RECONSTRUCTION D'IMAGES PAR LA STATISTIQUE BAYESIENNE

Projet de recherche CRSNG

Par Audrey Béliveau

Supervisé par Mylène Bédard

Université de Montréal

Département de mathématiques et de statistique

31 octobre 2008

Table des matières

Introduction **3**

Notions préalables **5**

Modélisation de l'image *5*

Un peu de notation *6*

Modélisation de la dégradation de l'image *8*

Mesures sur l'image **10**

Pour une image en noir et blanc *10*

Pour une image en niveaux de gris *12*

Pour une image en couleurs *14*

Le modèle bayésien **15**

Les probabilités à priori et à postériori et la formule de Bayes *15*

La forme gibbsienne *16*

Les estimateurs bayésiens *17*

Reconstruction de l'image par les MCMC **18**

Le problème de la minimisation *18*

Théorie sur les chaînes de Markov *18*

L'algorithme de Metropolis-Hastings *20*

L'algorithme de Gibbs *24*

L'algorithme de Metropolis-within-Gibbs *26*

Les estimateurs bayésiens plus en détails **28**

MAP *28*

MMSE *29*

Les résultats obtenus **31**

Images en noir et blanc *31*

Images en niveaux de gris *32*

Images en couleur *33*

Interprétation des résultats **34**

Bibliographie **35**

Introduction

Le projet de recherche qui suit a été effectué dans le cadre d'un stage d'été CRSNG, sous la supervision de la professeure Mylène Bédard. Le but principal du projet était de découvrir l'utilité des méthodes de Monte Carlo par Chaîne de Markov (MCMC) dans le domaine de la reconstruction d'image. Le principe de la reconstruction d'image part de l'observation d'une image dégradée, le but étant de retrouver l'image d'origine (non-dégradée). La démarche consiste à proposer plusieurs images afin de pouvoir retenir l'image la plus proche possible de l'image d'origine.

Pour mener à bien notre projet, nous avons d'abord défini des fonctions d'énergie sur l'image. Ces fonctions d'énergie ont pour résultat un nombre qui devrait être une mesure de l'adéquation de l'image. Plus la valeur d'énergie est petite, plus l'image devrait être « bonne », en général.

Puis, en passant par un modèle bayésien, nous avons établi une équivalence entre la valeur de la fonction d'énergie d'une image proposée donnée et sa probabilité dite à postériori. Il s'agit de la probabilité que cette image proposée soit la même que l'image d'origine étant donné l'image observée. La probabilité à postériori peut s'écrire sous une forme, dite gibbsienne, qui dépend uniquement de l'énergie. Cette équivalence repose sur la théorie des champs de Markov et le théorème de Hammersley-Clifford.

Avec l'équivalence ci-haut, il devenait facile de trouver la probabilité à postériori d'une image, car l'énergie d'une image se calcule facilement et le passage de l'énergie à la probabilité est aussi un calcul facile. On pouvait donc associer, à chaque image proposée possible, une probabilité que cette image soit l'image d'origine que nous cherchions.

Par la suite, il fallait un moyen de décider quelle image proposée choisir. Nous avons choisi de comparer deux estimateurs : le MAP (maximal posterior estimate) et le MMSE (posterior minimum mean square estimate). Le MAP est probablement l'estimateur le plus intuitif : il s'agit tout simplement de retenir l'image ayant la plus grande probabilité à postériori. Pour sa part, le MMSE consiste à effectuer la moyenne des images pondérées par leur probabilité à postériori.

Au point où nous en étions, nous avions toute la théorie nécessaire pour être capable de conceptualiser ce qu'est la reconstruction d'image. Mais, supposons que nous voulions reconstruire une image par le MAP. De prime abord, nous avons pensé à considérer toutes les images possibles pour choisir celle avec la probabilité à postériori maximale. Mais, ce n'était pas aussi simple. En effet, pour reconstruire une petite image en noir et blanc de 10 000 pixels seulement, le nombre d'image possibles à considérer se serait élevé à $2^{10\,000}$, ce qui représente un nombre à plus de 3 010 chiffres! Pour une image plus grande, en niveau de gris ou en couleurs, le nombre de possibilités à considérer devenait rapidement encore plus effrayant! Nous n'osions même pas imaginer le temps de calcul qu'aurait pu prendre un ordinateur.

C'est alors que sont entrées en jeu les méthodes de Monte Carlo par chaîne de Markov (MCMC). Grâce aux méthodes MCMC, le nombre d'images à considérer n'était plus astronomique : seulement un échantillon représentatif de ces images pouvait être utilisé. En fait, les algorithmes MCMC génèrent une suite d'images dont la loi converge vers la distribution de probabilité à postériori. Les images avec une grande probabilité à postériori sont donc plus nombreuses dans l'échantillon.

Nous avons comparé trois algorithmes MCMC différents : Metropolis-Hastings, Gibbs et Metropolis within Gibbs. Les algorithmes ont été testés pour des images en noir et blanc, des images en niveau de gris et des images en couleur. Chacun de ces cas a été traité d'une part avec l'estimateur MAP, d'autre part avec le MMSE. Au cours de mon stage, j'ai programmé les algorithmes de reconstruction d'images nécessaires à mon projet en langage C. Un certain nombre d'images obtenues seront présentées à la fin de ce rapport. Pour terminer, je recommande fortement d'accompagner la lecture de ce rapport par deux ouvrages, qui m'ont été particulièrement utiles tout au long de mon stage : *Image Analysis, Random Fields and Markov Chain Monte Carlo Methods* de Gerhard Winkler (pour les aspects théorique et pratique de la reconstruction d'image) et *Simulation* de Sheldon M. Ross (pour l'aspect théorique des méthodes MCMC).

Notions préalables

Modélisation de l'image

Pour effectuer de la reconstruction d'image par ordinateur, il faut d'abord savoir comment traduire une image en un modèle mathématique pouvant être recueilli et traité par l'ordinateur.

Le plus petit élément constitutif d'une image numérique est le *pixel*, mot provenant de la locution anglaise « ***pic****ture* ***el****ement* ». On peut se représenter les pixels par de minuscules points de couleurs. Ceux-ci sont disposés dans un tableau bidimensionnel, formant ainsi l'image. Mathématiquement parlant, l'intensité d'un pixel (sa couleur) est donnée par un nombre (ou par un vecteur de nombres, dans le cas des images en couleurs – cela sera expliqué un peu plus loin). Une image est donc représentée par un tableau bidimensionnel de nombres (ou de vecteurs de nombres), chacun d'eux représentant l'intensité du pixel associé sur l'image.

Les images peuvent être décrites différemment selon qu'elles soient en noir et blanc, en tons de gris ou en couleurs. Pour les images en noir et blanc (aussi appelées images binaires), seulement deux intensités sont possibles, par exemple, 0 (blanc) et 1 (noir).

Pour les images en niveaux de gris, les intensités peuvent, par exemple, prendre des valeurs entières entre 0 (noir) et 255 (blanc). Les valeurs intermédiaires traduisent les teintes de gris entre le noir et le blanc.

Quant aux images en couleurs, chacun de leurs pixels est décrit par trois composantes : rouge, vert et bleu. Ce format de codage des couleurs est abrévié par RVB. Le rouge, le vert et le bleu sont les couleurs choisies, car en variant leurs intensités, on peut obtenir toute la gamme des couleurs. Mais attention, les couleurs « lumières » d'un écran d'ordinateur ne se mélangent pas de la même façon que les couleurs « matières ». En effet, nous avons plus facilement l'intuition de ces dernières, car elles se mélangent comme la peinture. Mais le système dans lequel nous travaillerons (les couleurs « lumières ») en est complètement

l'inverse. En effet, le mélange de deux couleurs donnera toujours une couleur plus lumineuse. L'addition de rouge, de vert et de bleu donnera donc du blanc alors que l'absence de lumière donnera du noir. Les couleurs secondaires, le jaune, le magenta et le cyan, peuvent être obtenues par l'addition de deux couleurs primaires, comme le montre la figure 1.

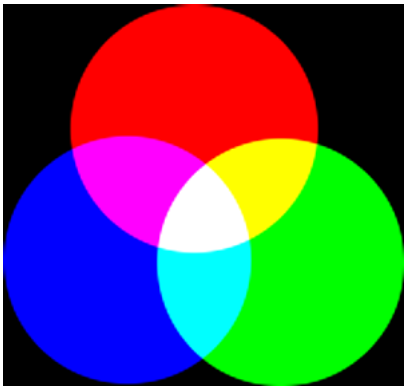


Figure 1 : Le rouge, le vert et le bleu s'additionnent aux intersections

L'intensité de chacune des trois composantes RVB d'un pixel prend, par exemple, une valeur entre 0 et 255. Ainsi, le rouge est représenté par (255, 0, 0), le vert par (0, 255, 0) et le bleu par (0, 0, 255). De même, le jaune est représenté par (255, 255, 0), le magenta par (255, 0, 255) et le cyan par (0, 255, 255). Enfin, le noir et le blanc sont respectivement représentés par les triplets (0, 0, 0) et (255, 255, 255).

Un peu de notation

Le principe de la reconstruction d'image part de l'observation d'une image dégradée, le but étant de retrouver l'image d'origine (non-dégradée). La démarche consiste à proposer des images améliorées pour l'image observée afin de pouvoir retenir l'image la plus proche possible de l'image d'origine.

Puisque nous utiliserons des modèles mathématiques pour évaluer la convenance d'une image, nous devrons utiliser des symboles pour désigner certains concepts. Voici donc un tableau résumant les principaux symboles qui seront utilisés dans ce rapport :

Symbole	Définition
s	site de pixel
S	ensemble des sites de pixels de l'image
x	image d'origine
X	image d'origine vue comme une variable aléatoire
y	image observée (dégradée)
Y	image observée (dégradée) vue comme une variable aléatoire
g	image proposée
G	image proposée vue comme une variable aléatoire
x_s	intensité du pixel s de l'image d'origine
y_s	intensité du pixel s de l'image observée
g_s	intensité du pixel s de l'image proposée
$s \sim t$	le pixel t est voisin du pixel s

Quelques clarifications s'imposent. D'abord pour s (un certain site de pixel de l'image), il est à savoir qu'il est possible de le noter de deux façons. Une première façon serait de noter $s = (i , j)$ où i et j représentent les coordonnées du site de pixel dans la matrice que forme l'image. Une deuxième façon serait de noter $s = i$, où i représente le rang du site de pixel dans la matrice de l'image, en comptant les pixels de gauche à droite puis de haut en bas. L'une et l'autre de ces notations peuvent être utilisées en tout temps. Nous prendrons donc la notation qui convient le mieux selon le cas.

Il reste aussi à définir ce qu'est un voisin. On dit que le pixel t est voisin de s s'il se trouve dans son voisinage. Il existe plusieurs modèles de voisinage. Celui que nous utiliserons dans ce rapport est le plus simple : les voisins de s sont les pixels qui sont à côté à gauche, à droite, en haut et en bas. Nous avons choisi de travailler avec ce modèle, car il est le moins demandant au niveau du temps de calcul (étant donné le petit nombre de pixels voisins). Un autre modèle communément utilisé est celui où l'on rajoute les quatre pixels à côté de s diagonalement. La figure 2 représente ces deux modèles, appelés respectivement modèle de premier ordre et modèle du second ordre.

Figure 2 : Modèles du 1er et 2e ordre, chaque point représente un pixel du voisinage

Des modèles de voisinage plus complexes peuvent aussi être utilisés bien qu'ils soient beaucoup plus coûteux en temps de calcul. Comment générer de tels modèles? D'abord, nous voulons qu'un modèle de voisinage soit symétrique autour du pixel central selon l'axe horizontal, l'axe vertical et les deux axes diagonaux. Pour générer le prochain modèle, il faut rajouter les pixels qui sont les plus près du pixel central parmi les pixels qui ne sont pas encore dans le voisinage. Pour savoir quels pixels sont les plus près, il suffit d'utiliser la bonne vieille formule de Pythagore. La figure 3 ci-dessous illustre les trois prochains modèles générés.

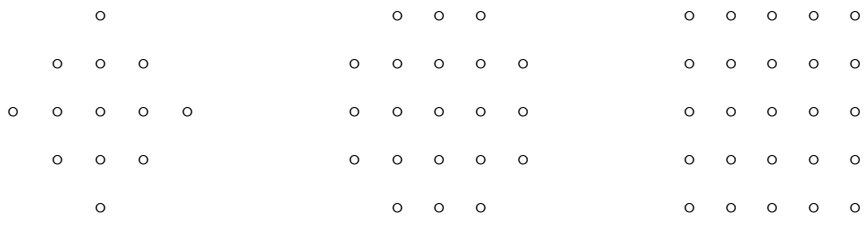


Figure 3 : Modèles du 3e, 4e et 5e ordre, chaque point représente un pixel du voisinage

L'importance de définir de tels modèles est due au fait qu'en reconstruction d'image par la statistique bayésienne, nous acceptons l'hypothèse selon laquelle l'intensité de chaque pixel de l'image ne dépend que de l'intensité de ses pixels voisins. Cette hypothèse est raisonnable, car nous savons que les pixels qui sont près les uns des autres ont davantage tendance à avoir des couleurs semblables que les pixels qui sont éloignés.

Modélisation de la dégradation de l'image

Lors de l'utilisation d'appareils d'imagerie tels que les appareils photos numériques ou les appareils d'imagerie médicale, les pixels de l'image subissent des dégradations pendant la

prise de vue. Parmi les dégradations possibles créées par la plupart des systèmes photochimiques et photoélectriques, on compte : le bruit, le flou et la distorsion.

Le bruit est une transformation qui suit une distribution aléatoire (discrète ou continue) s'appliquant sur chaque pixel soit par addition, par multiplication ou par une autre opération inversible. Les bruits s'appliquant sur chacun des pixels sont i.i.d. Ils peuvent provenir, par exemple, d'une distribution gaussienne ou d'une distribution de Poisson. Pour sa part, le flou est une transformation linéaire qui se représente par une matrice. Finalement, la distorsion est une transformation non-linéaire sans mémoire. En général, elle s'exprime par une fonction logarithmique (film photographique) ou algébrique (télévision). Ces trois éléments se combinent pour former le modèle de dégradation de l'image :

$$Y_s = \Phi\left(\Theta\left((Bx)_s\right), \eta_s\right) \ \forall s \in S \, .$$

Ici, B représente la matrice associée au flou, η_s le bruit appliqué au pixel s, Θ l'opération inversible selon laquelle le bruit s'applique et Φ la distorsion.

Mesures sur l'image

Pour une image en noir et blanc

Le but de notre projet étant de restaurer des images ayant subi des dégradations, il faut d'abord pouvoir juger mathématiquement de l'adéquation d'une image. Par exemple, jugez-vous que l'image observée ci-dessous est « bonne » ?

En effet, cette image pourrait être améliorée. Comment ? En changeant les pixels noirs aberrants par des pixels blancs. Mais, qu'est-ce qui permet de dire que ces pixels noirs sont aberrants ?

Ces pixels noirs sont jugés aberrants parce qu'ils sont entourés de pixels blancs. Mathématiquement, nous définissons la fonction suivante :

$$K(g) = \sum_{s \sim t} (g_s - g_t)^2 .$$

K est aussi appelée une *fonction d'énergie*. Cette fonction effectue la somme des différences au carré entre les intensités et ce, sur toutes les paires de pixels voisins constituant l'image observée. Il est à noter que même si *s* est voisin de *t* et *t* est voisin de *s*, il ne faut compter cette paire de voisins qu'une seule fois dans le calcul de l'énergie

Rappelez-vous que les intensités des pixels valent 0 pour un pixel blanc et 1 pour un pixel noir. Lorsque deux pixels voisins ont la même intensité, $(g_s - g_t)^2$ vaut 0. Par contre, lorsque

deux pixels voisins n'ont pas la même intensité, $(g_s - g_t)^2$ vaut 1. Les pixels dégradés sur notre image ont donc eu pour effet d'augmenter la valeur de *K*. Pour proposer une « bonne » image, on cherchera donc à minimiser *K*.

On peut donc interpréter *K* comme étant une mesure de la régularité de l'image. Toutefois, la régularité à elle seule n'est pas suffisante. Par exemple, regardez l'image proposée suivante :

Pour cette image, on a

$$\begin{aligned} K(g) &= \sum_{s \sim t} (g_s - g_t)^2 \\ &= \sum_{s \sim t} 0 \\ &= 0, \end{aligned}$$

car tous les pixels ont la même intensité. Nous avons donc trouvé la valeur minimale possible pour *K*. Cependant, l'image proposée diffère beaucoup trop de l'image observée précédemment. Voilà pourquoi nous introduisons une deuxième mesure sur l'image :

$$D(g, y) = \sum_{s} (g_s - y_s)^2 .$$

Cette fonction d'énergie est appelée en anglais « Hamming distance ». Elle permet de comparer l'image proposée avec l'image observée en donnant une mesure de leur ressemblance. La fonction d'énergie *D* effectue la somme, sur tous les pixels, de la différence au carré entre les intensités de chaque pixel de l'image observée et de l'image proposée. Lorsque le pixel *s* de l'image observée et celui de l'image proposée ont la même intensité, $(g_s - y_s)^2$ vaut 0. Par contre, lorsqu'ils n'ont pas la même intensité, $(g_s - y_s)^2$ vaut 1. Puisque nous

voulons proposer une image qui est meilleure que l'image observée sans toutefois qu'elle n'en diffère trop, nous chercherons aussi à minimiser *D*.

Il faut maintenant combiner ces deux modèles d'énergie. En effet, minimiser *K* uniquement mènera à une image de couleur uniforme alors que minimiser *D* uniquement mènera à une image identique à l'image observée. Voici donc plutôt le modèle général que nous utiliserons :

$$\begin{aligned} H(g,y) &= K(g) + D(g,y) \\ &= \lambda^2 \sum_{s\sim t} (g_s - g_t)^2 + \sum_s (g_s - y_s)^2. \end{aligned}$$

La fonction *H* représente l'énergie totale de l'image proposée. Nous souhaitons trouver l'image qui minimise *H*, car il s'agira probablement de l'image se rapprochant le plus possible de la « bonne » image que nous cherchons.

Remarquez qu'un paramètre λ a été ajouté dans *K*. Son but est de pondérer l'importance de *K* par rapport à *D*. Plus on choisit λ grand, plus on donne de poids à *K* et plus on obtiendra une image régulière. Par contre, plus on choisit λ petit, plus on donne de poids à *D* et plus on obtiendra une image semblable à l'image observée. Il faudra donc choisir λ judicieusement selon l'image observée. Pour donner un ordre d'idées, λ prend grosso modo des valeurs entre 0.1 et 2. Dans tous nos exemples, nous avons choisi λ expérimentalement (par essais-erreurs). Il s'agit en fait d'une technique plutôt répandue.

Pour une image en niveaux de gris

Le modèle pour une image en niveaux de gris est très similaire. L'énergie *H* telle que définie précédemment reste une bonne base pour décrire l'énergie globale de l'image :

$$\begin{aligned} H(g,y) &= K(g) + D(g,y) \\ &= \lambda^2 \sum_{s\sim t} (g_s - g_t)^2 + \sum_s (g_s - y_s)^2. \end{aligned}$$

L'avantage d'utiliser la puissance carrée dans un modèle à 256 intensités est qu'elle permet de pénaliser fortement les grandes différences d'intensité en augmentant beaucoup l'énergie.

Ici, on cherchera donc encore à trouver l'image qui minimise *H*. En effet, minimiser *H* signifie trouver *K* et *D* petits et minimisés dans une certaine proportion donnée par λ. Or, minimiser *K* permettra de trouver une image plutôt régulière; aussi, minimiser *D* permettra de trouver une image qui ressemble à l'image observée. C'est ce que nous voulons.

Toutefois, le passage à 256 intensités engendre un problème dû au contour des formes dans l'image. En effet, les pixels formant un contour ont généralement une intensité très différente des pixels autour. Ils sont donc perçus à tord comme une dégradation de l'image selon le modèle d'énergie défini précédemment. Cela aura comme effet de lisser les contours alors que nous préférerions que les contours soient bien définis. Voilà pourquoi nous introduisons la notion de micro-barrière (« microedge » en anglais). On peut visualiser la micro-barrière comme une barre entre deux pixels :

∘ | ∘ .

La micro-barrière peut occuper deux états : allumée ou éteinte, comme un interrupteur. Elle est allumée lorsque la différence d'intensité entre les deux pixels concernés est suffisamment grande, autrement elle est éteinte. On apporte donc la modification suivante au modèle d'énergie de l'image :

$$H(g,y) = \lambda^2 \sum_{s \sim t} \left(\left(g_s - g_t \right)^2 \left(1 - e_{st} \right) + \alpha e_{st} \right) + D\left(g, y \right)$$

où

$$e_{st} = \begin{cases} 0 \text{ si la micro-barrière est éteinte} \\ 1 \text{ si la micro-barrière est allumée.} \end{cases}$$

Ainsi, lorsqu'une micro-barrière est éteinte, l'intérieur de la somme vaut $(g_s - g_t)^2$ comme auparavant. Toutefois, lorsqu'une micro-barrière est allumée, c'est-à-dire lorsque la différence d'intensité entre les deux pixels voisins est considérée trop grande, l'intérieur de la

somme vaut α. Ce paramètre, α, est généralement choisi plus petit que les $(g_s - g_t)^2$, car ainsi, les contours peuvent survivre.

Il reste maintenant à définir ce qu'est un « écart d'intensité trop grand entre deux pixels voisins ». Pour cela, nous introduisons le seuil δ. Ainsi, la micro-barrière sera éteinte si $| g_s - g_t | \leq \delta$ et allumée si $| g_s - g_t | > \delta$. Le modèle final utilisé sera donc :

$$H(g, y) = \lambda^2 \sum_{s \sim t} \left(\left(g_s - g_t \right)^2 \left(1 - e_{st} \right) + \alpha e_{st} \right) + D\left(g, y \right)$$

où

$$e_{st} = \begin{cases} 0 \text{ si } | g_s - g_t | \leq \delta \\ 1 \text{ si } | g_s - g_t | > \delta. \end{cases}$$

Pour une image en couleurs

La formule d'énergie totale pour une image en couleur est la même que celle pour une image en niveaux de gris. La seule différence, pour une image en couleurs, est qu'il faut traiter l'image comme trois images distinctes : une avec les teintes de rouge, une avec les teintes de vert et une avec les teintes de bleu. Ces trois images doivent être corrigées séparément en utilisant la même formule d'énergie que pour les images en niveaux de gris. Une fois les trois images corrigées, il suffit de les recombiner en une seule image pour obtenir le résultat attendu.

Le modèle bayésien

Les probabilités à priori et à postériori et la formule de Bayes

Le modèle bayésien est un modèle basé sur les probabilités. Il repose avant tout sur la formule de Bayes. Supposons que nous souhaitions reconstruire une certaine image d'origine x après avoir observé une image dégradée y. Pour ce faire, nous proposons des images possibles pour x. Il faut toutefois être en mesure de décider, pour chaque image proposée, si celle-ci serait une bonne image pour x. On définit la probabilité *à postériori*, $P(X = g \mid Y=y)$, comme étant la probabilité qu'une image proposée g soit la même que l'image d'origine étant donné l'image observée. On peut donc avoir tendance à penser que les images proposées les plus près de l'image d'origine auront les plus grandes probabilités à postériori. Grâce à la formule de Bayes, nous pouvons expliciter la formule de probabilité à postériori :

$$P\left(X = g \mid Y = y\right) = \frac{\Pi\left(X = g\right) P\left(Y = y \mid X = g\right)}{\sum_g \Pi\left(X = g\right) P\left(Y = y \mid X = g\right)},$$

où $\Pi(X = g)$ est la probabilité que l'image proposée g soit en fait l'image d'origine et $P(Y=y|X=g)$ est la probabilité d'observer y si g est l'image d'origine.

Pour obtenir la probabilité à postériori, il est nécessaire avant tout de définir $\Pi(X=g)$, la probabilité à priori. Cette mesure de probabilité est appelée probabilité *à priori*, car elle peut être calculée avant même de connaître l'image observée. La probabilité à priori est définie d'après nos attentes et nos connaissances de ce qu'est une « bonne » image. En général, nous nous attendons à ce qu'une bonne image soit plutôt uniforme et régulière. La définition de $\Pi(X=g)$ devrait donc favoriser ces images. Ainsi, une image g_1, plus lisse et plus régulière que g_2, devrait avoir une plus grande probabilité à priori : $\Pi(g_1) > \Pi(g_2)$.

Ensuite, il faut proposer un modèle pour $P(Y=y|X=g)$. Le modèle que nous proposons est celui-ci : plus les images y et g sont semblables, plus $P(Y=y|X=g)$ devrait être grande. On

aurait aussi pu définir $P(Y=y|X=g)$ telle qu'elle soit grande quand y semble suivre, s'il est connu, le modèle de dégradation appliqué à x. Toutefois, au cours de notre projet, nous avons supposé que le modèle de dégradation était toujours inconnu.

La forme gibbsienne

Dans une image, les pixels voisins ont souvent tendance à être de couleurs semblables. On peut donc poser l'hypothèse que l'intensité d'un pixel ne dépend que de ses voisins, selon le modèle de voisinage choisi. Nous prendrons cette hypothèse pour acquise.

Lorsque cette hypothèse est satisfaite et que toutes les configurations de pixels possibles de l'image ont une probabilité positive, on dit que l'image est modélisée par un champ de Markov, ce qui est le cas ici.

De plus, le théorème de Hammersley-Clifford[1] nous garantit que la distribution de probabilité d'une image modélisée par un champ de Markov est une distribution gibbsienne. Nous ne nous étendrons pas plus sur les fondements de ces résultats, car ils ne sont pas nécessaires à la suite du projet.

Donc, $P(X = g|Y=y)$ suit une distribution gibbsienne de même que $\Pi(X=g)$ et $P(Y=y|X=g)$. On peut donc toutes les écrire sous la forme gibbsienne :

$$\Pi(X=g) = Z_1^{-1} e^{-K(g)},$$

$$P(Y=y \mid X=g) = Z_2^{-1} e^{-D(g,y)}$$

$$\begin{aligned} \text{et donc, } P(X=g \mid Y=y) &= Z_3^{-1} e^{-K(g)} e^{-D(g,y)} \\ &= Z_3^{-1} e^{-K(g)-D(g,y)} \\ &= Z_3^{-1} e^{-H(g,y)} \end{aligned}$$

[1] Besag, J. (1974). Spatial Interaction and the Statistical Analysis of Lattice Systems. *Journal of the Royal Statistical Society : Series B (Methodological)*, **36**, 192-236.

où Z_1, Z_2 et Z_3 sont des constantes de normalisation et où $K(g)$, $D(g,y)$ et $H(g,y)$ sont les énergies présentées à la section précédente. Le lecteur averti avait probablement déjà remarqué que les expressions $\Pi(X=g)$ et $K(g)$ étaient liées, témoignant toutes les deux la régularité de l'image sans dépendre de l'image observée. De même, $P(Y=y|X=g)$ et $D(g,y)$ témoignent toutes deux de la ressemblance entre l'image proposée g et l'image observée y. Mais, après tout, c'est la probabilité à postériori, $P(X=g|Y=y)$, qui témoigne de la qualité de g étant donné l'image observée, car elle est associée à l'énergie totale H. Maintenant que la probabilité à postériori est exprimée sous la forme gibbsienne, il devient facile de travailler avec cette probabilité, car cela revient à travailler avec l'énergie totale.

Les estimateurs bayésiens

Une question persiste : quelle image retenir parmi toutes les images possibles pour x? En effet, la distribution à postériori nous permet d'associer à chaque image possible une probabilité que cette image soit l'image d'origine que nous cherchons étant donné l'image observée. Et donc, à partir de ces probabilités, quel estimateur choisir pour x?

Un premier estimateur, probablement le plus intuitif, est le MAP (maximal posterior estimate). L'estimateur MAP est tout simplement l'image ayant la plus grande probabilité à postériori. Autrement dit, l'estimateur MAP est l'image ayant la plus petite énergie globale H.

Un second estimateur est le MMSE (posterior minimum mean square estimate). Il consiste à effectuer la moyenne des images pondérées par leur probabilité à postériori. La moyenne entre des images est une image dont l'intensité d'un pixel est la moyenne des intensités du pixel des images dont on veut la moyenne, et ce pour tous les pixels. Si la moyenne n'est pas un nombre entier, elle est arrondie à l'entier le plus près.

D'autres estimateurs existent, mais ne seront pas traités dans ce rapport, notamment le MPME (marginal posterior mode estimate) et l'ICM (iterated conditional mode).

Reconstruction de l'image par les MCMC

Le problème de la minimisation

Au point où nous en sommes, toute la théorie nécessaire a été introduite pour être capable de conceptualiser ce qu'est la reconstruction d'image. Supposons que nous voulions reconstruire une image par le MAP, où il s'agit de choisir l'image avec l'énergie minimale. De prime abord, on pourrait penser à essayer toutes les images possibles pour choisir celle avec l'énergie minimale. Mais, ce n'est pas aussi simple. En effet, pour reconstruire une petite image en noir et blanc de 10 000 pixels seulement, le nombre d'image possibles à considérer s'élève à $2^{10\ 000}$, ce qui représente un nombre à plus de 3 010 chiffres! Lorsque l'image est plus grande, en niveau de gris ou en couleurs, le nombre de possibilités à considérer devient rapidement encore plus effrayant! Nous n'osons même pas imaginer le temps de calcul que pourrait prendre un ordinateur selon cette approche.

Les méthodes de Monte Carlo par chaîne de Markov (MCMC) constituent une bonne solution à ce problème. Elles permettent de générer un échantillon d'images selon la distribution de probabilité à postériori. Les images avec de petites énergies seront donc plus nombreuses dans l'échantillon, car elles ont une plus grande probabilité à postériori. Grâce aux MCMC, le nombre d'images à considérer n'est plus astronomique : seulement un échantillon représentatif de ces images sera utilisé.

Théorie sur les chaînes de Markov

Les MCMC sont basées sur la construction d'une chaîne de Markov. Voici la définition d'une chaîne de Markov dans le contexte de la reconstruction d'images :

Définition : Une chaîne de Markov est une suite d'images aléatoires, G_0, G_1, G_2 ,…, où chaque image ne dépend que de celle qui la précède dans la suite.

Notre but est de construire, à l'aide des MCMC, une chaîne de Markov dont la loi converge vers la distribution à postériori. Pour ce faire, notre chaîne de Markov a besoin de deux propriétés :

Propriété 1 : Irréductibilité

Une chaîne de Markov est irréductible s'il est possible d'aller de l'image *i* à l'image *j* en un certain nombre d'étapes, et ce pour toutes les images *i* et *j* possibles :

$$\exists n > 0 \text{ tel que } P(G_n = j \mid G_0 = i) > 0 \ \forall i, j$$

Propriété 2 : Apériodicité

Une chaîne de Markov est apériodique si l'on ne peut pas subdiviser les images possibles en ensembles qui seront visités de façon cyclique. Bref, la chaîne de Markov ne doit pas rester emprisonnée dans un cycle d'images à partir d'un certain moment. Mathématiquement, le plus grand commun diviseur de l'ensemble $\{n \geq 1; P(G_n = i \mid G_0 = i > 0\}$ doit égaler 1, et ce pour toutes les images i possibles.

La loi d'une chaîne de Markov possédant ces deux propriétés est assurée de converger vers une distribution limite. Mais une difficulté subsiste : comment construire une chaîne qui converge vers la distribution limite que nous voulons? En effet, il n'est pas commode, avec seulement les deux propriétés énoncées ci-haut, de trouver une chaîne de Markov convergeant vers la distribution désirée. Heureusement, il suffit de considérer une propriété supplémentaire : la réversibilité.

Propriété 3 : Réversibilité

Une chaîne de Markov est réversible par rapport à la distribution à postériori si la probabilité de passer de l'image *i* à l'image *j* en *n* étapes est la même que la probabilité de passer de l'image *j* à l'image *i* selon le même nombre d'étapes, pour tout $n>0$:

$$P(X = i \mid Y = y)P(G_n = j \mid G_0 = i) = P(X = j \mid Y = y)P(G_n = i \mid G_0 = j) \ \forall i, j \, .$$

Ensemble, ces trois propriétés nous permettent de construire une chaîne de Markov dont la loi converge vers la distribution que nous souhaitons : la distribution à postériori. La construction se fait en suivant les étapes d'un des algorithmes constituant les méthodes de Monte Carlo par chaînes de Markov.

L'algorithme de Metropolis-Hastings

Théorie

Un des algorithmes MCMC permettant de générer une chaîne de Markov convergeant vers la distribution à postériori est celui de Metropolis-Hastings. Une particularité très importante de l'algorithme de Metropolis-Hastings est qu'il faut choisir une distribution instrumentale, q. Cette distribution sert à modifier les pixels d'une image afin de la proposer pour qu'elle soit acceptée ou refusée dans l'échantillon. Cette distribution q peut dépendre de l'image précédente. Elle modifie les pixels d'une image indépendamment les uns des autres.

Pour débuter l'algorithme, il faut choisir une image de départ. Une des particularités des MCMC est que peu importe l'image de départ choisie, l'algorithme convergera éventuellement. Par contre, l'algorithme convergera plus rapidement si l'image de départ est choisie judicieusement de façon à avoir une grande probabilité à postériori, plutôt que de se retrouver dans les queues de la distribution à postériori. Au cours de notre projet, nous avons toujours utilisé l'image dégradée comme image de départ.

Pour commencer, l'algorithme parcourt les pixels de l'image de départ g_0 selon un trajet au choix. Les pixels peuvent être visités, par exemple, lignes par lignes ou bien par bonds de deux comme sur les cases d'un échiquier. Le trajet peut aussi être aléatoire. Pour chaque pixel visité, une nouvelle intensité suivant la distribution au choix q est émise. Lorsqu'autant de pixels ont été visités que le nombre de pixels de l'image, un premier cycle est complété et l'image générée est appelée g_1^*. On calcule alors la probabilité d'accepter cette image proposée dans notre échantillon:

$$\alpha(g_0, g_1^*) = \min\left(\frac{P(X = g_1^* \mid Y = y)Q(g_0 \mid g_1^*)}{P(X = g_0 \mid Y = y)Q(g_1^* \mid g_0)}, 1\right)$$

où $P(X=g_1^* \mid Y=y)$ et $P(X=g_0 \mid Y=y)$ sont les probabilités à postériori de l'image proposée et de l'image de départ. De plus, $Q(g_0 \mid g_1^*)$ et $Q(g_1^* \mid g_0)$ sont respectivement la probabilité de proposer g_0 si g_1^* est l'image actuelle dans la chaîne de Markov et la probabilité de proposer g_1^* si g_0 est l'image actuelle.

Si $\alpha(g_0,g_1^*)=1$, l'image proposée est acceptée automatiquement. Autrement, si $\alpha(g_0,g_1^*)<1$, on accepte l'image proposée avec probabilité $\alpha(g_0,g_1^*)$. Si l'image proposée est acceptée, $g_1=g_1^*$. Toutefois, si l'image proposée n'est pas acceptée, nous conservons l'image actuelle et $g_1=g_0$.

Le processus continue ainsi de façon à générer une suite d'image : la chaîne de Markov. À chaque itération, les pixels de l'image actuelle sont parcourus selon le trajet choisi et modifiés par q. Lorsqu'un cycle a été complété, on calcule

$$\alpha(g_k, g_{k+1}^*) = \min\left(\frac{P(X = g_{k+1}^* \mid Y = y)Q(g_k \mid g_{k+1}^*)}{P(X = g_k \mid Y = y)Q(g_{k+1}^* \mid g_k)}, 1\right)$$

et l'image proposée est encore acceptée avec probabilité $\alpha(g_k,g_{k+1}^*)$. En effet, nous accepterons parfois des images avec une probabilité à postériori inférieure à l'image actuelle. Mais, cela est une bonne chose, car l'algorithme évitera ainsi de rester pris dans des maximums locaux de probabilité. Si l'image proposée n'est pas acceptée, nous conservons l'image actuelle et $g_{k+1} = g_k$. Donc, à chaque cycle, on retient une image : soit l'image proposée, soit l'image actuelle. Cette image est ajoutée à la chaîne de Markov. Le processus est appliqué jusqu'à ce que le nombre d'images désiré ait été généré.

Pratique

Pour utiliser α en pratique, il faut expliciter son expression. D'abord, on peut exprimer les probabilités à postériori sous la forme gibbsienne :

$$\alpha(g_k, {g_{k+1}}^*) = \min\left(\frac{P(X = {g_{k+1}}^* \mid Y = y)Q(g_k \mid {g_{k+1}}^*)}{P(X = g_k \mid Y = y)Q({g_{k+1}}^* \mid g_k)}, 1\right)$$
$$= \min\left(\frac{Z^{-1}e^{-H({g_{k+1}}^*, y)}Q(g_k \mid {g_{k+1}}^*)}{Z^{-1}e^{-H(g_k, y)}Q({g_{k+1}}^* \mid g_k)}, 1\right)$$
$$= \min\left(\frac{e^{-H({g_{k+1}}^*, y)}Q(g_k \mid {g_{k+1}}^*)}{e^{-H(g_k, y)}Q({g_{k+1}}^* \mid g_k)}, 1\right).$$

Ensuite, on peut exprimer Q en fonction de q en gardant en tête que les valeurs d'intensité données par q aux différents pixels sont indépendantes :

$$\alpha(g_k, {g_{k+1}}^*) = \min\left(\frac{e^{-H({g_{k+1}}^*, y)}\prod_{s\in S} q\left((g_k)_s \mid ({g_{k+1}}^*)_s\right)}{e^{-H(g_k, y)}\prod_{s\in S} q\left(({g_{k+1}}^*)_s \mid (g_k)_s\right)}, 1\right)$$
$$= \min\left(\frac{e^{-H({g_{k+1}}^*, y)}}{e^{-H(g_k, y)}}\prod_{s\in S}\frac{q\left((g_k)_s \mid ({g_{k+1}}^*)_s\right)}{q\left(({g_{k+1}}^*)_s \mid (g_k)_s\right)}, 1\right).$$

Pour programmer l'algorithme, nous avons choisi d'utiliser q comme étant une distribution normale centrée à $\mu = (g_k)_s$ (s est le pixel actuel) et de variance σ^2. En pratique, on cherche à choisir q de façon à accepter les images proposées environ 25 % du temps. Ce pourcentage est un résultat théorique obtenu sur l'algorithme de Metropolis-Hastings pour les distributions instrumentales symétriques[2]. Les hypothèses d'application de ce résultat ne sont pas vérifiées ici et ne le sont pas non plus en général, mais ce résultat donne tout de même un ordre de grandeur pour le taux d'acceptation, ce qui est préférable que la méthode essais-erreurs. En effet, si les images générées sont trop semblables ou trop différentes de l'image qui les précède (taux d'acceptation trop près de 0 % ou de 100 %), l'algorithme mettra énormément de temps à converger. Dans notre cas où $q \sim N((g_k)_s, \sigma^2)$, la variance σ^2 est donc un paramètre à choisir de façon à ce que le taux d'acceptation soit d'environ 25 %.

Notons que dans le cas où q est une distribution symétrique, l'expression de α se simplifie:

[2] Roberts, G.O., Gelman, A. and Gilks, W.R. (1997). Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms. *Ann. Appl. Probab.* 7, p. 110-120.

$$\alpha(g_k, g_{k+1}^{*}) = \min\left(\frac{e^{-H(g_{k+1}^{*},y)}}{e^{-H(g_k,y)}}\prod_s 1,1\right)$$
$$= \min\left(\frac{e^{-H(g_{k+1}^{*},y)}}{e^{-H(g_k,y)}},1\right)$$
$$= \min\left(e^{-\left(H(g_{k+1}^{*},y)-H(g_{k-1},y)\right)},1\right).$$

Dans le cas où $\alpha(g_k,g_{k+1}^{*}) < 1$, il faut accepter l'image proposée avec probabilité $\alpha(g_k,g_{k+1}^{*})$. Pour ce faire, on génère une variable uniforme U_{k+1} entre 0 et 1. Si $U_{k+1} < \alpha(g_k,g_{k+1}^{*})$, alors l'image proposée est acceptée et $g_{k+1}= g_{k+1}^{*}$; autrement, elle est rejetée et $g_{k+1}= g_k$. Afin d'améliorer la précision des calculs, il est préférable de vérifier la condition $\ln(U_{k+1})<\ln(\alpha(g_k,g_{k+1}^{*}))$, ce qui revient à vérifier si $\ln(U_{k+1})<H(g_k,y)-H(g_{k+1}^{*},y)$ dans le cas où q est une distribution symétrique.

Voici les étapes de l'algorithme de Metropolis-Hastings :

1 . Choisir une distribution instrumentale facile à simuler par ordinateur : q.
Choisir une image de départ : g_0.
Choisir le nombre de cycles : n.
Choisir le trajet sur l'image.
Initialiser k=0, le numéro du cycle.

2. Pour chaque pixel de l'image g_k et en suivant le trajet sélectionné, générer un nouveau pixel $(g_{k+1}^{*})_s$ dont l'intensité provient de la distribution q. Ainsi, une nouvelle image g_{k+1}^{*} est proposée.

3. Calculer $\alpha = \min\left(e^{-\left(H(g_{k+1}^{*},y)-H(g_k,y)\right)}\prod_{s\in S}\frac{q\left((g_k)_s \mid (g_{k+1}^{*})_s\right)}{q\left((g_{k+1}^{*})_s \mid (g_k)_s\right)},1\right)$.

4. Si $\alpha(g_k,g_{k+1}^{*})=1$, $g_{k+1}=g_{k+1}^{*}$ et aller directement à l'étape 7.

5. Générer une variable aléatoire uniforme U_{k+1} entre 0 et 1.

6. Si $U_{k+1}<\alpha(g_k,g_{k+1}^*)$, $g_{k+1}=g_{k+1}^*$; sinon $g_{k+1}=g_k$.

7. $k = k+1$.

8. Si $k < n$, retourner à l'étape 2; sinon l'échantillon désiré a été généré et l'algorithme est terminé.

L'algorithme de Gibbs

L'algorithme de Gibbs est semblable à celui de Metropolis-Hastings avec comme différence majeure qu'il ne nécessite pas de test d'acceptation des images proposées. En commençant avec l'image de départ, les pixels sont parcourus les uns après les autres selon le trajet choisi (aléatoire ou déterminé). Lorsqu'un pixel est visité, son intensité est remplacée par une nouvelle intensité qui dépend uniquement du voisinage du pixel concerné. Pour que cette méthode soit valide, il faut proposer des nouvelles intensités en utilisant la distribution à postériori de l'intensité d'un pixel conditionnelle aux pixels voisins. Pour ce faire, on calcule l'énergie globale du voisinage du pixel visité, et ce pour chaque intensité que peut prendre ce pixel. Les énergies sont ensuite mises sous la forme gibbsienne et elles sont normalisées pour obtenir une probabilité. Ainsi, à chaque intensité possible du pixel visité, on associe une probabilité. Il s'agit de la probabilité de choisir cette intensité.

Pour le cas des images en noir et blanc par exemple, on a :

$$P(s=0 \mid G\backslash\{s\}=g_k\backslash\{s\}, Y=y)=\frac{e^{-H(g_k\backslash\{s\},y,s=0)}}{e^{-H(g_k\backslash\{s\},y,s=0)}+e^{-H(g_k\backslash\{s\},y,s=1)}}$$

$$\text{et } P(s=1 \mid G\backslash\{s\}=g_k\backslash\{s\}, Y=y)=\frac{e^{-H(g_k\backslash\{s\},y,s=1)}}{e^{-H(g_k\backslash\{s\},y,s=1)}+e^{-H(g_k\backslash\{s\},y,s=0)}}.$$

Il est à remarquer, avec ces formules, que nous n'avons besoin de trouver $H(g_k\backslash\{s\}, y, s=0)$ et $H(g_k\backslash\{s\}, y, s=1)$ qu'à une constante près seulement, car

$$P(s=0 \mid G\backslash\{s\}=g_k\backslash\{s\}, Y=y) = \frac{e^{-H(g_k\backslash\{s\},y,s=0)}}{e^{-H(g_k\backslash\{s\},y,s=0)}+e^{-H(g_k\backslash\{s\},y,s=1)}}$$
$$= \frac{e^{-H(g_k\backslash\{s\},y,s=0)}e^{C}}{e^{-H(g_k\backslash\{s\},y,s=0)}e^{C}+e^{-H(g_k\backslash\{s\},y,s=1)}e^{C}}$$
$$= \frac{e^{-H(g_k\backslash\{s\},y,s=0)+C}}{e^{-H(g_k\backslash\{s\},y,s=0)+C}+e^{-H(g_k\backslash\{s\},y,s=1)+C}}.$$

Or, peu importe la valeur de *s* (0 ou 1), la contribution à l'énergie globale *H* des pixels se trouvant à l'extérieur du voisinage de *s* est la même. Donc, les formules resteront vraies si les énergies sont calculées avec *s* fixe et en ne considérant que les pixels dans son voisinage. Cela apportera un gros avantage au niveau du temps de calcul.

Une fois $P(s=0 \mid G\backslash\{s\}=g_k\backslash\{s\}, Y=y)$ (ou $P(s=1 \mid G\backslash\{s\}=g_k\backslash\{s\}, Y=y)$) calculé, il suffit de tirer une variable aléatoire uniforme U_{k+1} entre 0 et 1 pour choisir la nouvelle intensité à donner au pixel. Si $U_{k+1} < P(s=0 \mid G\backslash\{s\}=g_k\backslash\{s\}, Y=y)$, alors une nouvelle image identique est adoptée à l'exception qu'elle prendra la valeur 0 à l'intensité du pixel *s*. Sinon, elle prendra la valeur 1.

Quand autant de pixels ont été visités que le nombre de pixels de l'image, un cycle est complété. La procédure est ensuite répétée pour le nombre de cycles voulu. Nous avons aussi choisi, pour l'algorithme de Gibbs, de débuter la chaîne avec l'image dégradée.

Voici les étapes de l'algorithme de Gibbs pour une image en noir et blanc :

1 . Choisir une image de départ : g_0.
Choisir le nombre de cycles : *n*.
Choisir le trajet sur l'image.
Initialiser *k*=0, le numéro de l'image proposée.
Initialiser *l*=0, le numéro du cycle.

2. Choisir un pixel de l'image g_k, suivant le trajet sélectionné.

3. Calculer $H(g_k \backslash \{s\}, y, s=0)$ et $H(g_k \backslash \{s\}, y, s=1)$.

4. Calculer $P(s=0 \mid G\backslash\{s\} = g_k \backslash \{s\}, Y = y) = \frac{e^{-H(g_k \backslash \{s\}, y, s=0)}}{e^{-H(g_k \backslash \{s\}, y, s=0)} + e^{-H(g_k \backslash \{s\}, y, s=1)}}$.

5. Générer une variable aléatoire uniforme U_{k+1} entre 0 et 1.

6. Si $U_{k+1} < P(s=0 \mid G\backslash\{s\} = g_k \backslash \{s\}, Y = y)$, alors $g_{k+1} = g_k \backslash \{s\}$ et s=0. Sinon, $g_{k+1} = g_k \backslash \{s\}$ et s=1.

7. k=k+1. Si un cycle a été complété aller à l'étape 8, sinon retourner à l'étape 2.

8. l=l+1.

9. Si $l < n$, retourner à l'étape 2; sinon l'algorithme est terminé et un échantillon contenant n images a été généré.

L'algorithme de Metropolis-within-Gibbs

Comme son nom l'indique, cet algorithme est une combinaison des deux algorithmes déjà présentés. Il s'agit en fait de l'algorithme de Gibbs, avec comme différence que la nouvelle intensité d'un pixel est proposée selon une distribution q (plutôt que choisie selon la distribution à postériori de l'intensité d'un pixel conditionnelle aux pixels voisins). Encore, ici, nous avons choisi d'utiliser $q \sim N((g_k)_s, \sigma^2)$. Comme nous ne générons pas la valeur d'intensité d'un pixel à l'aide de la distribution à postériori conditionnelle aux pixels voisins, il faut appliquer le test d'acceptation, comme dans l'algorithme de Metropolis-Hastings.

Voici les étapes de l'algorithme de Metropolis-within-Gibbs :

1 . Choisir une distribution instrumentale facile à simuler par ordinateur : q.
Choisir une image de départ : g_0.

Choisir le nombre de cycles : n.

Choisir le trajet sur l'image.

Initialiser k=0, le numéro du cycle

2. Choisir un pixel de l'image g_k, suivant le trajet sélectionné.

3. Générer un nouveau pixel $(g_{k+1}^{*})_s$ dont l'intensité provient de la distribution q.

4. Calculer $\alpha(g_k, g_{k+1}^{*}) = \min\left(e^{-\left(H(g_{k+1}^{*},y)-H(g_k,y)\right)} \prod_s \frac{q\left((g_k)_s \mid (g_{k+1}^{*})_s\right)}{q\left((g_{k+1}^{*})_s \mid (g_k)_s\right)}, 1 \right)$.

5. Générer une variable aléatoire uniforme U_{k+1} entre 0 et 1.

6. Si $U_{k+1}<\alpha(g_k,g_{k+1}^{*})$, $(g_{k+1})_s=(g_{k+1}^{*})_s$; sinon $(g_{k+1})_s=(g_k)_s$.

7. Retourner à l'étape 2 jusqu'à ce qu'un cycle ait été complété.

8. $k=k+1$.

9. Si $k<n$, retourner à l'étape 2; sinon l'algorithme est terminé.

Les estimateurs bayésiens plus en détails

MAP

Lorsque nous utilisons les MCMC, la loi de la chaîne de Markov que nous générons converge vers la distribution à postériori de l'image proposée étant donné l'image observée. Or, l'estimateur MAP requiert de trouver l'image avec la plus grande probabilité à postériori. Une première façon de faire pour trouver l'image avec la plus grande probabilité à postériori dans l'échantillon serait d'utiliser une variable initialisée à 0, appelons-la M. Après chaque nouvelle image générée, on vérifie si $H(g_{k+1},y)<H(g_k,y)$, et dans le cas où l'expression est vraie : $M=k+1$. Ainsi, à la fin de l'algorithme, la variable M contient le numéro de l'image avec la plus grande probabilité à postériori.

Toutefois, en pratique, nous utilisons une méthode plus efficace appelée « simulated annealing ». Elle consiste à introduire, peu importe l'algorithme MCMC choisi, une variable $\beta(k)$ dans l'expression de la probabilité à postériori, comme ceci :

$$P\left(X = g_k \mid Y = y\right) = Z^{-1} e^{-H(g_k,y)\beta(k)},$$

où k est le numéro du cycle.

La fonction β est initialisée à $\beta(0)=1$. De plus, elle est choisie croissante de sorte que les petites probabilités à postériori deviennent encore plus petites et les plus grandes probabilités deviennent encore plus grandes à mesure que l'algorithme avance. Si β est choisi de façon à croître suffisamment lentement, l'algorithme convergera vers l'image ayant la probabilité à postériori maximale (l'estimateur MAP). La méthode du « simulated annealing » est donc plus rapide et plus précise (dans la mesure où $\beta(k)$ est choisi judicieusement) que la première méthode proposée, car elle a l'avantage de converger directement vers l'image ayant la probabilité à postériori maximale.

Avec la méthode du « simulated annealing », dans le cas de l'algorithme de Metropolis-Hastings et de l'algorithme de Metropolis-within-Gibbs, l'expression de α devient :

$$\alpha(g_k, g_{k+1}^*) = \min\left(\left(\frac{e^{-H(g_{k+1}^*, y)}}{e^{-H(g_k, y)}}\right)^{\beta(k)} \frac{Q(g_k \mid g_{k+1}^*)}{Q(g_{k+1}^* \mid g_k)}, 1\right).$$

Ainsi, au fur et à mesure que l'algorithme avance, l'acceptation de l'image proposée devient de plus en plus stricte jusqu'à ce qu'à la limite, seules les meilleures images soient acceptées.

Dans le cas de l'algorithme de Gibbs, la croissance de β a pour effet de creuser l'écart entre les probabilités associées aux différentes intensités possibles pour un pixel donné, jusqu'à ce qu'à la limite, seules les meilleures intensités soient acceptées. L'expression de $P(s=0 \mid G\backslash\{s\} = g_k\backslash\{s\}, Y = y)$ et de $P(s=1 \mid G\backslash\{s\} = g_k\backslash\{s\}, Y = y)$ devient :

$$P(s=0 \mid G\backslash\{s\} = g_k\backslash\{s\}, Y = y) = \frac{e^{-H(g_k\backslash\{s\}, y, s=0)\beta(k)}}{e^{-H(g_k\backslash\{s\}, y, s=0)\beta(k)} + e^{-H(g_k\backslash\{s\}, y, s=1)\beta(k)}}$$

$$\text{et } P(s=1 \mid G\backslash\{s\} = g_k\backslash\{s\}, Y = y) = \frac{e^{-H(g_k\backslash\{s\}, y, s=1)\beta(k)}}{e^{-H(g_k\backslash\{s\}, y, s=1)\beta(k)} + e^{-H(g_k\backslash\{s\}, y, s=0)\beta(k)}}.$$

Pour une convergence rapide (mais pas nécessairement très bonne), on peut choisir $\beta(k)=k$ ou $\beta(k)=\varphi^k$, avec φ assez grand comme, par exemple, $\varphi=1.01$ ou $\varphi=1.05$. Pour une convergence assez lente, on peut choisir $\beta(k)= C \log(1+k)$ ou $\beta(k)= \varphi^k$, avec φ assez petit comme, par exemple, $\varphi=1.0005$.

MMSE

Le MMSE est un estimateur plus facile à comprendre en pratique. Le principe est simple : effectuer la moyenne des images pondérées par leur probabilité à postériori. Or, la loi de la chaîne de Markov converge vers la distribution à postériori. Il suffira donc d'effectuer la moyenne des images de la chaîne de Markov.

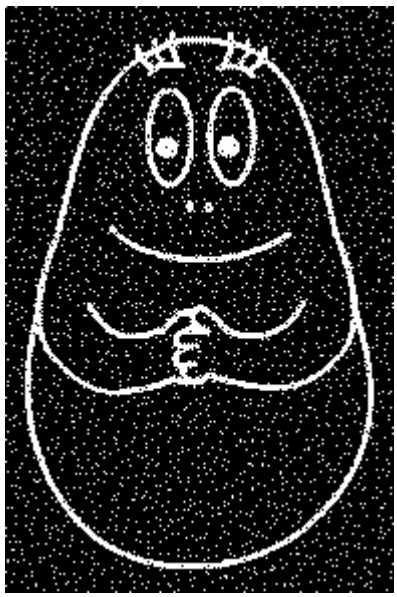
Toutefois, comme la loi de la chaîne de Markov converge vers la distribution à postériori, il est préférable de ne pas tenir compte des images générées trop tôt dans la suite. Le nombre d'images à rejeter est laissé au choix, car il s'agit d'un paramètre difficile à estimer. Une mauvaise estimation de ce paramètre ne devrait toutefois pas être très grave, car la moyenne devrait s'effectuer sur un très grand nombre d'images minimisant ainsi le poids des images moins favorables.

Les résultats obtenus

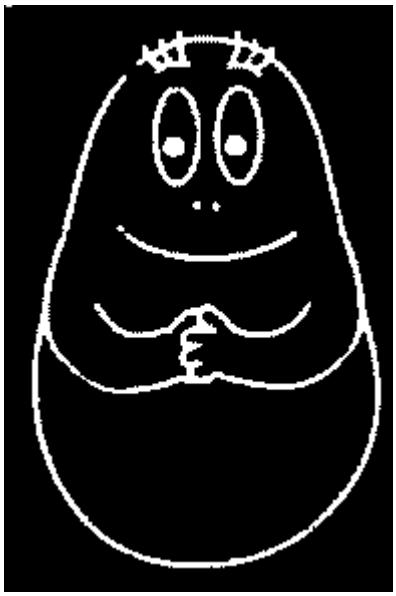
Images en noir et blanc

Image d'origine :

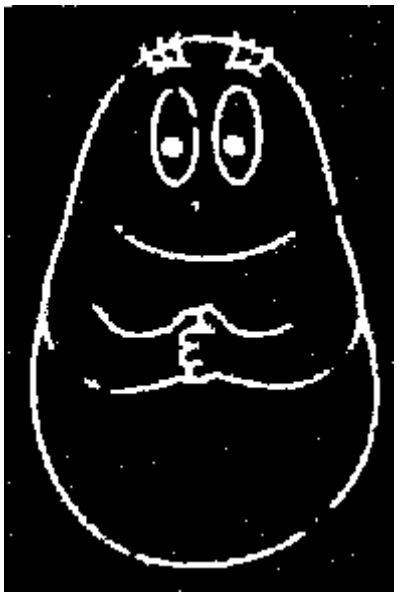
194 x 293 pixels



Images corrigées :



Algorithme : Gibbs
Estimateur : MMSE
λ=0.9
Nb cycles : 100
Burnin : 0
Temps : 5.54 s.



Algorithme : Gibbs
Estimateur : MAP
λ=1
$\beta(k)= \varphi^k$, avec φ=1.001
Nb cycles : 500
Temps : 25.54 s.

Images en niveaux de gris

Image d'origine :

147 x 218 pixels

Images corrigées :

Algorithme : Metropolis within Gibbs
Estimateur : MAP
λ=0.8
δ=65
α=3 000
$\beta(k)= \varphi^k$, avec φ=1.0005
Distribution inst. : $N(\mu,\sigma=5)$
Nb cycles : 12 000
Temps : 0 h 13 min.

Algorithme : Metropolis
Estimateur : MAP
λ=0.8
δ=65
α=3 000
$\beta(k)= \varphi^k$, avec φ=1.000014
Distribution inst. : $N(\mu,\sigma=0.18)$
Nb cycles : 500 000
Temps : 6 h 42 min.

Algorithme : Metropolis within Gibbs
Estimateur : MMSE
λ=0.8
δ=65
α=3 000
Distribution inst. : $N(\mu,\sigma=5)$
Nb cycles : 12 000
Burnin : 2000
Temps : 0 h 13 min.

Algorithme : Metropolis
Estimateur : MMSE
λ=0.8
δ=65
α=3 000
Distribution inst. : $N(\mu,\sigma=5)$
Nb cycles : 500 000
Burnin : 100 000
Temps : 6 h 48 min.

Images en couleur

Image d'origine :

88 x 106 pixels

Images corrigées :

Algorithme : Metropolis-within-Gibbs
Estimateur : MMSE
λ=0.8
δ=75
α=3 500
Distribution inst. : $N(\mu,\sigma=5)$
Nb cycles : 100 000
Burnin : 10 000
Temps : 1 h 25 min.

Algorithme : Metropolis
Estimateur : MMSE
λ=1
δ=65
α=3 000
Distribution inst. : $N(\mu,\sigma=0.26)$
Nb cycles : 450 000
Burnin : 75 000
Temps : 12 h 54 min.

Interprétation des résultats

D'après les résultats évoqués ci-haut et de nombreux autres résultats obtenus par d'autres essais au cours du projet, il nous est facile de nous prononcer quant à nos choix d'algorithmes et d'estimateurs préférés.

Pour les images en noir et blanc, l'algorithme de Gibbs a été le plus efficace, alors que pour les images en niveaux de gris et en couleurs, l'algorithme de Metropolis-within-Gibbs a donné les meilleurs résultats. En fait, on remarque que l'algorithme de Gibbs est à privilégier lorsque le nombre d'états possible que peut prendre un pixel est petit. En effet, lorsque ce nombre est petit, les probabilités conditionnelles se calculent rapidement et ce choix devient beaucoup plus avantageux par rapport à utiliser une distribution instrumentale comme dans les algorithmes de Metropolis et de Metropolis-within-Gibbs. Au contraire, lorsque le nombre d'états possibles est grand, le nombre de probabilités conditionnelles à calculer devient grand et il devient avantageux d'utiliser une distribution instrumentale.

Nous avons aussi remarqué qu'en général, l'algorithme de Metropolis-within-Gibbs fonctionne mieux que l'algorithme de Metropolis. En fait, l'avantage de l'algorithme de Metropolis-within-Gibbs est que le test d'acceptation est appliqué à chaque fois qu'on veut changer un pixel, alors que dans l'algorithme de Metropolis, le test d'acceptation est appliqué à chaque cycle. L'algorithme de Metropolis est donc beaucoup plus sensible au choix de la distribution instrumentale, car des paramètres mal choisis peuvent facilement conduire à un taux d'acceptation des images presque nul.

Pour ce qui est des estimateurs, nous avons préféré le MMSE au MAP parce que le MMSE est moins sensible au choix des paramètres. En effet, dans le cas du MAP, si β croît trop rapidement, la méthode du simulated annealing ne convergera pas vers l'estimateur MAP, car l'algorithme restera pris dans un minimum local d'énergie. De même, si β croît trop lentement, l'algorithme convergera, mais en beaucoup trop de temps. L'estimateur MMSE, pour sa part, ne nécessite comme paramètre que la période burn-in et donne de bons résultats même pour des estimations très différentes de la période burn-in.

Bibliographie

Besag, J. (1974). Spatial Interaction and the Statistical Analysis of Lattice Systems. *Journal of the Royal Statistical Society : Series B (Methodological)*, **36**, 192-236.

Geman, S et Geman D. (1984). Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Trans.* **PAMI-6**, 721-741.

Pollard, D. (2004). Hammersley-Clifford theorem for Markov random fields. *Handouts*. [http://www, star.yale.edu/ ~pollard/251.spring04/Handouts/Hammersley-Clifford.pdf]

Ross, S. M. (1997). *Simulation.* Academic Press, San Diego, 282 p.

Winkler, G. (2006). *Image Analysis, Random Fields and Markov Chain Monte Carlo Methods.* Springer, Berlin, 387 p.