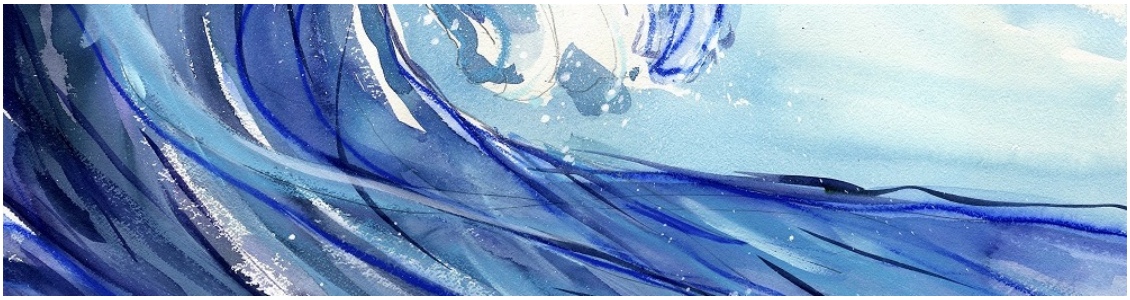
Une introduction aux sciences des donnÃ©es



Gabriel Peyré
CNRS & DMA
École Normale Supérieure
gabriel.peyre@ens.fr
https://mathematical-tours.github.io

March 18, 2019

PrÃ©sentation

Les quatres chapitres de ce texte sont indeÌpendants et preÌsentent des introductions en douceur aÌ quelques fondements matheÌmatiques importants des sciences des donnÃ©es :

- Le chapitre **??** prÃ©sente la thÃ©orie de Shannon sur la compression et insiste en particulier sur l'entropie liÃ©e au codage de l'information.
- Le chapitre **??** prÃ©sente les bases du traitement d'images, en particulier des traitements importants (quantification, dÃ©bruitage, couleurs).
- Le chapitre **??** prÃ©sente la thÃ©orie de l'Ã©chantillonnage, allant de l'Ã©chantillonnage classique de Shannon Ã l'Ã©chantillonnage comprimÃ©. Il constitue Ã©galement une introduction Ã la rÃ©gularisation des problÃ¨mes inverses.
- Le chapitre 4 prÃ©sente le transport optimal et ses applications.

Le niveau d'exposition pour les deux premiers chapitres est Ã©lÃ©mentaire. Le dernier chapitre prÃ©sente des concepts et rÃ©sultats mathÃ©matiques plus avancÃ©s.

Contents

Chapter 1

Claude Shannon et la compression des données

L'immense majorité des données (texte, son, image, vidéo, etc.) sont stockées et manipulées sous forme numérique, c'est-à-dire à l'aide de nombres entiers qui sont convertis en une succession de bits (des 0 et des 1). La conversion depuis le monde analogique continu vers ces représentations numériques discrètes est décrite par la théorie élaborée par Claude Shannon (30 avril 1916–24 février 2001), le père fondateur de la théorie de l'information. L'impact de cette théorie sur notre société est absolument colossal. Pourtant son nom est quasi inconnu du grand public. Le centenaire de la naissance de Claude Shannon est donc une bonne excuse pour présenter l'œuvre d'un très grand scientifique.

1.1 Données numériques et codage

Dans le monde numérique qui nous entoure, toutes les données (images, films, sons, textes, etc.) sont codées informatiquement sous la forme d'une succession de 0 et des 1. Ce codage n'est pas limité au stockage sur des ordinateurs, il est aussi central pour les communications sur internet (envois de courriels, Â« streaming Â» vidéo[1], etc.) ainsi que pour des applications aussi diverses que les lecteurs de musique, les liseuses électroniques ou les téléphones portables.

Cependant, les données (par exemple du texte, des sons, des images ou des vidéos) sont initialement représentées sous la forme d'une succession de *symboles*, qui ne sont pas forcément des 0 ou des 1. Par exemple, pour le cas d'un texte, les symboles sont les lettres de l'alphabet. Pour les cas des images, il s'agit des valeurs des pixels. Il faut donc pouvoir convertir cette suite de symboles en une suite de 0 et de 1. Il faut également pouvoir le faire de façon économe, c'est-à-dire en utilisant la suite la plus courte possible. Ceci est crucial pour pouvoir stocker efficacement ces données sur un disque dur, où bien les transmettre rapidement sur le réseau internet. Cette problématique de *compression* est devenue un enjeu majeur car les données stockées et transmises croissent de façon exponentielle.

La théorie élaborée par Claude Shannon décrit les bases théoriques et algorithmiques de ce codage. Il a formalisé mathématiquement les trois étapes clés de la conversion depuis le monde analogique vers le monde numérique :

[1] https://fr.wikipedia.org/wiki/Streaming

(i) l'*échantillonnage*[2], qui permet de passer de données continues à une succession de nombres ;

(ii) le *codage*[3] (on parle aussi de compression), qui permet de passer à une succession plus compacte de 0 et de 1 (on parle de code binaire) ;

(iii) le *codage correcteur d'erreurs*[4], qui rend le code robuste aux erreurs et aux attaques.

Pour chacune de ces étapes, Claude Shannon a établi dans [**? ?**], sous des hypothèses précises sur les données et le canal de transmission, des Â« bornes d'optimalité Â». Ces bornes énoncent des limites de performance indépassables, quelle que soit la méthode utilisée. Par exemple, pour la phase de codage (ii), cette borne correspond à la taille théorique minimale des messages binaires permettant de coder l'information voulue. Dans la deuxième moitié du 20^e siècle, des méthodes et des algorithmes de calculs efficaces ont été élaborés permettant d'atteindre les bornes de Shannon, débouchant au 21^e siècle sur l'explosion de l'ère numérique. Cet article se concentre sur la partie (ii) et présente les bases de la compression des données telles que définies par Claude Shannon. Pour la partie (iii), on pourra par exemple consulter cet article d'images des mathématiques[5].

Vous pourrez trouver à la fin de cet article un glossaire récapitulant les termes les plus importants.

1.2 Codage et décodage

Nous allons maintenant décrire et étudier la transformation (le codage) depuis la suite de symboles $\{0, 1, 2, 3\}$ vers un code binaire, c'est-à-dire une suite de 0 et de 1.

1.2.1 Exemple d'une image

Dans la suite de cet article, je vais illustrer mes propos à l'aide d'images en niveaux de gris. Une telle image est composée de pixels. Pour simplifier, nous allons considérer seulement des pixels avec 4 niveaux de gris :

- 0 : noir
- 1 : gris foncé,
- 2 : gris clair,
- 3 : blanc.

Cependant, tout ce qui va être décrit par la suite se généralise à un nombre arbitraire de niveaux de gris (en général, les images que l'on trouve sur internet ont 256 niveaux) et même aux images couleurs (que l'on peut décomposer en 3 images monochromes, les composantes rouge, vert et bleue).

La figure **??** montre un exemple d'une image avec 4 niveaux de gris, avec un zoom sur un sous-ensemble de 5×5 pixels.

Nous allons nous concentrer sur cet ensemble de 25 pixels (le reste de l'image se traite de la même façon). Si on met les unes à la suite des autres les 25 valeurs correspondantes, on obtient la suite suivante de symboles, qui sont des nombres entre 0 et 3

$$(0, 1, 3, 2, 0, 3, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 2, 1).$$

[2] https://fr.wikipedia.org/wiki/échantillonnage_(signal)
[3] https://fr.wikipedia.org/wiki/Compression_de_données
[4] https://fr.wikipedia.org/wiki/Code_correcteur
[5] http://images.math.cnrs.fr/Qui-est-ce

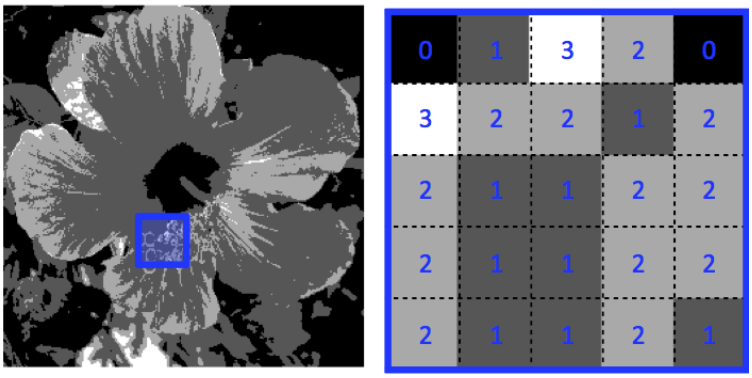


Figure 1.1: Une image en niveaux de gris et un zoom sur un carré de 5×5 pixels.

1.2.2 Codage uniforme

L'étape de codage procède donc en associant à chacun des symboles $\{0, 1, 2, 3\}$ un mot de code, qui est une suite de 0 et de 1.

Une stratégie possible est d'utiliser le codage

$$0 \mapsto 00, \quad 1 \mapsto 01, \quad 2 \mapsto 10, \quad 3 \mapsto 11.$$

Il s'agit d'un cas particulier de codage *uniforme*, qui associe à chaque symbole un mot de code de longueur fixe (ici de longueur constante 2).

Ainsi, la suite de symboles $(0, 1, 3)$ est codée comme

$$(0, 1, 3) \overset{\text{codage}}{\longmapsto} (00, 01, 11) \overset{\text{regroupement}}{\longmapsto} 000111.$$

La suite complète des symboles correspondant à l'image de 5×5 pixels montrée plus haut donnera le code

$$0001111000111010011010010110101001011010100101100 1.$$

La longueur (c'est-à-dire le nombre de 0 et de 1) de la suite de 0 et de 1 utilisée pour coder un message se mesure en nombre de *bits*. En utilisant le codage uniforme précédent, qui utilise 2 bits par symboles, comme l'on doit coder 25 symboles, on obtient une longueur

$$\bar{\mathcal{L}} = 25 \times 2 = 50 \text{ bits}$$

Le *bit* (Â« binary digit Â») est l'unité fondamentale de l'information, et a été introduite par John Tukey[6], qui était un collaborateur de Claude Shannon.

1.2.3 Logarithme et codage uniforme

Si le nombre N de symboles possibles (ici $N = 4$) est une puissance de 2, c'est à dire que $N = 2^\ell$ (ici $N = 4 = 2^2$ donc $\ell = 2$), on peut toujours construire un tel code *uniforme* où l'on associe à chaque symbole son écriture binaire. On a donné plus haut l'exemple du codage uniforme de $N = 4$ symboles, et le cas de $N = 8$ (donc $\ell = 3$) symboles correspond au codage

$$0 \mapsto 000, \quad 1 \mapsto 001, \quad 2 \mapsto 010, \quad 3 \mapsto 011,$$

[6] https://fr.wikipedia.org/wiki/John_Tukey

$$4 \mapsto 100, \quad 5 \mapsto 101, \quad 6 \mapsto 110, \quad 7 \mapsto 111.$$

Cette écriture binaire a une longueur ℓ, que l'on appelle le logarithme en base de 2[7] de N, ce que l'on note

$$N = 2^{\ell} \quad \Longleftrightarrow \quad \log_2(N) \overset{\text{def.}}{=} \ell.$$

La définition de $\log_2(x)$ s'étend aussi au cas où x n'est pas une puissance de 2, en utilisant la définition $\log_2(x) \overset{\text{def.}}{=} \ln(x)/\ln(2)$, où ln est le *logarithme népérien*. Dans ce cas, $\log_2(x)$ n'est pas un nombre entier. Pour un nombre réel strictement positif x, le logarithme vérifie $\log_2(1/x) = -\log_2(x)$, donc par exemple, on a $\log_2(1/4) = -\log_2(4) = 2$.

1.2.4 Codage à longueur variable

Une question importante est de savoir si l'on peut faire mieux (c'est-à-dire utiliser moins de bits pour coder la même suite de symboles). On peut par exemple utiliser à la place d'un code uniforme, le codage suivant

$$0 \mapsto 001, \quad 1 \mapsto 01, \quad 2 \mapsto 1, \quad 3 \mapsto 000.$$

Avec un tel codage, la suite de symboles $(0, 1, 3)$ est codée comme

$$(0, 1, 3) \overset{\text{codage}}{\longmapsto} (001, 01, 000) \overset{\text{regroupement}}{\longmapsto} 00101000.$$

La suite complète des symboles correspondant à l'image de 5×5 pixels donnera le code

$$00101000100100011011101011110101111010101101.$$

La longueur du code binaire obtenue est donc maintenant

$$\bar{\mathcal{L}} = 42 \text{ bits}$$

Ceci montre qu'on peut donc faire mieux qu'avec un codage *uniforme* en utilisant un codage *variable*, qui associe à chaque symbole un code de longueur variable.

On peut également définir le nombre de bits moyen par symbole $\mathcal{L}$, qui se calcule, ici pour une suite de 25 symboles, comme

$$\mathcal{L} \overset{\text{def.}}{=} \frac{\bar{\mathcal{L}}}{25} = \frac{42}{25} = 1.68 \text{ bits}.$$

Par rapport à un codage uniforme, on voit que le nombre de bits moyen par symbole est passé de $\log_2(N) = 2$ bits à 1.68 bits.

1.2.5 Codage préfixe et décodage

Ces codages, uniformes ou à longueur variable, seraient sans intérêt si l'on ne s'assurait pas que le message obtenu est *décodable*, c'est-à-dire que l'on puisse retrouver la suite de symboles à l'origine d'un code binaire. Tous les codages ne permettent pas de faire ce chemin inverse.

Pour les codages uniformes, comme le codage

$$0 \mapsto 00, \quad 1 \mapsto 01, \quad 2 \mapsto 10, \quad 3 \mapsto 11.$$

[7] https://fr.wikipedia.org/wiki/Logarithme_binaire

il suffit de séparer la suite de bits en paquets de longueur $\log_2(N)$ (ici $N = 4$ et $\log_2(N) = 2$) et d'utiliser la table de codage en sens inverse. Ainsi, le code binaire 000111 est décodé comme

$$000111 \overset{\text{séparation}}{\longmapsto} (00, 01, 11) \overset{\text{décodage}}{\longmapsto} (0, 1, 3).$$

Par contre, si l'on considère le codage

$$0 \mapsto 0, \quad 1 \mapsto 10, \quad 2 \mapsto 110, \quad 3 \mapsto 101,$$

alors la suite de bits 1010 peut être décodée de deux façons :

$$1010 \overset{\text{séparation}}{\longmapsto} (10, 10) \overset{\text{décodage}}{\longmapsto} (1, 1),$$

ou bien

$$1010 \overset{\text{séparation}}{\longmapsto} (101, 0) \overset{\text{décodage}}{\longmapsto} (3, 0).$$

Ceci signifie que cette suite peut être décodée soit comme la suite de symboles $(1, 1)$, soit comme la suite $(3, 0)$. On remarque que le mot de codage 10 utilisé pour coder 1 est le début du mot 101 utilisé pour coder 3.

Pour être capable de faire le décodage de façon non ambiguë, il suffit qu'aucun mot du codage ne soit le début d'un autre mot. Lorsque cette condition est satisfaite, on parle de codage *préfixe*[8], et l'on peut donc effectuer progressivement le décodage. On vérifie facilement que c'est bien le cas du codage non uniforme déjà considéré précédemment

$$0 \mapsto 001, \quad 1 \mapsto 01, \quad 2 \mapsto 1, \quad 3 \mapsto 000.$$

Le décodage progressif du message de 25 symboles des pixels de l'image est effectué ainsi :

$$001010001001000110111010111101011110101101 \longrightarrow \text{décode } 0$$
$$0\ 01000100100011011101011110101111010110 1 \longrightarrow \text{décode } 1$$
$$0\ 1\ 000100100011011101011110101111010110 1 \longrightarrow \text{décode } 3$$
$$0\ 1\ \ 3\ 1001000110111010111101011110101101 \longrightarrow \text{décode } 2\ldots$$

1.2.6 Codes et arbres

Comme le montre la figure **??**, en haut à gauche, il est possible de placer l'ensemble des codes binaires de moins de ℓ bits dans un arbre de profondeur $\ell + 1$. Les 2^ℓ mots de longueur exactement ℓ occupent les feuilles, et les mots plus courts sont les nœuds intérieurs.

Les codages préfixes sont alors représentés comme les feuilles des sous-arbres de cet arbre complet. La figure **??**, en haut à droite, montre à quel sous-arbre correspond le code à longueur variable

$$0 \mapsto 001, \quad 1 \mapsto 01, \quad 2 \mapsto 1, \quad 3 \mapsto 000.$$

Une fois que l'on a représenté un codage préfixe comme un sous-arbre binaire, l'algorithme de décodage est particulièrement simple à mettre en œuvre. Lorsque l'on commence le décodage, on se place à la racine, et on descend à chaque nouveau bit lu soit à gauche (pour un 0) soit à droite (pour un 1). Lorsque l'on atteint une feuille du sous-arbre, on émet alors le mot du code correspondant à cette feuille, et l'on redémarre à la racine. La figure précédente illustre le processus de décodage.

[8] https://fr.wikipedia.org/wiki/Code_préfixe

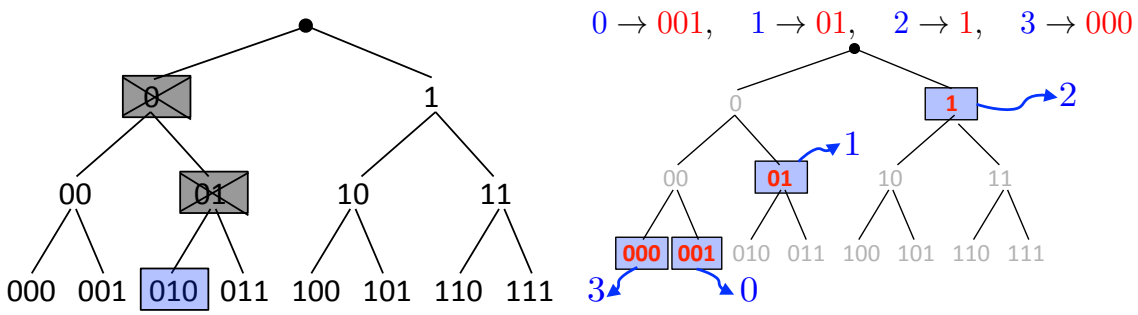


Figure 1.2: Gauche: arbre complet de tous les codes de longueur 3 ; droite : exemple de codage préfixe.

1.3 La borne de Shannon

Après avoir décrit les techniques de codage, nous allons maintenant expliquer la théorie de Shannon, qui analyse la performance de ces techniques (c'est-à-dire le nombre de bits nécessaire au codage) en effectuant une modélisation aléatoire du message à coder (qui est composé d'une suite particulière de symboles).

1.3.1 Code de longueur minimale et modélisation aléatoire

L'utilisation d'un codage préfixe à longueur variable montre que l'on peut obtenir un nombre de bits moyen $\mathcal{L}$ plus faible que le nombre $\log_2(N)$ de bits obtenu par un code uniforme. La question fondamentale, à la fois sur un plan pratique et théorique, est donc de savoir si l'on peut *trouver un codage préfixe donnant lieu à un nombre de bits moyen par symbole minimal.*

Cette question est mal posée, car sa réponse dépend du message qu'il faudra coder, et ce message est en général inconnu a priori. Il faut donc un modèle pour décrire les messages possibles. L'idée fondamentale introduite par Claude Shannon est d'utiliser un modèle probabiliste : on ne sait pas quels messages on aura à coder, mais on suppose qu'on connaît la probabilité d'apparition des symboles composant ce message.

Shannon suppose ainsi que les symboles qui composent le message modélisé sont tirés *indépendamment*[9] selon une variable aléatoire V (la source du message). Ceci signifie que les symboles composants le message modélisé sont des variables aléatoires indépendantes ayant la même distribution que V.

1.3.2 Fréquences empiriques

Afin d'appliquer ce modèle probabiliste à un message donné, on va faire comme si l'on tirait au sort chaque symbole l'un à la suite de l'autre selon des probabilités identiques aux fréquences que l'on observe (en moyenne) dans le cas étudié.

Ceci signifie que l'on impose à la distribution de la source V d'être égale aux fréquences empiriques observées dans le message. Les fréquences empiriques (p_0, p_1, p_2, p_3) sont les fréquences d'apparition des différents symboles $(0, 1, 2, 3)$. Pour la suite des 25 pixels de l'image en niveaux de

[9] https://fr.wikipedia.org/wiki/Indépendances_(probabilité)

gris

$$(0,1,3,2,0,3,2,2,1,2,2,1,1,2,2,2,1,1,2,2,2,1,1,2,1),$$

la fréquence p_1 est égale à $9/25$ car le symbole 1 apparaît 9 fois et que l'on souhaite coder une suite de 25 symboles. La liste des fréquences empiriques pour cette suite de symboles est ainsi

$$p_0 = \tfrac{2}{25}, \quad p_1 = \tfrac{9}{25}, \quad p_2 = \tfrac{12}{25}, \quad p_3 = \tfrac{2}{25}.$$

La modélisation aléatoire impose donc à la variable V d'avoir pour distribution de probabilité (p_0, p_1, p_2, p_3), c'est-à-dire que la probabilité qu'un symbole du message modélisé (supposé généré par la source V) soit égal à $v \in \{0,1,2,3\}$ vaut $\mathbb{P}(V = v) = p_v$.

Ceci constitue un exemple important de modélisation, qui n'est bien sûr pas toujours pertinente mais permet d'analyser finement le problème. Par exemple, dans le cas d'une image, si un pixel est noir, le suivant a de fortes chances de l'être aussi, même si la fréquence globale du noir est faible. Ceci met en défaut l'hypothèse d'indépendance (la section Â« Transformation de l'information Â» détaille cet exemple).

1.3.3 Entropie

Afin de répondre au problème de codage avec un nombre de bits moyen minimum, Shannon a introduit un objet mathématique fondamental : *l'entropie*[10]. L'entropie a été inventée par Ludwig Boltzmann[11] dans le cadre de la thermodynamique[12] et ce concept a été repris par Claude Shannon pour développer sa théorie de l'information. L'entropie de la distribution de la source V est définie par la formule

$$\mathcal{H}_V \overset{\text{def.}}{=} -\sum_{v=0}^{N-1} p_v \times \log_2(p_v).$$

Cette formule signifie que l'on fait la somme, pour tous les symboles v possibles, de la fréquence d'apparition p_v du symbole v multipliée par le logarithme $\log_2(p_v)$ de cette fréquence, puis que l'on prend l'opposé (signe moins) du nombre obtenu.

Comme le logarithme est une fonction croissante, et comme $\log_2(1) = 0$, on a $\log_2(p_v) \leqslant 0$ car $p_v \leqslant 1$ (une probabilité est toujours plus petite que 1). Le signe moins devant la formule définissant l'entropie assure que cette quantité est toujours positive.

Dans notre cas, on a $N = 4$ valeurs pour les symboles, et on utilise donc la formule

$$\mathcal{H}_V \overset{\text{def.}}{=} -p_0 \times \log_2(p_0) - p_1 \times \log_2(p_1) - p_2 \times \log_2(p_2) - p_3 \times \log_2(p_3).$$

Il est à noter que si jamais $p_v = 0$, alors il faut utiliser la convention $p_v \times \log_2(p_v) = 0 \times \log_2(0) = 0$. Cette convention signifie que l'on ne prend pas en compte les probabilités nulles (c'est-à-dire les événements impossibles) dans cette formule. De plus elle est cohérente avec la valeur limite de la fonction $x \mapsto x\ln(x)$ en $x = 0$.

Le but de l'entropie est de quantifier l'incertitude sur les suites de symboles possibles générées par la source V. On peut montrer que l'entropie vérifie

$$0 \leqslant \mathcal{H}_V \leqslant \log_2(N).$$

Les deux valeurs extrêmes correspondent ainsi à des incertitudes respectivement minimale et maximale.

[10] https://fr.wikipedia.org/wiki/Entropie
[11] https://fr.wikipedia.org/wiki/Ludwig_Boltzmann
[12] https://fr.wikipedia.org/wiki/Entropie_(thermodynamique)

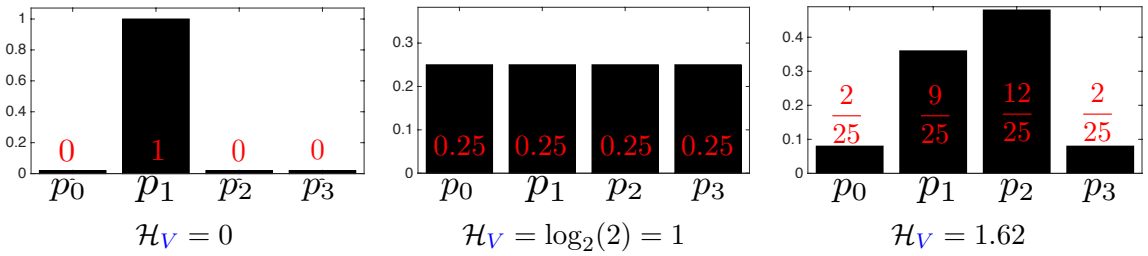


Figure 1.3: Trois exemples de distributions de probabilité avec les entropies correspondantes.

- **Entropie minimale.** L'entropie $\mathcal{H}_V = 0$ est minimale lorsque les fréquences p_v sont toutes nulles sauf une. La figure **??**, gauche, montre le cas où $p_1 = 1$ et toutes les autres probabilités sont nulles.

  Dans ce cas, on a

  $$\mathcal{H}_V = -0 \times \log_2(0) - 1 \times \log_2(1) - 0 \times \log_2(0) - 0 \times \log_2(0) = 0,$$

  où l'on rappelle que $\log_2(1) = 0$ et que, par convention, on a $0 \times \log_2(0) = 0$. Ceci correspond à la modélisation d'une suite constante de symboles, et la source génèrera par exemple avec probabilité 1 la suite suivante de 25 symboles

  $$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0).$$

- **Entropie maximale.** Au contraire, $\mathcal{H}_V = \log_2(N)$ est maximale lorsque toutes les fréquences sont égales, $p_v = 1/N$. Dans notre cas où $N = 4$, on a en effet

  $$\mathcal{H}_V = -\tfrac{1}{4} \times \log_2(\tfrac{1}{4}) - \tfrac{1}{4} \times \log_2(\tfrac{1}{4}) - \tfrac{1}{4} \times \log_2(\tfrac{1}{4}) - \tfrac{1}{4} \times \log_2(\tfrac{1}{4}) = \log_2(4) = 2,$$

  où l'on a utilisé le fait que $\log_2(1/x) = -\log_2(x)$ et donc en particulier $\log_2(\frac{1}{4}) = -\log_2(4)$. La figure **??**, centre, suivante montre l'histogramme correspondant à ce cas.

  Cette situation correspond intuitivement à la modélisation d'une suite maximalement *incertaine*. Voici par exemple deux suites de 25 symboles générés par une telle source V

  $$(2,2,1,1,3,0,3,3,3,0,1,1,2,0,2,0,2,1,3,2,0,2,2,1,3),$$

  $$(3,3,1,2,0,0,2,2,1,3,2,2,3,3,2,0,0,3,0,1,3,0,1,1,2).$$

- **Entropie intermédiaire.** Les situations intermédiaires entre ces deux extrêmes correspondent à des entropies intermédiaires. Par exemple, on peut considérer la distribution des 25 pixels considérés au début de cet article, qui correspondent au message

  $$(0,1,3,2,0,3,2,2,1,2,2,1,1,2,2,2,1,1,2,2,2,1,1,2,1).$$

  Pour cette distribution, on rappelle que l'on a les probabilités

  $$p_0 = \tfrac{2}{25}, \quad p_1 = \tfrac{9}{25}, \quad p_2 = \tfrac{12}{25}, \quad p_3 = \tfrac{2}{25},$$

  la figure **??**, droite, montre l'histogramme correspondant à ces valeurs.

  L'entropie vaut alors

  $$\mathcal{H}_V = -\tfrac{2}{25} \times \log_2(\tfrac{2}{25}) - \tfrac{9}{25} \times \log_2(\tfrac{9}{25}) - \tfrac{12}{25} \times \log_2(\tfrac{12}{25}) - \tfrac{2}{25} \times \log_2(\tfrac{2}{25}) \approx 1.62,$$

  ce qui correspond bien une valeur Â« intermédiaire Â» de l'entropie.

1.3.4 Nombre de bits moyen d'une source

Dans la suite, on note c_v le code associé à un symbole v. On note $L(c_v)$ la longueur (i.e. le nombre de bits) de chaque mot c_v de code. Pour un codage uniforme, alors la longueur est constante $L(c_v) = \log_2(N)$. Par contre, si l'on prend l'exemple du codage variable

$$0 \mapsto c_0 \overset{\text{def.}}{=} 001, \quad 1 \mapsto c_1 \overset{\text{def.}}{=} 01, \quad 2 \mapsto c_2 \overset{\text{def.}}{=} 1, \quad 3 \mapsto c_3 \overset{\text{def.}}{=} 000,$$

alors $L(c_0) = L(001) = 3$.

On remarque que l'on peut calculer le nombre de bit moyen $\mathcal{L}$ du codage d'un message à l'aide des fréquences empiriques comme suit :

$$\mathcal{L} = \sum_{v=0}^{N-1} p_v \times L(c_v).$$

Cette formule signifie que l'on fait la somme, pour tous les symboles v possibles, de la fréquence d'apparition p_v du symbole multipliée par la longueur $L(c_v)$ du mot de code c_v. Par exemple, dans notre cas, pour $N = 4$, on a la formule

$$\mathcal{L} = p_0 \times L(c_0) + p_1 \times L(c_1) + p_2 \times L(c_2) + p_3 \times L(c_3).$$

Dans le cadre de la modélisation aléatoire à l'aide d'une source V, on va noter $\mathcal{L}_V$ ce nombre de bit moyen, qui est associé à la source V ayant la distribution $(p_v)_v$.

1.3.5 Borne de Shannon pour le codage

Claude Shannon a montré dans son article [**?**] que l'entropie permettait de borner le nombre de bits moyen $\mathcal{L}_V$ dans le cadre de ce modèle aléatoire. Il a en effet montré que pour tout codage préfixe, on a

$$\mathcal{H}_V \leqslant \mathcal{L}_V.$$

Il s'agit d'une borne inférieure, qui dit qu'aucun codage préfixe ne peut faire mieux que cette borne.

Ce résultat est fondamental, car il décrit une limite indépassable, quelle que soit la technique de codage préfixe utilisée. Sa preuve est trop difficile pour être exposée ici, elle utilise la représentation sous forme d'arbre détaillée plus haut à la section **??**, on pourra regarder par exemple [**?**] pour obtenir tous les détails. Cette preuve montre qu'il faut dépenser en moyenne au moins $-\log_2(p_v)$ bits (qui est, comme on l'a déjà vu, toujours un nombre positif) pour coder un symbole v si l'on veut avoir un codage efficace. Les symboles les plus fréquents doivent nécessiter moins de bits, car p_v est plus petit, donc la longueur optimale $-\log_2(p_v)$ l'est également. Ceci qui est très naturel, comme on peut en particulier le voir pour les deux cas extrêmes :

- **Entropie minimale.** Si $\mathcal{H}_V = 0$, alors avec probabilité 1, la suite de symboles est composée d'un unique symbole. Dans ce cas de figure, l'utilisation d'un codage préfixe est très inefficace, car celui-ci doit utiliser au moins un bit par symbole i.e. $\mathcal{L}_V \geqslant 1$, et donc un tel codage est loin d'atteindre la borne de Shannon.

  L'entropie étant nulle, la borne dit que l'on souhaiterait ne rien dépenser pour le codage. Ceci est logique, car il n'y a pas besoin de coder une telle suite (puisque c'est toujours la

même). Des techniques de codage plus avancées (par exemple le codage arithmétiques[13] [?]) permettent de contourner ce problème et atteignent la borne de Shannon quand le nombre de symboles à coder tend vers l'infini.

- **Entropie maximale.** Si $\mathcal{H}_V = \log_2(N)$, alors tous les symboles sont équiprobables, donc on doit utiliser des mots de code de même longueur pour tous les symboles, ce qui est obtenu par un code uniforme. Comme on l'a vu plus haut, un tel code nécessite $\mathcal{L}_V = \log_2(N) = \mathcal{H}_V$ bits par symbole, et donc la borne inférieure de Shannon est atteinte dans ce cas.

- **Entropie intermédiaire.** Pour le cas de la distribution des 25 pixels considérés au début de cet article, qui correspondent au message

$$(0, 1, 3, 2, 0, 3, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 2, 1),$$

on rappelle que l'entropie et le nombre moyen de bits, qui ont déjà été calculés, valent respectivement

$$\mathcal{H}_V \approx 1.62 \text{ bits} \quad \text{et} \quad \mathcal{L}_V = 1.68 \text{ bits}.$$

Ces valeurs sont bien en accord avec la borne de Shannon, et montrent que le codage préfixe utilisé permet d'être assez proche de cette borne.

On peut se demander si cette borne est précise, et s'il est possible de construire des codes atteignant la borne de Shannon dans tous les cas (et pas juste les deux cas extrêmes). Huffmann a proposé dans [?] une construction d'un codage Â« optimal Â» (i.e. ayant la longueur moyenne $\mathcal{L}_V$ minimale pour une source V donnée) à l'aide d'un algorithme élégant. La longueur moyenne obtenue par ce codage vérifie

$$\mathcal{H}_V \leqslant \mathcal{L}_V \leqslant \mathcal{H}_V + 1.$$

Le fait que cette longueur moyenne puisse être potentiellement aussi grande que $\mathcal{H}_V + 1$ (et donc assez différente de la borne inférieure de Shannon $\mathcal{H}_V$) provient du fait que la longueur $L(c_v)$ d'un mot c_v du code est un nombre entier, alors que la longueur optimale devrait être $-\log_2(p_v)$, qui n'est pas en général un nombre entier. Pour pallier ce problème, il faut coder les symboles par groupes, ce qui peut être effectué de façon efficace à l'aide des codages arithmétiques[14] [?], qui atteignent la borne de Shannon lorsque l'on code une suite infinie de symboles.

La théorie de Shannon permet donc de borner la longueur moyenne, ce qui donne une information importante sur la performance d'une méthode de codage pour une source donnée. Cette borne ne donne cependant pas d'information sur d'autres quantités statistiques potentiellement intéressantes, telles que la longueur maximale ou la longueur médiane.

1.3.6 Transformation de l'information

La borne de l'entropie précédente fait l'hypothèse que les symboles qui composent le message à coder sont générés de façon *indépendante* par la source V. Cette hypothèse permet une analyse mathématique simple du problème, mais elle est en général fausse pour des données complexes, comme par exemple pour l'image montrée à la figure suivante. En effet, on voit bien que la valeur d'un pixel n'est pas du tout indépendante de celles de ses voisins. Par exemple, il y a de grandes zones homogènes où la valeur des pixels est quasi-constante.

[13] https://fr.wikipedia.org/wiki/Codage_arithmétique
[14] https://fr.wikipedia.org/wiki/Codage_arithmétique

Afin d'améliorer les performances de codage, et obtenir des méthodes de compression d'image efficaces, il est crucial de retransformer la suite de symboles, afin de réduire son entropie en exploitant les dépendances entre les pixels. Une transformation très simple permettant de le faire consiste à remplacer les valeurs des P pixels $(v_i)_{i=1}^{P}$ par celles de leurs différences $(d_i \overset{\text{def.}}{=} v_i - v_{i-1})_{i=1}^{P-1}$. En effet, dans une zone uniforme, les différences successives vont être nulles car les pixels ont la même valeur. La figure **??** montre comment effectuer un tel calcul. Elle montre aussi que cette transformation est bijective, c'est-à-dire que l'on peut revenir aux valeurs d'origine $(v_i)_i$ en effectuant une sommation progressive des différences, c'est-à-dire en calculant

$$v_i = v_0 + \sum_{j=1}^{i} d_j.$$

Afin de pouvoir faire cette inversion, il faut bien sûr avoir conservé la valeur v_0 du premier pixel. La bijectivité de la transformation

$$(v_0, \ldots, v_{P-1}) \longmapsto (v_0, d_1, \ldots, d_{P-1})$$

est cruciale pour pouvoir faire le décodage et afficher l'image décodée.

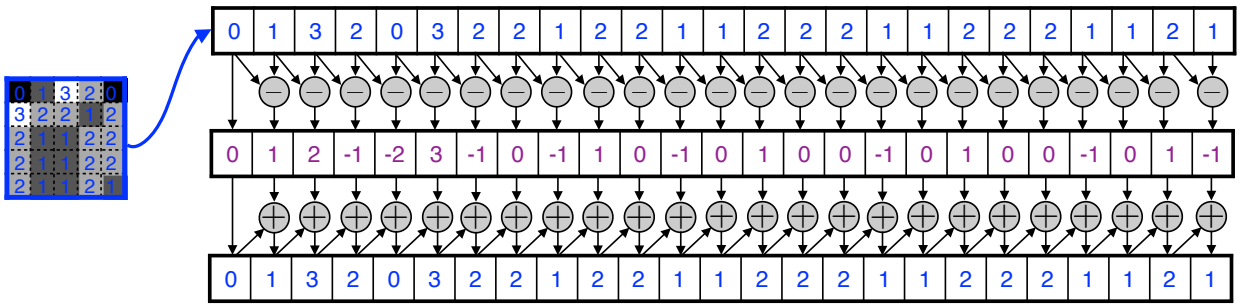


Figure 1.4: Représentation par différences

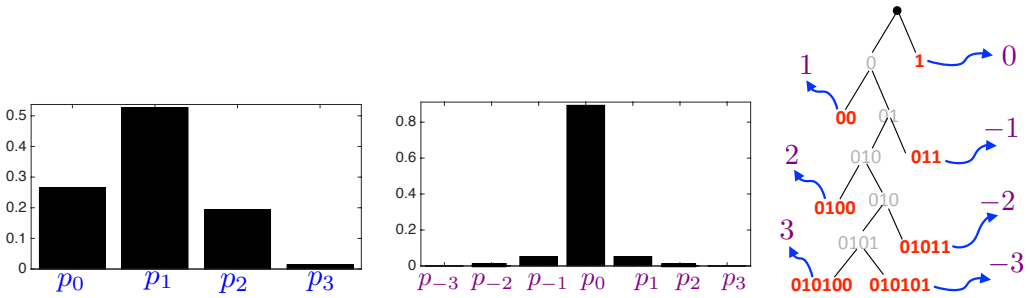
Comme les pixels peuvent prendre les valeurs $\{0, 1, 2, 3\}$, les différences peuvent prendre quant à elles les valeurs $\{-3, \ldots, 3\}$. Elles peuvent en particulier être négatives (ce qui ne pose pas de problème particulier pour définir un codage). La figure suivante compare les histogrammes des pixels et des différences. On constate que l'histogramme des différences est beaucoup plus Â« piqué Â» au voisinage de 0, ce qui est logique, car de nombreuses différences (correspondant aux zones homogènes) sont nulles ou petites. L'entropie $\mathcal{H}_D$ de l'histogramme des différences (que l'on peut modéliser avec une source D) est donc nettement plus faible que l'entropie $\mathcal{H}_V$ des pixels.

La figure **??** montre une comparaison des histogrammes des valeurs des pixels et des différences, calculés sur l'ensemble de l'image (et pas uniquement sur le sous-ensemble de 25 pixels initial). Elle montre également l'arbre d'un codage préfixe optimal (calculé par l'algorithme de Huffman [**?**]) associé à l'histogramme des différences.

Cet arbre correspond au codage

$$-3 \mapsto 010101, -2 \mapsto 01011, -1 \mapsto 011, 0 \mapsto 1, 1 \mapsto 00, 2 \mapsto 0100, 3 \mapsto 010100.$$

Ce codage a une longueur moyenne $\mathcal{L} \approx 1.16$ bits. Ce nombre moyen est bien conforme à la borne de l'entropie, et il est significativement plus petit que la longueur moyenne associée à l'histogramme



$\mathcal{H}_V \approx 1.54, \mathcal{L} \approx 1.67$ $\mathcal{H}_D \approx 0.61, \mathcal{L} \approx 1.16$ Arbre de codage

Figure 1.5: Comparaison des histogrammes des valeurs des pixels et des différences, et un arbre de codage de ces différences.

des pixels (1.67 bits), qui est elle-même plus petite que la longueur moyenne associée à un codage uniforme ($\log_2(4) = 2$ bits). Si l'on répercute ces longueurs au codage de la totalité de l'image de 256×256 en niveau de gris, on obtient ainsi les gains suivants, où 1 ko $= 8 \times 1024$ bits est un *kilo octet*.

Codage uniforme		Codage des pixels		Codage des différences
16.3 ko	$\longrightarrow$	13.7 ko	$\longrightarrow$	9.5 ko

Les méthodes les plus performantes de compression d'image utilisent des transformations plus complexes, et exploitent de façon plus fine la régularité locale des images. C'est le cas de la méthode de compression JPEG-2000[15], qui est considérée comme la plus efficace à l'heure actuelle, et qui utilise la théorie des ondelettes[16], voir le livre [**?**] pour plus de détails. Il existe bien d'autres cas où la non-indépendance des symboles peut être utilisée pour améliorer les performances de codage. Un exemple important est celui de la suite des lettres composant un texte.

1.4 Conclusion

La théorie mathématique initiée par Claude Shannon définit un cadre de pensée nécessaire à l'élaboration de techniques efficaces pour l'acquisition, le traitement, le stockage et la transmission des données sous forme numérique. Ce sont ces techniques qui ont révolutionné les communications et l'informatique durant la deuxième moitié du 20^e siècle, et ont permis la croissance d'internet au début du 21^e siècle. Sans les apports révolutionnaires de Shannon, vous ne pourriez pas partir en vacances avec votre bibliothèque entière dans votre liseuse électronique, et tous les épisodes de *Game of Thrones* sur votre tablette!

Pour obtenir plus de détails sur la théorie de l'information, on pourra consulter [**?**], pour son utilisation en traitement du signal et de l'image, on pourra regarder [**?**]. Les codes informatiques permettant de reproduire les figures de cet article sont disponibles en ligne[17], et d'autres codes sont accessibles sur le site `www.numerical-tours.com` [**?**].

[15] `https://fr.wikipedia.org/wiki/JPEG_2000`
[16] `https://fr.wikipedia.org/wiki/Ondelette`
[17] `https://github.com/gpeyre/2016-shannon-theory`

Glossaire

– **Pixel** : emplacement sur la grille carrée d'une image, parfois utilisé pour faire référence à la valeur associée.
– **Symbole** : élément v d'un ensemble fini, par exemple $\{0, \dots, N-1\}$.
– **Code** : succession de 0 et de 1 utilisé pour coder un symbole v.
– **Codage** : ensemble des correspondances entre les symboles v et les codes associés, par exemple $2 \mapsto 10$. Fait aussi référence à l'action de remplacer une suite de symboles par un ensemble de bits.
– **Distribution empirique** : fréquence p_v d'apparition des symboles v dans les suites de symboles à coder.
– **Histogramme** : représentation graphique de la distribution empirique, pouvant aussi par extension désigner cette distribution.
– **Source** : variable aléatoire V modélisant les symboles, avec la distribution $\mathbb{P}(V = v) = p_v$.
– **Entropie** : $\mathcal{H}_V$ est un nombre positif associé à la source V, et qui dépend de sa distribution de probabilité $(p_v)_v$.
– **Nombre de bits moyen d'une suite :** $\mathcal{L}$ est associé au codage d'une suite de symboles.
– **Nombre de bits moyen de la source :** $\mathcal{L}_V$ est associé au codage des symboles générés par V.

Remerciements

Je remercie Marie-Noëlle Peyré, Gwenn Guichaoua, François Béguin, Gérard Grancher, Aurélien Djament et François Sauvageot pour leurs relectures attentives.

L'image de la fleur est due à Maïtine Bergounioux. L'image de Shannon utilisée pour le logo de l'article est due à l'utilisateur telehistoriska du site flickr (sous license CC BY-NC 2.0).

Chapter 2

Le traitement d'images

Les appareils numériques photographient de manière très précise le monde qui nous entoure. L'utilisateur souhaite pouvoir stocker avec un encombrement minimal ses photos sur son disque dur. Il souhaite également pouvoir les retoucher afin d'améliorer leur qualité. Cet article présente les outils mathématiques et informatiques qui permettent d'effectuer ces différentes tâches. Il reprend en partie le contenu de l'article publié sur le site web *Images des mathématiques*[1].

2.1 Les pixels d'une image

Une image numérique en niveaux de gris est un tableau de valeurs. Chaque case de ce tableau, qui stocke une valeur, se nomme un pixel. En notant n le nombre de lignes et p le nombre de colonnes de l'image, on manipule ainsi un tableau de $n \times p$ pixels. La figure **??**, gauche, montre une visualisation d'un tableau carré avec $n = p = 240$, ce qui représente 240×240=57600 pixels. Les appareils photos numériques peuvent enregistrer des images beaucoup plus grandes, avec plusieurs millions de pixels.

Les valeurs des pixels sont enregistrées dans l'ordinateur ou l'appareil photo numérique sous forme de nombres entiers entre 0 et $255 = 2^8 - 1$, ce qui fait 256 valeurs possibles pour chaque pixel. La valeur 0 correspond au noir, et la valeur 255 correspond au blanc. Les valeurs intermédiaires correspondent à des niveaux de gris allant du noir au blanc. La figure **??** montre un sous-tableau de 6×6 pixels extrait de l'image précédente. On peut voir à la fois les valeurs qui composent le tableau et les niveaux de gris qui permettent d'afficher l'image à l'écran.

2.2 Stocker une image

2.2.1 Écriture binaire

Stocker de grandes images sur le disque dur d'un ordinateur prend beaucoup de place. Les nombres entiers sont stockés en écriture binaire, c'est-à-dire sous la forme d'une succession de 0 et de 1. Chaque 0 et chaque 1 se stocke sur une unité élémentaire de stockage, appelée bit. Pour obtenir l'écriture binaire d'un pixel ayant comme valeur 179, il faut décomposer cette valeur comme somme de puissances de deux. On obtient ainsi

$$179 = 2^7 + 2^5 + 2^4 + 2 + 1,$$

[1] http://images.math.cnrs.fr/Le-traitement-numerique-des-images.html

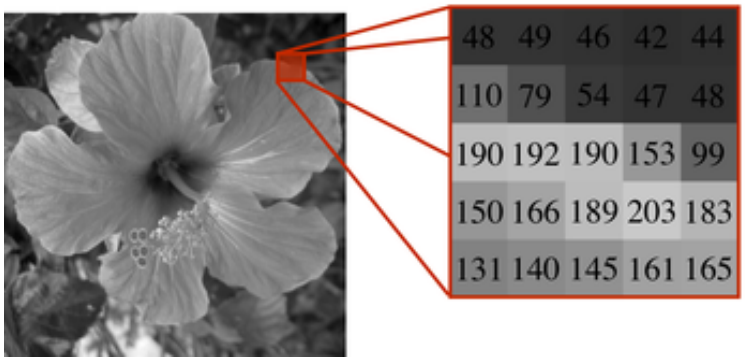


Figure 2.1: *Sous image de taille* 5×5

où l'on a pris soin d'ordonner les puissances de deux par ordre décroissant. Afin de faire mieux apparaître l'écriture binaire, on ajoute "1×" devant chaque puissance qui apparaît dans l'écriture, et "0×" devant les puissances qui n'apparaissent pas

$$179 = 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0.$$

Avec une telle écriture, la valeur de chaque pixel, qui est un nombre entre 0 et 255, nécessite $\log_2(256) = 8$ bits. L'écriture binaire de la valeur 179 du pixel est ainsi $(1, 0, 1, 1, 0, 0, 1, 1)$. On peut écrire toute valeur entre 0 et 255 de cet manière, ce qui nécessite d'utilisation de 8 bits. Il y a en effet 256 valeurs possibles, et $256 = 2^8$. Pour stocker l'image complète, on a donc besoin de $n \times p \times 8$ bits. Pour l'image montrée aux figure précédentes, on a ainsi besoin de

$$256 \times 256 \times 8 = 524288 \text{ bits.}$$

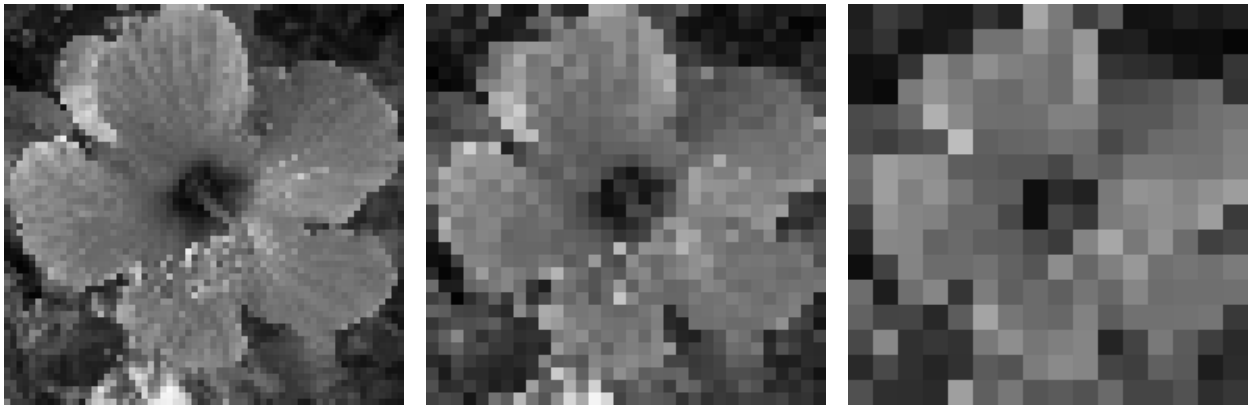
On utilise le plus souvent l'octet (8 bits) comme unité, de sorte que cette image nécessite 57,6ko (kilo octets).

2.2.2 Sous-échantilloner une image

Afin de réduire la place de stockage d'une image on peut diminuer le nombre de pixels. La façon la plus simple d'effectuer cette réduction consiste à supprimer des lignes et des colonnes dans l'image de départ. La figure **??**, en haut à gauche, montre ce que l'on obtient si l'on retient une ligne sur 4 et une colonne sur 4. On a ainsi divisé par $4 \times 4 = 16$ le nombre de pixels de l'image, et donc également divisé par 16 le nombre de bits nécessaire pour stocker l'image sur un disque dur. Sur la figure **??**, on peut voir les résultats obtenus en enlevant de plus en plus de lignes et de colonnes. Bien entendu, la qualité de l'image se dégrade vite.

2.2.3 Quantifier une image

Une autre façon de réduire la place mémoire nécessaire pour le stockage consiste à utiliser moins de nombres entiers pour chaque valeur. On peut par exemple utiliser uniquement des nombres entiers entre 0 et 3, ce qui donnera une image avec uniquement 4 niveaux de gris. On peut effectuer une conversion de l'image d'origine vers une image avec 4 niveaux de valeurs en effectuant les remplacements:



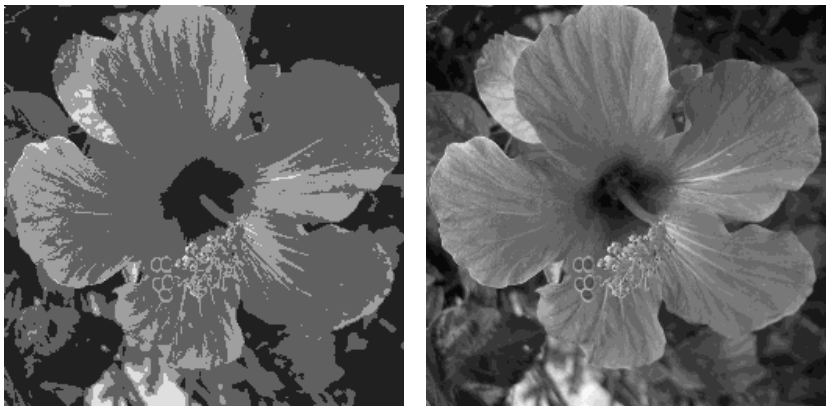
Une ligne/colonne sur 4 Une ligne/colonne sur 8 Une ligne/colonne sur 16

Figure 2.2: *Sous-échantillonnage d'une image*

– les valeurs dans $0, 1, \ldots, 63$ sont remplacées par la valeur 0 (noir),
– les valeurs dans $64, 1, \ldots, 127$ sont remplacées par la valeur 1 (gris clair),
– les valeurs dans $128, 1, \ldots, 191$ sont remplacées par la valeur 2 (gris foncé),
– les valeurs dans $192, \ldots, 255$ sont remplacées par la valeur 3 (blanc).

Une telle opération se nomme quantification. La **??**, au centre, suivante montre l'image résultante avec 4 niveaux de couleurs.

Nous avons déjà vu que l'on pouvait représenter toute valeur entre 0 et 255 à l'aide de 8 bits en utilisant l'écriture binaire. De façon similaire, on vérifie que toute valeur entre 0 et 3 peut se représenter à l'aide de 2 bits. On obtient ainsi une réduction d'un facteur 8/2=4 de la place mémoire nécessaire pour le stockage de l'image sur un disque dur. La figure **??** montre les résultats obtenus en utilisant de moins en moins de niveaux de gris.



4 niveaux de gris 16 niveaux de gris

Figure 2.3: *Quantification d'une image*

Tout comme pour la réduction du nombre de pixels, la réduction du nombre de niveaux de gris influe beaucoup sur la qualité de l'image. Afin de réduire au maximum la taille d'une image sans modifier sa qualité, on utilise des méthodes plus complexes de compression d'image.

La méthode la plus efficace s'appelle JPEG-2000. Elle utilise la théorie des ondelettes. Pour en savoir plus à ce sujet, vous pouvez consulter l'article d'Erwan Le Pennec sur le site web *Images des mathématiques*[2].

2.3 Enlever le bruit

2.3.1 Moyenne locale

Les images sont parfois de mauvaise qualité. Un exemple typique de défaut est le Â« bruit Â» qui apparait quand une photo est sous-exposée, c'est-à-dire qu'il n'y a pas assez de luminosité. Ce bruit se manifeste par de petites fluctuations aléatoires des niveaux de gris. La figure **??**, à gauche, montre une image bruitée.

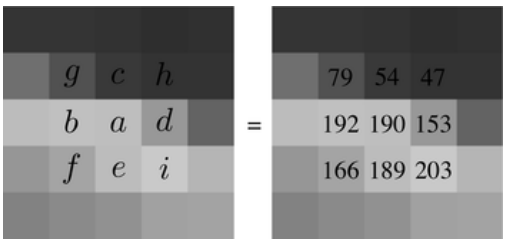


Figure 2.4: *Voisinage de pixels.*

Afin d'enlever le bruit dans les images, il convient de faire subir une modification aux valeurs de pixels. L'opération la plus simple consiste à remplacer la valeur a de chaque pixel par la moyenne de a et des 8 valeurs b, c, d, e, f, g, h, i des 8 pixels qui entourent a . La figure **??** montre un exemple de voisinage de 9 pixels. On obtient ainsi une image modifiée en remplaçant a par

$$\frac{a+b+c+d+e+f+g+h+i}{9}$$

puisque l'on fait la moyenne de 9 valeurs. Dans notre exemple, cette moyenne vaut

$$\frac{190+192+79+54+47+153+203+189+166}{9} \approx 141.$$

En effectuant cette opération pour chaque pixel, on supprime une partie du bruit, car ce bruit est constitué de fluctuations aléatoires, qui sont diminuées par un calcul de moyennes. La figure **??**, en haut à gauche, montre l'effet d'un tel calcul. Tout le bruit n'a pas été enlevé par cette opération. Afin d'enlever plus de bruit, on peut moyenner plus de valeurs autour de chaque pixel. La figure **??** montre le résultat obtenu en moyennant de plus en plus de valeurs.

Le moyennage des pixels est très efficace pour enlever le bruit dans les images, malheureusement il détruit également une grande partie de l'information de l'image. On peut en effet s'apercevoir que les images obtenues par moyennage sont floues. Ceci est en particulier visible près des contours, qui ne sont pas nets.

[2] http://images.math.cnrs.fr/Compression-d-image.html

Moyenne sur 9 pixels | Moyenne sur 25 pixels | Moyenne sur 49 pixels

Figure 2.5: *Moyenne de plus en plus forte*

2.3.2 Médiane locale

Afin de réduire ce flou, on remplace la moyenne par la médiane. Dans l'exemple du voisinage de 9 pixels utilisé à la section précédente, les 9 valeurs classées sont :

$$47, 54, 79, 153, 166, 189, 190, 192, 203.$$

La médiane de ces neuf valeurs est 166. Afin d'enlever plus de bruit, il suffit de calculer la médiane sur un nombre plus grand de pixels voisins, comme montré à la figure **??**. On constate que cette méthode est plus performante que le calcul de moyennes, car les images résultantes sont moins floues. Cependant, tout comme avec le calcul de moyennes, si l'on prend des voisinages trop grands, on perd aussi de l'information de l'image, en particulier les bords des objets sont dégradés.

Médiane sur 9 pixels | Médiane sur 25 pixels | Médiane sur 49 pixels

Figure 2.6: *Filtrage médian de plus plus en plus fort.*

2.4 Détecter les bords des objets

Afin de localiser des objets dans les images, il est nécessaire de détecter les bords de ces objets. Ces bords correspondent à des zones de l'image où les valeurs des pixels changent rapidement. C'est le cas par exemple lorsque l'on passe de la coque du bateau (qui est sombre, donc avec des valeurs petites) à la mer (qui est claire, donc avec des valeurs grandes).

Afin de savoir si un pixel avec une valeur a est le long d'un bord d'un objet, on prend en compte les valeurs b, c, d, e de ses quatre voisins qui ont un côté commun avec lui (figure **??**). Ceci permet de détecter aussi précisément que possible les bords des objets.

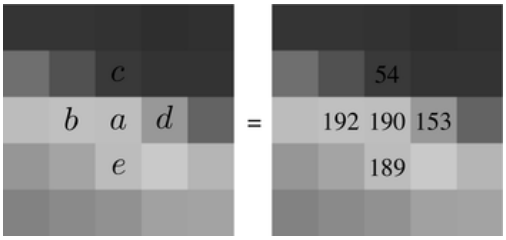


Figure 2.7: *Exemple d'un voisinage de 5 pixels.*

On calcule une valeur ℓ suivant la formule

$$\ell = \sqrt{(b-d)^2 + (c-e)^2}.$$

Dans notre exemple, on obtient donc

$$\ell = \sqrt{(192-153)^2 + (189-54)^2} = \sqrt{19746} \approx 141.$$

On peut remarquer que si $\ell = 0$, alors on a $b = c$ et $d = e$. Au contraire, si ℓ est grand, ceci signifie que les pixels voisins ont des valeurs très différentes, le pixel considéré est donc probablement sur le bord d'un objet.

La figure **??** montre une image dont la valeur des pixels est $\min(\ell, 255)$. Il est nécessaire de prendre le minium avec 255, car la valeur de ℓ peut dépasser la valeur maximale affichable (255, qui correspond au blanc). On a ainsi affiché ces valeurs avec du noir quand $\ell = 0$, du blanc quand ℓ est grand, et on a utilisé des niveaux de gris pour les valeurs intermédiaires. On peut voir que dans l'image de droite, les contours des objets ressortent en blanc, car ils correspondent aux grandes valeurs de ℓ.

2.5 Les images couleurs

2.5.1 Espace RVB

Une image couleur est en réalité composée de trois images indépendantes, afin de représenter le rouge, le vert, et le bleu. Chacune de ces trois images s'appelle un canal. Cette représentation en rouge, vert et bleu mime le fonctionnement du système visuel humain. La figure **??** montre les trois canaux constitutifs de l'image montrée sur la gauche de la figure **??**.

Chaque pixel de l'image couleur contient ainsi trois nombres (r, v, b), chacun étant un nombre entier entre 0 et 255. Si le pixel est égal à $(r, v, b) = (255, 0, 0)$, il ne contient que de l'information

Image d'origine Carte de contours ℓ

Figure 2.8: *Détection des bords.*

Image d'origine Luminance

Figure 2.9: *Image couleur.*

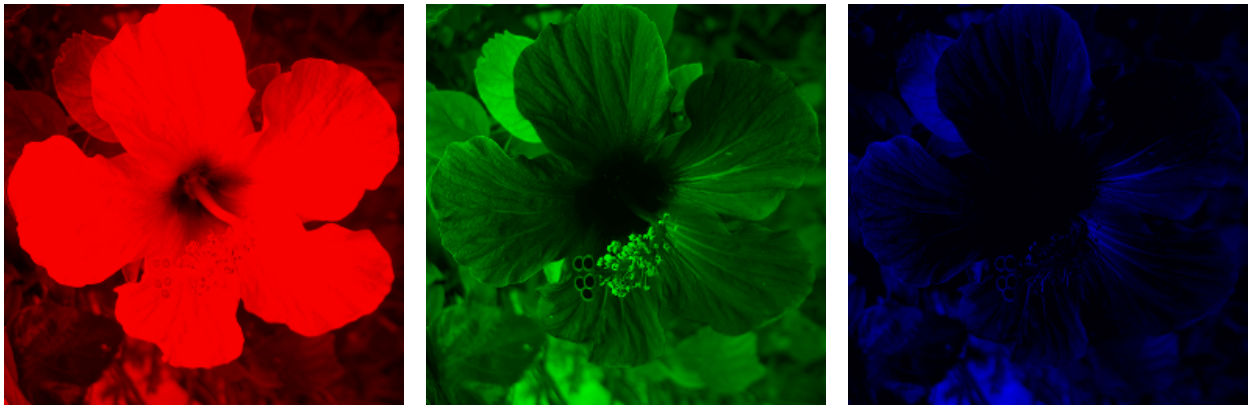
rouge, et est affiché comme du rouge. De façon similaire, les pixels valant $(0, 255, 0)$ et $(0, 0, 255)$ sont respectivement affichés vert et bleu.

On peut afficher à l'écran une image couleur à partir de ses trois canaux (r, v, b) en utilisant les règles de la synthèse additive des couleurs. Ces règles correspondent à la façon dont les rayons lumineux se combinent, d'où le qualificatif Â« additif Â». La figure **??**, gauche, montre les règles de composition cette synthèse additive des couleurs. Par exemple un pixel avec les valeurs $(r, v, b) = (255, 0, 255)$ est un mélange de rouge et de vert, il est donc affiché comme du jaune.

2.5.2 Espace CMJ

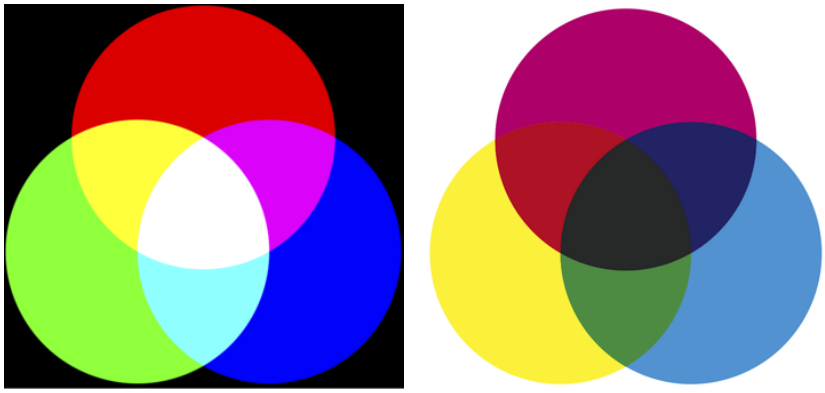
Une autre représentation courante pour les images couleurs utilise comme couleurs de base le cyan, le magenta et le jaune. On calcule les trois nombres (c, m, j) correspondant à chacun de ces trois canaux à partir des canaux rouge, vert et bleu (r, v, b) comme suit

$$c = 255 - r, \quad m = 255 - v, \quad j = 255 - b.$$



Canal rouge Canal vert Canal bleu

Figure 2.10: *Canaux couleurs*



Synthèse additive Synthèse soustractive

Figure 2.11: *Synthèse des couleurs*

Par exemple, un pixel de bleu pur $(r, v, b) = (0, 0, 255)$ va devenir $(c, m, j) = (255, 255, 0)$. La figure suivante montre les trois canaux (c, m, j) d'une image couleur.

Afin d'afficher une image couleur à l'écran à partir des trois canaux (c, m, j), on doit utiliser la synthèse soustractive des couleurs. La figure **??**, droite, montre les règles de composition cette synthèse soustractive. Elle correspondent en peinture à l'absorption de la lumière par les pigments colorés, d'où le qualificatif Â« soustractif Â». Le cyan, le magenta et le jaune sont appelés couleurs primaires.

On peut donc stocker sur un disque dur une image couleur en stockant les trois canaux, correspondant aux valeurs (r, g, b) ou (c, m, j). On peut modifier les images couleur tout comme les images en niveaux de gris. La façon la plus simple de procéder consiste à appliquer la modification à chacun des canaux.

Canal cyan | Canal magenta | Canal jaune

Figure 2.12: *Canaux CMJ*

2.6 Changer le contraste d'une image

2.6.1 Luminance

On peut calculer une image en niveaux de gris à partir d'une image couleur en moyennant les trois canaux. On calcule donc, pour chaque pixel, une valeur

$$a = \frac{r + v + b}{3}$$

qui s'appelle la luminance de la couleur. La figure **??** montre le passage d'une image couleur à une image de luminance en niveaux de gris.

2.6.2 Manipulations du contraste en niveaux de gris

Il est possible de faire subir différentes modifications à l'image afin de changer son contraste. On considère ici une image en niveaux de gris. Un manipulation simple consiste à remplacer chaque valeur a d'un pixel d'une image par $255 - a$ ce qui correspond à l'intensité de gris opposée. Le blanc devient noir et vice-et-versa, ce qui donne un effet similaire à celui des négatifs d'appareils photos argentiques, voir figure **??**, gauche.

On éclaircit ou assombrit l'image en utilisant une fonction croissante de $[0, 255]$ dans lui-même, que l'on applique aux valeurs a des pixels. On peut assombrir l'image en utilisant la fonction carré. Plus précisément, on définit la nouvelle valeur d'un pixel de l'image comme $a^2/255$ (voir figure **??** au centre). Le résultat n'étant en général pas un entier, on l'arrondit à l'entier le plus proche. De façon analogue, pour éclaircir l'image on remplace la valeur a de chaque pixel par l'arrondi entier de $\sqrt{255a}$. La figure **??**, à droite, montre l'éclaircissement obtenu. On pourra noter que ces deux opérations (éclaircissement par carré et assombrissement par racine carrée) sont inverses l'une de l'autre.

2.6.3 Manipulations du contraste en couleur

Afin de manipuler le contraste d'une image couleur, il est important de respecter autant que possible les teintes des couleurs. On souhaite donc ne manipuler que la composante de luminance

Négatif Carré Racine carrée

Figure 2.13: *Changement de contraste.*

$a = (r+v+b)/3$, en conservant constant le résidu $(r-a, v-a, b-a)$. On peut par exemple définir un changement de contraste en élevant la luminance a à la puissance $\gamma > 0$, afin d'obtenir

$$\tilde{a} = 255 \times \left(\frac{a}{255}\right)^{\gamma} = 255 \times \exp\left(\gamma \times \ln\left(\frac{a}{255}\right)\right),$$

(avec la convention $\tilde{a} = 0$ lorsque $a = 0$). On remarque que pour $\gamma = 1/2$ (respectivement $\gamma = 2$) on retrouve le changement de contraste par passage au carré (respectivement à la racine carrée) introduit à la section précédente. Et bien sûr, pour $\gamma = 1$, la luminance est inchangée.

Ce changement de contraste est ensuite répercuté sur l'image couleur en définissant trois canaux $(\tilde{r}, \tilde{v}, \tilde{b})$ d'une nouvelle image par

$$\begin{cases} \tilde{r} = \max(0, \min(255, r + \tilde{a} - a)), \\ \tilde{v} = \max(0, \min(255, v + \tilde{a} - a)), \\ \tilde{b} = \max(0, \min(255, b + \tilde{a} - a)). \end{cases}$$

Il est important de prendre le maximum avec 0 et le minimum avec 255 afin que le résultat reste dans l'intervalle $[0, 255]$, et soit affiché de manière correcte. La figure **??** montre le résultat obtenu pour différentes valeurs de γ. Pour $\gamma < 1$, l'image est éclaircie, alors que pour $\gamma > 1$, l'image est assombrie.

$\gamma = 0,5$ $\gamma = 0,75$ $\gamma = 1$ $\gamma = 1,5$ $\gamma = 2$ $\gamma = 3$

Figure 2.14: *Changement de contraste d'une image couleur.*

2.7 Images et matrices

2.7.1 Symétrie et rotation

Une image est un tableau de nombres, avec n lignes et p colonnes. Il est donc facile d'effectuer certaines transformations géométriques sur l'image. Les valeurs des pixels qui composent ce tableau (noté A) peuvent être représentées sous la forme $A = (a_{i,j})_{i,j}$ ou l'index i décrit l'ensemble des nombres $\{1, \ldots, n\}$ (les entiers entre 1 et n) et l'index j les nombres $\{1, \ldots, p\}$. On dit que $a_{i,j}$ est la valeur du pixel à la position (i, j).

Le tableau de pixels ainsi indexé se représente de la façon suivante

$$A = \begin{pmatrix} a_{1,1} & & & & a_{1,p} \\ & & \vdots & & \\ & & a_{i-1,j} & & \\ \ldots & a_{i,j-1} & a_{i,j} & a_{i,j+1} & \ldots \\ & & a_{i+1,j} & & \\ & & \vdots & & \\ a_{n,1} & & & & a_{n,p} \end{pmatrix},$$

Ceci correspond à la représentation de l'image sous forme d'une matrice. Transposer cette matrice correspond à effectuer une symétrie par rapport à la diagonale principale. On effectue cette transposition sur chacune des trois composantes couleurs (voir figure **??**, à gauche).

Matrice A | Matrice B (transposée) | Matrice C (rotation)

Figure 2.15: *Transposition et rotation.*

On peut également effectuer une rotation d'un quart de tour dans le sens des aiguilles d'une montre à l'image. Ceci est obtenu en définissant une matrice $C = (c_{i,j})_{j,i}$ de p lignes et n colonnes par $c_{j,i} = a_{n-i+1,j}$. La figure **??**, droite, montre l'action de cette rotation sur une image.

2.7.2 Fondu entres deux images

On souhaite effectuer une transition entre deux images A et B de même taille. On suppose donc que les deux images ont le même nombre n de lignes et le même nombre p de colonnes. On note $A = (a_{i,j})_{i,j}$ les pixels de l'image A et $B = (b_{i,j})_{i,j}$ les pixels de l'image B.

Pour une valeur t fixée entre 0 et 1, on définit l'image $C = (c_{i,j})_{i,j}$ comme

$$c_{i,j} = (1-t)a_{i,j} + tb_{i,j}.$$

Il s'agit de la formule d'une interpolation linéaire entre les deux images. Pour une image couleur, on applique cette formule à chacun des canaux R, V et B.

On peut constater que pour $t = 0$, l'image C est égale à l'image A. Pour $t = 1$, l'image C est égale à l'image B. Lorsque la valeur t progresse de 0 à 1, on obtient ainsi un effet de fondu, puisque l'image, qui au départ est proche de l'image A ressemble de plus en plus à l'image B. La figure **??** montre le résultat obtenu pour 6 valeurs de t réparties entre 0 et 1.

Image A, t=0 t=0,2 t=0,4 t=0,6 t=0,8 Image B, t=1

Figure 2.16: *Interpolation linéaire.*

Conclusion

Le traitement mathématique des images est un domaine très actif, où les avancées théoriques se concrétisent sous la forme d'algorithmes rapides de calcul. Ces algorithmes ont des applications importantes pour la manipulation des contenus numériques. Cet article n'a cependant fait qu'effleurer l'immense liste des traitements que l'on peut faire subir à une image. Les personnes intéressées pourront également consulter le site web *A Numerical Tour of Signal Processing*[3] pour de nombreux exemples de traitements d'images ainsi que des liens vers d'autres ressources disponibles en ligne.

Glossaire

- **Aléatoire** : valeur imprévisible souvent due au hazard, comme par exemple le bruit qui perturbe les images de mauvaises qualités.
- **Bit** : unité élementaire de stockage de l'information sous forme de 0 et de 1 dans un ordinateur.
- **Canal** : une des trois images élémentaires qui composent une image couleur.
- **Bords** : zone d'une image où les valeurs des pixels varient beaucoup, qui correspond aux contours des objets qui forment l'image.
- **Bruit** : petites perturbations qui dégradent la qualité d'une image.
- **Carré** : le carré b d'une valeur a est $a \times a$. Il est noté a^2.
- **Contraste** : quantité informelle qui indique la différence entre les zones claires et les zones sombres d'une image.

[3] http://www.numerical-tours.com/

– **Compression d'image** : méthode permettant de réduire la place mémoire nécessaire au stockage sur le disque dur d'une image.
– **Ecriture binaire** : écriture de valeurs numériques à l'aide uniquement de 0 et de 1.
– **Flou** : dégradation d'une image qui rend les contours des objets peu net, et donc difficile à localiser précisément.
– **Fondu** : interpolation linéaire entre deux images.
– **Image couleur** : ensemble de trois images en niveaux de gris, qui peut être affiché à l'écran en couleur.
– **Image numérique** : tableau de valeurs que l'on peut afficher à l'écran en assignant un niveaux de gris à chaque valeur.
– **Inverse** : opération ramenant une image dans son état d'origine.
– **JPEG-2000** : méthode récente de compression d'images qui utilise une transformation en ondelettes.
– **Luminance** : moyenne des différents canaux d'une image, qui indique la puissance lumineuse du pixel.
– **Matrice** : tableau de valeurs, représenté sous la forme $(a_{i,j})_{i,j}$.
– **Médiane** : valeur centrale lorsque l'on classe par ordre croissant un ensemble de valeurs.
– **Moyenne** : la moyenne d'un ensemble de valeurs est leur somme divisée par leur nombre.
– **Niveaux de gris** : nuances de gris utilisées pour afficher à l'écran une image numérique.
– **Nombres entiers** : nombres 0, 1, 2, 3, 4 ...
– **Octet** : ensemble de huit bits consécutifs.
– **Ondelettes** : transformation de l'image qui est utilisée par la méthode JPEG-2000 de compression d'images.
– **Ordre croissant** : classement d'un ensemble de valeurs de la plus petite à la plus grande.
– **Pixel** : une case dans un tableau de valeurs correspondant à une image numérique.
– **Quantification** : procédé consistant à réduire l'ensemble des valeurs possibles d'une image numérique.
– **Racine carrée** : la racine carrée b d'une valeur positive a est la valeur positive b vérifiant $a = b \times b$. On la note $\sqrt{a}$.
– **Résolution** : taille d'une image (nombre de pixels).
– **Sous-exposée** : photographie d'une scène trop sombre pour laquelle l'objectif photographique n'est pas resté assez longtemps ouvert.
– **Synthèse additive** : règle permettant de construire une couleur quelconque à partir des trois couleurs rouge, vert et bleu. C'est la règle qui régit le mélange des couleurs de faisceaux lumineux utilisés pour l'éclairage d'un mur blanc.
– **Synthèse soustractive** : règle permettant de construire une couleur quelconque à partir des trois couleurs cyan, magenta et jaune. C'est la règle qui régit le mélange des couleurs en peinture.

Chapter 3

Parcimonie, problèmes inverses et échantillonnage compressé

Les standards actuels pour compresser de la musique, de l'image ou de la vidéo (MP3, JPG ou MPEG) utilisent tous des méthodes issues de l'approximation non-linéaire. Ces méthodes calculent une approximation des données initiales à l'aide d'une combinaison linéaire d'un faible nombre de fonctions élémentaires (comme par exemple des sinusoïdes ou des ondelettes). Ces méthodes, initialement utilisées pour l'approximation, le débruitage ou la compression, ont été appliquées plus récemment à des problèmes plus difficiles, tels que l'augmentation de la résolution ou l'inversion d'opérateurs en imagerie médicale. Ces extensions nécessitent la résolution de problèmes d'optimisation de grande dimension, et sont le sujet d'une intense activité de recherche. Une des dernières avancées dans ce domaine, l'échantillonnage compressé, utilise la théorie des matrices aléatoires afin d'obtenir des garanties théoriques pour la performance de ces techniques. L'échantillonnage compressé permet d'envisager sous un angle nouveau la théorie de l'échantillonnage et de la compression de Claude Shannon. La compressibilité des données autorise en effet d'effectuer simultanément l'échantillonnage et la compression des données.

Cet article présente les concepts mathématiques clés qui ont permis l'évolution depuis l'échantillonnage classique de Shannon vers l'échantillonnage compressé. La notion de décomposition parcimonieuse, qui permet de formaliser l'idée de compressibilité de l'information, en est le fil directeur.

3.1 L'échantillonnage classique

Dans le monde numérique, la plupart des données (son, image, vidéo, etc.) sont discrétisées afin de les stocker, les transmettre et les modifier. A partir d'un signal *analogique*, qui est représenté par une fonction continue $s \mapsto \tilde{f}(s)$, l'appareil de mesure calcule un ensemble de Q valeurs *discrétisées* $f = (f_q)_{q=1}^{Q} \in \mathbb{R}^Q$. Ainsi, Q est le nombre d'échantillons temporels pour un morceau audio ou bien le nombre de pixels pour une image. La figure **??** montre des exemples de données discrétisées. Dans le cas d'une image, $\tilde{f}(s)$ représente la quantité de lumière arrivant en un point $s \in \mathbb{R}^2$ du plan focal de l'appareil photo, et $f_q = \int_{c_q} \tilde{f}(s)\mathrm{d}s$ est la quantité de lumière totale illuminant la surface c_q d'un capteur CCD indexé par q. Pour simplifier, nous faisons ici l'hypothèse de données scalaires (par exemple un son mono, une image ou une vidéo en niveaux de gris), mais les techniques décrites ici peuvent s'étendre au cas de données vectorielles (son stéréo, image couleur).

C'est la théorie élaborée par Claude Shannon [**?**] qui a posé les fondations de l'échantillonnage

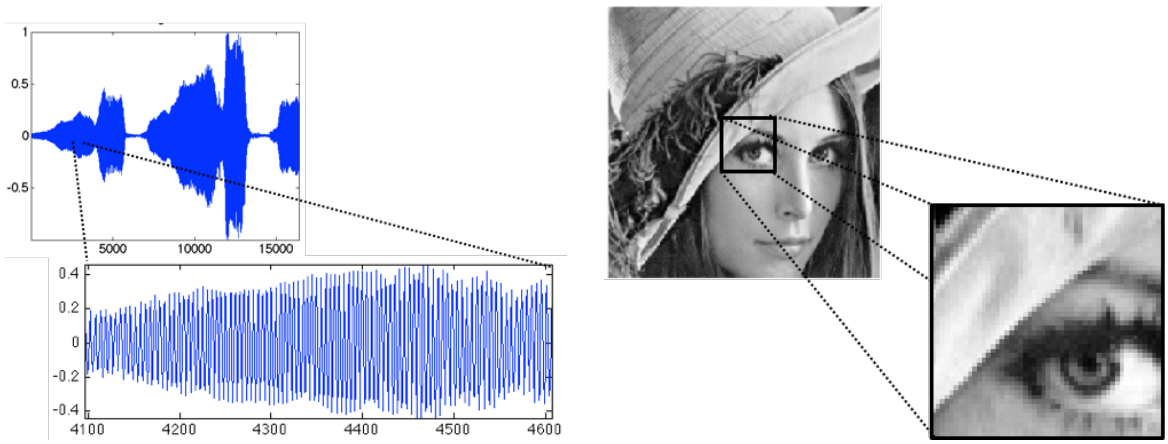


Figure 3.1: Exemples d'un signal sonore (données 1D) et d'une image (données 2D) discrétisés.

(l'utilisation d'un vecteur discret f afin de représenter fidèlement une fonction continue $\tilde{f}$) mais également celles de la compression sans perte. Nous allons voir comment les recherches actuelles ont permis de bâtir sur ces fondations des méthodes de compression avec pertes (i.e. avec un légère dégradation de la qualité), ainsi que de revisiter l'échantillonnage classique pour donner naissance à l'échantillonnage compressé.

3.2 Approximation non-linéaire et compression

Approximation non-linéaire. La dimension Q de ces données est en général très grande (de l'ordre du million pour une image, du milliard pour une vidéo) et il est nécessaire de calculer une représentation plus économe afin de pouvoir stocker f ou bien le transmettre sur un réseau. Toutes les méthodes de compression avec perte modernes (MP3, JPEG, MPEG, etc.) utilisent pour ce faire des décompositions parcimonieuses (c'est-à-dire composée de peu de coefficients non-nuls) dans un dictionnaire $\Psi = (\psi_n)_{n=1}^N$ composé d'atomes élémentaires $\psi_n \in \mathbb{R}^Q$. On recherche ainsi à approcher f à l'aide d'une combinaison linéaire

$$f \approx \Psi x \stackrel{\text{def.}}{=} \sum_{n=1}^{N} x_n \psi_n \in \mathbb{R}^Q$$

où les $x = (x_n)_{n=1}^N \in \mathbb{R}^N$ sont les coefficients que l'on va stocker où transmettre. Afin que cette représentation soit économe, et que le stockage prenne peu de place, il est nécessaire qu'un maximum de coefficients x_n soient nuls, de sorte que l'on n'ait à stocker que les coefficients non nuls. Etant donné un budget $M > 0$ de coefficients non-nuls, on cherche la meilleure combinaison possible afin d'approcher en norme ℓ^2 les données de départ. On cherche ainsi à résoudre le problème d'optimisation

$$x^\star \in \operatorname*{argmin}_{x \in \mathbb{R}^N} \{\|f - \Psi x\|_2 \; ; \; \|x\|_0 \leqslant M\} \quad \text{où} \quad \|f\|_2^2 \stackrel{\text{def.}}{=} \sum_{q=1}^{Q} |f_q|^2. \tag{3.1}$$

Ici, on a noté $\|x\|_0 \stackrel{\text{def.}}{=} \sharp\{n \; ; \; x_n \neq 0\}$ le nombre de coefficients non-nuls de x, qui est une mesure de comptage que l'on appelle souvent par abus de langage la Â« pseudo-norme Â» ℓ^0 (qui n'est pas

f $\quad$ $\Psi x^\star, M = N/4$ $\quad$ $\Psi x^\star, M = N/8$ $\quad$ $\Psi x^\star, M = N/16$

Figure 3.2: Exemples d'approximation $f \approx \Psi x^\star$ avec $M = \|x^\star\|_0$ qui varie, pour une image $f \in \mathbb{R}^N$ de $N = 256^2$ pixels.

une norme !). Cet abus de langage sera expliqué à la section **??**, voir en particulier la figure **??**.

Le problème (**??**) est en général impossible à résoudre : c'est un problème de nature combinatoire, qui, sans hypothèse supplémentaire sur Ψ, nécessite l'exploration de toutes les combinaisons de M coefficients non-nuls. Il a été prouvé que ce problème est en effet NP-difficile [**?**].

Approximation dans une base orthonormale. Il y a cependant un cas de figure simple, qui est très utile pour la compression : c'est le cas où Ψ est une base orthonormée de $\mathbb{R}^Q$, c'est à dire que $Q = N$ et

$$\langle \psi_n, \psi_{n'} \rangle = \begin{cases} 1 & \text{si} \quad n = n', \\ 0 & \text{sinon.} \end{cases} \qquad \text{où} \quad \langle f, g \rangle \stackrel{\text{def.}}{=} \sum_{q=1}^{Q} f_q g_q.$$

Ce cas est celui que l'on rencontre le plus souvent pour la compression de données, et on peut citer par exemple les bases orthogonales de Fourier discrètes, de cosinus locaux (utilisés pour MP3, JPG et MPG) et d'ondelettes (utilisées pour JPEG2000), voir le livre [**?**]. Dans ce cas, on a l'identité de Parseval qui correspond à la décomposition de f dans une base orthonormée

$$f = \sum_{n=1}^{N} \langle f, \psi_n \rangle \psi_n \quad \text{et} \quad \|f - \Psi x\|_2^2 = \sum_{n=1}^{N} |\langle f, \psi_n \rangle - x_n|^2. \tag{3.2}$$

Ces formules montrent que la solution de (**??**) se calcule très simplement. En effet, pour minimiser $\|f - \Psi x\|_2$, pour chaque x_n non-nul, il convient de choisir $x_n = \langle f, \psi_n \rangle$. Et comme on se fixe un budget maximum de M coefficients non nuls, il faut choisir les M plus grands coefficients $|\langle f, \psi_n \rangle|$ dans la formule (**??**). Mathématiquement, si on note $|\langle f, \psi_{n_1} \rangle| \geqslant |\langle f, \psi_{n_2} \rangle| \geqslant \ldots$ un classement des coefficients par ordre décroissant, alors une solution $x^\star$ de (**??**) est donnée par

$$x_n^\star = \begin{cases} \langle f, \psi_n \rangle & \text{si} \quad n \in \{n_1, \ldots, n_M\}, \\ 0 & \text{sinon.} \end{cases} \tag{3.3}$$

La figure **??** montre des approximations $f \approx \Psi x^\star$ ainsi calculées, avec un nombre $M = \|x^\star\|_0$ variable de coefficients. Ces approximations sont réalisées à l'aide d'une base orthogonale d'ondelettes Ψ, dite base de Daubechies 4, qui sont semblables aux fonctions utilisées dans le standard de compression d'image JPEG2000, et sont populaires car il existe un algorithme rapide pour calculer les

produits scalaires $(\langle f, \psi_n \rangle)_n$ en un temps de calcul proportionnel à Q (voir le livre [**?** , Chap. 7] pour une description complète de la théorie et la pratique numérique des ondelettes). On peut voir que la qualité de l'image reconstruite $\Psi x^\star$ se dégrade lorsque M diminue, mais on peut quand même réduire considérablement la quantité d'information à stocker (le taux de compression M/Q est petit), tout en gardant une qualité visuelle acceptable. Cette observation fondamentale correspond au fait (observé en pratique) que les images usuelles sont très bien approchées par une combinaison linéaire Â« parcimonieuse Â» de la forme $\Psi x^\star$ avec $\|x^\star\|_0 \leqslant M$. Il est important de remarquer que, bien que le calcul de $\Psi x^\star$ à partir $x^\star$ est une formule *linéaire*, le calcul de $x^\star$ à partir de f est *non-linéaire*, comme on peut le voir dans la formule (**??**). Le passage de f à son approximation $\Psi x^\star$ est appelé une approximation non-linéaire. La justification théorique de cette observation est l'objet d'étude de la théorie de l'approximation non-linéaire, qui cherche à prouver que $\|f - \Psi x^\star\|$ décroit rapidement lorsque M augmente sous certaines hypothèses de régularité sur f, par exemple si on suppose que l'image est lisse par morceaux, voir [**?** , Chap. 9].

Afin d'obtenir un réel algorithme de compression, il convient ensuite d'utiliser une technique permettant de convertir les M coefficients $(x_{n_1}, \ldots, x_{n_2})$ en écriture binaire et également de stocker les indices non-nuls $(n_1, \ldots, n_M)$. Ceci se fait simplement à l'aide de techniques issues de la théorie de l'information, en particulier les méthodes de codage entropique, voir [**?** , Chap. 10].

3.3 Problèmes inverses et parcimonie

Problèmes inverses. Avant de pouvoir stocker des données f, il est la plupart du temps nécessaire d'effectuer une étape préliminaire de restauration, qui consiste à améliorer la qualité des données à partir d'observations de basse qualité, c'est-à-dire de basse résolution, possiblement floues, entâchées d'erreurs et bruitées. Afin de prendre en compte toute la chaîne de formation des données, on modélise mathématiquement le processus d'acquisition sous la forme

$$y = \Phi f + w \in \mathbb{R}^P \tag{3.4}$$

où $y \in \mathbb{R}^P$ sont les P observations mesurées par l'appareil, $w \in \mathbb{R}^P$ est un bruit de mesure (inconnu), $f \in \mathbb{R}^Q$ est l'image (inconnue) que l'on souhaite récupérer, et $\Phi : \mathbb{R}^Q \to \mathbb{R}^P$ est un opérateur modélisant l'appareil d'acquisition, et que l'on suppose *linéaire*. Ceci signifie que l'on peut considérer Φ comme étant une (gigantesque) matrice $\Phi \in \mathbb{R}^{P \times Q}$. Il est important de noter que la plupart du temps, on ne stocke jamais explicitement cette matrice Φ, elle est manipulée de façon implicite à l'aide d'opérations rapides (convolution, masquage, etc.).

Ce modèle, qui peut paraître assez restrictif (en particulier l'hypothèse de linéarité) permet de modéliser une quantité surprenante de situations que l'on rencontre en pratique. On peut par exemple citer :

- le débruitage : $\Phi = \mathrm{Id}_{\mathbb{R}^Q}$, $P = Q$ et on est dans la situation (la plus simple) dans laquelle on ne cherche qu'à enlever le bruit w ;
- la déconvolution (voir figure (**??**), milieu) : $\Phi f = \varphi \star f$ est une convolution par un filtre φ modélisant par exemple le flou d'un appareil photo (soit un flou de bougé, soit un flou dû à la mise au point) ;
- les données manquantes (voir figure (**??**), droite) : $\Phi = \mathrm{diag}(\mu_q)_{q=1}^Q$ est un opérateur de masquage diagonal, tel que $\mu_q = 1$ si la données indexée par q (par exemple un pixel) est observée, et $\mu_q = 0$ si la donnée est manquante ;

Image originale f Φf (flou) Φf (masquage)

Figure 3.3: Observations (sans bruit, $w = 0$) $y = \Phi f$ dans le cas de la convolution ($\Phi f = \varphi \star f$ est une convolution par un filtre passe-bas φ) et des données manquantes ($\Phi = \mathrm{diag}(\mu_q)_{q=1}^{Q}$ est un opérateur de masquage).

- l'imagerie tomographique : Φ est un opérateur linéaire plus complexe, calculant des intégrales le long de lignes droites (la transformée de Radon), voir [**?** , Sect. 2.4].

Il existe quantité d'autres exemples (en imagerie médicale, sismique, astrophysique, etc.), et à chaque fois, calculer une bonne approximation de f à partir de y est très difficile. En effet, à l'exception du cas Â« facile Â» du débruitage (i.e. $\Phi = \mathrm{Id}_{\mathbb{R}^Q}$), on ne peut pas utiliser la formule $\Phi^{-1}y = f + \Phi^{-1}w$, soit parce que Φ n'est pas inversible (par exemple pour les données manquantes), soit parce que Φ a des valeurs propres très petites (pour la déconvolution ou la tomographie), de sorte que $\Phi^{-1}w$ va être très grand, et donc $\Phi^{-1}y$ est une approximation très mauvaise de f.

Régularisation parcimonieuse. Pour remédier à ce problème, il faut remplacer Φ^{-1} par une Â« inverse Â» approchée qui prend en compte des hypothèses supplémentaires sur le signal f que l'on cherche. Les méthodes récentes, qui donnent les meilleurs résultats sur des données complexes, utilisent une inverse approchée qui est non-linéaire. Ceci peut sembler contradictoire car Φ est linéaire, mais l'utilisation de méthodes non-linéaires est cruciale pour tirer parti d'hypothèses réalistes sur les données complexes telles que des images. En s'inspirant des techniques d'approximation et de compression discutées dans la section précédente, les méthodes actuelles cherchent à exploiter le fait que l'on peut bien approcher f à l'aide d'une approximation parcimonieuse Ψx avec $\|x\|_0 \leqslant M$. Etant donné un paramètre $M > 0$, on va chercher à approcher f par $f^\star = \Psi x^\star$ où $x^\star$ est une solution de

$$x^\star \in \operatorname*{argmin}_{x \in \mathbb{R}^N} \{\|y - \Phi\Psi x\|_2 \; ; \; \|x\|_0 \leqslant M\} \tag{3.5}$$

On voit que (**??**) est quasi-identique à (**??**), sauf que l'on a remplacé $f \in \mathbb{R}^Q$ (que l'on ne connaît pas) par $y \in \mathbb{R}^P$, et que l'on a remplacé la matrice $\Psi \in \mathbb{R}^{Q \times N}$ par le produit matriciel $\Phi\Psi \in \mathbb{R}^{P \times N}$. Dans le cas particulier du débruitage, $\Phi = \mathrm{Id}_{\mathbb{R}^Q}$, les problèmes (**??**) et (**??**) sont équivalents et ont la même solution, de sorte que l'approximation non-linéaire permet de résoudre le problème de débruitage.

Dans le cas d'un opérateur Φ quelconque, le problème (**??**) est cependant un problème d'optimisation extrêmement difficile à résoudre. En effet, même si Ψ est une base orthonormée, en général (sauf dans le cas du débruitage $\Phi = \mathrm{Id}_{\mathbb{R}^Q}$), la matrice $\Phi\Psi$ n'est pas orthogonale, de sorte que la formule (**??**) n'est pas applicable, et (**??**) est un problème de recherche combinatoire NP-difficile.

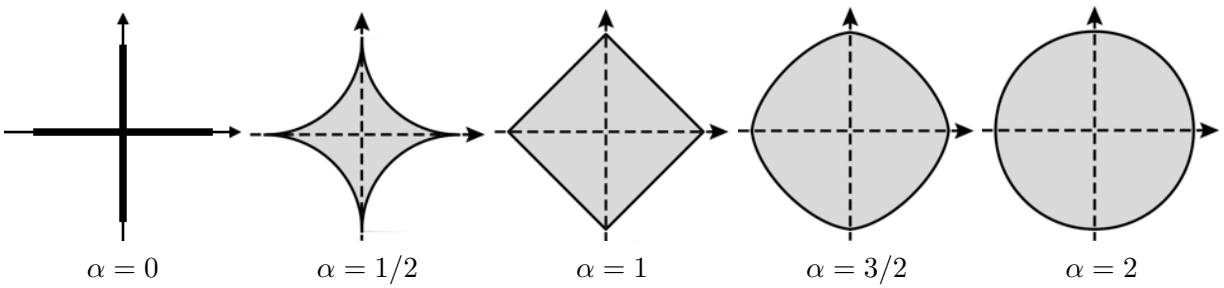


Figure 3.4: Boules B_α pour différentes valeurs de α.

Régularisation ℓ^1. L'approximation des solutions du problème (**??**) à l'aide de méthodes efficaces est un des sujets de recherche les plus actifs en traitement de données (et plus généralement en mathématiques appliquées, imagerie, statistique et apprentissage) de ces vingt dernières années. Il existe de nombreuses méthodes, parmi lesquelles les algorithmes gloutons (voir par exemple [**?**]) et les méthodes par relaxation convexe. Nous allons nous attarder principalement sur cette deuxième classe de méthodes. Une façon (heuristique) d'introduire ces techniques consiste à remplacer $\|\cdot\|_0$ dans le problème (**??**) par la fonction $\|\cdot\|_\alpha^\alpha$, qui est définie, pour $\alpha > 0$, par

$$\|x\|_\alpha^\alpha \stackrel{\text{def.}}{=} \sum_{n=1}^{N} |x_n|^\alpha.$$

La figure **??** montre dans le cas (irréaliste, mais bien pratique pour faire un dessin) de $N = 2$ coefficients, les boules unités $B_\alpha \stackrel{\text{def.}}{=} \{x \,;\, \|x\|_\alpha \leqslant 1\}$ associées à ces fonctionnelles $\|\cdot\|_\alpha$. On peut ainsi voir que B_α Â« tend Â» vers la Â« boule Â» unité associée à la mesure de comptage $\|\cdot\|_0$ à mesure que α tend vers 0, c'est-à-dire que

$$B_\alpha \stackrel{\alpha\to 0}{\longrightarrow} B_0 \stackrel{\text{def.}}{=} \left\{x \in [-1,1]^N \,;\, \|x\|_0 \leqslant 1\right\},$$

la convergence de ces ensembles (que l'on visualise bien sur la figure) étant au sens par exemple de la distance de Hausdorff. La boule limite B_0 est constituée de vecteurs extrêmement parcimonieux, puisqu'ils sont composés d'une seule composante non-nulle.

On est alors amené à prendre en compte deux éléments contradictoires pour choisir une valeur de α :

- Afin d'avoir une fonctionnelle privilégiant au maximum les vecteurs parcimonieux, on souhaite utiliser une valeur de α la plus faible possible pour remplacer $\|\cdot\|_0$ par $\|\cdot\|_\alpha$.
- Afin de pouvoir calculer la solution de (**??**) avec $\|\cdot\|_\alpha$ à la place de $\|\cdot\|_0$, il est important que la fonctionnelle $\|\cdot\|_\alpha$ soit *convexe*. La convexité est en effet essentielle afin d'obtenir un problème qui ne soit pas NP-difficile et pouvoir bénéficier d'algorithmes rapides de calcul. Ces algorithmes trouvent une solution exacte $x^\star$ en temps polynomial ou bien convergent rapidement vers cette solution.

La contrainte de convexité de $\|\cdot\|_\alpha$ impose que l'ensemble B_α soit convexe, ce qui, de façon équivalente, signifie que $\|\cdot\|_\alpha$ doit être une *norme*. Ceci impose que $\alpha \geqslant 1$. La prise en compte de ces deux contraintes mène ainsi naturellement au choix Â« optimal Â» $\alpha = 1$, de sorte que l'on va considérer

f original Observations y Reconstruction $f^\star$

Figure 3.5: Exemples de reconstruction avec données manquantes, $\Phi = \mathrm{diag}(\mu_q)_{q=1}^{Q}$ avec $\mu_q \in \{0,1\}$ et un nombre de données observées $\sharp\{q\ ;\ \mu_q = 1\}/Q = 10\%$.

le problème d'optimisation convexe (c'est-à-dire que l'on cherche à minimiser une fonction convexe sur un ensemble convexe)

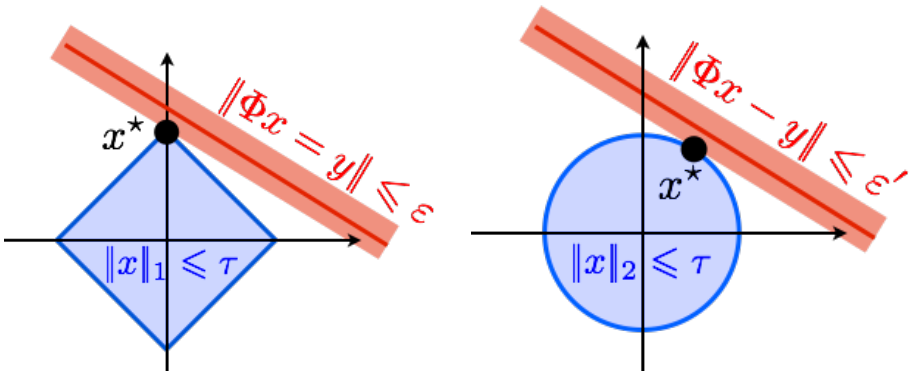
$$x^\star \in \operatorname*{argmin}_{x \in \mathbb{R}^N} \left\{ \|y - \Phi\Psi x\|_2\ ;\ \|x\|_1 = \sum_{n=1}^{N} |x_n| \leqslant \tau \right\}, \tag{3.6}$$

de sorte que l'image calculée comme solution est $f^\star = \Psi x^\star$. On peut noter que l'on a utilisé ici un paramètre $\tau > 0$ qui joue un rôle similaire au paramètre M qui apparaît dans (**??**). La question du choix de ce paramètre τ est cruciale. Si le bruit w est petit, alors on souhaite que $\Phi f^\star = \Phi\Psi x^\star$ soit proche de y, et donc on va choisir τ grand. Au contraire, si le bruit w est important, afin d'obtenir un effet de débruitage plus important, on va réduire la valeur de τ. Le choix d'un τ Â« optimal Â» est un problème de recherche difficile, et il n'existe pas de réponse Â« universelle Â», les stratégies existantes dépendent fortement de l'opérateur Φ ainsi que de la famille d'atomes Ψ.

Le problème (**??**) a initialement été proposé par des ingénieurs dans les domaines de l'imagerie sismique (voir par exemple [**?**]), et il a été introduit conjointement en traitement du signal sous le nom Â« basis pursuit Â» [**?**] et en statistique sous le nom Â« Lasso Â» [**?**].

Le problème (**??**), bien que convexe, reste un problème difficile à résoudre à cause de la non-différentiabilité de la norme $\|\cdot\|_1$ et de la grande taille des données (N est très grand). C'est le prix à payer pour obtenir des résultats de bonne qualité. Comme nous allons l'expliquer dans le paragraphe qui suit, c'est en effet la non-différentiabilité de $\|\cdot\|_1$ qui permet d'obtenir de la parcimonie. Le développement d'algorithmes efficaces pour résoudre (**??**) est un domaine de recherche très actif, et nous renvoyons à [**?** , section 6] pour un tour d'horizon de ces méthodes. La figure **??** montre un exemple d'interpolation de données manquantes réalisée en résolvant (**??**) dans une famille Ψ d'ondelettes invariantes par translation.

De l'intuition à l'analyse théorique des performances. La figure **??** montre intuitivement pourquoi la solution $x^\star$ calculée en remplaçant $\|\cdot\|_0$ par $\|\cdot\|_\alpha$ dans (**??**) est meilleure (au sens qu'elle est plus parcimonieuse) si on choisit $\alpha = 1$ (c'est-à-dire si on résout (**??**)) que si on choisit $\alpha = 2$ (une conclusion similaire est obtenue pour d'autres valeurs de $\alpha > 1$). La figure est fait dans le cas (très simple) de $N = 2$ coefficients et $P = 1$ observations. Le point crucial, qui rend la solution de (**??**) parcimonieuse, est que la boule B_1 associée à la norme ℓ^1 est Â« pointue Â»



Minimisation ℓ^1 Minimisation ℓ^2

Figure 3.6: Comparaison de la minimisation avec des contraintes de type $\|x\|_\alpha \leqslant \tau$ pour $\alpha \in \{1, 2\}$. Une solution $x^\star$ est obtenue lorsque l'on trouve un tube $\{x \; ; \; \|\Phi x - y\| \leqslant \varepsilon\}$ assez grand (i.e. en faisant croitre progressivement ε) tel qu'il soit tangent en $x^\star$ à la boule $\{x \; ; \; \|x\|_\alpha \leqslant \tau\}$.

de sorte que la solution $x^\star$ est située le long des axes. Ceci n'est pas le cas pour la boule B_2 associée à la norme ℓ^2, qui donne une solution $x^\star$ qui n'est pas le long des axes, et n'est donc pas parcimonieuse. Ce phénomène, déjà visible en dimension 2, est en fait accentué lorsque la dimension augmente, de sorte que l'approximation obtenue en remplaçant $\|\cdot\|_0$ par $\|\cdot\|_1$ devient meilleure en grande dimension. Ce phénomène est appelé par David Donoho la Â« bénédiction de la grand dimension Â» [**?**] : bien que les données deviennent très coûteuses et complexes à traiter (la Â« malédiction de la dimension Â») on dispose de techniques efficaces pour les analyser si elles sont suffisamment parcimonieuses. Rendre cette intuition rigoureuse est cependant difficile, et c'est l'objet de recherches encore en cours pour des opérateurs Φ tels que des convolutions [**?** **?**]. L'analyse dans le cas des opérateurs que l'on rencontre par exemple en imagerie médicale est un problème mathématique ouvert.

3.4 L'échantillonnage compressé

Il existe une classe particulière d'opérateurs Φ pour laquelle il est possible d'analyser très précisément les performances obtenues lorsque l'on résout (**??**). Il s'agit du cas où Φ est tiré aléatoirement selon certaines distributions de matrices aléatoires. Utiliser des matrices aléatoires peut sembler étrange, car les opérateurs mentionnés plus haut (convolution, tomographie, etc.) ne le sont pas du tout. En fait, ce choix est motivé par une application concrète proposée conjointement par Candès, Tao et Romberg [**?**] ainsi que Donoho [**?**], et que l'on appelle communément Â« échantillonnage compressé Â» (Â« compressed sensing Â» en anglais).

Appareil photo Â« pixel unique Â». Afin de rendre l'explication plus parlante, nous allons aborder le prototype d'appareil photo Â« pixel unique Â» (Â« single pixel caméra Â» en anglais) développé à Rice University [**?**], et qui est illustré par la figure **??** (gauche). Il s'agit de développer une nouvelle classe d'appareils photos permettant de réaliser à la fois *l'échantillonnage* et la *compression* d'une image. Au lieu de d'abord échantillonner très finement (i.e. avec Q très grand) le signal analogique $\tilde{f}$ pour obtenir une image $f \in \mathbb{R}^Q$ puis de compresser énormément (i.e. avec M petit) en utilisant (**??**), on aimerait disposer directement d'une représentation économique $y \in \mathbb{R}^P$

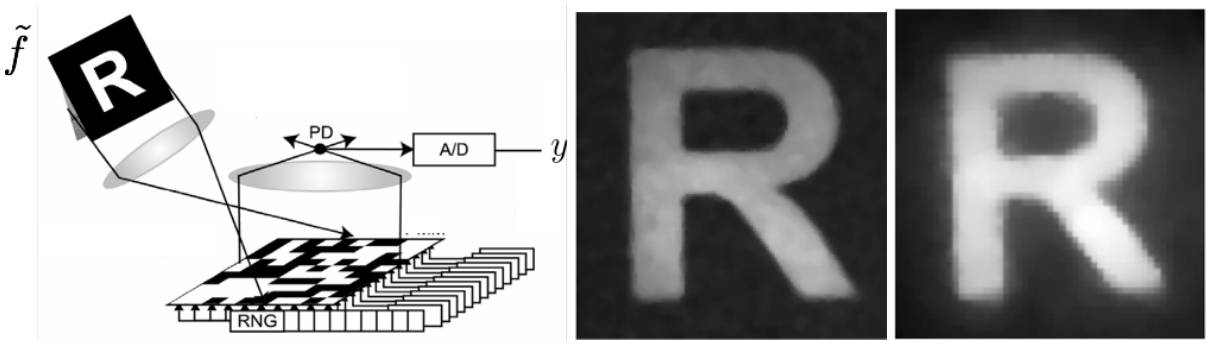


Schéma de l'appareil f $f^\star$, $P/Q = 6$

Figure 3.7: Gauche : schéma de la méthode d'acquisition par pixel unique. Centre : image $f \in \mathbb{R}^Q$ Â« idéale Â» observée dans le plan focal des micro-miroirs. Droite : image $f^\star = \Psi x^\star$ reconstruite à partir d'observation $y \in \mathbb{R}^P$ avec un facteur de compression $P/Q = 6$.

de l'image, avec un budget P aussi proche de M et tel que l'on soit capable de Â« décompresser Â» y pour obtenir une bonne approximation de l'image f.

L'appareil Â« pixel unique Â» permet de réaliser l'échantillonnage compressé d'une scène observée $\tilde{f}$ (la lettre Â« R Â» sur la Figure **??**), qui est une fonction continue indiquant la quantité de lumière $\tilde{f}(s)$ atteignant chaque point $s \in \mathbb{R}^2$ du plan focal de la camera. Pour ce faire, la lumière est focalisée contre un jeu de Q micro-miroirs tapissant le plan focal. Ces micro-miroirs ne sont pas des capteurs. Contrairement à l'échantillonnage classique (décrit à la section **??**), ils n'enregistrent aucune information, mais ils peuvent chacun être positionné pour refléter ou absorber la lumière. Pour réaliser l'enregistrement complet, on change très rapidement P fois les configurations des micro-miroirs. Pour $p = 1, \ldots, P$, on note ainsi $\Phi_{p,q} \in \{0, 1\}$ suivant que le micro-miroir à la position q a été mis en position absorbante (valeur 0) ou réfléchissante (valeur 1) à l'étape p de l'acquisition. La lumière totale réfléchie à l'étape p est ensuite accumulée en un capteur unique (d'où le nom de Â« pixel unique Â», en fait il s'agit plutôt d'un Â« capteur unique Â»), noté Â« PD Â» sur la figure, ce qui réalise une somme linéaire des intensités réfléchies pour obtenir la valeur $y_p \in \mathbb{R}$ enregistrée. Au final, si l'on note (comme à la section **??**) $f_q = \int_{c_q} \tilde{f}(s)\mathrm{d}s$ l'intensité de lumière qui arrive sur la surface c_q du miroir indexé par q, l'équation qui relie l'image discrète $f \in \mathbb{R}^Q$ Â« vue par les miroirs Â» aux P mesures $y \in \mathbb{R}^P$ est

$$\forall\, p = 1, \ldots, P, \quad y_p = \sum_q \Phi_{p,n} \int_{c_n} \tilde{f}(s)\mathrm{d}s = (\Phi f)_p,$$

ce qui correspond exactement à (**??**). Il est important de noter que les miroirs n'enregistrent rien, donc en particulier, l'image discrète f n'est jamais calculée ou enregistrée, l'appareil calculant directement la représentation compressée y depuis le signal analogique $\tilde{f}$. Le terme w modélise ici les imperfections d'acquisition (bruit de mesure). L'échantillonnage compressé correspond donc au passage de la scène observée $\tilde{f}$ au vecteur directement compressé y. La Â« décompression Â» correspond à la résolution d'un problème inverse, qui a pour but de retrouver une bonne approximation de f (l'image discrète Â« idéale Â» telle que vue par les micro-miroirs) à partir de y.

Garanties théoriques. Une particularité importante de ce problème inverse est que l'on peut choisir comme on le souhaite les configurations des micro-miroirs, ce qui revient à dire que l'on

peut choisir librement la matrice $\Phi \in \{0,1\}^{P \times Q}$. La question est donc de faire le meilleur choix, de sorte que l'on puisse résoudre efficacement le problème inverse. Si l'on fait l'hypothèse que le signal f à reconstruire est compressible dans une base orthonormée Ψ (c'est-à-dire que $f \approx \Psi x_0$ avec $M \stackrel{\text{def.}}{=} \|x_0\|_0$ petit), alors de nombreux travaux, à commencer par [**?** **?**], ont montré que la méthode (**??**) était efficace si l'on choisit Φ comme une réalisation de certaines matrices aléatoires. Pour le cas de l'appareil photo à pixel unique, on peut ainsi tirer chaque $\Phi_{p,n}$ aléatoirement avec une probabilité de 1/2 pour les valeurs 0 et 1. En pratique, on utilise un générateur pseudo-aléatoire, de sorte qu'à la fois la personne qui compresse les données et la personne qui va les décompresser connaîssent parfaitement la matrice Φ (car elles peuvent se communiquer la graine du générateur). La figure **??** (droite) montre un exemple de reconstruction obtenue pour le cas de l'appareil à pixel unique avec un tel choix aléatoire de matrice Φ, avec pour dictionnaire Ψ une famille d'ondelettes invariantes par translation (voir [**?** , Sect. 5.2] pour une description de cette famille).

Il a ainsi été montré par [**?** **?**] qu'il existe une constante C telle que si l'on note $f = \Psi x_0$ où x_0 sont les coefficients de l'image à retrouver, où Ψ est une base orthogonale (donc en particulier $Q = N$), et si le nombre P de mesures vérifie

$$\frac{P}{M} \geqslant C \log\left(\frac{N}{M}\right) \quad \text{où} \quad M \stackrel{\text{def.}}{=} \|x_0\|_0 \tag{3.7}$$

alors une solution $f^\star = \Psi x^\star$ calculée par (**??**) tend vers f lorsque le bruit w tend vers 0 et τ tend vers $+\infty$. Ce résultat est vrai Â« avec forte probabilité Â» sur le tirage aléatoire de la matrice Φ, c'est-à-dire une probabilité tendant rapidement vers 1 lorsque N augmente. En particulier, s'il n'y a pas de bruit, $w = 0$, en prenant $\tau \to +\infty$, la méthode permet de retrouver exactement f si P vérifie (**??**). Cette théorie permet aussi de prendre en compte des données Â« compressibles Â», c'est à dire si l'on suppose uniquement que f est proche de (mais pas nécessairement égal à) Ψx_0 avec $M \stackrel{\text{def.}}{=} \|x_0\|_0$ petit.

De façon intuitive, ce résultat théorique signifie que l'échantillonnage compressé arrive à faire quasiment Â« aussi bien Â» en calculant $\Psi x^\star$ à partir de y (en résolvant (**??**)) qu'une méthode de compression usuelle (MP3, JPEG, JPEG2000, MPEG, etc.) qui connaitrait exactement le signal f et calculerait la meilleure approximation Ψx_0 avec $M \stackrel{\text{def.}}{=} \|x_0\|_0$ coefficients (en résolvant (**??**) via la formule (**??**)). La signification précise du qualificatif Â« aussi bien Â» correspond au facteur multiplicatif $C \log(N/M)$, qui borne P/M. Ce facteur correspond au Â« surcoût Â» de la méthode d'échantillonnage compressé (qui calcule P mesures) par rapport à une méthode de compression usuelle (qui calcule M coefficients). Malgré ce surcoût, la méthode de l'échantillonnage compressé présente de nombreux avantages : gain de temps et d'énergie (on fait en même temps l'échantillonnage et la compression), codage Â« démocratique Â» (tous les coefficients y_n jouent le même rôle, et donc aucun n'a de rôle prépondérant, contrairement au codage des coefficients de x_0 qui ont une importance proportionnelle à leur amplitude), codage automatiquement crypté (si on ne connaît pas Φ, on ne peut pas retrouver f à partir de y). La valeur de la constante C dépend du sens que l'on donne au terme Â« avec forte probabilité Â». Si cette probabilité porte uniquement sur Φ, mais doit être vraie pour tous les x_0 (analyse au pire cas), alors elle est très grande (voir [**?**]). Si par contre on veut qu'elle porte à la fois sur Φ et sur x_0 (pour que le résultat théorique soit vrai pour presque tous les signaux) alors on peut montrer que par exemple, pour $N/P = 4$ (compression d'un facteur 4), on a $C \log(N/M) \sim 4$ (voir [**?**]), ce qui reste un surcoût conséquent, mais qui est acceptable pour certaines applications.

L'appareil photo Â« pixel unique Â» est une déclinaison particulière de la technique d'échantillonnage compressé. Les applications à la photographie sont limitées, car les capteurs CCD des appareils

photos sont performants et peu chers. L'échantillonnage compressé aura probablement un impact pour des applications où les mesures sont difficiles à acquérir ou coûtent chers. Une autre source d'applications potentielles est l'imagerie médicale, par exemple par résonance magnétique. Dans ces domaines, il est cependant impossible d'obtenir des matrices totalement aléatoires, de sorte que l'on ne peut pas appliquer directement la théorie de l'échantillonnage compressé. Des résultats encourageants sur ces applications ont cependant été obtenus, voir par exemple [**?** **?**].

3.5 Conclusion

Les avancées récentes de l'analyse de données ont permis d'étendre le champ d'application de la compression afin de traiter des problèmes inverses difficiles en imagerie, mais aussi dans d'autres domaines (sytème de recommandation, analyse de réseaux, etc.). Ces avancées ont été rendues possibles par l'utilisation d'un spectre très large de techniques en mathématiques appliquées, qui couvre à la fois l'analyse harmonique, l'approximation non-linéaire, l'optimisation non-lisse et les probabilités, mais également l'analyse fonctionnelle et les EDPs (qui n'ont pas été mentionnées dans cet article). Les méthodes parcimonieuses associées à la régularisation ℓ^1 ne sont pourtant que la partie émergée de l'iceberg, et des régularisations plus fines permettent d'obtenir de meilleurs résultats en prenant en compte les structures géométriques complexes des données. Pour plus de détails sur ces dernières avancées, nous recommandons la lecture de l'article [**?**], ainsi que la visite du site web Â« Numerical Tours of Signal Processing Â» [**?**], qui présente de nombreux codes informatiques pour réaliser les expériences numériques présentées ici, ainsi que de nombreuses autres.

Remerciements Je tiens à remercier Charles Dossal, Jalal Fadili, Samuel Vaiter, Stéphane Seuret et le relecteur anonyme pour leur aide précieuse.

Chapter 4

Le Transport Optimal et ses Applications

4.1 Le Transport Optimal de Monge

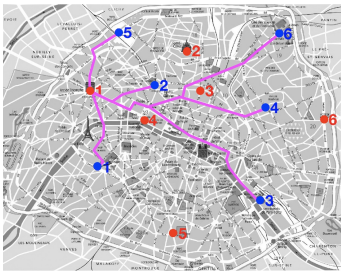
Gaspard Monge, en plus d'Ãªtre un grand mathÃ©maticien, a participÃ© activement Ã la rÃ©volution FranÃ§aise, et a crÃ©Ã© l'École Polytechnique ainsi que l'École Normale SupÃ©rieure. MotivÃ© par des applications militaires, il a formulÃ© en 1781 le problÃ¨me du transport optimal [**?**]. Il s'est posÃ© la question du calcul de la faÃ§on la plus Ã©conomique de transporter de la terre entre deux endroits pour faire des remblais. Dans son texte original, il a fait l'hypothÃ¨se que le coÃ»t du dÃ©placement d'une unitÃ© de masse est Ã©gal Ã la distance parcourue, mais on peut utiliser n'importe quel coÃ»t adaptÃ© au problÃ¨me Ã rÃ©soudre.

Le problÃ¨me de Monge

Pour illustrer le problÃ¨me et sa formulation mathÃ©matique, intÃ©ressons-nous Ã la faÃ§on optimale de distribuer les croissants depuis les boulangeries vers les cafÃ©s, le matin dans Paris. Pour simplifier, nous allons supposer qu'il y a uniquement six boulangeries et cafÃ©s, que l'on peut voir Ã la figure **??** (les boulangeries sont en rouge et les cafÃ©s en bleu). On suppose que chaque boulangerie produit le mÃªme nombre de croissants et que cafÃ© demande Ã©galement le mÃªme nombre de croissant. Le coÃ»t Ã minimiser est le temps total des trajets, et l'on note $C_{i,j}$ le temps entre la boulangerie $i \in \{1,\dots,6\}$ et le cafÃ© $j \in \{1,\dots,6\}$. Par exemple, on a $C_{3,4} = 10$, ce qui signifie qu'il y a dix minutes de trajet entre la boulangerie numÃ©ro 3 et le cafÃ© numÃ©ro 4.

Afin de satisfaire la contrainte d'approvisionnement (que l'on appelle aussi la conservation de la masse), il faut que chaque boulangerie soit connectÃ©e Ã un et un seul cafÃ©. Comme il y a le

c_{ij}	y_1	y_2	y_3	y_4	y_5	y_6
x_1	12	10	31	27	10	30
x_2	22	7	25	15	11	14
x_3	19	7	19	10	15	15
x_4	10	6	21	19	14	24
x_5	15	23	14	24	31	34
x_6	35	26	16	9	34	15



c_{ij}	y_1	y_2	y_3	y_4	y_5	y_6
x_1	12	10	31	27	10	30
x_2	22	7	25	15	11	14
x_3	19	7	19	10	15	15
x_4	10	6	21	19	14	24
x_5	15	23	14	24	31	34
x_6	35	26	16	9	34	15

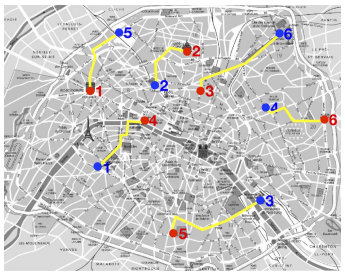
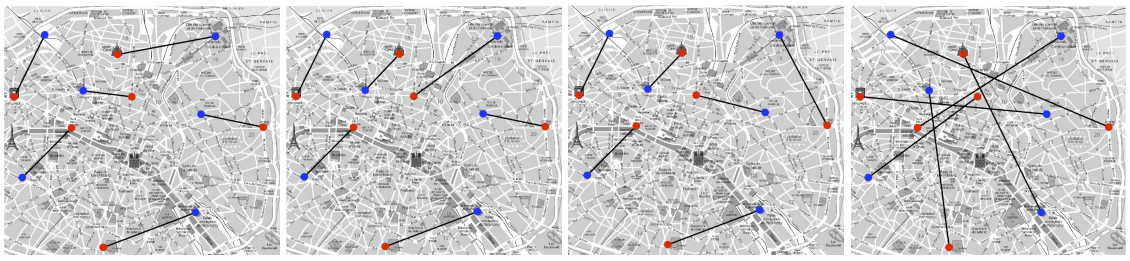


Figure 4.1: Matrice de coÃ»t et connexions associÃ©es. Gauche : une ligne de la matrice coÃ»t. Droite : un exemple particulier de permutation.



CoÃ»t=64 CoÃ»t=65 CoÃ»t=66 CoÃ»t=152

Figure 4.2: Exemples de permutations avec diffÃ©rent coÃ»ts.

mÃªme nombre de boulangeries que de cafÃ©s, ceci implique que chaque cafÃ© est Ã©galement connectÃ© Ã une et une seule boulangerie. On va noter

$$\sigma : i \in \{1, \dots, 6\} \longmapsto j \in \{1, \dots, 6\}$$

un tel choix de connexions. La figure **??** illustre au centre et Ã droite l'exemple

$$\sigma(1) = 5,\ \sigma(2) = 2,\ \sigma(3) = 6,\ \sigma(4) = 1,\ \sigma(5) = 3,\ \sigma(6) = 4. \tag{4.1}$$

La contrainte de conservation de masse signifie que σ est une bijection de l'ensemble $\{1, \dots, 6\}$ dans lui-mÃªme. On dit aussi que σ est une permutation.

Le coÃ»t de transport associÃ© Ã une telle bijection est la somme des coÃ»ts $C_{i,\sigma(i)}$ sÃ©lectionnÃ©s par la permutation σ, c'est-Ã -dire

$$\text{CoÃ»t}(\sigma) \overset{\text{def.}}{=} C_{1,\sigma(1)} + C_{2,\sigma(2)} + C_{3,\sigma(3)} + C_{4,\sigma(4)} + C_{5,\sigma(5)} + C_{6,\sigma(6)}. \tag{4.2}$$

Par exemple, pour la bijection (**??**) montrÃ©e Ã la figure **??**, on obtient comme coÃ»t

$$C_{1,5} + C_{2,2} + C_{3,6} + C_{4,1} + C_{5,3} + C_{6,4} = 10 + 7 + 15 + 10 + 14 + 9 = 65.$$

Le problÃ¨me de Monge consiste Ã chercher la permutation σ qui a le coÃ»t minimum, c'est-Ã -dire rÃ©soudre le problÃ¨me d'optimisation

$$\min_{\sigma \in \Sigma_6} \text{CoÃ»t}(\sigma), \tag{4.3}$$

oÃ¹ *l'onanot*Σ_6 l'ensemble des permutations de l'ensemble $\{1, \dots, 6\}$.

La figure **??** montre que la permutation (**??**) n'est pas la meilleure : il existe par exemple une autre permutation qui a un coÃ»t de 64. Mais est-ce la meilleure ? Il se trouve que oui, on peut en effet tester sur un ordinateur toutes les permutations de $\{1, \dots, 6\}$ et calculer leur coÃ»t. Combien y a-t-il de permutations au total ? Pour effectuer ce dÃ©nombrement, on voit qu'il y a six choix d'affectation possible de 1 Ã $\sigma(1) \in \{1, \dots, 6\}$, puis cinq choix possibles pour affecter 2 Ã $\sigma(2) \in \{1, \dots, 6\} - \{\sigma(1)\}$, et ainsi de suite. Le nombre total de possibilitÃ©s est donc $6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$ que l'on note 6!, Â« factorielle 6 Â». Si on considÃ¨re un nombre n de boulangeries, alors le nombre de permutations Ã tester pour trouver la meilleure est $n! = n \times (n-1) \times \ldots \times 2 \times 1$. Ce nombre croit extrÃªmement vite avec n, par exemple $70! \approx 1,198 \times 10^{100}$, Ã comparer avec les 10^{11} neurones dans le cerveau et les 10^{79} atomes dans

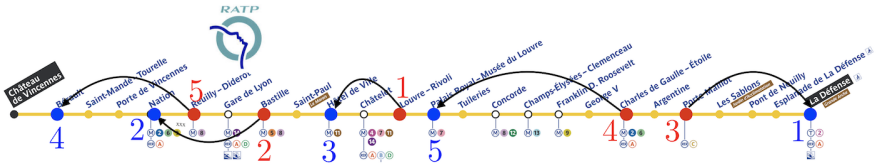


Figure 4.3: Le transport optimal en 1D le long d'une ligne de mÃ©tro. La bijection optimale est $\sigma : (1, 2, 3, 4, 5) \mapsto (3, 2, 1, 5, 4)$.

l'univers. Cette stratÃ©gie de recherche exhaustive n'est donc possible que pour de toute petites valeurs de n.

En 1D et 2D

La section **??** explique comment des avancÃ©es mathÃ©matiques ont permis de dÃ©velopper des techniques efficaces pour calculer un transport optimal σ mÃªme pour de grandes valeurs de n. Mais il aura fallu attendre prÃ¨s de 200 ans pour y arriver. Dans certains cas simples, on peut cependant calculer le transport optimal de faÃ§on simple. Le cas le plus Ã©lÃ©mentaire est lorsque les points Ã apparier sont le long d'un axe 1D, par exemple si les cafÃ©s et les boulangeries sont situÃ©s le long d'une ligne de mÃ©tro. Il faut Ã©galement que le coÃ»t $C_{i,j}$ soit la distance le long de cet axe (par exemple le temps de trajet en mÃ©tro entre les stations). Dans ce cas, il suffit de classer les indices i et j par ordre croissant (donc de gauche Ã droite le long de la ligne de mÃ©tro) et d'apparier le premier indice i au premier indice j ensemble, puis le deuxiÃ¨me indice, etc. Ce procÃ©dÃ© est illustrÃ© Ã la figure **??**. Le temps de calcul nÃ©cessaire pour calculer le transport optimal en mÃ©tro est donc le temps nÃ©cessaire pour classer les indices. L'algorithme le plus simple pour effectuer un classement est celui utilisÃ© habituellement pour trier un jeu de n cartes : il s'agit du tri par insertion, qui insÃ¨re itÃ©rativement chaque carte Ã sa place par rapport aux cartes dÃ©jÃ classÃ©es. Il effectue $n(n-1)/2$ comparaisons. Pour $n = 70$, ceci nÃ©cessite donc seulement 21415 operations, ce qui rend la mÃ©thode utilisable, au contraire de la recherche exhaustive de toutes les $n!$ permutations. On dispose d'algorithmes encore plus rapides (par exemple le tri fusion), qui effectuent de l'ordre de $n \log(n)$ opÃ©rations, et donc pour $n = 70$, de telles mÃ©thodes nÃ©cessitent moins de 1000 opÃ©rations.

Malheureusement, il n'est plus possible d'utiliser cette technique de classement dans des cas plus gÃ©nÃ©raux. Pour des points en dimension 2, si on prend comme coÃ»t $C_{i,\sigma(i)}$ la distance euclidienne (la distance en vol d'oiseau) entre les points, alors Gaspard Monge a montrÃ© dans son papier original (voir la figure **??**, Ã gauche) qu'un transport optimal ne peut par contenir de croisement. Par exemple, comme le montre la figure **??** (Ã droite), si l'on trace tous les segments entre les points $i \mapsto j = \sigma(i)$ que l'on relie par la bijection dÃ©finie par un σ optimal, ceux-ci ne se croisent jamais.

Cette observation gÃ©omÃ©trique n'est cependant pas suffisante pour calculer un transport optimal en 2D : il existe en effet beaucoup de permutations σ telles que les segments associÃ©s ne se croisent pas. Il va falloir analyser de faÃ§on plus fine la structure des permutations optimales afin de pouvoir les calculer de faÃ§on efficace. Nous allons maintenant voir comment Leonid Kantorovitch a reformulÃ© le problÃ¨me de Monge afin d'y parvenir.

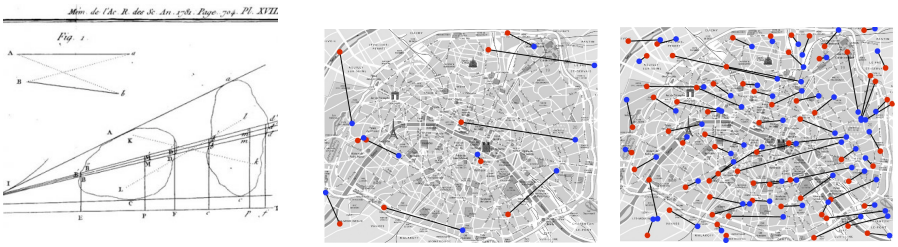


Figure 4.4: Gauche: extrait de l'article de Monge [**?**]. Droite: le transport optimal en 2D pour un coÃ»t euclidien.

4.2 Le Transport Optimal de Kantorovitch

Leonid Kantorovitch est un mathÃ©maticien et un Ã©conomiste soviÃ©tique qui a rÃ©volutionnÃ© la thÃ©orie du transport optimal pendant les annÃ©es 40. Ses recherches sont issues de considÃ©rations pratiques qui l'ont occupÃ© avant et aprÃ¨s la seconde guerre mondiale. Il y a jouÃ© un rÃ´le important pour assurer une distribution optimale des ressources, en particulier durant le siÃ¨ge de LÃ©ningrad. Il a par la mÃªme occasion participÃ© au dÃ©veloppement de l'optimisation moderne, laquelle a eu un impact Ã©norme dans de trÃ¨s nombreux domaines appliquÃ©s. Il a ainsi obtenu en 1975 le prix Nobel d'Ã©conomie, car les premiÃ¨res applications (mais certainement pas les seules !) de sa thÃ©orie l'ont Ã©tÃ© dans ce domaine.

Le problÃ¨me de Kantorovitch

L'idÃ©e centrale de Kantorovitch est de modifier le problÃ¨me de Monge en remplaÃ§ant l'ensemble des permutations par un ensemble plus grand mais plus simple. Tout d'abord on remarque que l'on peut reprÃ©senter une permutation $\sigma \in \Sigma_n$ Ã l'aide d'une matrice de permutation P qui est une matrice binaire (remplie de 0 et de 1) de taille $n \times n$ telle que $P_{i,j} = 0$ sauf si $j = \sigma(i)$ auquel cas $P_{i,\sigma(i)} = 1$. Par exemple, pour $n = 3$ points, les permutations $(1,2,3) \mapsto (1,2,3)$ (l'identitÃ©), $(1,2,3) \mapsto (3,2,1)$ et $(1,2,3) \mapsto (2,1,3)$ sont reprÃ©sentÃ©es par les matrices de taille 3×3

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad \text{et} \quad \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Dans la suite, on note $\mathcal{P}_n$ l'ensemble des $n!$ matrices de permutation de taille $n \times n$.

Comme la matrice est binaire, avec seulement n Ã©lÃ©ments non-nuls Ã©gaux Ã 1, on peut remplacer la somme de n termes qui apparait dans CoÃ»t(σ) dÃ©fini en (**??**) par une somme sur l'ensemble des $n \times n$ indices (i,j), c'est-Ã -dire que si P est la matrice de permutation associÃ©e Ã σ, on a

$$\text{CoÃ»t}(\sigma) = \sum_{i=1}^{n} \sum_{j=1}^{n} P_{i,j} C_{i,j}.$$

On peut ainsi remplacer le problÃ¨me de Monge (**??**) par le problÃ¨me Ã©quivalent

$$\min_{P \in \mathcal{P}_n} \sum_{i=1}^{n} \sum_{j=1}^{n} P_{i,j} C_{i,j}. \tag{4.4}$$

Le gÃ©nie de Kantorovitch a Ã©tÃ© de remarquer que l'on peut remplacer l'ensemble discret $\mathcal{P}_n$ (c'est-Ã -dire composÃ© d'un ensemble fini, mais trÃ¨s grand, de $n!$ matrices) par un ensemble Â« continu Â» (donc en particulier infini) mais plus simple. On remarque en effet que les matrices de permutation de $\mathcal{P}_n$ sont exactement les matrices qui ont un et un seul 1 le long de chaque ligne et de chaque colonne. Ceci peut aussi s'exprimer comme le fait qu'une matrice de permutation est une matrice binaire dont la somme de chaque ligne et de chaque colonne vaut 1, c'est-Ã -dire

$$\mathcal{P}_n = \left\{ P \in \{0,1\}^{n\times n} \; ; \; \forall i, \sum_j P_{i,j} = 1, \forall j, \sum_i P_{i,j} = 1 \right\}.$$

Ce qui rend cet ensemble trÃ¨s compliquÃ©, c'est la contrainte binaire, c'est-Ã -dire que ces matrices sont contraintes Ã Ã a tre dans $\{0,1\}^{n\times n}$. Kantorovitch propose alors de Â« relaxer Â» cette contrainte en supposant simplement que les entrÃ©es de P sont entre 0 et 1. Ceci dÃ©finit un ensemble plus grand, l'ensemble des matrices bistochastiques

$$\mathcal{B}_n \stackrel{\text{def.}}{=} \left\{ P \in [0,1]^{n\times n} \; ; \; \forall i, \sum_j P_{i,j} = 1, \forall j, \sum_i P_{i,j} = 1 \right\}. \tag{4.5}$$

Le problÃ¨me de Kantorovitch s'obtient en effectuant ce remplacement dans (**??**), afin de rÃ©soudre

$$\min_{P\in\mathcal{B}_n} \sum_{i=1}^{n}\sum_{j=1}^{n} P_{i,j}C_{i,j}. \tag{4.6}$$

L'immense avantage du problÃ¨me de Kantorovitch (**??**) par rapport Ã celui de Monge (**??**) est que l'ensemble des matrices bistochastique est convexes, c'est-Ã -dire que si l'on considÃ¨re deux matrices bistochastiques $P, Q \in \mathcal{B}_n$, alors leur moyenne $\frac{P+Q}{2} \in \mathcal{B}_n$ est encore bistochastique. Ceci n'est pas vrai pour les matrices de permutation, puisque la moyenne de deux matrices binaires (P, Q) n'est pas binaire (sauf si bien sÃ»r si $P = Q$). Cette convexitÃ© est la clef pour le dÃ©veloppement d'algorithmes efficaces. Cette nouvelle formulation a en effet pu bÃ©nÃ©ficier d'une deuxiÃ¨me rÃ©volution initiÃ©e par George Dantzig [**?**], qui, Ã la mÃªme Ã©poque, a proposÃ© l'algorithme du simplexe. Celui-ci permet de rÃ©soudre efficacement une certaine classe de problÃ¨mes d'optimisation convexe : les problÃ¨mes de programmation linÃ©aire, dont (**??**) est un cas particulier. Dans le cas du problÃ¨me de Kantorovitch, il existe en effet un algorithme du simplexe qui a une complexitÃ© de l'ordre de n^3 opÃ©rations, ce qui permet de faire des calculs pour de grands n, de l'ordre de plusieurs milliers.

L'Ã©quivalence Monge–Kantorovitch

L'ensemble des matrices bistochastiques est plus grand que celui des matrices de permutations, $\mathcal{P}_n \subset \mathcal{B}_n$, de sorte que l'on a l'inÃ©galitÃ©

$$\min_{P\in\mathcal{B}_n} \sum_{i=1}^{n}\sum_{j=1}^{n} P_{i,j}C_{i,j} \leqslant \min_{P\in\mathcal{P}_n} \sum_{i=1}^{n}\sum_{j=1}^{n} P_{i,j}C_{i,j} \tag{4.7}$$

entre les problÃ¨mes de Kantorovitch et de Monge. Mais de faÃ§on Ã premiÃ¨re vue surprenante, un thÃ©orÃ¨me fondamental dÃ» Ã George Birkhoff et Ã John von Neumann [**?** **?**] assure qu'en fait il y a Ã©galitÃ© entre les valeurs de ces deux minimisations. En effet, ce thÃ©orÃ¨me

montre qu'il existe toujours une matrice solution du problÃ¨me de Kantorovitch qui est une matrice de permutation, de sorte qu'elle est aussi solution du problÃ¨me de Monge. Attention cependant, en gÃ©nÃ©ral il n'y a pas unicitÃ© des solutions de ces problÃ¨mes : il peut exister une matrice bistochastique solution du problÃ¨me de Kantorovitch qui n'est pas une permutation. La conjonction de deux avancÃ©es spectaculaires, dues Ã Kantorovitch et Ã Dantzig, a permis de rendre le transport optimal applicable Ã des problÃ¨mes de grande taille, puisque l'algorithme du simplexe permet de rÃ©soudre en pratique ces problÃ¨mes.

Le cas pondÃ©rÃ©

Outre son intÃ©rÃªt pratique, la formulation de Kantorovitch a aussi permis de gÃ©nÃ©raliser le problÃ¨me initial de Monge, en donnant le bon cadre pour le formaliser et l'Ã©tudier mathÃ©matiquement. En effet, le problÃ¨me de Monge est trÃ¨s limitÃ©. Que se passe-t-il par exemple si il n'y pas le mÃªme nombre n de cafÃ©s et m de boulangeries ? Le problÃ¨me initial (**??**) n'a pas de solution, car on ne peut pas mettre en bijection deux ensembles de tailles diffÃ©rentes. Le bon concept n'est pas le nombre de boulangeries et de cafÃ©s, mais plutÃ´t les distributions $(a_1, \dots, a_n)$ de production (associÃ©es au boulangeries) et les distributions $(b_1, \dots, b_m)$ de consommation des cafÃ©s. Par exemple, si la premiÃ¨re boulangerie produit 45 croissants par jour, on prendra $a_1 = 45$, de mÃªme $b_3 = 34$ signifie que le 3^{e} cafÃ© consomme 34 croissants par jour. Dans le cas initialement considÃ©rÃ©, oÃ¹ $n=m, \textit{toutes les quantit\'es} a_i$ et b_j sont Ã©gales Ã 1. Mais dans de nombreux cas concrets, ces quantitÃ©s sont quelconques. Ces quantitÃ©s doivent Ãªtre positives, et vÃ©rifier

$$a_1 + \cdots + a_n = b_1 + \cdots + b_m,$$

de sorte qu'il y ait autant de production que de consommation. La construction de Kantorovitch s'adapte naturellement Ã ce cas de distributions gÃ©nÃ©rales, en remplaÃ§ant les matrices bistochastique (**??**) par des matrices de Â« couplage Â» qui satisfont la contrainte de conservation de la masse

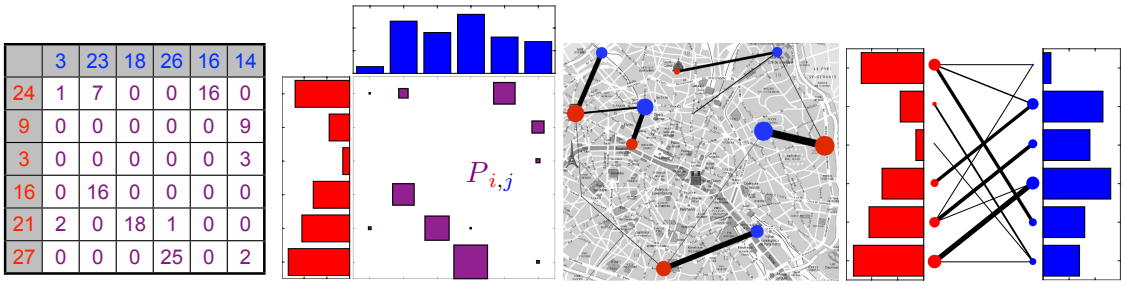
$$\mathcal{B}(a,b) \stackrel{\text{def.}}{=} \left\{ P \in [0,1]^{n \times m} \;;\; \forall\, i, \sum_j P_{i,j} = a_i, \forall\, j, \sum_i P_{i,j} = b_j \right\}.$$

Dans le cas initial oÃ¹ $n=m et a_i = b_j = 1$, alors $\mathcal{B}(a,b) = \mathcal{B}_n$ et l'on retrouve des matrices bistochastiques. Dans le cas gÃ©nÃ©ral, Ã chaque fois qu'une entrÃ©e $P_{i,j}$ est non-nulle, ceci signifie que l'on transfert de la Â« masse Â» (ici une certaine quantitÃ© de croissants) entre i et j. Comme le montre la figure **??**, on peut visualiser de diffÃ©rentes faÃ§ons une telle matrice P couplant deux distributions (a,b). Contrairement au cas des matrices bistochastiques, pour lequel il y a toujours une solution qui est une permutation, ici un couplage optimal $\mathcal{B}(a,b)$ peut avoir plus d'une seule entrÃ©e non-nulle $P_{i,j}$ le long d'une ligne indexÃ©e par i (voir la figure **??**). Ceci signifie que cette boulangerie i est connectÃ©e Ã plusieurs cafÃ©s, de sorte que sa production est alors sÃ©parÃ©e en plusieurs lots de croissants distribuÃ©s, tout en satisfaisant la contrainte de conservation de la masse $\sum_j P_{i,j} = a_i$.

Le problÃ¨me de Kantorovitch qui gÃ©nÃ©ralise (**??**) s'Ã©crit alors

$$\min_{P \in \mathcal{B}(a,b)} \sum_{i=1}^{n} \sum_{j=1}^{m} P_{i,j} C_{i,j} \tag{4.8}$$

ce qui signifie que l'on doit payer un coÃ»t $C_{i,j}$ Ã chaque fois que l'on transfert une unitÃ© de



(a) matrice (b) histogrammes (c) segments (d) graphe biparti

Figure 4.5: DiffÃ©rentes faÃ§ons de reprÃ©senter une matrice de couplage $P \in \mathcal{B}(a, b)$: (a) un tableau de nombres dont les lignes et colonnes ont des sommes prescrites ; (b) un histogramme bidimensionnel dont la taille de carrÃ© est propositionnelle Ã $P_{i,j}$; (c) un ensemble de segments dont la largeur est proportionnelle est $P_{i,j}$. (d) un graphe biparti, c'est-Ã -dire avec deux ensembles de sommets tels que les arrÃªtes soient seulement entre ces deux ensembles.

masse entre i et j. Tout comme le problÃ¨me original (**??**), on peut le rÃ©soudre de faÃ§on efficace avec l'algorithme du simplexe. La figure **??** montre un exemple de couplage optimal.

4.3 Les applications

Bien que les motivations initiales de Monge et Kantorovitch Ã©taient respectivement militaires et Ã©conomiques, le transport optimal a trouvÃ© d'innombrables applications, Ã la fois thÃ©oriques mais aussi plus concrÃ¨tes. Sur le plan mathÃ©matique, on peut considÃ©rer des distributions Â« continues Â» de masses, en quelque sorte la limite quand le nombre de point n tend vers l'infini. Ceci permet de dÃ©finir le problÃ¨me de transport entre des mesures de probabilitÃ©s quelconques. Ce point de vue thÃ©orique est extrÃªmement fructueux, et c'est le mathÃ©maticien franÃ§ais Yann Brenier qui a le premier montrÃ© l'Ã©quivalence dans le cadre continu des formulations de Monge et de Kantorotich [**?**]. Ces travaux pionniers ont montrÃ© la connexion entre le problÃ¨me de transport et les Ã©quations aux dÃ©rivÃ©es partielles, et ont dÃ©bouchÃ©, entre autres, sur les mÃ©dailles Fields de CÃ©dric Villani (2010) et Alessio Figalli (2018).

Le transport optimal est depuis peu au cœur de problÃ©matiques plus appliquÃ©es en sciences des donnÃ©es, en particulier pour rÃ©soudre des problÃ¨mes en traitement d'image et en apprentissage machine. La premiÃ¨re idÃ©e, la plus immÃ©diate, est d'utiliser la bijection σ afin de transformer des donnÃ©es, par exemple des images. Dans ce cas, on considÃ¨re les pixels $(x_i)_{i=1}^n$ et $(y_j)_{j=1}^n$ de deux images couleur. Chaque pixel $x_i, y_j \in \mathbb{R}^3$ est un vecteur de dimension 3, qui reprÃ©sente les intensitÃ©s de chacune des trois couleurs Ã©lÃ©mentaires, rouge, vert et bleu. Afin de changer les couleurs de la premiÃ¨re image, et lui imposer la palette de la deuxiÃ¨me image, on calcule le transport σ pour la matrice de coÃ»t $C_i = \|x_i - y_j\|^2$ (c'est-Ã -dire le carrÃ© de la norme euclidienne dans $\mathbb{R}^3$), c'est-Ã -dire le carrÃ© de la distance euclidienne entre les pixels. L'image avec les couleurs modifiÃ©es est $(y_{\sigma(i)})_{i=1}^n$, c'est Ã dire que l'on remplace dans la premiÃ¨re image le pixel x_i par le pixel $y_{\sigma(i)}$. Cette image ressemble Ã la premiÃ¨re, mais a la palette de couleurs de la deuxiÃ¨me image. La figure **??** illustre ce procÃ©dÃ© pour imposer la palette de couleurs de Picasso Ã un tableau de CÃ©zanne.

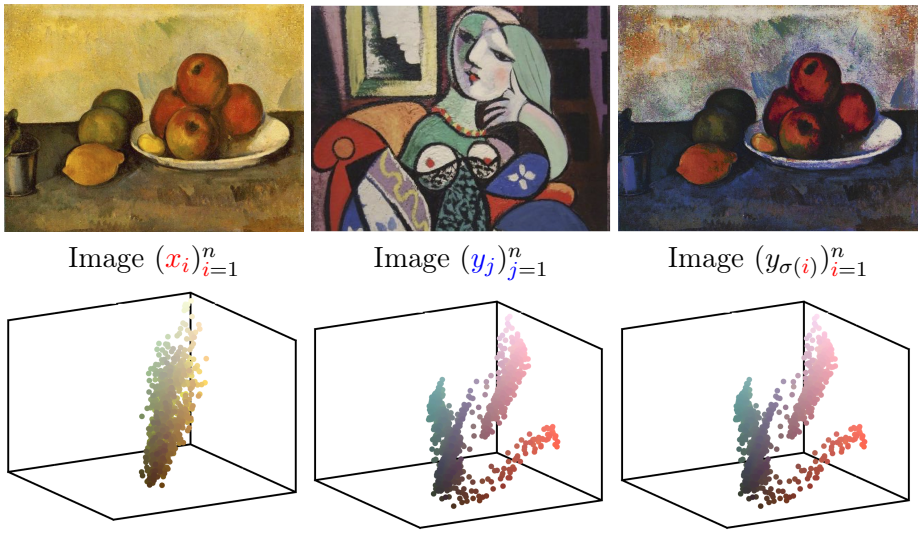


Figure 4.6: Exemple de transfert de palettes de couleurs Ã l'aide du transport optimal. Haut: les pixels sont sur la grille d'affichage pour former une image couleur. Bas: les pixels sont placÃ©s Ã leurs positions dans $\mathbb{R}^3$ pour former un nuage de points.

On peut Ã©galement utiliser le transport optimal pour des problÃ¨mes plus difficiles, en n'utilisant que de faÃ§on indirecte la bijection σ ou bien la matrice de couplage optimal $P \in \mathcal{B}(a,b)$. L'idÃ©e centrale est que la quantitÃ© associÃ©e Ã un couplage optimal P solution de (**??**)

$$W(a,b) \stackrel{\text{def.}}{=} \sum_{i,j} P_{i,j} C_{i,j}$$

dÃ©finit en quelque sorte l'effort nÃ©cessaire pour dÃ©placer la masse de la distribution a vers la distribution b. Elle permet donc de quantifier combien ces deux distributions sont Â« proches Â». Par exemple, si $C_i = \|x_i - y_j\|^2$ est le carrÃ© de la distance euclidienne entre des points, alors la quantitÃ© $W(a,b)^{1/2}$ est une distance entre les distributions, en particulier elle vÃ©rifie $W(a,b) = 0$ si et seulement si $a = b$, et elle vÃ©rifie l'inÃ©galitÃ© triangulaire. Ces propriÃ©tÃ©s sont trÃ¨s importantes pour permettre d'appliquer le transport Ã des problÃ¨mes pratiques.

Un exemple typique d'application de cette quantitÃ© W consiste Ã calculer des barycentres entre des distributions [**?**]. La figure **??** montre un exemple oÃ¹ $l'on considÃ¨re trois distributions a, b, c (montrÃ©es aux tr$ – *dire que la masse* a_i associÃ©e au i^{e} point est 0 Ã l'extÃ©rieur de la premiÃ¨re forme et prend une valeur constante Ã l'intÃ©rieur). On calcule un barycentre pondÃ©rÃ© de ces trois distributions en imitant le fait que dans un espace Euclidien, le barycentre pondÃ©rÃ© r de trois points x, y, z minimise la somme des distances au carrÃ©

$$\min_r \alpha\|x-r\|^2 + \beta\|y-r\|^2 + \gamma\|z-r\|^2,$$

oÃ¹ *les poids* (α, β, γ) sont les pondÃ©rations du barycentre, qui sont des rÃ©els positifs et tels que $\alpha + \beta + \gamma = 1$. Le barycentre pondÃ©rÃ© d de (a,b,c) minimise ainsi la somme pondÃ©rÃ©e de distances de transport optimal

$$\min_d \alpha W(a,d) + \beta W(b,d) + \gamma W(c,d). \tag{4.9}$$

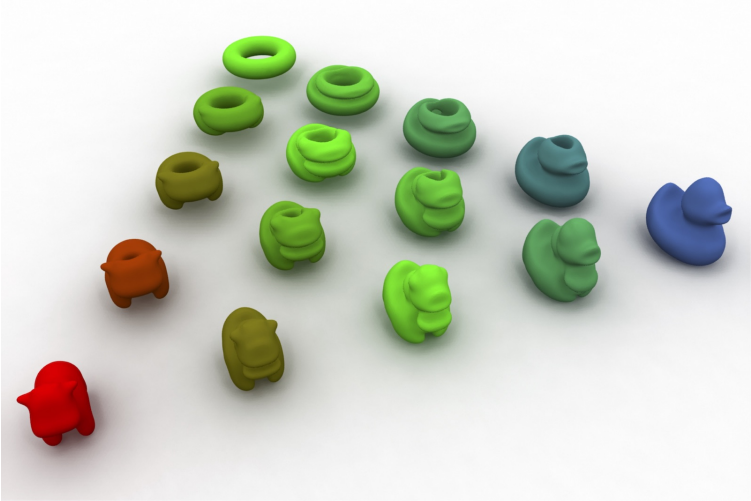


Figure 4.7: Exemple d'interpolation barycentrique entre des formes 3D, obtenu en minimisant (**??**).

Figure 4.8: Exemples d'histogrammes de distributions des mots dans deux textes diffÃ©rents (seuls les mots les plus frÃ©quents sont montrÃ©s).

En modifiant les poids (α, β, γ), on modifie la forme obtenue en se dÃ©plaÃ§ant Ã l'intÃ©rieur d'un triangle de transport optimal. On peut utiliser cette distance W pour bien d'autres applications oÃ¹ $l'ondoitcomparerdesdistributionsdeprobabilit©.C'estlecasenapprentissagemachine, parexemplepourcompar$ Une question difficile dans ce cas est de savoir quelle matrice de coÃ»t $C_{i,j}$ utiliser entre deux mots (i, j). Il s'agit d'un travail de linguistique (caractÃ©riser la proximitÃ© sÃ©mantique entre des mots du langages), que l'on peut chercher Ã rÃ©soudre en mÃªme temps que le transport optimal [**?**].

Conclusions

Le transport optimal a connu de nombreuses rÃ©volutions. Sous l'impulsion de mathÃ©maticiens tels que Monge, Kantorovitch, Dantzig et Brenier, il est progressivement devenu un outil thÃ©orique et numÃ©rique fondamental. Il est maintenant au cœur de questions importantes en science des donnÃ©es pour modÃ©liser, rÃ©soudre numÃ©riquement et analyser thÃ©oriquement les problÃ¨mes de l'apprentissage machine. Les opportunitÃ©s pour dÃ©velopper de nouvelles thÃ©ories et des algorithmes performants sont immenses. Pour plus d'informations sur les aspects thÃ©oriques du transport optimal, on pourra consulter les livres [**? ?**]. Les aspects numÃ©riques et applicatifs sont couverts dans le livre [**?**].

Remerciements

Je tiens Ã remercier Vincent Beck, Gwenn Guichaoua et Marie-NoÃ«lle PeyrÃ© pour leurs relectures attentives.