Semaine d'Etude Mathématiques et Entreprises 6

Grenoble — 10 au 14 juin 2013

Autour d'une méthode de compression d'image respectant une information perceptuelle

Cyrille BEAUDRY - Amandine BERGER - Vincent CHABOT
- Franck GABRIEL - Kole KEITA

Sujet proposé par :

Encadré par Emmanuel Maitre et Valérie Perrier

Contact : c.beaudry32@gmail.com - amandine.berger@unine.ch - vincent2chabot@gmail.com -
franck.gabriel@normalesup.org - kole.keita@imag.fr

Table des matières

Résumé **3**

Remerciements **4**

1 Introduction **4**

2 Décomposition en clusters **5**
2.1 Clusters fixes et indépendants des valeurs 5
2.2 Clusters variables et indépendants des valeurs 6
2.3 Clusters dépendants des valeurs 6
2.4 Conclusion et perspectives concernant la décomposition en clusters 7

3 Méthodes d'approximation **7**
3.1 Base de départ 8
3.2 Approximation linéaire 9
3.3 Approximation quadratique 10
3.4 Tirer parti d'autres informations disponibles 13
3.4.1 Connaissance d'une borne inférieur et d'une borne supérieur 13
3.4.2 Connaissance de la croissance de la fonction 15
3.5 Conclusion et perspectives concernant l'approximation 17

4 Conclusion et perspectives **18**

Résumé

Il existe aujourd'hui de nombreux formats d'images. L'un des plus connus est la norme JPEG. Cette norme correspond à un algorithme de compression d'image précis. Bien que très répandue, notamment car efficace, cette norme ne permet pas d'obtenir de très bons résultats sur des éléments de texte.
Le but du sujet proposé par STMicroelectronics lors de la sixième semaine d'Études pour les Mathématiques en Entreprise (SEME) à Grenoble est de trouver une méthode de compression d'images alternative à la norme JPEG permettant d'obtenir des résultats plus fidèles pour les zones de texte.
Ce document constitue une synthèse des idées explorées ou non lors de cette semaine dans le cadre défini par les représentants de l'entreprise.

Remerciements

Nous tenons tout d'abord à remercier les organisateurs de cette SEME, ainsi que les différentes composantes de l'Université de Grenoble mises à contribution (prêts de locaux, matériels, personnel ...).
Nous remercions également l'entreprise STMicroelectronics, plus particulièrement Fritz Lebowsky et Marina Nicolas, pour nous avoir donné la possibilité de travailler sur ce sujet, et pour le temps qu'ils nous ont consacré.

1 Introduction

Une image consiste en un ensemble de pixels auxquels sont associées certaines valeurs (couleurs, luminosité ...). Pour stocker une image, il faut donc stocker de nombreuses données : pour chaque pixel, sa position ainsi que ses valeurs, mais aussi le nombre de pixels, la taille des pixels ... Le stockage d'images demande donc beaucoup de mémoire et la transmission, même avec des débits importants, peut-être lente.
Cependant, il n'est pas forcément nécessaire de conserver la totalité des informations. Selon l'utilisation de l'image, une version légèrement différente, mais stockée avec moins de données peut tout à fait convenir. Pour cela nous utilisons des algorithmes de compression qui vont permettre de stocker des images avec moins de données, et donc de permettre une transmission plus rapide, mais qui ne permettront pas de représenter exactement l'image initiale.
L'exemple le plus classique est celui de la norme JPEG. Néanmoins, le résultat obtenu par cette norme pour les zones de texte n'est pas de très bonne qualité. Nous souhaiterions donc trouver une méthode alternative donnant de meilleurs résultats pour les zones de texte.

Une première remarque évidente est que plus le nombre de paramètres utilisés sera grand, meilleure sera l'image restituée, mais également plus la mémoire nécessaire au stockage sera grande. Au contraire, plus le nombre de paramètres utilisés sera petit, plus la mémoire nécessaire sera petite, mais moins bonne sera la qualité de l'image restituée.
Il ne faut pas non plus oublier que dès qu'on commence à travailler sur l'image on fait des calculs et que ces calculs ont un coût.
Tout l'enjeu dans la création d'une norme d'image est de trouver un juste équilibre entre la quantité de mémoire nécessaire au stockage, le nombre de calculs à effectuer pour compresser et décompresser l'image et la qualité de l'image obtenue.

Fritz Lebowsky et Marina Nicolas ont commencé à développer une méthode qu'ils souhaiteraient améliorer si possible. Voyons en quoi consiste cette méthode.

1. Tout d'abord l'image est découpée en blocs de 4×4 pixels et le reste du travail s'effectuera sur chacun des blocs. Bien évidemment, il faudra penser à stocker la position des différents blocs.
2. Chaque bloc de 16 pixels est alors divisé en deux clusters différents de 8 éléments. La composition de ces clusters est indépendante des informations correspondant à chaque pixel. Il a été proposé de séparer les clusters en suivant un damier, des lignes horizontales ou des lignes verticales (voir Fig. 1 pour plus de détails). Si l'une de ces méthodes, et une seule, est choisie par défaut dans l'algorithme, alors seules quelques lignes dans l'algorithme seront nécessaires : pas de calculs particuliers, pas de méthode de séparation à stocker ...
3. On trie ensuite les valeurs de chaque cluster par ordre croissant. Nous obtenons donc un nuage de points que nous souhaitons approcher par une fonction monotone. Le but est de trouver une fonction nécessitant peu de paramètres permettant d'approcher suffisamment bien les valeurs pour obtenir des valeurs proches des valeurs initiales lors de la décompression. Notons qu'on souhaite approcher un nuage de 8 points.

La méthode retenue consiste à approcher ce nuage par deux segments de droites, la première commençant au point minimum et la seconde terminant par le point maximum, le point de changement de segments étant choisi afin de rendre l'erreur la plus faible possible. Dans ce cas, les points extremaux sont nécessaires, ainsi que les deux pentes et l'abscisse du point de changement de segment.

Partant de cette base de travail, le but est d'améliorer au moins l'un des critères suivants (en tentant de ne pas trop aggraver les autres) :
– rendu visuel ;
– place mémoire utilisée ;
– nombre d'opérations nécessaires.

Pour cela, nous pouvions :
– changer la taille des blocs, mais de préférence conserver des blocs de moins de 32 pixels ;
– changer la composition des clusters et éventuellement le nombre de clusters ;
– changer la méthode d'approximation ainsi que la méthode de calcul de l'erreur.

2 Décomposition en clusters

Dans cette section nous considérons que le décomposition en blocs a déjà été effectuée et nous travaillerons sur un bloc donné. Nous allons utiliser ici des blocs 4×4 pour les exemples mais tout peut être appliqué pour d'autres tailles de blocs.

Les décompositions en clusters introduites par les représentants présentent l'avantage de ne nécessiter aucun calcul et très peu de stockage. Par contre ils introduisent une géométrie non pré-existante a priori, susceptible de se retrouver dans le résultat obtenu. Pour éviter d'ajouter de telles géométries on peut par exemple envisager d'introduire de l'aléa dans le choix des clusters. Néanmoins, ces deux méthodes générales ne tiennent pas compte des valeurs alors que toutes sortes de nuages de points seront à approcher lors de la seconde phase. On peut donc essayer dès cette étape de tenir compte des valeurs afin de faciliter l'étape d'approximation.
Plusieurs points de vue, parfois contradictoires, sont à envisager.
Si on considère des zones homogènes, avec peu de variations des valeurs, il est évident que le nuage de point sera très regroupé et sera proche d'une droite. De plus, l'erreur possible sera plus faible car cela revient à faire un zoom. Regrouper les valeurs "proches" semble donc une bonne idée.
Au contraire, dans les zones de fortes variations, le nuage de points va risquer de faire apparaître des "plateaux", des escaliers. Si nous obtenons plus de deux de ces plateaux, alors approcher le nuage de points par deux segments de droites se révélera peu efficace. Ainsi, séparer les plateaux, et donc les valeurs proches, semble également être une idée raisonnable.

Ainsi lors de la définition des clusters, plusieurs possibilités sont à envisager : des clusters fixes et indépendants des valeurs, des clusters variables mais indépendants des valeurs ou bien des clusters dépendants des valeurs.

2.1 Clusters fixes et indépendants des valeurs

L'avantage des clusters fixes et non dépendants des valeurs est qu'il n'y a pas de calculs à faire lors de la composition des clusters et rien, ou presque, à stocker.
Les décompositions initialement proposées sont des exemples de décompositions en clusters fixes et indépendants des variables. On trouve en Fig. 1 quelques propositions de telles décompositions ainsi que des résultats obtenus pour ces décompositions.

Nous avons déjà évoqué les avantages, en terme de coût de stockage et de calcul, de ces méthodes. Néanmoins le résultat est fortement dépendant de l'image, et plus précisément de la zone de l'image, à compresser. Comme on peut le voir dans nos exemples, les zones de textes et les zones à fortes variations ne sont pas très bien représentées.
L'utilisation d'autres méthodes peut donc être utile pour de telles zones.

2.2 Clusters variables et indépendants des valeurs

La manière la plus simple de garder des clusters indépendants des valeurs mais ne faisant pas apparaître de géométrie est d'utiliser de l'aléa. Cet aléa peut-être utilisé pour choisir le nombre de clusters, le nombre d'éléments dans chaque cluster ou tout simplement pour déterminer à quel cluster appartiendra chaque pixel.

Quelques exemples de résultats peuvent être trouvés en Fig. 2.

Les avantages de cette méthode sont qu'aucune géométrie n'est a priori créée, même si cela reste une possibilité, et les valeurs ne jouent aucun rôle. Par contre on peut voir sur les exemples données en Fig. 2 que les résultats obtenus ne sont pas très bons.
De plus, cette méthode est très coûteuse. En effet, il faut tout d'abord effectuer un calcul afin de déterminer à quel cluster appartiendra chaque pixel. Ces choix seront à stocker pour la décompression. Si on choisis de traiter indépendamment chaque bloc, ces coûts sont à multiplier par le nombre de blocs.
Un autre inconvénient est que, comme les clusters sont remplis aléatoirement, à moins de se fixer à l'avance un nombre d'éléments par cluster, nous ne connaissons pas, a priori, la taille de chaque cluster ce qui rend difficile l'évaluation des coûts de tris et d'approximation.
Enfin il ne faut pas oublier de tenir compte du fait que dans ce cas toutes les tailles de clusters sont possibles. On peut donc obtenir des clusters vides ou ne contenant qu'un seul élément ... Pour empêcher de se retrouver dans des situations non voulues, il faut donc ajouter des tests à vérifier, ce qui augmente le nombre de calculs à effectuer.

En conclusion, même si l'utilisation d'aléa pouvait sembler être une bonne idée, les inconvénients paraissent bien plus nombreux et importants que les faibles avantages apportés.

2.3 Clusters dépendants des valeurs

Comme expliqué plus tôt, plusieurs approches peuvent être envisagées. Mais dans tous les cas, comme les clusters seront dépendants des valeurs dans chaque blocs, il sera nécessaire de stocker à quel cluster appartient chaque pixel du bloc, et donc de l'image.

Tout d'abord on peut vouloir faire des clusters par valeurs. Ainsi on peut décider de classer les pixels dans deux clusters différents selon que leurs valeurs sont inférieures ou supérieures à la moyenne ou bien la médiane ou encore la moitié de la différence entre le maximum et le minimum. De façon similaire on peut séparer les pixels en 3 clusters.
L'idée ici est que la différence entre le minimum et le maximum de chaque cluster est plus faible, ce qui devrait permettre de rendre l'erreur obtenue plus faible également.
Comme on peut le voir en Fig. 3, cette approche donne des résultats satisfaisants. Néanmoins, un certain nombre de calculs est nécessaire pour obtenir ces clusters. Il faut en effet déterminer la valeur qui va nous servir pour le classement. Pour cela il faut faire des calculs et/ou des tests sur toutes les valeurs. Une fois la valeur intermédiaire définie, il faut tester tous les pixels afin de déterminer à quel cluster il appartient.

On peut également vouloir regrouper les plateaux, c'est-à-dire regrouper les valeurs très proches.

On voit immédiatement arriver des problèmes. Tout d'abord, qu'entend-on par plateau ou par valeurs très proches ? Ensuite, selon ces définitions, il est possible d'obtenir beaucoup de clusters avec très peu de pixels. Pour de tels clusters, notre travail va revenir à stocker tous les pixels, ce qu'on ne souhaitait pas faire, mais en ayant fait des calculs en plus.
Il faut donc faire attention au résultat obtenu et ajouter des tests afin de s'assurer de ne pas obtenir un nombre de clusters trop grand ni des clusters avec trop peu de pixels. Si c'est le cas, alors utiliser une autre méthode pourrait être plus adapté.
Néanmoins tous ces tests ont des coûts qui s'ajoutent à ceux nécessaires à l'obtention des clusters. En effet, afin d'obtenir de tels clusters, une idée est de procéder par ordre croissant (ou décroissant). On considère le minimum et on ajoute dans le même cluster tous les pixels dont les valeurs sont suffisamment proches de ce minimum pour être dans son plateau. Une fois cela fait, on procède de même en partant du plus petit élément non trié ... On voit bien que le nombre d'opérations et de tests nécessaires est élevé.
Par contre, comme on peut le constater dans la Fig. 3, les résultats, notamment sur les zones de textes, sont plutôt satisfaisants.

Enfin, de façon totalement opposée, on peut vouloir séparer les différents plateaux. Dans ce cas, la quasi-totalité des remarques faites précédemment s'applique également. Nous avons les mêmes difficultés de définition, il y a autant de calculs à effectuer, de tests à faire, le nombre de clusters et d'éléments par clusters sont à vérifier ...
Toutefois, sur nos exemples, cette méthode semble donner de moins bons résultats, même si meilleurs qu'avec les méthodes indépendantes des valeurs.

2.4 Conclusion et perspectives concernant la décomposition en clusters

Nous avons vu que les méthodes indépendantes des valeurs ne donnent pas de très bons résultats, notamment concernant les zones de textes. Au contraire, les méthodes faisant intervenir les valeurs donnent de bons résultats mais sont beaucoup plus coûteuses en terme de mémoire et d'opérations.
La méthode à utiliser est donc fortement dépendante de l'image initiale (voire du bloc initial) et du résultat visuel souhaité. On peut donc envisager, pour une image donnée, de ne pas traiter tous les blocs avec la même méthode. Il n'est en effet pas nécessaire d'utiliser une méthode coûteuse si une méthode moins chère peut donner un résultat correspondant au résultat recherché.
Pour cela, il faudrait arriver à dégager des critères permettant de déterminer quelle méthode utiliser en fonction du bloc à traiter. Bien évidemment, pour l'utiliser dans la pratique, il faudra faire des tests supplémentaires.

Concernant les méthodes consistant à regrouper ou à séparer les plateaux, les limitations du nombre de clusters souhaités et du nombre minimal d'éléments par cluster acceptable sont importantes. Néanmoins, plus les blocs contiendront de pixels, plus le nombre de clusters acceptables sera grand, et moins il y aura de risque d'obtenir des clusters avec très peu de pixels.

3 Méthodes d'approximation

On considère maintenant un cluster, et plus précisément le nuage de ses valeurs classées par ordre croissant. Les valeurs de ces points sont notées

$$y_i = y(x_i) \quad \text{pour } i = 1, \dots, n. \tag{1}$$

La fonction la plus simple susceptible d'approcher ce nuage de points est sa moyenne :

$$\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i. \tag{2}$$

Cependant, sauf dans certains cas spécifiques, il est peu probable d'approcher correctement un nuage de points par sa moyenne.
Une approximation polynomiale d'ordre plus important est envisageable. De telles approximations ne permettent pas de représenter correctement une forte discontinuité au sein des données. Hors par exemple dans les zones de textes (qui nous intéressent particulièrement) il y a souvent la présence d'une forte discontinuité. Cette dernière est due au contraste entre le texte et le fond. Par conséquent, la méthode d'approximation proposée doit pouvoir gérer ce problème de discontinuité.

3.1 Base de départ

La méthode présentée en début de semaine par Fritz Lebowsky et Marina Nicolas consiste à approcher une courbes (de 8 points) par deux segments de droites. Cette approximation requiert 5 paramètres : le minimum (y_1), le maximum (y_8), deux pentes (a_1 et a_2) et la position d'un point d'inflexion (y_k avec $k \in \{2,\dots,7\}$). Les segments de droites sont définis :
– entre le premier point et le point d'inflexion. On a alors $a_1 = \frac{y_k - y_1}{k-1}$;
– entre le dernier point et le point suivant le point d'inflexion. Dans ce cas $a_2 = \frac{y_{k+1} - y_n}{n-k-1}$.
La valeur approchée de y_i est alors

$$y_i \approx f_k(x_i) = \begin{cases} f^1(x_i) = y_1 + (x_i - 1)a_1 \text{ si } i \leq k \\ f^1(x_i) = y_n + (n - x_i)a_2 \text{ si } i > k \end{cases} \tag{3}$$

Pour trouver le point d'inflexion k, la méthode employée consiste à tester les 6 points possibles ($k \in \{2,\dots,7\}$) et de choisir celui minimisant l'erreur entre l'approximation effectuée et la vérité :

$$k = \underset{k}{\operatorname{argmin}} \left(\sum_{i=1}^{n} |f_k(x_i) - y_i| \right) \tag{4}$$

Une telle approche est résumée en figure 4.

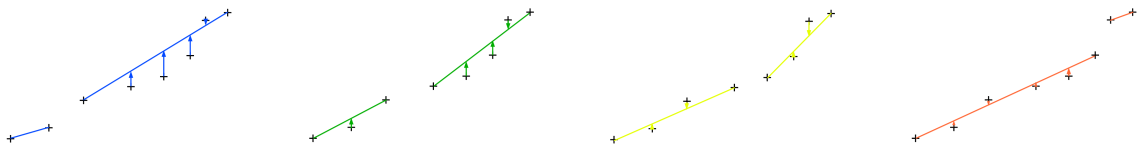

Figure 4 – Méthode d'approximation par deux segments de droites. On considère les différents cas possibles et on choisit le cas minimisant l'erreur.

Afin de mieux s'apercevoir de l'effet des différentes approches proposées, dans cette partie nous travaillons sur des jeux de 16 points et non de 8. Dans le but de comparer les différentes techniques d'approximation, les erreurs sont calculées en norme L_2 par

$$\|f_k - y\|_2 = \sqrt{\sum_{i=1}^{n} (f_k(x_i) - y_i)^2} \tag{5}$$

et en norme L_1 par

$$\|f_k - y\|_1 = \sum_{i=1}^{N} |f_k(x_i) - y_i| \tag{6}$$

La figure 5 présente les résultats obtenus, avec la méthode d'approximation par deux segments, sur deux jeux de points qui serviront d'exemples tout au long de cette partie.

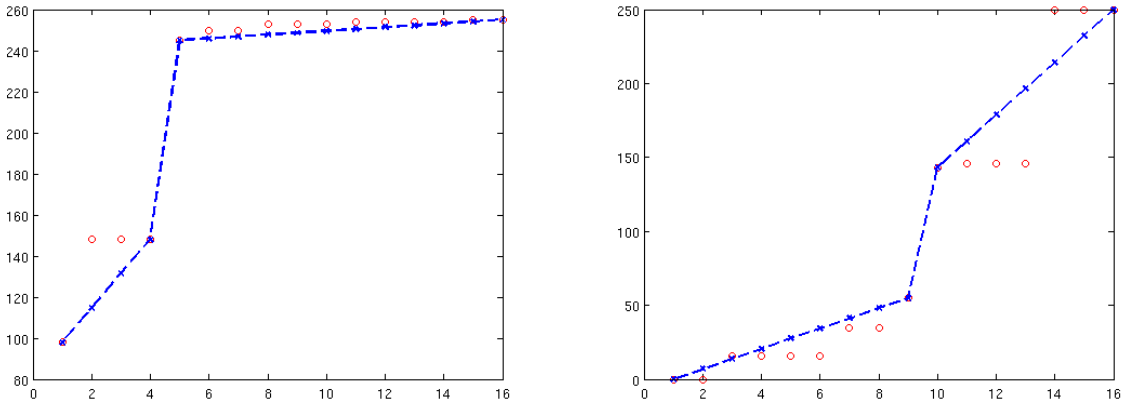

Figure 5 – Dans la suite nous considérerons principalement l'approximation de deux jeux de 16 points différents. L'approximation (en bleu) des points (en rouge) fournie par la méthode de départ est présentée. Pour la figure de gauche l'erreur, en norme L_2 entre la vérité et l'approximation est de 38.71. Pour la figure de droite, l'erreur en norme L_2 est de 78.63. Le point d'inflexion est $k = 4$ (resp. $k = 9$) pour la figure de gauche (resp. droite).

3.2 Approximation linéaire

Au lieu de chercher à approcher la courbe par deux segments (passant par les extrémités), on peut chercher à l'approcher par deux droites :

$$y_i \approx f_k(x_i) = \begin{cases} f^1(x_i) = a_1 x_i + b_1 \text{ si } i \leq k \\ f^2(x_i) = a_2 x_i + b_2 \text{ si } i > k \end{cases} \tag{7}$$

Dans ce cas le nombre de paramètres à stocker reste inchangé (5). On cherche à résoudre le problème suivant :

$$\begin{cases} \text{Trouver } k, (a_1, b_1), (a_2, b_2) \text{ minimisant } \left(\sum_{i=1}^{n} (f_k(x_i) - y_i)^2 \right) \\ \text{où} \\ (a_1, b_1) = \underset{a,b}{\operatorname{argmin}} \left(\sum_{i=1}^{k} (y_i - a x_i - b)^2 \right) \\ (a_2, b_2) = \underset{a,b}{\operatorname{argmin}} \left(\sum_{i=k+1}^{n} (y_i - a x_i - b)^2 \right) \end{cases} \tag{8}$$

L'approximation linéaire obtenue pour les deux jeux de données précédents est présentée en figure 6. Le fait de ne pas imposer les extrémités des droites laisse plus de liberté. Ainsi, bien que le point d'inflexion trouvé soit le même que dans la figure 5, l'erreur résultant d'une approximation linéaire est moins importante.

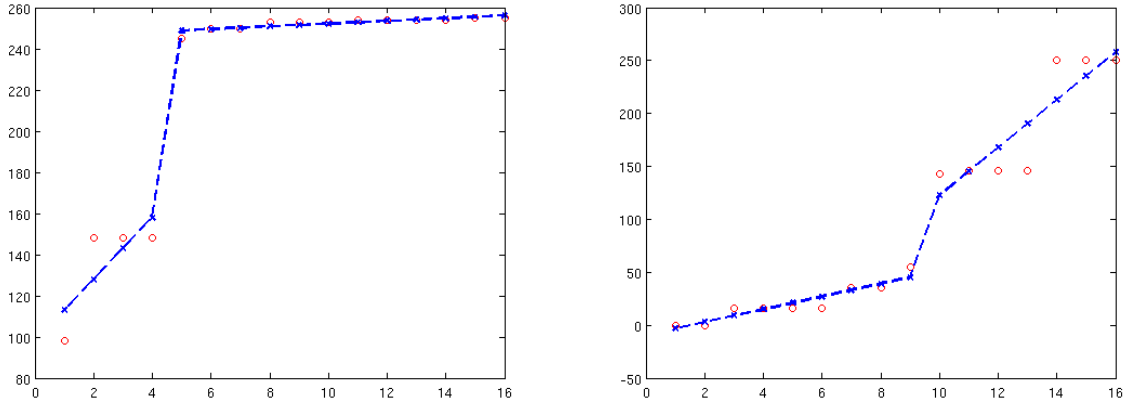

Figure 6 – Approximation linéaire des même jeux de points qu'en figure 5. La norme L_2 entre la vérité et l'approximation est de 27.9 pour le jeu de données de gauche et de 69.6 pour le jeu de données de droite.

3.3 Approximation quadratique

Après avoir étudié une classe de fonctions d'approximation linéaires, nous avons envisagé chercher une approximation quadratique. L'idée de cette partie est de trouver une fonction quadratique ayant moins de paramètres à stocker, pouvant passer au plus près des points au sens des moindres carrés pour les blocs de l'image. Le choix des types de fonctions quadratiques est porté sur un polynôme de second degré au lieu d'un monôme dont le nombre de paramètres à stocker est 1. Nous devons trouver un polynôme de la forme $ax^2 + bx + c$. Cela revient à optimiser trois coefficients a, b et c pour que la norme euclidienne de l'écart entre la courbe passant par les points du bloc et le polynôme soit minimal ou encore déterminer les coefficients permettant aux valeurs de la fonctionnelle quadratique d'être aussi proches des amplitudes des pixels.
L'écart à minimiser peut s'écrire sous la forme suivante :

$$E(a,b,c) = \sum_{j=1}^{N} (ax_j^2 + bx_j + c - y_j)^2 = \sum_{j=1}^{N} E_j^2$$

où N est le nombre de pixels, x_j l'abscisse du pixel j après le tri, y_j son amplitude et a, b et c sont les coefficients du polynôme de second degré à déterminer.
Pour déterminer les valeurs optimales de ces coefficients, nous nous sommes intéressés tout d'abord aux 16 pixels d'un bloc de 4x4 quelconque et ensuite, dans le but de diminuer les erreurs, nous avons cherché une manière de séparer les pixels de façon optimale.

Polynôme associé aux 16 pixels

Cherchons à déterminer les coefficients a, b et c pour que $E(a,b,c)$ donné plus haut soit minimal. C'est aussi la résolution d'un système d'équations rectangulaire car le nombre d'inconnues est inférieur au nombre d'équations. La méthode utilisée est la méthode des moindres carrés qui permet de déterminer les valeurs optimales des coefficients en calculant les zéros du gradient de $E(a,b,c)$ par rapport aux coefficients (a,b,c). Elle va nous permettre de trouver les valeurs optimales a^*, b^* et c^* telles que $E(a^*,b^*,c^*) = \min E(a,b,c)$.
Puisque le système à résoudre est linéaire, il est possible de l'écrire simplement sous une forme matricielle, qui dans notre cas peut s'écrire sous la forme suivante :

$$M \begin{pmatrix} a \\ b \\ c \end{pmatrix} = Y,$$

avec $M = \begin{pmatrix} x_1^2 & x_1 & 1 \\ . & . & . \\ . & . & . \\ . & . & . \\ x_N^2 & x_N & 1 \end{pmatrix}$ la matrice de Vandermonde et

$$Y = \begin{pmatrix} y_1 \\ . \\ . \\ . \\ y_N \end{pmatrix}.$$

La fonctionnelle $E(a, b, c)$ peut être réécrite tout simplement comme $I(X) = ||MX - Y||^2$. Pour déterminer le gradient, une manière efficace est de procéder avec la formule de Taylor. Cette formule nous énonce que $\forall h \in \mathbb{R}^3$,

$$I(X + h) = I(X) + h\nabla I(X) + O(h)$$

donc

$$\begin{aligned} I(X+h) &= ||MX + Mh - Y||^2 =< MX + Mh - Y, MX + Mh - Y > \\ &= < MX - Y, MX - Y > +2 < Mh, MX + Mh - Y > +O(h) \end{aligned}$$

alors on peut dire que

$$h\nabla I(X) = 2 < Mh, MX + Mh - Y >$$
$$\Rightarrow \nabla I(X) = 2M^T MX - 2M^T Y = 0.$$

La valeur optimale est $X^* = (a^*, b^*, c^*)^t = (M^T M)^{-1} M^T Y$.
On peut montrer que la matrice $M^T M$ est de rang plein ou égal à 3 à partir des vecteurs colonnes indépendants de M, alors $M^T M$ est bien inversible.
Dans les codes de calcul de X^*, plusieurs choix se présentent, différents par rapport au nombre de paramètres à stocker. Nous nous sommes intéressés à deux de ces choix dont le premier consiste à calculer la fonction quadratique obtenue à partir des paramètres optimaux sans lui exiger de passer par aucun point mais par contre le second choix exige le passage de la fonctionnelle par les deux points aux extrémités. Le second choix permet de se retrouver avec deux paramètres supplémentaires à stocker dont les deux points aux extrémités.
La norme euclidienne des erreurs commises lors de ces deux choix ou la racine carrée de $E(a, b, c)$ sont fournies dans le tableau 1.

Choix de passage par les extrémités	Rouge	Verte	Bleue
oui	82.6906	69.3438	69.8666
non	87.2570	71.8331	75.3877

Table 1 – La racine carrée de E(a,b,c) pour les deux choix

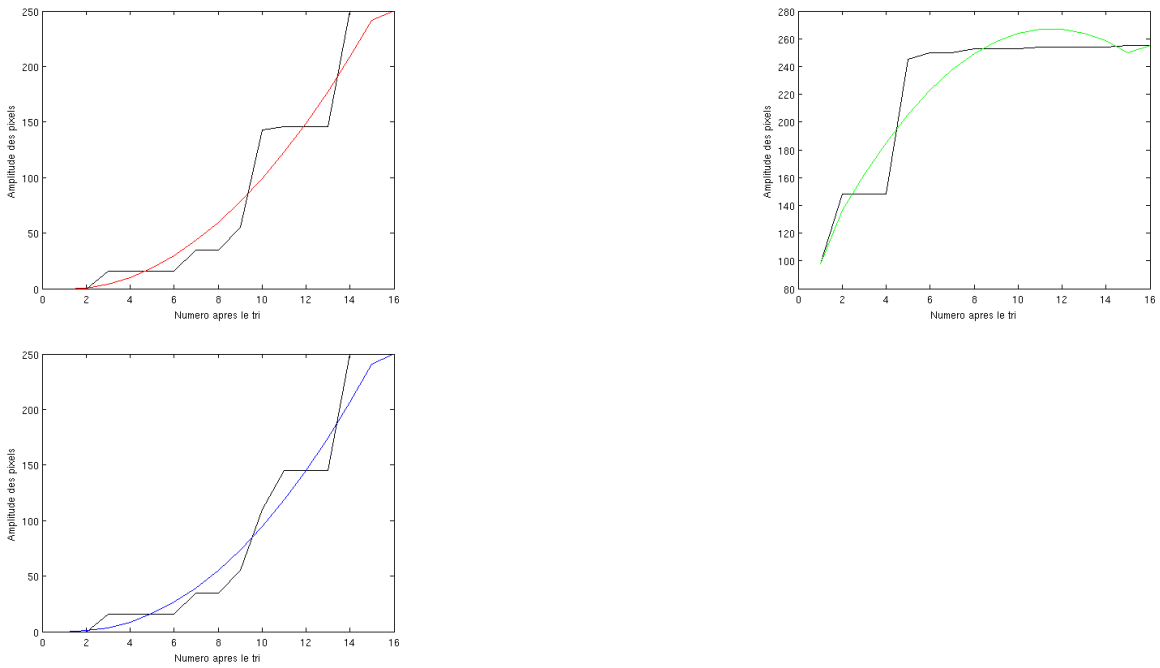

Figure 7 – Fonctions d'approximation quadratique des canaux rouge, vert et bleu : chaque couleur correspond à celle du canal

Les valeurs dans le tableau montrent que les erreurs diminuent au prix de deux paramètres de stockage supplémentaires dont les amplitudes des deux extrémités.
On peut aussi exiger le passage de la fonctionnelle par les deux extrémités avec seulement trois paramètres à stocker et cela se fait en ajoutant aux systèmes deux contraintes telles que $ax_1^2 + bx_j + c = 0$ et $ax_N^2 + bx_N + c = 0$. Ce genre de problème ne donne pas toujours les bonnes valeurs optimales.

Polynômes associés à deux groupes pixels

Pour pouvoir réduire les erreurs commises par la méthode précédente, nous envisageons de n'approcher que les points qui sont voisins par une fonction quadratique. Dans ce cas, la technique utilisée est de former deux groupes dont chaque groupe contient les éléments proches les uns des autres. Dans ce cas, il faut créer une fonction permettant de distribuer les autres points dans deux groupes formés au départ par les points aux extrémités (amplitude max et min). Le nombre de paramètres à stocker pour cette méthode est 7 dont les trois coefficients des deux polynômes et le point d'inflexion pour lier les deux fonctions. Bref la fonction d'approximation est en quelque sorte deux fonctions quadratiques liées par une droite. Avec le même bloc que précédemment, on obtient les courbes de la figure 8.

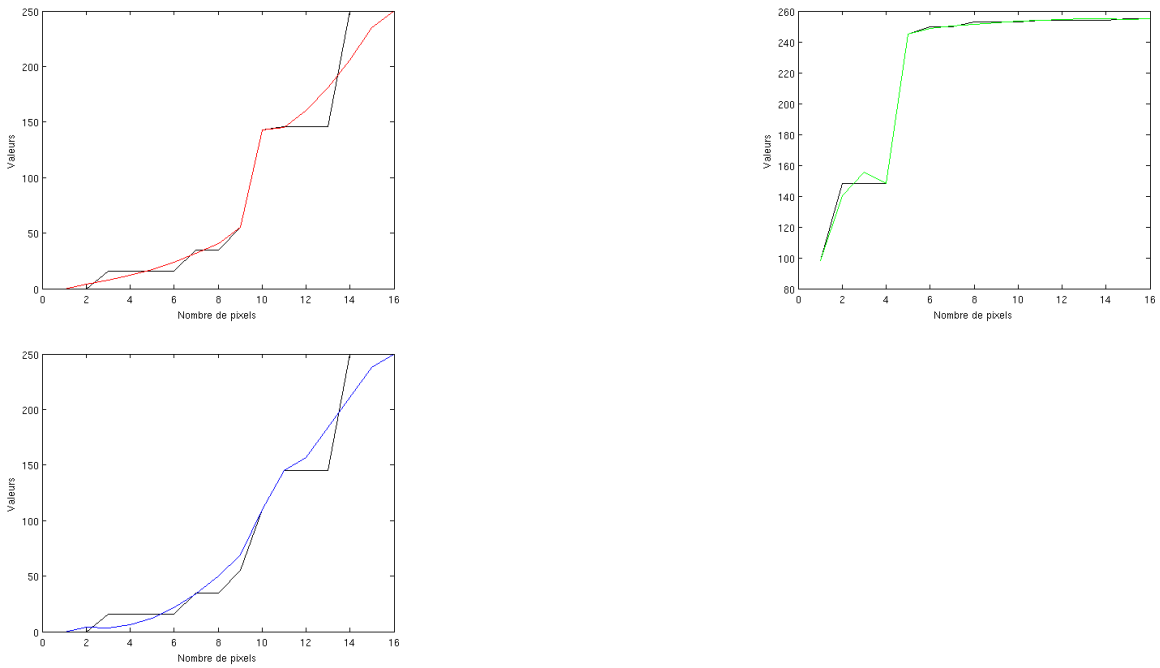

Figure 8 – Fonctions d'approximation quadratique des canaux rouge, vert et bleu : chaque couleur correspond à celle du canal.

Les erreurs commises pour cette méthode sont données dans le tableau suivant :

Couleur	Rouge	Verte	Bleue
Erreur	61.7567	10.9166	63.7112

Les erreurs ont diminué pour un prix de 4 paramètres supplémentaires. Dans certains cas, la différence d'erreur entre les deux cas n'est pas aussi importante.

3.4 Tirer parti d'autres informations disponibles

On sait que la fonction recherchée est croissante (les pixels sont triés par ordre croissants) et bornée (par exemple par 0 et 255 dans le cas de valeurs RGB). Ces informations peuvent être :
- perçues comme des contraintes. Dans ce cas on recherche une fonction satisfaisant ces conditions (par exemple l'approximation par deux segments présentés dans la partie précédente) ;
- utilisées lors de la reconstruction afin d'avoir une approximation pertinente (par exemple en bornant les valeurs possibles en chaque point) ;
- utilisées lors de la phase de compression si l'on connaît les règles appliquées lors de la reconstruction afin d'avoir une approximation respectant les informations que l'on connaît.

3.4.1 Connaissance d'une borne inférieur et d'une borne supérieur

On donne ici l'exemple du cas RGB. Ainsi les valeurs sont comprises entre 0 et 255.
Nommons

$$f_{min} = \min_{x_i}(f_k(x_i) > 0) \tag{9}$$

et

$$f_{max} = \max_{x_i}(f_k(x_i) < 255) \tag{10}$$

le minimum et maximum de la fonction f_k compris dans l'intervalle $[0; 255]$. Nommons les valeurs de x pour lesquelles le minimum et le maximum sont atteints x_{min} et x_{max}. Lors de la recon-

struction, on va borner la fonction f_k :

$$f_k^{recon}(x_i) = \begin{cases} 0 & \text{si } x_i < x_{min} \\ f_k(x_i) & \text{si } x_{min} \leq i \leq x_{max} \\ 255 & \text{si } x_i > x_{max} \end{cases} \tag{11}$$

Cette opération obligeant la fonction reconstruite à être comprise dans un intervalle n'apporte que très peu de modifications si elle est uniquement vue comme un post-traitement. Cependant, si l'on sait que cette opération va être effectuée lors de la reconstruction, on peut le prendre en compte (de différentes manières) lorsqu'on définit le problème de compression (8).

La méthode employée ici pour tenir compte de cette information lors de la recherche de la meilleure approximation consiste à effectuer une régression linéaire sans prendre en compte une partie des points proche des limites, puis de reconstruire ces points par la fonction f_k^{recon} définie en équation (11).

Soit k un point d'inflexion fixé. On peut tester de manière récursive l'approximation obtenue lorsqu'on ne tient compte que de certains points pour rechercher l'approximation linéaire.
Les paramètres (a_1, b_1) et (a_2, b_2), utilisés dans f^1 et f^2, ne sont dès lors définis que pour certains points :

$$(a_1, b_1) = \underset{a,b}{\operatorname{argmin}} \left(\sum_{i=r_{min}}^{k} (y_i - ax_i - b)^2 \right)$$

et

$$(a_2, b_2) = \underset{a,b}{\operatorname{argmin}} \left(\sum_{i=k+1}^{r_{max}} (y_i - ax_i - b)^2 \right)$$

où r_{min} et r_{max} contrôlent le nombre de points pour lesquels les approximations f^1 et f^2 doivent être optimales. Lorsque $r_{min} = 1$ et $r_{max} = n$ on retrouve, pour chaque point k, les couples (a_1, b_1) et (a_2, b_2) du problème (8).

La fonction reconstruite est :

$$f_{k,r_{min},r_{max}}^{recon}(x_i) = \begin{cases} f^{1,bord}(x_i) = \max(0, f^1(x_i)) & \text{si } i \leq k \\ f^{2,bord}(x_i) = \min(255, f^2(x_i)) & \text{si } i > k \end{cases} \tag{12}$$

Le problème à résoudre devient :

$$\begin{cases} \text{Trouver } k,\ (a_1, b_1),\ (a_2, b_2),\ r_{min},\ r_{max} \text{ minimisant} \\ \left(\sum_{i=1}^{n} (f_{k,r_{min},r_{max}}^{recon}(x_i) - y_i)^2 \right) \end{cases} \tag{13}$$

Si on fixe $r_{min} = 1$ et $r_{max} = n$ on retrouve le problème (8).

Cette méthode requiert un nombre d'évaluations de la fonction coût important (pour chaque point d'inflexion k possible, il faut tester différentes valeurs de r_{min} et r_{max}). Si le temps passé à compresser l'information n'est pas une contrainte forte, on peut envisager d'utiliser directement cette technique. Sinon il est certainement possible de trouver des méthodes plus efficace utilisant cette idée (voir remarque 3).

La reconstruction effectuée en équation (11) n'est pas l'unique solution : on peut envisager de faire une interpolation linéaire entre le dernier point à l'intérieur de l'intervalle et les bornes de

celui-ci :

$$f_k^{recon}(x_i) = \begin{cases} \dfrac{x_i - 1}{x_{min} - 1} f_{min} & \text{si } x_i < x_{min} \\ f_k(x_i) & \text{si } 0 \leq f(x) \leq 255 \\ f_{max} + \dfrac{x_i - x_{max}}{n - x_{max}}(255 - f_{max}) & \text{si } x_i > x_{max} \end{cases} \tag{14}$$

De la même manière que précédemment on peut effectuer des recherches avec cette fonction. La figure 9 présente le résultat d'une telle minimisation utilisant la fonction de reconstruction présentée en équation (14). On s'aperçoit que la reconstruction est meilleure lorsqu'on utilise cette information dès la phase de compression. À noter qu'en incorporant de nouvelles informatiosn, le point d'inflexion k est changé.

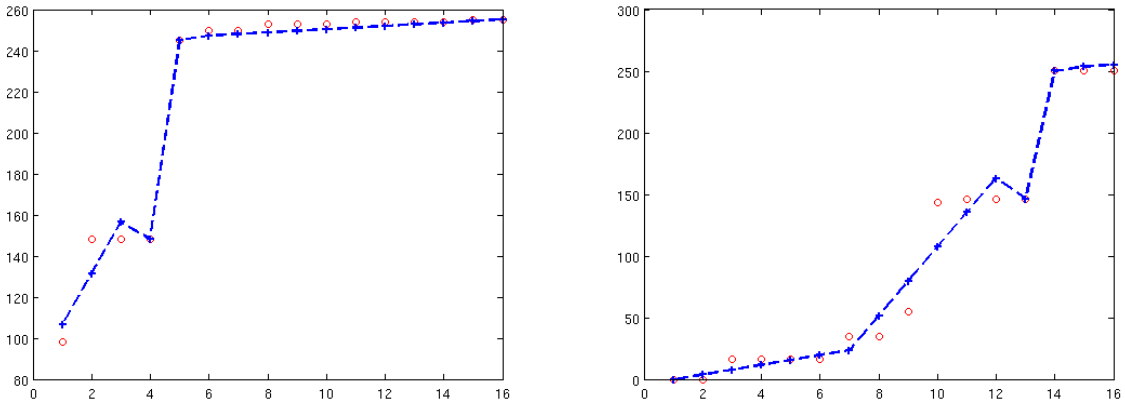

Figure 9 – Dans cet exemple, on prend en compte lors de la recherche de la courbe linéaire par morceaux que la fonction est bornée. Cependant on ne prend pas en compte lors de l'approximation et de la reconstruction qu'elle doit être croissante. La norme L_2 entre la vérité et l'approximation est de 21.8 pour le jeu de données de gauche et de 53.2 pour le jeu de données de droite. Le point d'inflexion pour la figure de gauche (resp. de droite) est $k = 3$ (resp. $k = 12$).

Remarque 1 *On peut reconstruire de différentes manières les valeurs de la fonction f_k hors des bornes. Il est donc nécessaire, lorsqu'on effectue la minimisation, de savoir quelle fonction de reconstruction est utilisée lors de la recherche du minimum.*

3.4.2 Connaissance de la croissance de la fonction

On sait que la fonction est croissante. On peut se servir de cette information dans de nombreux cas lors de la reconstruction lorsque l'approximation est effectué par des polynômes de degrés supérieurs à 1 ou par morceaux. Dans le cas de l'utilisation de deux approximations linéaires, cela peut être fait de plusieurs manières. On peut par exemple définir que le minimum obtenu par f^2 ne peut être dépassé en aucun point de f^1. Ceci conduit à définir la fonction de reconstruction suivante :

$$f_k^{recon}(x_i) \begin{cases} f^1(x_i) & \text{si } f^1(x_i) \leq \min\limits_{x=\{k\ldots n\}}(f^2(x)) \\ \min\limits_{x=\{k\ldots n\}}(f^2(x)) & \text{si } f^1(x_i) > \min\limits_{x=\{k\ldots n\}}(f^2(x)) \\ f^2(x_i) & \text{si } i > k \end{cases} \tag{15}$$

Un telle fonction de reconstruction a été appliquée à l'approximation donnée en figure 9 lors de la reconstruction. Le résultat est présenté en figure 10. On s'aperçoit que contrairement au cas précédent on a bien reconstruit une fonction monotone. L'erreur est quelque peu réduite lors de la phase de reconstruction mais le changement n'est pas très important.

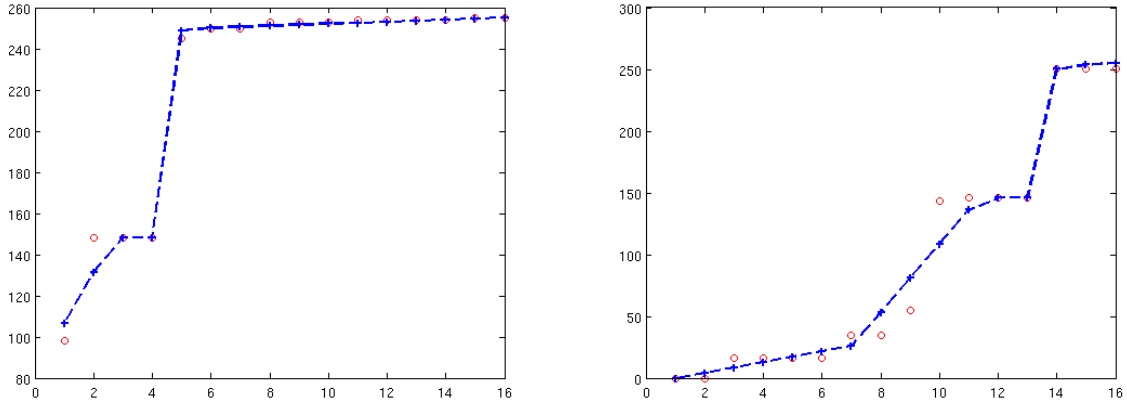

Figure 10 – Dans cet exemple, on prend en compte lors de la recherche de la courbe linéaire par morceaux que la fonction est bornée. On prend en compte lors de la reconstruction qu'elle doit être croissante. La norme L_2 entre la vérité et l'approximation est de 20.2 pour le jeu de données de gauche et de 50.3 pour le jeu de données de droite.

Remarque 2 *On peut très bien choisir une autre fonction d'approximation entre les deux courbes. Le tout est, comme dans le cas précédent, de définir quelle est la fonction utilisée pour la reconstruction avant d'effectuer la minimisation.*

A première vue, il n'y a pas de solution aisée pour utiliser cette information lors de la phase de compression. Une solution consiste, comme dans les cas précédents, à tester un grand nombre de possibilités et de voir celle qui convient le mieux. Un exemple de résultats obtenus de cette manière est présenté en figure 11. Dans cet exemple, on s'aperçoit que l'erreur est sensiblement diminuée.

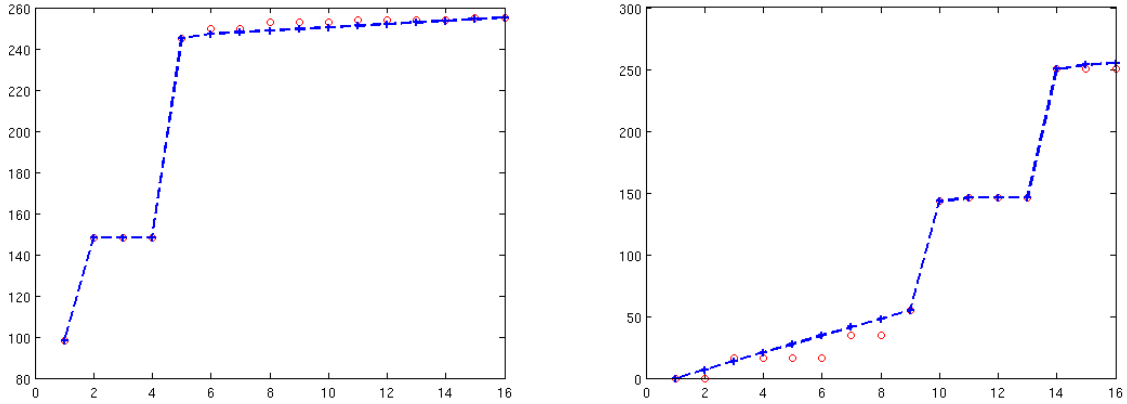

Figure 11 – Dans cet exemple, on prend en compte lors de la recherche de la courbe linéaire par morceaux que la fonction est bornée et croissante. La fonction de reconstruction pour les points hors des bornes est la fonction linéaire de l'équation (14). La norme L_2 entre la vérité et l'approximation est de 7.8 pour le jeu de données de gauche et de 28.2 pour le jeu de données de droite.

La figure 12 présente une explication plus détaillé de la manière dont l'information sur la monotonie et les bornes a été incorporée. En vert sont représentés les points reconstruits grâce aux fonctions d'approximations f^1 et f^2. Ces dernières sont représentées en violet. Les points reconstruits grâce aux autres informations sont présentés en bleu. Entre les courbes f^1 et f^2 l'information utilisée est la monotonie de la fonction. À droite du dernier point vert, l'information utilisée est la borne supérieur de la fonction. On voit cependant sur cet exemple qu'il nous

	Premier set		Second set	
	$\|.\|_2$	$\|.\|_1$	$\|.\|_2$	$\|.\|_1$
Figure 5	38.7	80	78.6	214.5
Figure 6	27.8	63.4	69.6	193.4
Figure 9	21.8	55.6	53.2	144.8
Figure 10	20.2	47.3	50.3	128.0
Figure 11	7.8	22.3	28.2	71.8

Table 2 – Résumé de l'erreur en norme L_2 et L_1 effectuée lors de l'approximation par les différentes fonctions testées. Le nombre de points est 16. Dans le meilleur des cas, on a fait une erreur moyenne de 1.4 unité (contre 5 pour l'approximation par la méthode des deux segments).

reste quelques marges de manœuvre possibles pour améliorer encore cette technique. Ainsi, si le dernier point vert avait été placé un peu plus haut (comme le point correspondant sur la figure 6), l'erreur serait moins importante.

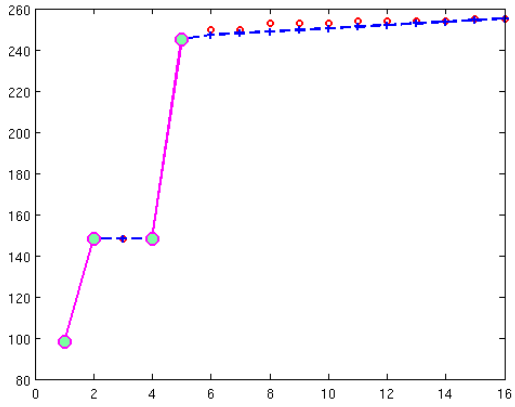

Figure 12 – Explication plus détaillée de l'approximation du jeu de points rouges. En magenta on a les parties de courbes qui sont approchées par les fonctions d'approximation f^1 et f^2. En bleu, on a ce qui est reconstruit grâce aux informations sur la monotonie de la fonction (première courbe bleu) et grâce aux informations sur les bornes (seconde courbe bleu).

Le tableau 2 résume l'erreur effectuée par les différentes approximations testées aux §3.1, §3.2 et §3.4. Pour chaque approximation seuls 5 paramètres nécessitent d'être stockés.

Remarque 3 *Au final, dans les deux exemples présentés dans cette partie, l'utilisation d'autres sources d'informations nous a permis d'approcher les jeux de points en approchant correctement les deux sauts présents au sein des jeux de données. Il parait donc envisageable de ne pas faire de nombreux tests en « relâchant » successivement les contraintes sur certains points mais de juste voir si, en utilisant une fonction de reconstruction appropriée, l'approximation des deux plus grandes discontinuités n'est pas une meilleure approximation.*

3.5 Conclusion et perspectives concernant l'approximation

Dans cette partie nous avons développé différents moyens d'approcher un nuage de points : ceci peut être fait par une approximation linéaire (§3.2) ou quadratique (§3.3) ou par des approximations polynomiale de plus haut rang au prix d'un nombre de paramètres à stocker plus important.

Au §3.4 des méthodes d'approximations utilisant des informations concernant le jeux de données

ont été présentées dans le cas d'une approximation linéaire par morceaux. Les mêmes méthodes peuvent être employée pour des approximations polynomiales d'ordre plus élevé. Ces méthodes se sont révélés très efficaces sur les cas testés au prix d'un coût de calcul plus important lors de la compression. Cependant la remarque 3 laisse envisager qu'il est possible d'utiliser ces idées avec un coût de calcul assez faible.
Une information non utilisée dans cette partie est la valeur discrète des pixels en chaque point. Il est envisageable d'utiliser aussi cette information lors de la compression (et de la reconstruction) pour gagner un petit quelque chose sur chaque pixel.

Dans ces parties, on a toujours supposé que le meilleur moyen de reconstruire un jeux de données était de reconstruire de manière optimale les valeurs de chaque pixels sur chacun des canaux (rouge, vert, bleu). Il est envisageable de chercher un minimum tenant compte des 3 canaux en même temps. Dans ce cas l'erreur, en un pixel, pourrait s'écrire :

$$\text{Erreur} = \sqrt{(f^{red}(x_i) - y_i^{red})^2 + (f^{green}(x_i) - y^{green})^2 + (f^{blue}(x_i) - y^{blue})^2} \tag{16}$$

Ceci permettrait d'éviter que l'erreur se concentre sur le même pixel pour chaque canal.

4 Conclusion et perspectives

Dans ce document nous avons résumé les différentes idées suivies et testées lors des 3 jours de la SEME pendant lesquels nous avons eu la chance de travailler sur ce sujet. De nouvelles méthodes concernant le choix des clusters ont été abordées aux §2.1, §2.2 et §2.3. Des exemples concrets obtenus pour les différents clusters en utilisant la méthode de compression introduite en §3.1 sont présentés dans les figures 1-3.
Dans la seconde partie, des méthodes utilisant des approximations polynomiales d'ordre plus important ont été décrites. Il a aussi été montré dans cette partie que la connaissance d'information a priori sur les fonctions à approcher peut être utilisée afin de réduire l'erreur d'approximation. Durant cette semaine de travail nous avons cependant aussi abordé la question concernant l'utilisation d'autres espaces de travail (par exemple travailler dans l'espace *luminance/chrominance* plutôt que l'espace *couleur*).
Nous nous sommes aussi demandé si le tri des pixels était indispensable. Dans les faits, ce tri permet d'avoir un a priori important sur la fonction à approcher (utile pour une approximation linéaire mais moins pour des approximations polynomiales de plus haut degré). Cependant il est gourmand en terme de stockage. Nous nous sommes donc brièvement penché sur l'opportunité de construire d'autres a priori sur la fonction à reconstruire, sans utiliser de tri. Ceci a débouché sur l'idée de reconstruire la valeur de chaque pixel en *approchant* la fonction somme cumulée des valeurs des pixels. En effet, la somme cumulée est aussi une fonction croissante, dont les sauts sont bornées à 255 (ce qui nous fournit deux a priori importants). Contrairement à la méthode utilisée jusqu'à présent il n'y a pas besoin de stocker un tri lors de la définition de la somme cumulée. Ainsi, seuls les paramètres utilisés pour approcher la fonction croissante doivent être gardés en mémoire. Bien que non testée (par manque de temps), cette idée semble une perspective prometteuse afin d'améliorer grandement le taux de compression et de dé-corréler les erreurs au maximum.

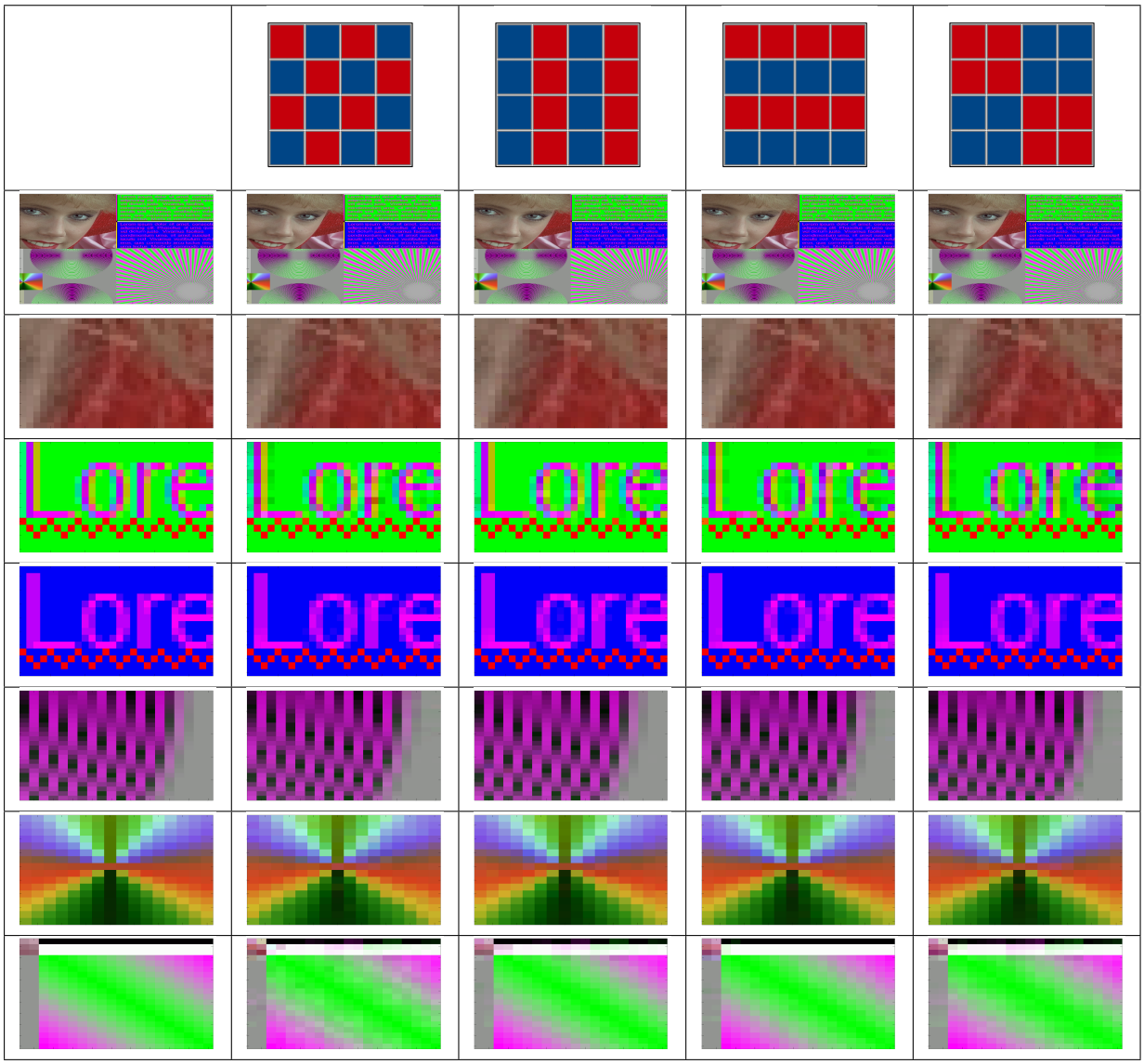

Figure 1 – En première ligne sont présentés des exemples de décompositions en clusters fixes et indépendants des variables d'un bloc 4×4 (chaque couleur correspond à un cluster). Dans les lignes suivantes on présente pour une image donnée les résultats obtenus pour chacune de ces décompositions (on approche les nuages de points avec la technique présentée en début de section 3).

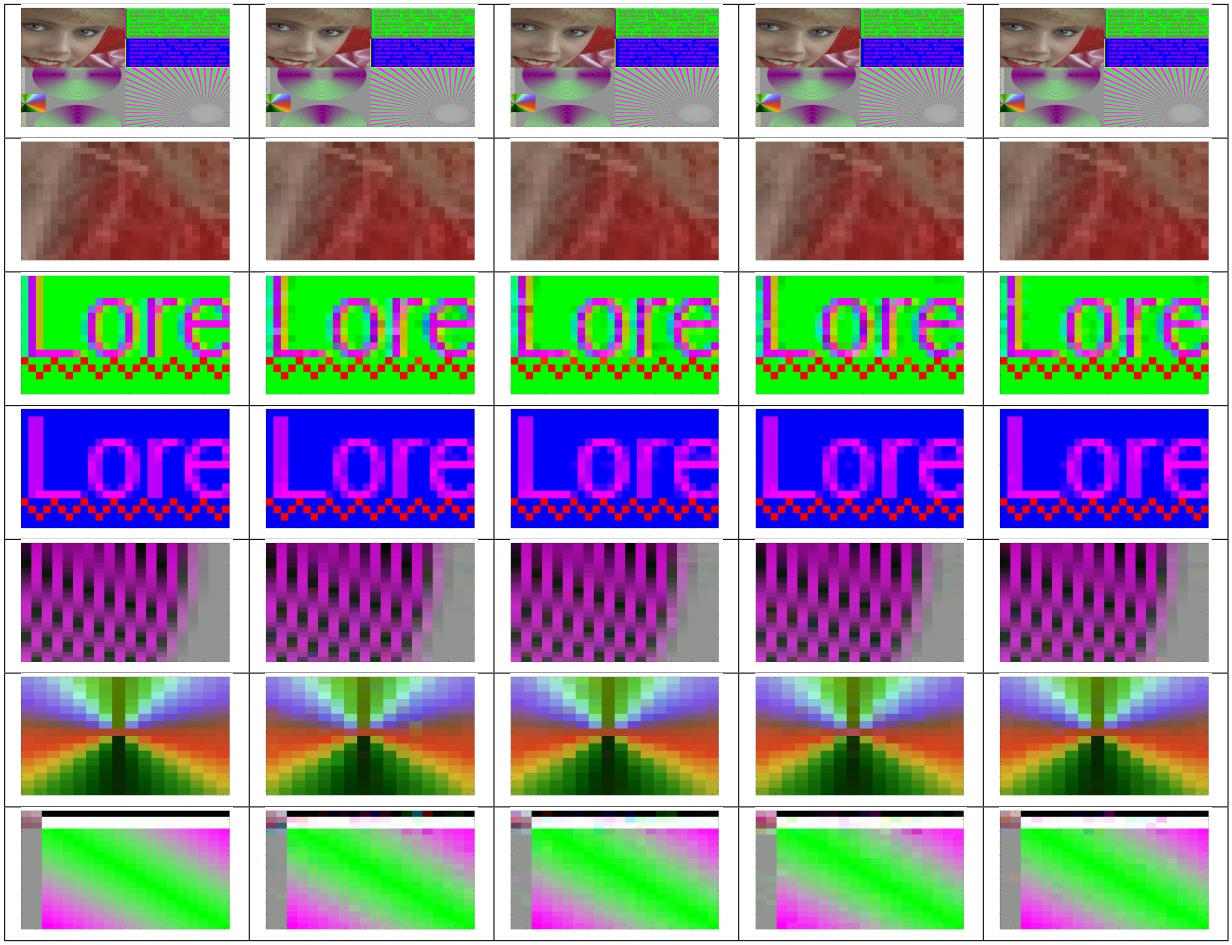

Figure 2 – Dans chaque ligne, des exemples de résultats obtenus en faisant des clusters aléatoires pour chaque bloc (la première colonne est l'image initiale) (on approche les nuages de points avec la technique présentée en début de section 3).

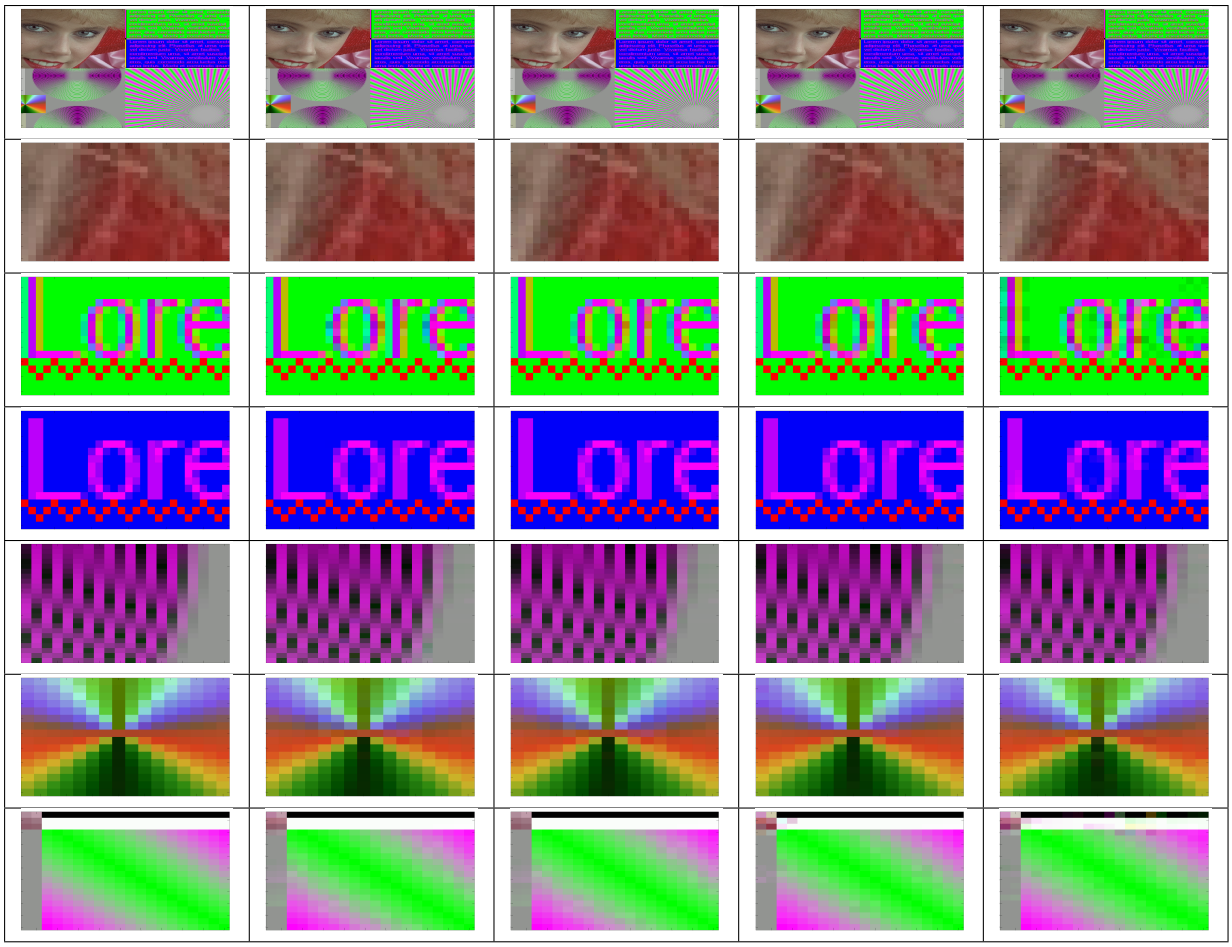

Figure 3 – Dans chaque ligne, des exemples de résultats obtenus en faisant des clusters dépendants des valeurs pour chaque bloc (on approche les nuages de points avec la technique présentée en début de section 3). Dans la première colonne on retrouve les images initiales. Dans la seconde nous avons séparer les valeurs inférieures et supérieures à la moyenne du maximum et du minimum. On a procédé de façon similaire dans la troisième colonne, mais en faisant trois clusters. Une tentative de séparation en plateaux se trouve dans la quatrième colonne alors que dans la dernière nous avons essayé de les éviter.