HES-SO - University of Applied Sciences of western Switzerland - MSE

Optimization

Resume of the MSE lecture

by

Jérôme KEHRLI

prepared at HES-SO - Master - Provence,

written in Mai-Jul, 2011

largeley inspired from the work by TODO TODO

Resume of the Software Engineering lecture

Abstract:

TODO

Keywords: Optimization

Contents

Part I

Differential Optimisation

CHAPTER 1

Introduction

Contents

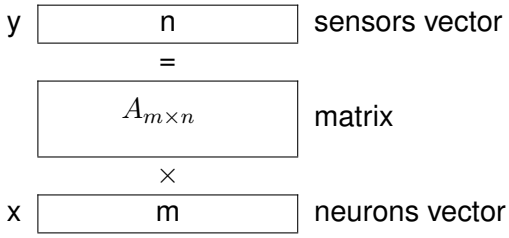
1.1 Purpose

1.2 Introductory example

In **digital medical imaging**, one of the current challenges computer sciences and mathematics are trying to solve is the building of a m ap of the neuronal activity within the brain. This implies devices laying 256 sensors on the head which measres the electromagnetic field generated by the 60'000 neurons.
We end up with a system of 256 equations with 60'000 unknown variables. The system is largely underdetermined.

One can pose:

$$m = 60'000\ neurons$$
$$n = 256\ sensors$$



1.2.1 Notation

- Vectors such as x or y above should generally be layed in column. However, in order to simplify the notation, we will write them down in row (thus considering the transposed vector ${x_1}^T$.
- An n dimension vector is normally noted $x = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}$, we will use the shorter form x = (x_i) considering $x \in \mathbb{R}^n$.
- Matrix $A \in \mathbb{M}_{m\times n}(\mathbb{R})$ = an n lines and m columns matrix with its coefficients in $\mathbb{R}$.

1.2.2 Formalizing the problem

y is known, A is known (given by medical experts) $\Rightarrow$ we need to find x.

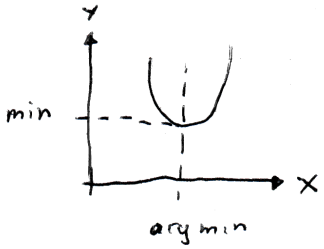
But we cannot simply compute ~~$x = A^{-1} \times y$~~ as A is not reversible (only squared matrices are reversible).

Besides, we have an additional dimension which is the *time* as we get a complete new set of measures every millisecond.
We are actually facing $L = number\ of\ measures$ systems.

1.2.2.1 Solution

We are looking for an x such that $\vec{z} = y_t - A \times x_t$ is as little as possible.
$\Rightarrow$ minimize the length of the vector z.
$\Rightarrow$ minimize $||z||_2 = \sqrt{\sum_{i=1}^{n} {z_i}^2}$

Note: $||z||_2$ is the *euclidean* norm.



We want to minimize $\min(||y_t - A \times x_t||_2)$ but we don't care in this actualy minimal value.
What we want is argmin:

> **argmin** = values of the x_i = coordinates where the minimum is realized, where it occurs.

$\Rightarrow$ We are looking for $\arg\min(||y_t - A \times x_t||_2)$

Notes: the matrix A gives us the way the signal is distorded and screwed by the tissues, the bones. etc. The great strength of the optimization techniques is that they enable us to ignore the noise.

1.2.2.2 Constraints

Yet we still have way too many solutions to our system $\Rightarrow$ we need to introduce constraints.

Contraint 1 : An active neuron should have each of its neighbours active either (perhaps less active, yet still active).

$$\min(\sum_{t=1}^{T}\sum_{i=1}^{m}||x_t^i - x_t^{\delta i}||_2)$$

where the neuron $x_t^{\delta i}$ is a neigbour of the neuron x_t^i

Contraint 2 : It takes time to activate a neuron or deactivate it. Thus, the state changes for each neuron should be minimized as well.

$$\min(\sum_{t=1}^{T}\sum_{i=1}^{m}||x_t^i - x_{t-1}^i||_2)$$

In addition, we should try to give each constraint a ponderation.

1.2.2.3 formula

The complete formula is:

$$\arg\min(||y_t - A \times x_t||_2 + \lambda^S \sum_{t=1}^{T}\sum_{i=1}^{m}||x_t^i - x_t^{\delta i}||_2 + \lambda^T \sum_{t=1}^{T}\sum_{i=1}^{m}||x_t^i - x_{t-1}^i||_2)$$

where:

λ^S is the spatial ponderation

λ^T is the temporal ponderation

One should note that because of the temporal variation, x and y are no longer vectors but matrices.
The dimension of the problem is quite big : 60000 neurons $\times$ 2000 measures = 10^8 dimensions!

1.3 Modeling

Modeling is a necessariy peliminary step to each optimisation process. How otherwise would one convert a concret problem into a mathematical formulation which enables its optimisation and resolution ?

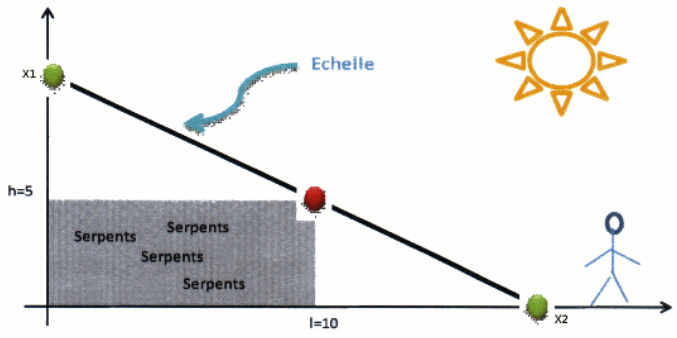
1.3.1 Introductory example : Indiana Jones

In his quest pursuing the *Coronado* cross, *Indiany Jones* is blocked in front of a big room filled with venimous snakes. This room is 10m long and 5m high. Indiany Jones is looking for a way to pass above the room.

He owns a ladder he can use for this purpose. He blocks one side of the ladder on the ground with a stone and lays the other side on the wall on the other side of the room.

The question is, where should he pose the ladder exactly in order to make its length as small as possible thus reducing the chances to break it?

The problem looks this way:



1.3.1.1 Decision variables

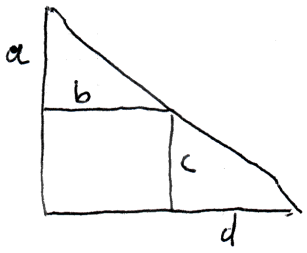
- x_1 position of the first side of the ladder on the ground
- x_2 height of the other side of the ladder on the wall

1.3.1.2 Score function

The goal is to minimize the length of the ladder:

$$f(x) = \sqrt{{x_1}^2 + {x_2}^2}$$

Remainder: Pythagore, similar triangles



similar triangles:

$$\frac{a+c}{b+d} = \frac{c}{d} = \frac{a}{b}$$

In our case, this gives us :

$$\frac{x_1}{x_2} = \frac{h}{x_1 - l} = \frac{x_2 - h}{l}$$

We have some constraints as the borders of the ladder should be outside the room of the snakes:
$x_1 \geq l$
$x_2 \geq h$

1.3.1.3 Complete model

$$(O) \begin{cases} \min\limits_{x \in \mathbb{R}} \sqrt{x_1^2 + x_2^2} \\ sc \begin{cases} x_1 x_2 - h x_1 - l x_2 = 0 \\ x_1 \geq l \\ x_2 \geq h \end{cases} \end{cases}$$

1.3.1.4 Notation

The solution to this problem (O) is the minimal value of the score function, i.e. the length of the ladder.
But we are not interested in this value, rather we are looking for the corresponding *decision variables* values - x_1 and x_2 - which enables the system to reach this minimal value (the values that realize the minimal value).

Moreover if f is the *score function* and X the set of constraints, one can pose:

$$x^* = \underset{x \in X \subseteq \mathbb{R}^n}{\arg\min}\, f(x)$$

where x is the *decision variable realizing the minimum*.

1.4 Transformation

There is usually a lot more than one single way to modelize a given problem. Numerical algorithms as well as computer software often require a very specific form. Some problems can also be simplified:

Let $g : \mathbb{R}^n \to \mathbb{R}$ strictly growing on $Im(f)$

(R1) $\boxed{\min_{x \in X \subseteq \mathbb{R}^n} g(f(x)) = g(\min_{x \in X \subseteq \mathbb{R}^n} f(x))}$ min of g = g (min of f)

$\boxed{\arg\min_{x \in X \subseteq \mathbb{R}^n} g(f(x)) = \arg\min_{x \in X \subseteq \mathbb{R}^n} f(x)}$ argmin of g is argmin of f

Specific cases

(R11) $\boxed{\arg\min_{x \in X \subseteq \mathbb{R}^n}(f(x) + c) = \arg\min_{x \in X \subseteq \mathbb{R}^n} f(x)}$ $\forall c \in \mathbb{R}$

(R12) $\boxed{\min_{x \in X \subseteq \mathbb{R}^n}(f(x) + c) = (\min_{x \in X \subseteq \mathbb{R}^n} f(x)) + c}$ $\forall c \in \mathbb{R}$

(R13) $\boxed{\arg\min_{x \in X \subseteq \mathbb{R}^n} \log(f(x)) = \arg\min_{x \in X \subseteq \mathbb{R}^n} f(x)}$ $\forall c \in \mathbb{R}$

Example

$$\min_{x \in X \subseteq \mathbb{R}^2} \sqrt{x_1^2 + x_2^2} = \min_{x \in X \subseteq \mathbb{R}^2} (x_1^2 + x_2^2)$$

(R2) $\boxed{\max_{x \in X \subseteq \mathbb{R}^n} f(x) = -\min_{x \in X \subseteq \mathbb{R}^n} -f(x)}$

(R3) *Constraints and inequalities*

(R31) $\boxed{g(x) \leq 0 \Leftrightarrow -g(x) \geq 0}$

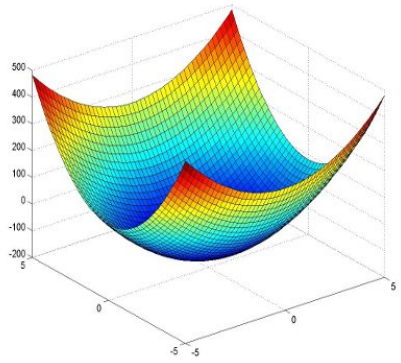
(R32) $\boxed{g(x) = 0 \Leftrightarrow \begin{cases} g(x) \geq 0 \\ g(x) \leq 0 \end{cases}}$

(R33) $\boxed{x \geq a \Leftrightarrow \begin{cases} \tilde{x} \geq 0 \\ \text{with } \tilde{x} = x - a \end{cases}}$

1.5 The score function

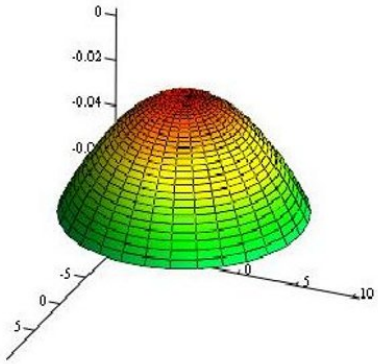
1.5.1 Definition

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function and x, y two number such that $x, y \in \mathbb{R}$ and $\lambda \in [0, 1]$



A surface is convexe
$\Leftrightarrow f(\lambda x+(1-\lambda)y) \leq \lambda f(x)+(1-\lambda)f(y))$

In other terms, a convex surface is always above the tangent hyper plans.



A surface is concave
$\Leftrightarrow f(\lambda x+(1-\lambda)y) \geq \lambda f(x)+(1-\lambda)f(y))$

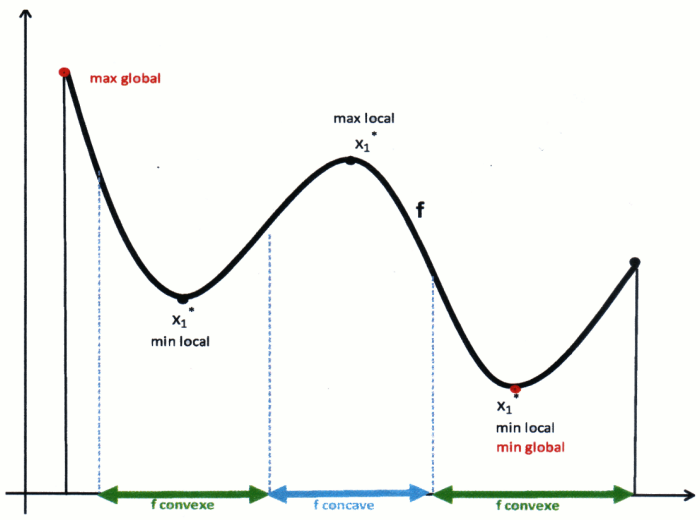
In other terms, a concave surface is always below the tangent hyper plans.

1.5.2 Note

Convex and concave surfaces play a major role in differential optimisation. One should note that **only convex surfaces are minimizable** while **only concave surfaces are maximizable**.

Also, a function is rarely globally convex or globally concave. Being concave or convex is rather a local property. Every function can be decomposed or split into chunks where each chunk is either concave or convex.

The inflection points are the separators of these domains. On each domain where the function is locally either convex or concave, the function accepts a local minimum, respectively a local maximum.



The global minimum (resp. maximum) is then the minimum (rexp. maximum) of the local optimums. One should take great care not to forget to evaluate the function on the inflection points as well as these might well be the local optimums !

1.5.3 Local differential optimisation

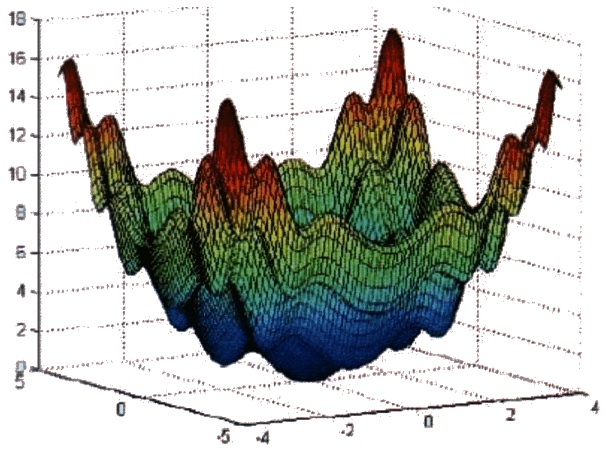
From the above reflection comes the term of local differential optimisation. Each and every algorithm we will be seing are very sensitive to the starting point. From the good and keen choice of this starting point depends the local optimum the algorithm will converge to.

The right choice of the starting point is thus an essential step in differential optimisation. The way this starting point is chosen is usually induced by the nature of the problem to be solved. There is no universal algorithm for this choice.
Whenever this is possible, one should perform a senstitivity analysis (plot) in an attempt to feel as good as possible the score function.

Surfaces of multiple variables functions can as well posess local minimums and maximums as one can see in the following function:

$$f(x_1, x_2) = \frac{({x_1}^2 + {x_2}^2)}{2} + sin({x_1}^2) + sin({x_2}^2)$$

which has the following surface:



1.5.4 Hypothesis

The this lesson, we will mostly considere convexe score functions and will focus on spotting the minimum in this domain. Also, in this introductory document, we wont take constraints in consideration.

1.6 Practice

1.6.1 Exercise 1: Transformation

Let $f(x) = e^{ln(s)}$ be a function we are trying to maximize. Let's assume we got some NaN messages from the optimizing software. We decide then to inject only the $h(x)$ function instead of the complete one in order to make it behave correctly. The results we get this way are:

$$\max h(x) = m_h$$
$$\arg\max h(x) = x^*$$

What can we conclude for

$$\max f(x)?$$
$$\arg\max f(x)?$$

From (R21) $\min g(f(x)) = g(\min f(x))$ we get $\mathbf{max\ f(x) = e^{max\ h(x)} = e^{m_h}}$
From (R21) $\arg\min g(f(x)) = \arg\min f(x)$ we get $\mathbf{argmax\ f(x) = argmax\ h(x) = x^*}$

CHAPTER 2

Mathematical introduction

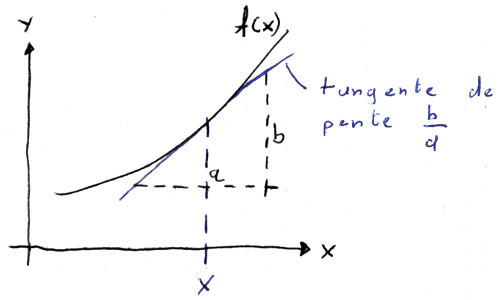
Contents

2.1 Recall of mathematical analysis

2.1.1 1D derivative

The derivative function gives us the slop of the line tangent to a specific point on a curve.

Below a 2D dimensionnal derivative function. In this lecture we consider this as 1D (one single dimension) as the response variable (y) depends on a single input variable (x)



Notation: $f'(x) = \frac{df}{dx}(x) = \partial x$

Properties:

Let $f, g : \mathbb{R} \to \mathbb{R}$

$$\boxed{(f+g)' = f' + g'}$$

$$\boxed{(\alpha f)' = \alpha f'} \qquad \forall \alpha \in \mathbb{R}$$

We have the following canonical derivative:

$$\boxed{(e^x)' = e^x} \qquad \boxed{(x^n)' = nx^{n-1}} \qquad \forall n \in \mathbb{N}$$

Also:

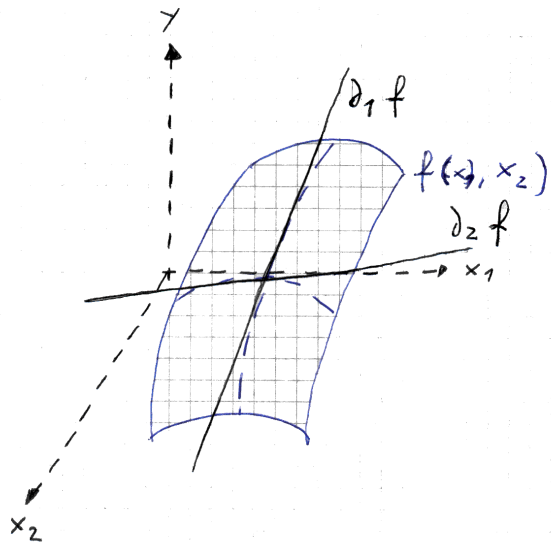
$$\boxed{(a \cdot x)' = a \cdot x^0 = a \cdot 1 = a}$$

$$\boxed{(x)' = 1} \qquad \boxed{(a)' = 0}$$

See http://en.wikipedia.org/wiki/Derivative for other derivatives.

2.1.2 2D derivative

In 2D (2 variable dimensions means 3D considering in addition the response variable) we usually considere partial differential equations, i.e **partial derivative** (or directionnal derivatives which we will see later).



In partial differential analysis, only 1 variable varies at a time. It's as taking a slice of the surface and considering the tangent to this curve delimited by the slice. Slicing occurs parallel to the axis.

→ let's fix x_2 and vary x_1

$$f_{x_2}(x_1) = \frac{df_{x_2}(x_1)}{dx_1} = \frac{\partial f}{\partial x_1}(x_1, x_2) = \partial_1 f(x_1, x_2)$$

→ Now let's fix x_1 and vary x_2

$$f_{x_1}(x_2) = \frac{df_{x_1}(x_2)}{dx_2} = \frac{\partial f}{\partial x_2}(x_1, x_2) = \partial_2 f(x_1, x_2)$$

2.1.3 Secondary partial derivative

A secondary partial derivative conists in deriving twice the function according to 1 dimension.

Partial derivative in x_1 and x_2 : $\frac{\partial^2 f}{\partial x_1 \partial x_2}(x_1, x_2) = \partial_{12} f(x_1, x_2) \equiv \partial_{21} f(x_1, x_2)$

The equivalence $\partial_{12} f(x_1, x_2) \equiv \partial_{21} f(x_1, x_2)$ is true $\Leftrightarrow$ the function is continue. **This is the Cauchy-Schwartz theorem.**

Deriving twice in the same dimension also makes it a secondary partial derivative, for instance $\partial_{11} f(x_1, x_2)$, or $\partial_{22} f(x_1, x_2)$, etc.

For a question of simplicity, we will often use as shorter notation when using partial derivatives:

$$\partial_1 f(x_1, x_2) = \partial_1 f \qquad \partial_2 f(x_1, x_2) = \partial_2 f \qquad \partial_{21} f(x_1, x_2) = \partial_{21} f$$

(See 2.4.1 Exercise 2: recall on derivatives)

2.1.4 Partial differential equations

Quoting wikipedia on http://en.wikipedia.org/wiki/Partial_differential_equation : ***Partial differential equations*** *(PDE) are a type of differential equation, i.e., a relation involving a function of several independent variables and their* ***partial derivatives*** *with respect to those variables.*

C^2 functions:

Let $f : \mathbb{R}^2 \to \mathbb{R}$ a continue fonction twice differentiable with continued derivatives.
This type of functions such as f are called C^2 functions.

A function that is at least one time differentiable is called a C^1 function.

2.2 The *Gradient* vector

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a C^1 function.

The function $\nabla f(x) : \mathbb{R}^n \to \mathbb{R}^n$ is called the gradient of f and is defined by:

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \dots \\ \frac{\partial f(x)}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \partial_1 f(x) \\ \dots \\ \partial_n f(x) \end{pmatrix}$$

Whenever f is a 2D function:

$$f : \mathbb{R}^2 \to \mathbb{R}$$
$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \to f(x_1, x_2)$$

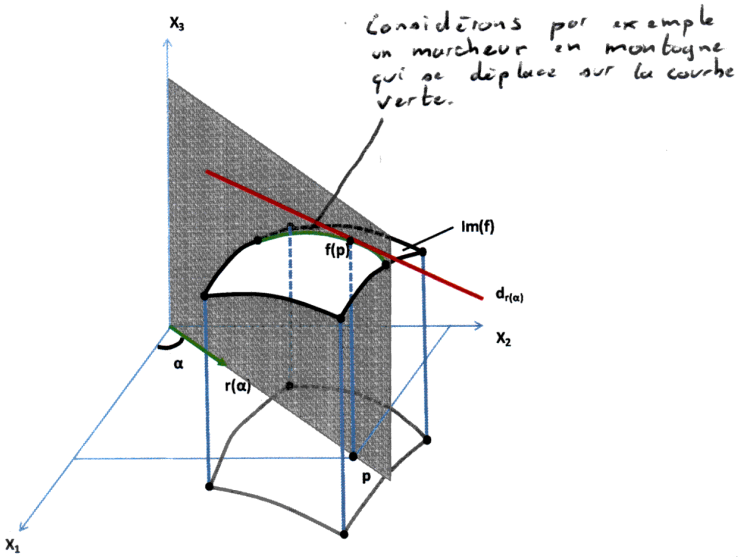
$$\nabla f(x) = \begin{pmatrix} \partial_1 f(x) \\ \partial_2 f(x) \end{pmatrix}$$

The **gradient vector** is the vector formed by the set of *partial derivatives* in each dimension.
In higher than 1 dimension, the gradient vector, in its function, is a very similar concept to the derivative in 1 Dimension..
In 1D, for instance, when the slope of a curve is 0, meaning we reached a maximum or a minimum (perhaps local), the derivative is 0. In higher dimensions, at the places where the tangent plane is parrallel to the ground, *the gradient vector is the null vector* as the project of the normal on the surface is a point.

2.2.1 Directional derivatives

In the section 2.1.2 above about **partial derivatives** we have seen that the partial derivative consists in performing a derivation following of the the dimension axis (x_1 or x_2) $\Rightarrow$ this means cutting the surface with a plan parallel to one of the axis.

But one can derive a function following an arbitrary direction. This is called **directional derivative**. Thus, we do not anymore considere a plan parallel to one of the axis, but following an arbitrary angle α:



Let $f : \mathbb{R}^n \to \mathbb{R}$ be a C^1 function. Let $x \in \mathbb{R}$ and $d \in \mathbb{R}^n$, d is a *direction vector*.
The *directionnal derivative* of f in x following d is:

$$\boxed{\partial_d f(x) = \nabla f(x)^t d \in \mathbb{R}}$$

(value of the matricial/scalar product between the transpose of the gradient vector and the direction vector)
= slope of the red line

It is not strictly required that d is a unit vector. If one replaced d by a colinear vector such as αd, the directional derivatove needs to be multiplied by the α factor as well:

$$\boxed{\partial_{\alpha d} f(x) = \alpha \partial_d f(x)}$$

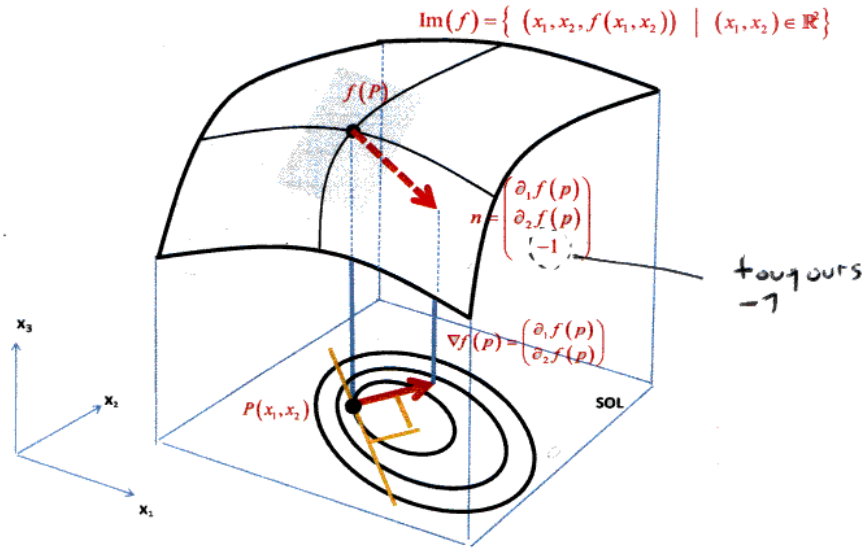
In order to get the ***uniformed standardized* directionnal derivative**, one needs to use a vector of length 1:

$$\partial_d f(x) = \frac{\nabla f(x)^t \cdot d}{||d||}$$

2.2.1.1 Properties

- By using $d = e_i$ the standard base vector of $\mathbb{R}^n$, we get the (usual) partial derivatives.
- Caution: the directionnal derivative being defined relatively to a multiplicative factor, **it is recommended to work with directions d having a *norm* of length 1, i.e. *unit vectors* when one wants to compare slopes**.
  This is called **normalizing the directional derivative**

2.2.2 Properties of the gradient vector



Here on the $Im(f)$ surface we block a point $P = (x_1, x_2)$ on the ground. The position of the point P on the surface is given by $(x_1, x_2, f(x_1, x_2))$.

The *surface normal* vector in P is given by $n = \begin{pmatrix} \partial_1 f(p) \\ \partial_2 f(p) \\ -1 \end{pmatrix}$.

The *gradient* in P is the projection of the *normal normal* on the ground. One simply gets rid of the third coordinate and finds: $\nabla f(p) = \begin{pmatrix} \partial_1 f(p) \\ \partial_2 f(p) \end{pmatrix}$

Properties:

- The **surface normal**, i.e. a vector, of the surface $f(x_1, x_2)$ is $n = (\partial_1 f, \partial_2 f, -1)$
  Reminder: the normal vector is a vector that is perpendicular to that surface. [1]
  This vector is not necessarily a unit vector, i.e. it's norm is not necessarily 1 (and most likely won't be).

- The *gradient* is a **projection** on the ground of n = the *surface normal* of $f(x)$.

- The *gradient* is always **perpendicular** to the contour lines (*"courbes de niveau"*).
  I.e. the gradient is always perpendicular to the *tangent line* to the contour lines.

- The *gradient* gives the direction of the steepest slope.

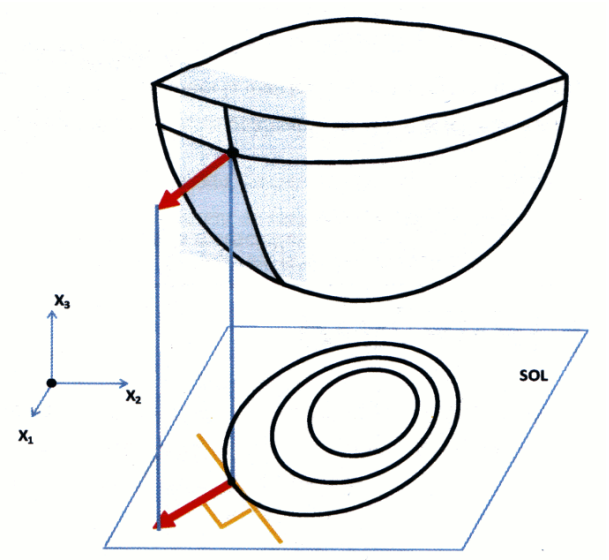
- The *anti-gradient* (negative of the gradient) always gives the steepest descent.
  For a convexe function, d is a descending direction $\Leftrightarrow d^t \nabla f(x) < 0$

We should always *follow* the gradient ($\nabla f(x)$) to climb as fast as possible and always *follow* the anti-gradient ($-\nabla f(x)$) to go down as fast as possible.
For instance, on a mountain, to reach the top, an approach could be to follow the gradient all the time at every step.

One should note that when the surface is concave, the *normal vector* points to the inside of the surface. On the contrary when the surface is convexe, the *normal vector* points on the outside.

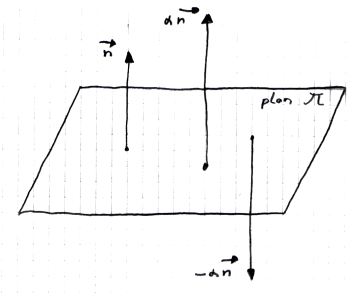
In every case, the *normal vector* heads to the bottom as its third component is -1, a negative value.

[1] A normal to a non-flat surface at a point P on the surface is a vector perpendicular to the tangent plane to that surface at P



2.2.3 Demonstration: the formula of the normal vector

We provide here a demonstration of the formula for the normal vector $n = \begin{pmatrix} \partial_1 f(p) \\ \partial_2 f(p) \\ -1 \end{pmatrix}$.



A plan surface has an infinity of normal vectors $\vec{n}$, $\alpha\vec{n}$, $-\alpha\vec{n}$, etc.

Let's imagine $\vec{n}$ is define with the following components, $n = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$

$-\frac{1}{c} \times \vec{n}$ is still a normal vector: $\begin{pmatrix} -\frac{a}{c} \\ -\frac{b}{c} \\ -1 \end{pmatrix}$ yet this with the last component set to -1.

Let us now considere our vector n such as $n = \begin{pmatrix} -\frac{a}{c} \\ -\frac{b}{c} \\ -1 \end{pmatrix} = \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix}$ with $n_3 = -1$ and let's try to resolve n_1 and n_2 to something more graceful.

For this, let us considere the point A on the plan π. Now if we considere another point P. we now that the point P also belongs to plan $\pi \Leftrightarrow$ the vector AP is orthogonal to the normal vector

$\Leftrightarrow \vec{AP} \perp \vec{n}$.

$$
\begin{aligned}
P \in \pi &\Leftrightarrow \vec{AP} \perp \vec{n} && (\vec{AP} \text{ ortho. to normal}) \\
&\Leftrightarrow \vec{AP} \cdot \vec{n} = 0 \\
&\Leftrightarrow (\vec{OP} - \vec{OA}) \cdot \vec{n} = 0 && \text{(Chasles relation)} \\
&\Leftrightarrow \begin{pmatrix} x_1 - a_1 \\ x_2 - a_2 \\ x_3 - a_3 \end{pmatrix} \cdot \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix} = 0 \\
&\Leftrightarrow n_1(x_1 - a_1) + n_2(x_2 - a_2) + n_3(x_3 - a_3) = 0 \\
&\Leftrightarrow x_1 n_1 - n_1 a_1 + n_2 x_2 - a_2 n_2 + n_3 x_3 - a_3 n_3 = 0 \\
&\Leftrightarrow x_1 n_1 + n_2 x_2 + n_3 x_3 = \underbrace{n_1 a_1 + a_2 n_2 + a_3 n_3}_{c} && \text{(constant as doesn't depend on x)} \\
&\Leftrightarrow x_3 = \frac{c - (x_1 n_1 + x_2 n_2}{n_3} \\
&\Leftrightarrow f(x_1, x_2) = \frac{c - (x_1 n_1 + x_2 n_2}{n_3} && (x_3 \text{ = val of the func. = } f(x_1, x_2))
\end{aligned}
$$

Now what if we derive $f(x_1, x_2)$ according to that new expression of f:

$$
\begin{aligned}
\partial_1 f = -\frac{n_1}{n_3} \quad \text{with } n_3 = -1 \Rightarrow \boxed{\partial_1 f = n_1} \\
\partial_2 f = -\frac{n_2}{n_3} \quad \text{with } n_3 = -1 \Rightarrow \boxed{\partial_2 f = n_2}
\end{aligned}
$$

And hence $n = \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix} = \begin{pmatrix} \partial_1 f \\ \partial_2 f \\ -1 \end{pmatrix}$

(See practice **TODO ref** for another demonstration)

2.3 The *Hessian* matrix

The **hessian matrix** (*fr: matrice hessienne*) is built using the secondary partial derivatives. The function it defines is called the **Hessian** (*fr: Le Hessien*).

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a C^2 function. The function $\nabla^2 f(x) : \mathbb{R}n \to M_{n\times n}(\mathbb{R})$ is called the *Hessian* of f. It is defined by the *hessian* matrix:

$$
\boxed{\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial {x_1}^2} & \dots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \dots & \dots & \dots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \dots & \frac{\partial^2 f(x)}{\partial {x_n}^2} \end{pmatrix} = \begin{pmatrix} \partial_{11} f(x) & \dots & \partial_{1n} f(x) \\ \dots & \dots & \dots \\ \partial_{n1} f(x) & \dots & \partial_{nn} f(x) \end{pmatrix} \in M_{n\times n}(\mathbb{R})}
$$

$\nabla^2 f(x)$ is a symetric matrix as $\partial_{12} f(x) = \partial_{21} f(x) \Leftarrow$ Cauchy-Schwartz, the derivation order is not relevant.

As such one only needs to compute, for instance for a 3×3 matrix:

$$\nabla^2 f = \begin{pmatrix} \partial_{11} f(x) & * & * \\ \partial_{21} f(x) & \partial_{22} f(x) & * \\ \partial_{31} f(x) & \partial_{32} & \partial_{33} f(x) \end{pmatrix}$$

The *hessian matrix* provides a description of the **curvature** (*fr: courbure*) of the slope.

2.3.1 The curvature

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a C^2 function. Let $x \in \mathbb{R}$ and $d \in \mathbb{R}^n$, d is a *direction vector*.
The *curvature* of f in x following d is:

$$curvature_{(f,d)}(x) = \frac{d^t \nabla^2 f(x) d}{d^t d}$$

One should note that the curvature in the opposite direction of d is exactly the same that the curvature in the d direction.

$\leftarrow$ (combination of matrix (scalar) product between the direction vector, the transpose of the direction vector and the Hessian)

2.4 Practice

2.4.1 Exercise 2: recall on derivatives

2.4.1.1 Part I : simple derivatives

Compute:

- $(3e^x)' = 3(e^x)' = 3e^x$
- $(2x^3)' = 3(x^3)' = 2 \times 3 \times x^{3-1} = 6x^2$
- $(\frac{1}{2}x^2 - 5e^x)' = (\frac{1}{2}x^2)' - (5e^x)' = x - 5e^x$

2.4.1.2 Part II : partial derivatives

Let $f(x_1, x_2) = 3e^{x_1} - 4x_1x_2 + 7{x_2}^2$

Compute:

- $\partial_1 f(x_1, x_2) = 3e^{x_1} - 4x_2$
- $\partial_2 f(x_1, x_2) = -4x_1 + 14x_2$
- $\partial_{12} f(x_1, x_2) = 0$
- $\partial_{21} f(x_1, x_2) = 0$

As an observation, one should note that as expected $\partial_{12} f(x_1, x_2) = \partial_{21} f(x_1, x_2)$

2.4.2 Exercise 3: The normal vector

2.4.2.1 Part I : Gradient

Let $n = (-3 \;\; 4 \;\; \pi)^t$ a vector normal to the surface of $f(x_1, x_2)$ at the point x_a. What is the gradient of f at the point x_a ?

Let's first express the normal vector in the usual form, with the third component set to -1:

$$\begin{pmatrix} \partial_1 f \\ \partial_2 f \\ -1 \end{pmatrix} = \begin{pmatrix} -3 \\ 4 \\ \pi \end{pmatrix} \cdot -\frac{1}{\pi} = \begin{pmatrix} \frac{3}{\pi} \\ -\frac{4}{\pi} \\ -1 \end{pmatrix}$$

We know now $\partial_1 f = \frac{3}{\pi}$ and $\partial_2 f = -\frac{4}{\pi}$

Thus $\nabla f = (\frac{3}{\pi} \quad -\frac{4}{\pi})$

2.4.2.2 Part II : Normal vector

Let $\nabla f(x_a) = (7 \;\; 3)^t$ the gradient of $f(x_1, x_2)$ at the point x_a. What is the normal vector to the surface of $f(x_1, x_2)$ at the point x_a ?

We know the gradient is the projection on the ground of the normal vector, so we know x_{a1} and x_{a2} of the normal vector. In addition, we know the third component is -1.
Thus $n_{x_a} = (7 \;\; 3 \;\; -1)^t$

2.4.3 Exercise 4 : Gradient

Let $f(x_1, x_2, x_3) = e^{x_1} + {x_1}^2 x_3 - x_1 x_2 x_3$

2.4.3.1 Part I : Gradient

Compute the gradient $\nabla f(x)$ of f:

$$\nabla f(x_1, x_2, x_3) = \begin{pmatrix} \partial_1 f \\ \partial_2 f \\ \partial_3 f \end{pmatrix} = \begin{pmatrix} e^{x_1} + 2x_1 x_3 - x_2 x_3 \\ -x_1 x_3 \\ {x_1}^2 - x_1 x_2 \end{pmatrix}$$

2.4.3.2 Part II : Directional derivative

Compute the directional derivative of f in the direction $d = (d_1 \;\; d_2 \;\; d_3)^t$

The directional derivative $\partial_d f(x)$ is the transpose of the gradient multiplied by the direction: $\nabla f(x)^t d$. Thus:

$$\begin{aligned}\partial_d f(x) &= \nabla f(x)^t d \\ &= d^t \nabla f(x) \\ &= (d_1\ d_2\ d_3) \begin{pmatrix} e^{x_1} + 2x_1x_3 + x_2x_3 \\ -x_1x_3 \\ {x_1}^2 - x_2x_3 \end{pmatrix} \\ &= d1(e^{x_1} + 2x_1x_3 + x_2x_3) - d_2(x_1x_3) + d_3({x_1}^2 - x_2x_3)\end{aligned}$$

2.4.4 Exercise 5 : Steepest descend, curvature

Let f be a function of the form:

$$f(x_1, x_2) = \frac{1}{2}{x_1}^2 + 2{x_2}^2$$

Part I

Compute the direction of the steepest slope in (1, 1)

Part I - Solution

The gradient is $\nabla f(x) = (x_1\ \ 4x_2)^t$

The steepest slope occurs in the direction of the anti-gradient:

$$-\nabla f(x) = (-x_1\ \ -4x_2)^t = (-1\ \ -4)^t$$

Part II

Compute the directionnal derivatives, then the normalized directionnal derivatives in the following directions:

- $d_1 = -\nabla f$
- $d_2 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$
- $d_3 = \begin{pmatrix} 1 \\ -3 \end{pmatrix}$

And make sure the anti-gradient actually is the *steepest slope*!

Part II - Solution

The directionnal derivative is computed as follows:

$$\partial_d f(x) = \nabla f(x)^t d \qquad = (x_1\ \ 4x_2) \begin{pmatrix} x_1 \\ 4x_2 \end{pmatrix} = d_1x_1 + 4d_2x_2$$

- $d_1 = -\nabla f \Rightarrow \partial_{d_1} f(x) = -{x_1}^2 - 16{x_2}^2 \Rightarrow$ in $(1,1) : \partial_{d_1} f(1,1) = -1 - 16 = -17$

- $d_2 = \begin{pmatrix} -1 \\ -1 \end{pmatrix} = -x_1 - 4x_2 \Rightarrow$ in $(1,1) : \partial_{d_1} f(1,1) = -1 - 4 = -5$
- $d_3 = \begin{pmatrix} 1 \\ -3 \end{pmatrix} = x_1 - 12x_2 \Rightarrow$ in $(1,1) : \partial_{d_1} f(1,1) = 1 - 12 = -11$

Caution here : one cannot take any conclusion with these values since the direction vector needs to be normalized before any comparison can be performed.

Normalizing the direction vectors:

- $d_1 = \frac{d_1}{||d_1||} = \frac{1}{||\sqrt{-x_1{}^2 + (-4x_2)^2}||} \begin{pmatrix} -x_1 \\ -4x_2 \end{pmatrix} \Rightarrow$ in $(1,1) : \frac{1}{||\sqrt{1+16}||} \begin{pmatrix} -1 \\ -4 \end{pmatrix}$
- $d_2 = \frac{d_2}{||d_2||} = \frac{1}{||\sqrt{-1^2 + (-1)^2}||} \begin{pmatrix} -1 \\ -1 \end{pmatrix} = \frac{1}{||\sqrt{2}||} \begin{pmatrix} -1 \\ -1 \end{pmatrix}$
- $d_3 = \frac{d_3}{||d_3||} = \frac{1}{||\sqrt{1^2 + (-3)^2}||} \begin{pmatrix} 1 \\ -3 \end{pmatrix} = \frac{1}{||\sqrt{10}||} \begin{pmatrix} 1 \\ -3 \end{pmatrix}$

From here we can compute the *normalized directionnal derivatives*:

- $\partial_{d_1} f(1,1) = \frac{-17}{\sqrt{17}} = -4.1231$
- $\partial_{d_2} f(1,1) = \frac{-5}{\sqrt{2}} = -3.5355$
- $\partial_{d_2} f(1,1) = \frac{-11}{\sqrt{10}} = -3.4785$

Which confirms that the anti-gradient is the steepest slope

Part III

Compute the curvature of f in the following directions

- $d_1 = (1 \;\; 4)^t$
- $d_2 = (1 \;\; 1)^t$
- $d_3 = (-1 \;\; 3)^t$

Then compute the condition number of f. Specifically, compute the highest and the littlest curvature of function f. Check the coherence of this new informations with the curvatures computed above.

Part III - Solution

The curvature has the following formula : $curvature_{(f,d_i)}(x) = \frac{{d_i}^t \nabla^2 f(x) d_i}{{d_i}^t d_i}$. We need the hessian to compute it:

$$\nabla^2 f(x) = \begin{pmatrix} \partial_{11} f(x) & \partial_{12} f(x) \\ \partial_{21} f(x) & \partial_{22} f(x) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix} \in M_{2\times 2}(\mathbb{R}) \text{ (symetrical)}$$

The curvature is hence independent from the position of x. As such:

- $curvature_{(f,d_1)}(x) = \frac{{d_1}^t \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix} d_1}{{d_1}^t d_1} = \frac{(1 \;\; 4) \begin{pmatrix} 1 \\ 16 \end{pmatrix}}{17} = \frac{65}{17}$
- $curvature_{(f,d_2)}(x) = \frac{5}{2}$

- $curvature_{(f,d_3)}(x) = \frac{37}{10}$

The greatest and the littlest possible curvatures of the function f are given by the *eigenvalues* of the hessian. Here the hessian is a diagonal matrix hence the eigenvaluesd are read in the diagonal: 1 and 4.
The condition number hence is $\frac{4}{1} = 4$
In addition, one can check that the curvatures computed above are indeed within the bounds $[1, 4]$.

2.4.5 Exercise 6 : Hessian

Let $f(x_1, x_2, x_3) = e^{x_1} + {x_1}^2 x_3 - x_1 x_2 x_3$

Compute the hessian matrix of f:

$$\nabla^2 f(x_1, x_2, x_3) = \begin{pmatrix} \partial_{11} f & * & * \\ \partial_{21} f & \partial_{22} f & * \\ \partial_{31} f & \partial_{32} f & \partial_{33} f \end{pmatrix} = \begin{pmatrix} e^{x_1} + 2x_3 & * & * \\ -x_3 & 0 & * \\ 2x_1 - x_2 & -x_1 & 0 \end{pmatrix}$$

Using the fast calculatiom method:

$$\begin{aligned}\partial_{12}f &= \partial_{12}(e^{x_1}) + \partial_{12}({x_1}^2 x_3) - \partial_{12}(x_1 x_2 x_3)\\ &= 0 + 0 - x_3\\ &= -x_3\end{aligned}$$

$$\begin{aligned}\partial_{13}f &= \partial_{13}(e^{x_1}) + \partial_{13}({x_1}^2 x_3) - \partial_{13}(x_1 x_2 x_3)\\ &= 0 + 2x_1 - x_2\\ &= 2x_1 - x_2\end{aligned}$$

$$\begin{aligned}\partial_{23}f &= \partial_{23}(e^{x_1}) + \partial_{23}({x_1}^2 x_3) - \partial_{23}(x_1 x_2 x_3)\\ &= 0 + 0 - x_1\\ &= -x_1\end{aligned}$$

$$\begin{aligned}\partial_{11}f &= \partial_{11}(e^{x_1}) + \partial_{11}({x_1}^2 x_3) - \partial_{11}(x_1 x_2 x_3)\\ &= e^{x_1} + 2x_3 - 0\\ &= e^{x_1} + 2x_3\end{aligned}$$

$$\begin{aligned}\partial_{22}f &= \partial_{22}(e^{x_1}) + \partial_{22}({x_1}^2 x_3) - \partial_{22}(x_1 x_2 x_3)\\ &= 0 + 0 - 0\\ &= 0\end{aligned}$$

$$\begin{aligned}\partial_{33}f &= \partial_{33}(e^{x_1}) + \partial_{33}({x_1}^2 x_3) - \partial_{33}(x_1 x_2 x_3)\\ &= 0 + 0 - 0\\ &= 0\end{aligned}$$

2.4.6 Exercise 7 : Normal and gradient vectors

TODO

2.4.7 Exercise 10 : Plan curvature

Compute the curvature of a plan surface and give your observations. A plan surface function has the form: $f(x_1, x_2) = a_1 x_1 + 1_2 x_2 + a_3$

2.4.7.1 Primary derivatives

- $\partial_1 f = 1$
- $\partial_2 f = 1$

2.4.7.2 Secondary derivatives

- $\partial_{11} = \partial_{12} = \partial_{21} = \partial_{22} = 0$

2.4.7.3 Hessian matrix

- $\nabla^2 f = \begin{pmatrix} \partial_{11} & \partial_{12} \\ \partial_{21} & \partial_{22} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$

2.4.7.4 Curvature

Whatever direction d we look at, and whatever point $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ we stand at, the curvature would be:

$$curvature_{(f,d)}(x_1, x_2) = \frac{d^t \nabla^2 f d}{d^t d} = \frac{d^t 0 d}{d^t d} = 0$$

CHAPTER 3

Introduction to matrix calculations

Contents

3.1 Introduction

We have seen in the previous chapter that derivatives and differential calculations enable us to get information on the geometry of a surface.
But there is another tool for this purpose: matrix calculation !
→ As we will now see, both are required for differential optimisation.

We will mostly limit our introduction on 2x2 matrices in this chapter as such matrix are sufficient for most cases in differential optimisation.

3.1.1 General form

The general form of a Matrix of size $m \times n$ with its coefficient in R is as follows:

$$\text{Matrix A :} A \in \mathbb{M}_{m\times n}(\mathbb{R}) = \begin{pmatrix} a_{11} & ... & a_{1n} \\ ... & ... & ... \\ a_{m1} & ... & a_{mn} \end{pmatrix} = (a_{ij}) \in \mathbb{M}_{m\times n}(\mathbb{R})$$

3.1.2 Transpose

The **transpose of a matrix** is another matrix corresponding to the initial one where the rows have become columns and the columns have become rows.
One should note that a matrix is equals to its transpose when it is a symetrical matrix, a diagonal matrix or the null matrix.

$$\begin{pmatrix} a & b & c \\ * & * & * \\ * & * & * \end{pmatrix} = \begin{pmatrix} a & * & * \\ b & * & * \\ c & * & * \end{pmatrix}^t$$

(← These notions are introduce later).

3.2 2 x 2 matrices

As stated in the introduction, we will mostly focus on 2×2 matrices in this chapter and in the remainder of the document for simplicity purpose as they are most often sufficient in differential optimisation.
A 2×2 has the following form:

$$\text{Matrix A : } A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = (a_{ij}) \in \mathbb{M}_{2\times 2}(\mathbb{R})$$

3.2.1 Properties

A few different characteristics of a 2 x 2 matrix can be easily computed.

Determinant : $\boxed{\det(A) = a_{11}a_{22} - a_{21}a_{12} \in \mathbb{R}}$

Notation: the determinant is sometimes noted $|A|$.
(Note: this is much more complicated for a greater than 2 x 2 matrix)

Trace : $\boxed{trace(A) = a_{11} + a_{22} \in \mathbb{R}}$

(Sum of the diagonal elements)

3.2.2 Inverting a 2 x 2 matrix

If $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = (a_{ij}) \in \mathbb{M}_{2\times 2}(\mathbb{R})$

Then $\boxed{A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix}}$

with $\det(A) = |A| = a_{11}a_{22} - a_{12}a_{21}$

3.2.3 Diagonal matrix

2 x 2 diagonal matrices are interesting in many ways, one of them being that they are easily inversible:

If $A = \begin{pmatrix} a_{11} & 0 \\ 0 & a_{22} \end{pmatrix} = (a_{ij}) \in \mathbb{M}_{2\times 2}(\mathbb{R})$ (and $a_{ij} = 0 \Leftrightarrow i \neq j$)

Then $\boxed{A^{-1} = \begin{pmatrix} a_{11}^{-1} & 0 \\ 0 & a_{22}^{-1} \end{pmatrix}}$

In addition, as we will see later, the eigen-values of a diagonal matrix are the values shown on the diagonal, the values in the matrix itself.

Also, multipliying two diagonal matrices is straightforward:

$$A \in diag\_\mathbb{M}_{2\times 2} \cdot B \in diag\_\mathbb{M}_{2\times 2} = \begin{pmatrix} a_{11} & 0 \\ 0 & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & 0 \\ 0 & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} & 0 \\ 0 & a_{22}b_{22} \end{pmatrix}$$

3.3 Quick geometry reminder

In 2D, the equation of a line is as follows : $ax_1 + b = y$
One can write it this way : $ax_1 + b = x_2$
Or even this way : $n_1x_1 + n_2x_2 + k + k = 0$ which gives us the equation of the line

The n_1 and n_2 coefficients give us the normal vector ($\perp line$) : $n = \begin{pmatrix} n_1 \\ n_2 \end{pmatrix}$

And as well (one of) the vector of the line: $\begin{pmatrix} -n_2 \\ n_1 \end{pmatrix}$

3.4 Spectral matrix analysis

Shortly put, the spectral anlysis of a matrix consists in finding its **eigenvalues** and **eigenvectors**. One also calls it the **diagonalization** of the matrix.

Let $A_{n\times n}$ be a square matrix

An eigenvalue λ and an eigenvector v of the matrix A are such that they respect the following property:

$$\boxed{Av = \lambda v} \Leftrightarrow = \begin{cases} v & \text{eigenvector of } A, \\ \lambda & \text{eigenvalue of } A \\ \lambda \in \mathbb{R} & \end{cases}$$

Not every matrix has eigenvalues and eigenvectors. However, symetric matrices always have eigenvalues and eignenvectors.

There are usually an infinite number of eigenvectors andf eigenvalues for a given matrix. The whole purpose of the exercise consists in finding the **eigenvalues** λ_i and the **eigenvectors** v_i such that $Av_i = \lambda v_i$.
We want at least two of the eingenvalues and eigenvectors.

Comming back to our initial equation $Av = \lambda v$, the issue we have here is that this is a one equation, two variables system which means it has an infinite number of solutions.
We will see a way for finding at least 2 solutions to this system.

3.4.1 Find eigenvalues

First, let's find the eigenvalues. Let's say:

$$\begin{aligned} Av = \lambda v &\Leftrightarrow Av - \lambda v = 0 \\ &\Leftrightarrow \underbrace{(A - \lambda I)}_{\text{is a matrix}} v = 0 && \text{With } I \text{ the identity matrix} \\ &\Leftrightarrow (A - \lambda I) = 0 && \text{and } v \text{ cannot be null, hence:} \\ &\Leftrightarrow \boxed{\det(A - \lambda I) = 0} && \text{a matrix is null} \Leftrightarrow \text{its } \textit{det} \text{ is null} \end{aligned}$$

For a 2 x 2 matrix, this is happily quite easily solved:

$$\begin{aligned} \det(A - \lambda I) = 0 &\Leftrightarrow \left| \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right| = 0 && \text{|...| = det(...)} \\ &\Leftrightarrow \begin{vmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{vmatrix} = 0 \\ &\Leftrightarrow (a_{11} - \lambda)(a_{22} - \lambda) - a_{21}a_{12} = 0 && \text{using the det(..) formula} \\ &\Leftrightarrow \lambda^2 - \lambda(\underbrace{a_{11} + a_{22}}_{trace(A)}) + \underbrace{a_{11}a_{22} - a_{21}a_{12}}_{det(A)} = 0 && \text{simply resolving the product} \\ &\Leftrightarrow \boxed{\lambda^2 - \lambda \cdot trace(a) + |A| = 0} && \text{(note: only works for 2 x 2 matrix) [1]} \end{aligned}$$

Note: $\lambda^2 - \lambda \cdot trace(a) + |A| = P_A(\lambda)$ *is the characteristic polynomial of A*

And now we face a well known quadratic equation we can easily solve using *Viet*:

$$ax^2 + bx + c = 0 \Leftrightarrow x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \rightarrow \text{in our case: } \begin{cases} x = \lambda & b = -trace(A) \\ a = 1 & c = det(A) \end{cases}$$

And thanks to this method, we find normally 2 (but at least 1 if both are equals) eigenvalues $\lambda_{1,2}$ which are the root of the characteristic polynom $P_A(\lambda)$.

Note λ_1 and λ_2 are the root of the characteristic polynoms and enable it's factorization $P_A(\lambda) = \lambda^2 - \lambda \cdot trace(a) + |A|) = (\lambda - \lambda_1)(\lambda - \lambda_2)$

3.4.2 Find eigenvectors

Having 2 (or at least 1) eigenvalue(s), we can now find the eigenvectors. This is quite easily done actually by simply injecting the eigenvalue in the initial system of equations. We actually do it two times, once for each egeinvalue and find this way usually different eigenvectors.

Let's assume we have λ_1 and λ_2. In order to solve **one of them** we can pose:

$$\begin{aligned} Av = \lambda v &\Leftrightarrow \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \lambda \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \\ &\Leftrightarrow \begin{cases} a_{11}v_1 + a_{12}v_2 - \lambda v_1 = 0 \\ a_{21}v_1 + a_{22}v_2 - \lambda v_2 = 0 \end{cases} \\ &\Leftrightarrow \boxed{\begin{cases} v_1(a_{11} - \lambda) + a_{12}v_2 = 0 \\ v_2(a_{22} - \lambda) + a_{21}v_1 = 0 \end{cases}} \end{aligned}$$

So all we need to do is inject first λ_1 to find v_1, then λ_2 to find v_2 in the above system of equations and we have our both eigenvectors.

3.4.3 Diagonalization

Once the eigenvalues $\lambda_i \in \mathbb{R}$ are found, the **diagonalization** of matrix A, **is a diagonal matrix** which possess the λ_i values on the diagonal and 0 everywhere else:

$$\boxed{diag(A) = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & ... & 0 \\ 0 & 0 & \lambda_n \end{pmatrix}}$$

It has the following important properties :

$\lambda_i > 0 \Rightarrow$ A is Positive-definite

A matrix $A_{n\times n}$ is positive-definite if each and every of its eigenvalues are positive.

$\lambda_i \geq 0 \Rightarrow$ A is Positive-**semi**definite

A matrix $A_{x\times x}$ is positive-semidefinite if each and every of its eigenvalues are positive or null.

Note: Every *symetric* matrix is *diagonalizable*.

3.4.3.1 Diagonal matrices specific cases

A diagonal matrix is a special matrix of the form: $\begin{pmatrix} x_1 & 0 & 0 \\ 0 & ... & 0 \\ 0 & 0 & x_n \end{pmatrix}$

In this case, the x_i values on the diagonal of the matrix **correspond strictly** to its eigenvalues λ_i. A *diagonal matrix* is always equals/identical to its own *diagonalization*.

3.4.4 Example

Let $A = \begin{pmatrix} 0 & -1 \\ 3 & 4 \end{pmatrix} \in \mathbb{M}_{n\times n}$

3.4.4.1 Eigenvalues

First we need the $trace(A)$ and $det(A)$:

- $trace(A) = a_{11} + a_{22} = 0 + 4 = \mathbf{4}$
- $\det(A) = a_{11}a_{22} - a_{21}a_{12} = 0 \times 4 - 3 \times (-1) = \mathbf{3}$

Let's inject those values in :

$$ax^2 + bx + c = 0 \Leftrightarrow x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \rightarrow \text{with: } \begin{cases} x = \lambda & b = -trace(A) = -4 \\ a = 1 & c = det(A) = 3 \end{cases}$$

Which gives us:

$$\lambda_{1,2} = \frac{4 \pm \sqrt{4^2 - 4 \times 3}}{2} = \frac{4 \pm \sqrt{4}}{2} = \begin{cases} 3 \\ 1 \end{cases}$$

3.4.4.2 Eigenvectors

First for $\lambda_1 = 1$

Let's assume v is the eigenvector matching the eigenvalue $\lambda_1 = 1$:

One simply needs to inject $\lambda_1 = 1$ as well as the matrix values in the equation system we have seen above:

$$\begin{cases} v_1(a_{11} - \lambda) + a_{12}v_2 = 0 \\ v_2(a_{22} - \lambda) + a_{21}v_1 = 0 \end{cases} \Leftrightarrow \begin{cases} v_1(0-1) + (-1)v_2 = 0 \\ v_2(4-1) + 3v_1 = 0 \end{cases}$$

$$\Leftrightarrow \begin{cases} -v_1 - v_2 = 0 \\ 3v_1 + 3v_2 = 0 \end{cases}$$

$$\Leftrightarrow \begin{cases} v_1 + v_2 = 0 \\ v_1 + v_2 = 0 \end{cases}$$ Can happen $\Rightarrow$ unsolved

$\Leftrightarrow$ An infinite number of eigenvectors

The above situation, where the system of equations appears to be unsolved can happen. In this case we have the equation of a line and all vectors parralel to this line are egienvectors. Using the quick geometry reminder introduced in 3.3, we can get one of these vectors:

$$v_1 + v_2 = 0 \Rightarrow n = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \Rightarrow v = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

One might want to normalize it:
$v = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

Second for $\lambda_2 = 3$

Let's assume w is the eigenvector matching the eigenvalue $\lambda_1 = 1$, injecting again $\lambda_2 = 3$ as well as the matrix values in the equation system we have seen above:

$$\begin{cases} w_1(a_{11} - \lambda) + a_{12}w_2 = 0 \\ w_2(a_{22} - \lambda) + a_{21}w_1 = 0 \end{cases} \Leftrightarrow \begin{cases} w_1(0-3) + (-1)w_2 = 0 \\ w_2(4-3) + 3w_1 = 0 \end{cases}$$

$$\Leftrightarrow \begin{cases} -3w_1 - w_2 = 0 \\ 3w_1 + 1w_2 = 0 \end{cases}$$

$$\Leftrightarrow \begin{cases} w_1 + w_2 = 0 \\ w_1 + w_2 = 0 \end{cases}$$

$\Leftrightarrow$ 1 equation, 2 variables $\Rightarrow$ infinite number of eigenvectors

Again using the equation of the line, on can find one of the eigenvectors:

$$3w_1 + w_2 = 0 \Rightarrow n = \begin{pmatrix} 3 \\ 1 \end{pmatrix} \Rightarrow w = \begin{pmatrix} -1 \\ 3 \end{pmatrix} \Rightarrow w_{norm} = \frac{1}{sqrt10} \begin{pmatrix} -1 \\ 3 \end{pmatrix}$$

3.5 Geometrical interpretation of the eigenvalues

Let $A_{n\times n}$ be a squared symetrical matrix and d_i an eigenvector eigen-λ_i. We know that:

$$\begin{aligned} Av = \lambda v \Leftrightarrow Ad_i &= \lambda_i d_i && \text{simple variable rename } v = d_i \\ \Leftrightarrow d_i^t A d_i &= d_i^t \lambda_i d && \text{divide by } d^t \\ \Leftrightarrow \boxed{\frac{d_i^t A d}{d_i^t d_i} = \lambda_i} & \end{aligned}$$

Now what if A is the Hessian of f, $A = \nabla^2 f(x)$?

$$\boxed{\lambda_i = \frac{d_i^t \nabla^2 f(x) d}{d_i^t d_i} = curvature_{(f,d_i)}(x)}$$

In other terms, if $A_{n\times n}$ is the *Hessian* of a twice differentiable ($\in C^2$) function f, then λ_i **gives us the curvature of f in the direction of the eigenvector** d_i.

The Hessian is a matrix of *functions of* x. Only for specific x values it is defined as a matrix of numbers.
Hence, the λ_i one might find provided solely the f function will be expressed as *functions of x* as well. Only for specific x values, i.e. specific and precise points in the domain, the λ_i eigenvalues and the eigenvectors will be defined as numbers, hence giving a *real direction* and *curvature* value.

3.5.1 The Rayleigh-Ritz theorem

Let f be a function in C^2.

- The highest possible curvature in x of $f : \mathbb{R}^2 \to \mathbb{R}$ is given by the highest eigenvalue of the Hessian $\nabla^2 f$.
- The smallest possible curvature in x of f is given by the smallest eigenvalue of the hessian.
- The corresponding eigenvectors provide us with the direction of these curvatures

The interesting implication of this is that by moving from a point in the direction of one of the eigenvector, one can follow either the minimal curvature or the maximale curvature.

3.6 Condition number

The condition number of a function with respect to an argument measures the asymptotically worst case of how much the function can change in proportion to small changes in the argument. The "function" is the solution of a problem and the "arguments" are the data in the problem. A problem with a low condition number is said to be **well-conditioned**, while a problem with a high condition number is said to be **ill-conditioned**.

Condition Number - Matrix:
The condition number of a *matrix* is the ratio between its highest eigenvalue and its lowest eigenvalue.

Condition Number - Function:
The condition number of a *function* is the condition number of its Hessian matrix $\nabla^2 f$, i.e. the ratio between its highest and its smallest eigenvalues.

As we have seen in 3.4, not every function has eigenvalues, but symetrical matrices always have. The hessian $\nabla^2 f$ is a symetrical matrix. How convenient ☺.

3.6.1 Geometrical interpretation of the condition number

Geometrically, the more the condition number of f is far from the value 1, the highest are the difference of the curvatures in the different directions we look at.

A function is:

ill-conditioned :	if it has an important difference of curvature - condition mumber - in two different directions.
well-conditioned :	if its condition number of **close to 1**

A condition number close to 1 means the *countour curves* are very close to concentric circles.

3.7 Practice

3.7.1 Exercise 8 : eigenvalues

Let A be the following matrix: $\begin{pmatrix} 5 & -3 \\ 6 & -4 \end{pmatrix}$

Compute the eigenvectors and the eigenvalues of matrix A.
Then answer: is A *positive-definite* or *positive-semidefinite* or nothing at all ?

3.7.1.1 Eigenvalues

According to 3.4.1, we need to resolve the *characteristic polynomial* of A :
$P_A(\lambda) = \lambda^2 - \lambda \cdot trace(a) + |A|$.

With:

- $|A| = \det(A) = a_{11}a_{22} - a_{12}a_{21} = -20 + 18 = -2$
- $trace(A) = a_{11} + a_{22} = 1$

Hence $P_A(\lambda) = 1\lambda^2 - 1\lambda - 2$, using Viet with $a = 1$, $b = -1$, $c = -2$:

$$\lambda_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{1 \pm \sqrt{1+8}}{2} = \frac{1 \pm 3}{2} = \begin{cases} 2 \\ -1 \end{cases}$$

Hence $\lambda_1 = -1$ and $\lambda_2 = 2$

3.7.1.2 Eigenvectors

$\vec{v}$ **for** $\lambda_1 = -1$

Using the formla in 3.4.2, we can pose:

$$\begin{cases} w_1(a_{11} - \lambda_2) + a_{12}v_2 = 0 \\ v_2(a_{22} - \lambda_2) + a_{21}v_1 = 0 \end{cases} \Rightarrow \begin{cases} v_1(5+1) + (-3)v_2 = 0 \\ v_2(-4+1) + 6v_1 = 0 \end{cases}$$
$$\Rightarrow \begin{cases} 6v_1 - 3v_2 = 0 \\ 6v_1 - 3v_2 = 0 \end{cases}$$
$$\Rightarrow 2v_1 = v_2$$
$$\Rightarrow \text{an infinite amount of solution, including: } \mathbf{v} = (\mathbf{1} \ \ \mathbf{2})^{\mathbf{t}}$$

$\vec{w}$ **for** $\lambda_2 = 2$

$$\begin{cases} v_1(a_{11} - \lambda_1) + a_{12}w_2 = 0 \\ w_2(a_{22} - \lambda_1) + a_{21}w_1 = 0 \end{cases} \Rightarrow \begin{cases} w_1(5-2) + (-3)w_2 = 0 \\ w_2(-4-2) + 6w_1 = 0 \end{cases}$$
$$\Rightarrow \begin{cases} 3w_1 - 3w_2 = 0 \\ 6w_1 - 6w_2 = 0 \end{cases}$$
$$\Rightarrow w_1 = w_2$$
$$\Rightarrow \text{an infinite amount of solution, including: } \mathbf{w} = (\mathbf{1} \ \ \mathbf{1})^{\mathbf{t}}$$

Normalized **eigenvectors**

One might want to normalize the eigenvectors:

- The eigenvector $v_{norm} = \frac{v}{||v||} = \frac{1}{\sqrt{5}}(12)^t$ matches the eigenvalue $\lambda_1 = -1$
- The eigenvector $w_{norm} = \frac{w}{||w||} = \frac{1}{\sqrt{2}}(11)^t$ matches the eigenvalue $\lambda_1 = 2$

3.7.1.3 Positive definition

As one of the eigenvalues is negative, the matrix A is neither *positive-definite* nor *positive-semidefinite*.

3.7.2 Exercise 9 : Eigenvalues of a diagonal matrix

Compute the eigenvalues of matrix A below and give your observations.

$$A = \begin{pmatrix} 4 & 0 \\ 0 & -3 \end{pmatrix}$$

According to 3.4.1, we need to resolve the *characteristic polynomial* of A :
$P_A(\lambda) = \lambda^2 - \lambda \cdot trace(a) + |A|$.

With:

- $|A| = \det(A) = a_{11}a_{22} - a_{12}a_{21} = -12 + 0 = -12$
- $trace(A) = a_{11} + a_{22} = 1$

Hence $P_A(\lambda) = 1\lambda^2 - 1\lambda - 12$, using Viet with $a = 1$, $b = -1$, $c = -12$:

$$\lambda_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{1 \pm \sqrt{1 + 48}}{2} = \frac{1 \pm 7}{2} = \begin{cases} 4 \\ -3 \end{cases}$$

which gives us the following diagonal matrix:

$$\mathrm{Diag}(A) = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} = \begin{pmatrix} 4 & 0 \\ 0 & -3 \end{pmatrix}$$

Observation: as expected, one falls back on the original diagonal matrix. As stated in 3.4.3.1, a *diagonal matrix* is always equal to its *diagonalization*

CHAPTER 4

Preconditionning

Contents

4.1 Preconditionning

It is a lot easier with any algorithm to optimise a *well-conditionned* function. For this reason, before running any algorithm, a *variable change* might need to be applied in order to have a *better condition-number* of the score function.
This variable change is called **Preconditionning**.

$\rightarrow$ The optimisation algorithm we will be seing later require us to have *countour lines* as close as possible to *concentric circles*.

4.1.1 Definition: preconditionning

Let $f : \mathbb{R}^n \to \mathbb{R}$ be C^2 function and x a vector in $\mathbb{R}^n$.

The **preconditionning** of f in x defines a **variable change** $\tilde{x} = Mx$ using :

- an inversible matrix $M_{n\times n}$ and
- a function $\tilde{f}(\tilde{x}) = f(M^{-1}\tilde{x})$

in such a way that the **condition number** of $\tilde{f}$ in $\tilde{x} = Mx$ is **better** than the **condition number** of f in x

4.1.2 Principle

Let $f(x)$ be an unknown function.

On can pose $\tilde{x} = g(x)$ and g is inversible
So $x = g^{-1}(\tilde{x})$
And hence $f(x) = \underbrace{f(g^{-1}}_{=\tilde{f}}(\tilde{x})) = \tilde{f}(\tilde{x})$

Example: let $f(x) = x^2 + 2$,
let's pose $\tilde{x} = x^2$,
hence $\tilde{f}(\tilde{x}) = \tilde{x} + 2$ which is simplier.

4.1.3 The Cholesky theorem

Let $f : \mathbb{R}^n \to \mathbb{R}$ be C^2 function with a Hessian matrix $\nabla^2 f$ *positive-definite* and x a vector in $\mathbb{R}^n$.

The best possible preconditionning of f in x occurs with the **variable change** induced by the ***Cholesky* decomposition of the Hessian**:

$$\nabla^2 f = LL^t = \begin{pmatrix} * & 0 & 0 \\ * & * & 0 \\ * & * & * \end{pmatrix} = \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{pmatrix}$$

(example for 3×3 matrix)

where $L_{n\times n}$ is a *triangular lower matrix*

If such an L matrix can be found, then the induced variable change is:

$$\tilde{x} = L^t x = \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{pmatrix} x$$

Shortly put, Cholesky states that:

- if the Hessian $\nabla^2 f$ of a function f is positive-definite, it is always possible to compute a Cholesky decomposition $\nabla^2 f = L \cdot L^t$
- Those both new matrices enables us to build the best possible preconditionning of the function f by posing
  - the variable change $\tilde{x} = L^t \cdot x$
  - the new function $\tilde{f}(\tilde{x}) = f(L^{-1}\tilde{x})$

4.1.3.1 The diagonal matrices case

Finding the cholesky decomposition for an usual positive-definite matrix usually is a tough task. But *diagonal matrices* form an interesting case as the Cholesky decomposition can be computed *out-of-the-box*.

Let's A be a matrix, $A \in diag\_\mathbb{M}_{2\times 2}$, then the matrix $L \in \mathbb{M}_{2\times 2}$ such that $A = L \cdot L^t$ is easy to find:

$$A = \begin{pmatrix} a_{11} & 0 \\ 0 & a_{22} \end{pmatrix} = \underbrace{\begin{pmatrix} \sqrt{a_{11}} & 0 \\ 0 & \sqrt{a_{22}} \end{pmatrix}}_{L} \cdot \underbrace{\begin{pmatrix} \sqrt{a_{11}} & 0 \\ 0 & \sqrt{a_{22}} \end{pmatrix}}_{L^t}$$

4.2 Example in 2D

Precondition function $f(x_1, x_2) = 2{x_1}^2 + 9{x_2}^2$

4.2.1 Compute Hessian

The condition number of a function is defined as the condition number of its Hessian matrix. We first need the Hessian matrix:

$$\nabla^2 f = \begin{pmatrix} \partial_{11} & \partial_{12} \\ \partial_{21} & \partial_{22} \end{pmatrix} = \begin{pmatrix} 4 & 0 \\ 0 & 18 \end{pmatrix}$$

$\rightarrow$ Luckily, the hessian is a diagonal matrix. Further computation will be easy.

Note: On can see that the *condition number* of the function f is indeed not very good. The condition number of a function is the condition number of its Hessian matrix.
Here the hessian matrix is a *diagonal matrix* and hence the eigenvalues are simply the numbers in the diagonal.
Hence $cond(f) = \dfrac{\lambda_{max}}{\lambda_{min}} = \dfrac{18}{4} = 4.5$ which is far from the target value 1.

4.2.2 Cholesky Decomposition

Now we should find the L matrix such that $\nabla^2 f = L \cdot L^t$. As we have seen in 4.1.3.1, this is easy for diagonal matrices:

$$\nabla^2 f = \begin{pmatrix} 4 & 0 \\ 0 & 18 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 3\sqrt{2} \end{pmatrix} \cdot \begin{pmatrix} 2 & 0 \\ 0 & 3\sqrt{2} \end{pmatrix}$$

4.2.3 Variable Change

We can now compute the $\tilde{x}$ variable from the x variable using the L matrix:

$$\tilde{x} = \begin{pmatrix} 2 & 0 \\ 0 & 3\sqrt{2} \end{pmatrix} x \Leftrightarrow \begin{cases} \tilde{x}_1 = 2x_1 \\ \tilde{x}_2 = 3\sqrt{2}x_2 \end{cases}$$

As well as x from the $\tilde{x}$ variable:

$$x = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{3\sqrt{2}} \end{pmatrix} \tilde{x} = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{\sqrt{2}}{6} \end{pmatrix} \tilde{x} \Leftrightarrow \begin{cases} x_1 = \frac{1}{2}\tilde{x}_1 \\ x_2 = \frac{\sqrt{2}}{6}\tilde{x}_2 \end{cases}$$

4.2.4 Compute function $\tilde{f}$

We can now compute the *preconditonned* $\tilde{f}$ function by substituting the x_1 and x_2 expressions in the original f function:

$$\begin{aligned} & f(x_1, x_2) = 2{x_1}^2 + 9{x_2}^2 \text{ and } \begin{cases} x_1 = \frac{1}{2}\tilde{x}_1 \\ x_2 = \frac{\sqrt{2}}{6}\tilde{x}_2 \end{cases} \\ & \Rightarrow \tilde{f}(\tilde{x}_1, \tilde{x}_2) = 2(\frac{1}{2}\tilde{x}_1)^2 + 9(\frac{\sqrt{2}}{6}\tilde{x}_2)^2 \\ & \Rightarrow \boxed{\tilde{f}(\tilde{x}_1, \tilde{x}_2) = \frac{1}{2}\tilde{x}_1^2 + \frac{1}{2}\tilde{x}_2^2} \end{aligned}$$

4.2.5 Condition Number

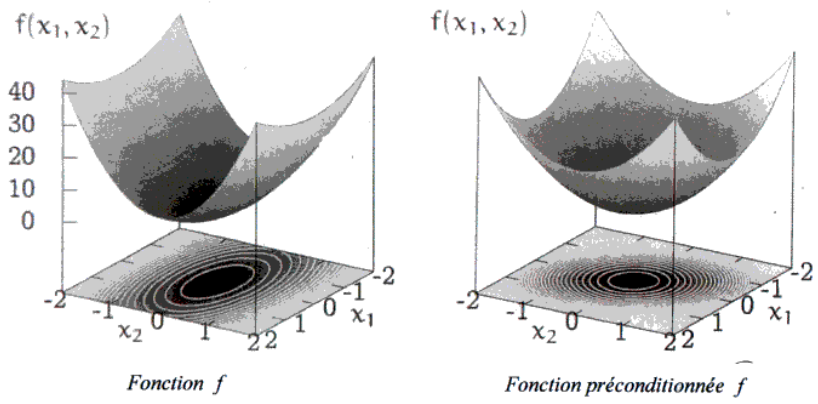
The *preconditionning* should have make the condition number better, as close as possible to 1. Let's check this.

The Hessian of $\tilde{f}$ is $\nabla^2 f = \begin{pmatrix} \partial_{11} & \partial_{12} \\ \partial_{21} & \partial_{22} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ which is a diagonal matrix as well and hence $cond(\tilde{f}) = \frac{\lambda_{max}}{\lambda_{min}} = \frac{1}{1} = 1$

A *condition number* value of 1 means the *countour curves* are perfect concentric circles, i.e. we have achieved the best possible preconditionning.

4.2.6 Contour lines

Below is a visualization of the countour curves for both functions:



Fonction f *Fonction préconditionnée $\tilde{f}$*

4.3 Practice

4.3.1 Exercise 11: preconditionning and variable change

Let $f(x)$ be an *ill-conditionned* function and $\tilde{f}(\tilde{x})$ be its best linear *preconditionning* obtained throug the variable change $\tilde{x} = Mx$.

Let's assume we know M as $M = \begin{pmatrix} 2 & -1 \\ 0 & 3 \end{pmatrix}$

What is $argmin\ f(x)$ if $argmin\ \tilde{f}(\tilde{x}) = \tilde{x}^*$?

4.3.2 Reverting variable change

If $\tilde{x} = Mx$ then $x = M^{-1}\tilde{x}$. Hence we need to compute the inverse of the matrix M

4.3.3 Inverting the matrix

Let's compute the inverse of the matrix M. There are several ways, for instance using the formula presented in 3.2.2 or a simple equations system:

$$M \cdot M^{-1} = I \Leftrightarrow \begin{pmatrix} 2 & -1 \\ 0 & 3 \end{pmatrix} \cdot \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\Leftrightarrow \begin{cases} 2a - c = 1 \\ 3c = 0 \\ 2b - d = 0 \\ 3d = 1 \end{cases} \Leftrightarrow \begin{cases} 2a - 0 = 1 \\ c = 0 \\ 2b - \frac{1}{3} = 0 \\ d = \frac{1}{3} \end{cases} \Leftrightarrow \begin{cases} a = \frac{1}{2} \\ c = 0 \\ b = \frac{1}{6} \\ d = \frac{1}{3} \end{cases}$$

Hence $M^{-1} = \begin{pmatrix} \frac{1}{2} & \frac{1}{6} \\ 0 & \frac{1}{3} \end{pmatrix}$

4.3.4 Computing x^*

So $x^* = M^{-1}\tilde{x}^* = \begin{pmatrix} \frac{1}{2} & \frac{1}{6} \\ 0 & \frac{1}{3} \end{pmatrix} \tilde{x}^*$

or $\begin{pmatrix} {x^*}_1 \\ {x^*}_2 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{6} \\ 0 & \frac{1}{3} \end{pmatrix} \cdot \begin{pmatrix} \tilde{x}_1^* \\ \tilde{x}_2^* \end{pmatrix} \Leftrightarrow \begin{pmatrix} {x^*}_1 \\ {x^*}_2 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\tilde{x^*}_1 + \frac{1}{6}\tilde{x}_2^* \\ \frac{1}{3}\tilde{x}_2^* \end{pmatrix}$

4.3.5 Exercise 12: preconditionning and variable change

4.3.5.1 Part I: preconditioning

Precondition function $f(x) = \frac{1}{2}{x_1}^2 + \frac{25}{2}{x_2}^2 + 3x_1x_2 - 12x_1 - \sqrt{\pi}x_2 - 6$

Indications: $\begin{pmatrix} 1 & 0 \\ 3 & 4 \end{pmatrix}$ could be useful

Compute Hessian:

Computing the secondary derivatives in mind, we get the following Hessian matrix:

$$\nabla^2 f = \begin{pmatrix} \partial_{11} & \partial_{12} \\ \partial_{21} & \partial_{22} \end{pmatrix} = \begin{pmatrix} 1 & 3 \\ 3 & 25 \end{pmatrix}$$

Cholesky Decomposition

Let's try the Cholesky decomposition with the matrix provided in the indications:

$$\nabla^2 f = L \cdot L^t = \begin{pmatrix} 1 & 3 \\ 3 & 25 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} 1 & 3 \\ 0 & 4 \end{pmatrix} \Rightarrow \text{OK}$$

Variable Change

We want to express $\tilde{x}$ in function of x and x in function of $\tilde{x}$

$$\tilde{x} = L^t \cdot x \Leftrightarrow \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} = \begin{pmatrix} 1 & 3 \\ 0 & 4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 + 3x_2 \\ 4x_2 \end{pmatrix}$$
$$\Leftrightarrow \begin{cases} x_1 = \tilde{x}_1 - 3x_2 = \tilde{x}_1 - \frac{3}{4}\tilde{x}_2 \\ x_2 = \frac{\tilde{x}_2}{4} \end{cases}$$

Compute function $\tilde{f}$

We can now inject these values in the original $f(x)$ function:

$$\begin{aligned} f(x_1, x_2) &= \frac{1}{2}{x_1}^2 + \frac{25}{2}{x_2}^2 + 3x_1x_2 - 12x_1 - \sqrt{\pi}x_2 - 6 \Leftrightarrow \\ \tilde{f}(\tilde{x}_1, \tilde{x}_2) &= \frac{1}{2}(\tilde{x}_1 - \frac{3}{4}\tilde{x}_2)^2 + \frac{25}{2}(\frac{1}{4}\tilde{x}_2)^2 + 3(\tilde{x}_1 - \frac{3}{4}\tilde{x}_2)(\frac{1}{4}\tilde{x}_2) - 12(\tilde{x}_1 - \frac{3}{4}\tilde{x}_2) - \sqrt{\pi}(\frac{1}{4}\tilde{x}_2) - 6 \\ &= \frac{1}{2}\tilde{x}_1^2 - \frac{3}{4}\tilde{x}_1\tilde{x}_2 + \frac{9}{32}\tilde{x}_2^2 + \frac{25}{32}\tilde{x}_2^2 + \frac{3}{4}\tilde{x}_1\tilde{x}_2 - \frac{9}{16}\tilde{x}_2^2 - 12\tilde{x}_1 + 9\tilde{x}_2 - \frac{\sqrt{\pi}}{4}\tilde{x}_2 - 6 \\ &= \frac{1}{2}\tilde{x}_1^2 + \frac{1}{2}\tilde{x}_2^2 - 12\tilde{x}_1 + (9 - \frac{\sqrt{\pi}}{4})\tilde{x}_2 - 6 \end{aligned}$$

4.3.5.2 Part II: check condition number

Condition number of $f(x)$

The condition number of function $f(x)$ is the condition number of its Hessian matrix $\nabla^2 f$, i.e. the ration between its highest eigenvalue and its smallest eigenvalue.

From the Hessian

$$\nabla^2 f = \begin{pmatrix} \partial_{11} & \partial_{12} \\ \partial_{21} & \partial_{22} \end{pmatrix} = \begin{pmatrix} 1 & 3 \\ 3 & 25 \end{pmatrix}$$

According to 3.4.1, we need to resolve the *characteristic polynomial* of A :
$P_A(\lambda) = \lambda^2 - \lambda \cdot trace(a) + |A|$.

With:

- $|A| = \det(A) = a_{11}a_{22} - a_{12}a_{21} = 25 - 9 = 16$
- $trace(A) = a_{11} + a_{22} = 1 + 25 = 26$

Hence $P_A(\lambda) = 1\lambda^2 - 26\lambda + 16$, using Viet with $a = 1$, $b = -26$, $c = 16$:

$$\lambda_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{26 \pm \sqrt{676 + 64}}{2} = \frac{26 \pm 24.74}{2} = \begin{cases} 25.37 \\ 0.63 \end{cases}$$

The condition number is thus $cond(f) = \frac{max(\lambda_1,\lambda_2)}{min(\lambda_1,\lambda_2)} = \frac{25.37}{0.63} = \mathbf{40.22}$

Condition number of $\tilde{f}(\tilde{x})$

We first need to compute the Hessian of function $\tilde{f}(\tilde{x})$:

$$\nabla^2 \tilde{f} == \begin{pmatrix} \partial_{11} & \partial_{12} \\ \partial_{21} & \partial_{22} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

There is not need to go any further as the Hessian of $\tilde{f}(\tilde{x})$ is a diagonal matrix. In diagonal matrices, the eigenvalues are readable on the diagonal.
Here the values of $\lambda_{1,2} = 1$ imply a **condition number of 1** which is optimal.

4.3.5.3 Part III: countour curves

What can you say about the countour curves of the preconditionned $\tilde{f}(\tilde{x})$ function ?

The condition number being one, the countour curves are perfect concentric circles and a naive descent algorithm (see further chapters) would converge to the optimal solution in one single iteration.

CHAPTER 5

Stopping criterion / Optimality condition

Contents

Before looking in the next chapters on algorithms aimed at identifying solutions to an optimisation problem, we need to be able to decide whether a given point is optimal or not. This ability relies on checking **optimality conditions**

Optimality conditions play three essential roles in the development of algorithms:

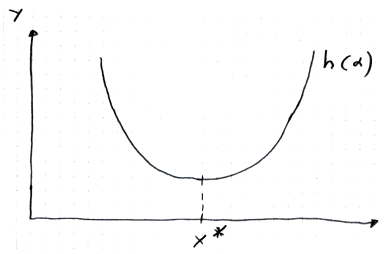
- They provide a theoretical analysis of the problem
- They inspire directly the ideas used to develop algorithms
- More importantly, **they provide a criteria for the halt of the iterative algorithm**

5.1 Introduction

5.1.1 1-dimension

In 1D, the optimility condition verification is quite easy. The lowest point on the curve is the place where the derivative (i.e. the slope) is null (0).

$$x^* = argmin\ f \Leftrightarrow f'(x^*) = 0$$



5.1.2 2-dimensions or more

In 2D, unfortunately, the optimality condition verification is not as easy as in 1D.

Reminder: When we use a derivative in 1D, we most offten use the gradient vector for the same purpose when facing more dimensions.

Hence, we would be tempted to use the same principle than in 1D, and limit ourselves to check that the gradient vector is the null vector as an optimality condition.
This, however doesn't work:

5.1.2.1 Counter-example

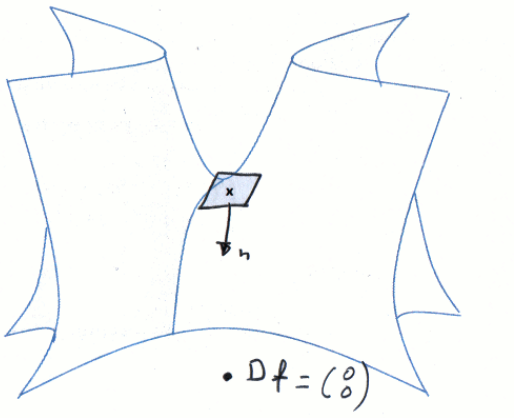
The function $f(x_1, x_2) = {x_1}^2 - {x_2}^2$ represents a *mountain pass* or a *saddle*.

Let's have a closer look at point $x = (0, 0)$.

First we need the derivatives:

$$\partial_1 = 2x_1 \quad \partial_{11} = 2$$
$$\partial_2 = 2x_2 \quad \partial_{22} = -2$$

Hence, at point $x = (0, 0)$, the gradient $\nabla f(x_1, f_2) = \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix} = \begin{pmatrix} 2 \cdot 0 \\ 2 \cdot 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$
However, as one can sees, it is obviously not a minimum nor a maximum.



Conclusion: the technique of checking only the derivative (or here the gradient) doesn't work in higher dimension

The theorem we will be seing in the next section states that a minimum (or maximum) has a **positive-definite Hessian matrix**. The hessian at point $x = (0, 0)$ is $\nabla^2 f(x_1, x_2) = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}$ which has 2 and -2 as eigenvalues (diagonal values on diagonal matrix) which are not all strictly positives and hence the hessian $\nabla^2 f(x_1, x_2)$ is not positive-definite.

5.2 Optimality condition

5.2.1 Theorem: necessary condition

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function in C^2.

$$x^* \text{is a local minimum of f} \Rightarrow \begin{cases} \nabla f(x^*) = 0 \\ \nabla^2 f(x^*) \text{is } \textbf{positive-semidefinite} \end{cases}$$

One should not that this condition is **necessary** but **not sufficient**. The reciprocity is not true !

5.2.2 Theorem: sufficient condition

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function in C^2.

$$\left.\begin{array}{l} \nabla f(x^*) = 0 \\ \nabla^2 f(x^*) \text{is } \textbf{positive-definite} \end{array}\right\} \Rightarrow x^* \text{is a local minimum of f}$$

Reminder: $\nabla^2 f(x)$ is **positive-definite** when each and every eigenvalue are strictly positive.

CHAPTER 6

Differential Optimisation

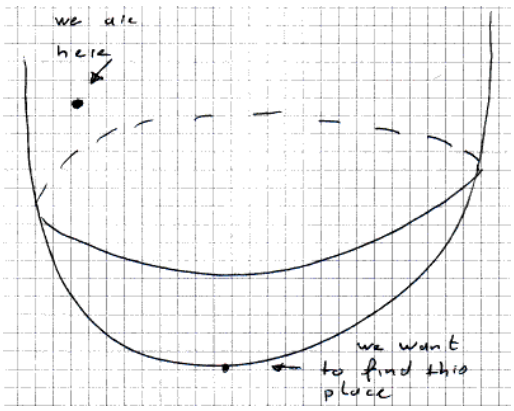
Contents

6.1 Introduction

Let's first think of an idea enabling us to find a *minima* on a surface.

Idea for an optimisation algorithm

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a C^1 function for whom we are looking for the *optimum without constraints*.



Idea: Look for the direction where the slope is the *steepest descend* $\Rightarrow$ follow the anti-gradient direction.

A **naive algorithm** could be :

> Follow the anti-gradient direction until we start to rise. At this exact location, recompute the anti-gradient and start all over again.

$\rightarrow$ which gives us a simple algorithm, yet very *inneficient*.

The intuition that comes immediately consists in following the *steepest slope* direction, given by the anti-gradient. This actually works but is desperately slow, especially for *ill-conditioned* functions.

Note

One should note that *whenever the countour curves are* ***perfect concentric circles***, the naive algorithm finds the optimum in 1 single iteration.
Hence the interest we have in good conditionned functions, as we have seen in chapter 4.

6.2 Principle

As we are looking for going down, we will attempt to generate a series of iterations $(x_k)_k$ such that

$$f(x_{k+1}) \leq f(x_k) \qquad k = 1, 2, ...$$

This is an iterative approach. We start with x_0, then the algorithm provides x_1, x_2, etc. that converge towards x^* = argmin / argmax $f(x)$.

Each iteration should find a *descending direction*, i.e. a direction d such that

$$\boxed{\partial_d f(x) = d^t \nabla f(x_k) < 0}$$

Reminder: $d^t \nabla f(x_k) < 0$ is a formula giving the *directional derivative* $\partial_d f(x)$ [1]. Whenever moving in a direction d implies a descent, the directional derivative in that direction is < 0.

We will focus within this document on minimization problems. One should not however that this is absolutely not a restriction since $min\ f(x) = -max\ -f(x)$ and $argmin\ f(x) = argmax\ f(x)$.

[1] Oen way to compute the *directional derivative* is actally to compute the scalar (or matricial) product between the gradient vector and the direction vector

6.2.1 Descent method

Such type of method is called **descent method**. It consists in a series of iterations based on three steps:

(E1) Find a **direction** $\mathbf{d_k} = d_k(x_k)$ such that ${d_k}^t \nabla f(x_k) < 0$

(E2) Find a **step** $\alpha_\mathbf{k} = \alpha_k(x_k)$ such that $f(x_{k+1}) = f(x_k + \alpha_k d_k) < f(x_k)$

(E3) Compute $x_{k+1} = x_k + \underbrace{\alpha_k(x_k) d_k(x_k)}$

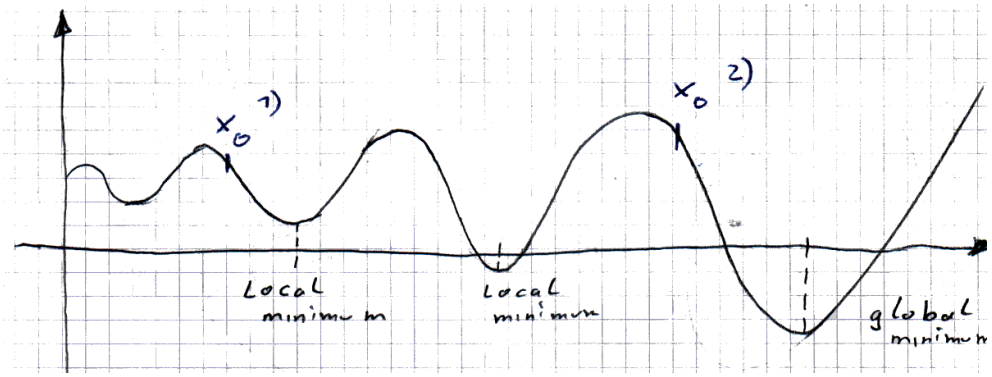
$\alpha_k \cdot d_k \rightarrow$ follow direction d_k with a ponderation α_k

Note: the $alpha$ parameter answers the question "how big should be each step in an iteration ?". (Later in this document, for the sake of a shorted notation, $\alpha_k(x_k)$ might be simply denoted α_k)

6.2.2 Local algorithms

This method is called a **local method**. The starting point x_0 plays an essential role in order to ensure a quick convergence towards the searched minimum value (maybe local).

We will never attempt to search a starting point x_0 by ourselves. This is a complicated matter which is usually left to the business to decide. And this is why:



point, we can only a find a *local minimum*.
On the other hand, of we choose $2)$ as the start point, we can find the *global minimum*.

These algorithms are called **local algorithms**. They rely heavily on a clever choice for the start point.

6.3 Steepest slope method

The *very first idea* that comes in mind when attempting to find a concrete descent method is to **follow the direction of the steepest slope**, i.e. the *opposite* direction of the gradient vector, the **anti-gradient**.

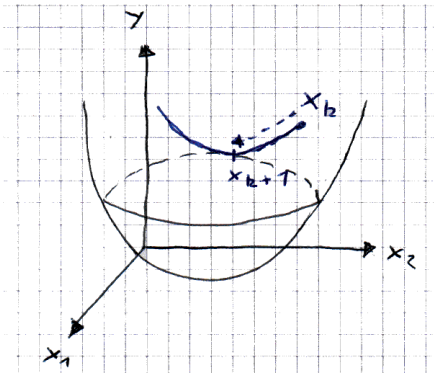
6.3.1 Iteration

This gives us the following iteration in the above method:

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

6.3.2 The step α

The α step is dynamic, i.e. it is recomputed (and will likely change) at each iteration, hence the notation $\alpha_k(x_k)$. One should not that this value is not straightforward to compute.



In the $d_k = -\nabla f(x_k)$ direction, should we always follow that direction, we are facing a 1D (2D) curve. This curve has a minimal value and a function $h(x)$. It is represented in dark blue on the schema on the left

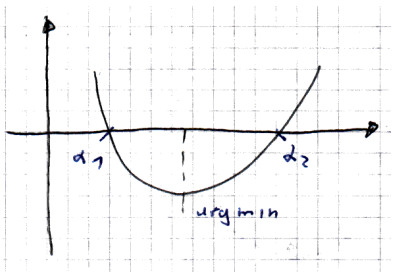
We want an α_k that makes us stop right at the lowest possible point in the d_k direction, which is the one that diminish the most the function $h(x)$. The function $h(x)$ is defined as $h(\alpha) = f(x_k + \alpha_k d_k(x_k))$.

$$\begin{aligned}\alpha_k = \underset{\alpha>0,\alpha\in\mathbb{R}}{\arg\min}\, h(\alpha) &= \underset{\alpha>0,\alpha\in\mathbb{R}}{\arg\min}\, f(x_k + \alpha_k(x_k)d_k(x_k)) \\ &= \underset{\alpha>0,\alpha\in\mathbb{R}}{\arg\min}\, f(x_k - \alpha_k(x_k)\nabla f(x_k))\end{aligned}$$

Finding α_k is a new optimisation problem that we hope might be solved analyticaly. One should note that this new optimisation problem is unidimensional.
This issue is further discussed in section 6.5.

6.3.3 Recall on parabolas

At first, let's assume $h(\alpha)$ is a parabola, i.e. a second-degree function. (One should note that this is rarely the case in practice).



Hence $h(\alpha) = a\alpha^2 + b\alpha + c = a(\alpha - \alpha_1)(\alpha - \alpha_2)$

1. Hence $\arg\min(\alpha) = \frac{\alpha_1+\alpha_2}{2}$
2. Another option consists in using the derivative.
   The derivative $h'(\alpha) = 2a\alpha + b$
   In argmin, $h'(\arg\min) = 0 \Leftrightarrow \arg\min = -\frac{b}{2a}$

6.3.4 Example

Let $f(x)$ be

$$f(x_1, x_2)) = \frac{1}{2}{x_1}^2 + \frac{9}{2}{x_2}^2$$

It's steepest slope is

$$d_k(x_k) = -\nabla f(x_k) = -\begin{pmatrix} (x_k)_1 \\ 9(x_k)_2 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$$

In order to find the length of the step α_k, we need to solve the 1D optimisation problem:

$$\alpha_k = \arg\min_{\alpha>0} h(\alpha) = \arg\min_{\alpha>0} f(x_k - \alpha\nabla f(x_k))$$

The variable of this optimisation problem is the scalar $\alpha \in \mathbb{R}$ and $h(\alpha)$ is the undimensional function:

$$\begin{aligned} h(\alpha) &= f(x_k - \alpha\nabla f(x_k)) \\ &= \frac{1}{2}{x_1}^2 + \frac{9}{2}{x_2}^2 \qquad \text{using } z_1 \text{ and } z_2 \\ &= \frac{1}{2}((x_k)_1 - \alpha(x_k)_1)^2 + \frac{9}{2}((x_k)_2 - 9\alpha(x_k)_2)^2 \end{aligned}$$

which is a second degree polynomial, hence a parabola. Finding its vertex is easy (using x instead of x_k for simplyfing the notation):

$$\begin{aligned} h(\alpha) &= \frac{1}{2}(x_1 - \alpha x_1)^2 + \frac{9}{2}(x_2 - 9\alpha x_2)^2 \\ &= \frac{1}{2}({x_1}^2 + (\alpha x_1)^2 - 2\alpha {x_1}^2) + \frac{9}{2}({x_1}^2 + (9\alpha x_2)^2 - 2\cdot 9\alpha {x_2}^2) \\ &= \frac{{x_1}^2}{2} + {x_1}^2\alpha^2 - {x_1}^2\alpha + \frac{9}{2}{x_2}^2 + \frac{729}{2}{x_2}^2\alpha\ 2 - 81{x_2}^2\alpha \\ &= \underbrace{(\frac{{x_1}^2}{2} + \frac{729}{2}{x_2}^2)}_{a}\alpha^2 \underbrace{-({x_1}^2 + 81{x_2}^2)}_{b}\alpha + \underbrace{(\frac{{x_1}^2}{2} + \frac{9}{2}{x_2}^2)}_{c} \end{aligned}$$

Hence $\arg\min h(\alpha)$ becomes easy to find:

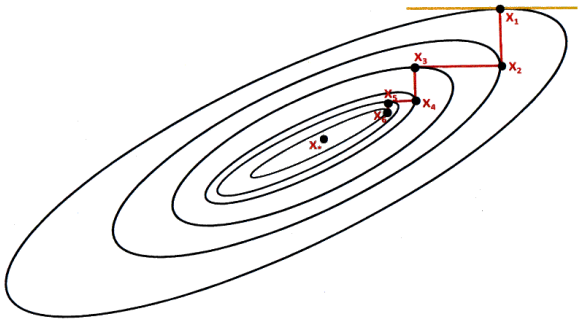
$$\alpha_k(x_k) = \arg\min_{\alpha>0} h(\alpha) = -\frac{b}{2a} = \frac{{(x_k)_1}^2 + 81{(x_k)_2}\ 2}{{(x_k)_1}^2 + 729{(x_k)_2}^2} > 0$$

In conclusion, the iterations become:

$$\begin{aligned} x_{k+1} &= x_k - \alpha_k(x_k)\nabla f(x_k) \\ &= x_k - \frac{{(x_k)_1}^2 + 81{(x_k)_2}\ 2}{{(x_k)_1}^2 + 729{(x_k)_2}^2}\begin{pmatrix} (x_k)_1 \\ 9(x_k)_2 \end{pmatrix} \end{aligned}$$

6.4 Limitations of the steepest slope method

In theory, the method seems interesting. In practive, however, it often implies teh necessity of an overwhelimg amount of iterations before it reached convergence, specifically when the function is *ill-conditionned*.



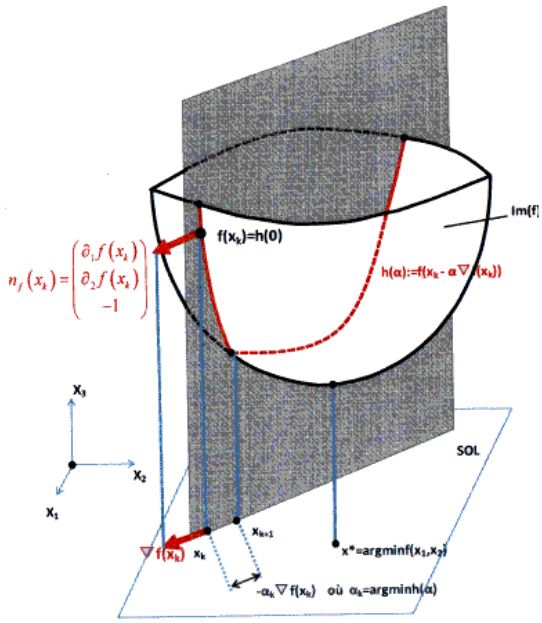
In this case, the series of x_k zigzag a lot and the convergence is very slow.

See for instance graphic on the left.

The slowness of the method is here due to the ill-conditionning of f.
With a good conditionning, the convergence would happen within one single iteration.
See exercise **TODO**

6.5 Estimating the *ideal* step α

Principle: one cuts the surface following the plan in black below in order to find the curve in red. That curve is called $h(\alpha)$ and is a *specific set of points* of the surface, i.e. it can be expressed from the surface.



The search of the ideal step α_k resumes to the resolution of a *new 1D minimsation problem*:

$$\alpha_k = \underset{\alpha>0, \alpha\in\mathbb{R}}{\arg\min} f(x_k + \alpha d_k)$$

Let's say

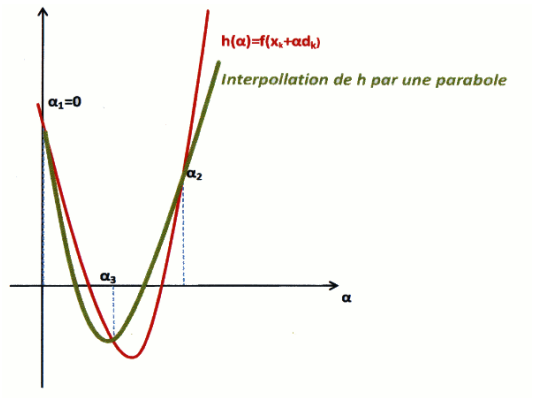
$$h(\alpha) = f(x_k + \alpha d_k)$$

is the unidimensionnal function to be minimized. Unfortunately $h(x)$ does almost never posess an explicite analytic solution, or even an easy to compute solution. In practice it is never a parabola.

What we do in this case is that we build an approximation of $h(x)$ using a parabola. The minimum of the parabola is analyticaly computable and might well be a quite good estimation for our step α_k.

6.5.1 The parabola algorithm

The idea is to approximate the $h(\alpha)$ function using a parabola. We know the equation of the parabola has the form $h(\alpha) = a\alpha^2 + b\alpha + c$.



In order to estimate the coefficients a, b and c, we need to know **three points** representatives of the parabola. In order to find these 3 *unknown* coefficients, we need 3 equations. Building these 3 equations is easy, all we need to do is use the function well known $h(\alpha)$ function and inject in 3 arbitrary values in it.

Hence we need to take 3 distinct values for the α parameter: α_1, α_3 and α_3:

- $\alpha_1 = 0$ (always)
- $\alpha_2 = ?$ $\rightarrow$ needs to be chosen
- $\alpha_3 = \frac{\alpha_2}{2}$ (implied by the choice of α_2)

6.5.1.1 Computing the coefficients

We need to compute the coefficients a, b and c of the parabola with the α_1, α_2 and α_3 parameters. By injecting the α_i coefficients in the $h(\alpha) = f(x_k + \alpha d_k)$ function, we get a 3 unknown variables, 3 equations system:

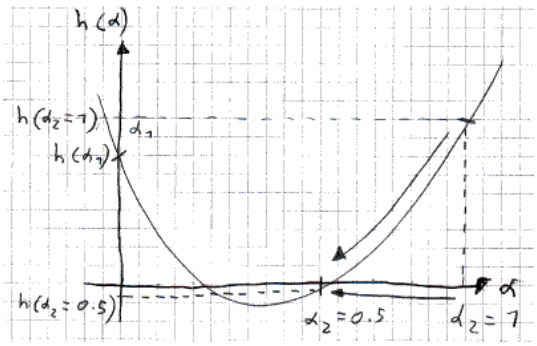
$$\begin{cases} a\alpha_1 + b\alpha_1 + c = h(\alpha_1) \\ a\alpha_2 + b\alpha_2 + c = h(\alpha_2) \\ a\alpha_3 + b\alpha_3 + c = h(\alpha_3) \end{cases} \Rightarrow \begin{cases} c = h(0) \\ a\alpha_2 + b\alpha_2 + h(0) = h(\alpha_2) \\ a\alpha_3 + b\alpha_3 + h(0) = h(\alpha_3) \end{cases}$$

Hence we're left with a 2 unknown variables, 2 equations system to be solved.

Once this system is solved, we have the a, b and c coefficients and we can compute the α parameter with $\alpha = \arg\min h(\alpha) = -\frac{b}{2a}$

6.5.1.2 Choosing the α_2 value

α_1 is always 0. α_3 is computed from α_2. α_2 needs to be chosen. We want ideally to have points as close as possible to *argmin* in order to end up with a good approximation of the interesting range.



So here's an algorithm for choosing α_2:

1. start with α_2 = 1
2. while $h(\alpha_2) > h(\alpha_1)$ do
   $\alpha_2 = \alpha_2/2$

6.6 Stop condition = optimality condition

The whole idea is to stop when we reach a point in the algorithm where we are not making very much progress anymore after each new iteration, i.e when x_{k+1} *is close to* x_k.
The problem here is that we need to define "*close to*". We use a value provided by the business for this, which is ε.

At first, we might be tempted to use as stopping condition $||x_{k+1} - x_k|| < \varepsilon_x$. But this suffers from a varying unit which makes it hardly usable.

Hence we'll use $\boxed{\frac{||x_{k+1} - x_k||}{||x_k||} < \varepsilon_x}$ in order to get rid of varying unit issues (i.e. get an absolute order).

6.7 Algorithm for the steepest descent

Inputs :

- (E1) A function $f : \mathbb{R}^2 \to \mathbb{R} \in C^1$, for which we are searching the *zero*
- (E2) A first approximation of the solution $x_0 \in \mathbb{R}^n$
- (E3) The asked precision $\varepsilon_x \geq 0, \varepsilon_f \geq 0$

Initializations :

- (O1) $k = 0$

Iterations :

- (I1) Compute $d_k = \nabla f(x_k)$
- (I2) Compute $a_k = \underset{\alpha>0, \alpha\in\mathbb{R}}{\arg\min} f(x_k + \alpha d_k)$
  (For instance by using a 2nd degree interpolation (parabola algorithm))
- (I3) Compute $x_{k+1} = x_k + \alpha_k d_k$
- (I4) k = k + 1

Stop :

- (A1) $\frac{||x_{k+1}-x_k||}{||x_k||} < \varepsilon_x$
- (A2) $|f(x_{k+1})| \leq \varepsilon_f$

Output :

- (O1) $x^* = x_{k+1}$ is an approximation of the zero of f

The convergence is guaranteed but slow !

6.7.1 Performance Optimizations

- (O1) There is a generic version of the steepest slope method which includes a generic pre-conditionning, with different variations.
- (O2) The search of the optimal *step length* can be very costy. Several methods exist to speed up the search of the ideal step length (See *Insearch* or *linear search*).

6.8 Practice

6.8.1 Exercise 13 : steepest slope descent

Precondition $f(x)$ defined as:

$$f(x_1, x_2) = \frac{1}{2}{x_1}^2 + \frac{9}{2}{x_2}^2$$

And show that the *steepest slope descent* algorithm coverge within one single iteration. Indication: play the algorithm here, no subtle deduction !

The start point is $x_0 = (9 \;\; 1)^t$

6.8.1.1 Preconditionning

Compute derivatives

$$\partial_{11} = 1 \quad \partial_{12} = \partial_{21} = 0 \quad \partial_{22} = 9$$

Hessian and Cholesky

$$\text{Hessian } \nabla^2 f(x) = \begin{pmatrix} 1 & 0 \\ 0 & 9 \end{pmatrix} = LL^t = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}$$

Preconditionning

Variable change :

$$\begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \qquad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{3} \end{pmatrix} \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix}$$

Compute function $\tilde{f}$

$$\begin{aligned} f(x_1, x_2) &= \frac{1}{2}{x_1}^2 + \frac{9}{2}{x_2}^2 \text{ and } \begin{cases} x_1 = \tilde{x}_1 \\ x_2 = \frac{1}{3}\tilde{x}_2 \end{cases} \\ \Rightarrow \tilde{f}(\tilde{x}_1, \tilde{x}_2) &= \frac{1}{2}{\tilde{x}_1}^2 + \frac{9}{2}(\frac{1}{3}{\tilde{x}_2}^2) \\ \Rightarrow \tilde{f}(\tilde{x}_1, \tilde{x}_2) &= \frac{1}{2}{\tilde{x}_1}^2 + \frac{1}{2}{\tilde{x}_2}^2 \end{aligned}$$

Gradient and Hessian of $\tilde{f}$

$$\text{Gradient } \nabla\tilde{f}(x) = \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} = \tilde{x} \qquad \text{Hessian } \nabla^2\tilde{f}(x) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

6.8.1.2 Steepest slope algorithm

We can compute the formula for the iterations knowing the gradient of $\tilde{f}(\tilde{x})$:

$$\begin{aligned} \tilde{x}_{k+1} &= \tilde{x}_k + \alpha_k d_k(\tilde{x}_k) \\ &= \tilde{x}_k - \alpha_k \tilde{x}_k \qquad \text{with } d_k(\tilde{x}_k) = -\nabla\tilde{f}(x_k) = -\tilde{x}_k = \begin{pmatrix} (\tilde{x}_1)_k \\ (\tilde{x}_2)_k \end{pmatrix} \end{aligned}$$

The α_k parameter can be analytically computed in this case, i.e. there is no need to perform an interpollation.
(Note: recall α_k is dependent of x_k, it is normally noted $\alpha_k(x_k)$. Hence the result of the computation should be a function of x_k, or a constant in case the $alpha_k$ is alwayys the same whatever the current x_k position on the surface. This later situation usually indicated a completion of the algorithm in one step)

$$\alpha_k = \underset{\alpha>0, \alpha\in\mathbb{R}}{\arg\min}\, h(\alpha) = \underset{\alpha>0, \alpha\in\mathbb{R}}{\arg\min}\, \tilde{f}(\tilde{x}_k - \alpha\tilde{x}_k)$$

Let's resolve $\tilde{f}(\tilde{x}_k - \alpha\tilde{x}_k)$:

$$\begin{aligned} \tilde{f}(\tilde{x}_k - \alpha\tilde{x}_k) &= \frac{1}{2}[(\tilde{x}_1)_k - \alpha(\tilde{x}_1)_k]^2 + \frac{1}{2}[(\tilde{x}_2)_k - \alpha(\tilde{x}_2)_k]^2 \qquad \text{with } \tilde{f}(\tilde{x}) = \frac{1}{2}{\tilde{x}_1}^2 + \frac{1}{2}{\tilde{x}_2}^2 \\ &= \frac{1}{2}[{(\tilde{x}_1)_k}^2 - 2{(\tilde{x}_1)_k}^2\alpha + \alpha^2{(\tilde{x}_1)_k}^2] + \frac{1}{2}[{(\tilde{x}_2)_k}^2 - 2{(\tilde{x}_2)_k}^2\alpha + \alpha^2{(\tilde{x}_2)_k}^2] \\ &= \underbrace{\frac{1}{2}[{(\tilde{x}_1)_k}^2 + {(\tilde{x}_2)_k}^2]}_{a} \cdot\alpha^2 \underbrace{-[{(\tilde{x}_1)_k}^2 + {(\tilde{x}_1)_k}^2]}_{b} \cdot\alpha + \underbrace{\frac{1}{2}[{(\tilde{x}_1)_k}^2 + {(\tilde{x}_1)_k}^2]}_{c} \\ &= \qquad a \qquad\qquad b \qquad\qquad c \end{aligned}$$

Hence $\arg\min h(\alpha)$ becomes easy to find:

$$\begin{aligned}\alpha_k(x_k) &= \arg\min_{\alpha>0} h(\alpha) = -\frac{b}{2a} \\ &= \frac{[(\tilde{x}_1)_k{}^2 + (\tilde{x}_1)_k{}^2]}{2 \cdot \frac{1}{2}[(\tilde{x}_1)_k{}^2 + (\tilde{x}_2)_k{}^2]} \\ &= \frac{[(\tilde{x}_1)_k{}^2 + (\tilde{x}_1)_k{}^2]}{[(\tilde{x}_1)_k{}^2 + (\tilde{x}_1)_k{}^2]} \\ &= 1\end{aligned}$$

Running the algorithm

Now we have the formula for $\alpha_k(x_k)$ - here simply the constant 1 - we can now run the algorithm:

$$\begin{aligned}\tilde{x}_{k+1} &= \tilde{x}_k + \alpha_k(\tilde{x}_k)\ d_k(\tilde{x}_k) \Rightarrow \\ \tilde{x}_{k+1} &= \tilde{x}_k - \alpha_k \tilde{x}_k \Rightarrow \\ \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix}_1 &= \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix}_0 - \alpha \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix}_0 \\ &= \begin{pmatrix} 9 \\ 1 \end{pmatrix} - 1 \begin{pmatrix} 9 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ 0 \end{pmatrix}\end{aligned}$$

Getting back in the original base

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{3} \end{pmatrix} \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{3} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Result

We can check that the point $(0\ \ 0)^t$ is really the minimum of original the paraboloid.

Remarks

- The result of the $f(x_1, x_2)$ function cannot be negative, hence $f(0,0) = 0$ cleary is a minimum.
- The Hessian of the preconditionned function $\tilde{f}$ is strictly positive-definite at the origine (as in every other point) which confirms of the the optimality conditions.

6.8.2 Exercise 14 : interpolating the step length

Let function f be defined as:

$$f(x_1, x_2) = (x_1 - 2)^4 + (x_1 - 2x_2)^2$$

Without any preconditionning, apply one iteration of the *steepest slope descent method*. The length of the step should be computed using the parabola interpolation method.
Use $x_0 = (0\ \ 0)^t$ as start point.

6.8.2.1 First iteration

Compute the Gradient

$$\nabla f(x) = \begin{pmatrix} \partial_1 f \\ \partial_2 f \end{pmatrix} = \begin{pmatrix} 4(x_1-2)^3 + 2(x_1 - 2x_2) \\ -4(x_1 - 2x_2) \end{pmatrix}$$

Define the first iteration, x_1

$$x_1 = x_0 - \alpha_0 \nabla f(x_0) \text{with } \alpha_0 = \arg\min \underbrace{f(x_0 - \alpha \nabla f(x_0))}_{h(\alpha)}$$

Compute α_0**, i.e.** $\alpha_k(x_k)$ **for** $x_k = x_0 = (0 \;\; 0)^t$

Here we use a different approach. There no use to perform the full computation for $h(\alpha)$ since we only need to compute the parabola interpolation for the first iteration. Hence we'll directly inject the $x_0 = (0 \;\; 0)^t$ in the formula instead of keeping x_k

Resolve Gradient

$$\nabla f(x_0) = \nabla f((0 \;\; 0)^t) = \begin{pmatrix} 4(0-2)^3 + 2(0 - 2\cdot 0) \\ -4(0 - 2\cdot 0) \end{pmatrix} = \begin{pmatrix} -32 \\ 0 \end{pmatrix}$$

Compute $h(\alpha)$

$$\begin{aligned} h(\alpha = &= f(x_0 - \alpha \nabla f(x_0)) \\ &= f\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix} + \alpha \begin{pmatrix} 32 \\ 0 \end{pmatrix}\right) \\ &= f\left(\begin{pmatrix} 32\alpha \\ 0 \end{pmatrix}\right) \\ &= (32\alpha - 2)^4 + (32\alpha)^2 \end{aligned}$$

6.8.2.2 Parabola interpolation

Let's say $(\alpha_1 = 0, \alpha_2 = ?, \alpha_3 = \alpha_2/2)$ and search for α_2 as long as $h(\alpha_2) > h(\alpha_1)$, starting with $\alpha_2 = 1$.
We need $h(\alpha_1 = 0) = (-2)^4 = 16$.

$$\begin{aligned} h(\alpha_2 = 1) &= (32-2)^4 + (32)^2 = 811024 \quad &> h(\alpha_1 = 0) = 16 \\ h(\alpha_2 = \frac{1}{2}) &= (16-2)^4 + (16)^2 = 38672 \quad &> h(\alpha_1 = 0) = 16 \\ &... \\ h(\alpha_2 = \frac{1}{16}) &= (2-2)^4 + (2)^2 = 4 \quad &\leq h(\alpha_1 = 0) = 16 \end{aligned}$$

Eventually the triple $(\alpha_1 = 0, \alpha_2 = \frac{1}{16}, \alpha_3 = \frac{1}{32})$ defines the parabola and we get the following system:

$$\begin{cases} a{\alpha_1}^2 + b\alpha_1 + c = h(\alpha_1) \\ a{\alpha_2}^2 + b\alpha_2 + c = h(\alpha_2) \\ a{\alpha_3}^2 + b\alpha_3 + c = h(\alpha_3) \end{cases} \Rightarrow \begin{cases} c = 16 \\ a(\frac{1}{16})^2 + b\frac{1}{16} + 16 = 4 \quad \cdot 16^2 \\ a(\frac{1}{32})^2 + b\frac{1}{32} + 16 = 2 \quad \cdot 32^2 \end{cases}$$

$$\Rightarrow \begin{cases} c = 16 \\ a + 16b + 16^3 = 4 \cdot 16^2 \\ a + 32b + 16 \cdot 32^2 = 2 \cdot 32^2 \end{cases}$$

$$\Rightarrow \begin{cases} c = 16 \\ a + 16b = (4 - 16) \cdot 16^2 \\ a + 32b = (2 - 16) \cdot 32^2 \end{cases}$$

$$\Rightarrow \ldots \Rightarrow \begin{cases} c = 16 \\ a = 8192 \\ b = -704 \end{cases}$$

Interpolate α_0

Our parabol hence is $p(\alpha) = 8192\alpha^2 - 704\alpha + 16$.

We want $\alpha_0 = \arg\min p(\alpha) = -\frac{b}{2a} = \frac{704}{2 \cdot 8192} = \frac{11}{256}$

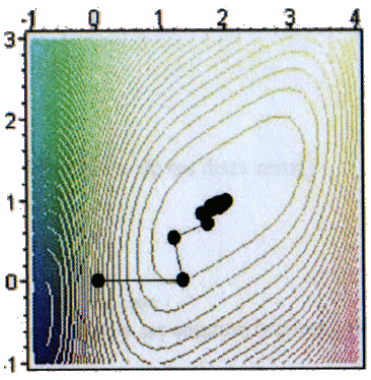
6.8.3 Compute first iteration

Finally:

$$x_1 = x_0 - \alpha_0 \nabla f(x_0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} - \frac{11}{256} \begin{pmatrix} -32 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{11}{8} \\ 0 \end{pmatrix} = \begin{pmatrix} 1.375 \\ 0 \end{pmatrix}$$

6.8.4 What if we keep going on ?

k	xk
0	(0.000 , 0.000)
1	(1.375 , 0.000)
2	(1.204 , 0.532)
3	(1.754 , 0.710)
4	(1.694 , 0.842)
5	(1.825 , 0.901)
6	(1.822 , 0.910)
7	(1.848 , 0.917)
8	(1.847 , 0.923)
9	(1.864 , 0.927)
10	(1.864 , 0.931)
11	(1.877 , 0.935)
12	(1.876 , 0.938)
13	(1.886 , 0.940)



CHAPTER 7

Solving nonlinear systems - Newton

Contents

7.1 Introduction

We have seen in the previous chapters the *necessary optimality conditions* that enables on to find the *ciritcal points* which are good candidates to solve an optimisation problem. **That necessary condition consists in finding the points making the gradient null (=0)**. One should recall that in 1D we needed to make the derivative null. One should recall as well that this was a necessary but *not a sufficient condition*.

$$\arg\min f(x) = ??? \rightarrow f'(x) = 0(1D) \Rightarrow \text{good candidates}$$

Actually, making the gradient null consists in **finding the zeros of the gradient function**. In n dimensions, this means solving a system of n non-linear equations with n unknown variables (because the gradient has a different equation for each dimension).

This is a question of equivalent problems. Finding the good candidates for *argmin* resolves to finding the zeros of the gradient function, i.e. solving the non-linear system of equations of the gradient. We will use the *Newton algorithm* to find these points and then build a method -

called the *Newton method* for solving optimisation problem. The *Newton method* is presented in chapter 9.

This chapter focuses on presenting the *Newton algorithm* for solving non-linear equations systems

The **Newton algorithm** plays an essential role in the resolution of non-linear systems, and, by extension, in the field of non-linear optimisation. It enables to quickly find the neaarest point where the gradient is nullified (local optimum only).

We will study the *Newton algorithm* here as a way to sole non-linear equations and systems while the chapter 9 presents an *optimisation method* based on the *Newton algorithm*.

7.1.1 Principle

Recall $x^* = \arg\min f \Leftrightarrow \begin{cases} \nabla f(x^*) = 0 \text{ necessary condition } * \\ \nabla^2 f(x^*) \text{ is positive definite sufficient condition} \end{cases}$

$* \rightarrow$ use the *Newton algorithm* to get candidates and check them with the *Hessian*.

(This is furtherly developed in 9)

7.2 Newton in 1D

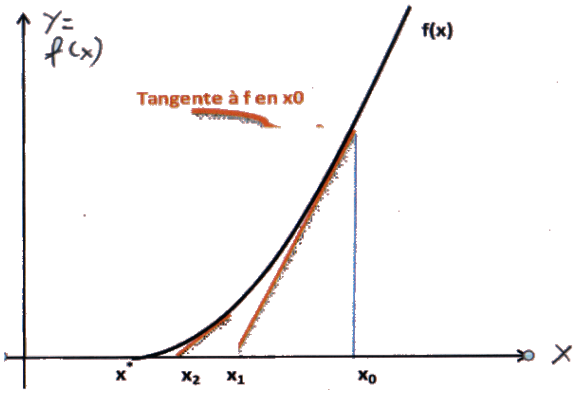
In 1D, we focus on a function of the form $f : \mathbb{R} \rightarrow \mathbb{R} \in C^1$ with **one single** variable.
The values of x which nullifies f (i.e. make f return 0) are called **zeros** or **roots** of f.

7.2.1 Graphical approach - 1D Newton

7.2.1.1 Principle

The *Newton algorithm* consists in making a series of iterations $(x_k)_k$ converging towards the *root* x^* of f.
The series is defined using the geometrical approach illustrated on the graph on the right.

We take the tangent (derivative) in x_0 and considere the point x_1 is the point where the tangent crosses the x axe.



The tangent at point x_0 forms a *rectangle triangle* with the two axes. The slope of the *longest side* is given

- either the derivative $f'(x_0)$

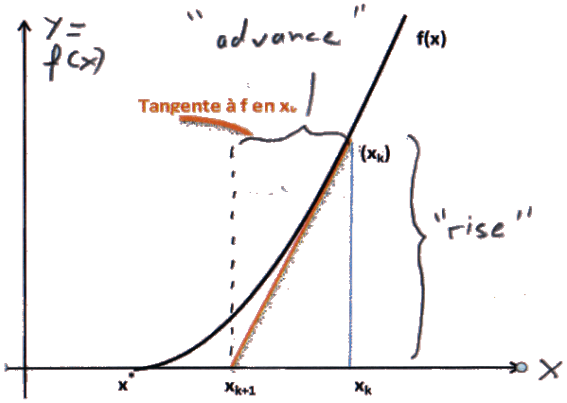
- or the pythagorean way $\frac{f(x_0)}{x_0 - x_1}$

7.2.1.2 The iterations

What we have seen for x_0 and x_1 above can be generalized for every iteration. Hence, the way an iteration can be defined is straightforward:

$$\textit{slope} = \frac{\textit{"rise"}}{\textit{"advance"}} = \frac{f(x_k)}{x_k - x_{k+1}} = f'(x_k)$$

$$\Leftrightarrow \boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}}$$



The iterative algorithm becomes:

$$\begin{cases} x_0 \text{ needs to be close from } x* \\ x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \end{cases}$$

Again, the algorithm is highly sensitive to a *very good choice* of the starting point x_0. If the chosen x_0 is to far from the solution, the algorithm migth even diverge.
On the other hand, if Newton converges, it is very fast and efficient. It's its great strength!

7.2.1.3 Stopping criterions

Two distinct stopping criterions are required, one on x and one on $y = f(x)$

on x, one stops when one doesn't progress on x anymore, i.e. when the iterations in x_k are close enough to eachothers, i.e. when:

$$\boxed{\frac{|x_k - x_{k+1}|}{|x_k|} \leq \varepsilon_x}$$

on y, one stops when one doesn't progress on y anymore, i.e. when the $f(x)$ is close enough to 0, i.e. when:

$$\boxed{|f(x_k)| \leq \varepsilon_x}$$

Note: the ratios for stopping on ε_x to avoid subscribing to problems with different scales.

7.2.2 Analytical approach - 1D Newton

We have seen the graphical approach in the previous section which already enabled us to build the iterations. The goal here is to see whether an analytical approach leads us to the same conclusions... which will turn out to be the case.
The reader might well skip this section is it doesn't brind anything additional regarding the way to use the *Newton algorithm in 1D*.

Globally, the approach is still iterative: we build a series of iterations (x_k) converging towards the searched *zero*.

But we use a *trick* here:

The idea is to approximate the function f for which we are searching the zero by a *linear function*, and then to solve the problem for the linear function. The zero of the linear function approximating f en x_k gives us the next iteration point x_{k+1}.
The algorithm stops when it doesn't progress anymore neither on x, nor on $y = f(x)$.

One should note that the more the function is not linear, the worst is the approximation of it through a linear function, hence the more the algorithm might diverge. It might even more diverge if the x_0 point is not a clever choice.

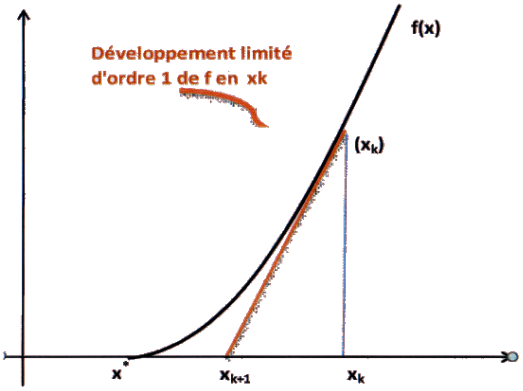
7.2.2.1 Mathematical reminder

Each and every *continued differentiable function* can be approximated locally on a point x_k by a polynomial of degree n. That polynomial comes from a *n-order Taylor series* (in french : *développement limité d'ordre n*) of the function f (the bigger the n, the better the approximation):

$$\begin{aligned} f_{x_k}(x) &\cong f(x_k) + \sum_{i=1}^{n} \frac{f^{(i)}(x_k)}{i!}(x - x_k)^i \\ &= f(x_k) + \frac{f'(x_k)}{1!}(x - x_k)^1 + ... + \frac{f^{(n)}(x_k)}{n!}(x - x_k)^n \end{aligned}$$

The *first-order Taylor series* (in french *développement limité d'ordre 1*) of f in x_k is:

$$f_{x_k}(x) \cong f(x_k) + f'(x_k)(x - x_k)$$



Let's call this one the **linear model** and write it:

$$\boxed{m_{(f,x_k)}(x) \cong f(x_k) + f'(x_k)(x - x_k)}$$

With this approach, x_{k+1} is nothing more than the *zero* or *root* of the linear model $m_{(f,x_k)}(x)$, i.e.:

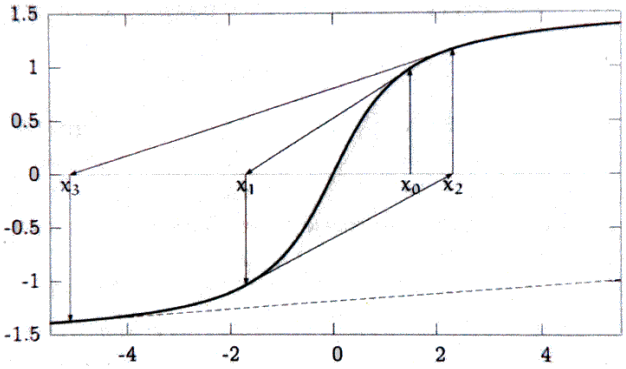
$$0 = f(x_k) + f'(x_k)(x - x_k) \Leftrightarrow x = x_k - \frac{f(x_k)}{f'(x_k)}$$

Hence we end up with exactly the same formula we have built using the *geometrical approach*. Happily ☺

7.2.3 Divergence - an example

When the function for which we are searching the zero is not enough linear, and when the chosen start point x_0 is a very poor choice, the *Newton algorithm* diverges.

The schema on the right illustrates such a situation with the $arctan\ (x)$ function.



7.3 Newton in nD

7.3.1 Purpose

As stated in the previous chapter, the ***Newton method*** **is essentialy useful for solving equations, i.e. finding roots or zeros**

In nD, we will use the *Newton algorithm* to solve *sytems of equations of several unknown variables*. The algorithm is a generalization of the *1D Newton algorithm* to multi-dimensionnal C^2 functions:

$$f : \mathbb{R}^n \to \mathbb{R}^m$$

$$(x_1, ..., x_n) \to f(x_1, ..., x_n) = \begin{pmatrix} f_1(x_1, ..., x_n) \\ ... \\ f_m(x_1, ..., x_n) \end{pmatrix}$$

One just needs to recall that the generalization of the derivative in the context of $f : \mathbb{R}^n \to \mathbb{R}^m$ functions is the **Jacobian matrix**. See 7.3.1.2.

The *Newton* algorithm enables one to find a zero of function f, i.e. to find the solution of the non-linear system:

$$(S) \begin{cases} f_1(x_1, ..., x_n) = 0 \\ ... \\ f_m(x_1, ..., x_n) = 0 \end{cases}$$

In order for the system not to be *underdeterminated* (too many unknown variables for not enough equations) or *overdeterminated*, we will assume:

$$\boxed{n = m}$$

7.3.1.1 The gradient matrix

Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a C^2 function continued and differentiable. The **Gradient matrix** is defined as:

$$\nabla f(x) = \begin{pmatrix} \uparrow & & \uparrow \\ \nabla f_1(x) & \dots & \nabla f_n(x) \\ \downarrow & & \downarrow \end{pmatrix} \in \mathbb{M}_{n\times m}(\mathbb{R})$$

We can calculate the gradients of each sub-function of the function f. These gradients are vectors, we can put them in *column* in the matrix.

7.3.1.2 The jacobian matrix

When one puts the gradient vectors in row instead of in column, one gets the jacobian matrix (in french : la matrice de Jacobi) :

$$J_f(x) = \nabla f(x)^t = \begin{pmatrix} \leftarrow & \nabla f_1(x)^t & \rightarrow \\ \dots & & \\ \leftarrow & \nabla f_m(x)^t & \rightarrow \end{pmatrix} \in \mathbb{M}_{m\times n}(\mathbb{R})$$

The jacobian matrix is the generalization of the derivative for multidimensional models. [1] It plays an essential role in what we will be seeing next in this chapter.

7.3.2 Geometrical approach - nD Newton

The geometrical approach we have been studying in the 1D model, can easily be generalized to the nD model.

- For a **function** $f : \mathbb{R} \to \mathbb{R}$ **unidimensional** in C^1, the geometrical approach led us to the following relation, which we can write under *its generalisable form*:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - f'(x_k)^{-1} f(x_k)$$

- For a **function** $f : \mathbb{R}^n \to \mathbb{R}^m$ **multidimensional** in C^2, one only needs to replace $f'(x_k)^{-1}$ by the inverse of the *jacobian matrix* ${J_f}^{-1}(x_k)$, which gives:

$$\boxed{x_{k+1} = x_k - {J_f}^{-1}(x_k) f(x_k)}$$

  In the case where $m = n$, the jacobian matrix is a *squared matrix* and hence its inverse makes sense!

[1] One should note that the specific case $f : \mathbb{R}^n \to \mathbb{R}$ leads to a jacobian matrix with one single, row, i.e. the gradient, which - as we have seen in the chapters before - is the generalization of the derivative in this case.

7.3.3 Analytical approach - nD Newton

Again, the goal here is to see whether an analytical approach leads us to the same conclusions... which will turn out to be the case. This section might well be skipped.

The principle is just the same as in 1D: instead of searching the zero of function $f : \mathbb{R}^n \to \mathbb{R}^m$, we search the zero of the *linear function* which is the closest to the original function, i.e. the *linar function* given by the *first-order Taylor series*.
For a multi-dimensionnal function, the linear model is:

$$m_{(f,x_k)} = f(x_k) + J_f(x_k)(x - x_k)$$

which is simply the linear model of an *unidimensionnal function* where the derivative has been replaced by the *jacobian function*.
The zero of this model is given by the equation below from which we extract the **iterations**:

$$\begin{aligned} 0 &= f(x_k) + J_f(x_k)(x - x_k) \\ \Leftrightarrow & \boxed{x_{k+1} = x_k - {J_f}^{-1}(x_k) f(x_k)} \end{aligned}$$

which is the same that what we have obtained with the graphical approach. Happily ☺

7.3.4 Newton's equation

The inverse of a matrix can be very costy to compute, hence one sometimes write the **iteration of the Newton algorithm** the way below.

This is called **the Newton equation**

$$\boxed{x_{k+1} = x_k + d_k} \text{ where } \boxed{J_f(x_k) d_k = -f(x_k)}$$

There are **numerical analysis technics** that helps find a solution d_k to this problem without the need to invert the *jacobian matrix*.

7.4 The *Newton* algorithm

Inputs :

- (E1) A function $f : \mathbb{R}^n \to \mathbb{R}^m \in C^2$ for which we search the zero
- (E2) The jacobian matrix $J_f(x_k)$
- (E3) A first approximation of the solution $x_0 \in \mathbb{R}^n$
- (E4) The asked precison $\varepsilon_x \geq 0$, $\varepsilon_f \geq 0$

Initializations :

- (O1) k = 0

Iterations :

- (I1) Compute $d_k = d_k(x_k)$, the solution of $J_f(x_k)d_k = -f(x_k)$
- (I2) $x_k + 1 = x_k + d_k$
- (I3) $k = k + 1$

Stop :

- (A1) $\frac{||x_{k+1}-x_k||}{||x_k||} < \varepsilon_x$
- (A2) $|f(x_{k+1})| \leq \varepsilon_f$

Output :

- (O1) $x^* = x_{k+1}$ is an approximation of the zero of f

The convergence is not guaranteed whenever the function is not enough linear or if the initial solution x_0 is too far from the solution. On the other hand, when the algorithm converges, it converges fast.

7.5 Practice

7.5.1 Exercise 15-a : from a system to the zero

(This exercise has a sequel 15-b under 8.6.1 in the next chapter)

Modelize under an appropriate form the following problem in order to be able to resolve it with the *Newton algorithm*.

$$(S) \begin{cases} {x_1}^2 - x_1x_2 + {x_2}^2 = 21 \\ {x_1}^2 + 2x_1x_2 - 8{x_2}^2 = 0 \end{cases}$$

Note: this system produces 4 solutions: $(-2\sqrt{7} \quad -\sqrt{7})^t$, $(2\sqrt{7} \quad \sqrt{7})^t$, $(-4 \quad 1)^t$ and $(4 \quad -1)^t$

In addition, mention the functions (*gradient*, *hessian*, etc.) that need to be computed in order to achieve its resolution.

7.5.1.1 Model

This system can be resolved using the *Newton algorithm* by searching for the zero of the function:

$$f : \mathbb{R}^2 \to \mathbb{R}^2$$
$$f(x_1, x_2) = \begin{pmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{pmatrix} = \begin{pmatrix} {x_1}^2 - x_1x_2 + {x_2}^2 - 21 \\ {x_1}^2 + 2x_1x_2 - 8{x_2}^2 \end{pmatrix}$$

7.5.1.2 Required functions

Newton requires the calculation of the *jacobian matrix*.

Let's first compute the individual gradients for each sub-function:

$$\nabla f_1(x_1, x_2) = \begin{pmatrix} \partial_1 f_1 \\ \partial_2 f_1 \end{pmatrix} = \begin{pmatrix} 2x_1 - x_2 \\ -x_1 + 2x_2 \end{pmatrix}$$
$$\nabla f_2(x_1, x_2) = \begin{pmatrix} \partial_1 f_2 \\ \partial_2 f_2 \end{pmatrix} = \begin{pmatrix} 2x_1 + 2x_2 \\ 2x_1 - 16x_2 \end{pmatrix}$$

which we inject in the *gradient matrix*:

$$\nabla f(x_1, x_2) = \begin{pmatrix} \uparrow & \uparrow \\ \nabla f_1 & \nabla f_2 \\ \downarrow & \downarrow \end{pmatrix} = \begin{pmatrix} 2x_1 - x_2 & 2x_1 + 2x_2 \\ -x_1 + 2x_2 & 2x_1 - 16x_2 \end{pmatrix}$$

which gives the jacobian matrix with its transpose:

$$J_f(x_1, x_2) = \nabla f^t(x_1, x_2) = \begin{pmatrix} \leftarrow & \nabla f_1 & \rightarrow \\ \leftarrow & \nabla f_2 & \rightarrow \end{pmatrix} = \begin{pmatrix} 2x_1 - x_2 & -x_1 + 2x_2 - x_1 + 2x_2 \\ 2x_1 + 2x_2 & 2x_1 - 16x_2 \end{pmatrix}$$

7.5.2 Exercise 17 : the *Newton algorithm*

Let (S) be the following system of equations:

$$(S) \begin{cases} (x_1 + 1)^2 + {x_2}^2 = 2 \\ e^{x_1} + {x_2}^3 = 2 \end{cases}$$

Solve it using the *Newton algorithm*, by using the start point $x_0 = (1 \ \ 1)^t$. **Only the first iteration should be executed**

7.5.2.1 Model

Let's use *Newton* to find a zero of the function:

$$f(x_1, x_2) = \begin{pmatrix} (x_1 + 1)^2 + {x_2}^2 - 2 \\ e^{x_1} + {x_2}^3 - 2 \end{pmatrix}$$

7.5.2.2 The *jacobian matrix*

First, let's compute the gradient of the sub-functions:

$$\nabla f_1(x_1, x_2) = \begin{pmatrix} 2(x_1 + 1) \\ 2x_2 \end{pmatrix}$$
$$\nabla f_2(x_1, x_2) = \begin{pmatrix} e^{x_1} \\ 3{x_2}^2 \end{pmatrix}$$

The *jacobian matrix* is built by putting these gardient in a row:

$$J_f(x_1,x_2) = \begin{pmatrix} \leftarrow & \nabla f_1(x_1,x_2)^t & \rightarrow \\ \leftarrow & \nabla f_2(x_1,x_2)^t & \rightarrow \end{pmatrix} = \begin{pmatrix} 2(x_1+1) & 2x_2 \\ e^{x_1} & 3{x_2}^2 \end{pmatrix}$$

7.5.2.3 First iteration

Let $x_0 = (1 \;\; 1)^t$ the start point, then

$$x_1 = x_0 + d_0$$

where $d_0 \in \mathbb{R}$ is the solution of the system:

$$J_f(x_0)d_0 = -f(x_0) \Rightarrow \boxed{d_0 = -J_f(x_0)f(x_0)}$$

7.5.2.4 Resolving the first iteration

We have:

$$J_f(x_0) = J_f(1,1) = \begin{pmatrix} 2(x_1+1) & 2x_2 \\ e^{x_1} & 3{x_2}^2 \end{pmatrix} = \begin{pmatrix} 4 & 2 \\ e & 3 \end{pmatrix}$$

$$f(x_0) = f(1,1) = \begin{pmatrix} (x_1+1)^2 + {x_2}^2 - 2 \\ e^{x_1} + {x_2}^3 - 1 \end{pmatrix} = \begin{pmatrix} 3 \\ e-1 \end{pmatrix}$$

The *jacobi matrix* being a 2×2 matrix, we can easily afford the cost of inverting it:

$${J_f}^{-1}(x_0) = \begin{pmatrix} 4 & 2 \\ e & 3 \end{pmatrix}^{-1} \qquad \text{with } A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \Leftrightarrow A\,{-1} = \frac{1}{det(A)} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

$$\text{and } det(A) = |A| = ad - bc$$

$$= 12 - 2e$$

$$= \frac{1}{12-2e} \begin{pmatrix} 3 & -2 \\ -e & 4 \end{pmatrix}$$

We can now compute d_0

$$\begin{aligned} d_0 &= -J_f(x_0)f(x_0) \\ &= -\frac{1}{12-2e} \begin{pmatrix} 3 & -2 \\ -e & 4 \end{pmatrix} \begin{pmatrix} 3 \\ e-1 \end{pmatrix} \\ &= -\frac{1}{12-2e} \begin{pmatrix} 9+2-2e \\ e-4 \end{pmatrix} \end{aligned}$$

The first iteration of *Newton* is:

$$\begin{aligned} x_1 &= x_0 + d_0 \\ &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} - \frac{1}{12-2e} \begin{pmatrix} 9+2-2e \\ e-4 \end{pmatrix} \\ &= \begin{pmatrix} 0.1523592 \\ 1.195282 \end{pmatrix} \end{aligned}$$

CHAPTER 8

Solving nonlinear systems - Quasi-Newton methods

Contents

8.1 Introduction

In the *Newton algorithm*, it is required to compute either the derivative (unidimensional function) or the Jacobi matrix (multidimensional function) of the function, which can very well be pretty costy in terms of computation time.
The *Quasi-Newton* methods are all attempts to address this issue with the same idea: *replace the tangent by an approximative linear model* expected to be easier to compute.

The three methods we will be seeing try to estimate the slope of the tangent, and replace it by a line expected to be close enough. For instance, the approximative linear model can be:

- A string
- An estimate of the derivative
- A secant

The *secant* method is particularly interesting as it is easily generalizable to multi-dimensionnal functions and, enhancent with the *Broyden* technic, it is one of the most powerful algorithm discovered so far.

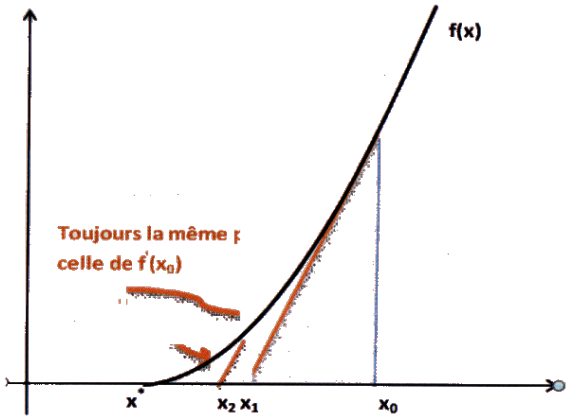
8.1.1 Principle

The *Quasi-Newton* methods all try to address the following issue in the algorithm iterations:

$$x_{k+1} = x_k - \frac{f(x)}{???} \leftarrow \text{put something easy here}$$

8.2 The *string method*

The *string method* is base on the geometrical principle illustrated on the schema on the right. It consists in derivating the target $f(x)$ function only once on x_0 and reusing every time time the same derivative.



The iterations become:

1D $\boxed{x_{k+1} = x_k - f'(x_0)^{-1} f(x_k)}$

nD $\boxed{x_{k+1} = x_k - {J_f}^{-1}(x_0) f(x_k)}$

- Advantage: less calculations
- Drawback: more iterations

8.3 Finite difference method

(In french: *méthode des différences finies*)

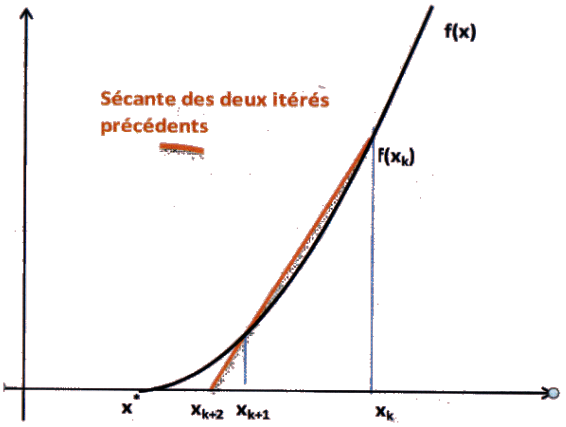
The *String method* eliminates the calculation of the derivative at each iteration, yet it is still required to compute the derivative at the start point x_0 which implies an *analytic knowledge* of the derivative.
With the Finite difference method, one computes an estimate of the derivative at each iteration, which supress the need to explicitly compute the derivative itself. This estimation is undoubtly better that with the *string method*

8.3.1 Idea : the secant principle

As the *Newton algorithm* provides as series of iterations (x_k) and $f(x_k)$, an estimation of the derivative could be

$$est(f'(x)) = \frac{f(x_k) - f(x_{k+1}}{x_k - x_{k+1}}$$

Geometricaly, this resolves to take as estimation of the derivative in x_{k+1} the *secant* connecting the two previous iterations $(x_k, f(x_k))$ and $(x_{k+1}, f(x_{k+1}))$.



With *Quasi-Newton-Secant*, the iterations becomes:

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

Note: one should note that this required to have two start pointx_0 and x_1. Both need to be sufficiently close to x^*. Sometimes, one uses *Newton* to get x_1 from x_0.

8.4 The Broyden method

The Broyden method consists in formalizing the idea of *the secant* presented above in the context of the *Newton algorithm* to multidimensionnal functions of the form $f : \mathbb{R}^n \to \mathbb{R}^m$.

8.4.1 The linear estimated model

The *method of Newton* is based on the approximation of the local f function for which we are searching a zero by the closest possible linear function.
That linear function is obtained by the *first order Taylor series* (french : *développement limité d'ordre 1*) of f which we call in this context the **linear model of f**.

- For a unidimensional function, the model is a line:

$$m_{(f,x_k)} = f(x_k) + f'(x_k)(x - x_k)$$

- For a multidimensional function, the model is an hyper-plan:

$$m_{(f,x_k)} = f(x_k) + J_f(x_k)(x - x_k)$$

- The *quasi-Newton* methods use an *estimated model* noted:

$$\boxed{\hat{m}_{(f,x_k)} = f(x_k) + A_{n\times n}(x_k)(x - x_k)}$$

where $A_{n\times n}$ is a

- a **squared matrix** in case f is multidimensional
- a scalar in case f is unidimensional

When A is replaced by the Jacobi matrix (multidimensional world) or the derivative (unidimensional) world, one gets the *exact* model

8.4.2 The *Quasi-Newton* equation

The *Newton algorithm*, just as the *Quasi-Newton algorithm*, searches the zero of the associated *linear model*:

$$f(x_k) + A_{n\times n}(x_k)(x - x_k) = 0 \Leftrightarrow x = x_k - {A_{n\times n}}^{-1}(x_k)f(x_k)$$

In practice, inverting the $A_{n\times n}(x_k)$, matrix is not very clever, on might rather write:

$$\boxed{x_{k+1} = x_k + d_k} \quad (I) \quad \text{where} \quad \boxed{A_{n\times n}(x_k)d_k = -f(x_k)} \quad (II)$$

This is called the **Quasi-Newton equation**. The quality and the efficiency of the convergence of the *quasi-Newton* algorithms will depend of the choice of the matrix $A_{n\times n}$

8.4.3 Multi-dimensional *secant*

We have seen that for a n unidimensional function, the *secant* method proposes to use the *slope of the secant of thje previous iteration* as $A_{n\times n}$:

$$A_{n\times n}(x_k) = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

The problem in a multidimensional world is that dividing by the $(x_k - x_{k-1})$ makes no sense! We prefere hence write:

$$\boxed{A_{n\times n}(x_k)(x_k - x_{k-1}) = f(x_k) - f(x_{k-1})} \quad (III)$$

8.4.4 Algorithm principle

Now one has first to resolve the linear system (II) and then to inject the result in in (II) and resolve this other linear system, and eventually inject the result in (I).
But be careful, the unknown variable which we are resolving in (III) is not d_{k-1} but the matrix $A_{n\times n}$ which posesses n^2 unknown values.
As the system only posesses n equations, **the system is largely under-determined**. Geometrically, thete is an infinite number of hyper-plans going through 2 points.
This is where Broyden kicks in. He proposes a subtle and simple way to compute the $A_{n\times n}$ matrix.

8.4.5 Broyden

Chosing amongst the infinity of solutions for the system (III) resolves to defining $A_{n\times n}$, i.e. define the estimated linear model:

$$\hat{m}_{(f,x_k)}(x) = f(x_k) + A_{n\times n}(x_k)(x - x_k)$$

The idea propose by Broyden (1965) is to choose amongst this infinity of secant linear models, the one that is the closest to the model established during the previous iteration. This way we keep as much as possible what has already been computed.
Broyden accepts a little sacrifice in terms of quality by not searching the very best approcimation in favor of a model requiering only few calculations. The focus is clearyl put on performances. It is today one of the most powerful algorithm and is widely used.

8.4.5.1 The Broyden theorem

The linear model estimated in x_k is:

$$\boxed{A_k = A_{k-1} + \frac{(y_{k-1} - A_{k-1}d_{k-1}){d^t}_{k-1}}{{d^t}_{k-1}d_{k-1}} \in GL_n(\mathbb{R})}$$

with:

- A_{k-1} is the *approximated model matrix*, i .e **the approximation of the derivative**, computed during the previous iteration.
  By default, A_0 is initialized at I.
- $y_{k-1} = f(x_k) - f(x_{k-1})$
- $d_{k-1} = x_k - x_{k-1}$
- $GL_n(\mathbb{R})$ the set of invertible squared matrices $n \times n$

8.4.5.2 Conclusion

It is possible to find efficiently a zero of a multidimensionnal function associated to a non-linear system of n-equations with n unknown values by using the *Quasi-Newton secant* algorithm associated to the *Broyden* secant model. The Broyden secant model is used to approximate the *Jacobian*.

8.5 Algorithm

Inputs :

- (E1) A function $f : \mathbb{R}^n \to \mathbb{R}^n \in C^2$ for which we search the zero
- (E2) A first approximation of the solution $x_0 \in \mathbb{R}^n$

- (E3) A first approximation of *Jacobian*, i.e a matrix A_0 (by default $A_0 = I_n$)
- (E4) The asked precison $\varepsilon_x \geq 0$, $\varepsilon_f \geq 0$

Initializations :

- (O1) $x_1 = x_0 - {A_0}^{-1} f(x_0)$
- (O2) $d_0 = x_1 - x_0$
- (O3) $y_0 = f(x_1) - f(x_0)$
- (O4) $k = 1$

Iterations :

- (I1) The Broyden update:

$$A_k(x_k) = A_{k-1} + \frac{(y_{k-1} - A_{k-1}d_{k-1}){d^t}_{k-1}}{{d^t}_{k-1}d_{k-1}}$$

- (I2) Compute $d_k = d_k(x_k)$ the solution of $A_k d_k = -f(x_k)$
- (I3) $x_{k+1} = x_k + d_k(x_k)$
- (I4) $y_k = f(x_{k+1}) - f(x_k)$
- (I5) $k = k + 1$

Stop :

- (A1) $\frac{||x_{k+1} - x_k||}{||x_k||} < \varepsilon_x$
- (A2) $|f(x_{k+1})| \leq \varepsilon_f$

Output :

- (O1) $x^* = x_{k+1}$ is an approximation of the zero of f

The convergence is not guaranteed whenever the function is not enough linear or if the initial solution x_0 is too far from the solution. On the other hand, when the algorithm converges, it converges very fast.

8.6 Practice

8.6.1 Exercise 15-b : from a system to the zero

(This exercise has a prequel 15-a under 7.5.1 in the previous chapter)

This problem can be resolved with the *Quasi-Newton-Secant-Broyden* algorithm wich has the advantage of not using the *Jacobian* matrix. The latest is hence approximated and it is not required to compute its analytical form which can well be quite complicated.

8.6.2 Exercice 16 : Zero Newton unidimensional

Solve the problem given in the followin parts using:

- Newton
- String
- Secant

At each step, one should leave the trace of the result of the calculations in an array of the form:

k	x_k	d_k	$f(x_k)$	$f'(x_k)$	$f(x_k)/f'(x_k)$
0					
1					
2					
3					

8.6.2.1 Part 1

Solve the equation $x^2 - 2 = 0$. Use $x_0 = 2$ as start point and $10\ -3$ as stop criteria.

Resolution with Newton

Using :

- $d_k = d_k(x_k)$, the solution of $J_f(x_k)d_k = -f(x_k)$
- $x_k + 1 = x_k + d_k = x_k - \frac{f(x_k)}{f'(x_k)}$ (using this form here)
- $f'(x) = 2x$
- column d_k is used to store the relative error (stop criteria)

		Arrêt relatif	x^2-2	2x	*f/fprime*
k	x_k	d_x	$f(x_k)$	$f'(x_k)$	$f(x_k)/f'(x\,k)$
0	2		2	4	0.5
1	1.5	0.25	0.25	3	0.08333333
2	1.41666667	0.05555556	0.006944444	2.833333333	0.00245098
3	1.41421569	0.0017301	6.0073E-06	2.828431373	2.1239E-06
4	**1.41421356**	1.5018E-06	4.51061E-12	2.828427125	1.5947E-12
		<1e-3	<1e-3		

Resolution with the String

Just the same as above but using systematically $f'(x) = f'(x_0) = 2x_0 = 2 \cdot 2 = 4$

		Arrêt relatif	x^2-2	2x	*f/fprime*
k	x_k	d_x	$f(x_k)$	$f'(x_k)$	$f(x_k)/f'(x_0)$
0	2		2	4	0.5
1	1.5	0.33333333	0.25		0.0625
2	1.4375	0.04347826	0.06640625		0.01660156
3	1.42089844	0.01168385	0.01895237		0.00473809
4	1.41616035	0.00334573	0.005510123		0.00137753
5	1.41478281	0.00097367	0.001610412		0.0004026
6	**1.41438021**	0.00028465	0.000471382		0.00011785
		<1e-3	<1e-3		

Resolution with the Secant

Using the secant method, we get the first iteration (x_2) this way

$$
\begin{aligned}
x_{k+1} = x_k - \frac{f(x_k)}{\frac{f(x_{k-1})-f(x_k)}{x_{k-1}-x_k}} \Rightarrow \\
x_2 = x_1 - \frac{f(x_1)}{\frac{f(x_0)-f(x_1)}{x_0-x_1}} \\
= 1 - \frac{-1}{\frac{2-(-1)}{2-1}} \\
= 1 - \frac{-1}{3} = 1 - (-0.33333) = 1.33333
\end{aligned}
$$

It is particulary interesting to see that the Broyden method gives us just the same (as expected, Broyden is just the same as secant except it applies as well to nD)

Recall, the Broyden method requires :

- $x_0 = 2$ and $x_1 = 1$ are both provided
- $A_0 = 1$
- $d_0 = x_1 - x_0$
- $y_k = f(x_{k+1}) - f(x_k)$
- $A_k = A_{k-1} + \frac{(y_{k-1}-A_{k-1}d_{k-1})d^t{}_{k-1}}{d^t{}_{k-1}d_{k-1}}$
- $d_k = d_k(x_k)$ the solution of $A_k d_k = -f(x_k)$ (considering d_k is the approximation of $f'(x_k)$
- $x_{k+1} = x_k + d_k(x_k)$

For instance the first iteration :

- $k = 1$, $k - 1 = 0$, $x_0 = 2$ and $x_1 = 1$
- $A_0 = 1$
- $d_0 = 1 - 2 = -1$
- $y_0 = f(x_1) - f(x_0) = -1 - 2 = -3$
- $A_1 = A_0 + \frac{(y_0 - A_0 d_0)d_0}{d_0 d_0} = 1 + \frac{(-3-1\cdot(-1))\cdot(-1)}{-1\cdot(-1)} = 1 + \frac{(-3+1)\cdot(-1)}{1} = 1 + \frac{2}{1} = 3$
- $d_1 = -f(x_1) \cdot {A_1}^{-1} = 1 \cdot \frac{1}{3} = 0.3333$
- $x_2 = x_1 + d_1 = 1 + 0.3333 = 1.33333$

		Arrêt relatif	x^2-2		f/fprime
k	x_k	d_x	$f(x_k)$	apprimation $f'(x_k)$	$f(x_k)/f'(xk)$
0	2		2		
1	1	1	-1	3	-0.33333333
2	1.33333333	0.25	-0.222222222	2.333333333	-0.0952381
3	1.42857143	0.06666667	0.040816327	2.761904762	0.01477833
4	1.4137931	0.01045296	-0.001189061	2.842364532	-0.00041834
5	1.41421144	0.00029581	-6.00729E-06	2.828004542	-2.1242E-06
		<1e-3	<1e-3		

8.6.2.2 Part 2

Solve the equation $x = cosx$. Use $x_0 = 1.5$ (radians) as start point and $10 - 3$ as stop criteria.

Resolution with Newton

	rad	Arrêt relatif	cox-x	-sinx-1	f/fprime
k	x_k	d_x	$f(x_k)$	$f'(x_k)$	$f(x_k)/f'(xk)$
0	1.5		-1.429262798	-1.997494987	0.7155276
1	0.7844724	0.4770184	-0.076711304	-1.706451863	0.04495369
2	0.73951871	0.05730436	-0.000725709	-1.673932416	0.00043354
3	0.73908517	0.00058624	-6.94382E-08	-1.67361206	4.149E-08
		<1e-3	<1e-3		

Resolution with the String

	rad	Arrêt relatif	cox-x	-sinx-1	f/fprime
k	x_k	d_x	$f(x_k)$	$f'(x_k)$	$f(x_k)/f'(x_0)$
0	1.5		-1.429262798	-1.997494987	0.7155276
1	0.7844724	0.91211317	-0.076711304		0.03840375
2	0.74606864	0.05147483	-0.011705673		0.00586018
3	0.74020847	0.00791693	-0.001880493		0.00094143
4	0.73926704	0.00127346	-0.000304458		0.00015242
5	0.73911462	0.00020622	-4.93543E-05		2.4708E-05
		<1e-3	<1e-3		

Resolution with the Secant

	rad	Arrêt relatif	cox-x	Sécante	f/fprime
k	x_k	d_x	$f(x_k)$	apprimation $f'(x_k)$	$f(x_k)/f'(xk)$
0	1.5		-1.429262798		
1	1	0.5	-0.459697694	-1.939130208	0.23706386
2	0.76293614	0.31072569	-0.040126019	-1.769867721	0.02267176
3	0.74026438	0.03062657	-0.001974111	-1.682794154	0.00117312
4	0.73909126	0.00158724	-1.0258E-05	-1.674049918	6.1277E-06
5	0.73908513	8.2909E-06	-2.66935E-09	-1.673614295	1.595E-09
		<1e-3	<1e-3		

CHAPTER 9

Optimisation with *The Newton method*

Contents

9.1 Introduction

Let $f : \mathbb{R}^n \to \mathbb{R}$ a C^2 function for which we search the optimum. We will apply blindly here the *Newton* and *Quasi-newton* method to the optimisation problem, yet knowing they wont necessarily converge in every case.
The limitations are the same than that of the *non-linear system* solving algorithms. For the sake of completness, the reader should know that there are more complete and robust methods such as, for instance, BFGS which is a combination of both the *Descent* and *Quasi-Newton* methods.

9.1.1 Principle

The *necessary optimality condition* (but not sufficient) is $\nabla f(x) = 0$. Hence the idea of *the Newton method* is to use the *Newton* or *quasi-Newton* algorithms to find the zeros of the function:

$$g(x) = \nabla f(x)$$

Caution: The algorithm doesn't tell the nature of the zero. It can be a *maximum*, *minimum* or a *mountain pass*. Hence, a good knowledge of the problem is required and, in addition, it is always required to check the *sufficient condition*:

$\nabla^2 f(x^*)$ is **positive-definite**

in order to know whether we have a solution for the problem.

This is the second drawback of the *Newton method*. After the *lack of a guarantee for convergence*, it doesn't indicate the nature of the solution it finds.

9.2 The Newton method

Let's note that if f is a $\mathbb{R}^n \to \mathbb{R}$ function, then ∇f is a $\mathbb{R}^n \to \mathbb{R}^n$ function, we we can call g for instance:

$$\begin{aligned} f &: \mathbb{R}^n \to \mathbb{R} \Rightarrow \\ g = \nabla f &: \mathbb{R}^n \to \mathbb{R}^n \\ &: (x_1 ... x_n)^t \to (\partial_1 f .... \partial_n f)^t \end{aligned}$$

Hence, we can apply the *Newton* algorithm seen in the previous chapters (to resolve a n unknwon values, n equations system).

The iteration is given by the following relations:

$$x_{k+1} = x_k + d_k(x_k) \quad \text{where } J_g(x_k) d_{k+1} = -g(x_k)$$

By replacing $g(x)$ with $\nabla f(x)$, we get :

$$x_{k+1} = x_k + d_k(x_k) \quad \text{where } J_{\nabla f}(x_k) d_{k+1} = -\nabla f(x_k)$$

One should note:

$$\nabla g(x)^t = J_g(x_k) = \boxed{J_{\nabla_f}(x_k) = \nabla^2 f(x_k)}$$

as is demonstrated in the next section. Hence, the final notation of the iteration becomes:

$$\boxed{x_{k+1} = x_k + d_k(x_k)} \quad \text{where } \boxed{\nabla^2 f(x_k) d_{k+1} = -\nabla f(x_k)}$$

9.2.1 Relation between *jacobian of the gradient* and the *hessian*

In this relation resides the great simplicity of the approach. As stated above, if we considere the gradient of function f as a $g : \mathbb{R}^n \to \mathbb{R}^n$ function, the jacobian matrix of this function is equals to the hessian.

Considering:

$$\nabla f^t = (\partial_1 f ... \partial_n f) = (g_1 ... g_n) = g^t$$

Then:

$$J_g = \begin{pmatrix} \leftarrow & {\nabla_g}^t{}_1 & \rightarrow \\ & \dots & \\ \leftarrow & {\nabla_g}^t{}_n & \rightarrow \end{pmatrix} = \begin{pmatrix} \partial_1 g_1 & \dots & \partial_n g_1 \\ \dots & \dots & \dots \\ \partial_1 g_n & \dots & \partial_n g_n \end{pmatrix}$$
$$= \begin{pmatrix} \partial_1\partial_1 f & \dots & \partial_n\partial_1 f \\ \dots & \dots & \dots \\ \partial_1\partial_n f & \dots & \partial_n\partial_n f \end{pmatrix} = \begin{pmatrix} \partial_{11} f & \dots & \partial_{n1} f \\ \dots & \dots & \dots \\ \partial_{1n} f & \dots & \partial_{nn} f \end{pmatrix} = \nabla_2 f$$

9.2.1.1 Example

Let's say $f(x_1, x_2) = \frac{1}{2}(x_1 x_2 - 2)^2 + \frac{1}{2}(x_1 - 2)^2$. Let's compute the Jacobian of the gradient matrix and the hessian.

First, we need a resolved way of $f(x)$:

$$f(x_1, x_2) = \frac{1}{2}{x_1}^2{x_2}^2 - 3x_1x_2 + 2 + \frac{1}{2}{x_1}^2 + \frac{1}{2}{x_2}^2$$

Let's compute the derivatives:

$$\begin{aligned} \partial_1 f &= x_1{x_2}^2 - 3x_2 + x_1 \\ \partial_2 f &= {x_1}^2 x_2 - 3x_1 + x_2 \\ \partial_1 1 f &= {x_2}^2 + 1 \\ \partial_1 2 f &= \partial_2 1 f = 2x_1x_2 - 3 \\ \partial_2 2 f &= {x_1}^2 + 1 \end{aligned}$$

The hessian is:

$$\nabla^2 f(x) = \begin{pmatrix} {x_2}^2 + 1 & 2x_1x_2 - 3 \\ 2x_1x_2 - 3 & {x_1}^2 + 1 \end{pmatrix}$$

The gradient matrix is:

$$\nabla^2 f(x) = \begin{pmatrix} x_1{x_2}^2 - 3x_2 + x_1 \\ {x_1}^2 x_2 - 3x_1 + x_2 \end{pmatrix}$$

The jacobi of the gradient is:

$$\nabla^2 f(x) = \begin{pmatrix} {x_2}^2 + 1 & 2x_1x_2 - 3 \\ 2x_1x_2 - 3 & {x_1}^2 + 1 \end{pmatrix}$$

9.2.2 Algorithm for the *Newton method*

Inputs :

- (E1) A function $f : \mathbb{R}^n \to \mathbb{R} \in C^2$ which we want to optimize
- (E2) The gradient $\nabla f :: \mathbb{R}^n \to \mathbb{R}^n$ continued
- (E3) The hessian $\nabla^2 f :: \mathbb{R}^n \to \mathbb{R}^n$ continued
- (E4) A first approximation of the solution $x_0 \in \mathbb{R}^n$
- (E5) The asked precison $\varepsilon_x \geq 0$, $\varepsilon_{\nabla f} \geq 0$

Initialization :

- (O1) k = 0

Iterations :

- (I1) Compute $d_k(x_k)$ the solution of $\nabla^2 f(x_k) d_k = -\nabla f(x_k)$
- (I2) $x_{k+1} = x_k + d_k(x_k)$
- (I3) $k = k + 1$

Stop :

- (A1) $\frac{||x_{k+1} - x_k||}{||x_k||} < \varepsilon_x$
- (A2) $|\nabla f(x_{k+1})| \leq \varepsilon_{\nabla f}$

Output :

- (O1) $x^* = x_{k+1}$ is an approximation of the zero of f
- Caution: one doesn't know whether the returned optimum is a minimum, a maximum or a mountain pass.

The convergence is not guaranteed whenever the function is not enough linear or if the initial solution x_0 is too far from the solution. On the other hand, when the algorithm converges, it converges very fast.

9.3 The *Quasi-Newton-Secant-Broyden method*

In concrete problems, the *hessian* is often very difficult to compute and/or code, often because of the many indices. In the end, it is rarely correctly computed.
This is where the *Quasi-Newton* method kicks in. It avoid the computation of the hessian by using an approximation instead.

9.3.1 Algorithm for the *Quasi-Newton-Secant-Broyden method*

Inputs :

- (E1) A function $f : \mathbb{R}^n \to \mathbb{R} \in C^2$ which we want to optimize
- (E2) The gradient $\nabla f :: \mathbb{R}^n \to \mathbb{R}^n$ continued
- (E3) A first approximation of the hessian A_0 (by default $A_0 = I$)
- (E4) A first approximation of the solution $x_0 \in \mathbb{R}^n$
- (E5) The asked precison $\varepsilon_x \geq 0$, $\varepsilon_{\nabla f} \geq 0$

Initializations :

- (O1) $x_1 = x_0 - {A_0}^{-1} \nabla f(x_0)$
- (O2) $d_0 = x_1 - x_0$
- (O3) $y_0 = \nabla f(x_1) - \nabla f(x_0)$
- (O4) $k = 1$

Iterations :

- (I1) The Broyden update:

$$A_k(x_k) = A_{k-1} + \frac{(y_{k-1} - A_{k-1}d_{k-1})d^t{}_{k-1}}{d^t{}_{k-1}d_{k-1}}$$

- (I2) Compute $d_k = d_k(x_k)$ the solution of $A_k d_k = -\nabla f(x_k)$
- (I3) $x_{k+1} = x_k + d_k(x_k)$
- (I4) $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
- (I5) $k = k + 1$

Stop :

- (A1) $\frac{||x_{k+1}-x_k||}{||x_k||} < \varepsilon_x$
- (A2) $|\nabla f(x_{k+1})| \leq \varepsilon_{\nabla f}$

Output :

- (O1) $x^* = x_{k+1}$ is an approximation of the zero of f
- Caution: one doesn't know whether the returned optimum is a minimum, a maximum or a mountain pass.

The convergence is not guaranteed whenever the function is not enough linear or if the initial solution x_0 is too far from the solution. On the other hand, when the algorithm converges, it converges very fast.

9.4 Practice

9.4.1 Exercise 19 : *Quasi-Newton-Secant-Broyden*

One wants to resolve the following optimisation problem:

$$\min_{(x_1,x_2)\in X\subset\mathbb{R}^2} f(x_1, x_2)$$

using the following algorithms:

- Newton
- Quasi-Newton-Secant-Broyden

What are the input parameters (gradient, etc.) one needs to compute for each of the algorithm ?

9.4.1.1 Solution

For Newton, the gradient and the hessian need to be computed.
For Quasi-Newton-Secant-Broyden, only the gradient needs to be computed.

Part II

Linear Programming

CHAPTER 10

Linear programming

Contents

10.1 Introduction

In **Linear Programming**, we are facing a function f of the form $f : \mathbb{R}^n \to \mathbb{R}$. The goals is to find *argmin* or *argmax* of function f **but** by taking additionnal linear constraints into consideration. The **additional constraints form a linear system**. The constraint for the solution x to belong to a specific domain D. $x \in D \in \mathbb{R}^n$ **and** D **is convexe**.

The approach enabling one to solve such kind of problems is called **Linear optimization or Linear Programming**. We are looking for efficient algorithms enabling one to solve systems with thousands of variables and/or constraints.

We'll introduce the typical problems one has to solve using Linear Programming (LP or **PL** in french) with an example.

10.1.1 The problem of a manufacturing company

A company is specialized in the manufacturing of two types of products:

- Air conditioners
- Fans (french: ventilateurs)

The two types of product required a given amount of work hours from:

- The machines
- The man workforce (labor)

The following array illustrates the amount of work hours required from both resources in order to complete the products manufacturing as well as the profit. Each are given per *monetary unit (MU)*:

	Machine work hours	**Labor work hour**	**Profit**
Air conditioner	2h / unit	3h / unit	25 UM / unit
Fan	2h / unit	1h / unit	16 UM / unit
Total available	240	140	

10.1.2 Modeling

10.1.2.1 Decision Variables

The company wants to decide the amount of air conditioners and the amount of fans it should be manufacturing in order to maximize the profit. This leads to choosing two **decision variables**:

- $x_1 =$ number of air conditioners
- $x_2 =$ number of fans

10.1.2.2 The score function

The objective of the company (implicit in the text) is to find out the production program that should maximize its profit. Hence the score is the profit which is given by:

$$\max z(x) = 25x_1 + 15x_2$$

10.1.2.3 Constraints of the model

The resources are limited, which imply a series of constraints in the model:

$$\begin{aligned} 2x_1 + 2x_2 &\leq 240 && \text{available machine workhour} \\ 3x_1 + x_2 &\leq 140 && \text{available labour workhour} \\ x_1 \geq 0 \;\; x_2 &\geq 0 && \text{non-negativity constraints} \end{aligned}$$

10.1.2.4 Complete Model

$$(LP)\begin{cases}\max z(x) = 25x_1 + 15x_2 \\ u.c. \left|\begin{array}{l}(S)\begin{cases}2x_1 + 2x_2 \leq 240 \\ 3x_1 + x_2 \leq 140\end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)}\end{array}\right.\end{cases}$$

10.2 Definitions

10.2.1 Linear Programming

In the context of **Linear Programming** (LP or PL), one needs to solve the type of problems illustrated above. The linearity appears as much in the score function $z(x)$ than in the constraints of the system (S).

10.2.2 Feasible solutions

The set of points satisfying the constraints are called **feasible solutions** (in french *solutions admissibles*). A solution is *feasible* $\Leftrightarrow$ (iff) it is a solution of the linear system (S) and if it satisfies each of the *positivity constraints*.

10.2.3 The score function

The function to be minimized (resp. maximized) is called the **score function**, or *cost function* or even *economic function*.

10.3 Math reminder

10.3.1 The Gauss method

Let's consider, for instance, such a system:

$$\underbrace{\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}}_{A} \cdot \underbrace{\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} b_1 \\ b_2 \end{pmatrix}}_{b}$$

We want an efficient way to solve such kind of systems because systematically inversing the matrix the good'ol way is a lot to costy. There is therefore the *Gauss method*.

10.3.1.1 Method 1 : build the identity matrix

Build a *matrix-form* representation of the problem which ends up with the matrix A on the left and the vector b on the right.
Then perform a series of **elementary row operations** to transform the left-side matrix A into the *identity matrix*. This ends up with the solution of the vector x on the right.

$$\begin{array}{c} (\ A \mid b\ ) \\ \wr \\ (\ I \mid x\ ) \end{array} \qquad \text{row operations}$$

The possible operations on rows are exhaustively:

- Multiply a row by a factor (fraction, negative, whatever)
- Add a row to another row (with whatever factor)

10.3.1.2 Method 2 : enrich method 1 to get the inverse of the matrix A

One can enrich the previous method to get the inverse of the matrix A as well.

$$\begin{array}{c} (\ A \mid \ I \ \mid b\ ) \\ \wr \\ (\ I \mid A^{-1} \mid x\ ) \end{array} \qquad \text{row operations}$$

10.3.1.3 Method 3 : Gauss pivot

This third method is just a way to sytematize the resolution. The idea is to perform the *row operations* towards a specific order of resolution.

First we try to build a left-side matrix which has 0 everywhere under the diagonal line, only 1's on the diagonal line without considerating what is above the diagional line:

$$\left(\begin{array}{cccc|c} 1 & . & . & * & * \\ 0 & 1 & . & . & * \\ ... & ... & ... & ... & * \\ 0 & 0 & 0 & 1 & x_n \end{array}\right)$$

As shown above, this enables one to find the value of the last index of x, i.e x_n. Once this one is obtained, and at least one row is composed by only 0's and a single 1, it is a lot easier to resolve the other rows and find the other indices of x.
This method is called the **Gauss pivot**.

The order to follow when attempting to fill the part below the diagonal line with 0's should be the following:

$$\left(\begin{array}{cccc|c} ... & ... & ... & ... & * \\ 3 & ... & ... & ... & * \\ 2 & 5 & ... & ... & * \\ 1 & 4 & 6 & ... & * \end{array}\right)$$

10.3.2 Algebra reminder

$$a \geq b \Leftrightarrow a - \varepsilon = b \text{ with } \varepsilon \geq 0$$
$$a \leq b \Leftrightarrow a + \varepsilon = b \text{ with } \varepsilon \geq 0$$
$$a \leq b \Leftrightarrow -a \geq -b$$
$$a \in \mathbb{R} \Leftrightarrow \begin{cases} a = a' - a'' \\ a' \geq 0 \\ a'' \geq 0 \end{cases}$$

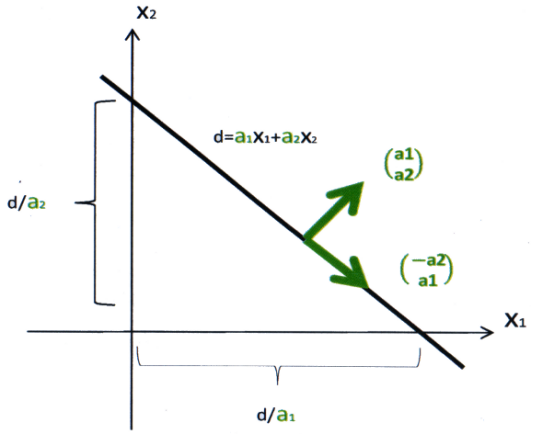
The last one states that each and every *real number* can be expressed as the substraction of two other real numbers.

10.3.3 Draw a line on a graph

Just a quick reminder on how to draw a line on a plan when given the equation of the line.
Let's assume one is given the equation of a line in the form

$x_2 = u \cdot x_1 + v$ Let's transform it in :
$d = a_1 \cdot x_1 + a_2 \cdot x_2$

Drawing the line becomes then easy. See graph on the right.



10.4 Practice

10.4.1 Exercise 1 : Gauss

Let (S) be the following system of linear equations:

$$(S) \begin{cases} x_1 - x_2 - x_3 = 0 \\ 2x_1 + x_2 - 2x_3 = -3 \\ -x_1 + 2x_2 - x_3 = -5 \end{cases}$$

One can write it in matrix form:

$$(S) \underbrace{\begin{pmatrix} 1 & -1 & -1 \\ 2 & 1 & -2 \\ -1 & 2 & -1 \end{pmatrix}}_{A} \cdot \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 0 \\ -3 \\ -5 \end{pmatrix}}_{b}$$

Use *elementary row operations* to find the solution of the system (S) and *at the same time* the inverse matrix A^{-1}.
Then check that the solution obtained for x is valid and ensure the A^{-1} matrix is correct.

10.4.1.1 Run Gauss Pivot

$$(S)\left(\begin{array}{ccc|ccc|c} 1 & -1 & -1 & 1 & 0 & 0 & 0 \\ 2 & 1 & -2 & 0 & 1 & 0 & -3 \\ -1 & 2 & -1 & 0 & 0 & 1 & -5 \end{array}\right)\begin{array}{c} I \\ II \\ III \end{array}$$

$$(S)\left(\begin{array}{ccc|ccc|c} 1 & -1 & -1 & 1 & 0 & 0 & 0 \\ 2 & 1 & -2 & 0 & 1 & 0 & -3 \\ 0 & 1 & -2 & 1 & 0 & 1 & -5 \end{array}\right)\begin{array}{c} I \\ II \\ III + I \end{array}$$

$$(S)\left(\begin{array}{ccc|ccc|c} 1 & -1 & -1 & 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & -2 & 1 & 0 & -3 \\ 0 & 1 & -2 & 1 & 0 & 1 & -5 \end{array}\right)\begin{array}{c} I \\ II - 2I \\ III \end{array}$$

$$(S)\left(\begin{array}{ccc|ccc|c} 1 & -1 & -1 & 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & -2 & 1 & 0 & -3 \\ 0 & 0 & -2 & \frac{5}{3} & -\frac{1}{3} & 1 & -4 \end{array}\right)\begin{array}{c} I \\ II \\ III - \frac{1}{3}II \end{array}$$

$$(S)\left(\begin{array}{ccc|ccc|c} 1 & -1 & -1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\frac{2}{3} & \frac{1}{3} & 0 & -1 \\ 0 & 0 & 1 & -\frac{5}{6} & \frac{1}{6} & -\frac{1}{2} & 2 \end{array}\right)\begin{array}{c} I \\ \frac{1}{3}II \\ -\frac{1}{2}III \end{array}$$

$$(S)\left(\begin{array}{ccc|ccc|c} 1 & 0 & 0 & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & 1 \\ 0 & 1 & 0 & -\frac{2}{3} & \frac{1}{3} & 0 & -1 \\ 0 & 0 & 1 & -\frac{5}{6} & \frac{1}{6} & -\frac{1}{2} & 2 \end{array}\right)\begin{array}{c} I + II + III \\ II \\ III \end{array}$$

10.4.1.2 Results

$$A^{-1} = \begin{pmatrix} -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{2}{3} & \frac{1}{3} & 0 \\ -\frac{5}{6} & \frac{1}{6} & -\frac{1}{2} \end{pmatrix} \quad \text{and } x = \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix}$$

10.4.1.3 Verifications

$$A \cdot A^{-1} = \begin{pmatrix} 1 & -1 & -1 \\ 2 & 1 & -2 \\ -1 & 2 & -1 \end{pmatrix} \cdot \begin{pmatrix} -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{2}{3} & \frac{1}{3} & 0 \\ -\frac{5}{6} & \frac{1}{6} & -\frac{1}{2} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$A \cdot x = \begin{pmatrix} 1 & -1 & -1 \\ 2 & 1 & -2 \\ -1 & 2 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1+1-2 \\ 2-1-4 \\ -1-2-2 \end{pmatrix} = \begin{pmatrix} 0 \\ -3 \\ -5 \end{pmatrix} = b$$

10.4.2 Exercise 2 : Modeling

Put in LP form the following problems

10.4.2.1 Part I : feeding the cattle

A farmer wants to find out the amount of three type of cereal grains to be given to the cattle in order to meet their nutritional needs, to the minimum possible cost. The required data is given below:
(In french: Corn=maïs, wheat=blé, barley=orge)

	Corn	**Wheat**	**Barley**	**Minimum required**
Protein (mg / kg)	10	9	11	20
Calcium (mg / kg)	50	45	58	70
Iron (mg / kg)	9	8	7	12
Calories (Cal / kg)	1000	800	850	4000
Cost / kg	5.5 UM	4.7 UM	4.5 UM	

Under LP form:

x_1 = amount of *corn* in kg, x_2 = amount of *wheat* in kg, x_3 = amount of *barley* in kg

$$(LP)\begin{cases}\min z(x) = 5.5x_1 + 4.7x_2 + 4.5x_3 \\ u.c. \quad \left|\begin{array}{l} (S)\begin{cases} 10x_1 + 9x_2 + 11x_3 \geq 20 \\ 50x_1 + 45x_2 + 58x_3 \geq 70 \\ 9x_1 + 8x_2 + 7x_3 \geq 12 \\ 1000x_1 + 800x_2 + 850x_3 \geq 4000 \end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)} \end{array}\right. \end{cases}$$

10.4.2.2 Part II : transportation

A company has 3 manufacturing centers (U_1, U_2, U_3) and three resellers (V_1, V_2, V_3). The CEO wants to *minimize* the total transportation cost of the products brought to the reselling centers. The required data is given below:

Manufact.	**production**
U1)	200
U2)	150
U3)	300

Resellers.	**demand**
V1)	150
V2)	200
V3)	200

Transp. cost	**V1**	**V2**	**V3**
U1)	10	7	8
U2)	15	12	9
U3)	7	8	12

Under LP form:

x_{ij} = amount of units transported between manuf. center Ui and reseller Vj

$$(LP)\begin{cases}\min z(x) = 10x_{11} + 7x_{12} + 8x_{13} + 15x_{21} + 12x_{22} + 9x_{23} + 7x_{31} + 8x_{32} + 12x_{33} \\ u.c. \quad (S)\begin{cases} x_{11} + x_{12} + x_{13} \leq 200 \\ x_{21} + x_{22} + x_{23} \leq 150 \\ x_{31} + x_{32} + x_{33} \leq 300 \\ x_{11} + x_{21} + x_{31} \geq 150 \\ x_{12} + x_{22} + x_{32} \geq 200 \\ x_{13} + x_{23} + x_{33} \geq 200 \end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)}\end{cases}$$

10.4.2.3 Part III : Bike production

A bike manufacturer produces 4 different types of bikes:

Cycle type	Unit Profit	Assembly time	Break type
c1	12 UM	30 min /unit	gummy
c2	25 UM	60 min /unit	gummy
c3	12 UM	45 min /unit	disk
c4	16 UM	45 min /unit	gummy

The available resources are:

- 32 assembly hours
- 48 tires for C1
- 144 tires for the other models
- 30 disk breaks
- 55 gummy breaks

There is also an additional constraint: the manufacturer must satisfy daily a minimum amout of 6 bikes C1 et 6 bikes C2.

Under LP form:

x_i = number of bike Ci to be manufactured everyday.

$$(LP)\begin{cases}\max z(x) = 12x_1 + 25x_2 + 12x_3 + 16x_4 \\ u.c. \quad (S)\begin{cases} 0.5x_1 + x_2 + 0.75x_3 + 0.74x_4 \leq 32 \\ 2x_1 \leq 48 \\ 2x_2 + 2x_3 + 2x_4 \leq 144 \\ x_1 \geq 6 \\ x_2 \geq 6 \\ x_1 + x_2 + x_4 \leq 55/2 \\ x_3 \leq 30/2 \end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)}\end{cases}$$

CHAPTER 11

Geometric Approach

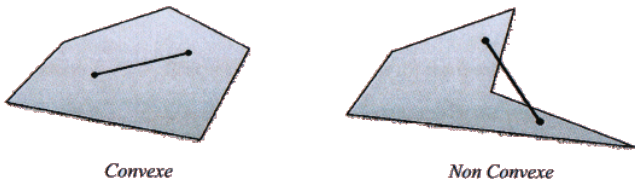
Contents

This approach does only work for very little LP where the number of decision variables is no more than 2 or perhaps 3.

11.1 Introduction

11.1.1 Definition - Convexe

A set is **Convexe** whenever every edge connecting two vertices of the form is only within the form itself, never outside.
See image on the left for a formal illustration.

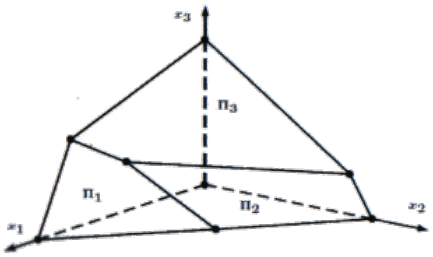


Convexe *Non Convexe*

11.1.2 Definition - polyhedron

A **polyhedron** is a geometrical form *closed bounded* (french : fermée bornée) formed by the intersection of *hyperplans*.
Examples:

- (E1) A polygon is a polyhedron of dimension 2, whose sides are segments
- (E2) A diamond is a convexe polyhedron of dimension 3.
- (E3) A PL with three decision variables (x_1, x_2, x_3) and three linear constraints represented by the plans (Π_1, Π_2, Π_3) posess *feasible solutions* in the polyhedron defined by the three plans Π as well as the ground and the walls formed b y the positivity constraints.



11.2 Approach

The approach strongly relies on the fact that **the set of solution belongs to a convexe polyhedron**. As a first step, one should ensure the targer LP sytem respects this condition. (theorem)

Then, knowing we are looking for an extreme value, either an argmin or an argmax following a linear score function, the solution is mandatorily a vertex of the polyhedron. (theorem)

11.2.1 Naive algorithm

Based on these theorems, one can extract a naive algorithm:

- Find the vertices of the polyhedron
- For each of them, evaluate the score function
- Keep the one giving the best result

This algorithm is systematic and programmable. It can however not be applied on large LP problems as the nuber of vertices is prohibitive.

11.2.2 geometrical approach

A geometrical approach is largely based on the naive algorithm presented above. However, the ability to draw a little schema of the solution polyhedron enables one to find the solution faster than testing each and every vertex.
The next section presents an example to illustrate the technic.

11.3 Illustration example

Let's solve the example LP from the previous chapter using the geometrical approach. The PL is the

$$(LP)\begin{cases}\max z(x) = 25x_1 + 15x_2 \\ w.r. \left| \begin{array}{l} (S)\begin{cases}2x_1 + 2x_2 \leq 240 \\ 3x_1 + x_2 \leq 140\end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)}\end{array}\right.\end{cases}$$

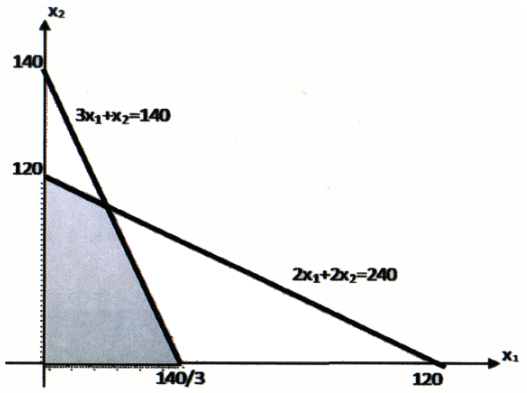
Observations

1. Each and every constraint of (S) represent the equation of a *half-plan*.
2. The set of every constraints of (S), form an intersection of half-plans, i.e. a convexe polygon.
3. The set of *feasible solutions* is represented by this polygon.
4. The graph os the score function $z(x)$ is linear, i.e. it is by linearity a plan.
5. The *countour curves* of the score function are lines.

11.3.1 Stage 1 : draw the polygon

Let's use the different constraints to draw the convexe polygon representing the set of *feasible solutions*.

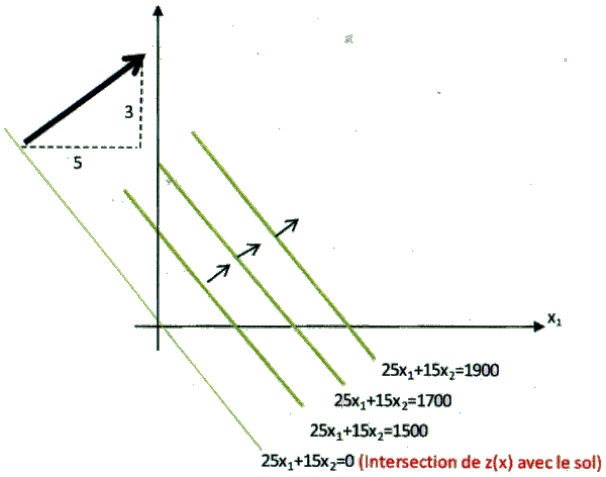
On the graph on the right, the polygon is the area in gray.



11.3.2 Stage 2 : draw the countour curves

A countour curve of the score function $z(x) = 25x_1 + 15x_2$ is a curve that contains all the points on the surface of the graphe $z(x)$ that are at *the same height*.
One can choose a constant c that fixes a specific value for that height. Hence the equation $c = 25x_1 + 15x_2$ represents a countour curve of $z(x)$. These countour curves are obviously lines since the graphe $z(x)$ is a plan.

The graph on the right represents various countour curves. The more the curve is on the "*right*", the more the height it represent is high. The height is viven by the value of c.



As the function $z(x)$ is 0 at the origin and the *normal vector* to the line is $z(x)$ is

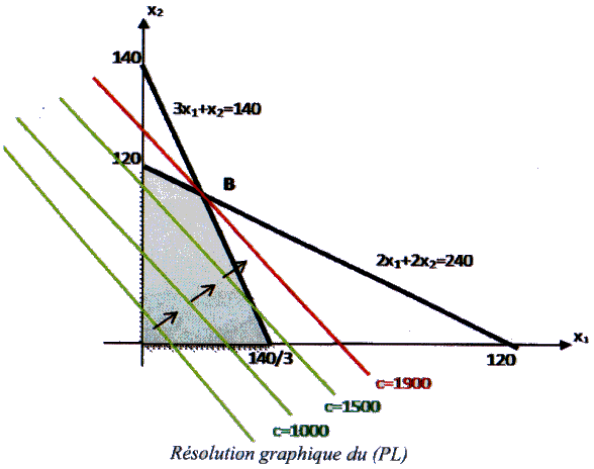
$$n = \begin{pmatrix} 25 \\ 15 \end{pmatrix} // \begin{pmatrix} 5 \\ 3 \end{pmatrix} \quad \text{(gradient of } z(x) \text{)}$$

it is easy to draw the line for height 0. The other curves are parallel.

11.3.3 Stage 3 : find the highest curve

Ae're now left with searching the countour curve, the closest to the highest possible value of $z(x)$ in the convexe polygon.

By overlaying the countour curves on the set of of feasible solutions represented by the convexe polygon, one immediately sees the solution of the LP.
Indeed, we are looking to maximize $z(x)$, i.e. to find the highest point on top of the convexe polygon. This implies searching the highest countour curve on top of the convexe polygon.
Here, this is the right-most curve runnin on point B, where B is the solution of the system:



Résolution graphique du (PL)

$$\begin{cases} 2x_1 + 2x_2 = 240 \\ 3x_1 + x_2 = 140 \end{cases} \Rightarrow B = z(10, 110)$$

In conclusion, the solution of the LP is $\arg\max z(x_1, x_2) = (10, 110$ and $\max z(x) = 1900$.

11.4 Graphical sensitivity analysis

We study here the sensitivity of the optimal solution on a variation in the coefficients of the score function.

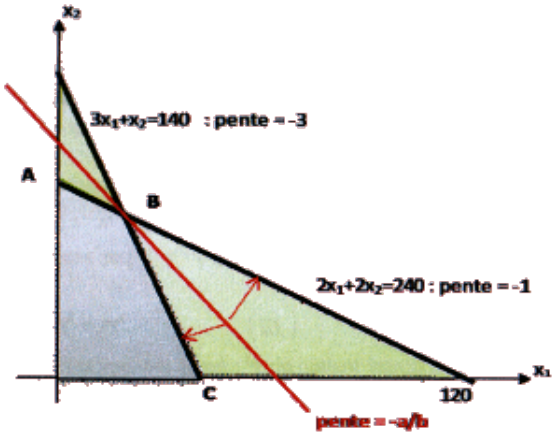
Let's considere the same LP as previously:

$$(LP)\begin{cases}\max z(x) = 25x_1 + 15x_2 \\ w.r. \left| \begin{array}{l} (S)\begin{cases}2x_1 + 2x_2 \leq 240 \\ 3x_1 + x_2 \leq 140\end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)}\end{array}\right.\end{cases}$$

Let's replace the coefficients 25 and 15 by a and b :

$$(LP)\begin{cases}\max z(x) = ax_1 + bx_2 \\ w.r. \left| \begin{array}{l} (S)\begin{cases}2x_1 + 2x_2 \leq 240 \\ 3x_1 + x_2 \leq 140\end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)}\end{array}\right.\end{cases}$$

We can see in the graph below that the solution does not vary as long as the slope of the lines of the countour curves is kept within a specific range.
Teh slope of the line is $-a/b$ given by teh coefficients of the score function. A slight variation in these coefficients doesn't change the solution of the LP



As long as the red line (countour curve) remains in the green cone, the solution of the PL remains unchanged and s stays on the point B of the polyhedron.
If, when rotating over B, the slope of the countour line in red gets under the line AB, A becomes the solution.
If the slope of the countour line in red gets above the line BC, C becomes the solution.

In other terms:

Slope	Solution
$]-\infty, -3[$	C
$]-3, -1[$	B
$]-1, \infty[$	C

For -3, all the points on BC are solutions. For -1, all the points on AB are solutions.

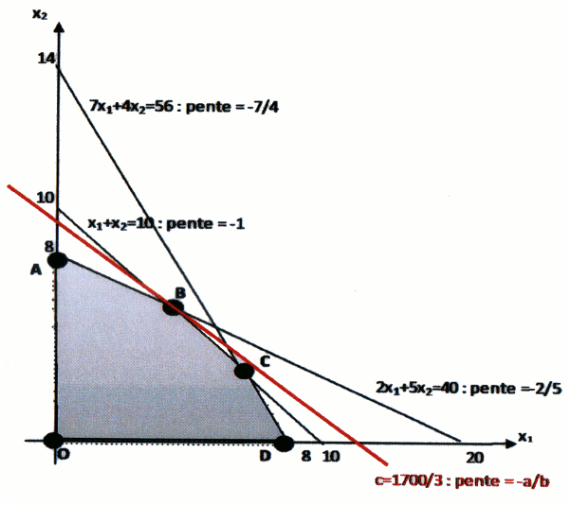
11.5 Practice

11.5.1 Exercise 3 : geometrical approach

Solve the following LP graphically and run a graphical sensitivity analysis:

$$(LP)\begin{cases}\max z(x) = 30x_1 + 70x_2 \\ w.r. \left| \begin{array}{l} (S)\begin{cases}2x_1 + 5x_2 \leq 40 \\ 7x_1 + 4x_2 \leq 56 \\ x_1 + x_2 \leq 10\end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)}\end{array}\right.\end{cases}$$

11.5.1.1 Graphical solving of the PL



$A = A(0,8)$ $z = 560$
$B = B(\frac{10}{3}, \frac{20}{3})$ $z = \frac{1700}{3}$
$C = C(\frac{16}{3}, \frac{14}{3})$ $z = \frac{1460}{3}$
$D = D(8,0)$ $z = 240$
$O = O(0,0)$ $z = 0$

Hence B is the solution point.

11.5.1.2 Sensitivity Analysis

$$(LP) \begin{cases} w.r. \\ \left| \begin{array}{l} (S) \begin{cases} 2x_1 + 5x_2 \leq 40 \\ 7x_1 + 4x_2 \leq 56 \\ x_1 + x_2 \leq 10 \end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)} \end{array} \right. \end{cases}$$

Slope	**Max**
$]-\infty, -7/4[$	D
$]-7/4, -1[$	C
$]-1, -2/5[$	B
$]-2/5, \infty[$	A

$-\frac{7}{4}$ on segment CD
-1 on segment BC
$-\frac{2}{5}$ on segment AB

CHAPTER 12

Algebraic Approach - The *Simplex* algorithm

Contents

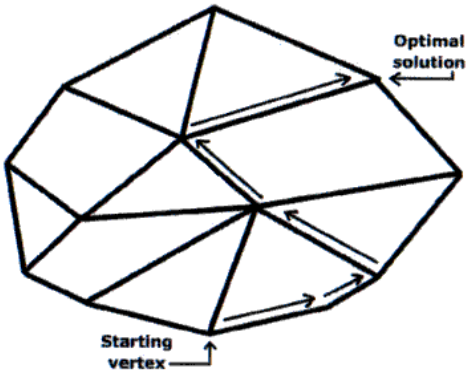
12.1 Introduction

The idea here is to find an efficient algorithm which can be applied to very large systems with a high number of dimensions. The algorithm is **the Simplex**.

The idea at the root of the Simplex is still to find the set of feasible solutions, knowing it contains the vertex of the **polytope** (= convexe polyhedron of higher dimensions) giving the best score. As the set of vertices can well be very huge for a large size LP, the idea is not to go through each and every vertex but only a few of them.
One starts with an initial vertex and, for each iteration of the Simplex, one moves to an adjacent vertex where the score is better. When there is not better in the neighbourhood of the current one, the algorithm stops and the current vertex is the solution to the LP.

The algorithm doesn't run through each and every vertex of the polytope since it always moves to a better vertex. There is never no going back nor any regression.

Below, we will see a "naive" approach which underlines that the method illustrated on the right schema on the right comes from a very simple algebraic approach quite easy to understand.



12.2 Illustration example

The *Simplex* - despite its name - is not an easy algorithm, not that its overwhelmingly complicated either. Yet explaining the Simplex on a purely theoretical plan is a bit tricky.
We will thus run through an example to present the algorithm.

The example we will be using is:

$$(LP)\begin{cases}\min z(x) = 2x_1 - 3x_2 + 5x_3 \\ w.r. \left| \begin{array}{l} (S)\begin{cases} x_1 - 2x_2 + x_3 - x_4 = 4 \\ x_2 + 3x_3 + x_5 = 6 \\ 2x_1 + x_3 + 2x_4 + x_6 = 7 \end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)} \end{array} \right. \end{cases}$$

We don't care about the score function at start. We focus first on finding an admissible solution, i.e. an initial solution that satisfies the constraints.
This first step resolved in solving a 3 equations with 6 unknown variables system. The set of solutions is infinite.

Amongst this infinity, we are looking for a first one that satisfies the constraints as well as the positivity constraints.

The *degree of freedom* is 3 $\Rightarrow$ 3 variables are taken out of the system and replaced by *lambda* values as we'll see below.

12.2.1 The technique of parameterization

We are facing a 6 unknown variables system with only 3 equations. Let's make it a solvable system. We choose 3 variables to be represented as parameters. Thanks to this parameterization of three of the variables, we're left with a 3 equations with 3 unknown variables, hence a solvable system.

12.2.1.1 First attempt - choice 1

Let's say (x_1, x_2, x_3) = (λ_1, λ_2, λ_3) are the chosen parameters

This gives us the following system:

$$(S)\begin{cases}\lambda_1 - 2\lambda_2 + \lambda_3 - x_4 = 4 & (I)\\ \lambda_2 + 3\lambda_3 + x_5 = 6 & (II)\\ 2\lambda_1 + \lambda_3 + 2x_4 + x_6 = 7 & (II)\end{cases}$$

The parameterized variables $(x_1,\ x_2,\ x_3) = (\lambda_1,\ \lambda_2,\ \lambda_3)$ are called the **off-base variables** and the remaining ones are the **base variables**.
Next, let's express the *base variables* using the *off-base variables*:

For (I) and (II), the resolution is quite straightforward:

$$(S)\begin{cases}x_4 = -4 + \lambda_1 - 2\lambda_2 + \lambda_3 & (I)\\ x_5 = 6 - \lambda_2 - 3\lambda_3 & (II)\\ 2\lambda_1 + \lambda_3 + 2x_4 + x_6 = 7 & (III)\end{cases}$$

by substituting (I) in (III), The system (S) is solved

$$(S)\begin{cases}x_4 = -4 + \lambda_1 - 2\lambda_2 + \lambda_3 & (I)\\ x_5 = 6 - \lambda_2 - 3\lambda_3 & (II)\\ X_6 = 15 - 4\lambda_1 + 4\lambda_2 - 3\lambda_3 & (III)\end{cases}$$

Once the system is resolved, any possible value for $(\lambda_1,\ \lambda_2,\ \lambda_3)$ still provides a solution of (S), such as, for instance $(\lambda_1,\ \lambda_2,\ \lambda_3) = (0,\ 0,\ 0)$.
This gives us $(x_1,\ x_2,\ x_3) = (-4,\ 6,\ 1)$. Unfortunately this solution is not admissible since -4 doesn't satisfy one of the positivity constraint. No luck !

12.2.1.2 Second attempt - choice 2

Let's try another triple of x_i for the *off-base variables* λ_1, hoping this time to have a little more success. In addition, we want a solution with the *off-base variables*, the λ_1 parameters set to 0.

Let's say $(x_2,\ x_4,\ x_5) = (\lambda_2,\ \lambda_4,\ \lambda_5)$ are *off-base*.

This gives us the following system:

$$(S)\begin{cases}x_1 - 2\lambda_2 + x_3 - \lambda_4 = 4 & (I)\\ \lambda_2 + 3x_3 + \lambda_5 = 6 & (II)\\ 2x_1 + x_3 + 2\lambda_4 + x_6 = 7 & (II)\end{cases}$$

As before, lets try to solve this system by expressing the *base variables* $(x_1,\ x_3,\ x_6)$ with the *off-base variables* $(x_2,\ x_4,\ x_5) = (\lambda_2,\ \lambda_4,\ \lambda_5)$.

the extraction of x_3 is free in (II):

$$(S)\begin{cases}x_1 - 2\lambda_2 + x_3 - \lambda_4 = 4 & (I)\\ x_3 = 2 - \frac{1}{3}\lambda_2 - \frac{1}{3}\lambda_5 & (II)\\ 2x_1 + x_3 + 2\lambda_4 + x_6 = 7 & (II)\end{cases}$$

then by substituting (II) in (I), one gets:

$$(S)\begin{cases}x_1 = 2 + \frac{7}{3}\lambda_2 + \lambda_4 + \frac{1}{3}\lambda_5 & (I)\\ x_3 = 2 - \frac{1}{3}\lambda_2 - \frac{1}{3}\lambda_5 & (II)\\ x_6 = 7 - 2x_1 - x_3 - 2\lambda_4 & (III)\end{cases}$$

finally with (I) and (II) in (III)

$$(S)\begin{cases}x_1 = 2 + \frac{7}{3}\lambda_2 + \lambda_4 + \frac{1}{3}\lambda_5 & (I)\\ x_3 = 2 - \frac{1}{3}\lambda_2 - \frac{1}{3}\lambda_5 & (II)\\ x_6 = 1 - \frac{13}{3}\lambda_2 - 4\lambda_4 - \frac{1}{3}\lambda_5 & (III)\end{cases}$$

In this system, using $(\lambda_2,\ \lambda_4,\ \lambda_5) = (0,\ 0,\ 0)$ gives $(x_1,\ x_3,\ x_6) = (4,\ 2,\ 1)$ which is a *feasible solution*.
This is called the *base feasible solution*. Finally $(x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ x_6) = (4,\ 0,\ 2,\ 0,\ 0,\ 1)$ is a feasible solution of base $(x_1,\ x_3,\ x_6)$ but certainly not an optimal solution of $\min z(x) = 2x_1 - 3x_2 + 5x_3$.

Now let's express the score function with the *off-base variable* (x_2, x_4, x_5) = (λ_2, λ_4, λ_5). By substituting (I) and (II) in the score function, we get:

$$\begin{aligned}\min z(x) &= 14 + 2\lambda_4 - \lambda_5 \\ &= 14 + \underbrace{(0 \;\; 2 \;\; -1)}_{\Delta} \begin{pmatrix} \lambda_2 \\ \lambda_4 \\ \lambda_5 \end{pmatrix}\end{aligned}$$

where $\Delta^t = (0 \;\; 2 \;\; -1)^t$ is called the *marginal cost* or *reduced cost* of the score function $z(x)$.. The score we get with the initial solution is $\min z(x) = 14$.

12.2.2 The Simplex algorithm

12.2.2.1 Detecting the Pivot

Looking carefully at the *marginal costs*, we can see that at least one of it is negative $\Rightarrow$ it is still possible to get a better score. Indeed, by taking a positive value above 0 for λ_5 (we took 0 for every *off-base variables*) we can inevitably get a lower score since its *marginal cost* is negative.

Danzig I : one comes in:

This step is essential in the Simplex algorithm:

- If none of the *marginal costs* is < 0, the algorithm is finished. It is indeed not possible to minimize further with a value above 0 (positivity constraint) without a negative coefficient to play with.
- If several of the *marginal costs* are negative, one chooses the one with the biggest coefficient

The chosen variable λ_5 is called **the Danzig I** criteria. By taking a value different than the 0 we have fixed so far, we can ensure:

- $z(x)$ will mandatorily be lower than 14.
- x remains a feaible solution because the set of parameters (λ_2, λ_4, λ_5) givbe a family of *feasible solutions* whatever their values as long as they are ≥ 0.

Shortly put, chosing the Danzig I criteria means chosing the lambda λ with the highest negative coefficient in the score.

The Danzig I criteria will get into the *base variables*

Danzig II : and one goes out:

As we put an new variable in the *base variables*, one of those already in the *base variables* must get out and join the *off-base variables*. The way that variable is chosen is a bit of a *magic recipe*. The principle is as follows:

By putting another variable in the *base variables* and giving it a value, one might break either one of the *positivity constraints*.
In order to avoid this, we need to check the highest value the λ_5 parameter might take before one of the former base variables *base variables* (x_1, x_3, x_6) becomes negative.

Let's analyze the situation. What happens on the system regarding the *Danzig I critera* whenever it becomes significant. i.e. we considere it but let the others out, i.e. $(\lambda_2, \lambda_4) = (0, 0)$

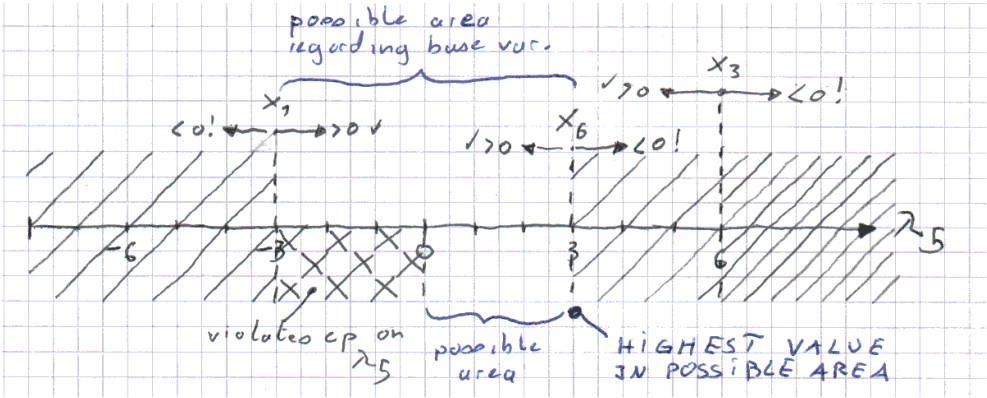
$$\begin{array}{lll} \text{From (I)} & x_1 = 2 + \frac{7}{3}\lambda_2 + \lambda_4 + \frac{1}{3}\lambda_5 = 0 & \Leftrightarrow \lambda_5 = -6 < 0 \text{ (violates p.c.)} \\ \text{From (II)} & x_3 = 2 - \frac{1}{3}\lambda_2 - \frac{1}{3}\lambda_5 = 0 & \Leftrightarrow \lambda_5 = 6 > 0 \\ \text{From (III)} & x_6 = 1 - \frac{13}{3}\lambda_2 - 4\lambda_4 - \frac{1}{3}\lambda_5 = 0 & \Leftrightarrow \lambda_5 = 3 > 0 \end{array}$$

Which we analyze this way:

- From (I), considering x_1, we get $\lambda_5 = -6$ which violates a p.c $\Rightarrow$ discarded.
- The choice remains between (II) considering x_3 or (II) considering x_6. Amongst those, we need to chose the most restrictive.
  The value that the analysis gives us for λ_5 is the maximum value that λ_5 can take before the corresponding x_i becomes negative.
  - From (II), considering x_3, the highest possible value for λ_5 is 6.
  - From (III), considering x_6, the highest possible value for λ_5 is 3.
- Hence we choose (III), considering x_6 as it is **the most restrictive variable** yet still respecting the feasible solution condition

$\lambda_5 = 3 > 0$ is the highest possible positive λ_5 minimizing the score without violating the positivity constraints on the base variables.
The corresponding variable x_6 - the one matching the most restrictive value - is **The Danzig II** criteria. The Danzig II criteria will leave the *base variables* and get into the *off-base variables*

A graphical representation of the analysis above is as follows:



Hence λ_5 is top-bounded by $x_6 \Rightarrow x_6$ is the Denzig II criteria.

12.2.2.2 Pivoting

Thanks to the above analysis, we get a new solution:

- Danzig I = λ_5 called **incoming variable** (french: variable entrante)
- Danzig II = x_6 called **outgoing variable** (french: variable sortante)

Hence x_6 becomes a parameter replacing λ_5 which becomes a variable. Considering the highest value for λ_5 which was 3, we get a new feasible solution :

$$(x_1, \lambda_2, x_3, \lambda_4, x_5, \lambda_6) = (4, 0, 2, 0, 3, 0)$$

which has better score than the previous feasible solution. We have hence progressed in our search of the optimum.
The new set of *base variables* now is (x_1, x_3, x_5) and the *off-base variables* now are (x_2, x_4, x_6) = (λ_2, λ_4, λ_6).

12.2.2.3 Reparameterization

Once the couple (x_i, λ_j) to be exchanged has been determined, the pivot needs to be propagated to the system and the score variable. Both need to be expressed in terms of the *off-base* variables.

Before pivot :

$$(S)\begin{cases} x_1 = 2 + \frac{7}{3}\lambda_2 + \lambda_4 + \frac{1}{3}\lambda_5 & (I) \\ x_3 = 2 - \frac{1}{3}\lambda_2 - \frac{1}{3}\lambda_5 & (II) \\ x_6 = 1 - \frac{13}{3}\lambda_2 - 4\lambda_4 - \frac{1}{3}\lambda_5 & (II) \end{cases}$$

After pivot :

$$(S)\begin{cases} x_1 = 2 + \frac{7}{3}\lambda_2 + \lambda_4 + \frac{1}{3}\lambda_5 & (I) \\ x_3 = 2 - \frac{1}{3}\lambda_2 - \frac{1}{3}\lambda_5 & (II) \\ \lambda_6 = 1 - \frac{13}{3}\lambda_2 - 4\lambda_4 - \frac{1}{3}x_5 & (III) \end{cases}$$

From the former line describing the *outgoing variable* x_6, i.e. (III), we extract the new *incoming variable* x_5 in terms of λ_i. Then, by substituing x_5 in (II) and (I), we get the x_i's in terms of the λ_i's.

The new system is:

$$(S)\begin{cases} x_1 = 3 - 2\lambda_2 - 3\lambda_4 - \lambda_6 & (I) \\ x_3 = 1 + 4\lambda_2 + 4\lambda_4 + \lambda_6 & (II) \\ x_5 = 3 - 13\lambda_2 - 12\lambda_4 - 3\lambda_6 & (II) \end{cases}$$

Let's do the same with the score function:

$$\begin{aligned} \text{before pivot } z(x) &= 14 + 2\lambda_4 - \lambda_5 \\ \text{after pivot } z(x) &= 14 + 2\lambda_4 - (3 - 12\lambda_2 - 12\lambda_4 - 3\lambda_6) \\ &= 11 + 13\lambda_2 + 14\lambda_4 + 3\lambda_6 \end{aligned}$$

12.2.2.4 Iteration

The algorithm then iterates. It starts over with the detection of the pivot and repeats the next steps: Danzig I, Danzig II, reparameterization, etc.

12.2.2.5 Stopping criteria

The algorithm stops when a new pivot point cannot be found, i.e. when all the marginal costs of the score function $z(x)$ are positive, i.e. when the $\Delta_i \geq 0$.

Here in our example, after this first iteration, the score is $z(x) = 11 + 13\lambda_2 + 14\lambda_4 + 3\lambda_6$. We can see that all our marginal costs are positive. The Simplex algorithm is finished because it is not possible to optimize the score any further.

The *optimal feasible solution* is extracted with every λ_i set to 0. It can be read in the last parameterization:

The solution stands in the box:

$$(S)\begin{cases}\boxed{\begin{array}{l}x_1=3\\x_3=1\\x_5=3\end{array}} & \begin{array}{ll}-2\lambda_2-3\lambda_4-\lambda_6 & (I)\\ +4\lambda_2+4\lambda_4+\lambda_6 & (II)\\ -13\lambda_2-12\lambda_4-3\lambda_6 & (II)\end{array}\end{cases}$$

Just as the score can be read in:

$$\begin{aligned}z(x)&=11+13\lambda_2+14\lambda_4+3\lambda_6\\ z(x)&=11\end{aligned}$$

12.2.2.6 Notes

- With a little luck, this partitionment with *base variables* (x_1, x_3, x_5) and *off-base variables* (x_2, x_4, x_6) = (λ_2, λ_4, λ_6) might well have been chosen is the initial *feasible solution* and help us save an iteration.
- The terminology *lambda* λ for the *off-base variables* is not always used, sometimes one keeps the x_i notation.

12.3 The Simplex algorithm

The Simplex algorithm consists in iterating the approach illustrated above, i.e. :

(E0) Find a first partitioning in *base variables* and *off-base variables* in such a way that after the parameters (the *off-base variables*) are put to 0 we get a system of equations
- squared
- solvable (i.e. the det. of the matrix is different than 0)
- with solutions respecting the positivity constraints (for the *base variables*

Note: the search of the start point is very likely the most difficult part.

(E1) Detect an incoming variable (Danzig I)
All the *off-base variables* with a negative *marginal cost* are valid candidates.
However, we'll always choose the one that has the highest possible marginal cost in absolute value.

(E2) Detect the outgoing variable (Danzig II), i.e the one that is the fasted nullified when rising the outgoing variable. The more the incoming variable rises, the more the score is minimized, the more the outgoing variable might break a positivity constraint.
We stop rising the incoming variable when the outgoing variable becomes null.

(E3) Run the pivot.
The goals is to
- Write the *base variables* in terms of the *off-base variables* (by substitution)
- Write the score function in terms of the *off-base variables* (by substitution)

(E4) Stop when it it not possible to optimize the score $z(x)$ any further, i.e. when all the marginal costs of the score Δ_i are positive or null, i.e. when it is impossible to find any new incoming variable.

12.3.1 Resumed form

The Simplex algorithm can be resumed this way:

- (A1) Find a start point
- (A2) Iterate on:
  - Danzig I *Greatest* (absolute) λ_i strictly negatif in the score $z(x)$
  - Danzig II The $x_i = 0$ with the smallest possible $\lambda_{DanzigI}$ strictly positif
  - Pivot $\lambda_{DanzigI}$ becomes $x_{DanzigI}$, $x_{DanzigII}$ becomes $\lambda_{DanzigII}$
  - Reparameterization Express all x_i and $z(x)$ in terms of λ_i
- (A3) Check the stop criteria (each λ_i positive or null in the score)

12.4 Notes

- The solution of the (LP) is mandatorily build starting with a base *feasible solution*. Depending on the size of the problem, chosing a base *feasible solution* randomly is impossible. This issue is furtherly developed in chapter 14.
- *Geometrical interpretation of the Simplex:* The Simplex is a local iterative algorithm. Every solution is on a vertex of the *polytope* formed by the hyperplans given by the constraint equations system. The pivot / reparameterization from a base to another makes us move from one vertex to an adjacent one having a better score.
  The algorithm makes us move from the base vertex to the one in its neighbourhood with the best score.

12.5 Practice

12.5.1 Exercise 5 : Algebraic Simplex

12.5.1.1 Instructions

Part I With the help of the algebraic Simplex, solve the following (LP):

$$(LP)\begin{cases}\min z(x) = -x_1 - 2x_2 \\ w.r. \left| \begin{array}{l} (S)\begin{cases} -3x_1 + 2x_2 + x_3 = 2 \\ -x_1 + 2x_2 + x_4 = 4 \\ x_1 + x_2 + x_5 = 5 \end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)} \end{array} \right. \end{cases}$$

Part II With the help of the algebraic Simplex, solve the following (LP):

$$(LP)\begin{cases}\max z(x) = x_1 + 2x_2 \\ w.r. \left| \begin{array}{l} (S)\begin{cases} -3x_1 + 2x_2 \leq 2 \\ -x_1 + 2x_2 \leq 4 \\ x_1 + x_2 \leq 5 \end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)} \end{array} \right. \end{cases}$$

Indications:

- (I1) Transform the max in min
- (I2) Transform inequalities in equalities using *slack variable* (french : *variable d'écart*)

(See section 14.2 for help on (LP) transformation)

12.5.1.2 Solution

We can solve **part I** and **part II** in one shot by introducing the *slack variables* (x_3 ,x_3, x_5) in **part II** and by searcjing for a *min* instead of a *max* by inverting the score function. This way, both parts of the instructions are equivalent

Initialization

A good approach is to try a base *feasible solution* that gives us as less work as possible. We know the slack variables are present only once in each equation of the constraints system, hence chosing them as base variables is a good choice for an initial solution.

We have 3 equations with 5 variables $\Rightarrow$ We need to take 2 variables out of the base, i.e. as parameters.

We choose hence:

$$BV = (x_3, x_4, x_5) \qquad OBV = (\lambda_1, \lambda_2)$$

This way we get the parameterized system without much work:

$$(S)\begin{cases} x_3 = 2 + 3\lambda_1 - 2\lambda_2 & (I) \\ x_4 = 4 + \lambda_1 - 2\lambda_2 & (II) \\ x_5 = 5 - \lambda_1 - \lambda_2 & (III) \end{cases}$$
$$z(\lambda) = -\lambda_1 - 2\lambda_2$$

Iteration 1

The *incoming variable* (Danzig I) is given by the highest negative coefficient associated to a λ_i in the score. here it is $-2\lambda 2$, hence λ_2 **comes in**.

For $\lambda_i = 0 (i \neq 2)$:

$$\begin{array}{lllll} (I) & x_3 = 0 & \Leftrightarrow 2 - 2\lambda_2 = 0 & \Leftrightarrow \lambda_2 = 1 \\ (II) & x_4 = 0 & \Leftrightarrow 4 - 2\lambda_2 = 0 & \Leftrightarrow \lambda_2 = 2 \\ (III) & x_5 = 0 & \Leftrightarrow 5 - \lambda_2 = 0 & \Leftrightarrow \lambda_2 = 5 \end{array}$$

The *outgoing variable* is the one that violates the first the positivity constraint when the *incoming variable* rises, i.e. the one that gives the smallest positive value for the *incoming variable*.
In our case, $\mathbf{x_3}$ **goes out**.

This leads to the following pivot:

$$\lambda_\mathbf{2} \leftrightarrow \mathbf{x_3}$$

We should now reparameterize the constraint system (S) and the score functions using the new partitioning:

$$BV = (x_2, x_4, x_5) \qquad OBV = (\lambda_1, \lambda_3)$$

We start by pivoting the variables in the former system:

$$(S)\begin{cases}\lambda_3 = 2 + 3\lambda_1 - 2x_2 & (I)\\ x_4 = 4 + \lambda_1 - 2x_2 & (II)\\ x_5 = 5 - \lambda_1 - x_2 & (III)\end{cases}$$
$$z(\lambda) = -\lambda_1 - 2x_2$$

The line (I) is straightforward (as expected since related to the outgoing variable):

$$(S)\begin{cases}x_2 = 1 + \frac{3}{2}\lambda_1 - \frac{1}{2}\lambda_3 & (I)\\ x_4 = 4 + \lambda_1 - 2x_2 & (II)\\ x_5 = 5 - \lambda_1 - x_2 & (III)\end{cases}$$
$$z(\lambda) = -\lambda_1 - 2x_2$$

The others are rewriten by injecting (I)

$$(S)\begin{cases}x_2 = 1 + \frac{3}{2}\lambda_1 - \frac{1}{2}\lambda_3 & (I)\\ x_4 = 4 + \lambda_1 - 2(1 + \frac{3}{2}\lambda_1 - \frac{1}{2}\lambda_3) & (II)\\ x_5 = 5 - \lambda_1 - (1 + \frac{3}{2}\lambda_1 - \frac{1}{2}\lambda_3) & (III)\end{cases}$$
$$z(\lambda) = -\lambda_1 - 2(1 + \frac{3}{2}\lambda_1 - \frac{1}{2}\lambda_3)$$

Which is resolved to:

$$(S)\begin{cases}x_2 = 1 + \frac{3}{2}\lambda_1 - \frac{1}{2}\lambda_3 & (I)\\ x_4 = 2 - 2\lambda_1 + \lambda_3 & (II)\\ x_5 = 4 - \frac{5}{2}\lambda_1 + \frac{1}{2}\lambda_3 & (III)\end{cases}$$
$$z(\lambda) = -2 - 4\lambda_1 + \lambda_3$$

Iteration 2

The *incoming variable* (Danzig I) is given by the highest negative coefficient associated to a λ_i in the score. here it is $-4\lambda 1$, hence λ_1 **comes in**.

For $\lambda_i = 0 (i \neq 1)$:

$$\begin{array}{lllll}(I) & x_2 = 0 & \Leftrightarrow 1 + \frac{3}{2}\lambda_1 = 0 & \Leftrightarrow \lambda_1 = -\frac{2}{3} \leq 0\\ (II) & x_4 = 0 & \Leftrightarrow 2 - 2\lambda_1 = 0 & \Leftrightarrow \lambda_1 = 1 \geq 0\\ (III) & x_5 = 0 & \Leftrightarrow 4 - \frac{5}{2}\lambda_1 = 0 & \Leftrightarrow \lambda_1 = \frac{8}{5} \geq 0\end{array}$$

The *outgoing variable* is the one that violates the first the positivity constraint when the *incoming variable* rises, i.e. the one that gives the smallest positive value for the *incoming variable*.
In our case, $\mathbf{x_4}$ **goes out**.

This leads to the following pivot:

$$\lambda_1 \leftrightarrow \mathbf{x_4}$$

We should now reparameterize the constraint system (S) and the score functions using the new partitioning:

$$BV = (x_1, x_2, x_5) \qquad OBV = (\lambda_3, \lambda_4)$$

We start by pivoting the variables in the former system:

$$(S)\begin{cases}x_2 = 1 + \frac{3}{2}x_1 - \frac{1}{2}\lambda_3 & (I)\\ \lambda_4 = 2 - 2x_1 + \lambda_3 & (II)\\ x_5 = 4 - \frac{5}{2}x_1 + \frac{1}{2}\lambda_3 & (III)\end{cases}$$
$$z(\lambda) = -2 - 4x_1 + \lambda_3$$

The line (II) is straightforward, we solve it and invert it with (I):

$$(S)\begin{cases}x_1 = 1 + \frac{1}{2}\lambda_3 - \frac{1}{2}\lambda_4 & (I)\\ x_2 = 1 + \frac{3}{2}x_1 - \frac{1}{2}\lambda_3 & (II)\\ x_5 = 4 - \frac{5}{2}x_1 + \frac{1}{2}\lambda_3 & (III)\end{cases}$$
$$z(\lambda) = -2 - 4x_1 + \lambda_3$$

The others are rewriten by injecting (I)

$$(S)\begin{cases} x_1 = 1 + \frac{1}{2}\lambda_3 - \frac{1}{2}\lambda_4 & (I) \\ x_2 = 1 + \frac{3}{2}(1 + \frac{1}{2}\lambda_3 - \frac{1}{2}\lambda_4) - \frac{1}{2}\lambda_3 & (II) \\ x_5 = 4 - \frac{5}{2}(1 + \frac{1}{2}\lambda_3 - \frac{1}{2}\lambda_4) + \frac{1}{2}\lambda_3 & (III) \end{cases}$$

$$z(\lambda) = -2 - 4(1 + \frac{1}{2}\lambda_3 - \frac{1}{2}\lambda_4) + \lambda_3$$

Which is resolved to:

$$(S)\begin{cases} x_1 = 1 + \frac{1}{2}\lambda_3 - \frac{1}{2}\lambda_4 & (I) \\ x_2 = \frac{5}{2} + \frac{1}{4}\lambda_3 - \frac{3}{4}\lambda_4 & (II) \\ x_5 = \frac{3}{2} - \frac{3}{4}\lambda_3 + \frac{5}{4}\lambda_4 & (III) \end{cases}$$

$$z(\lambda) = -6 - 1\lambda_3 + 2\lambda_4$$

Iteration 3

The *incoming variable* (Danzig I) is given by the highest negative coefficient associated to a λ_i in the score. here it is $14\lambda 3$, hence λ_3 **comes in**.

The *outgoing variable* is the one that violates the first the positivity constraint when the *incoming variable* rises, i.e. the one that gives the smallest positive value for the *incoming variable*.
In our case, $\mathbf{x_5}$ **goes out**.

For $\lambda_i = 0 (i \neq 3)$:

$$\begin{array}{lllll} (I) & x_1 = 0 & \Leftrightarrow 1 + \frac{1}{2}\lambda_3 = 0 & \Leftrightarrow \lambda_3 = -2 \leq 0 \\ (II) & x_2 = 0 & \Leftrightarrow \frac{5}{2} + \frac{1}{4}\lambda_3 = 0 & \Leftrightarrow \lambda_3 = -10 \leq 0 \\ (III) & x_5 = 0 & \Leftrightarrow \frac{3}{2} - \frac{3}{4}\lambda_3 = 0 & \Leftrightarrow \lambda_3 = 2 \geq 0 \end{array}$$

This leads to the following pivot:

$$\lambda_3 \leftrightarrow x_5$$

We should now reparameterize the constraint system (S) and the score functions using the new partitioning:

$$BV = (x_1, x_2, x_3) \qquad OBV = (\lambda_4, \lambda_5)$$

We start by pivoting the variables in the former system:

$$(S)\begin{cases} x_1 = 1 + \frac{1}{2}x_3 - \frac{1}{2}\lambda_4 & (I) \\ x_2 = \frac{5}{2} + \frac{1}{4}x_3 - \frac{3}{4}\lambda_4 & (II) \\ \lambda_5 = \frac{3}{2} - \frac{3}{4}x_3 + \frac{5}{4}\lambda_4 & (III) \end{cases}$$

$$z(\lambda) = -6 - 1x_3 + 2\lambda_4$$

The line (III) is straightforward, we solve it easily as expected:

$$(S)\begin{cases} x_1 = 1 + \frac{1}{2}x_3 - \frac{1}{2}\lambda_4 & (I) \\ x_2 = \frac{5}{2} + \frac{1}{4}x_3 - \frac{3}{4}\lambda_4 & (II) \\ x_3 = 2 + \frac{5}{3}\lambda_4 - \frac{4}{3}\lambda_5 & (III) \end{cases}$$

$$z(\lambda) = -6 - 1x_3 + 2\lambda_4$$

The others are rewriten by injecting (I)

$$(S)\begin{cases} x_1 = 1 + \frac{1}{2}(2 + \frac{5}{3}\lambda_4 - \frac{4}{3}\lambda_5) - \frac{1}{2}\lambda_4 & (I) \\ x_2 = \frac{5}{2} + \frac{1}{4}(2 + \frac{5}{3}\lambda_4 - \frac{4}{3}\lambda_5) - \frac{3}{4}\lambda_4 & (II) \\ x_3 = 2 + \frac{5}{3}\lambda_4 - \frac{4}{3}\lambda_5 & (III) \end{cases}$$

$$z(\lambda) = -6 - 1(2 + \frac{5}{3}\lambda_4 - \frac{4}{3}\lambda_5) + 2\lambda_4$$

Which is resolved to:

$$(S)\begin{cases} x_1 = 2 + \frac{1}{3}\lambda_4 - \frac{2}{3}\lambda_5 & (I) \\ x_2 = 3 - \frac{1}{3}\lambda_4 - \frac{1}{3}\lambda_5 & (II) \\ x_3 = 2 + \frac{5}{3}\lambda_4 - \frac{4}{3}\lambda_5 & (III) \end{cases}$$

$$z(\lambda) = -8 + \frac{1}{3}\lambda_4 + \frac{4}{3}\lambda_5$$

Stop

Since all the λ_i coefficients in the score $z(\lambda)$ are positive, we can not make the score any better for any new partitioning. The minimum is hence -8 and it is obtained when the *off-base variables* are 0. We get the values of the remaining x_i from (S):

(The other x_i are equal to 0 since they are the parameters)

$$\begin{aligned} x_1 &= 2 + \tfrac{1}{3}\lambda_4 - \tfrac{2}{3}\lambda_5 = 2 \\ x_2 &= 3 - \tfrac{1}{3}\lambda_4 - \tfrac{1}{3}\lambda_5 = 3 \\ x_3 &= 2 + \tfrac{5}{3}\lambda_4 - \tfrac{4}{3}\lambda_5 = 3 \end{aligned}$$

Result

$\arg\min z(x) = (x_1, x_2, x_3, x_4, x_5) = (2, 3, 2, 0, 0)$. (One can check that the constraints are well satisfied) with as core of -8.

12.5.2 Exercise 6 : Algebraic Simplex

12.5.2.1 Instructions

With the help of the algebraic Simplex, solve the following (LP):

$$(LP) \begin{cases} \max z(x) = x_1 \\ w.r. \left| \begin{array}{ll} (S) \begin{cases} x_1 - x_2 \leq 1 \\ 2x_1 - x_2 \leq 2 \\ x_1 + x_2 \leq 7 \end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)} \end{array} \right. \end{cases}$$

Indications:

- Transform max in min
- Transform inequalities in equalities usign *slack variables*

(See section 14.2 for help on (LP) transformation)

12.5.2.2 Solution

We start by transform the (LP) in standard form (see section 14.2 by introducing *slack variables* in order to transform the inequalities iinto equalities.

$$(LP) \begin{cases} \min z(x) = -x_1 \\ w.r. \left| \begin{array}{ll} (S) \begin{cases} x_1 - x_2 + x_3 = 1 \\ 2x_1 - x_2 + x_4 = 2 \\ x_1 + x_2 + x_5 = 7 \end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)} \end{array} \right. \end{cases}$$

Initialization

We have 3 equations with 5 variables $\Rightarrow$ We need to take 2 variables out of the base, i.e. as parameters.

We choose hence:

$$BV = (x_3, x_4, x_5) \qquad OBV = (\lambda_1, \lambda_2)$$

This way we get the parameterized system without much work:

$$(S) \begin{cases} x_3 = 1 - \lambda_1 + \lambda_2 & (I) \\ x_4 = 2 - 2\lambda_1 + \lambda_2 & (II) \\ x_5 = 7 - \lambda_1 - \lambda_2 & (III) \end{cases}$$

$$z(\lambda) = -1\lambda_1$$

Iteration 1

The *incoming variable* (Danzig I) is given by the highest negative coefficient associated to a λ_i in the score. here it is $-1\lambda 1$, hence λ_1 **comes in**.

For $\lambda_i = 0 (i \neq 2)$:

$$\begin{array}{llll} (I) & x_3 = 0 & \Leftrightarrow 1 - \lambda_1 = 0 & \Leftrightarrow \lambda_1 = 1 \geq 0 \\ (II) & x_4 = 0 & \Leftrightarrow 2 - 2\lambda_1 = 0 & \Leftrightarrow \lambda_1 = 1 \geq 0 \\ (III) & x_5 = 0 & \Leftrightarrow 7 - \lambda_1 = 0 & \Leftrightarrow \lambda_1 = 7 \geq 0 \end{array}$$

The *outgoing variable* is the one that violates the first the positivity constraint when the *incoming variable* rises, i.e. the one that gives the smallest positive value for the *incoming variable*.
In our case, we have the choice (either x_3 or x_4). Let's say $\mathbf{x_3}$ **goes out**.

This leads to the following pivot:

$$\lambda_1 \leftrightarrow \mathbf{x_3}$$

We should now reparameterize the constraint system (S) and the score functions using the new partitioning:

$$BV = (x_1, x_4, x_5) \qquad OBV = (\lambda_2, \lambda_3)$$

We start by pivoting the variables in the former system:

$$(S) \begin{cases} \lambda_3 = 1 - x_1 + \lambda_2 & (I) \\ x_4 = 2 - 2x_1 + \lambda_2 & (II) \\ x_5 = 7 - x_1 - \lambda_2 & (III) \end{cases}$$
$$z(\lambda) = -1x_1$$

The line (I) is straightforward (as expected since related to the outgoing variable):

$$(S) \begin{cases} x_1 = 1 + \lambda_2 - \lambda_3 & (I) \\ x_4 = 2 - 2x_1 + \lambda_2 & (II) \\ x_5 = 7 - x_1 - \lambda_2 & (III) \end{cases}$$
$$z(\lambda) = -1x_1$$

The others are rewriten by injecting (I)

$$(S) \begin{cases} x_1 = 1 + \lambda_2 - \lambda_3 & (I) \\ x_4 = 2 - 2(1 + \lambda_2 - \lambda_3) + \lambda_2 & (II) \\ x_5 = 7 - (1 + \lambda_2 - \lambda_3) - \lambda_2 & (III) \end{cases}$$
$$z(\lambda) = -1(1 + \lambda_2 - \lambda_3)$$

Which is resolved to:

$$(S) \begin{cases} x_1 = 1 + \lambda_2 - \lambda_3 & (I) \\ x_4 = -\lambda_2 + 2\lambda_3 & (II) \\ x_5 = 6 - 2\lambda_2 + \lambda_3 & (III) \end{cases}$$
$$z(\lambda) = -1 - 1\lambda_2 + \lambda_3$$

Iteration 2

The *incoming variable* (Danzig I) is given by the highest negative coefficient associated to a λ_i in the score. here it is $-1\lambda 2$, hence λ_2 **comes in**.

For $\lambda_i = 0 (i \neq 1)$:

$$\begin{array}{llll} (I) & x_1 = 0 & \Leftrightarrow 1 + \lambda_2 = 0 & \Leftrightarrow \lambda_2 = -1 \leq 0 \\ (II) & x_4 = 0 & \Leftrightarrow -\lambda_2 = 0 & \Leftrightarrow \lambda_2 = \frac{0}{-1} = 0 \geq 0 \\ (III) & x_5 = 0 & \Leftrightarrow 6 - 2\lambda_2 = 0 & \Leftrightarrow \lambda_2 = 3 \geq 0 \end{array}$$

The *outgoing variable* is the one that violates the first the positivity constraint when the *incoming variable* rises, i.e. the one that gives the smallest positive value for the *incoming variable*.
In our case, $\mathbf{x_4}$ **goes out**.

This leads to the following pivot:

$$\lambda_2 \leftrightarrow \mathbf{x_4}$$

Note: the above is an interesting case. The line (II) has been retained. This seems coherent

whith what has been introduced before. We'll see in the next iteration that this is not always the case and we'll provide the user with a rule to handle such situations.

We should now reparameterize the constraint system (S) and the score functions using the new partitioning:

$$BV = (x_1, x_2, x_5) \qquad OBV = (\lambda_3, \lambda_4)$$

We start by pivoting the variables in the former system:

$$(S)\begin{cases} x_1 = 1 + x_2 - \lambda_3 & (I) \\ \lambda_4 = -x_2 + 2\lambda_3 & (II) \\ x_5 = 6 - 2x_2 + \lambda_3 & (III) \end{cases}$$
$$z(\lambda) = -1 - 1x_2 + \lambda_3$$

The line (II) is straightforward (as expected since related to the outgoing variable):

$$(S)\begin{cases} x_1 = 1 + x_2 - \lambda_3 & (I) \\ x_2 = 2\lambda_3 - \lambda_4 & (II) \\ x_5 = 6 - 2x_2 + \lambda_3 & (III) \end{cases}$$
$$z(\lambda) = -1 - 1x_2 + \lambda_3$$

The others are rewriten by injecting (I)

$$(S)\begin{cases} x_1 = 1 + (2\lambda_3 - \lambda_4) - \lambda_3 & (I) \\ x_2 = 2\lambda_3 - \lambda_4 & (II) \\ x_5 = 6 - 2(2\lambda_3 - \lambda_4) + \lambda_3 & (III) \end{cases}$$
$$z(\lambda) = -1 - 1(2\lambda_3 - \lambda_4) + \lambda_3$$

Which is resolved to:

$$(S)\begin{cases} x_1 = 1 + \lambda_3 - \lambda_4 & (I) \\ x_2 = 2\lambda_3 - \lambda_4 & (II) \\ x_5 = 6 - 3\lambda_3 + 2\lambda_4 & (III) \end{cases}$$
$$z(\lambda) = -1 - 1\lambda_3 + \lambda_4$$

Iteration 3

The *incoming variable* (Danzig I) is given by the highest negative coefficient associated to a λ_i in the score. here it is $-1\lambda 3$, hence $\boldsymbol{\lambda_3}$ **comes in**.

The *outgoing variable* is the one that violates the first the positivity constraint when the *incoming variable* rises, i.e. the one that gives the smallest positive value for the *incoming variable*.
In our case, $\mathbf{x_5}$ **goes out**.

For $\lambda_i = 0 (i \neq 1)$:

$$\begin{array}{lllll} (I) & x_1 = 0 & \Leftrightarrow 1 + \lambda_3 = 0 & \Leftrightarrow \lambda_3 = -1 \leq 0 \\ (II) & x_2 = 0 & \Leftrightarrow 2\lambda_3 = 0 & \Leftrightarrow \lambda_3 = \frac{0}{+2} = 0 \geq 0 \\ (III) & x_5 = 0 & \Leftrightarrow 6 - 3\lambda_3 = 0 & \Leftrightarrow \lambda_3 = 2 \geq 0 \end{array}$$

This leads to the following pivot:

$$\boldsymbol{\lambda_3 \leftrightarrow x_5}$$

Note: Following what has been said after the previous iteration, we can see that we have here a similar case, i.e. $\lambda_3 = 0$. *However, contrary to what we have done at the previous iteration,* we don't chose here the λ_3 parameter.
The rule is as follows:

- If the resolution of the equation lead to -0, we take that parameter.
- If the resolution of the equation lead to $+0$, we **do not** take that parameter.

When the denominator is **negative**, one has to take that parameter as *outgoing variable*. When the denominator is **positive**, one has to take another variable.

One can actually confirm the relevance of this rule by seing that chosing x_5 actually is the best choice in this case. Chosing the line (III) makes us make a bigger step **without violating the p.c. on** x_2, as confirmed by:

$$x_2 = 2\lambda_3 - \lambda_4 \qquad \text{with } \lambda_i = 0 (i \neq 3) \text{ and } \lambda_3 = 2, \text{ we get:}$$
$$x_2 = +2(2) - 0 = +4 \geq 0$$

We should now reparameterize the constraint system (S) and the score functions using the new partitioning:

$$BV = (x_1, x_2, x_3) \qquad OBV = (\lambda_4, \lambda_5)$$

We start by pivoting the variables in the former system:

$$(S) \begin{cases} x_1 = 1 + x_3 - \lambda_4 & (I) \\ x_2 = 2x_3 - \lambda_4 & (II) \\ \lambda_5 = 6 - 3x_3 + 2\lambda_4 & (III) \end{cases}$$
$$z(\lambda) = -1 - 1x_3 + \lambda_4$$

The line (II) is straightforward (as expected since related to the outgoing variable):

$$(S) \begin{cases} x_1 = 1 + x_3 - \lambda_4 & (I) \\ x_2 = 2x_3 - \lambda_4 & (II) \\ x_3 = 2 + \frac{2}{3}\lambda_4 - \frac{1}{3}\lambda_5 & (III) \end{cases}$$
$$z(\lambda) = -1 - 1x_3 + \lambda_4$$

The others are rewriten by injecting (I)

$$(S) \begin{cases} x_1 = 1 + (2 + \frac{2}{3}\lambda_4 - \frac{1}{3}\lambda_5) - \lambda_4 & (I) \\ x_2 = 2(2 + \frac{2}{3}\lambda_4 - \frac{1}{3}\lambda_5) - \lambda_4 & (II) \\ x_3 = 2 + \frac{2}{3}\lambda_4 - \frac{1}{3}\lambda_5 & (III) \end{cases}$$
$$z(\lambda) = -1 - 1(2 + \frac{2}{3}\lambda_4 - \frac{1}{3}\lambda_5) + \lambda_4$$

Which is resolved to:

$$(S) \begin{cases} x_1 = 3 - \frac{1}{3}\lambda_4 - \frac{1}{3}\lambda_5 & (I) \\ x_2 = 4 + \frac{1}{3}\lambda_4 - \frac{2}{3}\lambda_5 & (II) \\ x_3 = 2 + \frac{2}{3}\lambda_4 - \frac{1}{3}\lambda_5 & (III) \end{cases}$$
$$z(\lambda) = -3 + \frac{1}{3}\lambda_4 + \frac{1}{3}\lambda_5$$

Stop

Since all the λ_i coefficients in the score $z(\lambda)$ are positive, we can not make the score any better for any new partitioning. The minimum is hence -8 and it is obtained when the *off-base variables* are 0. We get the values of the remaining x_i from (S):

(The other x_i are equal to 0 since they are the parameters)

$$x_1 = 3 - \frac{1}{3}\lambda_4 - \frac{1}{3}\lambda_5$$
$$x_2 = 4 + \frac{1}{3}\lambda_4 - \frac{2}{3}\lambda_5$$
$$x_3 = 2 + \frac{2}{3}\lambda_4 - \frac{1}{3}\lambda_5$$

Result

$\arg\min z(x) = (x_1, x_2, x_3, x_4, x_5) = (2, 3, 2, 0, 0)$. (One can check that the constraints are well satisfied) with a score of -3.

CHAPTER 13

Tabular Approach - The *Simplex* algorithm

Contents

13.1 Purpose

The objectif of the *tabular* approach is to provide a quicker form (and easier to implement as an algorithm) of resolution for the Simpley. The only issue is that it is even more a magic recipe than the algebraic way.

One should just keep in mind that it is actually exactly the same that the algebraic way except it happens in a shorter way:

- the variable names are replaced by matrix indices
- The pivot and reparameterization happen by *elementary row operations* on the matrix.

13.2 Illustration example

We'll use the following (LP) as an illustration example:

$$(LP)\begin{cases}\min z(x) = 2x_1 - 3x_2 + 4x_3 - x_4 + 0x_5 + 0x_6 + 0x_7 \\ w.r. \left|\begin{array}{ll} (S)\begin{cases} x_1 - x_2 + 2x_3 - 2x_4 + 1x_5 + 0x_6 + 0x_7 = 7 \\ -5x_1 + 3x_2 - 2x_3 + x_4 + 0x_5 + 1x_6 + 0x_7 = 1 \\ 3x_1 + x_2 + 4x_3 + 5x_4 + 0x_5 + 0x_6 + 1x_7 = 9 \end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)} \end{array}\right. \end{cases}$$

13.2.1 base feasible solution

Let's take:

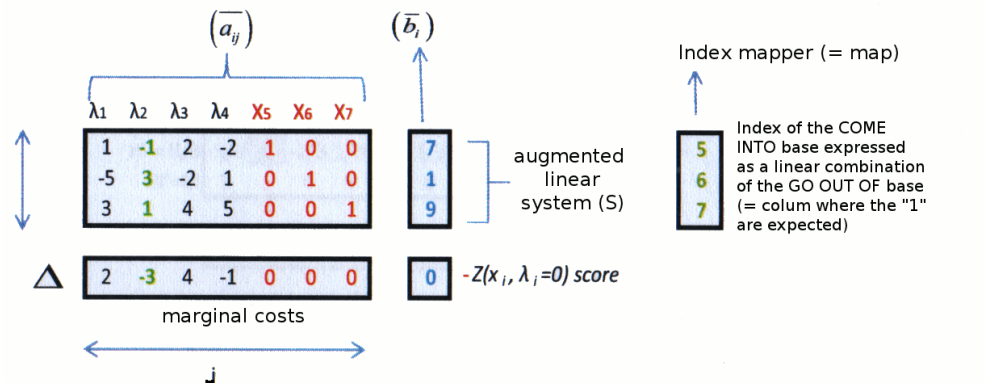
$$(x_5, x_6, x_7) \text{ as } \textit{base variables}$$
$$(x_1, x_2, x_3, x_4) \text{ as } \textit{off-base variables}$$

One should not that this choice is not anectodic:

- the score function $z(x)$ is already almost expressed in a linear combination of the *off-base variables*.
- The *bae variables* are already almost expressed in terms of the *off-base variables*.

One should also not that this is a *demo case*, in practice the choice of the base is pretty tricky and a specific field on its own.

The simplex tabular associated with the illustration case posess an *initial feasible solution* the following partioning:



Notes:

- One should note the identity matrix in the red part of the base variables. There is nothing to compute, the coefficients depend directly from the choice of the *base variables*.
- Also note the null marginal costs.
- The map is useful to avoid the need to search for the "1" on the row.

13.2.2 Iteration 1

13.2.2.1 Danzig Formulas

Let m be the *number of variables* and n be the *number of constraints*

Danzig I criteria - the *incomming variable* is λ_{j_e} with

$$j_e = \underset{j\in[1,m]}{\arg\min}(\Delta_j < 0) \quad \text{with column index = variable index } \in [1,m]$$

Danzig II criteria - the *outgoing variable* is x_{i_s} with

$$i_s = \underset{i\in[1,n]}{\arg\min}\left(\frac{\bar{b_i}}{\bar{a_{ij_e}}} \geq 0\right) \qquad \text{with } \bar{a_{ij_e}} > 0 \text{ line index } \in [1,n]$$

$$j_s = map[i_s] \qquad \text{with varable index } \in [1,m]$$

- map is a function that gives the index of the *base variable* j associated to the line i, where associated means *"on this line, x_j is expressed as a linear combination of the off-base variables"*.
- Whenever all the $\Delta_j \geq 0$, the stop criteria is reached

13.2.2.2 Danzig I criteria

The *incomming variable* is the one with the *highest negative marginal cost*.

The coefficients of the λ_i parameters are read in the last line of the tabular. In the tabular above, the parameter with the highest negative coefficient is λ_2.

Hence the *incomming variable*, i.e. the Danzig I criteria is λ_2 with $\Delta(\lambda_2) = -3$.

13.2.2.3 Danzig II criteria

The *outgoing variable* is the one that is the most restrictive for the *incomming variable* to come in. The way this variable is extracted consists in putting all coefficient to 0 in the equation system and keeping only the constants as well as the coefficients of the Danzig I critera, i.e. the *incomming variable*.

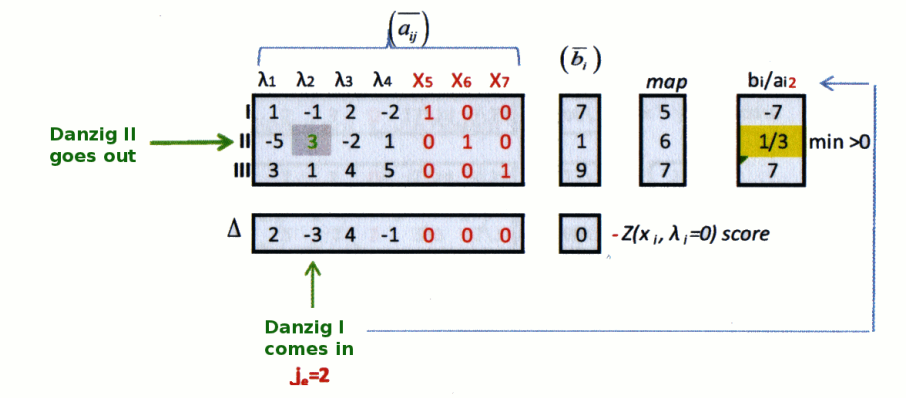
(Recall Danzig II presentation in 12.2.2.1)

One should not that resolving the equations in terms of the Danzig I criteria by putting each and every coefficient at 0 except the ones of the *incomming variable* and the constants is like considering only the column of the Danzig I parameter and the column of the constants b.

One can build a new column into the array giving directly the searched result for $\bar{b_i}/\bar{a_{ij}}$ as shown on the next reepresentation on the tabular. For Danzig II, we are looking for the *most restrictive* (i.e. smallest positive) ratio $\bar{b_i}/\bar{a_{ij}}$.
(if it is null, then $\bar{a_{ij}}$ must be strictly positive)

Hence the *outgoinf variable*, i.e. the Danzig II criteria is x_6.

13.2.3 Pivot Point



13.2.3.1 Reparameterization

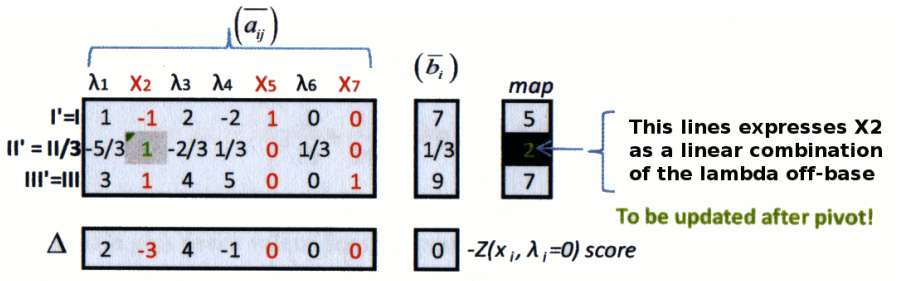
As an equation sytem doesn't wary when basic lines operations are applied, we are enabled to perform the reparameterization using these operations (see 10.3.1).

Just recall the initial situation:

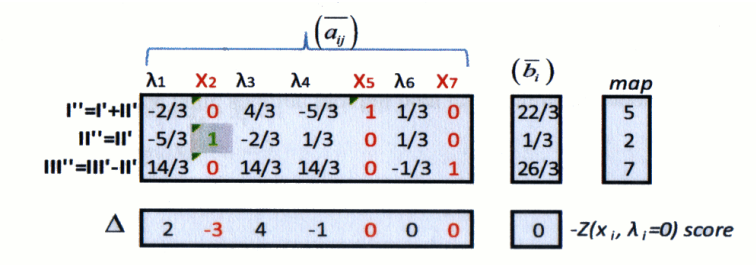
- The identity matrix in the simplex tabular under the *base variables*
- Null marginal costs in the score under the *base variables*

The goal is to reestablish this situation after the base change, i.e. these two properties needs to be respected under the *base-variables* (x_2, x_5, x_7)

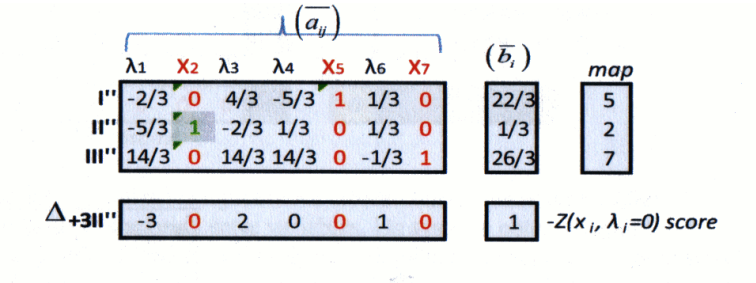
Step 1 : the pivot should be 1. Divide the line by 3:



Step 2 : We want 0 everywhere else in the column of the pivot:



Step 3 : Express marginal costs exclusively with *off-base* variables:

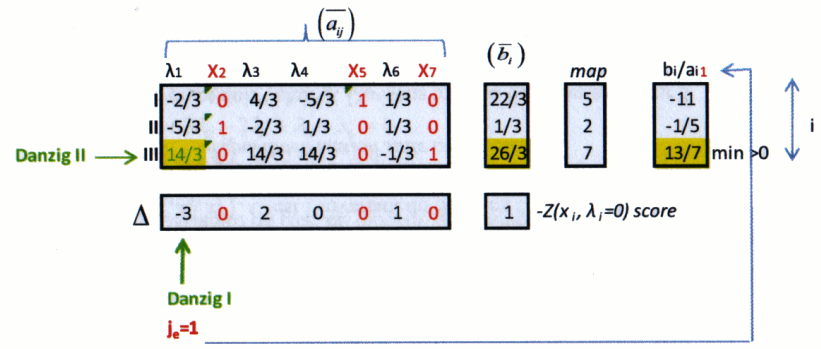


The reparameterization is finished, we can start a new iteration since there are still negative *marginal costs* in the score.
One should note that a maximization problem would considere a stop criteria when the marginal costs, therefor named *marginal profits* are all negatives or null.

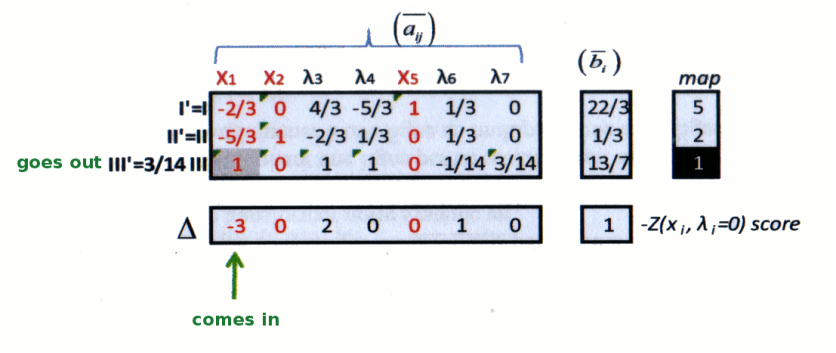
13.2.4 Iteration 2

We'll run the iteration 2 a bit faster since we have now seen the principles.

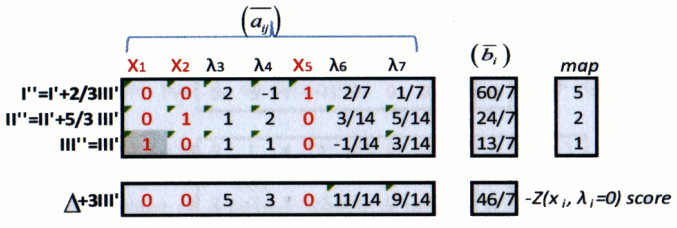
Here as $\Delta(\lambda_1) = -1 < 0$, we can still minimize the score $z(x)$ further by changing the base. The Danzig I criteria λ_1 *comes in*.
Then we compute a new $\bar{b_i}/\bar{a_{ij}}$ column and find out the Danzig II critera, x_7 *goes out.*



Step 1 : the pivot should be 1.



Step 2 and 3 : We want 0 everywhere else in the column of the pivot and the marginal costs should be expressed exclusively with *off-base* variables:



13.2.5 Stop and results

As we do not have any more negative coefficient in the score after the reparameterization, the algorithm stops. We have found the optimum vertex of the polytop.

The cell $z(x)$ (the cell on the bottom-most right-most part of the tabular), i.e. the intersection of the $(\bar{b_i})$ column and the Δ row contains **the opposite of the result** with:

$$\lambda_i = 0$$
$$x_{mapBase[i]} = \bar{b_i}$$

i.e. the values for the *base variables* x_j are given by the column $(\bar{b_i})$ and the values of the *off-base variables* are all 0.

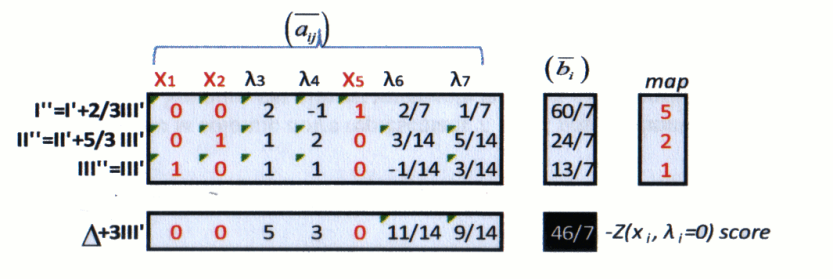
Solution :

$$x_1 = \frac{13}{7} \quad x_2 = \frac{24}{7} \quad x_3 = 0 \quad x_4 = 0 \quad x_5 = \frac{60}{7} \quad x_6 = 0 \quad x_7 = 0$$
$$\min z(x) = -\frac{46}{7}$$

One can check that the solution indeed respects the constraints system.

13.3 Why does the tabular contrain *the opposite of the score* ?

In our example the *tabular Simplex* stops in the following situation :



In the dark cell is the **opposite of the score** and not the score, why ?

13.3.1 Representaiton of the initial LP

The initial (LP) was:

$$(LP)\begin{cases}\min z(x) = 2x_1 - 3x_2 + 4x_3 - x_4 + 0x_5 + 0x_6 + 0x_7 \\ w.r. \left|\begin{array}{l}(S)\begin{cases}x_1 - x_2 + 2x_3 - 2x_4 + x_5 = 7 \\ -5x_1 + 3x_2 - 2x_3 + x_4 + x_6 = 1 \\ 3x_1 + x_2 + 4x_3 + 5x_4 + x_7 = 9\end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)}\end{array}\right.\end{cases}$$

13.3.2 Integrating the score into the constraint system

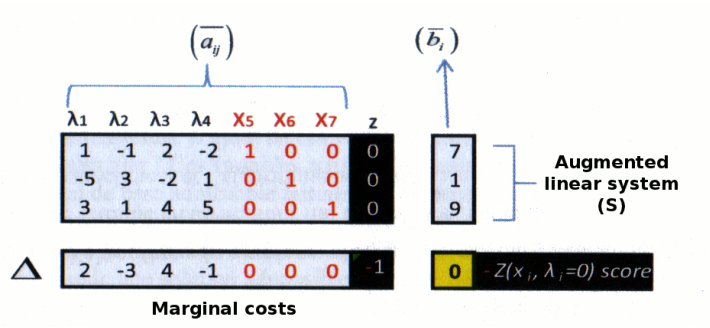
The trick consists in considering $z(x)$ as an additional variable. The relation between $z(x)$ can hence be written:

$$2x_1 - 3x_2 + 4x_3 - x_4 + 0x_5 + 0x_6 + 0x_7 - z = 0$$

Which can then be integrated in the contraint system S as a constraint binding z to thre other variables

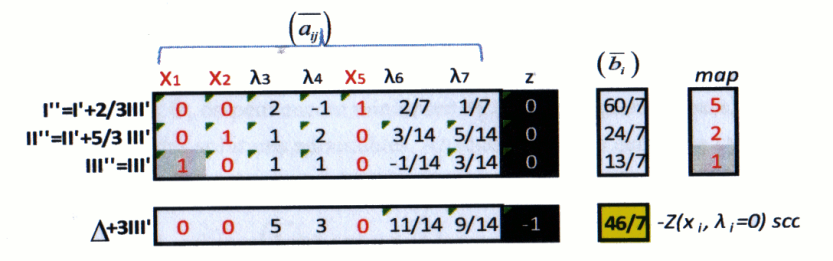
$$(S)\begin{cases}x_1 - x_2 + 2x_3 - 2x_4 + x_5 = 7 \\ -5x_1 + 3x_2 - 2x_3 + x_4 + x_6 = 1 \\ 3x_1 + x_2 + 4x_3 + 5x_4 + x_7 = 9 \\ 2x_1 - 3x_2 + 4x_3 - x_4 + 0x_5 + 0x_6 + 0x_7 - z = 0\end{cases}$$

The associated Simplex tabular hence is:



Thanks to this trick, all the lines of the tabular have the same status and can be manipulated together using *elementary row operations*.
The weigth of z is **-1** which explains why one needs to considere the opposite of the score in the yellow cell..

Hence, when the simplex is finished:



the score is $-46/7$ and not $46/7$ because the last lin of the simpley resolves to the following equation:

$$0x_1 + 0x_2 + 5\lambda_3 + 3\lambda_4 + 0x_5 + \frac{11}{4}\lambda_6 + \frac{9}{14}\lambda_7 - z = \frac{46}{7} \qquad \text{with} \lambda_i = 0 \Rightarrow$$
$$-z = \frac{46}{7} \Rightarrow z = -\frac{46}{7}$$

13.4 Convergence

13.4.0.1 Degeneration

The *feasible initial solution* **degenerates** whenever at least one of the base variable takes the value 0. In other terms, an initial solution degenerates whenever at lest of the of the component of the $\bar{b_i}$ vector is 0.

13.4.0.2 Convergence and cycling

If during the Simplex algorithm each and every of the encountered base is not degenerated, then the algorithm ends with a finished number of iterations. Otherwise, a cacling process might

appear sometimes and the algorithm might never converge.

Anti-cycling measures

There is a simple mesure against the cycling issue. It however decreases significantly the performances and needs thus to be used with great care:

- **Danzig I** takes **the littlest index** for which the *marginal cost* is negative
- **Danzig II** takes **the littlest index** in case of equality.

13.5 Practice

13.5.1 Exercise 7 : Simplex Tabular approach

Solve the follofing (LP) using the tabular algorithm for the Simplex:

$$(LP)\begin{cases}\max z(x) = x_1 + 2x_2 \\ w.r. \quad \left|\begin{array}{ll} (S)\begin{cases}-3x_1 + 2x_2 \leq 2 \\ -x_1 + 2x_2 \leq 4 \\ x_1 + x_2 \leq 5\end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)}\end{array}\right.\end{cases}$$

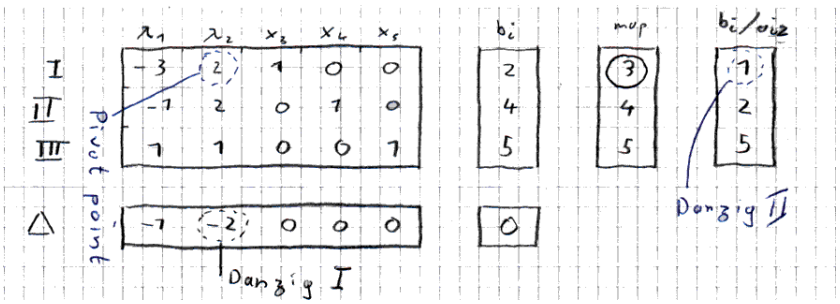
First we need to convert it under standard form:

$$(LP)\begin{cases}\min z(x) = -x_1 - 2x_2 \\ w.r. \quad \left|\begin{array}{ll} (S)\begin{cases}-3x_1 + 2x_2 + x_3 = 2 \\ -x_1 + 2x_2 + x_4 = 4 \\ x_1 + x_2 + x_5 = 5\end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)}\end{array}\right.\end{cases}$$

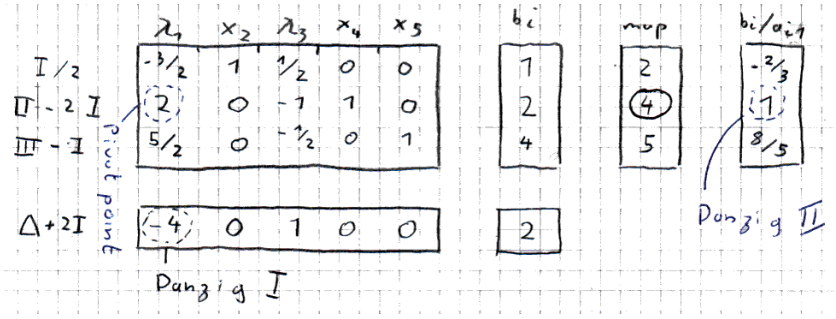
(See section 14.2 for help on (LP) transformation)

13.5.1.1 Solution

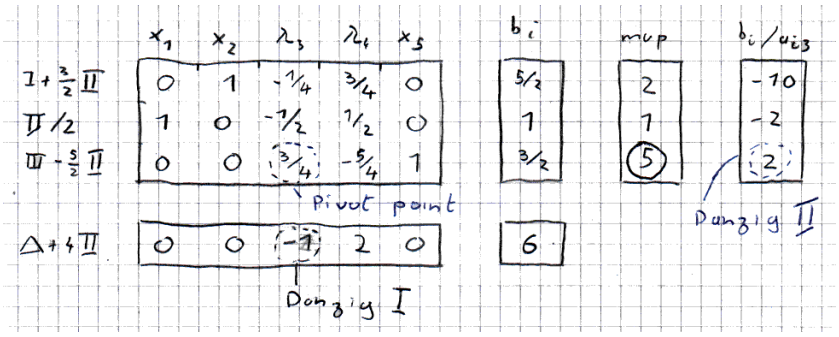
Tabular representation + Danzig Iteration 1



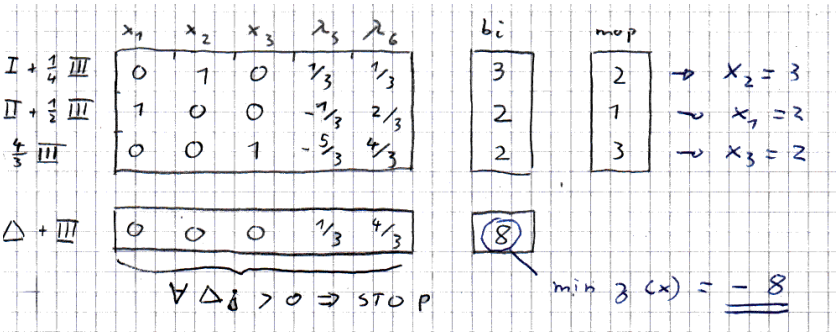
Reparameterization iteration 1 + Danzig Iteration 2



Reparameterization iteration 2 + Danzig Iteration 3



Reparameterization iteration 3



Which gives us as result:

$$\arg\min z(x) = (2, 3, 2, 0, 0, 0)$$
$$\min z(x) = -8$$

13.5.2 Exercise 8 : Simplex Tabular approach

Solve the follofing (LP) using the tabular algorithm for the Simplex:

$$(LP)\begin{cases}\max z(x) = x_1 \\ w.r. \left| \begin{array}{l} (S)\begin{cases} x_1 - x_2 \leq 1 \\ 2x_1 - x_2 \leq 2 \\ x_1 + x_2 \leq 7 \end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)} \end{array} \right. \end{cases}$$

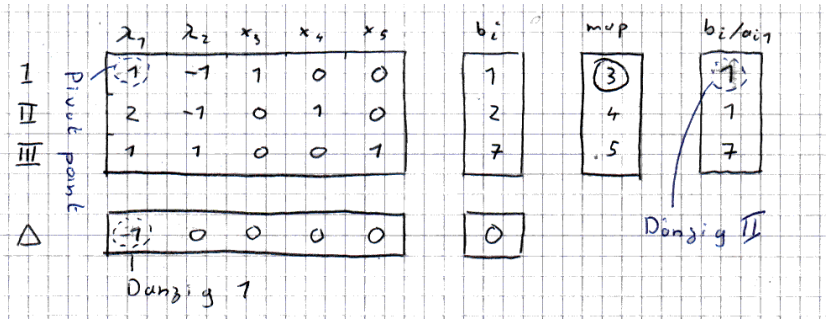
First we need to convert it under standard form:

$$(LP)\begin{cases}\min z(x) = -x_1 \\ w.r. \left| \begin{array}{l} (S)\begin{cases} x_1 - x_2 + x_3 = 1 \\ 2x_1 - x_2 + x_4 = 2 \\ x_1 + x_2 + x_5 = 7 \end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)} \end{array} \right. \end{cases}$$

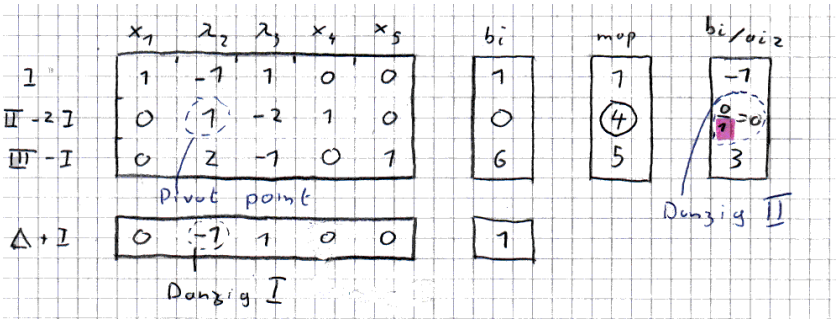
(See section 14.2 for help on (LP) transformation)

13.5.2.1 Solution

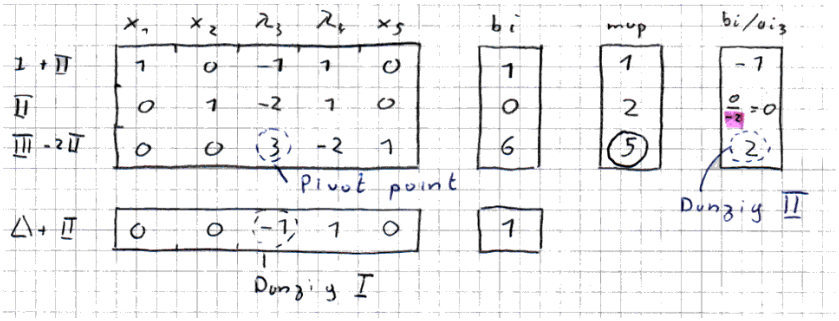
Tabular representation + Danzig Iteration 1



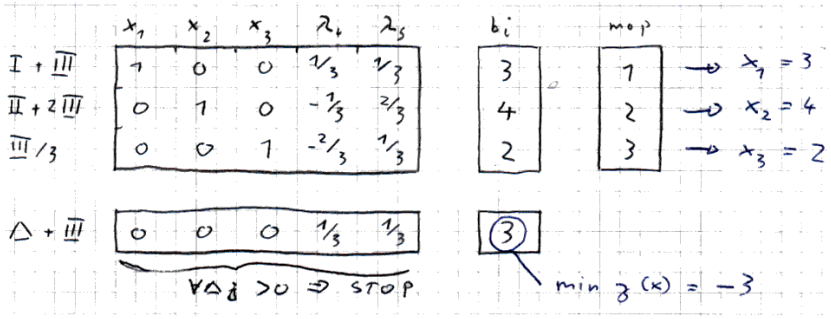
Reparameterization iteration 1 + Danzig Iteration 2



Reparameterization iteration 2 + Danzig Iteration 3



Reparameterization iteration 3



Which gives us as result:

$$\arg\min z(x) = (3, 4, 2, 0, 0, 0)$$
$$\min z(x) = -3$$

CHAPTER 14

Simplex - Additional concerns

Contents

14.1 Lack of a feasible initial solution

14.1.1 Motivation

One should note that the simpley might be an iterative algorithm, it still need a *feasible initial solution* to start. In practice, chosing a feasible initial solution is far from easy.

Whenever an initial solution doesn't appear easily, one needs to chose a solution amongst the $(n\ \ m)^t$ possible base one that is feasible. That search can be very long.
There are m variables and and n constraints with $m > n$, one need to chose n base variables amongst the m available hoping the squared sub-system of linear equations posess a solution, which will be the case whenever the matrixs is inversible. That inversibility property is very costy to check and doesn't apply to all the bases, far from it!

For this reason, there is an algorithm that helps finding a base feasible solution.

14.1.2 The artificial variables algorithm

14.1.2.1 Idea

The idea is to put the (LP) system in a larger dimensions space by adding artificial variables in such a way that a feasible initial solution becomes obvious. The augemented (LP) will need to

posess strong relations with the initial (LP) in order for the work performed on the augmented system to be appliable on the initial system.
For instance, whenever the augmented (LP) posesses a feasible initial solution where all artificial variables are null (0) then thre relation becomes obvious.

14.1.2.2 The *two-phase* algorithm

Let's considere the following (LP):

$$(LP)\begin{cases}\min z(x) = \sum_{j=1}^{m} c_j x_j \\ w.r. \left| \begin{array}{l} \sum_{j=1}^{m} a_{ij} x_j = b_i \\ x_j \geq 0 \quad \text{(positivity constraints)} \end{array} \right.\end{cases}$$

So we build the following augmented (LP)

$$(LP^A)\begin{cases}\min \sum_{i=1}^{n} v_i \\ w.r. \left| \begin{array}{l} \sum_{j=1}^{m} a_{ij} x_j + v_i = b_i \\ x_j \geq 0, v_i \geq 0 \quad \text{(positivity constraints)} \end{array} \right.\end{cases}$$

Let's asume this (LP) doesn't have an obvious *feasible initial solution*

Each constraint is augmented by an artificial variable v_i and the function to be minimized has been replaced by the sum of all artificial variables. This choice of augmented (LP) is based on the following property:

(LP) pos. a feasible initial solution $\Leftrightarrow$ (LP^A) pos. a feasible optimal solution where $\sum_{i=1}^{n} v_i = 0$

One should note that this doesn't provide an optimal solution for (LP), only a feasible initial solution.

14.1.2.3 Caution

An (LP) doesn't posess any feasible initial solution if

- The feasible optimum solution if $(LP^A$ doesn't have a null score (=0).
- The base variables of the optimal solution of $(LP^A$ contains any artificial variable. In this case one might want to try additionnal pivots.

14.2 (LP) transformations

14.2.1 Limitations

The Simplex algorithm we have seen is only able to handle (LP) under **Standard form**. We have often need to transform the given problem intro its *standard form* before being able apply the algorithm on it.

The standard form of an (LP) is as follows:

$$(LP)\begin{cases}\min z(x) = \sum_{j=1}^{m} c_j x_j \\ w.r. \left| \begin{array}{ll} \sum_{j=1}^{m} a_{ij} x_j = b_i & i \in [1,n] \text{ with } b_i > 0 \\ x_j \geq 0 \quad \text{(positivity constraints)} & \end{array} \right.\end{cases}$$

This can seem very limited at first, due to the lack fo support for *greater than* or *smaller than* constraints and the support of minimization problems only. But it actually is not since there are parades to handle the other cases.

14.2.2 parades

For the situation that doesn't match the *standard form*, there are technics to be used to bring the problem to its standard form.

14.2.2.1 Slack variables

(in french: *variables d'écarts*)

A *greater than* constraint:

Let's imagine a constraint of (S) under a non-standard form, pour the i-th constraint:

$$\sum_{j=1}^{m} a_{ij} xj \geq b_i \quad \text{(non standard form)}$$

In order to get it under standars form, one simply adds a *slack variable* s_i

$$\begin{cases} \sum_{j=1}^{m} a_{ij} xj - s_i = b_i & \text{(standard form)} \\ s_i > 0 \end{cases}$$

This is possible because as we have seen:

$$a \geq b \Leftrightarrow a - \varepsilon = b \text{ with } \varepsilon \geq b$$

A *smaller than* constraint:

The same thing applies for the following case:

$$\sum_{j=1}^{m} a_{ij} xj \leq b_i \quad \text{(non standard form)}$$

In order to get it under standars form, one simply adds a *slack variable* s_i

$$\begin{cases} \sum_{j=1}^{m} a_{ij} xj + s_i = b_i & \text{(standard form)} \\ s_i > 0 \end{cases}$$

This is possible because as we have seen:

$$a \geq b \Leftrightarrow a - \varepsilon = b \text{ with } \varepsilon \geq b$$

14.2.2.2 Maximization problem

In case we are facing a maximization problem instead of a minimization problem, there are two strategies:

Strategy 1:

One simply applies the Simplex as usual on the opposite of f, then invert the sign of the solution, because

$$\boxed{\max(f) = -\min(-f)} \qquad \boxed{\arg\max(f) = \arg\min(-f)}$$

Strategy 2:

One can also change the Danzig I criteria and take as *incoming variable* the one that has the highest positive *marginal profit*. Whenever all *marginal profits* are positive or null, the algorithm is finished.

14.2.2.3 Lack of a positivity constraint

Whenever one of the x_i variable doesn't posess a *positivity constraint*, it is replaced by two variables, each of them posessis a *positivity constraint*:

$$x_i \in \mathbb{R} \Leftrightarrow \begin{cases} x_j = u_j - v_j \\ u_j \geq 0 \\ v_j \geq 0 \end{cases}$$

14.2.2.4 Number b_i negative

Whenever one of the constraint has the form:

$$\sum_{j=1}^{m} a_{ij} x_j \geq b_j \text{ with } b_j < 0 \quad \text{(non standard form)}$$

One can simply multiply the inequation by -1 and the right-side member becomes positive

$$-\sum_{j=1}^{m} a_{ij} x_j \leq -b_j$$

14.2.3 Canonical form - definition

An (LP) in **Canonical form** is as follows:

$$(LP)\begin{cases}\min z(x) = \sum_{j=1}^{m} c_j x_j \\ w.r. \left| \begin{array}{ll} \sum_{j=1}^{m} a_{ij} x_j \geq b_i & i \in [1,n] \text{ with } b_i > 0 \\ x_j \geq 0 \quad \text{(positivity constraints)} & \end{array} \right.\end{cases}$$

In order to resolve an (LP) in canonical form, one simply needs transform in under **standard form** by adding *slack variables*. These *slack variables* doesn't appear in the score but only *one time each* in the constraints. As such they can be part of a feasible initial solution of the standard (LP)

14.3 Simplex using R

Solving the simplex with R happens this way:

Let's assume we face a problem expressed this way:

$$(LP)\begin{cases}\max \left| \min z(x) = a^t \cdot x \right. \\ w.r. \left| \begin{array}{ll} (S)\begin{cases} A1 \cdot x \leq b1^t \\ A2 \cdot x \geq b2^t \\ A3 \cdot x = b3^t \end{cases} & \\ x_i \geq 0 \text{ (p.c.)} & \end{array} \right.\end{cases}$$

This problem is solved with R using the following call:

```
simplex(a,
        A1 = NULL, b1 = NULL,
        A2 = NULL, b2 = NULL,
        A3 = NULL, b3 = NULL,
        maxi = FALSE,
        n.iter = n + 2 * m,
        eps = 1e-10)
```

The parameters have the following meanings

a	A vector of length n which gives the coefficients of the objective function.
A1	An m1 by n matrix of coefficients for the "<=" type of constraints.
b1	A vector of length m1 giving the right hand side of the "<=" constraints. This argument is required if A1 is given and ignored otherwise. All values in b1 must be non-negative.
A2	An m2 by n matrix of coefficients for the ">=" type of constraints.
b2	A vector of length m2 giving the right hand side of the ">=" constraints. This argument is required if A2 is given and ignored otherwise. All values in b2 must be non-negative. Note that the constraints $x >= 0$ are included automatically and so should not be repeated here.
A3	An m3 by n matrix of coefficients for the equality constraints.
b3	A vector of length m3 giving the right hand side of equality constraints. This argument is required if A3 is given and ignored otherwise. All values in b3 must be non-negative.

maxi	A logical flag which specifies minimization if FALSE (default) and maximization otherwise. If maxi is TRUE then the maximization problem is recast as a minimization problem by changing the objective function coefficients to their negatives.
n.iter	The maximum number of iterations to be conducted in each phase of the simplex method. The default is $n + 2 * (m1 + m2 + m3)$.
eps	The floating point tolerance to be used in tests of equality.

14.4 Practice

14.4.1 Exercise 4 : transformation

Transform the following (LP)s under *standard form*

14.4.1.1 Part I

Original (LP)

$$(LP)\begin{cases}\min z(x) = 6x_1 + 4x_2 \\ w.r. \left| \begin{array}{l} (S)\begin{cases} 3_1 + 2x_2 \geq 18 \\ 2x_1 + 4x_2 = 20 \\ 2x_2 \leq 8 \end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)} \end{array} \right. \end{cases}$$

standard form:

$$(LP)\begin{cases}\min z(x) = 6x_1 + 4x_2 \\ w.r. \left| \begin{array}{l} (S)\begin{cases} 3_1 + 2x_2 - s_1 = 18 \\ 2x_1 + 4x_2 = 20 \\ 2x_2 + s_2 = 8 \end{cases} \\ x_i \geq 0, s_j \geq 0 \quad \text{(p.c.)} \end{array} \right. \end{cases}$$

14.4.1.2 Part II

Original (LP)

$$(LP)\begin{cases}\max z(x) = x_1 + 2x_2 \\ w.r. \left| \begin{array}{l} (S)\begin{cases} 3x_1 + 2x_2 \leq 40 \\ x_1 - x_2 \geq 30 \end{cases} \\ x_i \geq 0 \quad \text{(positivity constraints)} \end{array} \right. \end{cases}$$

standard form:

$$(LP)\begin{cases}\min z(x) = -x_1 - 2(u_1 - v_1) \\ w.r. \left| \begin{array}{l} (S)\begin{cases} 3x_1 + 2(u_1 - v_1) + s_1 = 40 \\ x_1 - (u_1 - v_1) - s_2 = 30 \end{cases} \\ x_i \geq 0, s_j \geq 0, u_1 \geq 0, v_1 \geq 0 \quad \text{(p.c.)} \end{array} \right. \end{cases}$$

Part III

Integer linear Programming

CHAPTER 15

Integer Linear programming

Contents

15.1 Introduction

Thus far we have been dealing with models in which the variables can take on real values, for example a solution value of 7.3 is perfectly fine. But the variables in some models are restricted to taking only integer or discrete values. You can assign 6 or 7 people to a team, for example, but not 6.3 people; or you can choose to make a transistor from silicon dioxide or gallium arsenide, but not some mixture

If the unknown variables are all required to be integers, then the (LP) problem is called an *integer programming* (IP) or *integer linear programming* (ILP) problem.
In contrast to linear programming, which can be solved efficiently in the worst case, integer programming problems are in many practical situations (those with bounded variables) *NP-hard*. If only some of the unknown variables are required to be integers, then the problem is called a *mixed integer programming* (MIP) problem. These are generally also NP-hard.

There are however some important subclasses of IP and MIP problems that are efficiently solvable, most notably problems where the constraint matrix is totally unimodular and the right-hand sides of the constraints are integers.

Advanced algorithms for solving integer linear programs include:

- branch and bound
- cutting-plane method
- branch and cut

15.1.1 Example

We'll see now a typical example of a problem expressed as an (ILP).

A company provides its client with various *raw materials* it can *cut* under several *forms*. The clients are placing order to the company indicating the various *forms* and the quantities of each *forms* it wants.

The variables are as follows:

- There are a total of m commands
- There are a total of n forms
- The amount of pieces (# forms $\times$ # pieces by form) is $q_j(j = 1..m)$
- The amount of form i is $x_i(i = 1..n)$
- The cost to manufacture form i is $c_i(1 = 1..n)$
- the amount of pieces for form i in command j is $a_{ij}(i = 1..m, j = 1..n)$

The problem is formalized this way:

$$(ILP)\begin{cases}\min z(x) = \sum_{i=1}^{b} c_i x_i \\ w.r. \left| \begin{array}{l} \sum_{i=1}^{n} a_{ij} x_j \geq q_j \ \ \forall j = 1..m \\ x_i \geq 0 \ \ \forall i = 1..n \end{array} \right. \end{cases}$$ (positivity constraints)

Other example involves:

- Affectation of frequencies in broadband mobile network
- Affectations of flight lines to airway companies
- ...

15.2 Differences with (LP)

15.2.1 Different results

(ILP) problems cannot be solve as easily as (LP) problems. In addition, solving the same problem with *rationnal numbers* or *integer numbers* can lead to significantly different results. Let's see an illustration:

Considere the following problem:

$$(LP)\begin{cases}\max z(x) = x_1 + x_2 \\ w.r. \left| \begin{array}{l} (S)\begin{cases} -2x_1 + 2x_2 \geq 1 \\ -8x_1 + 10x_2 \leq 13 \end{cases} \\ x_i \geq 0 \end{array} \right. \end{cases}$$ (positivity constraints)

Solving the problem in *rationnal number* yields:

$$x_1 = 4, x_2 = \frac{9}{2}$$

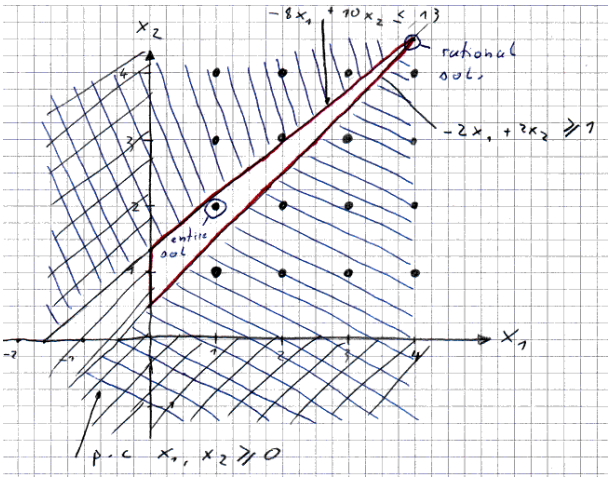
A *naive* and **wrong** approach one could think of would be to round the values to the closest integer, for instance:

$$x_1 = 4, x_2 = 5$$

but this breaks the following constraint:

$$-8x_1 + 10x_2 \leq 13 \Rightarrow$$
$$-8 \cdot 4 + 10 \cdot 5 = -32 + 50 = 16 > 13$$

Rounding is not a good idea



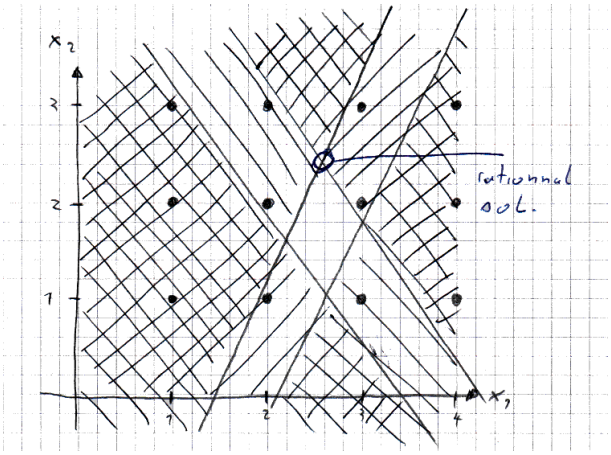
As one can sees on the schema above, using integer numbers, the solution to the (ILP) is significantly different than the solution of the (LP):

$$x_1 = 1, x_2 = 2$$

15.2.2 Even more different

This new schema on the left underlines another issue. While a problem likely have solution in rationnal numbers when the constraint system make it possile, there is not guarantee it will necessarily have a solution in integer number.

The problem illustrated here has a system of constraints that binds the solution between the adjacent integer number. While this system has a solution in rationnal numbers, it has none in integer numbers.



CHAPTER 16

The *Branch and Bound* algorithm

Contents

16.1 Introduction

Branch and bound (*BB* or *B & B*) (in french: *"Méthode par séparation et évaluation"*) is a general algorithm for finding optimal solutions of various optimization problems, especially in discrete and combinatorial optimization.
It consists of a systematic enumeration of all candidate solutions, where large subsets of fruitless candidates are discarded massively, by using upper and lower estimated bounds of the quantity being optimized.

The method was first proposed by A. H. Land and A. G. Doig in 1960 for discrete programming and is the basic workhorse technique for solving integer and discrete programming problems. The method is based on the observation that the enumeration of integer solutions has a tree structure.

16.2 Principle

This is the *divide and conquer* method. We divide a large problem into a few smaller ones. (This is the *branch* part.) The *conquering* part is done by estimate how good a solution we can get for each smaller problems (to do this, we may have to divide the problem further, until we get a problem that we can handle), that is the *bound* part.

We will use the *linear programming relaxation* to estimate the optimal solution of an integer programming. For an integer programming model (P) , the linear programming model we get by dropping the requirement that all variables must be integers is called the **linear programming relaxation of P**.

16.2.1 Steps

The steps are:

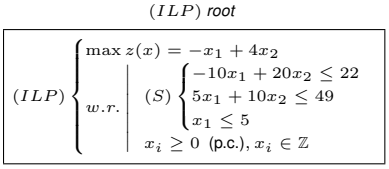
1. Divide a problem into subproblems
2. Calculate the LP relaxation of a subproblem (using for instance The *Simplex algorithm*)
    - The LP problem has no feasible solution, done;
    - The LP problem has an integer optimal solution; done. Compare the optimal solution with *the best solution we know* = **the incumbent**.
    - The LP problem has an optimal solution that is worse than the incumbent, done. (In all the cases above, we know all we need to know about that subproblem. We say that subproblem is **fathomed.**)
    - The LP problem has an optimal solution that are not all integer, better than the incumbent. In this case we would have to divide this subproblem further and repeat.

A subproblem is fathomed whenever:

1. The relaxation of the subproblem has an optimal solution with $z < z^*$ where z^* is the current best solution;
2. The relaxation of the subproblem has no feasible solution;
3. The relaxation of the subproblem has an optimal solution that has all integer values

16.3 Illustration example

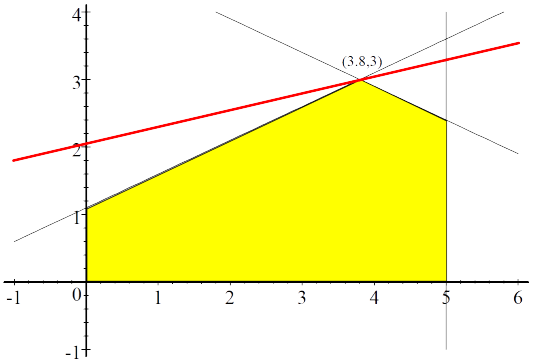
We start with the following problem which becomes the root of the tree:



16.3.1 Root (ILP)

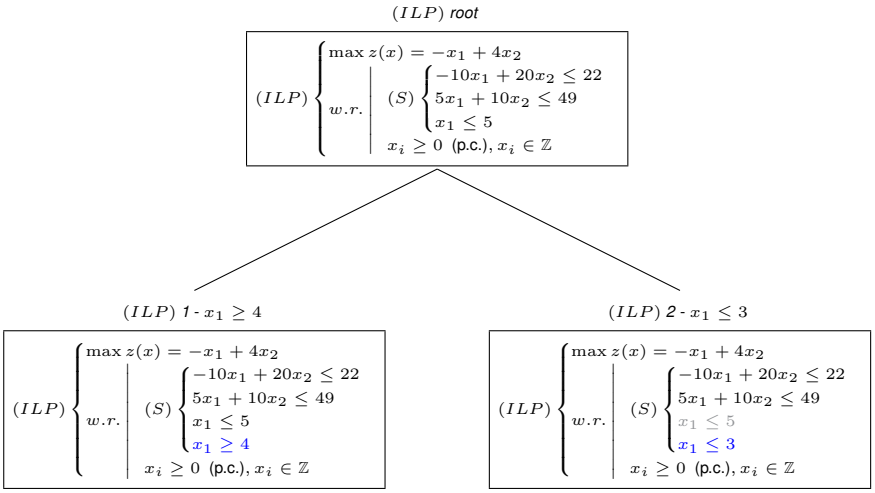
We start by solving the *LP relaxation* of the root problem:

$$(LP)\begin{cases}\max z(x) = -x_1 + 4x_2 \\ w.r. \mid (S)\begin{cases}-10x_1 + 20x_2 \leq 22 \\ 5x_1 + 10x_2 \leq 49 \\ x_1 \leq 5\end{cases} \\ x_i \geq 0 \text{ (p.c.)}\end{cases}$$



The *Optimal solution* of the relaxation is $(x_1, x_2) = (3.8, 3)$ with $z = 8.2$.

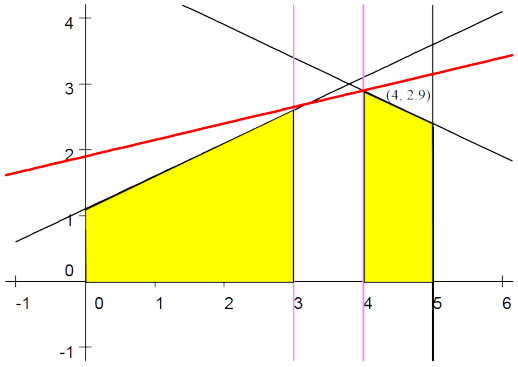
Then we consider two cases : $x_1 \geq 4$ **and** $x_1 \leq 3$**.** This gives us a the first split of the tree:



16.3.2 (ILP) 1 - Root $\rightarrow$ Left

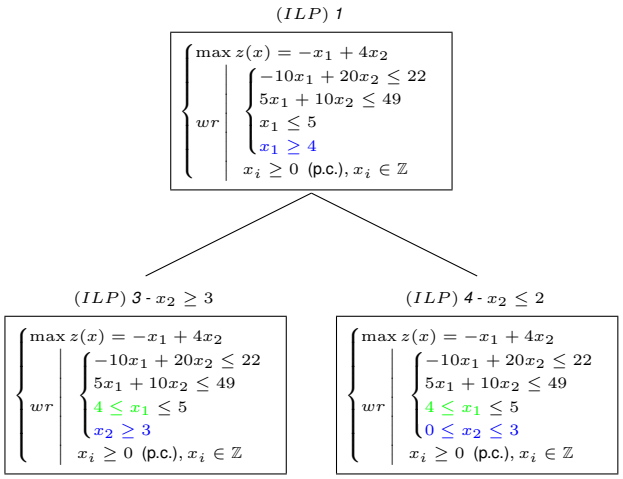
The *LP relaxation* of the problem:

$$(LP)\begin{cases}\max z(x) = -x_1 + 4x_2 \\ w.r. \quad (S)\begin{cases}-10x_1 + 20x_2 \leq 22 \\ 5x_1 + 10x_2 \leq 49 \\ x_1 \leq 5 \\ x_1 \geq 4\end{cases} \\ x_i \geq 0 \text{ (p.c.)}\end{cases}$$



has as *Optimal solution* : $(x_1, x_2) = (4, 2.9)$ with $z = 7.6$.

This gives us two new cases : $x_2 \geq 3$ **and** $x_2 \leq 2$ and makes us the split the tree further:



(Only partial tree, see figure 16.1 on page on page 156 for full tree)

16.3.3 (ILP) 3 - Root $\rightarrow$ Left $\rightarrow$ Left

The *LP relaxation* of the problem:

$$(LP)\begin{cases}\max z(x) = -x_1 + 4x_2 \\ w.r. \quad (S)\begin{cases}-10x_1 + 20x_2 \leq 22 \\ 5x_1 + 10x_2 \leq 49 \\ 4 \leq x_1 \leq 5 \\ x_2 \geq 3\end{cases} \\ x_i \geq 0 \text{ (p.c.)}\end{cases}$$

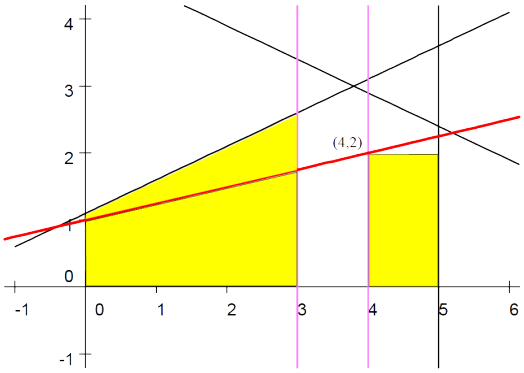
has no feasible solution at all due to the constraints voiding the set of feasible solution.

Hence the (ILP) has no solution either.

16.3.4 (ILP) 4 - Root $\rightarrow$ Left $\rightarrow$ Right

The *LP relaxation* of the problem:

$$(LP)\begin{cases}\max z(x) = -x_1 + 4x_2 \\ w.r. \quad \left|\begin{array}{l}(S)\begin{cases}-10x_1 + 20x_2 \leq 22 \\ 5x_1 + 10x_2 \leq 49 \\ 4 \leq x_1 \leq 5 \\ 0 \leq x_2 \leq 2\end{cases} \\ x_i \geq 0 \text{ (p.c.)}\end{array}\right.\end{cases}$$



has as *Optimal solution* : $(x_1, x_2) = (4, 2)$ with $z = 4$.

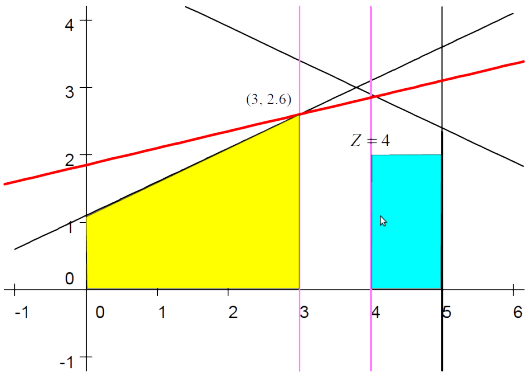
This is the optimal solution of the (ILP) as well.

Currently the best value of z for the original (ILP) is 4 with $(x_1, x_2) = (4, 2)$.

16.3.5 (ILP) 2 - Root $\rightarrow$ Right

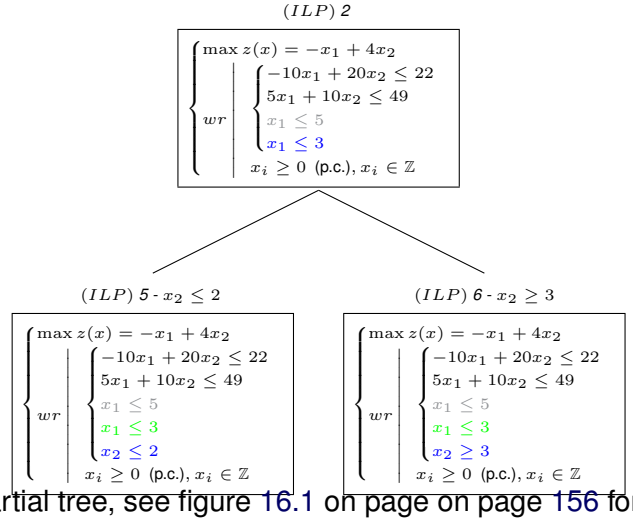
The *LP relaxation* of the problem:

$$(LP)\begin{cases}\max z(x) = -x_1 + 4x_2 \\ w.r. \quad \left|\begin{array}{l}(S)\begin{cases}-10x_1 + 20x_2 \leq 22 \\ 5x_1 + 10x_2 \leq 49 \\ x_1 \leq 5 \\ x_1 \leq 3\end{cases} \\ x_i \geq 0 \text{ (p.c.)}\end{array}\right.\end{cases}$$



has as *Optimal solution* : $(x_1, x_2) = (3, 2.6)$ with $z = 7.4$.

This gives us two new cases : $x_2 \leq 2$ and $x_2 \geq 3$ and makes us the split the tree further:



(Only partial tree, see figure 16.1 on page on page 156 for full tree)

16.3.6 (ILP) 6 - Root $\rightarrow$ Right $\rightarrow$ Right

The *LP relaxation* of the problem:

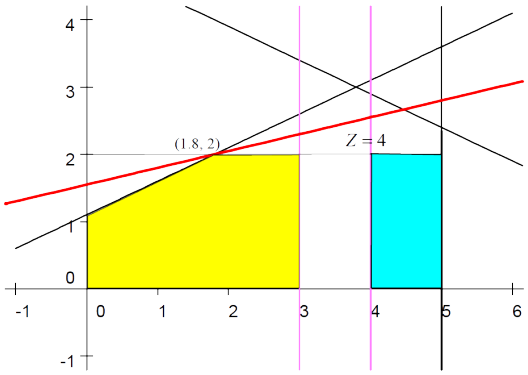
$$(LP)\begin{cases}\max z(x) = -x_1 + 4x_2 \\ w.r. \begin{vmatrix} (S)\begin{cases} -10x_1 + 20x_2 \leq 22 \\ 5x_1 + 10x_2 \leq 49 \\ x_1 \leq 5 \\ x_1 \leq 3 \\ x_2 \geq 3 \end{cases} \\ x_i \geq 0 \text{ (p.c.)} \end{vmatrix}\end{cases}$$

has no feasible solution, hence the (ILP) has no solution either we stop there.

16.3.7 (ILP) 5 - Root $\rightarrow$ Right $\rightarrow$ Right

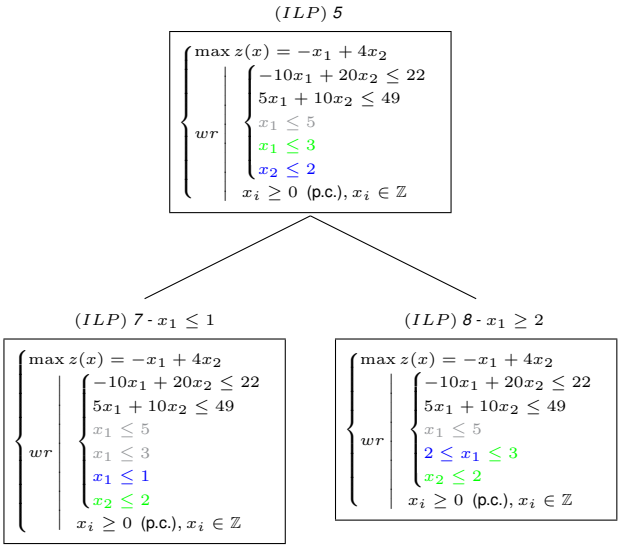
The *LP relaxation* of the problem:

$$(LP)\begin{cases}\max z(x) = -x_1 + 4x_2 \\ w.r. \begin{vmatrix} (S)\begin{cases} -10x_1 + 20x_2 \leq 22 \\ 5x_1 + 10x_2 \leq 49 \\ x_1 \leq 5 \\ x_1 \leq 3 \\ x_2 \leq 2 \end{cases} \\ x_i \geq 0 \text{ (p.c.)} \end{vmatrix}\end{cases}$$



has as *Optimal solution* : $(x_1, x_2) = (1.8, 2)$ with $z = 6.2$.

This gives us two new cases : $x_1 \leq 1$ **and** $x_1 \geq 2$ and makes us the split the tree further:

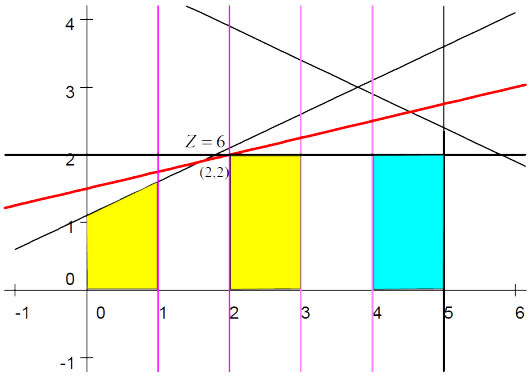


(Only partial tree, see figure 16.1 on page on page 156 for full tree)

16.3.8 (ILP) 7 - Root $\rightarrow$ Right $\rightarrow$ Left $\rightarrow$ Left

The *LP relaxation* of the problem:

$$(LP)\begin{cases}\max z(x) = -x_1 + 4x_2 \\ w.r. \quad (S)\begin{cases}-10x_1 + 20x_2 \leq 22 \\ 5x_1 + 10x_2 \leq 49 \\ x_1 \leq 5 \\ x_1 \leq 3 \\ x_1 \leq 1 \\ x_2 \leq 2\end{cases} \\ x_i \geq 0 \text{ (p.c.)}\end{cases}$$



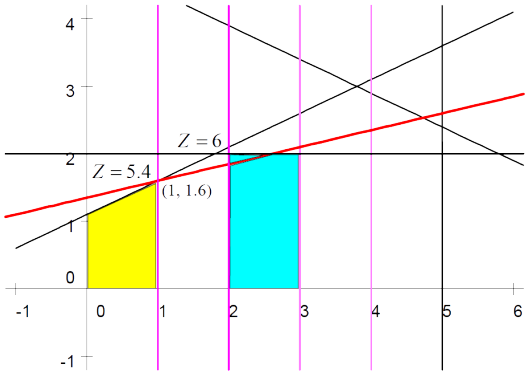
The (LP) has as *Optimal solution* : $(x_1, x_2) = (2, 2)$ with $z = 6$.

Since this is better than the incumbent $z = 4$ at $(x_1, x_2) = (4, 2)$, this new integer solution is our current best solution.

16.3.9 (ILP) 8 - Root $\rightarrow$ Right $\rightarrow$ Left $\rightarrow$ Right

The *LP relaxation* of the problem:

$$(LP)\begin{cases}\max z(x) = -x_1 + 4x_2 \\ w.r. \quad (S)\begin{cases}-10x_1 + 20x_2 \leq 22 \\ 5x_1 + 10x_2 \leq 49 \\ x_1 \leq 5 \\ 2 \leq x_1 \leq 3 \\ x_2 \leq 2\end{cases} \\ x_i \geq 0 \text{ (p.c.)}\end{cases}$$



has as *Optimal solution* : $(x_1, x_2) = (1, 1.6)$ with $z = 5.4$.

Then any integer solution in this region can not give us a solution with the value of z greater than 5.4. **This branch is fathomed.**

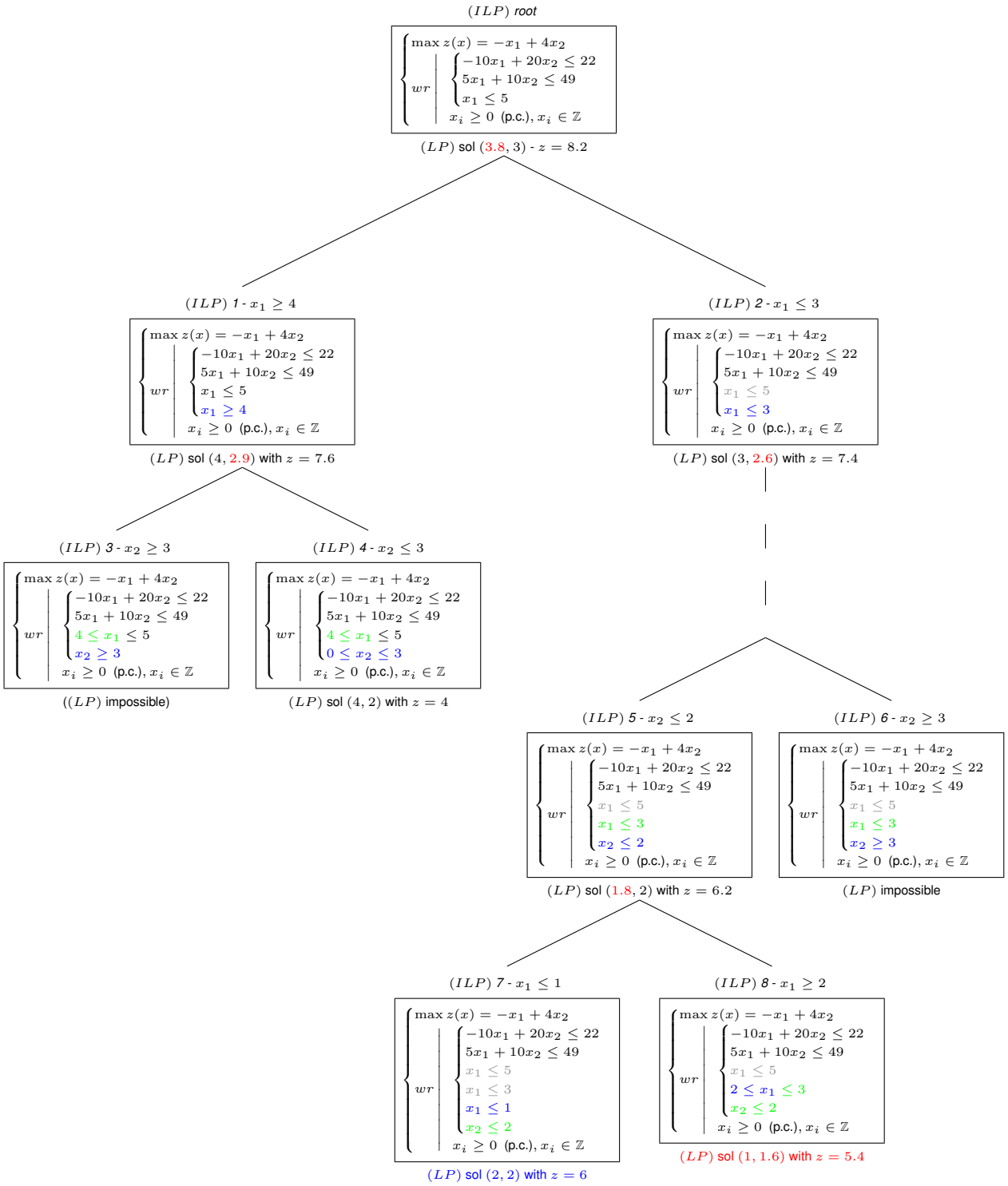


Figure 16.1: Full tree of illustration example

16.4 The *Branch-and-Bound* method

16.4.1 General Form

The following phases are repeated at least until a integer solution is produced:

- *branch* : selection and branch
- *bound* : limitation and elimination

Selection

A sub-problem not yet eliminated and not yet branched is chosen to start the process (all) over. In general, one always continue the algorithm from the best node not yet selected.

Branching

This consists in splitting the region of feasible solution (reprsented as a node in the tree) in two distinct sets. Each of these new set resolves to a new problem, i.e a new nod in the tree). Whenever more that one variable are not integer, one can choose either the on the with the highest impact on the score (the most valuable) or the first one in terms of index number.

Limitation

Bounds on the optimal solution are extracted from the sub-problems.

Elimination

Exclusion of the sub-problem(s) where wither the (LP) is impossible to solve ot the solution to the LP is not better than the current (ILP) best solution found so far.

16.4.2 Assumptions

There are a few assumptions on which the *B&B* algorithm is built

- The *B&B* algorithm relies on an underlying algorithm offering a way to solve the (LP) resulting from the LP relaxation of the (ILP) problem.
- Also, it needs a **branching rule** that can be used to split the set of feasible solutions into distinct sub-sets from a current solution.

16.5 Algorithm of *Branch-and-Bound*

Notations :

- NF set of the feasible solutions, set of the possible tree nodes
- z_l lower bound on the solution of the problem, i.e. value of the best solution found so far
- $f : F \to \mathbb{R}$ the score (objective) function. F is the region of the *feasible solutions*
- F^r region of the feasible solution for an LP relaxation of the original problem
- (z, x) an optimal solution of a sub-problem

Input : parameters of the optimization problem (ILP)

Ouput : either an optimal solution or no solution

Initializations :

- (O1) $F_0 = F^r$
- (O2) $NF = 0$ (root of the tree)
- (O3) $Z_l = -\inf$

Iteration :

Step 1 - selection :

- (E1) Choose $k \in NF$
- (E2) Let $S_k = \max\{f(x)|x \in F^k\}$ with $F^k \subset F$
  (S_k is the solution to the relaxed problem)

Step 2 - limitation :

- (E3) Compute an optimal solution (z_k, x_k) for S_k
- (E4) Whenever S_k doesn't have a solution, let $z_k = -\inf$

Step 3 - elimination :

- (E5) S_k is excluded whenever
  - $z_k = -\inf$ (no solution)
  - $z_k < z_l$ (no possible amelioration)
  - $z_k = z_l$ and $x_k \notin F$ (no possible amelioration)
- (E6) If $z_k > z_l$ then $z_l = z_k$, $NF = NF\ \{k\}$, goto Step 5

Step 4 - branch :

- (E7) Partition sol $F_k = F_{k_1} \cup F_{k_2} \cup ... \cup F_{k_n}$
- (E6) Pose $NF = (NF\ \{k\}) \cup \{k_1, k_2, ..., k_n\}$
- (E8) goto Stop

Stop :

- (A1) If $NF \neq$, then goto Step 1
- (A2) z_l is the optimum solution
- (A3) If $z_l = -\inf$, then the problem has no solution

16.6 Practice

16.6.1 Exercise 1 : Branch & Bound - Simplex

Solve the following (ILP) problem using B&B and the Simplex:

$$(ILP)\begin{cases}\max z(x) = x_1 + 4x_2 \\ w.r. \left| \begin{array}{l} (S)\begin{cases}5x_1 + 8x_2 \leq 40 \\ -2x_1 + 3x_2 \leq 9\end{cases} \\ x_i \geq 0 \text{ (p.c.)}, x_i \in \mathbb{Z}\end{array}\right.\end{cases}$$

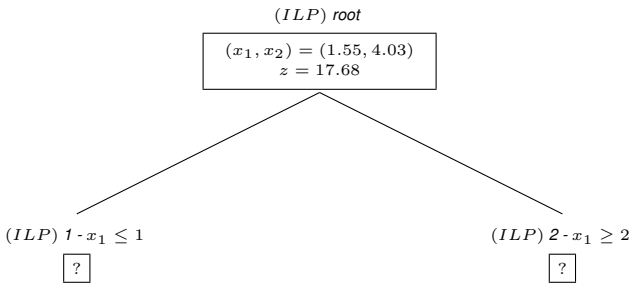
16.6.1.1 Solution

For the sake of simplifying the iterations, we'll use the R environment to solve the Simplex on the LP relaxed problems (see 14.3).

Root node :

Solving the LP relaxation with R

```
a<-c(1, 4)
A1<-matrix(c(5, 8, -2, 3), nrow = 2, ncol=2, byrow=TRUE)
b1<-c(40, 9)
simplex(a, A1, b1, maxi = TRUE)
...
      x1       x2
1.548387 4.032258
The optimal value of the objective  function is 17.6774193548387.
```



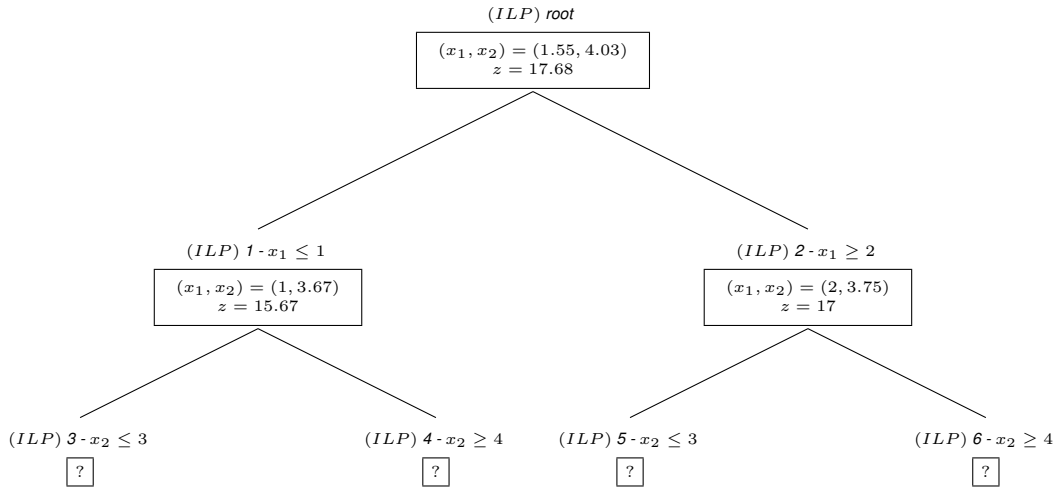
(ILP) **1 -** $x_1 \leq 1$ **:**

```
a<-c(1, 4)
A1<-matrix(c(5, 8, -2, 3, 1, 0), nrow = 3, ncol=2, byrow=TRUE)
b1<-c(40, 9, 1)
simplex(a, A1, b1, maxi = TRUE)
...
      x1       x2
1.000000 3.666667
The optimal value of the objective  function is 15.6666666666667.
```

(ILP) **2 -** $x_1 \geq 2$ **:**

```
a<-c(1, 4)
A1<-matrix(c(5, 8, -2, 3), nrow = 2, ncol=2, byrow=TRUE)
b1<-c(40, 9)
A2<-matrix(c(1, 0), nrow = 1, ncol=2, byrow=TRUE)
b2<-c(2)
simplex(a, A1, b1, A2, b2, maxi = TRUE)
...
  x1   x2
2.00 3.75
The optimal value of the objective  function is 17.
```

Next split



(ILP) **3 -** $x_2 \leq 3$ **:**

```
a<-c(1, 4)
A1<-matrix(c(5, 8, -2, 3, 1, 0, 0, 1), nrow = 4, ncol=2, byrow=TRUE)
b1<-c(40, 9, 1, 3)
simplex(a, A1, b1, maxi = TRUE)
...
x1 x2
 1  3
The optimal value of the objective  function is 13.
```

(ILP) **4 -** $x_2 \geq 4$ **:**

```
a<-c(1, 4)
A1<-matrix(c(5, 8, -2, 3, 1, 0), nrow = 3, ncol=2, byrow=TRUE)
b1<-c(40, 9, 1)
A2<-matrix(c(0, 1), nrow = 1, ncol=2, byrow=TRUE)
b2<-c(4)
simplex(a, A1, b1, A2, b2, maxi = TRUE)
...
No feasible solution could be found.
```

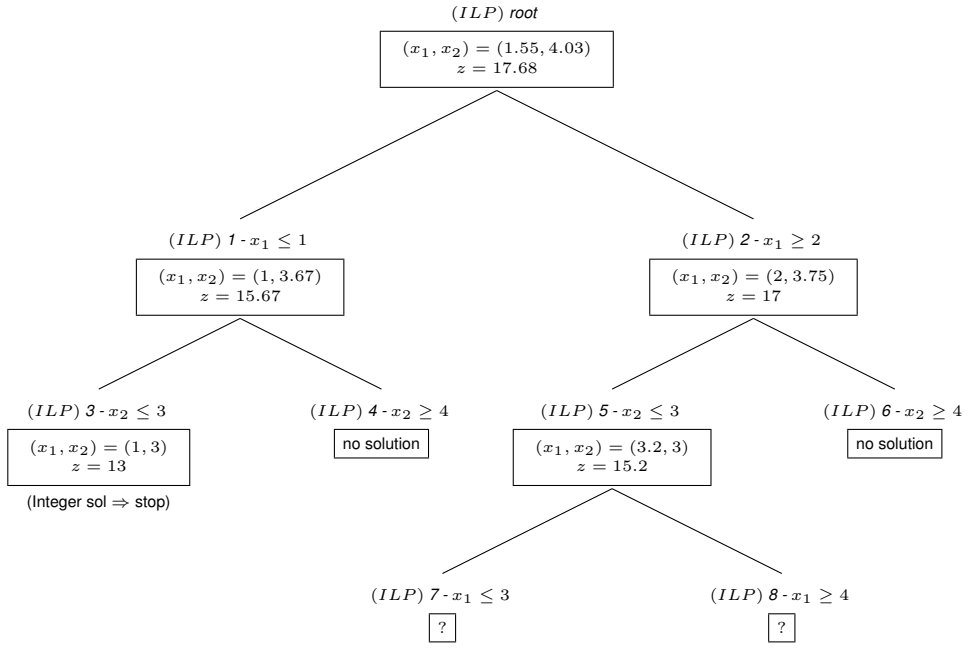
(ILP) **5 -** $x_2 \leq 3$ **:**

```
a<-c(1, 4)
A1<-matrix(c(5, 8, -2, 3, 0, 1), nrow = 3, ncol=2, byrow=TRUE)
b1<-c(40, 9, 3)
A2<-matrix(c(1, 0), nrow = 1, ncol=2, byrow=TRUE)
b2<-c(2)
simplex(a, A1, b1, A2, b2, maxi = TRUE)
...
 x1  x2
3.2 3.0
The optimal value of the objective  function is 15.2.
```

(ILP) **6 -** $x_2 \geq 4$ **:**

```
a<-c(1, 4)
A1<-matrix(c(5, 8, -2, 3), nrow = 2, ncol=2, byrow=TRUE)
b1<-c(40, 9)
A2<-matrix(c(1, 0, 0, 1), nrow = 2, ncol=2, byrow=TRUE)
b2<-c(2, 4)
simplex(a, A1, b1, A2, b2, maxi = TRUE)
...
No feasible solution could be found.
```

Next split



(ILP) **7 -** $x_1 \leq 3$ **:**

```
a<-c(1, 4)
A1<-matrix(c(5, 8, -2, 3, 0, 1, 1, 0), nrow = 4, ncol=2, byrow=TRUE)
b1<-c(40, 9, 3, 3)
A2<-matrix(c(1, 0), nrow = 1, ncol=2, byrow=TRUE)
b2<-c(2)
simplex(a, A1, b1, A2, b2, maxi = TRUE)
...
```

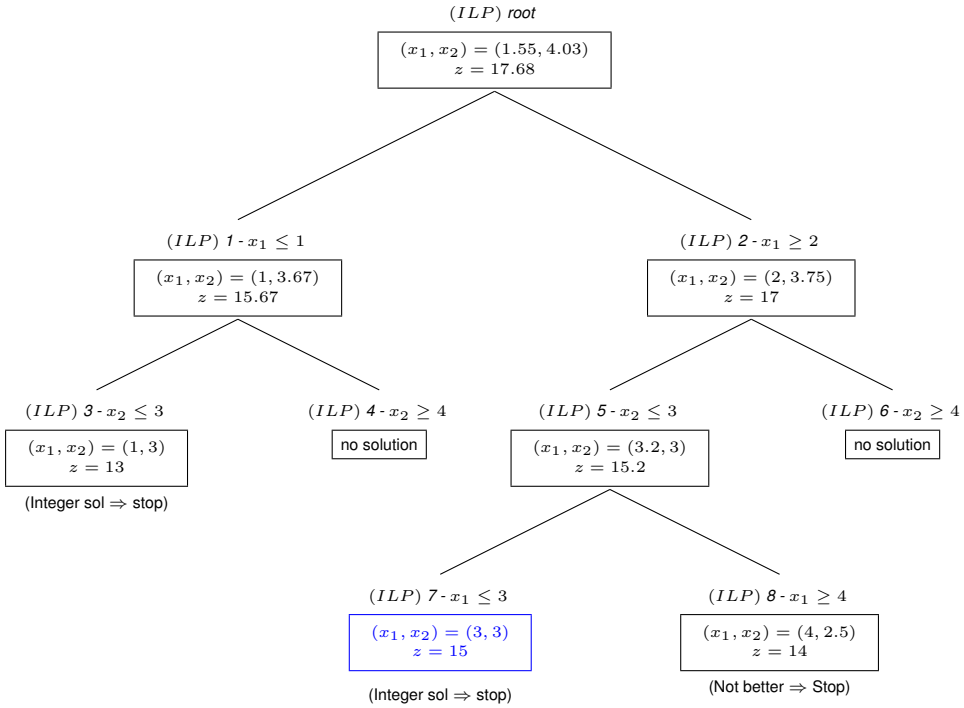
```
x1 x2
 3  3
The optimal value of the objective  function is 15.
```

(ILP) 8 - $x_1 \geq 4$:

```
a<-c(1, 4)
A1<-matrix(c(5, 8, -2, 3, 0, 1), nrow = 3, ncol=2, byrow=TRUE)
b1<-c(40, 9, 3)
A2<-matrix(c(1, 0, 1, 0), nrow = 2, ncol=2, byrow=TRUE)
b2<-c(2, 4)
simplex(a, A1, b1, A2, b2, maxi = TRUE)
...
 x1  x2
4.0 2.5
The optimal value of the objective  function is 14.
```

No need to go any further here since a child of (ILP) 8 cannot be better than (ILP) 7

Final tree



16.6.1.2 Result

The solution of the (ILP) is $(x_1, x_2) = (3, 3)$ with result $z = 15$

16.6.2 Exercise 2 : The Knapsack problem

The problam is as follows:

Various items are put in a bag. Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

16.6.2.1 The problem

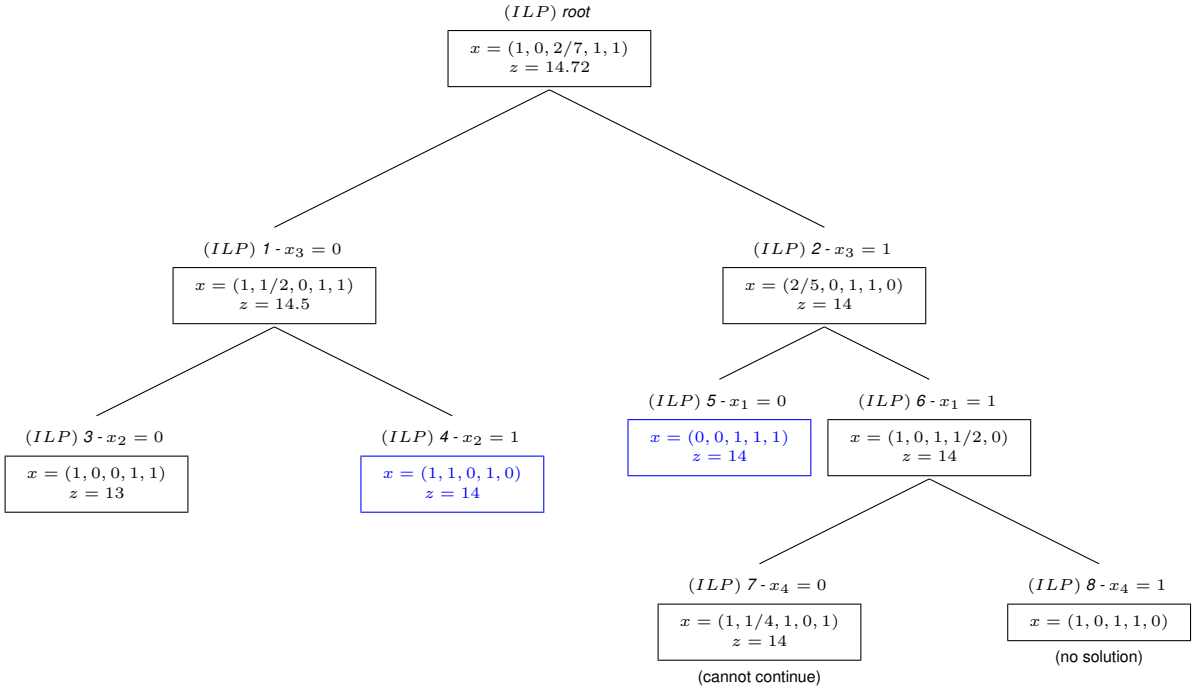
- the volume of the bag is 15 (lt).
- there are five objects

Object	Value	Volume	Rank
1	5	5	1
2	3	4	5
3	6	7	4
4	6	6	1
5	2	2	1

The problem can be formalized as follows:

$$(ILP)\begin{cases}\max z(x) = 5x_1 + 3x_2 + 6x_3 + 6x_4 + 2x_5 \\ w.r. \quad \left|\begin{array}{l}(S)\left\{5x_1 + 4x_2 + 7x_3 + 6x_4 + 2x_5 \leq 15\right. \\ x_i \in \{0,1\}\end{array}\right.\end{cases}$$

16.6.2.2 Solution



16.6.3 Exercise 3 : an (ILP) as a binary problem (Knaspack

TODO

CHAPTER 17

The *Cutting Plane* method

Contents

17.1 Introduction

In mathematical optimization, the **cutting-plane method** (french: *"Méthode de coupes"* is an umbrella term for optimization methods which iteratively refine a feasible set or objective function by means of linear inequalities, termed *cuts*.

Cutting plane methods for (ILP) work by solving a non-integer linear program, the linear relaxation of the given integer program. The obtained optimum is tested for being an integer solution.

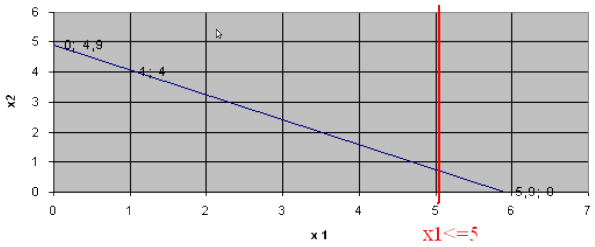
- If it is, the algorithm is over
- If it is not, there is guaranteed to exist a linear inequality that separates the optimum from the *convex hull* (frenchh : *enveloppe convexe*) of the true feasible set.

Finding such an inequality is the separation problem, and such an inequality is a *cut*. A cut can be added to the relaxed linear program. Then, the current non-integer solution is no longer feasible to the relaxation.
This process is repeated until an optimal integer solution is found.

17.1.1 Example on the Knapsack problem

Let's illustrate this technic on the following *Knapscak* problem:

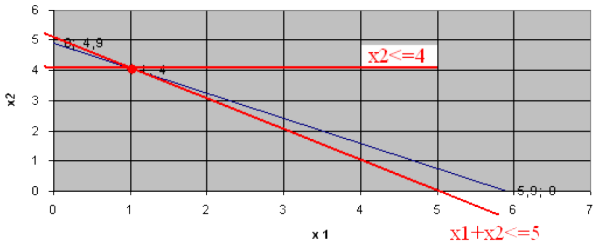
$$(ILP)\begin{cases}\max z(x) = 10x_1 + 11x_2 \\ w.r. \quad \left|\begin{array}{l}(S)\begin{cases}10x_1 + 12x_2 \leq 59 \\ x_1 \leq 5\end{cases} \\ x_i \geq 0 \text{ (p.c.)}, x_i \in \mathbb{Z}\end{array}\right.\end{cases}$$



The *optimum solution* of the *LP relaxation of the problem* is the right-most point $x = (5.9, 0)$. $x_1 \leq 5$ is **a cut** since it **eliminates this optimum solution without any impact on integer solutions**. Also, any inequations of the form $x_1 + x_2 \leq \alpha$ respecting $5 \leq \alpha \leq 5.9$ is a cut.

Thanks to successvie cuts, one can build the *convexe hull* around the *integer solutions*, at least in the neihgbourhood of the optimal solution. When all the constraints required to build this convexe hull have been added to the LP relaxation of the problem, the nest solution of this problem will return the integer solution.

In the previous example, one only need the cuts $x_1 + x_2 \leq 5$ and $x_2 \leq 4$ to build this *concexe hull*. When these cuts are added to the LP relaxation of the problem, the next resolution of this problem gives the *optimal integer solution* $x = (4, 4)$



As one can guess, chosing the right cuts amongst the set of possible cuts is crucial to reach the optimal integer solution. In the previous example, for instance, had one chosen a cut of the form $\alpha_k = \alpha_k - 1 - \frac{1}{10^k}$, one would have gotten close to the *convexe hull* without ever reaching it. In conclusion, one needs to choose efficient cuts.

17.2 Gomory's cut

17.2.1 The principle

Cutting planes were proposed by R. Gomory in the 1950s as a method for solving integer programming and *mixed-integer programming* problems. However most experts, including Gomory himself, considered them to be impractical due to numerical instability, as well as ineffective because many rounds of cuts were needed to make progress towards the solution.
Gomory cuts, however, are very efficiently generated from a simplex tabular, whereas many other types of cuts are either expensive or even NP-hard to separate

Let's assume we face an (ILP) problem expressed **under standard form** this way:

$$(ILP)\begin{cases}\max \mid \min z(x) = a^t \cdot x \\ w.r. \mid (S)\begin{cases} A \cdot x = b^t \\ x_i \geq 0 \text{ (p.c.) and } x_i \in \mathbb{Z}\end{cases}\end{cases}$$

Let's use for instance:

$$(ILP)\begin{cases}\max z(x) = x_1 + x_2 \\ w.r. \mid (S)\begin{cases} 7x_1 + x_2 \leq 15 \\ -x_1 + x2 \leq 1 \end{cases} \\ x_i \geq 0 \text{ (p.c.) and } x_i \in \mathbb{Z}\end{cases}$$

The method proceeds by first dropping the requirement that the x_i be integers and solving the corresponding *LP relaxation problem* to obtain a *basic feasible solution*.
Geometrically, this solution will be a vertex of the convex polytope consisting of all feasible points.

If this vertex is not an integer point then the method finds a hyperplane with the vertex on one side and all feasible integer points on the other.

17.2.2 At start, the Simplex

Using the **simplex algorithm**, the solution is extracted from a *last system* expressing the *base variables* as well as the score in terms of the *off-base variables*.

In the *constraint equations system* we end up with a set of equations of the form:

$$x_i + \sum_{j=1}^{m} \bar{a}_{ij}\lambda_j = \bar{b}_i \text{ with } \begin{cases} x_i \text{ a } \textit{base variable} \\ \lambda_j \text{ an } \textit{off-base variable} \\ \bar{a}_{ij} \text{ the coefficient of } \lambda_j \end{cases}$$

In our example:

$$\begin{cases}\max z(x) = \frac{9}{2} - \frac{1}{4}x_3 - \frac{3}{4}x_4 \\ w.r. \mid (S)\begin{cases} x_1 + \frac{1}{8}x_3 - \frac{1}{8}x_4 = \frac{7}{4} \\ x_2 + \frac{1}{8}x_3 + \frac{7}{8}x_4 = \frac{11}{4} \end{cases} \\ x_i \geq 0 \text{ (p.c.)}\end{cases}$$

(with sol : $x = (\frac{7}{4}, \frac{11}{4}, 0, 0)$)

One should note that there are mandatorily fractionnal coefficients. If every coefficient were integers, we would have an entire solution of the LP relaxation and we would be done.

17.2.3 Chosing a source constraint

Amongst this set of equations,one should chose one write a new **equation** with integer parts on the left and fractionnal part on the right.. That new extraction simply consists in a few manipulation to peform on the former one.

The new equation has the form

$$x_i + \sum_{j=1}^{m} \lfloor \bar{a}_{ij} \rfloor \lambda_j - \lfloor \bar{b}_i \rfloor = \bar{b}_i - \lfloor \bar{b}_i \rfloor - \sum_{j=1}^{m} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor)\lambda_j$$

In our example:

$$x_1 + 0x_3 - 1x_4 - 1 = \frac{7}{4} - 1 - (\frac{1}{8} - 0)x_3 - (-\frac{1}{8} - (-1))x_4$$

This equation holds **only** for the LP relaxation, i.e. whenever $x_i \in \mathbb{R}$. But what happens for an entire solution ?

For any integer point in the *feasible region*, i.e. an entire solution $\Rightarrow x_i \in \mathbb{Z}$, the left side of this equation is mandatorily an integer. But we know that the right side is mandatorily less than 1 (<1).
Hence the *common part* must be less than or equal to 0.

17.2.4 Extracting the constraint

In the light of this, one knows that the following inequality must hold for any integer point in the feasible region.

We know this is true:

$$\bar{b}_i - \lfloor \bar{b}_i \rfloor + \sum_{j=1}^{m} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor)\lambda_j \leq 0 \Leftrightarrow$$

$$\boxed{\sum_{j=1}^{m} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor)\lambda_j \leq -\bar{b}_i + \lfloor \bar{b}_i \rfloor}$$

In our example:

$$\frac{7}{4} - 1 - (\frac{1}{8} - 0)x_3 - (-\frac{1}{8} - (-1))x_4 \leq 0 \Leftrightarrow$$

$$\frac{3}{4} - \frac{1}{8}x_3 - \frac{7}{8}x_4 \leq 0 \Leftrightarrow$$

$$\boxed{-\frac{1}{8}x_3 - \frac{7}{8}x_4 \leq -\frac{3}{4}}$$

And we end up with our new cut!

17.2.5 Introduce a new slack variable

So the inequality above excludes the basic feasible solution and thus is a cut with the desired properties. Introducing a new slack variable x_k for this inequality, a new constraint is added to the linear program.

Introducing x_k the new slack variable

$$x_k + \sum_{j=1}^{m} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor)\lambda_j = -\bar{b}_i + \lfloor \bar{b}_i \rfloor$$

In our example:

$$-\frac{1}{8}x_3 - \frac{7}{8}x_4 \leq -\frac{3}{4} \Leftrightarrow$$

$$\boxed{-\frac{1}{8}x_3 - \frac{7}{8}x_4 + x_5 = -\frac{3}{4}}$$

17.2.6 A new problem

The new constraint is added to the former LP relaxation of the (ILP) problem and the whole process starts all over until an entire solution is found (either luckily or because the full *convexe hull* around entire soltion has been built)

In our example, we end up with the following new problem:

$$\begin{cases} \max z(x) = \frac{9}{2} - \frac{1}{4}x_3 - \frac{3}{4}x_4 \\ w.r. \left| \begin{array}{l} (S) \begin{cases} x_1 + \frac{1}{8}x_3 - \frac{1}{8}x_4 = \frac{7}{4} \\ x_2 + \frac{1}{8}x_3 + \frac{7}{8}x_4 = \frac{11}{4} \\ -\frac{1}{8}x_3 - \frac{7}{8}x_4 + x_5 = -\frac{3}{4} \end{cases} \\ x_i \geq 0 \ \text{(p.c.)} \end{array} \right. \end{cases}$$

(with sol : $x = (\frac{7}{4}, \frac{11}{4}, 0, 0)$)

17.3 Example continued

Let's continue the example of the previous section as it provides forms an interesting case. When looking carefully at the new constraint, one sees that the *usual initial solution* (with each variable appearing only once in an equation considered a *base variable*) is a *non-feasible solution*. When putting the x_3, x_4 variable at 0 in the system of equations, we get:

$$(S) \begin{cases} x_1 + \frac{1}{8}x_3 - \frac{1}{8}x_4 = \frac{7}{4} \\ x_2 + \frac{1}{8}x_3 + \frac{7}{8}x_4 = \frac{11}{4} \\ -\frac{1}{8}x_3 - \frac{7}{8}x_4 + x_5 = -\frac{3}{4} \end{cases} \Rightarrow \begin{cases} x_1 + \frac{1}{8}0 - \frac{1}{8}0 = \frac{7}{4} \\ x_2 + \frac{1}{8}0 + \frac{7}{8}0 = \frac{11}{4} \\ -\frac{1}{8}0 - \frac{7}{8}0 + x_5 = -\frac{3}{4} \end{cases} \Rightarrow \begin{cases} x_1 = \frac{7}{4} \\ x_2 = \frac{11}{4} \\ x_5 = -\frac{3}{4} \end{cases}$$

At that point, either one tries to come up with another partitioning in *base variable* / *off-base variable* or one can try to solve the **Dual problem** (see 18). Let's transform the problem into its dual form.
First we need to get back in a primal form where we have strictly the same variables in every constraint and in the score. Whenever we should not be possible, one has to add all variables in every constraint and in the score with the coefficient 0 to perform the traduction $\rightarrow$ painful.
In our case, we can simply considere the variables appearing only once ion the constraints as slack variables and get rid of them

Primal form:

$$\begin{cases} \max z(x) = \frac{9}{2} - \frac{1}{4}x_3 - \frac{3}{4}x_4 \\ w.r. \left| \begin{array}{l} (S) \begin{cases} \frac{1}{8}x_3 - \frac{1}{8}x_4 \leq \frac{7}{4} \\ \frac{1}{8}x_3 + \frac{7}{8}x_4 \leq \frac{11}{4} \\ -\frac{1}{8}x_3 - \frac{7}{8}x_4 \leq -\frac{3}{4} \end{cases} \\ x_i \geq 0 \ \text{(p.c.)} \end{array} \right. \end{cases}$$

Dual form:

$$\begin{cases} \min w(x) = \frac{9}{2} + \frac{7}{4}y_1 + \frac{11}{4}y_2 - \frac{3}{4}y_3 \\ w.r. \left| \begin{array}{l} (S) \begin{cases} \frac{1}{8}y_1 + \frac{1}{8}y_2 - \frac{1}{8}y_3 \geq -\frac{1}{4} \\ -\frac{1}{8}y_1 + \frac{7}{8}y_2 - \frac{7}{8}y_3 \geq -\frac{3}{4} \end{cases} \\ x_i \geq 0 \ \text{(p.c.)} \end{array} \right. \end{cases}$$

We now rewrite the dual problem:

$$\begin{cases} \min w(x) = \frac{9}{2} + \frac{7}{4}y_1 + \frac{11}{4}y_2 - \frac{3}{4}y_3 \\ w.r. \left| \begin{array}{l} (S) \begin{cases} -\frac{1}{8}y_1 - \frac{1}{8}y_2 + \frac{1}{8}y_3 \leq \frac{1}{4} \\ +\frac{1}{8}y_1 - \frac{7}{8}y_2 + \frac{7}{8}y_3 \leq \frac{3}{4} \end{cases} \\ x_i \geq 0 \ \text{(p.c.)} \end{array} \right. \end{cases}$$

The big advantage here is that we end up with positive values on the right side of the constraint system equations.

Under standard form:

$$\begin{cases} \min w(x) = \frac{9}{2} + \frac{7}{4}y_1 + \frac{11}{4}y_2 - \frac{3}{4}y_3 \\ w.r. \left| \begin{array}{l} (S) \begin{cases} -\frac{1}{8}y_1 - \frac{1}{8}y_2 + \frac{1}{8}y_3 + y_4 = \frac{1}{4} \\ +\frac{1}{8}y_1 - \frac{7}{8}y_2 + \frac{7}{8}y_3 + y_5 = \frac{3}{4} \end{cases} \\ x_i \geq 0 \ \text{(p.c.)} \end{array} \right. \end{cases}$$

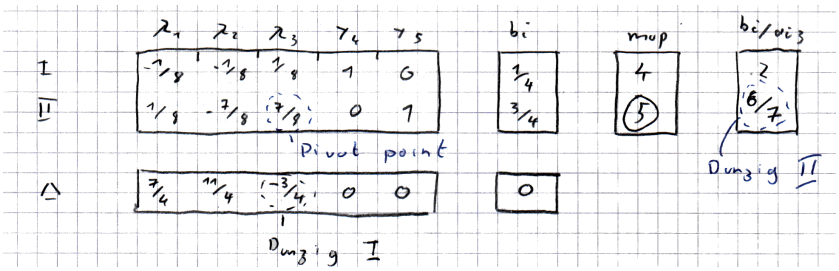
which has a feasible initial solution in the simplex.

This gives us the following variable mapping (primal in-base = dual *off-base* (slack)):

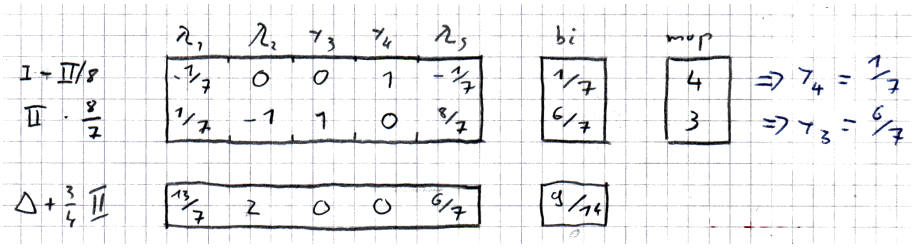
primal	x_1	x_2	x_3	x_4	x_5
dual	y_1	y_2	y_4	y_5	y_3

17.3.1 Solving the dual with the Simplex

Step 1 : We start with the following tabular :



Step 2 : And end up here after one iteration :



Let's represent this situation under the usual form

We now rewrite the dual problem:

$$\begin{cases} \min w(x) = \frac{13}{7}y_1 + 2y_2 + 0y_3 + 0y_4 + \frac{6}{7}y_5 \\ w.r. \left| \begin{array}{l} (S) \begin{cases} -\frac{1}{7}y_1 - 0y_2 + 0y_3 + 1y_4 - \frac{1}{7}y_5 = \frac{1}{7} \\ +\frac{1}{7}y_1 - 1y_2 + 1y_3 + 0y_4 + \frac{8}{7}y_5 = \frac{6}{7} \end{cases} \\ x_i \geq 0 \ \text{(p.c.)} \end{array} \right. \end{cases}$$

Under canonical form:

$$\begin{cases} \min w(x) = \frac{13}{7}y_1 + 2y_2 + \frac{6}{7}y_5 \\ w.r. \left| \begin{array}{l} (S) \begin{cases} -\frac{1}{7}y_1 - 0y_2 - \frac{1}{7}y_5 \leq \frac{1}{7} \\ +\frac{1}{7}y_1 - 1y_2 + \frac{8}{7}y_5 \leq \frac{6}{7} \end{cases} \\ x_i \geq 0 \ \text{(p.c.)} \end{array} \right. \end{cases}$$

Here, as it shows, y_3 and y_4 are the slack variables (see their values in score and weight matrix), hence the variables used back in the primal form will be x_3 and x_5. Let's get rid of the slack variable and use the canonical form:

17.3.2 Back under primal form

We can now convert it back to the primal poblem

$$\begin{cases} \max w(x) = \frac{1}{7}x_3 + \frac{6}{7}x_5 \\ w.r. \left| \begin{array}{l} (S) \begin{cases} -\frac{1}{7}x_3 + \frac{1}{7}x_5 \geq \frac{13}{7} \\ 0x_3 - x_5 \geq 2 \\ -\frac{1}{7}x_3 + \frac{8}{7}x_5 \geq \frac{6}{7} \end{cases} \\ x_i \geq 0 \ \text{(p.c.)} \end{array} \right. \end{cases}$$

Hence we end up with:

$$\begin{cases} \max w(x) = \frac{1}{7}x_3 + \frac{6}{7}x_5 + \frac{27}{7} \\ w.r. \left| \begin{array}{l} (S) \begin{cases} -\frac{1}{7}x_3 + \frac{1}{7}x_5 \geq \frac{13}{7} \\ 0x_3 - x_5 \geq 2 \\ -\frac{1}{7}x_3 + \frac{8}{7}x_5 \geq \frac{6}{7} \end{cases} \\ x_i \geq 0 \ \text{(p.c.)} \end{array} \right. \end{cases}$$

One should take care here that we dropped the constant $\frac{9}{2}$ when solving the dual problem, one has now to re-inject it in the equation with the result of the dual problem solving added to it: $\frac{9}{2} - \frac{9}{14} = \frac{27}{7}$

And so on until the resolution of the LP relaxation produces an integer solution ...

17.4 Notes

- These methods are very heavy in terms of both computation time and memory space
- A great precision is required in order to differentiate integer values from fractionnal values
- Each iteration produces a new bound in the form of a new constraint

CHAPTER 18

The *Dual* problem

Contents

18.1 Motivation

Associated with each (primal) LP problem is a companion problem called the **dual problem** (this has been demonstrated in a theorem).
For some problems, it might be easier to solve the dual problem instead of the primal one, and merge back the solutions in the primal problem.

For instance, when a *linear optimisation problem* is not in standard form, we have seen techniques in section 14.2 to add variables - or other manipulations - to make it in standard form in order to be able to run the *complex* algorithm on it.
Sometimes, it is more efficient to convert it to its *dual problem* instead of running those manipulations.

For instance if one of the b quantity of a contraint is negative, we have seen how to multiply both members of the constraint by -1 as a first step to make it in standard form (see section 14.2). Instead of performing this manipulation, one might want to look at the *dual problem* and see if it is easily resolvable.

18.2 Properties

There are two ideas fundamental to duality theory:

- The dual of a dual linear program is the original primal linear program.
- The strong duality theorem states that if the primal has an optimal solution, x^*, then the dual also has an optimal solution, y^*, such that $c^t x^* = b^t y^*$.

18.3 Transformation

We will see here the principle of the transformation between both problems expressed in various forms.

18.3.1 Formal form

Formally, the transformation occurs this way:

- To each variable in the primal space corresponds an inequality to satisfy in the dual space, both indexed by output type.
- To each inequality to satisfy in the primal space corresponds a variable in the dual space, both indexed by input type.
- The coefficients that bound the inequalities in the primal space are used to compute the objective in the dual space.
- The coefficients used to compute the objective in the primal space bound the inequalities in the dual space.
- the inequalities are inversed.

Primal problem

$$\begin{cases} min\ z = \sum_{j=1}^{n} c_j x_j\ or\ z = cx \\ WR \left| \begin{array}{l} \left\{ \sum_{j=1}^{n} a_{ij} x_j \leq b_i\ or\ Ax \leq b \right. \\ x_j \geq 0\ w.\ (i = 1, ..., m, j = 1, ..., n) \end{array} \right. \end{cases}$$

Dual problem

$$\leftrightarrow \begin{cases} max\ w = \sum_{i=1}^{m} b_j y_i\ or\ w = b^t y \\ WR \left| \begin{array}{l} \left\{ \sum_{i=1}^{m} a_{ij} y_i \geq c_j\ or\ A^t y \geq c \right. \\ y_i \geq 0\ w.\ (i = 1, ..., m, j = 1, ..., n) \end{array} \right. \end{cases}$$

18.3.2 Matrix form

In the matrix, representation, one ca sees that the transformation is pretty straightforward.

- Both the primal and the dual problems make use of the same matrix, transposed in the dual space.

One should not that, as stated before, the inequalities are reversed, Since each inequality can be replaced by an equality and a slack variable, this means each primal variable corresponds to a dual slack variable, and each dual variable corresponds to a primal slack variable.

Base problems

Primal problem

map	x_1	x_2	x_3	x_4	b
m_1	a_{11}	a_{12}	a_{13}	a_{14}	b_1
m_2	a_{21}	a_{22}	a_{23}	a_{24}	b_2
m_3	a_{31}	a_{32}	a_{33}	a_{34}	b_3
Δ	c_1	c_2	c_3	c_4	0

Dual problem

map	y_1	y_2	y_3	c
m_1	a_{11}	a_{21}	a_{31}	c_1
m_2	a_{12}	a_{22}	a_{32}	c_2
m_3	a_{13}	a_{23}	a_{33}	c_3
m_4	a_{14}	a_{24}	a_{34}	c_3
Δ	b_1	b_2	b_3	0

Introducting slack variables

The introduction of the slack variables enable to map all variables between both problems

Primal problem

map	x_1	x_2	x_3	x_4	λ_5	λ_6	λ_7	b
m_1	a_{11}	a_{12}	a_{13}	a_{14}	1	0	0	b_1
m_2	a_{21}	a_{22}	a_{23}	a_{24}	0	1	0	b_2
m_3	a_{31}	a_{32}	a_{33}	a_{34}	0	0	1	b_3
Δ	c_1	c_2	c_3	c_4	0	0	0	0

Dual problem

map	y_1	y_2	y_3	α_4	α_5	α_6	α_7	c
m_1	a_{11}	a_{21}	a_{31}	1	0	0	0	c_1
m_2	a_{12}	a_{22}	a_{32}	0	1	0	0	c_2
m_3	a_{13}	a_{23}	a_{33}	0	0	1	0	c_3
m_4	a_{14}	a_{24}	a_{34}	0	0	0	1	c_3
Δ	b_1	b_2	b_3	0	0	0	0	0

which enables us to build the following correspondans matrix, knowing the *base variables* of the primal problem maps to the *off-base* variables of the dual problem and the other way around:

primal	x_1	x_2	x_3	x_4	λ_5	λ_6	λ_7
dual	α_4	α_5	α_6	α_7	y_1	y_2	y_3

The coefficients we will find for these variables in one of each of the problem are the same that applies to the variables of the other problem following this mapping.

18.3.3 Example

Primal problem

$$\begin{cases} min\ z = 4x_1 + x_2 + 5x_3 + 3x_4 \\ WR \begin{cases} x_1 - x_2 - x3 + 3x_4 \leq 1 \\ 5x_1 + x_2 + 3x_3 + 8x_4 \leq 55 \\ -x_1 + x_2 + 3x_3 - 5x_4 \leq 3 \end{cases} \\ x_1, x_2, x_3, x_4 \geq 0 \end{cases}$$

Dual problem

$$\leftrightarrow \begin{cases} max\ w = y_1 + 55y_2 + 3y_3 \\ WR \begin{cases} y_1 + 5y_2 - y_3 \geq 4 \\ -y_1 + y_2 + 2y_3 \geq 1 \\ -y_1 + 3y_2 + 3y_3 \geq 5 \\ 3y_1 + 8y_2 - 5y_3 \geq 3 \end{cases} \\ y_1, y_2, y_3 \geq 0 \end{cases}$$

18.4 Notes

18.4.1 Which is better ?

Whenever the number of constraints m is greater than $>$ the number of free variables $n \Rightarrow$ whe're better off solving the dual problem as this will be quicker (less iterations)

18.4.2 Primal-Dual correspondance

Whenever the primal problem as an optimal solution $x = ({x_1}^*, ..., {x_n}^*)$, then the dual problem as a solution $y = ({x_1}^*, ..., {y_m}^*)$ such that:

$$z = \sum_{j=1}^{n} c_j {x_j}^* = \sum_{i=1}^{m} b_i {y_i}^* = w$$

18.5 Example

TODO : Build a complete solution taking one example of Cedric Bilat's lecture.