}$ is too small, so that we will have to do really small steps in the line-search, which will lead to a really slow convergence.

It is then clear why in the algorithm we check just (A) and not (W): if $\overline{\alpha}$ is large, we get an α_k of the same order because while looking for α_k we go inside $[0, \overline{\alpha}]$ from above. If we have refused α_k because it does not satisfy (A) it means that the step is too long, then if $\gamma \alpha_k$ is accepted it cannot be too small (it is just a factor γ smaller than α_k). If $\overline{\alpha}$ is small, after $b_{\max}$ backtracking steps we will not find $\alpha_k < \overline{\alpha}$. It never happens to have a too small α_k and it would therefore be redundant to check (W).

The backtracking algorithm has the flag IND in output: if IND = 0 it means that an α_k that satisfies both (A) and (W) has been found, otherwise if IND = -1 it means that after $b_{\max}$ backtracking steps α_k has not been found and so the backtracking has failed.

Each iteration of backtracking algorithm costs a function evaluation, so the algorithm costs at most $b_{\max}$ function evaluations. Usually $b_{\max} = 20$.

In the backtracking algorithm one of the inputs is α_0, which can be chosen in various ways, depending on which directions are used in the line-search algorithm within which the backtracking is used.

If in the line-search the steepest descent direction is chosen, a popular choice is to choose α_0 by assuming that the first-order change in the function at iteration k will be the same as that obtained at the previous iteration. We then impose $\alpha_0 \nabla f(\mathbf{x}_k)^T p_k = \alpha_{k-1} \nabla f(\mathbf{x}_{k-1})^T p_{k-1}$ so that

$$\alpha_0 = \alpha_{k-1} \frac{\nabla f(\mathbf{x}_{k-1})^T p_{k-1}}{\nabla f(\mathbf{x}_k)^T p_k}$$

(α_{k-1} is the value used at the previous iteration). Another popular choice is the Barzilai-Borwein (BB) choice

$$\alpha_0 = \frac{\mathbf{s}_{k-1}^T \mathbf{s}_{k-1}}{\mathbf{s}_{k-1}^T \mathbf{y}_{k-1}}, \qquad \mathbf{s}_{k-1} = \mathbf{x}_k - \mathbf{x}_{k-1}, \quad \mathbf{y}_{k-1} = \nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_{k-1}),$$

that takes inspiration from the quasi-Newton methods, as we will see.
If in the line-search the Newton's or quasi-Newton's direction is chosen, we choose $\alpha_0 = 1$ (computing a Newton's step is expensive, we try to use it, if possible, to benefit from the quadratic convergence of pure Newton method).

We deduce then the algorithm for a line-search with backtracking technique:

0. Given $\mathbf{x}_0, f, k_{\max}, \text{toll}, b_{\max}, c_1 \in (0,1), \gamma \in (0,1)$
1. For $k = 0, 1, \ldots, k_{\max}$
   1. Choose $\mathbf{p}_k$ descent direction for f in $\mathbf{x}_k$
   2. Use the backtracking algorithm to find α_k
   3. If in the backtracking algorithm IND $= 0$ then set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$, otherwise stop and set FLAG $= -1$ (failure)
   4. If $\|\nabla f(\mathbf{x}_{k+1})\| \leq$ toll then stop, return $\mathbf{x}_{k+1}$ and FLAG $= 1$
2. Set FLAG $= -2$ (failure)

We could put the last $\mathbf{x}_k + \alpha_k \mathbf{p}_k$ and $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$ in output of the backtracking algorithm, to save a function evaluation.
Assume that the line-search algorithm has been equipped with a proper stopping criterion (based on a tolerance toll and on a maximum number of iterations $k_{\max}$). The line-search algorithm outputs the flag IND:

- FLAG $= 1$ an approximation has been computed with the desired accuracy;
- FLAG $= -1$ the backtracking technique failed;
- FLAG $= -2$ the stopping criterion was not satisfied within the maximum number of iterations (failure of line-search method)

3.5 Newton's method

With the expression "Newton's method" we usually refer to the pure Newton's method (described in Section 2.3) plus a line-search procedure to select the step length. The resulting method is globally convergent thanks to the line-search, and in a neighbourhood of $\mathbf{x}^*$ has a quadratic rate of convergence for the following reason: thanks to the global convergence property, the sequence $\|\nabla f(\mathbf{x}_k)\|$ converges to zero. Then all the accumulation points of $\{\mathbf{x}_k\}$ are stationary points for f. If it exists an accumulation point x^* of $\{\mathbf{x}_k\}$ which is a minimizer for f, then it exists $\overline{k} > 0$ such that for each $k \geq \overline{k}$, $\mathbf{x}_k$ enters in the ball $B_\rho(\mathbf{x}^*)$, region in which we have the quadratic convergence of Newton's method. We can also prove that it exists $\tilde{k}$ such that $\alpha_k = 1$ satisfies Armijo's condition (A) for each $k \geq \tilde{k}$; i.e.,

starting from the iterate $\tilde{k}$, Newton's method with line-search corresponds to pure Newton's method and it then inherits its local quadratic convergence (see Theorem 3.6.1 in the following section).

3.6 Quasi-Newton methods with line search

Let us now suppose that the search direction has the form

$$\mathbf{p}_k = -B_k^{-1}\nabla f(\mathbf{x}_k)$$

where the symmetric and positive definite matrix B_k is updated at every iteration by a quasi- Newton updating formula. We assume here that the step length α_k will be computed by an inexact line search that satisfies (A) and (W) starting with $\alpha_0 = 1$ (the method will then accept will accept $\alpha_0 = 1$ if this value if it satisfies the two conditions). This turns out to be crucial in obtaining a fast rate of convergence. The following result shows that if the search direction of a quasi-Newton method approximates the Newton direction well enough, then the unit step length will satisfy the conditions as the iterates converge to the solution. It also specifies a condition that the search direction must satisfy in order to give rise to a superlinearly convergent iteration.

Theorem 3.6.1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable in an open convex set $\mathcal{D}$ and assume that $H(\mathbf{x})$ is Lipschitz continuous in $\mathcal{D}$ with constant L. Consider a sequence $\{\mathbf{x}_k\}$ generated by $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$, where $\nabla f(\mathbf{x}_k)^T\mathbf{p}_k < 0$ for all k and α_k is chosen to satisfy (A) with $c_1 < \frac{1}{2}$ and (W). If $\{\mathbf{x}_k\}$ converges to a point $\mathbf{x}^* \in \mathcal{D}$ at which $H(\mathbf{x}^*)$ is positive definite, and if*

$$\lim_{k\to\infty} \frac{\|\nabla f(\mathbf{x}_k) + H(\mathbf{x}_k)\mathbf{p}_k\|}{\|\mathbf{p}_k\|} = 0, \tag{3.5}$$

then

a) there is an index $k_0 \geq 0$ such that for all $k \geq k_0$, $\alpha_k = 1$ is admissible,

b) if $\alpha_k = 1$ for all $k > k_0$, $\nabla f(\mathbf{x}^*) = 0$ and $\{\mathbf{x}_k\}$ converges superlinearly to $\mathbf{x}^*$.

For the proof cf. Theorem 6.3.4 in the book of Dennis and Schnabel.

It is easy to see that if $c_1 > \frac{1}{2}$, then the line search would exclude the minimizer of a quadratic and unit step lengths may not be admissible. Notice that (3.5) is always satisfied for Newton's method. So, when the Newton's direction is employed, the (A) and (W) conditions will accept the step length $\alpha_k = 1$ for all large k.

If $\mathbf{p}_k$ is a quasi-Newton search direction, then (3.5) is equivalent to

$$\lim_{k\to\infty} \frac{\|(B_k - H(\mathbf{x}^*))\mathbf{p}_k\|}{\|\mathbf{p}_k\|} = 0. \tag{3.6}$$

Hence, we have the surprising (and delightful) result that a superlinear convergence rate can be attained even if the sequence of quasi-Newton matrices B_k does not converge to $H(\mathbf{x}^*)$, it suffices that the B_k become increasingly accurate approximations to $H(\mathbf{x}^*)$ along the search directions $\mathbf{p}_k$. An important remark is that condition (3.6) is both necessary and sufficient for the superlinear convergence of quasi-Newton methods.

Theorem 3.6.2. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable in an open convex set $\mathcal{D}$ and assume that $H(\mathbf{x})$ is Lipschitz continuous in $\mathcal{D}$ with constant L. Consider the iteration $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$ (that is, the step length α_k is uniformly 1) and that $\mathbf{p}_k$ is given by $-B_k^{-1}\nabla f(\mathbf{x}_k)$. Let us also assume that $\{\mathbf{x}_k\}$ converges to a point $\mathbf{x}^*$ such that $\nabla f(\mathbf{x}^*) = 0$ and $H(\mathbf{x}^*)$ is positive definite. Then $\{\mathbf{x}_k\}$ converges superlinearly if and only if* (3.6) *holds.*

Practice time

Implement the steepest descent and the Newton's method with backtracking line-search. Use them to minimize the Rosenbrock and the Rastrigin functions. First try the initial point $\mathbf{x}_0 = [1.2, 1.2]^T$ and then the more difficult point $\mathbf{x}_0 = [-1.2, 1]^T$.

a) How does the results change from the case without linesearch? Can you explain why?

b) Plot the value of the step length along the iterations, is it often reduced? Count the total number of function values and compare it to the case without linesearch.

c) Try to use the different values of α_0 reported in the notes for the gradient descent.

Chapter 4

Quasi-Newton methods

Quasi-Newton methods are based on the same model of Newton's method but instead of the exact Hessian $H(\mathbf{x}_k)$ we use an SPD approximation B_k. These can be built in various ways and each choice corresponds to a different sequence of quasi-Newton directions and so to a different quasi-Newton method.
A generic quasi-Newton method is based on the following iterative scheme:

$$\left\{ \begin{array}{l} \text{Given } \mathbf{x}_0 \\ \mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \end{array} \right\},$$

where $\mathbf{p}_k$ is such that $B_k \mathbf{p}_k = -\nabla f(\mathbf{x}_k)$, for some approximation B_k SPD of the Hessian matrix. It is a locally convergent method that can be coupled with a line-search to become a globally convergent method and has a superlinear local rate of convergence (this is the price to be paid to avoid computation of second order derivatives). How to compute B_k?

4.1 Finite differences

We could approximate the Hessian of f by finite differences. Let us introduce the finite differences in the easy case $f : \mathbb{R} \to \mathbb{R}$. We know that by definition

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

so we can expect that for h small enough $\dfrac{\Delta f}{h} := \dfrac{f(x+h) - f(x)}{h}$ will be a good approximation to the first order derivative. These are called the *forward finite differences*. We can show (see TD 4) that

$$\left| f'(x) - \frac{\Delta f}{h} \right| = O(h).$$

To get a good approximation in practice it is however crucial to select a good value of h: for too large values this approximation error will be large, but for too small values we will have rounding errors that can lead to numerical cancellation, cf. `https://nhigham.com/2020/10/06/what-is-the-complex-step-approximat`
As the total error on our approximation will be

$$\left| f'(x) - fl\left(\frac{\Delta f}{h} \right) \right|$$

where $fl(x)$ is the floating point representation of x, we then need to find a good balance between two sources of errors that have opposite behaviour with respect to h:

- the approximation error $\left|f'(x) - \frac{\Delta f}{h}\right|$ that increases with h,
- the rounding error $\left|fl\left(\frac{\Delta f}{h}\right) - \frac{\Delta f}{h}\right|$ that decreases with h.

We will prove in TD 4 that a good compromise between this two sources of errors is given by $h \sim \sqrt{u}$ where u is the machine precision. In order to improve this approximation we can use *central finite differences*:

$$f'(x) \sim \frac{\delta f}{h} := \frac{f(x + h/2) - f(x - h/2)}{h}$$

and it holds

$$\left|f'(x) - \frac{\delta f}{h}\right| = O(h^2).$$

This technique can be used for n-dimensional functions to approximate the gradient:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(\mathbf{x}) \\ \vdots \\ \frac{\partial f}{\partial x_n}(\mathbf{x}) \end{pmatrix} \sim \begin{pmatrix} \frac{f(\mathbf{x} + h\mathbf{e}_1) - f(\mathbf{x})}{h} \\ \vdots \\ \frac{f(\mathbf{x} + h\mathbf{e}_n) - f(\mathbf{x})}{h} \end{pmatrix}$$

where $\mathbf{e}_i$ is the i-th vector of the canonical basis ($\mathbf{e}_i(i) = 1$, $\mathbf{e}_i(j) = 0$ for $j \neq i$). Notice that this approximation requires $n+1$ function evaluations, it can therefore be expensive for n large (the central differences give a more accurate expression but they require $2n$ function evaluations).

Repeating the above procedure the Hessian also can be approximated in this way:

$$H(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(\mathbf{x}) \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n}(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial x_1}(\nabla f(\mathbf{x})^T) \\ \vdots \\ \frac{\partial}{x_n}(\nabla f(\mathbf{x})^T) \end{pmatrix} \sim \begin{pmatrix} \frac{(\nabla f(\mathbf{x} + h\mathbf{e}_1) - \nabla f(\mathbf{x}))^T}{h} \\ \vdots \\ \frac{(\nabla f(\mathbf{x} + h\mathbf{e}_n) - \nabla f(\mathbf{x}))^T}{h} \end{pmatrix}$$

Notice that this approximation is quite expensive: it requires $n+1$ gradient evaluations, each of which requires $n+1$ function evaluations, so the cost is $O(n^2)$ function evaluations, which may be prohibitive for large n.

It is then too expensive to build a new B_k at each iteration: we need to exploit the informations obtained in the previous iterations. Assuming to have finished iteration k and to have computed B_k and $\mathbf{x}_{k+1}$, the idea of quasi-Newton methods is to avoid building an approximation B_{k+1} ex-novo and to rather obtain B_{k+1} from an update of B_k which preserves the symmetry and the positive definiteness.

4.2 How to update B_k

Let us assume to be at the end of iteration k, i.e. to have computed $\mathbf{x}_k$, B_k, α_k, $\mathbf{p}_k = -B_k^{-1}\nabla f(\mathbf{x}_k)$, $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$. We need to compute B_{k+1}.

The idea of Quasi-Newton method is to impose the so-called secant equation:

$$\boxed{B_{k+1}\mathbf{s}_k = \mathbf{y}_k.}$$

where

$$\mathbf{s}_k := \mathbf{x}_{k+1} - \mathbf{x}_k, \qquad \mathbf{y}_k := \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k).$$

This condition can be seen as the extension to the n-dimensional case of the finite difference approximation of the second order derivative. For $f : \mathbb{R} \to \mathbb{R}$, using backward differences, it holds:

$$f''(x) \sim \frac{f'(x) - f'(x-h)}{h}$$

so that

$$f''(x_{k+1}) \sim \frac{f'(x_{k+1}) - f'(x_k)}{x_{k+1} - x_k}.$$

Replacing the second order derivative with the matrix B_{k+1} and the first order derivatives with the gradients, we understand where our secant condition comes from.

Anyway this equation only gives n conditions, which are not sufficient to univocally determine the n^2 coefficients of B_{k+1} (actually B_{k+1} is symmetric, so it has just $\frac{(n-1)n}{2}$ degrees of freedom, because the coefficients of the upper triangular part are equal to those of the lower triangular part). We need to impose other conditions: we ask that B_{k+1} is SPD.

Remark 4.2.1. *If it exists B_{k+1} SPD that satisfies the secant equation*

$$B_{k+1}\mathbf{s}_k = \mathbf{y}_k,$$

then the curvature condition is also satisfied

$$\mathbf{y}_k^T \mathbf{s}_k > 0.$$

Proof. By contradiction, let $\mathbf{y}_k^T \mathbf{s}_k \leq 0$, then

$$0 \geq \mathbf{y}_k^T \mathbf{s}_k \underbrace{=}_{\mathbf{y}_k = B_{k+1}\mathbf{s}_k} (B_{k+1}\mathbf{s}_k)^T \mathbf{s}_k = \mathbf{s}_k^T B_{k+1}^T \mathbf{s}_k = \mathbf{s}_k^T B_{k+1} \mathbf{s}_k,$$

which is impossible because B_{k+1} is positive definite by assumption. □

The curvature condition is not always satisfied for nonconvex functions, but it can be enforced imposing restrictions on the line-search, for example this is implied by the Wolfe condition:

Remark 4.2.2. *If α_k satisfies the Wolfe condition*

$$\nabla f(\mathbf{x}_k + \alpha \mathbf{p}_k)^T \mathbf{p}_k \geq c_2 \nabla f(\mathbf{x}_k)^T \mathbf{p}_k$$

for some $c_2 \in (0, 1)$, then the curvature condition $\mathbf{y}_k^T \mathbf{s}_k > 0$ is satisfied.

Proof. The proof is given in TD4. □

From Remark 4.2.1, if the curvature condition is not satisfied, it cannot exist B_{k+1} SPD that satisfies the secant equation. In a practical implementation of the algorithm is then advisable to verify if the curvature condition is verified. From Remark 4.2.2, we could check this condition by verifying that (W) is satisfied. However in line-search algorithm with backtrackinging technique (W) is not checked so usually we directly check the curvature condition.
These conditions are still not enough: asking that B_{k+1} is SPD imposes n additional inequalities (all leading principal minors must be positive[1]); there are still some degrees of freedom left.
To determine B_{k+1} univocally, we choose B_{k+1} as the matrix, among all the symmetric matrices that satisfy the secant equation, closer (in some sense) to B_k: we ask that

$$B_{k+1} = \text{argmin}\{\|B - B_k\| \mid B = B^T, B\mathbf{s}_k = \mathbf{y}_k\},$$

where we know that B_k is SPD and that the curvature condition $\mathbf{y}_k^T\mathbf{s}_k > 0$ is satisfied.
To solve this problem we can choose various matrix norms, and each of them leads to a different quasi-Newton method. The norm that makes the solution easier is the weighted Frobenius norm (cf. B.5.5) with weight W SPD such that $W\mathbf{y}_k = \mathbf{s}_k$. For example we can choose, assuming $H(\mathbf{x})$ to be positive definite in $[\mathbf{x}_k, \mathbf{x}_k + \alpha_k\mathbf{p}_k]$, $W = \bar{G}_k^{-1}$, where $\bar{G}_k$ is the average Hessian matrix

$$\bar{G}_k = \int_0^1 H(\mathbf{x}_k + t\alpha_k\mathbf{p}_k)\, \mathrm{d}t. \tag{4.1}$$

W is such that $W\mathbf{y}_k = \mathbf{s}_k$ because $\bar{G}_k^{-1}\mathbf{y}_k = \mathbf{s}_k$ as

$$\begin{aligned}\mathbf{y}_k &= \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k) = \Big[\nabla f(\mathbf{x}_k + t(\mathbf{x}_{k+1} - \mathbf{x}_k))\Big]_{t=0}^{t=1} \\ &= \int_0^1 H(\mathbf{x}_k + t(\mathbf{x}_{k+1} - \mathbf{x}_k))(\mathbf{x}_{k+1} - \mathbf{x}_k)\, \mathrm{d}t = \int_0^1 H(\mathbf{x}_k + t\alpha_k\mathbf{p}_k)\, \mathrm{d}t\, \mathbf{s}_k = \bar{G}_k\mathbf{s}_k.\end{aligned}$$

With this choice, the unique solution of the problem is (see TD4)

$$B_{k+1} = (I - \rho_k\mathbf{y}_k\mathbf{s}_k^T)B_k(I - \rho_k\mathbf{s}_k\mathbf{y}_k^T) + \rho_k\mathbf{y}_k\mathbf{y}_k^T, \quad \rho_k = \frac{1}{\mathbf{y}_k^T\mathbf{s}_k} > 0. \tag{DFP}$$

We can prove that if B_k is SPD then also B_{k+1} is SPD (see TD4).
This formula is called DFP because it was discovered empirically by physician Davidon in 1959, and in 1963 the mathematicians Fletcher and Powell explained rigorously why this update technique works: they understood that B_{k+1} was the solution of this minimum problem.
DFP is an update formula, because it allows to build B_{k+1} from B_k, $\nabla f(\mathbf{x}_{k+1})$ and $\nabla f(\mathbf{x}_k)$: the basic idea of quasi-Newton methods is indeed that of avoiding building ex-novo a matrix B_{k+1} to approximate $H(\mathbf{x}_{k+1})$, and rather to obtain B_{k+1} updating B_k (preserving the symmetry and the positive definiteness) by using the informations $\nabla f(\mathbf{x}_{k+1})$ and $\nabla f(\mathbf{x}_k)$.
It is possible to show that

$$B_{k+1} = B_k + \gamma\mathbf{u}\mathbf{u}^T + \delta\mathbf{v}\mathbf{v}^T$$

[1] A minor of A of order k is principal if it is obtained by deleting $n-k$ rows and the $n-k$ columns with the same numbers. For instance, in a principal minor where you have deleted row 1 and 3, you should also delete column 1 and 3. The leading principal minor of A of order k is the minor of order k obtained by deleting the last $n-k$ rows and columns. We denote D_k the determinant of the leading principal minor of order k. There are n leading principal minors and $\binom{n}{k}$ principal minors of order k. Let A be a symmetric $n \times n$ matrix. Then we have: A is positive definite $\iff$ $D_k > 0$ for all leading principal minors.

for

$$\gamma = \frac{1}{\mathbf{y}_k^T \mathbf{s}_k}, \quad \mathbf{u} = \mathbf{y}_k,$$
$$\delta = -\frac{1}{\mathbf{s}_k^T B_k \mathbf{s}_k}, \quad \mathbf{v} = B_k \mathbf{s}_k.$$

Then B_{k+1} is obtained from B_k by adding two rank 1 matrices (globally we obtain then B_{k+1} by a rank 2 modification of B_k). We can then use the Sherman-Morrison-Woodbury formula [2] to compute B_{k+1}^{-1} from B_k^{-1}:

$$B_{k+1}^{-1} = B_k^{-1} - \frac{B_k^{-1}\mathbf{y}_k\mathbf{y}_k^T B_k^{-1}}{\mathbf{y}_k^T B_k^{-1}\mathbf{y}_k} + \frac{\mathbf{s}_k\mathbf{s}_k^T}{\mathbf{y}_k^T\mathbf{s}_k}.$$

If we build the sequence B_k starting from a matrix B_0 of which the inverse B_0^{-1} is known, then using the prevoius formula, we can compute all the B_k^{-1} by performing just matrix-vector products, and we can also compute $\mathbf{p}_k$ by a matrix-vector product $\mathbf{p}_k = -B_k^{-1}\nabla f(\mathbf{x}_k)$ and we do not need to solve the linear system $B_k\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$.

The DFP updating formula is quite effective, but it was soon superseded by the BFGS formula, which is presently considered to be the most effective of all quasi-Newton updating formulae. It was proposed by Broyden, Fletcher, Goldfarb, Shanno. The BFGS update formula instead of approximating the Hessian matrix, imposes analogous conditions on approximations $\tilde{B}_k$ of the inverse of the Hessian matrix.

Let us assume to be at the and of iteration k, we have computed $\mathbf{x}_k$, $\tilde{B}_k$, α_k, $\mathbf{p}_k = -\tilde{B}_k\nabla f(\mathbf{x}_k)$, $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$. We need to compute $\tilde{B}_{k+1}$. To determine $\tilde{B}_{k+1}$ univocally, we ask $\tilde{B}_{k+1}$ to be, among all the symmetric matrices that satisfy the equation $\tilde{B}\mathbf{y}_k = \mathbf{s}_k$, the closest matrix to $\tilde{B}_k$:

$$\tilde{B}_{k+1} = \text{argmin}\{\|\tilde{B} - \tilde{B}_k\| \mid \tilde{B} = \tilde{B}^T, \tilde{B}\mathbf{y}_k = \mathbf{s}_k\},$$

where we know that $\tilde{B}_k$ is SPD and the curvature condition $\mathbf{y}_k^T\mathbf{s}_k > 0$ holds.
We choose again the Frobenius norm with weight W such that $W\mathbf{s}_k = \mathbf{y}_k$. Also in this case we can choose $W = \bar{G}_k$ (it holds $W\mathbf{s}_k = \mathbf{y}_k$). With this choice, the unique solution of the minimization problem is

$$\tilde{B}_{k+1} = (I - \rho_k\mathbf{s}_k\mathbf{y}_k^T)\tilde{B}_k(I - \rho_k\mathbf{y}_k\mathbf{s}_k^T) + \rho_k\mathbf{s}_k\mathbf{s}_k^T, \quad \rho_k = \frac{1}{\mathbf{y}_k^T\mathbf{s}_k} > 0. \tag{BFGS}$$

We can prove that if $\tilde{B}_k$ is SPD then also $\tilde{B}_{k+1}$ is SPD. We will then have a positive definite matrix, even if this has not been explicitly imposed in the problem's formulation.

We can derive a version of the BFGS algorithm that works with the Hessian approximation B_k rather than its inverse $\tilde{B}_k$. The update formula for B_k is obtained by simply applying the Sherman-Morrison-Woodbury formula to obtain

$$B_{k+1} = B_k - \frac{B_k\mathbf{s}_k\mathbf{s}_k^T B_k}{\mathbf{s}_k^T B_k\mathbf{s}_k} + \frac{\mathbf{y}_k\mathbf{y}_k^T}{\mathbf{y}_k^T\mathbf{s}_k}. \tag{4.2}$$

[2] Let $A \in \mathbb{R}^{n\times n}$ be invertible. Let $\overline{A}$ a matrix obtained from A by addition of a rank 1 matrix

$$\overline{A} = A + \mathbf{a}\mathbf{b}^T$$

with $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$. If $\overline{A}$ is invertible, then we can compute $\overline{A}^{-1}$ from A^{-1} and by doing just some matrix-vector products:

$$\overline{A}^{-1} = A^{-1} - \frac{A^{-1}\mathbf{a}\mathbf{b}^T A^{-1}}{1 + \mathbf{b}^T A^{-1}\mathbf{a}}.$$

Table 4.1: Gradient descent, BFGS and Newton methods applied to the Rosenbrock function, values of $\|\mathbf{x}_k - \mathbf{x}^*\|$ at last iterations.

GD	BFGS	Newton
1.827 e-4	1.70 e-3	3.48e-2
1.826 e-4	1.17 e-3	1.44e-2
1.824 e-4	1.34 e-4	1.82e-4
1.823 e-4	1.01 e-6	1.17e-8

The last question is how to choose $\tilde{B}_0$? Unlucky there is not a general formula that works well in all cases. Usually we choose $\tilde{B}_0 = I_n$, or $\tilde{B}_0 = \gamma I_n$ with $\gamma = \dfrac{\mathbf{y}_0^T \mathbf{s}_0}{\mathbf{y}_0^T \mathbf{y}_0}$, or $\tilde{B}_0 = B_0^{-1}$ after having computed an approximation B_0 of $H(\mathbf{x}_0)$ by finite differences.

Each iteration of quasi-Newton methods can be performed at a cost of $O(n^2)$ arithmetic operations (plus the cost of function and gradient evaluations); there are no $O(n^3)$ operations such as linear system solves or matrix-matrix operations. The algorithm is robust, and its rate of convergence is superlinear, which is fast enough for most practical purposes. Even though Newton's method converges more rapidly (that is, quadratically), its cost per iteration is higher because it requires the solution of a linear system. A more important advantage for BFGS is, of course, that it does not require calculation of second derivatives.

We describe the algorithm of the BFGS method.

Given $\mathbf{x}_0$, toll > 0, inverse Hessian approximation $\tilde{B}_0$, set $k = 0$.

While $\|\nabla f(\mathbf{x}_k)\| >$ toll

1. Compute the search direction $\mathbf{p}_k = -\tilde{B}_k \nabla f(\mathbf{x}_k)$.
2. Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ where α_k is computed from a line search procedure to satisfy (A)+(W)
3. Define $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$
4. Compute $\tilde{B}_{k+1}$ by means of (BFGS)
5. Set $k = k + 1$

We provide here and example of the application of the methods we have seen (Gradient descent, BFGS and Newton) to have an idea of their different rate of convergence. We consider the Rosenbrock function[3], the starting point $(-1.2, 1)$ and we choose the stopping criterion $\|\nabla f(\mathbf{x}_k)\| \leq \text{toll} = 10^{-5}$. We report in Table 4.2 the values of the error $\|\mathbf{x}_k - \mathbf{x}^*\|$ in the final iterations of each method.

According to the theory, close to the solution the Newton's method exhibits a fast quadratic convergence ($\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \sim \|\mathbf{x}_k - \mathbf{x}^*\|^2$), the gradient method has a slow linear convergence while BFGS is a good compromise between the two other methods.

4.3 Global convergence of the BFGS method

We study the global convergence of BFGS, with a practical line search, when applied to a smooth convex function from an arbitrary starting point $\mathbf{x}_0$ and from any initial Hessian approximation B_0

[3] `https://en.wikipedia.org/wiki/Rosenbrock_function`

that is symmetric and positive definite.

We make the following assumption.

Assumption 4.3.1. *We assume that*

1. *The objective function f is twice continuously differentiable.*
2. *The level set $\mathcal{L} = \{\mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}) \leq f(\mathbf{x}_0)\}$ is convex, and there exist positive constants m and M such that*
$$m\|\mathbf{z}\|^2 \leq \mathbf{z}^T H(\mathbf{x})\mathbf{z} \leq M\|\mathbf{z}\|^2 \tag{4.3}$$
for all $\mathbf{z} \in \mathbb{R}^n$ and $\mathbf{x} \in \mathcal{L}$.

The second part of this assumption implies that $H(\mathbf{x})$ is positive definite on $\mathcal{L}$ and that f has a unique minimizer $\mathbf{x}^*$. We have seen that $\mathbf{y}_k = \bar{G}_k \mathbf{s}_k$, where $\bar{G}_k$ is the average Hessian defined in (4.1). From this and (4.3) we obtain
$$\frac{\mathbf{y}_k^T \mathbf{s}_k}{\mathbf{s}_k^T \mathbf{s}_k} = \frac{\mathbf{s}_k^T \bar{G}_k \mathbf{s}_k}{\mathbf{s}_k^T \mathbf{s}_k} \geq m. \tag{4.4}$$
From the assumption $\bar{G}_k$ is positive definite, so its square root is well-defined. Therefore, we have by defining $\mathbf{z}_k = \bar{G}_k^{1/2} \mathbf{s}_k$ that $\mathbf{y}_k = \bar{G}_k \mathbf{s}_k = \bar{G}_k^{1/2} \bar{G}_k^{1/2} \mathbf{s}_k = \bar{G}_k^{1/2} \mathbf{z}_k$ and
$$\frac{\mathbf{y}_k^T \mathbf{y}_k}{\mathbf{y}_k^T \mathbf{s}_k} = \frac{\mathbf{z}_k^T \bar{G}_k \mathbf{z}_k}{\mathbf{z}_k^T \mathbf{z}_k} \leq M. \tag{4.5}$$
We are now ready to present the global convergence result for the BFGS method.

Theorem 4.3.1. *Let B_0 be any symmetric positive definite initial matrix, and let $\mathbf{x}_0$ be a starting point for which Assumption 4.3.1 is satisfied. Then the sequence $\{\mathbf{x}_k\}$ generated by BFGS Algorithm converges to the minimizer $\mathbf{x}^*$ of f .*

Proof. Some points of the proof (marked by "see TD") are treated in TD 4.

Let us define
$$m_k = \frac{\mathbf{y}_k^T \mathbf{s}_k}{\mathbf{s}_k^T \mathbf{s}_k}, \qquad M_k = \frac{\mathbf{y}_k^T \mathbf{y}_k}{\mathbf{y}_k^T \mathbf{s}_k} \tag{4.6}$$
and note from (4.4) and (4.5) that
$$m_k \geq m, \quad M_k \leq M. \tag{4.7}$$
By computing the trace of the BFGS approximation (4.2), we obtain that (see TD)
$$\mathrm{trace}(\mathrm{B_{k+1}}) = \mathrm{trace}(\mathrm{B_k}) - \frac{\|\mathrm{B_k s_k}\|^2}{\mathrm{s_k^T B_k s_k}} + \frac{\|\mathrm{y_k}\|^2}{\mathrm{y_k^T s_k}}.$$
Let us also define
$$\cos(\theta_k) = \frac{\mathbf{s}_k^T B_k \mathbf{s}_k}{\|\mathbf{s}_k\| \|B_k \mathbf{s}_k\|}, \quad q_k = \frac{\mathbf{s}_k^T B_k \mathbf{s}_k}{\mathbf{s}_k^T \mathbf{s}_k}, \tag{4.8}$$
so that θ_k is the angle between $\mathbf{s}_k$ and $B_k \mathbf{s}_k$. We then obtain that
$$\frac{\|B_k \mathbf{s}_k\|^2}{\mathbf{s}_k^T B_k \mathbf{s}_k} = \frac{\|B_k \mathbf{s}_k\|^2 \|\mathbf{s}_k\|^2}{(\mathbf{s}_k^T B_k \mathbf{s}_k)^2} \frac{\mathbf{s}_k^T B_k \mathbf{s}_k}{\|\mathbf{s}_k\|^2} = \frac{q_k}{\cos^2(\theta_k)}$$

and

$$\mathrm{trace}(\mathrm{B_{k+1}}) = \mathrm{trace}(\mathrm{B_k}) - \frac{\mathrm{q_k}}{\cos^2(\theta_\mathrm{k})} + \mathrm{M_k}. \tag{4.9}$$

We can also show (see TD) that

$$\det(\mathrm{B_{k+1}}) = \det(\mathrm{B_k})\frac{\mathbf{y}_\mathrm{k}^\mathrm{T}\mathbf{s}_\mathrm{k}}{\mathbf{s}_\mathrm{k}^\mathrm{T}\mathrm{B_k}\mathbf{s}_\mathrm{k}}. \tag{4.10}$$

In addition, we have from (4.6) and (4.8) that

$$\det(B_{k+1}) = \det(B_k)\frac{\mathbf{y}_k^T\mathbf{s}_k}{\mathbf{s}_k^T\mathbf{s}_k}\frac{\mathbf{s}_k^T\mathbf{s}_k}{\mathbf{s}_k^T B_k\mathbf{s}_k} = \det(B_k)\frac{m_k}{q_k}. \tag{4.11}$$

We now combine the trace and determinant by introducing the following function of a positive definite matrix B:

$$\psi(B) = \mathrm{trace}(B) - \ln(\det(B)) \tag{4.12}$$

where $\ln(\cdot)$ denotes the natural logarithm. It is not difficult to show that $\psi(B) > 0$ (see TD). By using (4.6) and (4.9)-(4.12), we have that

$$\begin{aligned}
\psi(B_{k+1}) =& \mathrm{trace}(B_{k+1}) - \ln(\det(B_{k+1})) \\
=& \mathrm{trace}(B_k) + M_k - \frac{q_k}{\cos^2(\theta_k)} - \ln(\det(B_k)) - \ln m_k + \ln q_k \\
=& \psi(B_k) + M_k - \frac{q_k}{\cos^2(\theta_k)} - \ln m_k + \ln q_k \\
=& \psi(B_k) + (M_k - \ln(m_k) - 1) \\
&+ \left[1 - \frac{q_k}{\cos^2(\theta_k)} + \ln\left(\frac{q_k}{\cos^2(\theta_k)}\right)\right] + \ln(\cos^2(\theta_k))
\end{aligned} \tag{4.13}$$

Now, since the function $h(t) = 1 - t + \ln(t) \leq 0$ is nonpositive for all $t > 0$ (see TD), the term inside the square brackets is nonpositive, and thus from (4.7) and (4.13) we have, defining $c = M - \ln(m) - 1$, that

$$\begin{aligned}
0 < \psi(B_{k+1}) &\leq \psi(B_k) + c + \ln(\cos^2(\theta_k)) \leq \psi(B_{k-1}) + 2c + \ln(\cos^2(\theta_{k-1})) + \ln(\cos^2(\theta_k)) \\
&\leq \cdots \leq \psi(B_1) + ck + \sum_{j=1}^{k} \ln(\cos^2(\theta_j)),
\end{aligned} \tag{4.14}$$

and we can assume the constant c to be positive, without loss of generality. We now relate these expressions to the results given in Chapter 3. Note that because $\mathbf{s}_k = -\alpha_k B_k^{-1}\nabla f(\mathbf{x}_k)$ and $B_k\mathbf{s}_k = -\alpha_k\nabla f(\mathbf{x}_k)$, θ_k results to be the angle between the search direction and the steepest descent direction. From the result of Zoutendijk's theorem we know that the sequence $\|\nabla f(\mathbf{x}_k)\|$ generated by the line search algorithm converges to zero if $\{\cos\theta_k\}$ is bounded away from 0. Let us then proceed by contradiction and assume that $\cos\theta_k \to 0$. Then there exists $k_1 > 0$ such that for all $j > k_1$, we have

$$\ln(\cos^2(\theta_j)) < -2c,$$

where c is the constant defined above. Using this inequality in (4.14) we find the following relations

to be true for all $k > k_1$:

$$0 < \psi(B_1) + ck + \sum_{j=1}^{k_1} \ln(\cos^2(\theta_j)) + \sum_{j=k_1+1}^{k} (-2c)$$
$$= \psi(B_1) + \sum_{j=1}^{k_1} \ln(\cos^2(\theta_j)) - 2c(k-k_1) = \psi(B_1) + \sum_{j=1}^{k_1} \ln(\cos^2(\theta_j)) + 2ck_1 - ck.$$

The right-hand-side is negative for large k, giving a contradiction. Therefore, there exists a subsequence of indices $\{j_k\}$ such that $\{\cos(\theta_{j_k})\} \geq \delta > 0$. By Zoutendijk's theorem this implies that $\liminf \|\nabla f(\mathbf{x}_k)\| \to 0$. Since the problem is strongly convex, the latter limit is enough to prove that $x_k \to x^*$. □

Practice time

a) Implement the BFGS method and test it on our test problems. Compare the results with Newton's and gradient methods and comment on them.

b) In gradient method replace the analytical expression of the gradient by finite differences. Solve the test problems. Does the accuracy change? Does the rate of convergence change? Does the computational time change (choose large n!)?

c) Optional: compute the Hessian approximation by finite differences using first the gradient analytical expression and then the gradient approximated by finite differences. How does the computational time change?

Chapter 5

Nonlinear least-squares problems

5.1 Background: modelling, regression

Nonlinear least-squares problems often arise when we want to fit a model to some data, i.e., when we solve a regression problem.

Assume to have some experimental measurements

$$(t_i, y_i), \qquad i = 1, ..., m.$$

The measurements are some (usually noisy) realisations of a function $y : \mathbb{R} \longrightarrow \mathbb{R}$ of the variable t that describes the observed phenomenon ($y_i = y(t_i) + \delta$, with δ the noise). We want to approximate this function by a model $m(\mathbf{x}; t)$, where $\mathbf{x} = (x_1, \ldots, x_n)^T$ are some parameters to be determined. We can choose the value of these parameters to adapt the model to the data at best (we want $m(\mathbf{x}; t) \sim y(t)$ for all t). [1]

The true variables of the model are then the parameters $\mathbf{x}$, the variable t of the original function will take the measured values.

To find the best parameters we try to minimize the distance of the model from the measurements, i.e., we look for $\mathbf{x} \in \mathbb{R}^n$ that minimizes the quantity

$$\frac{1}{2} \left\| \begin{pmatrix} m(\mathbf{x}; t_1) - y_1 \\ \vdots \\ m(\mathbf{x}, t_m) - y_m \end{pmatrix} \right\|^2 = \frac{1}{2} \sum_{i=1}^{m} \big(m(\mathbf{x}; t_i) - y_i\big)^2.$$

This is a nonlinear (if m is nonlinear with respect to $\mathbf{x}$) least-squares problem.

5.2 General concepts

Let

$$\begin{array}{rccc} F: & \mathbb{R}^n & \longrightarrow & \mathbb{R}^m \\ & \mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} & \longmapsto & \mathbf{F}(\mathbf{x}) = \begin{pmatrix} F_1(\mathbf{x}) \\ \vdots \\ F_m(\mathbf{x}) \end{pmatrix} \end{array}$$

[1] Usually we choose a family of functions to which the model belongs, which is parametrized by some parameters. For example we can have an exponential model $m(\mathbf{x}; t) = x_1 e^{x_2 t}$, with $\mathbf{x} = (x_1, x_2)^T$ or a linear model $m(\mathbf{x}; t) = x_1 + x_2 t$.

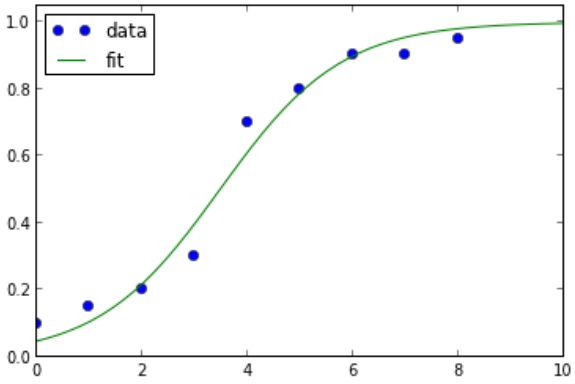


Figure 5.1: Example of data and model fitting them.

and

$$\begin{array}{rccl} f: & \mathbb{R}^n & \longrightarrow & \mathbb{R} \\ & \mathbf{x} & \longmapsto & f(\mathbf{x}) = \dfrac{1}{2}\|\mathbf{F}(\mathbf{x})\|^2 = \dfrac{1}{2}\displaystyle\sum_{i=1}^{m} F_i(\mathbf{x})^2. \end{array}$$

The general form of a nonlinear least-squares problem is

$$\min_{\mathbf{x}} f(\mathbf{x}) = \frac{1}{2}\|\mathbf{F}(\mathbf{x})\|^2.$$

In the following we will assume, as it is common in applications, that $m \geq n$.

Let $\mathbf{x} \in \mathbb{R}^n$. If F is differentiable in $\mathbf{x}$ (i.e., if $F_1, \ldots, F_m$ are differentiable in $\mathbf{x}$), the Jacobain matrix of F in $\mathbf{x}$ is $J(\mathbf{x}) \in \mathbb{R}^{m \times n}$

$$J(\mathbf{x}) = \begin{pmatrix} \nabla F_1(\mathbf{x})^T \\ \vdots \\ \nabla F_m(\mathbf{x})^T \end{pmatrix} = \begin{pmatrix} \dfrac{\partial F_1}{\partial x_1}(\mathbf{x}) & \cdots & \dfrac{\partial F_1}{\partial x_n}(\mathbf{x}) \\ \vdots & & \vdots \\ \dfrac{\partial F_m}{\partial x_1}(\mathbf{x}) & \cdots & \dfrac{\partial F_m}{\partial x_n}(\mathbf{x}) \end{pmatrix}.$$

Let $\mathbf{x}^*$ be a solution of the least-squares problem. The residual is the following quantity:

$$r = f(\mathbf{x}^*) = \frac{1}{2}\|\mathbf{F}(\mathbf{x}^*)\|^2.$$

If it exists a solution for which $r = 0$ we say that the problem is a zero residual problem.

Remark 5.2.1. *It holds*

(a) $\nabla f(\mathbf{x}) = J(\mathbf{x})^T \mathbf{F}(\mathbf{x})$,

(b) $H(\mathbf{x}) = J(\mathbf{x})^T J(\mathbf{x}) + \displaystyle\sum_{i=1}^{m} F_i(\mathbf{x}) H_{F_i}(\mathbf{x})$.

Proof. (a) $\forall j = 1, \ldots, n$

$$\begin{aligned}\Big(\nabla f(\mathbf{x})\Big)_j &= \frac{\partial}{\partial x_j} f(\mathbf{x}) = \frac{\partial}{\partial x_j} \frac{1}{2} \sum_{i=1}^{m} F_i(\mathbf{x})^2 = \frac{1}{2} \sum_{i=1}^{m} \frac{\partial F_i(\mathbf{x})^2}{\partial x_j} = \sum_{i=1}^{m} \frac{\partial F_i(\mathbf{x})}{\partial x_j} F_i(\mathbf{x}) \\ &= \sum_{i=1}^{m} \Big(J(\mathbf{x})\Big)_{ij} \Big(\mathbf{F}(\mathbf{x})\Big)_i = \sum_{i=1}^{m} \Big(J(\mathbf{x})^T\Big)_{ji} \Big(\mathbf{F}(\mathbf{x})\Big)_i = \Big(J(\mathbf{x})^T \mathbf{F}(\mathbf{x})\Big)_j.\end{aligned}$$

(b) $\forall j, k = 1, \ldots, n$

$$\begin{aligned}\Big(H(\mathbf{x})\Big)_{jk} &= \frac{\partial^2 f(\mathbf{x})}{\partial x_j \partial x_k} = \frac{\partial}{\partial x_k} \frac{\partial f(\mathbf{x})}{\partial x_j} = \frac{\partial}{\partial x_k} \Big(\nabla f(\mathbf{x})\Big)_j \underbrace{=}_{\text{(a)}} \\ &= \frac{\partial}{\partial x_k} \sum_{i=1}^{m} F_i(\mathbf{x}) \frac{\partial F_i(\mathbf{x})}{\partial x_j} = \sum_{i=1}^{m} \frac{\partial F_i(\mathbf{x})}{\partial x_k} \frac{\partial F_i(\mathbf{x})}{\partial x_j} + \sum_{i=1}^{m} F_i(\mathbf{x}) \frac{\partial^2 F_i(\mathbf{x})}{\partial x_j \partial x_k} \\ &= \sum_{i=1}^{m} \Big(J(\mathbf{x})\Big)_{ik} \Big(J(\mathbf{x})\Big)_{ij} + \sum_{i=1}^{m} F_i(\mathbf{x}) \Big(H_{F_i}(\mathbf{x})\Big)_{jk}.\end{aligned}$$

Because

$$\sum_{i=1}^{m} \Big(J(\mathbf{x})\Big)_{ik} \Big(J(\mathbf{x})\Big)_{ij} = \sum_{i=1}^{m} \Big(J(\mathbf{x})^T\Big)_{ki} \Big(J(\mathbf{x})\Big)_{ij} = \Big(\underbrace{J(\mathbf{x})^T J(\mathbf{x})}_{\text{is symmetric}}\Big)_{kj} = \Big(J(\mathbf{x})^T J(\mathbf{x})\Big)_{jk}$$

it holds

$$\Big(H(\mathbf{x})\Big)_{jk} = \Big(J(\mathbf{x})^T J(\mathbf{x}\Big)_{jk} + \sum_{i=1}^{m} F_i(\mathbf{x}) \Big(H_{F_i}(\mathbf{x})\Big)_{jk}.$$

□

Remark that $r = 0$ means $\mathbf{F}(\mathbf{x}^*) = \mathbf{0}$, i.e., $\mathbf{x}^*$ is the solution of the nonlinear system

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}.$$

If $r = 0$, then

$$H(\mathbf{x}^*) = J(\mathbf{x}^*)^T J(\mathbf{x}^*) + \sum_{i=1}^{m} \underbrace{F_i(\mathbf{x}^*)}_{=0} H_{F_i}(\mathbf{x}^*) = J(\mathbf{x}^*)^T J(\mathbf{x}^*).$$

In many situations the term $J(\mathbf{x})^T J(\mathbf{x})$ is a good approximation of the Hessian matrix $H(\mathbf{x})$, for example when the residuals are small ($F_i(\mathbf{x}) \approx 0$) or when the model is almost linear ($H_{F_i}(\mathbf{x}) \approx 0$). In such cases, for $\mathbf{x}$ close to $\mathbf{x}^*$, we can use the following approximation:

$$H(\mathbf{x}) = J(\mathbf{x})^T J(\mathbf{x}) + \sum_{i=1}^{m} F_i(\mathbf{x}) H_{F_i}(\mathbf{x}) \approx J(\mathbf{x})^T J(\mathbf{x}).$$

This is convenient because it allows us to get a reliable approximation to the second order derivatives by employing just first order derivatives, that can also be used to compute $\nabla f(x)$.

5.3 Linear least-squares problems

In the special case in which each function F_i is linear, we have $\mathbf{F}(\mathbf{x}) = A\mathbf{x} - b$, the Jacobian A is constant, and we can write

$$f(\mathbf{x}) = \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|^2.$$

We also have

$$\nabla f(\mathbf{x}) = A^T(A\mathbf{x} - \mathbf{b}) \qquad\qquad H(\mathbf{x}) = A^T A.$$

The second term in (b) disappears as the function is linear so $H_{F_i} = 0$ for all i. Function f is always convex and any solution must satisfy $\nabla f(\mathbf{x}) = A^T(A\mathbf{x} - \mathbf{b}) = 0$, which leads to the *normal equations*:

$$A^T A\mathbf{x} = A^T\mathbf{b}.$$

5.3.1 Geometric interpretation

Define

$$\mathcal{R}(A) = \{\mathbf{z} \in \mathbb{R}^m | \mathbf{z} = A\mathbf{x},\ \mathbf{x} \in \mathbb{R}^n\},$$
$$\mathcal{N}(A^T) = \{\mathbf{y} \in \mathbb{R}^n | A^T\mathbf{y} = 0\},$$

the range of A and the nullspace of A^T, respectively. These two spaces are orthogonal. Define the residual $\mathbf{r} = \mathbf{b} - A\mathbf{x}$. If $\mathbf{x}$ is a solution of the linear least-squares problem, it holds

$$0 = A^T A\mathbf{x} - A^T\mathbf{b} = A^T(A\mathbf{x} - \mathbf{b}) = -A^T\mathbf{r},$$

then $\mathbf{r} \in \mathcal{N}(A^T)$. Hence any least squares solution $\mathbf{x}$ uniquely decomposes the right-hand side into two orthogonal components:

$$\mathbf{b} = \underbrace{\mathbf{r}}_{\in\mathcal{N}(A^T)} + \underbrace{A\mathbf{x}}_{\in\mathcal{R}(A)}.$$

The least-squares problem

$$\min_{\mathbf{x}} \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|^2$$

can be interpreted as the minimization of the distance of $\mathbf{b}$ from the range of A. The solution is then obtained when $A\mathbf{x}$ is the projection of $\mathbf{b}$ onto $\mathcal{R}(A)$, which has an explicit expression:

$$P_{\mathcal{R}(A)}(b) = A(A^T A)^{-1}A^T\mathbf{b} = AA^\dagger\mathbf{b}$$

where $A^\dagger$ is the pseudoinverse of A (cf. Appendix B.5.3) and the last expression is valid only if $m \geq n$ and $\text{rank}(A) = n$. Then in this case $A\mathbf{x} = AA^\dagger\mathbf{b}$ and $\mathbf{x} = A^\dagger\mathbf{b}$ because A has full rank. This is coherent with the fact that $\mathbf{x}$ must satisfy the normal equations: $\mathbf{x} = (A^T A)^{-1}A^T\mathbf{b} = A^\dagger\mathbf{b}$.

5.3.2 How to solve a linear least-squares problem?

We outline briefly three major algorithms for the linear least-squares problem. We assume that $m \geq n$ and that A has full column rank.

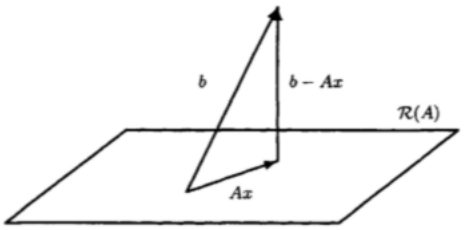


Figure 5.2: Illustration of the geometric property

1. The first and most obvious algorithm is simply to form and solve the normal equations, by computing the Cholesky factorization of the symmetric matrix $A^T A$, cf. Appendix B.6.1. This method is frequently used in practice and is often effective, but it has one significant disadvantage, namely, that the condition number of $A^T A$ is the square of the condition number of A . Since the relative error in the computed solution of any problem is usually proportional to the condition number, the solution obtained by the Cholesky-based method may be much less accurate than solutions obtained from algorithms that work directly with the least-squares formulation. The Cholesky-based algorithm is particularly useful when $m \ll n$ and it is practical to store $A^T A$ but not A itself.

2. A second approach is based on a QR factorization of the matrix A, that is we write $A = QR$ where $Q \in \mathbb{R}^{m\times m}$ is an orthogonal matrix and $R \in \mathbb{R}^{m\times n}$ is an upper triangular matrix. Let us write

$$Q = \begin{pmatrix} Q_1 \, Q_2 \end{pmatrix}, \quad R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

for $Q_1 \in \mathbb{R}^{m\times n}$, $Q_1 \in \mathbb{R}^{m\times m-n}$, $R_1 \in \mathbb{R}^{n\times n}$. Since the Euclidean norm of any vector is not affected by orthogonal transformations, we have

$$\begin{aligned} \|A\mathbf{x} + \mathbf{b}\|^2 = \|Q^T(A\mathbf{x} + \mathbf{b})\|^2 = \|Q^T A\mathbf{x} + Q^T\mathbf{b}\|^2 &= \left\| \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \mathbf{x} + \begin{pmatrix} Q_1^T\mathbf{b}_1 \\ Q_2^T\mathbf{b}_2 \end{pmatrix} \right\|^2 \\ &= \|R_1\mathbf{x} + Q_1^T\mathbf{b}_1\|^2 + \|Q_2^T\mathbf{b}_2\|^2. \end{aligned}$$

No choice of $\mathbf{x}$ has any effect on the second term of this last expression, but we can minimize $\|A\mathbf{x} + \mathbf{b}\|$ by driving the first term to zero, that is, by setting $\mathbf{x}^* = -R_1^{-1}Q_1^T\mathbf{b}_1$.

This QR-based approach does not degrade the conditioning of the problem unnecessarily. The relative error in the final computed solution is usually proportional to the condition number of A, not its square, and this method is usually reliable. The computation of the QR factorization costs $2mn^2$ flops. The QR method is then more expensive in term of flops than the solution of the normal equations, cf. Appendix B.6.1.

3. Some situations, however, call for greater robustness or more information about the sensitivity of the solution to perturbations in the data (A or $\mathbf{b}$). A third approach, based on the singular-value decomposition (SVD) of $A = USV^T$, can be used in these circumstances, cf. Appendix B.6.1. Following a similar reasoning to the one presented in the previous item (cf. TD5), we get that the optimal solution is given by

$$\mathbf{x}^* = \sum_{i=1}^{n} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i$$

where $\mathbf{u}_i, \mathbf{v}_i$ are the singular vectors of A (the columns of U, V, respectively. Note that for U we take just the first n columns) and σ_i are the singular values of A.

This formula yields useful information about the sensitivity of $\mathbf{x}$. When σ_i is small, $\mathbf{x}$ is particularly sensitive to perturbations in the data. Information such as this is particularly useful when A is nearly rank-deficient, that is, when $\sigma_n/\sigma_1 \ll 1$. This information is not generally available when the QR algorithm is used, and it is sometimes worth the extra cost of the SVD algorithm to obtain it. This approach is the most robust and reliable one for ill-conditioned problems. When A is actually rank-deficient, some of the singular values σ_i are exactly zero, and any vector $\mathbf{x}^*$ of the form

$$\mathbf{x}^* = \sum_{\sigma_i \neq 0} \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i + \sum_{\sigma_i = 0} \tau_i \mathbf{v}_i$$

(for arbitrary coefficients τ_i) is a minimizer. Frequently, the solution with smallest norm is most desirable, and we obtain this by setting each $\tau_i = 0$ in the previous expression, cf. TD5. When A has full rank but is ill conditioned, the last few singular values are small. The coefficients $\mathbf{u}_i^T \mathbf{b}/\sigma_i$ are then particularly sensitive to perturbations in $\mathbf{u}_i^T \mathbf{b}$ when σ_i is small, so a robust approximate solution is sometimes obtained by omitting these terms from the summation.

5.4 Algorithms for nonlinear least-squares problems

5.4.1 Gauss-Newton method

Let us assume that $r \approx 0$, that $J(\mathbf{x}_k) \in \mathbb{R}^{m \times n}$ with $m \geq n$, and that $J(\mathbf{x}_k)$ is a full rank matrix, i.e., $\text{rank}(J(\mathbf{x}_k)) = n$. The Gauss-Newton direction $\mathbf{p}_k^{GN}$ is the quasi-Newton direction obtained choosing

$$B_k = J(\mathbf{x}_k)^T J(\mathbf{x}_k),$$

i.e., the step $\mathbf{p}_k^{GN}$ is selected as the minimizer of the following model:

$$\begin{aligned} m_k^{GN}(\mathbf{p}) &= \frac{1}{2}\|\mathbf{F}(\mathbf{x}_k)\|^2 + J(\mathbf{x}_k)^T \mathbf{F}(\mathbf{x}_k)\mathbf{p} + \frac{1}{2}\mathbf{p}^T J(\mathbf{x}_k)^T J(\mathbf{x}_k)\mathbf{p} \\ &= f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \mathbf{p} + \frac{1}{2}\mathbf{p}^T J(\mathbf{x}_k)^T J(\mathbf{x}_k)\mathbf{p}, \end{aligned}$$

which can be computed by solving the quasi-Newton system

$$J(\mathbf{x}_k)^T J(\mathbf{x}_k)\mathbf{p} = -J(\mathbf{x}_k)^T \mathbf{F}(\mathbf{x}_k). \tag{5.1}$$

The matrix $J(\mathbf{x}_k)^T J(\mathbf{x}_k)$ is SPD when $J(\mathbf{x}_k)$ has full rank (all the eigenvalues are nonzero). Indeed $\forall \mathbf{v} \in \mathbb{R}^n$, $v \neq 0$

$$\mathbf{v}^T J(\mathbf{x}_k)^T J(\mathbf{x}_k)\mathbf{v} = \big(J(\mathbf{x}_k)\mathbf{v}\big)^T J(\mathbf{x}_k)\mathbf{v} = \|J(\mathbf{x}_k)\mathbf{v}\|^2 \quad > \quad 0.$$

$\mathbf{p}_k^{GN}$ is a descent direction for f in $\mathbf{x}_k$ because

$$\begin{aligned} \nabla f(\mathbf{x}_k)^T \mathbf{p}_k^{GN} &= \big(J(\mathbf{x}_k)^T \mathbf{F}(\mathbf{x}_k)\big)^T \mathbf{p}_k^{GN} = \Big(- J(\mathbf{x}_k)^T J(\mathbf{x}_k)\mathbf{p}_k^{GN}\Big)^T \mathbf{p}_k^{GN} \\ &= -(\mathbf{p}_k^{GN})^T J(\mathbf{x}_k)^T J(\mathbf{x}_k)\mathbf{p}_k^{GN} = -\|J(\mathbf{x}_k)\mathbf{p}_k^{GN}\|^2 < 0, \end{aligned}$$

where this last inequality follows from the fact that $J(\mathbf{x}_k)^T J(\mathbf{x}_k)$ is positive definite.
If $J(\mathbf{x}_k)$ is not full rank, then $J(\mathbf{x}_k)^T J(\mathbf{x}_k)$ is still symmetric but it is just positive semidefinite. In

this case a possibility is to choose $B_k = J(\mathbf{x}_k)^T J(\mathbf{x}_k) + \varepsilon I_n$ with $\varepsilon > 0$ small, so that B_k is SPD[2] and it approximates well $J(\mathbf{x}_k)^T J(\mathbf{x}_k)$ (because ε is small). We will discuss this in Section 5.5.

The Gauss-Newton method can be also derived by approximating the objective function F by a linear model at each iteration: $\mathbf{F}(\mathbf{x}_k + \mathbf{p}) \sim \mathbf{F}(\mathbf{x}_k) + J(\mathbf{x}_k)\mathbf{p}$. Using as a model for f the squared norm of the linear model of F, we obtain exactly the quadratic approximation $m_k^{GN}(\mathbf{p})$ to f with approximated Hessian:

$$\frac{1}{2}\|\mathbf{F}(\mathbf{x}_k) + J(\mathbf{x}_k)p\|^2 = \frac{1}{2}\|\mathbf{F}(\mathbf{x}_k)\|^2 + J(\mathbf{x}_k)^T\mathbf{F}(\mathbf{x}_k)\mathbf{p} + \frac{1}{2}\mathbf{p}^T J(\mathbf{x}_k)^T J(\mathbf{x}_k)\mathbf{p}.$$

To get the step at each iteration we minimize the model, which amounts to solve a linear least-squares problem, whose normal equations are exactly (5.1). By using the SVD decomposition of $J(\mathbf{x}_k) = USV^T$, we can write (see TD 5) the solution of this problem as

$$\mathbf{p}^* = \sum_{i=1}^{n} \frac{\mathbf{u}_i^T\mathbf{F}}{\sigma_i}\mathbf{v}_i.$$

Gauss-Newton method is then based on the following iterative scheme:

$$\left\{ \begin{array}{l} \text{Given } \mathbf{x}_0 \\ \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k^{GN} \end{array} \right\}.$$

The convergence of the method clearly depends on how important is the term we have discarded in the Hessian approximation. As shown in the following theorem, the method is locally convergent with quadratic local convergence in case of zero residual. If the residual is nonzero, if $\left\| \sum_{i=1}^{m} F_i(\mathbf{x}^*)H_{F_i}(\mathbf{x}^*) \right\|$ is small with respect to the smallest eigenvalue of $J(x^*)^T J(x^*)$, then the convergence is linear. Otherwise, there is no guarantee of convergence for the method.

Theorem 5.4.1. *Let $F : \mathbb{R}^n \to \mathbb{R}^m$ and let $f(x) = \frac{1}{2}\|\mathbf{F}(\mathbf{x})\|^2$ be twice continuously differentiable in an open convex set $\mathcal{D} \subset \mathbb{R}^n$. Assume that $J(\mathbf{x})$ is Lipschitz continuous in $\mathcal{D}$ with Lipschitz constant γ and that $\|J(\mathbf{x})\| \leq \alpha$ for all $\mathbf{x} \in \mathcal{D}$. Assume that there exists $\mathbf{x}^* \in \mathcal{D}$ such that $J(\mathbf{x}^*)^T\mathbf{F}(\mathbf{x}^*) = 0$. Let λ be the smallest eigenvalue of $J(\mathbf{x}^*)^T J(\mathbf{x}^*)$ and assume that*

$$\|(J(\mathbf{x}) - J(\mathbf{x}^*))^T\mathbf{F}(\mathbf{x}^*)\| \leq \sigma\|\mathbf{x} - \mathbf{x}_*\|$$

for some constant $\sigma \geq 0$ and for all $\mathbf{x} \in \mathcal{D}$. If $\sigma < \lambda$ then for any $c \in (1, \lambda/\sigma)$ there exists $\epsilon > 0$ such that for all $\mathbf{x_0} \in B_\epsilon(\mathbf{x}^*)$ the sequence $\{\mathbf{x}_k\}$ generated by the Gauss-Newton method is well-defined, converges to $\mathbf{x}^*$ and obeys

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \frac{c\sigma}{\lambda}\|\mathbf{x}_k - \mathbf{x}^*\| + \frac{c\alpha\gamma}{2\lambda}\|\mathbf{x}_k - \mathbf{x}^*\|^2,$$
$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \frac{c\sigma + \lambda}{2\lambda}\|\mathbf{x}_k - \mathbf{x}^*\| < \|\mathbf{x}_k - \mathbf{x}^*\|.$$

[2] If $A \in \mathbb{R}^{n\times n}$ is positive semidefinite, then $\forall \varepsilon > 0$ $A + \varepsilon I_n$ is positive definite.
Indeed, let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of A and $\mathbf{v}_1, \ldots, \mathbf{v}_n$ the relative eigenvectors. $\forall i = 1, \ldots, n$

$$(A + \varepsilon I_n)\mathbf{v}_i = A\mathbf{v}_i + \varepsilon\mathbf{v}_i = \lambda_i\mathbf{v}_i + \varepsilon\mathbf{v}_i = (\lambda_i + \varepsilon)\mathbf{v}_i,$$

i.e., $\mathbf{v}_i$ is an eigenvector of $A + \varepsilon I_n$ with eigenvalue $\lambda_i + \varepsilon$. Then the eigenvalues of $A + \varepsilon I_n$ are $\lambda_1 + \varepsilon, \ldots, \lambda_n + \varepsilon$, which are all positive because $\lambda_1, \ldots, \lambda_n \geq 0$ as A is positive semidefinite.

Corollary 5.4.1. *Let the assumptions of Theorem 5.4.1 hold. If* $\mathbf{F}(\mathbf{x}^*) = 0$*, then there exists* $\epsilon > 0$ *such that for all* $\mathbf{x_0} \in B_\epsilon(\mathbf{x}^*)$ *the sequence* $\{\mathbf{x}_k\}$ *generated by the Gauss-Newton method is well-defined and converges quadratically to* $\mathbf{x}^*$*.*

Theorem 5.4.1 shows that Gauss-Newton method may not be quickly locally convergent and that (when $S(\mathbf{x}^*)$ is too large) it may not be convergent at all. The constant σ plays a crucial role in the convergence. It may be seen as an absolute combined measure of linearity and residual size of the problem because it holds:

$$(J(\mathbf{x}) - J(\mathbf{x}^*))^T \mathbf{F}(\mathbf{x}^*) \simeq S(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}_*),$$

if F is linear or $\mathbf{F}(\mathbf{x}^*) = 0$ then $\sigma = 0$. For the convergence we must look at the ratio $\dfrac{\sigma}{\lambda}$, which must be less than 1. This can be interpreted as a relative combined measure of linearity and residual size of the problem.

Gauss-Newton method with line-search consists of choosing the Gauss-Newton direction in the line-search algorithm. In this way the method becomes globally convergent and close to $\mathbf{x}^*$ it has quadratic convergence in case $r = 0$.

5.5 Levenberg-Marquardt method

The Levenberg-Marquardt method is a modification of Gauss-Newton method that avoids one of the weaknesses of Gauss-Newton, namely, its behavior when the Jacobian is rank-deficient, or nearly so.

The Levenberg-Marquardt method is derived by modifying the Gauss-Newton model, by adding a regularization term that depends on a strictly positive regularization parameter λ_k:

$$m_k^{LM}(\mathbf{p}) = \frac{1}{2}\|J(\mathbf{x}_k)\mathbf{p} - \mathbf{F}(\mathbf{x}_k)\|^2 + \frac{\lambda_k}{2}\|\mathbf{p}\|^2. \tag{5.2}$$

The minimizer of this model satisfies a modification of the normal equations:

$$(J(\mathbf{x}_k)^T J(\mathbf{x}_k) + \lambda_k I)\mathbf{p} = -J(\mathbf{x}_k)^T \mathbf{F}(\mathbf{x}_k).$$

These are indeed the normal equations of the following linear least-squares problem:

$$\min_{\mathbf{p}} \frac{1}{2}\left\| \begin{bmatrix} J(\mathbf{x}_k) \\ \sqrt{\lambda_k} I \end{bmatrix} \mathbf{p} - \begin{bmatrix} \mathbf{F}(\mathbf{x}_k) \\ 0 \end{bmatrix} \right\|^2,$$

which is equivalent to

$$\min_{\mathbf{p}} m_k^{LM}(\mathbf{p}) \tag{5.3}$$

where m_k^{LM} is defined in (5.2). The term that is added ensures that $J(\mathbf{x}_k)^T J(\mathbf{x}_k)$ is positive definite. By using the SVD of $J(\mathbf{x}_k)$, we can write the solution of this problem as

$$\mathbf{p}^* = \sum_{i=1}^{n} \frac{\mathbf{u}_i^T \mathbf{F}}{\sigma_i + \lambda_k} \mathbf{v}_i. \tag{5.4}$$

Remark that when $\lambda_k \to 0$, $\mathbf{p}^*$ tends to the solution of the Gauss-Newton system. The Levenberg-Marquardt model is then

$$m_k^{LM}(\mathbf{p}) = \frac{1}{2}\|\mathbf{F}(\mathbf{x}_k)\|^2 + J(\mathbf{x}_k)^T \mathbf{F}(\mathbf{x}_k)\mathbf{p} + \frac{1}{2}\mathbf{p}^T J(\mathbf{x}_k)^T J(\mathbf{x}_k)\mathbf{p} + \frac{\lambda_k}{2}\|\mathbf{p}\|^2.$$

In the original version of the Levenberg-Marquardt method the parameter λ_k is updated at each iteration, similarly to the trust-region radius in trust-region methods and this choice directly affects the step length (cf. (5.4)). So in LM method the step-length is automatically selected and the iterate is simply updated as $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$.

The regularization parameter is increased or decreased by a certain factor according to whether or not the previous trial step was effective in decreasing f . If $\mathbf{p}_k \neq \mathbf{0}$ it means that $m_k^{LM}(\mathbf{x}_k, \mathbf{p}_k) < m_k^{LM}(\mathbf{x}_k, 0)$, otherwise the null step would have been selected in the minimization. However, it may not hold $f(\mathbf{x}_k + \mathbf{p}_k) < f(\mathbf{x}_k)$. If this happens it means that the model is not a good approximation of $f(\mathbf{x})$ and we need to regularize more to enforce a smaller step (this is like setting a larger weight in front of $\|\mathbf{p}\|^2$, which penalizes steps with large norm). We then choose $\lambda_{k+1} > \lambda_k$ and we solve again the minimization problem.
Otherwise we accept the step, i.e. we set

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$$

and we may decrease the regularization parameter to encourage larger steps in subsequent iterations. It is possible to show that after a finite number of steps $f(\mathbf{x}_k + \mathbf{p}_k) < f(\mathbf{x}_k)$. The practical criterion to update λ and to accept the step is the following one: if

$$\rho_k = \frac{f(\mathbf{x}_k + \mathbf{p}_k) - f(\mathbf{x}_k)}{m_k^{LM}(\mathbf{x}_k + \mathbf{p}_k) - m_k^{LM}(\mathbf{x}_k)} \geq \eta_1,$$

with $\eta_1 > 0$ (usually $\eta_1 = 0.25$) we accept the step.

The second-order Hessian component in (b) is still ignored, so the local convergence properties of the two methods are similar.

Chapter 6

Constrained optimization

We are interested in the minimizer $\mathbf{x}^*$ of

$$\begin{array}{rccc} f: & \mathbb{R}^n & \longrightarrow & \mathbb{R} \\ & \mathbf{x} & \longmapsto & f(\mathbf{x}) \end{array}$$

subject to some constraints on the variables, that is we assume that for some

$$\begin{array}{rccc} h: & \mathbb{R}^n & \longrightarrow & \mathbb{R}^p \\ & \mathbf{x} & \longmapsto & \mathbf{h}(\mathbf{x}), \end{array} \qquad \begin{array}{rccc} g: & \mathbb{R}^n & \longrightarrow & \mathbb{R}^m \\ & \mathbf{x} & \longmapsto & \mathbf{g}(\mathbf{x}) \end{array},$$

it holds

$$\mathbf{h}(\mathbf{x}) = \mathbf{0}, \qquad \mathbf{g}(\mathbf{x}) \geq \mathbf{0},^{1}$$

that is we have p equality constraints and m inequality constraints.
We look for a solution $\mathbf{x}^*$ of the problem

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \\ &\mathbf{h}(\mathbf{x}) = \mathbf{0}, \\ &\mathbf{g}(\mathbf{x}) \geq \mathbf{0}. \end{aligned}$$

The set

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{h}(\mathbf{x}) = \mathbf{0},\ \mathbf{g}(\mathbf{x}) \geq \mathbf{0}\}$$

is called *feasible set* for the problem. We can then state the problem as

$$\min_{\mathbf{x}\in\Omega} f(\mathbf{x}).$$

We say that an inequality constraint $i \in \{1, \ldots, m\}$ is active in $\mathbf{x}$ if $g_i(\mathbf{x}) = 0$, and inactive if $g_i(\mathbf{x}) > 0$.
We denote

$$\mathcal{A}(\mathbf{x}) = \{i \in \{1, \ldots, m\} \mid g_i(\mathbf{x}) = 0\}$$

the set of active constraints in $\mathbf{x}$.

Moreover,

- $\mathbf{x}^* \in \Omega$ is a local minimizer for f if it exists a neighbourhood $\mathcal{N}$ of $\mathbf{x}^*$ such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \qquad \forall \mathbf{x} \in \Omega \cap \mathcal{N}.$$

- $\mathbf{x}^*$ is an isolated minimizer for f if it exists a neighbourhood $\mathcal{N}$ of $\mathbf{x}^*$ such that x^* is the only minimizer in $\Omega \cap \mathcal{N}$.

[1] For $\mathbf{v}, \mathbf{w} \in \mathbb{R}^N$, $\mathbf{v} \geq \mathbf{w}$ means $v_i \geq w_i \ \ \forall i = 1, \ldots, N$.

6.1 One equality constraint

Let us consider the problem

$$\min_{\mathbf{x}\in\mathbb{R}^2} x_1 + x_2$$
$$2 - x_1^2 - x_2^2 = 0,$$

i.e., we look for the minimizer of $f(\mathbf{x})$ on the boundary of the circle centred at $(0,0)^T$ and of radius $\sqrt{2}$. From Figure 6.1, in which level curves of $f(\mathbf{x})$ are plotted, it is clear that $\mathbf{x}^* = (-1,-1)^T$. We remark that

$$\nabla f(\mathbf{x}) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \nabla h(\mathbf{x}) = -2 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.$$

Then for each $\mathbf{x}$ on the boundary of the circle $\nabla h(\mathbf{x})$ is orthogonal to the circle and points towards its interior.
Starting from a point on the circle it is easy to see how to move to remain on the constraint and at the same time to decrease the values of $f(\mathbf{x})$. For example, starting from $\mathbf{x} = (\sqrt{2}, 0)^T$, we can move on the circle clockwise, i.e. following the direction that is tangent to the circle and orthogonal to $\nabla h(\mathbf{x})$ and that is of descent for f in $\mathbf{x}$, i.e. we have to follow the direction $\mathbf{d}$ such that

$$\begin{cases} \nabla h(\mathbf{x})^T \mathbf{d} = 0, \\ \nabla f(\mathbf{x})^T \mathbf{d} < 0. \end{cases}$$

We remark that at the solution, the gradient of the constraint is parallel to the gradient of the function, that is $\exists \mu^* \in \mathbb{R}$ s.t.

$$\nabla f(\mathbf{x}^*) = \mu^* \nabla h(\mathbf{x}^*);$$

in particular it holds $\mu^* = \dfrac{1}{2}$.

Let us assume to have just one equality constraint:

$$\begin{matrix} \min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \\ h(\mathbf{x}) = 0 \end{matrix}.$$

If we are at a feasible point $\mathbf{x} : h(\mathbf{x}) = 0$, and we approximate $h(\mathbf{x} + \alpha\mathbf{d})$ with the first order Taylor series, we get:

$$h(\mathbf{x} + \alpha\mathbf{d}) \approx h(\mathbf{x}) + \alpha \nabla h(\mathbf{x})^T \mathbf{d} = \alpha \nabla h(\mathbf{x})^T \mathbf{d},$$

and to decrease $f(\mathbf{x})$ it is necessary that $\mathbf{d}$ is a descent direction for f in $\mathbf{x}$, Then, at first order, we have to move along a direction $\mathbf{d}$ such that

$$\begin{cases} \nabla h(\mathbf{x})^T \mathbf{d} = 0, \\ \nabla f(\mathbf{x})^T \mathbf{d} < 0. \end{cases} \tag{EC}$$

If we are at a feasible point $\mathbf{x} : h(\mathbf{x}) = 0$ and it exists a direction $\mathbf{d}$ that satisfies (EC), we can move along that direction and find a point on the constraint in which f has a lower value, then $\mathbf{x}$ is not $\mathbf{x}^*$. We can then infer the following necessary condition to have that $\mathbf{x}$ is a solution: if $\mathbf{x}$ is a solution, then it cannot exist $\mathbf{d} \in \mathbb{R}^n$ that satisfies (EC).
The only way that $\mathbf{d} \in \mathbb{R}^n$ that satisfies (EC) cannot exist is if $\nabla f(\mathbf{x})$ is parallel to $\nabla h(\mathbf{x})$. The necessary condition becomes then:

$$\mathbf{x}^* \text{ is a solution} \quad \Longrightarrow \quad \exists \mu^* \in \mathbb{R} \text{ s.t. } \nabla f(\mathbf{x}^*) = \mu^* \nabla h(\mathbf{x}^*).$$

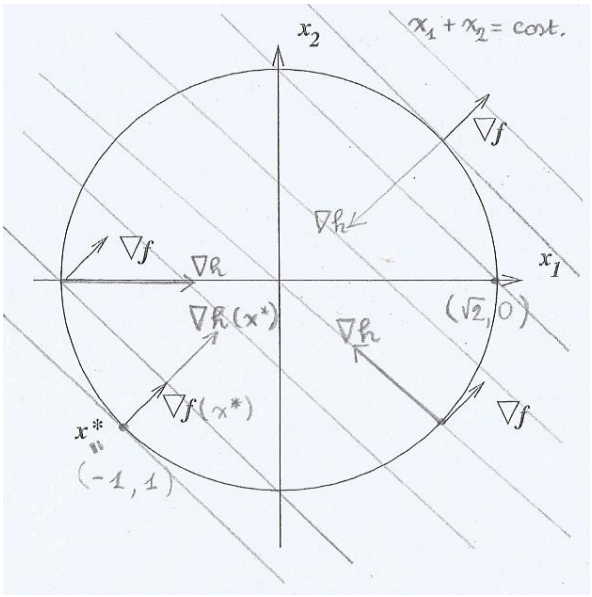


Figure 6.1: Problem of the example, showing constraint and function gradients at various feasible points.

We introduce the Lagrangian function

$$\mathcal{L}(\mathbf{x}, \mu) = f(\mathbf{x}) - \mu h(\mathbf{x}),$$

where μ is called Lagrange multiplier of the equality constraint, and by $\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}, \mu) = \nabla f(\mathbf{x}) - \mu \nabla h(\mathbf{x})$, the necessary condition becomes

$$\mathbf{x}^* \text{ is a solution} \quad \Longrightarrow \quad \exists \mu^* \in \mathbb{R} \text{ s.t. } \nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^*, \mu^*) = \mathbf{0}. \tag{NC}$$

This necessary condition is not sufficient. Indeed, in the previous example, if $\widetilde{\mathbf{x}} = (1, 1)^T$, $\exists \widetilde{\mu} \in \mathbb{R}$ s.t. $\nabla f(\widetilde{\mathbf{x}}) = \widetilde{\mu} \nabla h(\widetilde{\mathbf{x}})$, but $\widetilde{\mathbf{x}}$ is not a solution of the problem (this is a maximizer).

6.2 One inequality constraint

Let us consider again the previous problem, in which we replace the equality constraint by an inequality constraint:

$$\min_{\mathbf{x}\in\mathbb{R}^2} x_1 + x_2$$
$$2 - x_1^2 - x_2^2 \geq 0$$

We look for the minimizer of $f(\mathbf{x})$ in the close ball of centre $(0,0)^T$ and radius $\sqrt{2}$. From Figure 6.1 it is clear that the solution is still $\mathbf{x}^* = (-1,-1)^T$, point in which the constraint is active, i.e., $g(\mathbf{x}^*) = 0$. Remark that

$$\nabla f(\mathbf{x}) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \nabla g(\mathbf{x}) = -2 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},$$

then for each $\mathbf{x}$ on the boundary of the circle of the ball $\nabla g(\mathbf{x})$ is orthogonal to it and points towards the inside.
The gradient of f is parallel to the gradient of g in $\mathbf{x}^*$, that is $\exists \lambda^* \in \mathbb{R}$ s.t.

$$\nabla f(\mathbf{x}^*) = \lambda^* \nabla g(\mathbf{x}^*);$$

in particular it holds $\lambda^* = \dfrac{1}{2}$.

Suppose to have just one inequality constraint:

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x})$$
$$g(\mathbf{x}) \geq 0.$$

If $\mathbf{x} : g(\mathbf{x}) \geq 0$, we have to move along a direction $\mathbf{d}$ remaining on the constraint and at the same time decreasing f; using a first order approximation to g, to remain on the constraint is necessary to have

$$0 \leq g(\mathbf{x} + \alpha\mathbf{d}) \approx g(\mathbf{x}) + \alpha \nabla g(\mathbf{x})^T \mathbf{d},$$

and to decrease $f(\mathbf{x})$ it is necessary that $\mathbf{d}$ is a descent direction for f in $\mathbf{x}$. Then, at first order, we have to move along $\mathbf{d}$ such that

$$\begin{cases} g(\mathbf{x}) + \nabla g(\mathbf{x})^T \mathbf{d} \geq 0 \\ \nabla f(\mathbf{x})^T \mathbf{d} < 0 \end{cases} \tag{IC}$$

If $\mathbf{x} : g(\mathbf{x}) \geq 0$ and it exists a direction $\mathbf{d}$ that satisfies (IC), then we can move along that direction and find a point on the constraint where f has a lower value, then $\mathbf{x}$ is not $\mathbf{x}^*$. We infer then the following necessary condition for $\mathbf{x}$ to be a solution: if $\mathbf{x}$ is a solution, then it does not exist $\mathbf{d} \in \mathbb{R}^n$ that satisfies (IC).

- If the constraint is inactive in $\mathbf{x}$, i.e. $g(\mathbf{x}) > 0$, then the first condition in (IC) is always satisfied, if we choose $\mathbf{d}$ of sufficiently small length, so (IC) becomes

  $$\nabla f(\mathbf{x})^T \mathbf{d} < 0.$$

  It does not exist $\mathbf{d} \in \mathbb{R}^n$ that satisfies this condition if and only if $\nabla f(\mathbf{x}) = \mathbf{0}$. Then the necessary condition becomes

  $$\mathbf{x}^* \text{ is a solution} \quad \Longrightarrow \quad \nabla f(\mathbf{x}^*) = \mathbf{0}.$$

- If the constraint is active in $\mathbf{x}$, that is $g(\mathbf{x}) = 0$, then (IC) becomes

$$\begin{cases} \nabla g(\mathbf{x})^T \mathbf{d} \geq 0 \\ \nabla f(\mathbf{x})^T \mathbf{d} < 0 \end{cases}$$

The first of these two conditions defines a closed semispace of $\mathbb{R}^n$, while the second one an open semispace. It does not exist $\mathbf{d} \in \mathbb{R}^n$ that satisfies the condition if and only if such semispaces have void intersection, i.e., if and only if $\nabla f(\mathbf{x})$ and $\nabla g(\mathbf{x})$ are parallel and point towards the same direction (see Figure 6.2). Then the necessary condition becomes

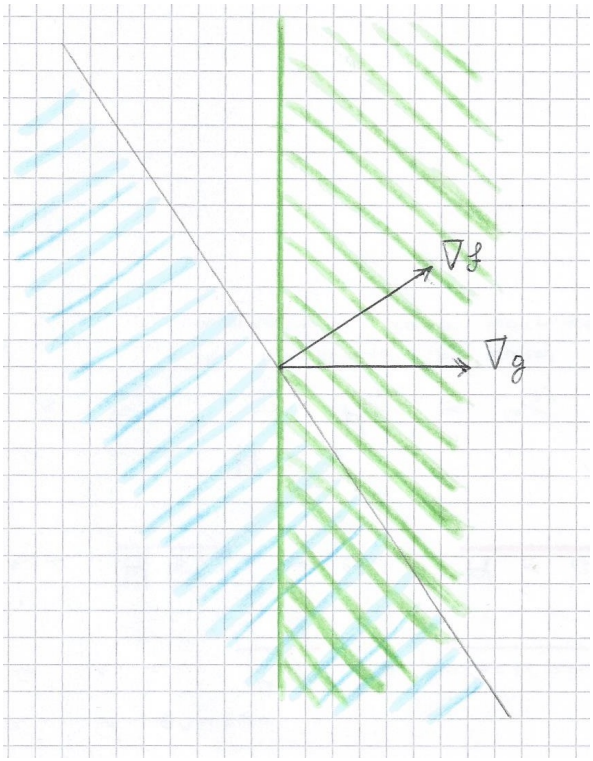


Figure 6.2: The light blue region is the open semispace of the vectors $\mathbf{d}$ such that $\nabla f^T \mathbf{d} < 0$; the green region is the closed semispace of the vectors $\mathbf{d}$ such that $\nabla g^T \mathbf{d} \geq 0$.

$$\mathbf{x}^* \text{ is a solution} \quad \Longrightarrow \quad \exists \lambda^* > 0 \text{ s.t. } \nabla f(\mathbf{x}^*) = \lambda^* \nabla g(\mathbf{x}^*).$$

The global necessary condition becomes

$$\mathbf{x}^* \text{ is a solution} \quad \Longrightarrow \quad \exists \lambda^* \geq 0 \text{ s.t. } \nabla f(\mathbf{x}^*) = \lambda^* \nabla g(\mathbf{x}^*), \;\; \lambda^* g(\mathbf{x}^*) = 0.$$

The condition $\lambda^* g(\mathbf{x}^*) = 0$ is called *complementarity condition*.
If the constraint is inactive in $\mathbf{x}^*$, that is $g(\mathbf{x}^*) > 0$, then the complementarity condition requires that $\lambda^* = 0$, and so the necessary condition becomes the one we use in unconstrained optimization.
If in $\mathbf{x}^*$ the constraint is inactive, i.e., $g(\mathbf{x}^*) = 0$, then the complementarity condition does not impose

a condition on λ^*.
Introducing the Lagrangian function

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x}),$$

where λ is called Lagrange multiplier of the inequality constraint, and by $\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}, \lambda) = \nabla f(\mathbf{x}) - \lambda \nabla g(\mathbf{x})$, the necessary condition becomes

$$\mathbf{x}^* \text{ is a solution} \quad \Longrightarrow \quad \exists \lambda^* \geq 0 \text{ s.t. } \nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^*, \lambda^*) = \mathbf{0}, \;\; \lambda^* g(\mathbf{x}^*) = 0.$$

6.3 First order optimality conditions

In general, the Lagrangian function of the problem is defined as

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i=1}^{p} \mu_i h_i(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x}),$$

where $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_p)^T$ is the vector of Lagrange multipliers for equality constraints and $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_m)^T$ is the vector of Lagrange multipliers for inequality constraints.

Generalizing what we have seen in the examples, if $\mathbf{x} : \mathbf{h}(\mathbf{x}) = \mathbf{0}, \mathbf{g}(\mathbf{x}) \geq \mathbf{0}$, and we want to move along a direction $\mathbf{d}$ remaining on the constraint, at first order we have to move along a direction $\mathbf{d}$ such that

$$\begin{cases} \nabla h_i(\mathbf{x})^T \mathbf{d} = 0 & \forall i = 1, \ldots, p \\ \nabla g_i(\mathbf{x})^T \mathbf{d} \geq 0 & \forall i \in \mathcal{A}(\mathbf{x}). \end{cases}$$

We define

$$F_1(\mathbf{x}) = \{\mathbf{d} \in \mathbb{R}^n \mid \nabla h_i(\mathbf{x})^T \mathbf{d} = 0 \;\; \forall i = 1, \ldots, p, \;\; \nabla g_i(\mathbf{x})^T \mathbf{d} \geq 0 \;\; \forall i \in \mathcal{A}(\mathbf{x})\}$$

set of feasible directions in $\mathbf{x}$. Remark that $F_1(\mathbf{x})$ is a cone[2].
However, this is just a first order approximation (this is exact only if the constraints are linear): in general we have to move along an arc. An arc α parametrized by a parameter $\vartheta \geq 0$ and such that $\boldsymbol{\alpha}(0) = \mathbf{x}$ is said admissible arc in $\mathbf{x}$ if

$$\left\{ \begin{array}{ll} h_i(\boldsymbol{\alpha}(\vartheta)) = 0 & \forall i = 1, \ldots, p \\ g_i(\boldsymbol{\alpha}(\vartheta)) \geq 0 & \forall i \in \mathcal{A}(\mathbf{x}) \end{array} \right\}$$

for ϑ small enough (i.e., for $\vartheta \in [0, \overline{\vartheta}]$ for some $\overline{\vartheta} > 0$). The set

$$T(\mathbf{x}) = \{\mathbf{d} \in \mathbb{R}^n \mid \mathbf{d} = \boldsymbol{\alpha}'(0) \text{ for some } \boldsymbol{\alpha}(\vartheta) \text{ admissible arc in } \mathbf{x}\}$$

of tangent directions to the admissible arcs in $\mathbf{x}$ is a cone and it is called *tangent cone* in $\mathbf{x}$.

Observation 1 $T(\mathbf{x}) \subseteq F_1(\mathbf{x})$.

Proof. Let $\mathbf{d} \in T(\mathbf{x})$. From the definition of tangent cone, $\mathbf{d} = \boldsymbol{\alpha}'(0)$ for some $\boldsymbol{\alpha}(\vartheta)$ admissible arc in $\mathbf{x}$. It holds $\forall i = 1, \ldots, p$

$$0 \underbrace{=}_{h_i(\boldsymbol{\alpha}(\vartheta)) = 0 \;\; \forall \vartheta \in [0, \overline{\vartheta}]} \left[\frac{\mathrm{d}}{\mathrm{d}\vartheta} h_i(\boldsymbol{\alpha}(\vartheta))\right]_{\vartheta = 0} = \nabla h_i(\boldsymbol{\alpha}(0))^T \boldsymbol{\alpha}'(0) = \nabla h_i(\mathbf{x})^T \mathbf{d}$$

[2] $C \subseteq \mathbb{R}^n$, $C \neq \varnothing$ is a cone if $\forall \mathbf{d} \in C$ it holds $\alpha \mathbf{d} \in C \;\; \forall \alpha \geq 0$

and $\forall i \in \mathcal{A}(\mathbf{x})$

$$0 \underbrace{\leq}_{\substack{g_i(\boldsymbol{\alpha}(0))=0 \text{ and} \\ g_i(\boldsymbol{\alpha}(\vartheta)) \geq 0 \ \ \forall \vartheta \in (0,\overline{\vartheta}]}} \left[\frac{\mathrm{d}}{\mathrm{d}\vartheta} g_i(\boldsymbol{\alpha}(\vartheta))\right]_{\vartheta=0} = \nabla g_i(\boldsymbol{\alpha}(0))^T \boldsymbol{\alpha}'(0) = \nabla g_i(\mathbf{x})^T \mathbf{d}.$$

□

Remark 2 $T(\mathbf{x}) \not\supseteq F_1(\mathbf{x})$.
Let's show this with two examples.

(1) Let us consider the problem

$$\begin{array}{l} \min\limits_{\mathbf{x}\in\mathbb{R}^2} x_1 + x_2 \\ (2 - x_1^2 - x_2^2)^2 = 0 \end{array}.$$

Remark that

$$\nabla h(\mathbf{x}) = \begin{pmatrix} 2(2 - x_1^2 - x_2^2)(-2x_1) \\ 2(2 - x_1^2 - x_2^2)(-2x_2) \end{pmatrix} = -4(2 - x_1^2 - x_2^2) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.$$

We notice that $\Omega = \{\mathbf{x} \in \mathbb{R}^2 \mid (2 - x_1^2 - x_2^2)^2 = 0\} = \{\mathbf{x} \in \mathbb{R}^2 \mid 2 - x_1^2 - x_2^2 = 0\}$ is the boundary of the circle of centre $\mathbf{0}$ and radius $\sqrt{2}$. Let $\mathbf{x} \in \Omega$. It is clear that the tangent directions and feasible arcs in $\mathbf{x}$ are just two, then $T(\mathbf{x})$ contains just two elements. However, because $\mathbf{x} \in \Omega$, it holds $2 - x_1^2 - x_2^2 = 0$, and then $\nabla h(\mathbf{x}) = \mathbf{0}$. Then

$$F_1(\mathbf{x}) = \{\mathbf{d} \in \mathbb{R}^2 \mid \underbrace{\nabla h(\mathbf{x})^T}_{=\mathbf{0}} \mathbf{d} = 0\} = \mathbb{R}^2.$$

(2) Let us consider the problem

$$\begin{array}{l} \min\limits_{\mathbf{x}\in\mathbb{R}^2} f(\mathbf{x}) \\ (x_1 - 1)^2 + x_2^2 - 1 = 0, \\ (x_1 + 1)^2 + x_2^2 - 1 = 0. \end{array}$$

Remark that

$$\nabla h_1(\mathbf{x}) = 2 \begin{pmatrix} x_1 - 1 \\ x_2 \end{pmatrix}, \qquad \nabla h_2(\mathbf{x}) = 2 \begin{pmatrix} x_1 + 1 \\ x_2 \end{pmatrix}.$$

The feasible set $\Omega = \{\mathbf{x} \in \mathbb{R}^2 \mid (x_1 - 1)^2 + x_2^2 - 1 = 0, \ \ (x_1 + 1)^2 + x_2^2 - 1 = 0\}$ is the intersection of two circles that pass from $\mathbf{0}$ but the first one belongs to the right part of the plane while the second one to the left part of the plane $\mathbb{R}^2$, so $\Omega = \{\mathbf{0}\}$.
There does not exist any admissible arc in $\mathbf{0}$, then there does not exist any tangent direction to an admissible arc in $\mathbf{0}$ and $T(\mathbf{0}) = \varnothing$. On the other hand, because

$$F_1(\mathbf{0}) = \{\mathbf{d} \in \mathbb{R}^2 \mid \nabla h_1(\mathbf{0})^T \mathbf{d} = 0, \ \ \nabla h_2(\mathbf{0})^T \mathbf{d} = 0\} = \{\mathbf{d} \in \mathbb{R}^2 \mid (-2, 0)\mathbf{d} = 0, \ \ (2, 0)\mathbf{d} = 0\},$$

it holds $\mathbf{d} \in F_1(\mathbf{0}) \ \forall \mathbf{d} = (0, d_2) : d_2 \neq 0$; then $F_1(\mathbf{0}) \neq \varnothing$.

Definition 6.3.1. *We say that in* $\mathbf{x}$ *the LICQ (Linear Indipendence Constrains Qualification) holds if the set*

$$\{\nabla h_i(\mathbf{x}) \ \ i = 1, \ldots, p, \qquad \nabla g_i(\mathbf{x}) \ \ i \in \mathcal{A}(\mathbf{x})\}$$

is formed by linearly independent vectors.

Examples In example (1) the LICQ does not hold in $\mathbf{x}$ because $\nabla h(\mathbf{x}) = \mathbf{0}$. In example (2) the LICQ does not hold in $\mathbf{0}$ because $\nabla h_1(\mathbf{0}) \parallel \nabla h_2(\mathbf{0})$.

Lemma 6.3.1. *If LICQ holds in* $\mathbf{x}$*, then* $T(\mathbf{x}) = F_1(\mathbf{x})$.

When the LICQ holds to find $T(\mathbf{x})$ we can rather compute $F_1(\mathbf{x})$, whose computation is much more simple.

Example Let us consider the problem

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^n} x_1 + x_2 \\ &2 - x_1^2 - x_2^2 \geq 0 \\ &x_2 \geq 0 \end{aligned}$$

We know that

$$\nabla g_1(\mathbf{x}) = -2\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \qquad \nabla g_2(\mathbf{x}) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Let $\mathbf{x} = (0, \sqrt{2})^T$. It holds $\mathcal{A}(\mathbf{x}) = \{1\}$, $\nabla g_1(\mathbf{x}) = (0, -2\sqrt{2})^T$. The LICQ holds in $\mathbf{x}$, so

$$\begin{aligned} T(\mathbf{x}) = F_1(\mathbf{x}) &= \{\mathbf{d} \in \mathbb{R}^2 \mid \nabla g_1(\mathbf{x})^T \mathbf{d} \geq 0\} = \left\{ \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; (0, -2\sqrt{2}) \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \geq 0 \right\} \\ &= \left\{ \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; -2\sqrt{2} d_2 \geq 0 \right\} = \left\{ \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; d_2 \leq 0 \right\}. \end{aligned}$$

Let now $\mathbf{x} = (-\sqrt{2}, 0)^T$. It holds $\mathcal{A}(\mathbf{x}) = \{1, 2\}$, $\nabla g_1(\mathbf{x}) = (2\sqrt{2}, 0)^T$, $\nabla g_2(\mathbf{x}) = (0, 1)^T$, then the LICQ holds in $\mathbf{x}$ and

$$\begin{aligned} T(\mathbf{x}) = F_1(\mathbf{x}) &= \{\mathbf{d} \in \mathbb{R}^2 \mid \nabla g_1(\mathbf{x})^T \mathbf{d} \geq 0, \;\; \nabla g_2(\mathbf{x})^T \mathbf{d} \geq 0\} \\ &= \left\{ \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; (2\sqrt{2}, 0) \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \geq 0, \;\; (0, 1) \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \geq 0 \right\} = \left\{ \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; 2\sqrt{2} d_1 \geq 0, \;\; d_2 \geq 0 \right\} \\ &= \left\{ \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; d_1 \geq 0, d_2 \geq 0 \right\}. \end{aligned}$$

Remark 6.3.1. *If in* $\mathbf{x}$ *the LICQ does not hold, we cannot derive a necessary condition of the form (NC). For example we can analyse the problem*

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^2} x_1 + x_2 \\ &(2 - x_1^2 - x_2^2)^2 = 0. \end{aligned}$$

For each $\mathbf{x} \in \Omega$*, the LICQ does not hold in* $\mathbf{x}$ *because* $\nabla h(\mathbf{x}) = \mathbf{0}$*. This makes it impossible to proceed as in the case of just one equality constraint to obtain (NC).*

Lemma 6.3.2.

$$\mathbf{x}^* \textit{ local minimum point} \quad \Longrightarrow \quad \nexists \mathbf{d} \in T(\mathbf{x}^*) \textit{ s.t. } \nabla f(\mathbf{x}^*)^T \mathbf{d} < 0$$

Proof. Let $\mathbf{d} \in T(\mathbf{x}^*)$, i.e., $\mathbf{d} = \boldsymbol{\alpha}'(0)$ for some $\boldsymbol{\alpha}(\vartheta)$ admissible arc in $\mathbf{x}^*$. Because $\boldsymbol{\alpha}(\vartheta)$ is an admissible arc in $\mathbf{x}^*$ and because $\mathbf{x}^*$ is the constrained minimum point, moving from $\mathbf{x}^* = \boldsymbol{\alpha}(0)$ along $\boldsymbol{\alpha}(\vartheta)$ we will find larger values of f:

$$0 \quad \leq \quad \left[\frac{\mathrm{d}f(\boldsymbol{\alpha}(\vartheta))}{\mathrm{d}\vartheta}\right]_{\vartheta=0} = \nabla f(\boldsymbol{\alpha}(0))^T \boldsymbol{\alpha}'(0) = \nabla f(\mathbf{x}^*)^T \mathbf{d}.$$

□

Theorem 6.3.1. *First order necessary condition*

$$\begin{cases} \mathbf{x}^* \text{ is a local minimum point} \\ \text{in } \mathbf{x}^* \text{ LICQ holds} \end{cases} \quad \Longrightarrow \quad \nexists \mathbf{d} \in F_1(\mathbf{x}^*) \text{ s.t. } \nabla f(\mathbf{x}^*)^T \mathbf{d} < 0.$$

The necessary condition is not sufficient. Let's see this with an example. Let us consider the problem

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^2} x_2 \\ &x_1^2 + x_2 \geq 0. \end{aligned}$$

It holds

$$\Omega = \{\mathbf{x} \in \mathbb{R}^2 \mid x_2 \geq -x_1^2\}.$$

Remark that f is not lower bounded in Ω, then the problem does not admit a solution.
Let us consider the point $\mathbf{x}^* = (0,0)^T$. In $\mathbf{x}^*$ the LICQ holds because

$$\nabla g(\mathbf{x}^*) = \Big[\nabla g(\mathbf{x})\Big]_{\mathbf{x}=\mathbf{x}^*} = \left[\begin{pmatrix} 2x_1 \\ 1 \end{pmatrix}\right]_{\mathbf{x}=\mathbf{x}^*} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \neq \mathbf{0}.$$

Because in $\mathbf{x}^*$ the constraint is active, it holds

$$F_1(\mathbf{x}^*) = \{\mathbf{d} \in \mathbb{R}^2 \mid \nabla g(\mathbf{x}^*)^T \mathbf{d} \geq 0\} = \left\{ \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \geq 0 \right\} = \left\{ \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; d_2 \geq 0 \right\}.$$

Remark that $\forall \mathbf{d} \in F_1(\mathbf{x}^*)$

$$\nabla f(\mathbf{x}^*)^T \mathbf{d} = \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} = d_2 \geq 0,$$

but $\mathbf{x}^*$ is not a solution.

6.3.1 Farkas Lemma

Theorem 6.3.2. *Farkas Lemma. Let $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. Then exactly one of the following sets must be empty:*

(i) $\{x \mid Ax = b,\ x \geq 0\}$,

(ii) $\{y \mid A^T y \leq 0,\ b^T y > 0\}$.

Remark:

- Systems (i) and (ii) are called strong alternatives, meaning that exactly one of them can be feasible.

- This theorem will be used also in Chapter 8 for proving infeasibility of a linear programming problem via an explicit and easily-verifiable certificate.

The proof of the theorem is in TD6.

Geometric interpretation of the Farkas lemma. For the geometric interpretation of the Farkas lemma we need some ingredients.

Theorem 6.3.3. *Separating hyperplane theorem. Let C and D be two convex sets in $\mathbb{R}^n$ that do not intersect (i.e., $C \cap D = \emptyset$). Then, there exists $\mathbf{a} \in \mathbb{R}^n$, $\mathbf{a} \neq 0$, $b \in \mathbb{R}$, such that $\mathbf{a}^T\mathbf{x} \leq b$ for all $\mathbf{x} \in C$ and $\mathbf{a}^T\mathbf{x} \geq b$ for all $\mathbf{x} \in D$.*

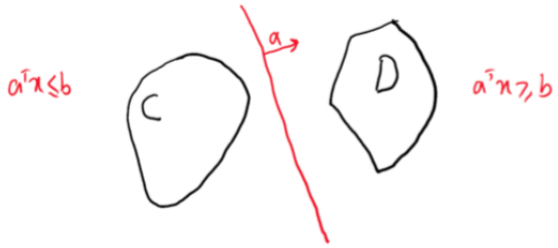


Figure 6.3: Illustration of the separating hyperplane theorem. The two convex sets are separated by the hyperplane $\mathbf{a}^T\mathbf{x} - b = 0$.

Definition 6.3.2. *Cone. A set $K \subset \mathbb{R}^n$ is a cone if $\mathbf{x} \in K$ implies $\alpha\mathbf{x} \in K$ for any scalar $\alpha \geq 0$.*

Definition 6.3.3. *Conic hull. Given a set of points $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, the conic hull of S, denoted by* cone(S)*, is the set of all conic combinations of the points in S:*

$$\text{cone}(S) = \Big\{ \sum_{i=1}^{n} \alpha_i \mathbf{x}_i | \alpha_i \geq 0, \mathbf{x}_i \in S \Big\}.$$

The geometric interpretation of Farkas lemma is then the following. Let $S = \{\tilde{\mathbf{a}}_1, \ldots, \tilde{\mathbf{a}}_n\}$ be the set of the columns of A and let cone(S) be the cone of all their nonnegative combinations. If $b \notin$ cone(S), then we can separate it from the cone with a hyperplane.

We consider an equivalent alternative formulation of Farkas lemma, that will be useful in the following. See TD6 for the proof.

Lemma 6.3.3. *Farkas Lemma*
Given $C \in \mathbb{R}^{n \times p}, B \in \mathbb{R}^{n \times M}$ we consider the following cone of $\mathbb{R}^n$:

$$K = \{C\mathbf{w} + B\mathbf{y} \mid \mathbf{w} \in \mathbb{R}^p, \mathbf{y} \in \mathbb{R}^M, \mathbf{y} \geq \mathbf{0}\}$$

For each $\mathbf{g} \in \mathbb{R}^n$ exactly one of the following sentences is true:

(a) $\mathbf{g} \in K$

(b) $\exists \mathbf{d} \in \mathbb{R}^n$ *s.t.* $\begin{cases} \mathbf{g}^T\mathbf{d} < 0 & (b.1) \\ C^T\mathbf{d} = \mathbf{0} & (b.2) \\ B^T\mathbf{d} \geq \mathbf{0} & (b.3) \end{cases}$

(a) and (b) are each other opposite: (b) $\iff \neg$(a).

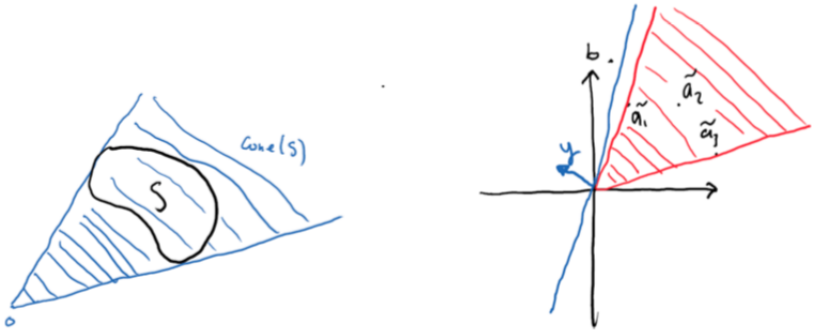


Figure 6.4: Left: conic hull. Right: geometrical interpretation of the Farkas lemma. The separating hyperplane is $\mathbf{y}^T\mathbf{x} - c = 0$, for some constant c.

6.3.2 KKT conditions

Theorem 6.3.4. *KKT, first order necessary conditions*
If $\mathbf{x}^*$ is a solution in which the LICQ holds, then there exist $\boldsymbol{\mu}^* \in \mathbb{R}^p, \boldsymbol{\lambda}^* \in \mathbb{R}^m$ such that

$$\begin{cases} \nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*) = \mathbf{0}, \\ \mathbf{h}(\mathbf{x}^*) = \mathbf{0}, \\ \mathbf{g}(\mathbf{x}^*) \geq \mathbf{0}, \\ \boldsymbol{\lambda}^* \geq \mathbf{0}, \\ \boldsymbol{\lambda}^{*T}\mathbf{g}(\mathbf{x}^*) = 0. \end{cases}$$

In this case we say that $(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*)$ satisfies the Karush-Kuhn-Tucker conditions, usually called KKT conditions. The last condition is the complementarity condition, which can be written also as $\lambda_i^* g(\mathbf{x}^*)_i = 0$ for all $i = 1, \dots, n$, because of the positivity of the vectors.

Proof. Let $\mathbf{x}^*$ be a solution in which the LICQ holds.
Let

$$\begin{aligned} \mathbf{g} &= \nabla f(\mathbf{x}^*), \\ C &= \Big(\nabla h_1(\mathbf{x}^*) \mid \cdots \mid \nabla h_p(\mathbf{x}^*)\Big) \in \mathbb{R}^{n\times p}, \\ B &= \Big(\nabla g_{i_1}(\mathbf{x}^*) \mid \cdots \mid \nabla g_{i_M}(\mathbf{x}^*)\Big) \in \mathbb{R}^{n\times M}, \qquad \{i_1, \dots, i_M\} = \mathcal{A}(\mathbf{x}^*) \end{aligned}$$

and

$$K = \{C\mathbf{w} + B\mathbf{y} \mid \mathbf{w} \in \mathbb{R}^p, \mathbf{y} \in \mathbb{R}^M, \mathbf{y} \geq \mathbf{0}\}. \tag{6.1}$$

With this notation it holds

$$\begin{aligned} F_1(\mathbf{x}^*) &= \{\mathbf{d} \in \mathbb{R}^n \mid \nabla h_i(\mathbf{x}^*)^T\mathbf{d} = 0 \ \ \forall i = 1, \dots, p, \ \ \nabla g_i(\mathbf{x}^*)^T\mathbf{d} \geq 0 \ \ \forall i \in \mathcal{A}(\mathbf{x}^*)\} \\ &= \{\mathbf{d} \in \mathbb{R}^n \mid \nabla h_i(\mathbf{x}^*)^T\mathbf{d} = 0 \ \ \forall i = 1, \dots, p, \ \ \nabla g_{i_j}(\mathbf{x}^*)^T\mathbf{d} \geq 0 \ \ \forall j = 1, \dots, M\} \\ &= \{\mathbf{d} \in \mathbb{R}^n \mid (C\mathbf{e}_i)^T\mathbf{d} = 0 \ \ \forall i = 1, \dots, p, \ \ (B\mathbf{e}_j)^T\mathbf{d} \geq 0 \ \ \forall j = 1, \dots, M\} \\ &= \{\mathbf{d} \in \mathbb{R}^n \mid \mathbf{e}_i^T C^T\mathbf{d} = 0 \ \ \forall i = 1, \dots, p, \ \ \mathbf{e}_j^T B^T\mathbf{d} \geq 0 \ \ \forall j = 1, \dots, M\} \\ &= \{\mathbf{d} \in \mathbb{R}^n \mid (C^T\mathbf{d})_i = 0 \ \ \forall i = 1, \dots, p, \ \ (B^T\mathbf{d})_j \geq 0 \ \ \forall j = 1, \dots, M\} \\ &= \{\mathbf{d} \in \mathbb{R}^n \mid C^T\mathbf{d} = \mathbf{0}, \ \ B^T\mathbf{d} \geq \mathbf{0}\}. \end{aligned}$$

By the first order necessary condition in Theorem 6.3.1, being $\mathbf{x}^*$ a solution in which the LICQ holds,

$$\nexists \mathbf{d} \in F_1(\mathbf{x}^*) \text{ s.t. } \mathbf{g}^T\mathbf{d} < 0,$$

that is

$$\nexists \mathbf{d} \in \mathbb{R}^n \text{ s.t. } \begin{cases} \mathbf{g}^T\mathbf{d} < 0 \\ C^T\mathbf{d} = \mathbf{0} \\ B^T\mathbf{d} \geq \mathbf{0} \end{cases}$$

Condition (b) of Farkas Lemma does not hold, then (a) must hold and

$$\mathbf{g} \in K.$$

This means that $\nabla f(\mathbf{x}^*) \in K$, that is there exist $\mathbf{w}^* \in \mathbb{R}^p, \mathbf{y}^* = \begin{pmatrix} y^*_{i_1} \\ \vdots \\ y^*_{i_M} \end{pmatrix} \in \mathbb{R}^M, \mathbf{y}^* \geq \mathbf{0}$ such that

$$\nabla f(\mathbf{x}^*) = C\mathbf{w}^* + B\mathbf{y}^* = \Big(\nabla h_1(\mathbf{x}^*) \mid \cdots \mid \nabla h_p(\mathbf{x}^*)\Big)\mathbf{w}^* + \Big(\nabla g_{i_1}(\mathbf{x}^*) \mid \cdots \mid \nabla g_{i_M}(\mathbf{x}^*)\Big)\mathbf{y}^*$$
$$= \sum_{i=1}^{p} \nabla h_i(\mathbf{x}^*) w_i^* + \sum_{j=1}^{M} \nabla g_{i_j}(\mathbf{x}^*) y^*_{i_j} = \sum_{i=1}^{p} w_i^* \nabla h_i(\mathbf{x}^*) + \sum_{i \in \mathcal{A}(\mathbf{x}^*)} y_i^* \nabla g_i(\mathbf{x}^*).$$

Set

$$\boldsymbol{\mu}^* = \mathbf{w}^*,$$

$$\boldsymbol{\lambda}^* = \begin{pmatrix} \lambda_1^* \\ \vdots \\ \lambda_m^* \end{pmatrix}, \qquad \lambda_i^* = \begin{cases} y_i^* & \text{if } i \in \mathcal{A}(\mathbf{x}^*) \\ 0 & \text{otherwise} \end{cases}$$

There exist $\boldsymbol{\mu}^* \in \mathbb{R}^p, \boldsymbol{\lambda}^* \in \mathbb{R}^m : \boldsymbol{\lambda}^* \geq \mathbf{0}, \lambda_i^* g_i(\mathbf{x}^*) = 0 \ \ \forall i = 1, \ldots, m$ such that

$$\nabla f(\mathbf{x}^*) = \sum_{i=1}^{p} \mu_i^* \nabla h_i(\mathbf{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(\mathbf{x}^*),$$

that is, (because $\boldsymbol{\lambda}^* \geq \mathbf{0}$ and $\mathbf{g}(\mathbf{x}^*) \geq \mathbf{0}$) there exist $\boldsymbol{\mu}^* \in \mathbb{R}^p, \boldsymbol{\lambda}^* \in \mathbb{R}^m : \boldsymbol{\lambda}^* \geq \mathbf{0}, \boldsymbol{\lambda}^{*T}\mathbf{g}(\mathbf{x}^*) = 0$ such that

$$\nabla f(\mathbf{x}^*) - \sum_{i=1}^{p} \mu_i^* \nabla h_i(\mathbf{x}^*) - \sum_{i=1}^{m} \lambda_i^* \nabla g_i(\mathbf{x}^*) = \mathbf{0}.$$

Because

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = \nabla f(\mathbf{x}) - \sum_{i=1}^{p} \mu_i \nabla h_i(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i \nabla g_i(\mathbf{x}),$$

there exist $\boldsymbol{\mu}^* \in \mathbb{R}^p, \boldsymbol{\lambda}^* \in \mathbb{R}^m : \boldsymbol{\lambda}^* \geq \mathbf{0}, \boldsymbol{\lambda}^{*T}\mathbf{g}(\mathbf{x}^*) = 0$ such that

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*) = \mathbf{0}.$$

□

Remark 6.3.2. *This necessary condition is not sufficient. Let's see this with an example. We consider again the problem*

$$\begin{aligned}&\min_{\mathbf{x}\in\mathbb{R}^2} x_2\\ &x_1^2 + x_2 \geq 0.\end{aligned}$$

We have seen that in $\mathbf{x}^* = (0,0)^T$ *the LICQ holds. Because*

$$\begin{aligned}\mathcal{L}(\mathbf{x},\lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x}) = x_2 - \lambda x_1^2 - \lambda x_2,\\ \nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x},\lambda) = \begin{pmatrix}-2\lambda x_1\\ 1-\lambda\end{pmatrix},\\ \nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^*,\lambda) = \begin{pmatrix}0\\ 1-\lambda\end{pmatrix},\end{aligned}$$

it holds

$$\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^*,\lambda) = \mathbf{0} \quad \iff \quad \lambda = 1,$$

then $(\mathbf{x}^*,\lambda^*)$ *with* $\lambda^* = 1$ *that satisfies the KKT because in* $\mathbf{x}^*$ *the constraint is active. But* $\mathbf{x}^*$ *is not a solution.*

Lemma 6.3.4. *Let* $\mathbf{x}^*$ *be a solution in which the LICQ is satisfied. The first order necessary condition and the KKT conditions are equivalent:*

$$\nexists \mathbf{d} \in F_1(\mathbf{x}^*) \quad s.t. \quad \nabla f(\mathbf{x}^*)^T\mathbf{d} < 0 \iff \exists \boldsymbol{\mu}^* \in \mathbb{R}^p, \boldsymbol{\lambda}^* \in \mathbb{R}^m \text{ s.t. } (\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*) \text{ satisfies the KKT.}$$

Proof. ($\Longrightarrow$) is the prof of the previous theorem.

($\Longleftarrow$) Because $(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*)$ satisfies KKT conditions it holds

$$\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathbf{0},$$

that is

$$\nabla f(\mathbf{x}^*) = \sum_{i=1}^{p} \mu_i^* \nabla h_i(\mathbf{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(\mathbf{x}^*),$$

and then

$$\begin{aligned}\nabla f(\mathbf{x}^*) &= \sum_{i=1}^{p} \mu_i^* \nabla h_i(\mathbf{x}^*) + \sum_{\substack{i=1\\ i\in\mathcal{A}(\mathbf{x}^*)}}^{m} \lambda_i^* \nabla g_i(\mathbf{x}^*) + \sum_{\substack{i=1\\ i\notin\mathcal{A}(\mathbf{x}^*)}}^{m} \underbrace{\lambda_i^*}_{\substack{=0 \text{ for}\\ \text{complementary}}} \nabla g_i(\mathbf{x}^*)\\ &= \sum_{i=1}^{p} \mu_i^* \nabla h_i(\mathbf{x}^*) + \sum_{\substack{i=1\\ i\in\mathcal{A}(\mathbf{x}^*)}}^{m} \underbrace{\lambda_i^*}_{\substack{\geq 0\\ \text{KKT}}} \nabla g_i(\mathbf{x}^*).\end{aligned}$$

So $\nabla f(\mathbf{x}^*)$ belongs to the cone defined in (6.1) and Farkas Lemma guarantees that there do not exist directions $\mathbf{d} \in F_1(\mathbf{x}^*) \quad s.t. \quad \nabla f(\mathbf{x}^*)^T\mathbf{d} < 0$. □

6.4 Second order optimality conditions

Let us assume that $(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*)$ satisfies the KKT conditions. Then $\nabla f(\mathbf{x}^*)^T\mathbf{d} \geq 0 \ \ \forall \mathbf{d} \in F_1(\mathbf{x}^*)$. If $\nabla f(\mathbf{x}^*)^T\mathbf{d} = 0$, with just first order informations we are not able to establish if along the direction $\mathbf{d}$

the values of f increase or decrease: we need second order informations. Then the directions in the set

$$\{\mathbf{d} \in F_1(\mathbf{x}^*) \mid \nabla f(\mathbf{x}^*)^T \mathbf{d} = 0\}$$

are called critical directions; this set is a cone. If $i \in \mathcal{A}(\mathbf{x}^*)$, i.e., $g_i(\mathbf{x}^*) = 0$, and $\lambda_i^* = 0$ then the i-th inequality constraint is said to be degenerate.
Remark that, given $\mathbf{d} \in F_1(\mathbf{x}^*)$,

$$\begin{aligned}
\nabla f(\mathbf{x}^*)^T \mathbf{d} = 0 \quad & \underbrace{\iff}_{\substack{\text{proof of} \\ \text{prev. thm.}}} \quad \sum_{\substack{i=1 \\ i \in \mathcal{A}(\mathbf{x}^*)}}^{m} \lambda_i^* \nabla g_i(\mathbf{x}^*)^T \mathbf{d} = 0 \\
\iff \quad & \sum_{\substack{i=1 \\ i \in \mathcal{A}(\mathbf{x}^*) \\ i \text{ degenerate}}}^{m} \underbrace{\lambda_i^*}_{=0} \nabla g_i(\mathbf{x}^*)^T \mathbf{d} \; + \sum_{\substack{i=1 \\ i \in \mathcal{A}(\mathbf{x}^*) \\ i \text{ non degenerate}}}^{m} \underbrace{\lambda_i^*}_{>0} \underbrace{\nabla g_i(\mathbf{x}^*)^T \mathbf{d}}_{\substack{\geq 0 \\ \mathbf{d} \in F_1(\mathbf{x}^*)}} = 0 \\
\iff \quad & \nabla g_i(\mathbf{x}^*)^T \mathbf{d} = 0 \;\; \forall i \in \mathcal{A}(\mathbf{x}^*) : \; i \text{ non degenerate} \\
\iff \quad & \nabla g_i(\mathbf{x}^*)^T \mathbf{d} = 0 \;\; \forall i \in \mathcal{A}(\mathbf{x}^*) : \lambda_i^* > 0.
\end{aligned}$$

i.e.,

$$\{\mathbf{d} \in F_1(\mathbf{x}^*) \mid \nabla f(\mathbf{x}^*)^T \mathbf{d} = 0\} = \{\mathbf{d} \in F_1(\mathbf{x}^*) \mid \nabla g_i(\mathbf{x}^*)^T \mathbf{d} = 0 \;\; \forall i \in \mathcal{A}(\mathbf{x}^*) : \lambda_i^* > 0\} := \mathcal{C}(\mathbf{x}^*, \boldsymbol{\lambda}^*).$$

This last set is the *critical cone* in $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$.

Theorem 6.4.1. *Second order necessary condition*
Let $\mathbf{x}^*$ a solution in which LICQ holds. By the first order necessary condition we know that there exist $\boldsymbol{\mu}^* \in \mathbb{R}^p, \boldsymbol{\lambda}^* \in \mathbb{R}^m$ such that $(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*)$ satisfies the KKT. Then

$$\mathbf{d}^T H_{\mathcal{L},\mathbf{x}}(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*) \mathbf{d} \geq 0 \qquad \forall \mathbf{d} \in \mathcal{C}(\mathbf{x}^*, \boldsymbol{\lambda}^*),$$

i.e., the matrix

$$H_{\mathcal{L},\mathbf{x}}(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*) = \begin{pmatrix} \dfrac{\partial \mathcal{L}}{\partial x_1 \partial x_1}(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*) & \cdots & \dfrac{\partial \mathcal{L}}{\partial x_1 \partial x_n}(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*) \\ \vdots & & \vdots \\ \dfrac{\partial \mathcal{L}}{\partial x_n \partial x_1}(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*) & \cdots & \dfrac{\partial \mathcal{L}}{\partial x_n \partial x_n}(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*) \end{pmatrix}$$

is the Hessian of $\mathcal{L}(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda})$ with respect to $\mathbf{x}$ in $(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*)$ is semipositive definite with respect to the vectors of the critical cone $\mathcal{C}(\mathbf{x}^*, \boldsymbol{\lambda}^*)$.

Theorem 6.4.2. *Second order sufficient condition*
Let $\mathbf{x}^* \in \Omega$ for which there exist $\boldsymbol{\mu}^* \in \mathbb{R}^p, \boldsymbol{\lambda}^* \in \mathbb{R}^m$ such that $(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*)$ satisfies the KKT. If

$$\mathbf{d}^T H_{\mathcal{L},\mathbf{x}}(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*) \mathbf{d} > 0 \qquad \forall \mathbf{d} \in \mathcal{C}(\mathbf{x}^*, \boldsymbol{\lambda}^*), \mathbf{d} \neq \mathbf{0},$$

i.e. if $H_{\mathcal{L},\mathbf{x}}(\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*)$ is positive definite with respect to the vectors of the critical cone $\mathcal{C}(\mathbf{x}^*, \boldsymbol{\lambda}^*)$, then $\mathbf{x}^*$ is a solution.

Remark 6.4.1. *In the unconstrained case, i.e. in the case in which $\Omega = \mathbb{R}^n$, it holds $\mathcal{L} = f$ and the set of admissible directions and the critical cone both are $\mathbb{R}^n$, then these conditions are equivalent to those we have seen in the first part of the course.*

Example 1

Let us consider again problem

$$\begin{array}{l}\min\limits_{\mathbf{x}\in\mathbb{R}^2} x_2\\ x_1^2 + x_2 \geq 0\end{array}.$$

We have seen that in $\mathbf{x}^* = (0,0)^T$ the LICQ holds and that $(\mathbf{x}^*, \lambda^*)$ with $\lambda^* = 1$ satisfies the KKT. We have also seen that

$$F_1(\mathbf{x}^*) = \left\{ \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; d_2 \geq 0 \right\},$$

then

$$\begin{aligned}\mathcal{C}(\mathbf{x}^*, \lambda^*) &= \{\mathbf{d} \in F_1(\mathbf{x}^*) \mid \nabla g(\mathbf{x}^*)^T \mathbf{d} = 0\} = \left\{ \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; d_2 \geq 0, \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} = 0 \right\} \\ &= \left\{ \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; d_2 = 0 \right\} = \left\{ \begin{pmatrix} d_1 \\ 0 \end{pmatrix} \;\middle|\; d_1 \in \mathbb{R} \right\}.\end{aligned}$$

We have seen that

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda) = \begin{pmatrix} -2\lambda x_1 \\ 1 - \lambda \end{pmatrix},$$

then

$$H_{\mathcal{L},\mathbf{x}}(\mathbf{x}, \lambda) = \begin{pmatrix} -2\lambda & 0 \\ 0 & 0 \end{pmatrix}$$

and

$$H_{\mathcal{L},\mathbf{x}}(\mathbf{x}^*, \lambda^*) = \begin{pmatrix} -2 & 0 \\ 0 & 0 \end{pmatrix}.$$

Remark that $\forall \mathbf{d} \in \mathcal{C}(\mathbf{x}^*, \lambda)$, that is for each $\mathbf{d} = \begin{pmatrix} d_1 \\ 0 \end{pmatrix}$ for some $d_1 \in \mathbb{R}$, it holds

$$\mathbf{d}^T H_{\mathcal{L},\mathbf{x}}(\mathbf{x}^*, \lambda^*) \mathbf{d} = \begin{pmatrix} d_1 & 0 \end{pmatrix} \begin{pmatrix} -2 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} d_1 \\ 0 \end{pmatrix} = \begin{pmatrix} d_1 & 0 \end{pmatrix} \begin{pmatrix} -2d_1 & 0 \end{pmatrix} = -2d_1^2 < 0,$$

then the second order necessary condition does not hold, and $\mathbf{x}^*$ is not a solution.

Example 2

Let us consider problem

$$\begin{array}{l}\min\limits_{\mathbf{x}\in\mathbb{R}^2} -\dfrac{1}{10}(x_1 - 4)^2 + x_2^2\\ x_1^2 + x_2^2 - 1 \geq 0.\end{array}$$

After computing

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x}) = -\frac{1}{10}(x_1 - 4)^2 + x_2^2 - \lambda(x_1^2 + x_2^2 - 1),$$

and

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda) = \begin{pmatrix} -(1/5)(x_1 - 4) - 2\lambda x_1 \\ 2x_2(1 - \lambda) \end{pmatrix}$$

it is easy to solve the KKT system

$$\begin{cases} \nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x},\lambda) = \mathbf{0} \\ g(\mathbf{x}) \geq 0 \\ \lambda \geq 0 \\ \lambda g(\mathbf{x}) = 0 \end{cases}$$

and to see that the solutions are

$$\begin{pmatrix} \mathbf{x}_1^* \\ \lambda_1^* \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \dfrac{3}{10} \end{pmatrix}, \qquad \begin{pmatrix} \mathbf{x}_2^* \\ \lambda_2^* \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \\ 0 \end{pmatrix}$$

and

$$\begin{pmatrix} \mathbf{x}_3^* \\ \lambda_3^* \end{pmatrix} = \begin{pmatrix} 4/11 \\ \sqrt{105/121} \\ 1 \end{pmatrix}, \qquad \begin{pmatrix} \mathbf{x}_4^* \\ \lambda_4^* \end{pmatrix} = \begin{pmatrix} 4/11 \\ -\sqrt{105/121} \\ 1 \end{pmatrix}.$$

Let us consider $(\mathbf{x}^*, \lambda^*)$. Because

$$H_{\mathcal{L},\mathbf{x}}(\mathbf{x},\lambda) = \begin{pmatrix} -2\lambda - \dfrac{1}{5} & 0 \\ 0 & 2 - 2\lambda \end{pmatrix},$$

we have that

$$H_{\mathcal{L},\mathbf{x}}(\mathbf{x}^*,\lambda^*) = \begin{pmatrix} -\dfrac{4}{5} & 0 \\ 0 & \dfrac{7}{5} \end{pmatrix},$$

which is an indefinite matrix. But $d^T H_{\mathcal{L},\mathbf{x}}(\mathbf{x}^*,\lambda^*)d > 0$, $\forall d \in \mathcal{C}(\mathbf{x}^*,\lambda^*) = \{d \in \mathbb{R}^2 \mid d_1 = 0\}$. Then, for the second order sufficient condition, $\mathbf{x}^*$ is a solution.

On the contrary, in $(\mathbf{x}_2^*, \lambda_2^*)$ the constraint is inactive, and as the Hessian matrix of the Lagrangian function in $(\mathbf{x}_2^*, \lambda_2^*)$ is indefinite, the point is not a minimum point.

6.5 Solution methods

6.5.1 Projected gradient method

If Ω is the feasible set of the constrained problem, a step of the projected gradient method is:

$$\mathbf{x}_{k+1} = P_\Omega(\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)),$$

that is, a step of the gradient method is taken, followed by the projection of the new iterate onto the feasible set. In this way all the iterates are feasible. This method can be applied only if the projection onto the feasible set is easy to compute and not too expensive, because it needs to be computed at each iteration.

Some important examples of known projection operators are:

- the projection of $\mathbf{z}$ onto $\mathcal{R}(A)$: $P_{\mathcal{R}(A)}(\mathbf{z}) = A(A^TA)^{-1}A^T\mathbf{z} = AA^\dagger\mathbf{z}$, where the last expression is valid only if $m \geq n$ and $\text{rank}(A) = n$.
- the projection of $\mathbf{z}$ onto the nullspace $\mathcal{N}(A)$ of A : $P_{\mathcal{N}(A)}(\mathbf{z}) = (I - P_{\mathcal{R}(A^T)})(F)$
- the projection of $\mathbf{z}$ onto the affine subspace $S = \{\mathbf{x} \mid A\mathbf{x} = \mathbf{b}\}$ with $A \in \mathbb{R}^{m\times n}$ with rank m and $\mathbf{b} \in \mathbb{R}^m$: $P_S(\mathbf{z}) = (I - A^T(AA^T)^{-1}A)z + A^T(AA^T)^{-1}b$.

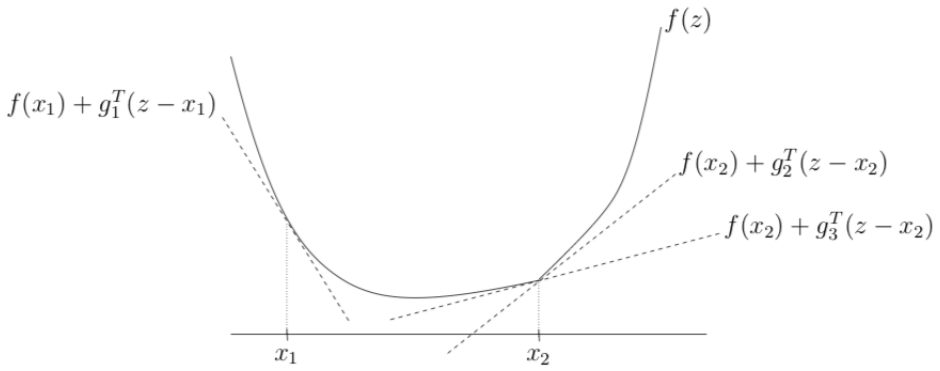


Figure 1: At x_1, the convex function f is differentiable, and g_1 (which is the derivative of f at x_1) is the unique subgradient at x_1. At the point x_2, f is not differentiable. At this point, f has many subgradients: two subgradients, g_2 and g_3, are shown.

Figure 6.5: Subgradients

Variant for non-smooth functions In many problems the objective function f may be non-smooth, for example when a sparsity condition is imposed through an ℓ_1 norm (cf. TD 7). This situation is often encountered in machine learning applications. In such cases the gradient cannot be computed. There exists however a variant of gradient methods, that is called *subgradient method*, which employs subgradients of a nondifferentiable function[3].

We say a vector $\mathbf{g} \in \mathbb{R}^n$ is a subgradient of $f : A \subset \mathbb{R}^n \to \mathbb{R}$ at $\mathbf{x} \in A$ if for all $\mathbf{z} \in A$

$$f(\mathbf{z}) \geq f(\mathbf{x}) + \mathbf{g}^T(\mathbf{z} - \mathbf{x}).$$

If f is convex and differentiable, then its gradient at $\mathbf{x}$ is a subgradient. But a subgradient can exist even when f is not differentiable at $\mathbf{x}$, as illustrated in the following figure. The same example shows that there can be more than one subgradient of a function f at a point $\mathbf{x}$.

A function f is called subdifferentiable at $\mathbf{x}$ if there exists at least one subgradient at $\mathbf{x}$. The set of subgradients of f at the point $\mathbf{x}$ is called the subdifferential of f at $\mathbf{x}$, and is denoted $\partial f(\mathbf{x})$. A function f is called subdifferentiable if it is subdifferentiable at all $\mathbf{x} \in A$.

Example 1. Absolute value. Consider $z \in \mathbb{R}$ and $f(z) = |z|$. For $x < 0$ the subgradient is unique: $\partial f(x) = -1$. Similarly, for $x > 0$ we have $\partial f(x) = 1$. At $x = 0$ the subdifferential is defined by the inequality $|z| \geq gz$ for all z, which is satisfied if and only if $g \in [-1, 1]$. Therefore we have $\partial f(0) = [-1, 1]$.

Example 2. The ℓ_1-norm $f(\mathbf{x}) = \|\mathbf{x}\|_1 = |x_1| + \cdots + |x_n|$ is a nondifferentiable convex function of $\mathbf{x}$. Its subdifferential is given by $\text{sign}(\mathbf{x})$.

If $f(\mathbf{x}) = f_1(\mathbf{x}) + f_2(\mathbf{x})$ is given by the sum of a differentiable function f_1 and a nondifferentiable function f_2, as it is typically the case when a regularization term in the ℓ_1 norm is added to the objective function to enforce sparsity of the solution, the subgradient of f is simply the sum of the gradient of f_1 and the subgradient of f_2.

[3]Reference: `https://see.stanford.edu/materials/lsocoee364b/01-subgradients_notes.pdf`

The subgradient method is used to solve problems of the form

$$\min_{\mathbf{x}} f(\mathbf{x})$$

with f nondifferentiable. Step k of this method is simply defined as looking for a $\mathbf{g}_k \in \partial f(\mathbf{x}_k)$ and by the update $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k$. If the problem also has constraints

$$\min_{\mathbf{x}\in\Omega} f(\mathbf{x})$$

and the projection operator onto Ω is known, we can use a projected subgradient method, whose k-th step is defined as looking for a $\mathbf{g}_k \in \partial f(\mathbf{x}_k)$ and by the update $\mathbf{x}_{k+1} = P_\Omega(\mathbf{x}_k - \alpha_k \mathbf{g}_k)$.

6.5.2 Penalty function methods

The general idea of penalty function methods is to transform a constrained problem into an unconstrained one, by adding the contraints to the objective function, as a penalty term. The simplest penalty function of this type is the quadratic penalty function, in which the penalty terms are the squares of the constraint violations:

$$Q(\mathbf{x}, \mu) = \min_{\mathbf{x}} f(\mathbf{x}) + \frac{1}{2\mu}\sum_{i=1}^{p} h_i(\mathbf{x})^2 + \frac{1}{2\mu}\sum_{i=1}^{m}([g_i(\mathbf{x})]^-)^2$$

where $[y]^- = \max(-y, 0)$ and $\mu > 0$ is a penalty parameter that balances the contribution of the objective function and of the constraints. If μ is too large, the constraints are not penalized with enough severity and the resulting solution may be unfeasible ($\mathbf{x} \notin \Omega$). If on the other hand μ is too small, the contribution of the constraints may be too important with respect to the objective function and the resulting solution may not be optimal for the original problem. The penalty function may be less smooth than the objective and constraint functions. For instance, if one of the inequality constraints is $x_1 \geq 0$, then the function $\min(0, x_1)^2$ has a discontinuous second derivative.

The idea of quadratic penalty function methods is to consider a sequence of values $\{\mu_k\}$ with $\mu_k \downarrow 0$ as $k \to \infty$ and to seek the approximate minimizer $\mathbf{x}_k$ of $Q(\mathbf{x}; \mu_k)$ for each k. We can prove that if each $\mathbf{x}_k$ is the exact global minimizer of $Q(\mathbf{x}; \mu_k)$ and if $\mu_k \downarrow 0$, then every limit point of the sequence $\{\mathbf{x}_k\}$ is a solution of the original problem. Convergence results can be proved also when we allow inexact (but increasingly accurate) minimizations of $Q(\mathbf{x}; \mu_k)$.

Another class of penalty functions is those of the exact penalty functions. These are functions with the property that for certain choices of the parameter μ, a single minimization yields the exact solution of the nonlinear programming problem. An example is given by the ℓ_1 penalty function:

$$\Phi_1(\mathbf{x}, \mu) = \min_{\mathbf{x}} f(\mathbf{x}) + \frac{1}{\mu}\sum_{i=1}^{p}|h_i(\mathbf{x})| + \frac{1}{\mu}\sum_{i=1}^{m}[g_i(\mathbf{x})]^-.$$

It is possible to show that the ℓ_1 function is an exact penalty function for all $\mu < \nu^*$, with

$$\nu^* = \max\{|\mu_i^*|, i = 1, \ldots, p,\ \lambda_i^*, i = 1, \ldots, m\},$$

the Lagrangian multipliers at the solution. For all sufficiently small, positive values of μ, one minimization of this unconstrained function will yield the solution of the nonlinear program. It is however difficult to determine μ a priori in most practical applications; rules for adjusting this parameter during the course of the computation are then usually devised. Minimization of Φ_1 is moreover made difficult by the fact that it is nonsmooth.

6.5.3 Sequential programming methods

The idea of sequential programming methods is to model the original constrained problem at the current iterate $\mathbf{x}_k$ by a linear or quadratic programming subproblem and to use the minimizer of this subproblem to define a new iterate $\mathbf{x}_{k+1}$.

In the subproblems the constraints are linearized:

$$h_i(\mathbf{x}) + \nabla h_i(\mathbf{x})^T \mathbf{p} = 0,$$
$$g_i(\mathbf{x}) + \nabla g_i(\mathbf{x})^T \mathbf{p} \geq 0$$

and the objective function is approximated by a linear or quadratic approximation:

$$\frac{1}{2}\mathbf{p}^T W_k \mathbf{p} + \nabla f(\mathbf{x}_k)^T \mathbf{p}$$

where $W_k = 0$ in sequential linear programming (SLP) methods and W_k is the Hessian of the Lagrangian function with respect to $\mathbf{x}$ in sequential quadratic programming methods (SQP). The simplest derivation of SQP methods views them as an application of Newton's method to the KKT optimality conditions for the constrained problem, that's why the arising matrix W_k is defined in this way (cf. chapter 18 of the book). Efficient solvers exist for the solution of such subproblems.

Chapter 7

Optimization methods for Machine Learning

The reference for this part is *Optimization Methods for Large-Scale Machine Learning*, by Leon Bottou, Frank E. Curtis, Jorge Nocedal, SIAM Review, Vol. 60, No. 2, pp. 223-311.

The most part of machine learning problems require the solution of a large-scale optimization problem, in the so-called *training phase*. This is the case for example of regression and classification problems, that are based on a collection of couples $\{(\mathbf{w}_i, y_i)\}$, $i = 1, \dots, m$ where $\mathbf{w}_i$ are called samples and y_i are called labels. In regression problems $y_i \in \mathbb{R}$, while in classification problems $y_i \in \{1, \dots, M\}$, where M is the number of classes into which the data can be divided, usually binary classification problems are considered in which $M = 2$ and $y_i \in \{0, 1\}$ (cf. Figure 7.1). The set $\{(\mathbf{w}_i, y_i)\}$, $i = 1, \dots, m$ is called a *training set* and it can be considered as a set of m realizations of the couple of random variables w, y that obeys to a hidden probability law $P(w, y)$. The aim of supervised learning is to use the available data to build a model h capable of predicting the labels for unseen examples (fit the data in regression problems, assign them to the correct class for classification problems). To do so, a loss function ℓ is chosen that measures a cost for predicting $h(\mathbf{x}; \mathbf{w}_i)$ when the true label is y_i ($\mathbf{x} \in \mathbb{R}^n$ are the variables of the model). A famous example is given by the least-squares loss function: $\ell(h(\mathbf{x}; \mathbf{w}_i) - y_i) = [h(\mathbf{x}; \mathbf{w}_i) - y_i]^2$ that we have studied in Chapter 5. To find the best parameters we can solve an optimization problem that is usually formulated as

$$\min_{\mathbf{x}} f_m(\mathbf{x}) = \frac{1}{m} \sum_{i=1}^{m} f_i(\mathbf{x}), \tag{7.1}$$

where $f_i = \ell(h(\mathbf{x}; \mathbf{w}_i) - y_i)$. This is the training phase and f_m is usually called the *empirical risk*.

Problem (7.1) can be seen as a discrete version of the true objective function. Ideally indeed, one would like to chose the parameter vector $\mathbf{x}$ to minimize the expected loss that would be incurred from any input-output pair $(\mathbf{w}_i, y_i)$. To state this idea formally, we consider the unknown probability distribution $P(w, y)$ representing the true relationship between inputs and outputs. The objective function we wish to minimize is the *expected risk*

$$f(\mathbf{x}) = \int \ell(h(\mathbf{x}; \mathbf{w}), \mathbf{y}) dP(\mathbf{w}, \mathbf{y}) = \mathbb{E}[\ell(h(\mathbf{x}; \mathbf{w}), \mathbf{y})].$$

While it may be desirable to minimize f, such a goal is untenable when one does not have complete

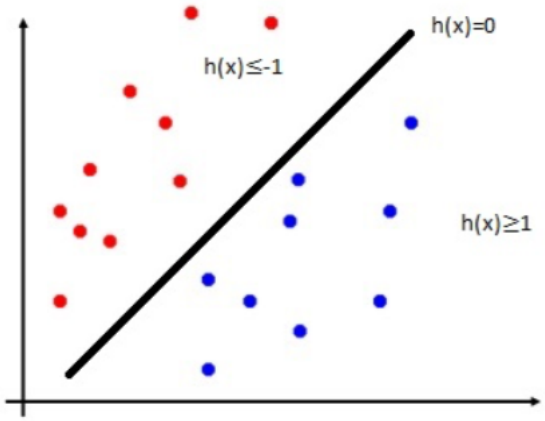


Figure 7.1: Binary classification problem where the samples $\mathbf{x}$ are separated by a hyperplane $h(\mathbf{x}) = \mathbf{z}^T\mathbf{x} + b$, that depends on the parameters $(\mathbf{z}, b) \in \mathbb{R}^{n+1}$.

information about P. Thus, in practice, one seeks the solution of problem (7.1) that involves the empirical risk as an estimate of the expected risk.

Given such a set-up, one has to be careful in the minimization of f_m. In particular finding an exact minimizer should be avoided because the risk is to do an *overtraining*: the model is adapted too well to the training set and is not well suited to generalize to unseen samples. That's why when employing an algorithm for minimizing f_m, an early stopping criterion is used, which means that the algorithm is stopped before an actual minimizer is found. This is explained in Figure 7.2.

Prior to the training phase the available data are divided into three sets: the training set, the validation set and the test set. The validation set is used to choose the hyperparameters of the method (additional parameters than the training parameters, that helps defining the "structure" of the model, for example the number of layers for a neural network, the number of neurons in the neural network, the degree of a polynomial model, and that help to balance the model complexity. A complex model adapts better to the data but will have less predictive power, cf. Figure 7.3). Over the training set, one minimizes an empirical risk measure f_m for various values of the hyperparameters. This results in a handful of candidate functions. The validation set is then used to estimate the expected risk corresponding to each candidate solution, after which one chooses the function yielding the lowest estimated risk value. The testing set is a set of samples that are not used during the training and are then used to estimate the prediction capacity of the model. The empirical risk computed on the testing set gives a better estimation of f than the value of f_m computed using the training samples. What we would like to have is a measure of the error expected in the prediction of unseen samples and from Figure 7.2 it is clear that we cannot employ the error achieved during training as a measure of this prediction error.

Being m usually very large, the simple evaluation of the objective function f_m becomes very expensive, not to mention the computation of the derivatives. In many problems, for example when neural networks are used, also the dimension of the problem n may be really large. For this reason classical optimization methods are not well-suited for these applications and new methods have been proposed. Consider for example Newton's method. We have not only to compute first and second order derivatives for m functions, but also to solve a linear system of size n at each iteration. This

Figure 7.2: Motivation for early stopping. Comparison between the empirical risk f_m and the expected risk f.

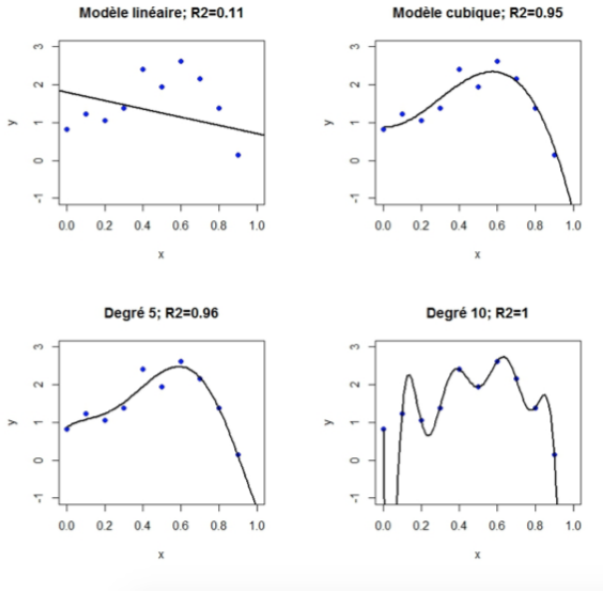


Figure 7.3: Models of increasing complexity (polynomial models of increasing degree). Polynomials with higher degree fit better the data, but will not be good in predicting unseen data.

may be impractical for large problems. We will see in the following some approaches to make the solution of (7.1) practical.

7.1 Stochastic gradient (SG)

The stochastic gradient method is a variant of the classical gradient method. It is an iterative method that starts from a given point $\mathbf{x}_0$. Its general iterative scheme is as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f_{i_k}(\mathbf{x}_k),$$

where α_k is a given step-length, i_k is an index among $\{1, \ldots, m\}$ that is randomly chosen at each iteration. Each iteration of this method is thus very cheap, involving only the computation of the gradient $\nabla f_{i_k}(\mathbf{x}_k)$, corresponding to one sample. This is a stochastic algorithm (as opposed to the algorithms that we have studied until now that are deterministic), meaning that the generated sequence is not uniquely determined from the starting point $\mathbf{x}_0$, but depends on the random sequence of indexes $\{i_k\}$. Note that each direction $-\nabla f_{i_k}(\mathbf{x}_k)$ might not be one of descent for f_m (in the sense of yielding a negative directional derivative), but we can show that if it is a descent direction in expectation (the expected value of the scalar product with $\nabla f(\mathbf{x}_k)$ is negative), then the sequence $\{\mathbf{x}_k\}$ can be guided toward a minimizer of f_m.

Here is the algorithm of the Stochastic gradient method (SG):

Stochastic Gradient (SG) Method

1. Given an initial iterate $\mathbf{x}_0$.
2. For k = 0,1,2,... do
   (a) choose randomly i_k in $\{1, \ldots, m\}$.
   (b) compute $\mathbf{p}_k = -\nabla f_{i_k}(\mathbf{x}_k)$
   (c) choose a stepsize $\alpha_k > 0$
   (d) set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$

The coefficients α_k are usually chosen heuristically, often a small constant value is fixed for which the method is convergent. This is a difficult choice that affects the speed of convergence of the method. Moreover, the performance of the stochastic gradient method worsens when the conditioning of the problem increases and this method is usually more suitable (and more studied) for convex problems.

The step computed by the standard gradient method is of course more expensive than that computed by SG, but one may expect that a better step is computed when all the samples are considered in an iteration, for more information is used. Stochastic and full approaches offer then different trade-offs in terms of per-iteration costs and expected per-iteration improvement in minimizing the empirical risk. Moreover, due to the sum structure of the objective function, the full method can easily benefit from parallelization.

There are however several reasons for which the stochastic gradient can be preferred to the full gradient method.

- In many large-scale applications the data does involve a good deal of (approximate) redundancy. This suggests that using all of the sample data in every optimization iteration is not necessary and therefore inefficient.

- Stochastic gradient method usually shows a fast convergence at the beginning of the procedure and then tends to stagnate. If high accuracy is not required (which is the case in many machine learning applications) stochastic gradient can be a good option. If on the contrary we were to consider a larger number of iterations, then one would see the gradient approach eventually overtake the method and yield a lower training error.

- If f_m is strongly convex the gradient method converges linearly:

$$f_m(\mathbf{x}_k) - f_m^* \leq O(\rho^k), \quad \rho \in (0,1),$$

  where f_m^* denotes the minimal value of f_m. The total number of iterations in which the training error can be above a given threshold $\epsilon > 0$ is proportional to $\log(1/\epsilon)$. This means that, with a per-iteration cost proportional to m (due to the need to compute the full gradient) the total work required to obtain accuracy ϵ for a full gradient method is proportional to $m\log(1/\epsilon)$. The rate of convergence of a basic stochastic method is slower than for a batch gradient method. If f_m is strictly convex, each i_k is drawn uniformly from $\{1,\ldots,m\}$, and $\alpha_k = O(1/k)$, then the stochastic gradient has a sublinear convergence in expectation:

$$\mathbb{E}(f_m(\mathbf{x}_k) - f_m^*) = O(1/k).$$

  However, it is crucial to note that neither the per-iteration cost nor the right-hand side depends on the sample set size m. This means that the total work required to obtain accuracy ϵ for SG is proportional to $1/\epsilon$. Admittedly, this can be larger than $m\log(1/\epsilon)$ for moderate values of m and ϵ, but the comparison favours SG when one moves to the big data regime where m is large.

Between full gradient method and stochastic gradient method we find the mini-batch methods. Using all the samples to compute the gradient is too expensive, so we use just some of them and we change the chosen indexes at each iteration. In stochastic methods we take the minimum possible, just one sample, in mini-batch approaches we take a small subset $\mathcal{I}_k \subset \{1,\ldots,m\}$ of the samples at each iteration:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{\alpha_k}{|\mathcal{I}_k|}\sum_{i\in\mathcal{I}_k}\nabla f_i(\mathbf{x}_k),$$

This allows some degree of parallelization to be exploited in the computation of mini-batch gradients. In addition, one often finds that, due to the reduced variance of the stochastic gradient estimates, the method is easier to tune in terms of choosing the stepsizes.

7.2 Noise reduction methods

These methods aim to improve the rate of convergence from sublinear to linear by dynamically replacing or incorporating new gradient information in order to construct a more reliable step with smaller variance than an SG step. These are particularly beneficial when the gradient estimates are noisy. This prevents the SG from converging to the solution when fixed stepsizes are used and leads to a slow, sublinear rate of convergence when a diminishing stepsize sequence is employed. Noise reduction methods possess a linear rate of convergence to the optimal value using a fixed stepsize. It is possible to establish a result that stipulates that for strongly convex objectives if the variance of the stochastic vectors $\nabla f_{i_k}(\mathbf{x}_k)$ decreases geometrically, i.e. it exist $M > 0$ and $\xi \in (0,1)$ such that

$$\text{var} = \mathbb{E}[(\|\nabla f_{i_k}(\mathbf{x}_k)\| - \mathbb{E}(\|\nabla f_{i_k}(\mathbf{x}_k)\|))^2] \leq M\xi^k,$$

an SG-type method can converge at a linear rate.

Methods that belong to this family are dynamic sample size and gradient aggregation methods (such as SVRG, SAGA).

7.2.1 Dynamic sampling methods

These methods achieve noise reduction by gradually increasing the mini-batch size used in the gradient computation, thus employing increasingly more accurate gradient estimates as the optimization process proceeds:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{\alpha}{|\mathcal{I}_k|} \sum_{i \in \mathcal{I}_k} \nabla f_i(\mathbf{x}_k),$$

where $|\mathcal{I}_k| = n_k = \lceil \tau^{k-1} \rceil$ for some $\tau > 1$. For this method we can prove that it exists $M > 0$ such that

$$\text{var} \leq \frac{M}{n_k}.$$

In practice one needs to find a good value of the parameter τ that guarantees good performance for the application at hand. In addition, one may want to avoid having too large values, to prevent the full sample set from being employed too soon (or at all). Then sometimes an adaptive sampling algorithm is rather used, which considers mechanisms for choosing the sample sizes not according to a prescribed sequence, but adaptively according to information gathered during the optimization process. To do so, usually the following condition is tested: $\text{var} \leq C\|\nabla f_{\mathcal{I}_k}(\mathbf{x}_k)\|^2$ for a positive constant $C > 0$. If this is not satisfied the size of the set is increased.

7.2.2 Gradient aggregation methods

The dynamic sample size methods described in the previous subsection require a careful balance in order to achieve the desired linear rate of convergence without jeopardizing per-iteration costs. Alternatively, one can attempt to achieve an improved rate by asking a different question: rather than computing new stochastic gradient information (increasingly accurate) in each iteration, is it possible to achieve a lower variance by reusing and/or revising previously computed information? After all, if the current iterate has not been displaced too far from previous iterates, then stochastic gradient information from previous iterates may still be useful. In addition, if one maintains indexed gradient estimates in storage, then one can revise specific estimates as new information is collected. These methods are able to achieve a linear rate of convergence on strongly convex problems. This improved rate is achieved primarily by either an increase in computation or an increase in storage.

SVRG The first method we consider operates in cycles. At the beginning of each cycle, an iterate $\mathbf{x}_k$ is available at which the algorithm computes a full (or batch) gradient

$$\nabla f_m(\mathbf{x}_k) = \frac{1}{m} \sum_{i=1}^{m} \nabla f_i(\mathbf{x}_k).$$

Then, after initializing $\tilde{\mathbf{x}}_1 = \mathbf{x}_k$, a set of N inner iterations indexed by j with an update $\tilde{\mathbf{x}}_{j+1} = \tilde{\mathbf{x}}_j - \alpha \tilde{\mathbf{g}}_j$ are performed, where

$$\begin{aligned}
\tilde{\mathbf{g}}_j &= \nabla f_{i_j}(\tilde{\mathbf{x}}_j) - (\nabla f_{i_j}(\mathbf{x}_k) - \nabla f_m(\mathbf{x}_k)) \\
&= \begin{pmatrix} \dfrac{\partial f_{i_j}(\tilde{\mathbf{x}}_j)}{\partial x_1} \\ \vdots \\ \dfrac{\partial f_{i_j}(\tilde{\mathbf{x}}_j)}{\partial x_n} \end{pmatrix} - \begin{pmatrix} \dfrac{\partial f_{i_j}(\mathbf{x}_k)}{\partial x_1} \\ \vdots \\ \dfrac{\partial f_{i_j}(\mathbf{x}_k)}{\partial x_n} \end{pmatrix} + \frac{1}{m} \begin{pmatrix} \dfrac{\partial f_1(\mathbf{x}_k)}{\partial x_1} + \cdots + \dfrac{\partial f_m(\mathbf{x}_k)}{\partial x_1} \\ \vdots \\ \dfrac{\partial f_1(\mathbf{x}_k)}{\partial x_n} + \cdots + \dfrac{\partial f_m(\mathbf{x}_k)}{\partial x_n} \end{pmatrix}
\end{aligned}$$

and $i_j \in \{1, \ldots, m\}$ is chosen at random.

This formula can be seen as the result of the application of a variance reduction technique to the stochastic gradient. Variance reduction is a statistical technique that is used to reduce the variance of a random variable X by using another random variable Y, which is positively correlated with X. A new variable Z_α is defined as

$$Z_\alpha = \alpha(X - Y) + \mathbb{E}(Y),$$

which has reduced variance.

Since the expected value of $\nabla f_{i_j}(\mathbf{x}_k)$ over all possible i_j is equal to $\nabla f_m(\mathbf{x}_k)$, one can view $\tilde{\mathbf{g}}_j$ as the result of the application of this technique to the stochastic gradient $\nabla f_{i_j}(\tilde{\mathbf{x}}_j)$, choosing $\alpha = 1$.

In every iteration, the algorithm randomly draws a stochastic gradient $\nabla f_{i_j}(\tilde{\mathbf{x}}_j)$ evaluated at the current inner iterate $\tilde{\mathbf{x}}_j$ and corrects it based on a perceived bias. Overall, $\tilde{\mathbf{g}}_j$ represents an unbiased estimator of $\nabla f_m(\tilde{\mathbf{x}}_j)$, but with a smaller variance than the one of $\nabla f_{i_j}(\tilde{\mathbf{x}}_j)$ (that is used in simple SG). This is the reason that the method is referred to as the stochastic variance reduced gradient (SVRG) method.

SVRG method

1. Given an initial iterate $\mathbf{x}_0$, stepsize $\alpha > 0$ and positive integer N.
2. For k = 0,1,2,... do
   (a) Compute the batch gradient $\nabla f_m(\mathbf{x}_k)$
   (b) Initialize $\tilde{\mathbf{x}}_1 = \mathbf{x}_k$
   (c) for $j = 1, \ldots, N$ do
      i. Chose i_j uniformly from $\{1, \ldots, m\}$.
      ii. Set $\tilde{\mathbf{g}}_j = \nabla f_{i_j}(\tilde{\mathbf{x}}_j) - (\nabla f_{i_j}(\mathbf{x}_k) - \nabla f_m(\mathbf{x}_k))$
      iii. Set $\tilde{\mathbf{x}}_{j+1} = \tilde{\mathbf{x}}_j - \alpha \tilde{\mathbf{g}}_j$.
   (d) Update $\mathbf{x}_{k+1}$
      i. Option 1: Set $\mathbf{x}_{k+1} = \tilde{\mathbf{x}}_{N+1}$.
      ii. Option 2: Set $\mathbf{x}_{k+1} = \frac{1}{N} \sum_{j=1}^{N} \tilde{\mathbf{x}}_{j+1}$
      iii. Option 3: Choose j uniformly from $\{1, ..., N\}$ and set $\mathbf{x}_{k+1} = \tilde{\mathbf{x}}_{j+1}$.

SAGA The second method we consider is also based on the idea of variance reduction as SVRG, but differs from it as it does not operate in cycles, and computes batch gradients only at the initial point. At the beginning of the optimization process the gradient of all the m components of the objective function are computed and stored in a $n \times m$ table. This table is updated along the optimization process: at each iteration k a random index i_k is chosen and the corresponding gradient $\nabla f_{i_k}(\mathbf{x}_k)$ is computed at the current solution approximation. This is used to update the table: the column i_k is updated with this new value.

The table is used to compute the new step. In iteration k, the integer $i_k \in 1, ..., m$ is chosen at random and the step direction is set by

$$\mathbf{g}_k = \nabla f_{i_k}(\mathbf{x}_k) - \nabla f_{i_k}(\mathbf{x}_{[i_k]}) + \frac{1}{m} \sum_{i=1}^{m} \nabla f_i(\mathbf{x}_{[i]})$$

where $\mathbf{x}_{[i]}$ represents the latest iterate at which ∇f_i was evaluated. This vectors are stored in the table.

Taking the expectation of $\mathbf{g}_k$ with respect to all choices of $j \in \{1, ..., m\}$, one again has that $\mathbb{E}[\mathbf{g}_k] = \nabla f_m(\mathbf{x}_k)$. Thus, the method employs unbiased gradient estimates, but with variances that are expected to be less than the stochastic gradients that would be employed in a basic SG routine. The algorithm of SAGA is given in the following. Notice that, as opposed to SVRG, SAGA needs to store all the gradients.

SAGA method

1. Given an initial iterate $\mathbf{x}_0$, stepsize $\alpha > 0$.
2. For $i = 1, \ldots, m$ do
   (a) Compute $\nabla f_i(\mathbf{x}_1)$.
   (b) Store $\nabla f_i(\mathbf{x}_{[i]}) = \nabla f_i(\mathbf{x}_1)$ (create the table).
3. For $k = 1, 2, ...$ do
   (a) Choose j uniformly in $\{1, ..., m\}$.
   (b) Compute $\nabla f_j(\mathbf{x}_k)$.
   (c) Set $\mathbf{g}_k = \nabla f_j(\mathbf{x}_k) - \nabla f_j(\mathbf{x}_{[j]}) + \frac{1}{m} \sum_{i=1}^{m} \nabla f_i(\mathbf{x}_{[i]})$
   (d) Store $\nabla f_j(\mathbf{x}_{[j]}) = \nabla f_j(\mathbf{x}_k)$ (update the table).
   (e) Set $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{g}_k$.

7.3 Accelerated gradient methods

The idea of accelerated gradient method is to decrease the number of iterations instead of the cost of each of them.

7.3.1 Gradient method with momentum

The problem with gradient descent is that the parameters update at an iteration k is governed by the learning rate and the gradient at that iteration only. It doesn't take into account the past steps taken. This leads to the following problems[1].

1. The gradient of the objective function at saddle points (in a plateau) is negligible or zero, which in turn leads to small or no parameters updates (very small steps). Hence, the optimization process may stop before reaching an optimal point
2. The path followed by gradient descent is very jittery, also when operating with mini-batch mode.

Consider the below surface.

Let's assume the starting guess corresponds to point A. With gradient descent, the objective function decreases rapidly along the slope AB as the gradient along this slope is high. But as soon as

[1] `https://towardsdatascience.com/gradient-descent-with-momentum-59420f626c8f`

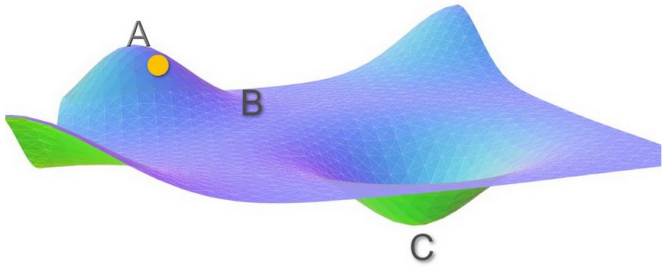


Figure 7.4: Surface corresponding to an objective function.

it reaches point B the gradient becomes very low. The step around B is very small. Even after many iterations, the sequence moves very slowly before getting stuck at a point in the plateau where the gradient eventually becomes zero.

In this case, ideally, the sequence should have moved to the global minima point C, but because the gradient disappears at point B, we are stuck with a sub-optimal solution.

How can momentum fix this? Now, imagine you have a ball rolling from point A. The ball starts rolling down slowly and gathers some momentum across the slope AB. When the ball reaches point B, it has accumulated enough momentum to push itself across the plateau region B and finally following slope BC to land at the global minima C.

How can this be used and applied to Gradient descent? To account for the momentum, we can keep memory of past gradients. In regions where the gradient is high like AB, the step will be large. Thus, in a way we are gathering momentum by keeping memory of these gradients. Thus, we could define the step as the average of past gradients instead of the gradient at the current iteration. But there is a problem with this method, it considers all the gradients over iterations with equal weightage. The gradient at $k = 0$ has equal weightage as that of the gradient at current iteration k. We need to use some sort of weighted average of the past gradients such that the recent gradients are given more weightage.

This can be done by using an *Exponential Moving Average (EMA)*[2]. Exponentially weighed averages deal with sequences of numbers. Suppose, we have some sequence y which is noisy. See for example in Figure 7.5 (left) the cosine function to which some Gaussian noise has been added. What we want to do with this data is, instead of using it, we want some kind of "moving" average which would "denoise" the data and bring it closer to the original function. Exponentially weighed averages can do this. An exponential moving average is an average that assigns a greater weight on the most recent values.

The EMA $s(t)$ for a series $y(t)$ may be calculated recursively as:

$$s(t) = \begin{cases} y(1) & \text{if } t = 1, \\ \beta s(t-1) + (1-\beta) y(t) & \text{if } t > 1 \end{cases}$$

where the coefficient β represents the degree of weighting increase, a constant smoothing factor between 0 and 1. A lower β discounts older observations faster; $y(t)$ is the value at the instant t; $s(t)$ is the

[2] https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d

value of the EMA at t.

Figure 7.5 (right) reproduces the EMA for the cosine function.

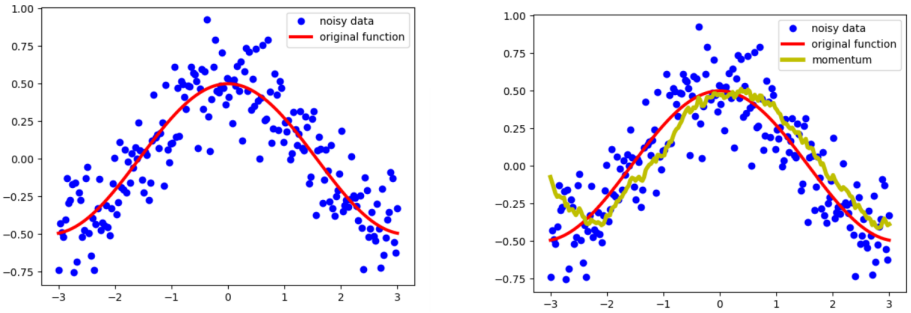


Figure 7.5: Left: data coming from noisy cosine function. Right: Exponential Moving Average.

In our case of a sequence of gradients, the new weight update equation at iteration k becomes

$$\mathbf{s}_k = \beta \mathbf{s}_{k-1} + (1-\beta)\mathbf{g}_k$$

where $\mathbf{s}_k$ is the new update done at iteration k, that is $\mathbf{x}_{k+1} - \mathbf{x}_k$, β is the momentum constant, and $\mathbf{g}_k$ is the gradient approximation at iteration k (it may be the full gradient, or a mini-batch or stochastic gradient).

Often the term $(1-\beta)$ is replaced with the learning rate, and a dynamically adjusted parameter β_k may be used instead of a fixed β, so the general update for gradient method with momentum is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) + \beta_k(\mathbf{x}_k - \mathbf{x}_{k-1}), \tag{7.2}$$

where also the scalar sequence $\{\alpha_k\}$ is either predetermined or set dynamically. The term $\beta_k(\mathbf{x}_k - \mathbf{x}_{k-1})$ is referred to as the *momentum term*, which, recursively, maintains the algorithm's movement along previous search directions.

Of course, when $\beta_k = 0$ for all $k \in \mathbb{N}$, it reduces to the steepest descent method. When $\alpha_k = \alpha$ and $\beta_k = \beta$ for some constants $\alpha > 0$ and $\beta > 0$ for all $k \in \mathbb{N}$, it is referred to as the *heavy ball method*.

An alternative view of the heavy ball method is obtained by expanding the update:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \sum_{j=1}^{k} \beta^{k-j} \nabla f(\mathbf{x}_k)$$

thus, each step can be viewed as an exponentially decaying average of past gradients.

Let's analyse the contribution of β. Assume $\alpha = 1$.

What if $\beta = 0.1$? At $k = 3$; the gradient $\mathbf{g}_3$ will contribute 100% of its value, the gradient $\mathbf{g}_2$ will contribute 10% of its value, and gradient $\mathbf{g}_1$ will only contribute 1% of its value, so contribution from earlier gradients decreases rapidly.

What if $\beta = 0.9$? At $k = 3$; the gradient $\mathbf{g}_3$ will contribute 100% of its value, $\mathbf{g}_2$ will contribute 90% of its value, and gradient $\mathbf{g}_1$ will contribute 81% of its value.

We can deduce that higher β will accommodate more gradients from the past. Hence, generally, β is kept around 0.9 in most of the cases. Note however that the actual contribution of each gradient in the step will be further subjected to the learning rate.

See Figure 7.6 for the case of the cosine function.

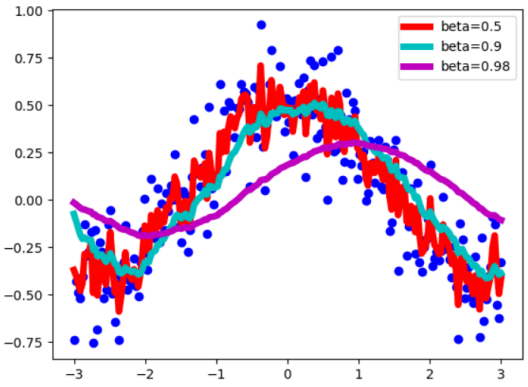


Figure 7.6: Values of β in the momentum term for the cosine function.

This addresses our first point where we said when the gradient at the current moment is negligible or zero the learning becomes zero. Using momentum with gradient descent, gradients from the past will push the sequence further to move around a saddle point.

In the cost surface shown earlier let's zoom into point C.

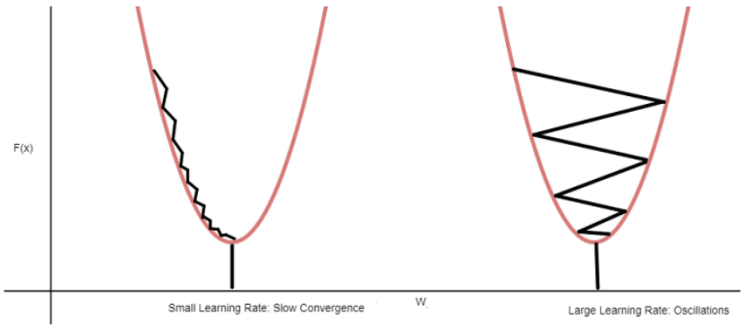


Figure 7.7: Gradient steps for large and small learning rate.

With gradient descent, if the learning rate is too small, the steps will also be small, hence convergence takes a lot of time even when the gradient is high. This is shown in the left side image below. If the learning rate is too high the sequence oscillates around the minima as shown in the right side image below.

How does momentum fix this? Let's look at the last summation equation of the momentum again.

1. *Case 1: When all the past gradients have the same sign*

   The summation term will become large and we will take large steps while updating the parameters. Along the curve BC, even if the learning rate is low, all the gradients along the curve will have the same direction (sign) thus increasing the momentum and accelerating the descent.

2. *Case 2: when some of the gradients have + sign whereas others have − sign*

The summation term will become small and the steps will be small. If the learning rate is high, the gradient at each iteration around the valley C will alter its sign between + and − and after few oscillations, the sum of past gradients will become small. Thus, making small steps from there on and damping the oscillations.

This to some amount addresses our second problem. Gradient descent with momentum takes small steps in directions where the gradients oscillate and take large steps along the direction where the past gradients have the same direction (same sign). See the effect of the momentum term on the 2D example in Figure 7.8.

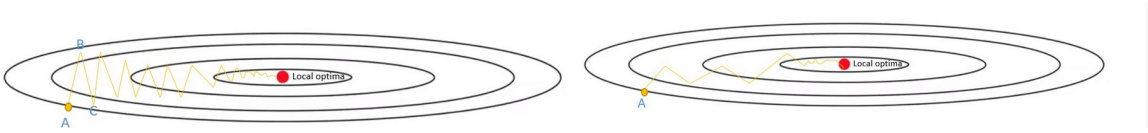


Figure 7.8: Gradient steps without (left) and with (right) momentum term.

The heavy ball method is known to yield a superior rate of convergence as compared to steepest descent with a fixed stepsize for certain functions of interest. For example, when f is a strictly convex quadratic with minimum and maximum eigenvalues given by $\lambda_{\min} > 0$ and $\lambda_{\max} \geq \lambda_{\min}$, respectively, steepest descent and the heavy ball method each yield a linear rate of convergence (in terms of the distance to the solution converging to zero) with contraction constants respectively given by

$$\frac{\kappa - 1}{\kappa + 1} \text{ and } \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \text{ with } \kappa = \frac{\lambda_{\max}}{\lambda_{\min}} \geq 1.$$

Choosing (α, β) to achieve these rates requires knowledge of $(\lambda_{\min}, \lambda_{\max})$, which might be unavailable. Still, even without this knowledge, the heavy ball method often outperforms steepest descent.

Even though momentum with gradient descent converges better and faster, it still doesn't resolve all the problems. First, the hyperparameter α (learning rate) still has to be tuned manually. Second, in some cases, where, even if the learning rate is low, the momentum term and the current gradient can alone drive and cause oscillations.

7.3.2 Nesterov accelerated gradient

This is an accelerated gradient method that was proposed by Nesterov. The core idea behind Nesterov momentum is that if the current parameter vector is $\mathbf{x}_k$, then the momentum term alone (i.e. ignoring the term with the gradient) is about to nudge the parameter vector by $\beta_k(\mathbf{x}_k - \mathbf{x}_{k-1})$. Therefore, if we are about to compute the gradient, we can treat the future approximate position $\mathbf{x}_k + \beta_k(\mathbf{x}_k - \mathbf{x}_{k-1})$ as a "lookahead" - this is a point in the vicinity of where we are soon going to end up. Hence, it makes sense to compute the gradient at $\mathbf{x}_k + \beta_k(\mathbf{x}_k - \mathbf{x}_{k-1})$ instead of at the old position $\mathbf{x}_k$, see Figure 7.9.

Written as a two-step procedure, the Nesterov update is then

$$\tilde{\mathbf{x}}_k = \mathbf{x}_k + \beta_k(\mathbf{x}_k - \mathbf{x}_{k-1}) \text{ and } \mathbf{x}_{k+1} = \tilde{\mathbf{x}}_k - \alpha_k \nabla f(\tilde{\mathbf{x}}_k),$$

which leads to the condensed form

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k + \beta_k(\mathbf{x}_k - \mathbf{x}_{k-1})) + \beta_k(\mathbf{x}_k - \mathbf{x}_{k-1}). \tag{7.3}$$

In this manner, it is easy to compare this approach with the momentum one. In particular, one can describe their difference as being a reversal in the order of computation. In (7.2), one can imagine

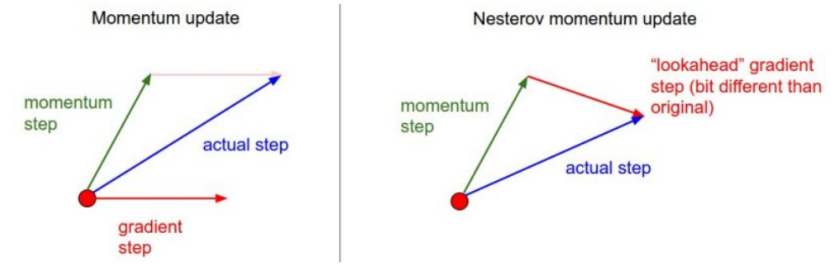


Figure 7.9: Momentum vs Nesterov update.

taking the steepest descent step and then applying the momentum term, whereas (7.3) results when one follows the momentum term first, then applies a steepest descent step (with the gradient evaluated at $\tilde{\mathbf{x}}_k$, not at $\mathbf{x}_k$).

While this difference may appear to be minor, it is well known that (7.3) with appropriately chosen $\alpha_k = \alpha > 0$ for all $k \in \mathbb{N}$ and $\{\beta_k\} \nearrow 1$ leads to an optimal iteration complexity when f is convex and continuously differentiable with a Lipschitz continuous gradient. Specifically, while in such cases a steepest descent method converges with a distance to the optimal value decaying with a rate $O(1/k)$, the iteration (7.3) converges with a rate $O(1/k^2)$, which is provably the best rate that can be achieved by a gradient method. Unfortunately, no intuitive explanation as to how Nesterov's method achieves this optimal rate has been widely accepted. Still, one cannot deny the analysis and the practical gains that the technique has offered.

7.3.3 Coordinate descent

Coordinate descent (CD) methods are among the oldest in the optimization literature. As their name suggests, they operate by taking steps along coordinate directions: one attempts to minimize the objective with respect to a single variable while all others are kept fixed, then other variables are updated similarly in an iterative manner. Such a simple idea is easy to implement. Their limitations have been documented and well understood for many years, but one can argue that their advantages were not fully recognized until recent work in machine learning and statistics demonstrated their ability to take advantage of certain structures commonly arising in practice. The iteration of CD method is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f_{i_k}(\mathbf{x}_k)\mathbf{e}_{i_k}, \text{ where } \nabla f_{i_k}(\mathbf{x}_k) := \frac{\partial f}{\partial x_{i_k}}(\mathbf{x}_k),$$

and x_{i_k} represents the i_k-th element of the parameter vector, and $\mathbf{e}_{i_k}$ represents the i_k-th coordinate vector for some $i_k \in \{1, \ldots, n\}$. In other words, the solution estimates $\mathbf{x}_{k+1}$ and $\mathbf{x}_k$ differ only in their i_k-th element.

Contrary to what intuition might suggest, a CD method is not guaranteed to converge when applied to minimize any given continuously differentiable function, counterexamples with nonconvex functions have been built. On the other hand, if the objective is strongly convex, the CD method will not fail and one can establish a linear rate of convergence.

7.4 Second-order methods

Due to its ability to achieve a quadratic rate of convergence in the neighborhood of a strong local minimizer, Newton's method represents an ideal in terms of optimization algorithms. It does not scale well with the dimension n of the optimization problem, but there are variants that can scale well while also being able to deal with nonconvexity and that can be preferable to first order methods. Gradient-based methods may exhibit a slow convergence rate, it may be difficult to properly tune the learning rate and they are generally more efficient in the solution of convex problems.

When minimizing our twice-continuously differentiable function f_m, a Newton iteration is

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \quad H_m(\mathbf{x}_k)\mathbf{p}_k = -\nabla f_m(\mathbf{x}_k)$$

This iteration demands much in terms of computation and storage when n is large. This is the case for example in the training of deep neural networks. To approximate highly nonlinear functions, a network with a large number of weights may be necessary to obtain a sufficient accuracy. However, some techniques exist to make second order methods less expensive.

7.4.1 Hessian-Free Inexact Newton Methods

Rather than solve the Newton system exactly through matrix factorization techniques, one can instead only solve it inexactly through an iterative approach such as the conjugate gradient (CG) method (see chapter 5 of the book). By ensuring that the linear solves are accurate enough, such an inexact Newton-CG method can enjoy a superlinear rate of convergence.[3] In fact, the computational benefits of inexact Newton-CG go beyond its ability to maintain classical convergence rate guarantees. Like many iterative linear system techniques, CG applied to the Newton's system does not require access to the Hessian itself, only Hessian-vector products. It is in this sense that such a method may be called Hessian-free. This is ideal when such products can be coded directly without having to form an explicit Hessian, as Example 7.4.1 below demonstrates. Each product is at least as expensive as a gradient evaluation, but as long as the number of products (one per CG iteration) is not too large, the improved rate of convergence can compensate for the extra per-iteration work required over a simple full gradient method.

Example 7.4.1. *Consider the function $f(\mathbf{x}) = e^{x_1 x_2}$. Let us define, for any $d \in \mathbb{R}^2$ the function $\Phi(\mathbf{x}; \mathbf{d}) = \nabla f(\mathbf{x})^T \mathbf{d} = x_2 e^{x_1 x_2} d_1 + x_1 e^{x_1 x_2} d_2$. Computing the gradient of Φ with respect to $\mathbf{x}$, we have*

$$\nabla \Phi_{\mathbf{x}}(\mathbf{x}, \mathbf{d}) = H_f(\mathbf{x})\mathbf{d} = \begin{bmatrix} x_2^2 e^{x_1 x_2} d_1 + (e^{x_1 x_2} + x_1 x_2 e^{x_1 x_2}) d_2 \\ (e^{x_1 x_2} + x_1 x_2 e^{x_1 x_2}) d_1 + x_1^2 e^{x_1 x_2} d_2 \end{bmatrix}$$

with $H_f(\mathbf{x})$ the Hessian matrix of f.

We have thus obtained, for any $d \in \mathbb{R}^2$ a formula for computing $H_f(\mathbf{x})\mathbf{d}$ that does not require $H_f(\mathbf{x})$ explicitly. Note that by storing the scalars x_1, x_2 and $e^{x_1 x_2}$ from the evaluation of f, the additional costs of computing the gradient-vector and Hessian-vector products are small. The idea illustrated in this example can be applied in general. For a smooth objective function f, one can compute $H_f(\mathbf{x})\mathbf{d}$ at a cost that is a small multiple of the cost of evaluating $\nabla f(\mathbf{x})$, and without forming the Hessian, which would require $O(n^2)$ storage.

In machine learning applications Hessian-vector products can be computed in this manner, and an inexact Newton-CG method can be applied. A concern is that, in certain cases, the cost of the

[3] R. S. Dembo, Eisenstat S. C., and T. Steihaug. Inexact Newton Methods. SIAM Journal on Numerical Analysis, 19(2):400-408, 1982.

CG iterations may render such a method uncompetitive with alternative approaches. Interestingly, however, the structure of the objective function can be exploited so that the resulting method has lighter computational overheads, as described next.

7.4.2 Subsampled Hessian-Free Newton Methods

The motivation for the method we now describe stems from the observation that, in inexact Newton methods, the Hessian matrix need not be as accurate as the gradient to yield an effective iteration. Translated to the context of large-scale machine learning applications, this means that the iteration is more tolerant to noise in the Hessian estimate than it is to noise in the gradient estimate. Based on this idea, the technique we state here employs a smaller sample for defining the Hessian than for the stochastic gradient estimate.

Given $\mathcal{I}_k \subset \{1, \ldots, m\}$ and $\mathcal{I}_k^H \subset \{1, \ldots, m\}$, the stochastic gradient estimate is

$$\nabla f_{\mathcal{I}_k}(\mathbf{x}_k) = \frac{1}{|\mathcal{I}_k|} \sum_{i \in \mathcal{I}_k} \nabla f_i(\mathbf{x}_k),$$

and the stochastic Hessian estimate is

$$H_{\mathcal{I}_k}(\mathbf{x}_k) = \frac{1}{|\mathcal{I}_k^H|} \sum_{i \in \mathcal{I}_k^H} H_{f_i}(\mathbf{x}_k),$$

where $\mathcal{I}_k^H$ is uncorrelated with $\mathcal{I}_k$ and $|\mathcal{I}_k^H| < |\mathcal{I}_k|$. If one chooses the subsample size $|\mathcal{I}_k|$ small enough, then the cost of each product involving the Hessian approximation can be reduced significantly, thus reducing the cost of each CG iteration. On the other hand, one should choose $|\mathcal{I}_k|$ large enough so that the curvature information captured through the Hessian-vector products is productive. If done appropriately, Hessian subsampling is robust and effective. An inexact Newton method that incorporates this techniques is outlined in Algorithm 6.1. The algorithm is stated with a backtracking line search, though other stepsize-selection techniques could be considered as well.

Subsampled Hessian-Free Inexact Newton Method

1. Given an initial iterate $\mathbf{x}_0$, $\rho \in (0, 1)$, and $\max_{CG} \in \mathbb{N}$.
2. For $k = 0, 1, 2, \ldots$ do
   (a) Choose randomly $\mathcal{I}_k$ and $\mathcal{I}_k^H$.
   (b) Compute $\mathbf{p}_k$ applying Hessian-free CG to solve $H_{\mathcal{I}_k}(\mathbf{x}_k)\mathbf{p}_k = -\nabla f_{\mathcal{I}_k}(\mathbf{x}_k)$ until $\max_{CG}$ iterations have been performed or a trial solution yields
   $$\|\mathbf{r}_k\| := \|H_{\mathcal{I}_k}(\mathbf{x}_k)\mathbf{p}_k + \nabla f_{\mathcal{I}_k}(\mathbf{x}_k)\| \leq \rho \|\nabla f_{\mathcal{I}_k}(\mathbf{x}_k)\|$$
   (c) Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$, where α_k satisfies (A)+(W)

The rate of convergence of the method can be controlled through the accuracy with which the systems are solved. Defining $\mathbf{r}_k := H_{\mathcal{I}_k}(\mathbf{x}_k)\mathbf{p}_k + \nabla f_{\mathcal{I}_k}(\mathbf{x}_k)$ for all $k \in \mathbb{N}$, the iteration can enjoy a linear, superlinear, or quadratic rate of convergence by controlling $\|\mathbf{r}_k\|$, where for the superlinear rates one must have

$$\frac{\|\mathbf{r}_k\|}{\|\nabla f(\mathbf{x}_k)\|} \to 0.$$

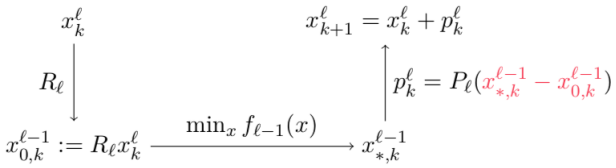


Figure 7.10: Basic idea of multilevel method.

To quantify the step computation cost in an inexact Newton-CG framework, let g_{cost} be the cost of computing a gradient estimate $\nabla f_{\mathcal{I}_k}(\mathbf{x}_k)$ and factor $\times$ g_{cost} denote the cost of one Hessian-vector product. If the maximum number of CG iterations, $\max_{CG}$, is performed for every outer iteration, then the step computation cost is

$$\max_{CG} \times \text{factor} \times g_{cost} + g_{cost}.$$

In a deterministic inexact Newton-CG method for minimizing the empirical risk f_m, i.e., when $|\mathcal{I}_k^H| = |\mathcal{I}_k| = n$ for all $k \in \mathbb{N}$, the factor is at least 1 and $\max_{CG}$ would typically be chosen as 5, 20, or more, leading to an iteration that is many times the cost of an SG iteration. However, in a stochastic framework using Hessian sub-sampling, the factor can be chosen to be sufficiently small such that $\max_{CG} \times \text{factor} \sim 1$, leading to a per-iteration cost proportional to that of SG.

7.4.3 Multilevel second order training methods

A different approach suitable when n is large are multilevel methods, which are inspired to classical multigrid methods, well-known for their efficiency in solving large scale linear systems arising from the discretization of linear or nonlinear elliptic partial differential equations (cf. W. Briggs, V. Henson, and S. McCormick, A Multigrid Tutorial).

The main idea on which multilevel methods are based is the exploitation of a hierarchy of problems, approximating the original one. The use of more levels is beneficial, as the coarse problems will generally be cheaper to solve than the original one.

We describe here a multilevel Levenberg-Marquardt training method[4].

At each iteration of the standard method, the objective function is approximated by the regularized Taylor model. The minimization of the model represents the major cost per iteration of the methods, which crucially depends on the dimension n of the problem. We want to reduce this cost by exploiting the knowledge of alternative simplified expressions of the objective function. More specifically, we assume that we know a collection of functions $\{f^\ell\}_{i=0}^{\ell_{\max}}$ such that each f^l is a twice-continuously differentiable function from $\mathbb{R}^{n_\ell} \to \mathbb{R}$ and $f^{\ell_{\max}}(x) = f(x)$ for all $x \in \mathbb{R}^n$. We will also assume that, for each $i = 1, \ldots, \ell_{\max}$, f^ℓ is more costly to minimize than $f^{\ell-1}$, for example because it has a larger number of variables.

The basic idea of the method is illustrated in Figure 7.10.

We assume to have at disposal two operators (a restriction R and a prolongation P, $R : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_{\ell-1}}$ and $P : \mathbb{R}^{n_{\ell-1}} \to \mathbb{R}^{n_\ell}$) that allows us to move vectors from a level to another. (See for example in Figure 7.11 how the operators R and P are defined when the coarse problems arise from different levels of discretization of an infinite dimensional problems, such as a PDE for example).

At a generic level ℓ, classical LM method minimizes the Taylor model to go from iterate $\mathbf{x}_k^\ell$ to $\mathbf{x}_{k+1}^\ell$ finding the step $\mathbf{p}_k^\ell$. The multilevel method instead uses the projection operator P to project

[4] `https://perso.ens-lyon.fr/elisa.riccietti/doc/pde.pdf`,`https://perso.ens-lyon.fr/elisa.riccietti/doc/multilevel.pdf`

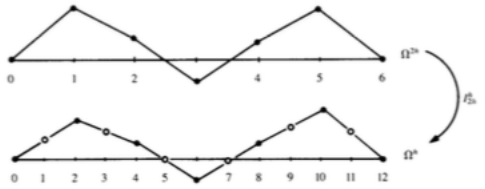


Figure 3.2: *Interpolation of a vector on coarse grid Ω^{2h} to fine grid Ω^h.*

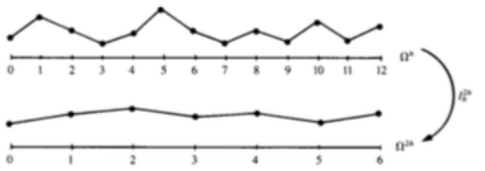


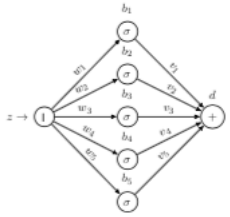
Figure 3.4: *Restriction by full weighting of a fine-grid vector to the coarse grid.*

Figure 7.11: Interpolation and weighting operators.

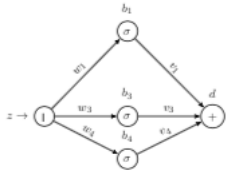
the current iterate to the lower level $\ell - 1$ and exploits the function approximation $f^{\ell-1}$ to find a coarse step (the vector in red in the picture). The coarse step is prolongated back to fine level, to get an approximation to $\mathbf{p}_k^\ell$ that is used to update the iterate.

The advantage of the procedure is that function $f^{\ell-1}$ has less variables and is therefore cheaper to optimize. Moreover, the procedure is recursive, so even at level $\ell - 1$ we can choose to use an even cheaper approximation, if it is available.

The method can be employed for the training of artificial neural networks, if a hierarchy of networks with less and less neurons is built, as in Figure 7.12, and if suitable operators P, R are defined.



$$R_1 \Downarrow P_1 \Uparrow$$



$$R_2 \Downarrow P_2 \Uparrow$$

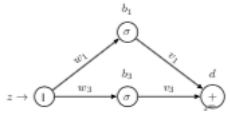


Figure 7.12: Hierarchy of neural networks.

Appendices

Appendix A

Stages opportunities

Give a look at `https://perso.ens-lyon.fr/elisa.riccietti/teaching.php` for the internships subjects.

- **Solution of ill-posed inverse problems in partial differential equations by neural networks** In collaboration with Stefania Bellavia, University of Florence, Italy, `https://www.unifi.it/p-doc2-2015-0-A-2b333b313b27-1.html`.
- **Multilevel derivative free optimization** In collaboration with Margherita Porcelli, University of Bologna, Italy, `https://sites.google.com/view/margherita-porcelli/`.

Appendix B

Elements of linear algebra

A good reference book for this part is "Matrix Computations" by Golub and Van Loan.

B.1 Matrix norm (I)

Suppose a vector norm $\|\cdot\|$ on $\mathbb{R}^m$ is given. Any $m \times n$ matrix A induces a linear operator from $\mathbb{R}^n$ to $\mathbb{R}^m$ with respect to the standard basis, and one defines the corresponding induced norm or operator norm on the space $\mathbb{R}^{m\times n}$ of all $m \times n$ matrices as follows:

$$\begin{aligned}\|A\| &= \sup\{\|Ax\| : x \in \mathbb{R}^n \text{ with } \|x\| = 1\} \\ &= \sup\left\{\frac{\|Ax\|}{\|x\|} : x \in \mathbb{R}^n \text{ with } x \neq 0\right\}.\end{aligned}$$

In particular, if the p-norm for vectors ($1 \leq p \leq \infty$) is used for both spaces $\mathbb{R}^n$ and $\mathbb{R}^m$, then the corresponding induced operator norm is:

$$\|A\|_p = \sup_{x\neq 0} \frac{\|Ax\|_p}{\|x\|_p}.$$

In the notes, when not otherwise specified, we use the Euclidean norm, i.e., $p = 2$. We give more details on matrix norms in Section B.5.5.

B.2 The condition number (I)

The condition number of an invertible matrix A is defined as:

$$\kappa(A) := \|A\|\|A^{-1}\|.$$

The condition number depends on the chosen norm, and generally the values do not coincide for the same matrix but different norms. Notice that

$$\kappa(A) := \|A\|\|A^{-1}\| \geq \|AA^{-1}\| = 1.$$

The condition number gives for example a measure of how the accuracy of the solution x of a linear system $Ax = b$ will be affected by the presence of errors in the right hand side b. Roughly, if b

is perturbed and the condition number is large, even a small error in b may cause a large error in x. On the other hand, if the condition number is small, then the error in x will not be much bigger than the error in b.

More precisely, assume that we have an error on b, and consider the solution $\hat{x}$ of the perturbed system $Ax = \hat{b}$. We want to know how the relative error $\|\hat{b} - b\|/\|b\|$ influences the relative error $\|\hat{x} - x\|/\|x\|$, that is we want to study the so-called *error propagation*. We have $A(\hat{x} - x) = \hat{b} - b$ and therefore

$$\|\hat{x} - x\| = \|A^{-1}(\hat{b} - b)\| \leq \|A^{-1}\|\|\hat{b} - b\|.$$

On the other hand we have

$$\|b\| = \|Ax\| \leq \|A\|\|x\|.$$

Combining the two we obtain

$$\frac{\|\hat{x} - x\|}{\|x\|} \leq \|A\|\|A^{-1}\|\frac{\|\hat{b} - b\|}{\|b\|} = \kappa(A)\frac{\|\hat{b} - b\|}{\|b\|}.$$

As the condition number of A is always larger than one, this determines how much the relative error of the right hand side vector can be amplified.

We give other details on the condition number in Section B.5.6.

B.3 Eigenvalues and eigenvectors of a matrix

Given a matrix $A \in \mathbb{R}^{n\times n}$, a vector $v \in \mathbb{R}^n$, $v \neq 0$, is an eigenvector of A if it exists $\lambda \in \mathbb{R}$, called an eigenvalue of A, such that

$$Av = \lambda v,$$

or equivalently $(A - \lambda I)v = 0$, meaning that $(A - \lambda I)$ must be a singular matrix. So $p(\lambda) := det(A - \lambda I) = 0$, where p is called the *characteristic polynomial* and the eigenvalues of A are the roots of p.

If $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A the following properties hold:

- The trace of A, defined as the sum of its diagonal elements, is also the sum of all eigenvalues:

$$\mathrm{tr}(A) = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i = \lambda_1 + \lambda_2 + \cdots + \lambda_n.$$

- The determinant of A is the product of all its eigenvalues:

$$\det(A) = \prod_{i=1}^{n} \lambda_i = \lambda_1\lambda_2\cdots\lambda_n.$$

B.3.1 Eigenvalue decomposition

A square $n \times n$ matrix A is called *diagonalizable* or nondefective if there exists an $n \times n$ invertible matrix P, such that $P^{-1}AP$ is a diagonal matrix (i.e., if A is *similar* to a diagonal matrix).

An $n \times n$ matrix A is diagonalizable if it has n distinct eigenvalues, i.e., if its characteristic polynomial has n distinct roots. However, the converse may be false.

Real symmetric matrices are diagonalizable by orthogonal matrices, i.e., if A is symmetric it always exists Q orthogonal such that $A = QDQ^T$.

If A is diagonalizable, we write the eigenvalue decomposition of A as: $A = PDP^{-1}$ where D is a diagonal matrix with the eigenvalues of A on the diagonal.

B.4 Positive definite matrices

A symmetric matrix A with real entries is positive-definite if the real number $z^T Az > 0$ for every nonzero vector z. A symmetric matrix A with real entries is positive-semidefinite if the real number $z^T Az \geq 0$ for every vector z. Negative-definite and negative semi-definite matrices are defined analogously. A matrix that is not positive semi-definite and not negative semi-definite is sometimes called indefinite.

Positive-definite and positive-semidefinite matrices can be characterized in many ways. A matrix A is positive-definite if and only if it satisfies any of the following equivalent conditions.

- A is congruent with a diagonal matrix with positive real entries D, that is there exists an invertible matrix P such that $P^T AP = D$.
- A is symmetric and all its eigenvalues are real and positive.
- A is symmetric and all its leading principal minors are positive.
- There exists an invertible matrix B with such that $A = B^T B$.

A matrix A is positive-semidefinite if and only if it satisfies any of the following equivalent conditions.

- A is congruent with a diagonal matrix with nonnegative real entries D, that is there exists an invertible matrix P such that $P^T AP = D$.
- A is symmetric and all its eigenvalues are real and nonnegaive.
- A is symmetric and all its principal minors are nonnegative.
- There exists a matrix B with such that $A = B^T B$.

Let A be an $n \times n$ symmetric matrix. For each $k = 1, \ldots, n$ the k-th order principal minors D_k of A are the determinants of the $k \times k$ matrices obtained by deleting $n - k$ rows and the corresponding $n - k$ columns of A (with the same index number). There are $\binom{n}{k}$ principal minors of order k. The *leading* principal minor of A of order k is the minor of order k obtained by deleting the last $n - k$ rows and columns.

Example : Let

$$A = \begin{bmatrix} 1 & 4 & 6 \\ 4 & 2 & 1 \\ 6 & 1 & 6 \end{bmatrix}$$

The leading principal minors are: $D_1 = 1$ (coefficient $(1,1)$ of the matrix), $D_2 = \begin{vmatrix} 1 & 4 \\ 4 & 2 \end{vmatrix} = -14$ (removing the third row and column) and $D_3 = \begin{vmatrix} 1 & 4 & 6 \\ 4 & 2 & 1 \\ 6 & 1 & 6 \end{vmatrix} = -109$, thus the matrix is indefinite.

The other principal minors are: $D_1 = 2, 6$ (the other elements on the diagonal, obtained removing the first and third rows and columns and the first and second rows and columns respectively), $D_2 = \begin{vmatrix} 1 & 6 \\ 6 & 6 \end{vmatrix}$ (removing the second row and column) and $D_2 = \begin{vmatrix} 2 & 1 \\ 1 & 6 \end{vmatrix}$ (removing the first row and column).

B.5 Singular values decomposition

Let $A \in \mathbb{R}^{m \times n}$, $m \geq n$. The singular value decomposition of A is a factorization of the form $U\Sigma V^T$ where

- U is an $m \times m$ orthogonal matrix
- Σ is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal
- V is an $n \times n$ orthogonal matrix.

The diagonal entries $\sigma_i = \Sigma_{ii}$ of Σ are known as the singular values of A. The number of non-zero singular values is equal to the rank of A. The columns of U and the columns of V are called the left-singular vectors and right-singular vectors of A, respectively.

The SVD is not unique. It is always possible to choose the decomposition so that the singular values are in descending order.

When the matrix has not full-rank, we can construct the compact SVD: $A = U_r \Sigma_r V_r^T$ in which Σ_r is square diagonal of size $r \times r$ where $r \leq \min\{m, n\}$ is the rank of A, and has only the non-zero singular values. In this variant, U_r is an $m \times r$ matrix and V_r is an $n \times r$ matrix, such that $U_r^T U_r = V_r^T V_r = I_r$.

The cost of computing the SVD is $O(mn \min\{m, n\})$. For the compact SVD of rank r the cost is $O(mnr)$.

Mathematical applications of the SVD include computing the pseudoinverse, matrix approximation, determining the rank, range, and null space of a matrix, cf. sections below.

B.5.1 Relation to eigenvalue decomposition

The singular value decomposition is very general in the sense that it can be applied to any $m \times n$ matrix, whereas eigenvalue decomposition can only be applied to diagonalizable matrices. Nevertheless, the two decompositions are related.

Given an SVD of A the following two relations hold:

$$A^T A = V \Sigma^T U^T U \Sigma V^T = V(\Sigma^T \Sigma) V^T$$
$$A A^T = U \Sigma V^T V \Sigma^T U^T = U \Sigma \Sigma^T U^T$$

The right-hand sides of these relations describe the eigenvalue decompositions of the left-hand sides. Consequently:

- The columns of V (right-singular vectors) are eigenvectors of $A^T A$.
- The columns of U (left-singular vectors) are eigenvectors of $A A^T$.
- The non-zero elements of Σ (non-zero singular values) are the square roots of the non-zero eigenvalues of $A^T A$ or $A A^T$.

In the special case that A is a symmetric matrix ($A = A^T$), which by definition must be square, the spectral theorem says that it can be diagonalized using a basis of eigenvectors, so that it can be written as $A = UDU^T$ for a orthogonal matrix U and a diagonal matrix D with elements λ_i along the diagonal. When A is positive semi-definite, λ_i will be non-negative real numbers so that the decomposition $A = UDU^T$ is also a singular value decomposition.

Thus, except for positive semi-definite matrices, the eigenvalue decomposition and SVD of A, while related, differ: the eigenvalue decomposition is $A = UDU^{-1}$, where U is not necessarily orthogonal and D is not necessarily positive semi-definite, while the SVD is $A = U\Sigma V^T$, where Σ is diagonal and positive semi-definite, and U and V are unitary matrices that are not necessarily related except through the matrix A. While only non-defective square matrices have an eigenvalue decomposition, any $m \times n$ matrix has a SVD.

B.5.2 Information on a matrix

Computing the SVD of the matrix and counting the number of non-zero singular values we can determine the rank of the matrix. Moreover, the SVD provides an explicit representation of the range and null space of a matrix A. The right-singular vectors corresponding to vanishing singular values of A (that is the columns of V from r+1 to n) span the null space of A and the left-singular vectors corresponding to the non-zero singular values of A (that is the first r columns of U) span the range of A:

$$\texttt{span}\{V^{(r+1)}, \ldots, V^{(n)}\} = Ker(A), \tag{B.1}$$

$$\texttt{span}\{U^{(1)}, \ldots, U^{(r)}\} = Im(A), \tag{B.2}$$

where for a matrix A, $A^{(i)}$ denotes its i-th column.

B.5.3 Pseudoinverse

Given $A \in \mathbb{R}^{m\times n}$, $A^\dagger \in \mathbb{R}^{n\times m}$ is the pseudoinverse of A, which is defined as the unique matrix satisfying all of the following four criteria, known as the Moore-Penrose conditions:

1. $AA^\dagger A = A$
2. $A^\dagger AA^\dagger = A$
3. $(AA^\dagger)^T = AA^\dagger$
4. $(A^\dagger A)^T = A^\dagger A$.

When A has full rank (that is, the rank of A is $\min\{m, n\}$), then $A^\dagger$ can be expressed as a simple algebraic formula.

- When A has linearly independent columns (and thus matrix $A^T A$ is invertible), it holds
$$A^\dagger = (A^T A)^{-1} A^T.$$
This particular pseudoinverse constitutes a left inverse, since, in this case,
$$A^\dagger A = I.$$
- When A has linearly independent rows (matrix AA^T is invertible), $A^\dagger$ can be computed as
$$A^\dagger = A^T (AA^T)^{-1}.$$
This is a right inverse, as
$$AA^\dagger = I.$$

The pseudoinverse can be computed from the singular value decomposition: $A^\dagger = V\Sigma^\dagger U^T$, where $\Sigma^\dagger \in \mathbb{R}^{n\times m}$ is the pseudoinverse of Σ, which is formed by replacing every non-zero diagonal entry by its reciprocal and transposing the resulting matrix. The pseudoinverse is one way to solve linear least squares problems.

B.5.4 Low rank matrix approximation

Consider the problem of approximating a matrix A with another matrix $\tilde{A}$ which has a specific rank r. In the case that the approximation is based on minimizing the Frobenius norm of the difference between A and $\tilde{A}$ under the constraint that $\text{rank}(\tilde{A}) = r$, it turns out that the solution can be obtained from the SVD of A, namely

$$\tilde{A} = U\tilde{\Sigma}V^T,$$

where $\tilde{\Sigma}$ is the same matrix as Σ except that it contains only the r largest singular values (the other singular values are replaced by zero). This is known as the Eckart-Young theorem.

B.5.5 Matrix norm (II)

In the special case of $p = 2$ (the Euclidean norm), the induced matrix norm is the spectral norm. The spectral norm of a matrix A is $\sigma_{\max}(A)$, the largest singular value of A (i.e., the square root of the largest eigenvalue of the matrix A^TA):

$$\|A\|_2 = \sqrt{\lambda_{\max}(A^TA)} = \sigma_{\max}(A).$$

Indeed, for an orthogonal matrix Q and any vector x it holds

$$\|Qx\|^2 = x^TQ^TQx = x^TIx = \|x\|^2.$$

So for an orthogonal matrix Q and any matrix A, it holds

$$\|QA\| = \sup\{\|QAx\| : x \in \mathbb{R}^n \text{ with } \|x\| = 1\} = \sup\{\|Ax\| : x \in \mathbb{R}^n \text{ with } \|x\| = 1\} = \|A\|.$$

Then, considering the SVD decomposition of A and from the orthogonality of U, V we have:

$$\begin{aligned}\|A\| =& \|U\Sigma V^T\| = \|\Sigma V^T\| = \sup\{\|\Sigma V^Tx\| : x \in \mathbb{R}^n \text{ with } \|x\| = 1\} \\ =& \sup\{\|\Sigma y\| : y \in \mathbb{R}^n \text{ with } \|y\| = 1\} = \|\Sigma\|,\end{aligned}$$

where we have set $y = V^Tx$. Computing $\|\Sigma\|$ from the definition we can see that $\|\Sigma\| = \sigma_{\max}$. Again from the SVD of A we have that

$$\|A^TA\|_2 = \|AA^T\|_2 = \|A\|_2^2$$

since

$$\|A^TA\|_2 = \sigma_{\max}(A^TA) = \sigma_{\max}(A)^2 = \|A\|_2^2$$

and similarly

$$\|AA^T\|_2 = \|A\|_2^2.$$

There is another important inequality:

$$\|A\|_2 = \sigma_{\max}(A) \leq \|A\|_{\text{F}} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} = \sqrt{\sum_{k=1}^{\min(m,n)}\sigma_k^2(A)} = \sqrt{\text{trace}(A^TA)},$$

where $\|A\|_{\text{F}}$ is the Frobenius norm ($\sigma_1, \ldots, \sigma_n$ are the singular values of A). Given $W \in \mathbb{R}^{n\times n}$ SPD, we know that

$$W = O^T\Lambda O \qquad \text{for some } O \in \mathbb{R}^{n\times n} \text{ orthogonal, } \Lambda = \begin{pmatrix}\lambda_1 & & \\ & \ddots & \\ & & \lambda_n\end{pmatrix},\ \lambda_1, \ldots, \lambda_n > 0,$$

and so we can define

$$\sqrt{W} = O^T \sqrt{\Lambda} O, \qquad \sqrt{\Lambda} = \begin{pmatrix} \sqrt{\lambda_1} & & \\ & \ddots & \\ & & \sqrt{\lambda_n} \end{pmatrix},$$

and it holds

$$\sqrt{W}\sqrt{W} = O^T \sqrt{\Lambda} O O^T \sqrt{\Lambda} O = O^T \sqrt{\Lambda}\sqrt{\Lambda} O = O^T \Lambda O = W.$$

Given $W \in \mathbb{R}^{n\times n}$ SPD, the weighted Frobenius of A with weight W is defined as

$$\|A\|_W = \|\sqrt{W} A \sqrt{W}\|_F.$$

Both the 2-norm and the Frobenius norm are unitarily invariant: `https://nhigham.com/2021/02/02/what-is-a-unitarily-invariant-norm/`. In linear algebra, a complex square matrix U is unitary if its conjugate transpose U^* is also its inverse, that is, if

$$U^*U = UU^* = I,$$

where I is the identity matrix. The equivalent of the unitary matrix in real numbers is the orthogonal matrix:

$$U^T U = UU^T = I.$$

B.5.6 The condition number (II)

The singular values can be used to compute the condition number of a matrix:

$$\kappa(A) = \|A\|\|A^{-1}\| = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$$

where $\sigma_{\max}(A)$ and $\sigma_{\min}(A)$ are maximal and minimal singular values of A respectively.

We have defined the condition number for a square invertible matrix as $\kappa(A) := \|A\|\|A^{-1}\|$. Using the singular values, this definition can be generalized to non-square matrices as $\kappa(A) = \dfrac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$ where $\sigma_{\min}$ is the smallest nonzero singular value. This amounts to define$\kappa(A) := \|A\|\|A^{\dagger}\|$, where $A\dagger$ is the pseudoinverse of A, see section B.5.3.

If A is normal ($A^TA = AA^T$), then

$$\kappa(A) = \frac{|\lambda_{\max}(A)|}{|\lambda_{\min}(A)|}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are maximal and minimal (by moduli) eigenvalues of A respectively. If A is orthogonal, then $\kappa(A) = 1$.

B.6 Solution of linear systems

Let $A \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$, consider the numerical solution of the linear system $A\mathbf{x} = \mathbf{b}$. Assume that the solution of the system exists and is unique. We distinguish two type of methods for the solution: direct methods that are based on a factorization of the matrix and find the solution in a finite number of steps, and iterative methods.

B.6.1 Direct methods

Direct methods are based on a factorization that makes the solution of the linear system easier. Among such factorizations we mention:

- **LU factorization**: factorization of A into two factors, a lower triangular matrix L and an upper triangular matrix U: $A = LU$. In the lower triangular matrix all elements in the diagonal are ones. This factorization is issued from the Gauss elimination process. The advantage of this factorization is that it allows to reduce the solution of the linear system to the solution of two triangular systems:

$$L\mathbf{y} = \mathbf{b};$$
$$U\mathbf{x} = \mathbf{y}$$

  that can be done by forward and backward distribution. The cost of computing the LU factorization is $O\left(\frac{2}{3}n^3\right)$. If A is invertible, then it admits an LU factorization if and only if all its leading principal minors are nonzero. If A is a singular matrix of rank r, then it admits an LU factorization if the first r leading principal minors are nonzero, although the converse is not true. It turns out however that a proper permutation in rows (or columns) is sufficient for the existance of the LU factorization. LU factorization with partial pivoting (LUP) refers often to LU factorization with row permutations only: $PA = LU$ where P is a permutation matrix. Any square matrix A admits such a factorization.

- **Cholesky factorization**: if A is SPD we can factorize it as $A = LL^T$, with L lower triangular, the cost of building this factorization is $O\left(\frac{n^3}{3}\right)$. The Cholesky decomposition always exists and is unique, its construction is is more efficient and numerically more stable than computing the LU decomposition.

B.6.2 Iterative methods

The main disadvantage of the direct methods is their cost: when n is large it may become prohibitive, also considering that the matrix needs to be stored in memory. The advantage of iterative methods is that they don't need to have access to the full matrix, but just on the action of the matrix on a vector. If the matrix-vector products involving the matrix A can be expressed analytically, cf. Example 7.4.1., then the iterative methods can approximate the solution of the linear system with no need of storing A. Moreover in many applications (for instance the step computation in Newton's method) an approximate solution is sufficient (that is solution for which $\|\mathbf{x}-\mathbf{x}^*\|/\|\mathbf{x}^*\| = tol$ with tol not necessarily very small). As iterative methods can be stopped after any number of iterations, they are particularly suitable to provide approximate solutions at a reasonable cost.

We mention as an example the Conjugate Gradient (CG) method, that can be used when A is SPD. This method updates the solution approximation as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k(\mathbf{b} - A\mathbf{x}_k),$$

with α_k appropriately chosen (for more details cf. `https://en.wikipedia.org/wiki/Conjugate_gradient_method`). Both the step-length and the direction require a matrix-vector product involving A: if this is coded as a function, the algorithm does not need A as an input. The convergence rate of the method depends on the condition number of the matrix A (cf. section B.5.6 below):

$$\|\mathbf{x}_k - \mathbf{x}^*\|_A \leq 2\left(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right)^k \|\mathbf{x}_0 - \mathbf{x}^*\|_A$$

having defined $\forall \mathbf{y} \in \mathbb{R}^n$, $\forall A \in \mathbb{R}^{n \times n}$ SPD the energy norm $\|\mathbf{y}\|_A^2 = \mathbf{y}^T A \mathbf{y}$. Then the condition number of the matrix has a strong influence on the convergence of the method: the closer $\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}$ is to 0, the faster the method converges. So if $\kappa(A)$ is to 1, i.e. if A is well-conditioned. On the contrary if A is ill-conditioned (i.e., $\kappa(A)$ is large) the factor will be close to one and the convergence very slow, i.e., it will take a lot of iterations to reach the solution.

B.6.3 Solving a linear system

Assume that A is square and invertible matrix, the SVD approximation can be used to compute the solution of a linear system $A\mathbf{x} = \mathbf{b}$, by considering $U\Sigma V^T \mathbf{x} = \mathbf{b}$, and by solving $\Sigma \mathbf{y} = U^T \mathbf{b}$ and setting $\mathbf{x} = V\mathbf{y}$.