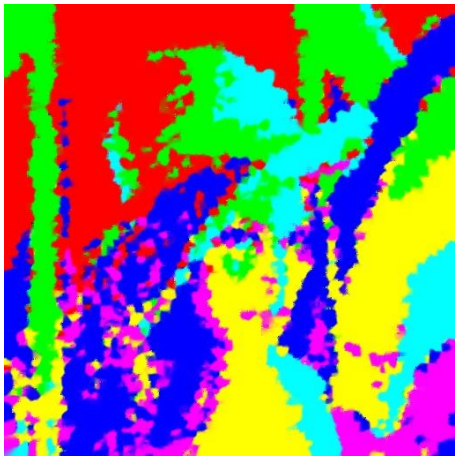
Bureau d'étude : Variations sur l'astuce des noyaux

STREIFF Axel / KLINGLER Mathieu / LAVARELO Julian / ALIX Simon

Classe : 3TS1



Professeur : M. COUFFIGNAL

Date de soumission : 21/03/2022

Table des matières

Introduction

Dans ce bureau d'étude nous utiliserons une méthode d'analyse statistique, le « clustering ». Cette méthode sert principalement à segmenter ou classifier une base de données, par exemple une image dans le deuxième exercice. On peut aussi extraire des informations pour essayer de former des sous-ensembles de données. Le clustering est utilisé dans l'imagerie spatiale pour organiser certaines données en cluster. Par exemple les forêts, villes ou zones agricoles, ainsi on permet de réduire la taille des paquets de données.

Cette méthode de clustering s'appuie sur des algorithmes des k-moyennes qu'on connait déjà bien.

Exercice 1 : Reconstruction de surface sans et avec noyaux

Partie 1 : Approximation linéaire

Dans cet exercice nous allons utiliser l'astuce des noyaux afin de reconstruire une surface 2D à partir de la donnée de points de $\mathbb{R}^3$.

Nous commencerons ce bureau d'étude par étudier la forme linéaire de la recherche d'une surface passant au plus près des points de l'ensemble P.

$$P := \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_m & y_m & z_m \end{pmatrix}$$

On cherche un plan Π tel que $\Pi = \alpha x + \beta y + \gamma z + \delta = 0$.

On suppose que $\gamma \neq 0$, on peut déjà en déduire que cette condition supprime certains types de plan et on peut noter le plan « table » de la surface, tout le plan qui est sur l'origine avec $\gamma = 0$.

On peut alors trouver l'équation du plan :

On divise l'équation du plan par γ :

$$\frac{\alpha}{\gamma}x + \frac{\beta}{\gamma}y + z + \frac{\delta}{\gamma} = 0$$

$$\frac{-\alpha}{\gamma}x - \frac{\beta}{\gamma}y - \frac{\delta}{\gamma} = z$$

On a donc : $a = \frac{-\alpha}{\gamma}$; $b = \frac{-\beta}{\gamma}$; $d = \frac{-\delta}{\gamma}$.

Le plan solution s'écrit bien sous la forme d'une fonction $f(x, y) = ax + by + d$

Montrons à présent que le problème de la recherche des coefficients a, b, d s'écrit sous la forme d'un problème de moindre carrée de la forme :

$$X^* := \arg\min \| AX - Z \|_2^2$$

Le problème s'écrit donc :

$$\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ d \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix}$$

Le vecteur $X* = (a, b, d)^T$ solution de cette équation au sens des moindres carrés s'exprime grâce à la matrice pseudo inverse de A.

On a

$$X* = A^+ Z$$

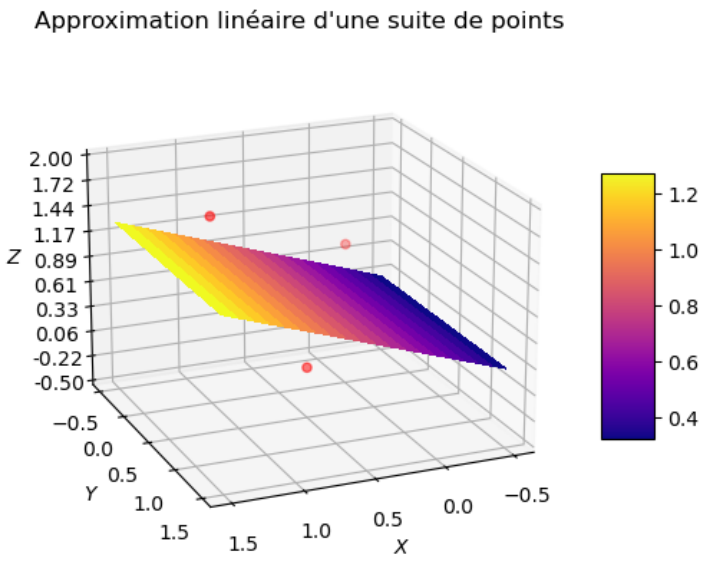
Une fois que nous avons déterminé le vecteur solution, il suffit de reconstruire le plan approché en appliquant a chaque point d'un espace choisi la fonction :

$$f(x, y) = ax + by + d$$

En appliquant ce processus à l'ensemble de point

$$P_{(m,n)} := \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1.5 \\ 0 & 1 & 0.5 \\ 1 & 1 & 1 \\ 0.5 & 0.5 & 0 \end{pmatrix}$$

Nous obtenons l'approximation suivante :



Nous pouvons calculer l'erreur commise lors de cette approximation avec la formule :

$$err = \|AX* - Z\|$$

Dans le cas de notre approximation linéaire, nous obtenons une erreur de 1.025. Essayons maintenant de minimiser cette valeur en passant par des méthodes non-linéaires.

Partie 2 : Approximation non-linéaire

A présent nous allons généraliser le procédé précédent et s'autoriser à avoir des surfaces non-linéaires grâce à l'astuce des noyaux. En repartant de l'équation :

$$X^*: = \arg\min \| AX - Z \|_2^2$$

Posons cette fois

$$\omega = A^T = \begin{pmatrix} x_1 & x_2 & \cdots & x_m \\ y_1 & y_2 & \cdots & y_m \\ 1 & 1 & \cdots & 1 \end{pmatrix}$$

Avec

$$\omega_i = \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$$

Alors

$$\|AX - Z\|_2^2 = \left\| \begin{pmatrix} {\omega_1}^T x - z_1 \\ {\omega_2}^T x - z_2 \\ \vdots \\ {\omega_m}^T x - z_m \end{pmatrix} \right\| = \sum_{i=1}^{m} ({\omega_i}^T x - z_i)^2$$

Ainsi

$$X^*: = \arg\min \sum_{i=1}^{m} ({\omega_i}^T x - z_i)^2$$

Soit $k: \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$ un noyau.

On applique l'astuce des noyaux :

$$\sum_{i=1}^{m} ({\omega_i}^T x - z_i)^2 = \sum_{i=1}^{m} (< R, \phi(\omega_i) > - z_i)^2$$

D'après le théorème de Mercer, nous avons $R = \sum_{j=1}^{m} c_j k(\omega_j , \cdot)$. Le problème revient à :

$$c^* = \arg\min \sum_{i=1}^{m} (< R, \phi(\omega_i) > - z_i)^2$$

Avec $c^* = \begin{pmatrix} {c_1}^* \\ {c_2}^* \\ \vdots \\ {c_m}^* \end{pmatrix}$

Soit $i \in [1, m]$

$$< R, \phi(\omega_i) > - z_i = < \sum_{j=1}^{m} c_j k(\omega_j , \cdot), \phi(\omega_i) > - z_i = \sum_{j=1}^{m} c_j < k(\omega_j , \cdot), \phi(\omega_i) > - z_i$$

$\phi(\omega_i)$ est une fonction de redescription.

$$= \sum_{j=1}^{m} c_j < k(\omega_j, \cdot), k(\omega_i, \cdot) > - z_i$$

$$= \sum_{j=1}^{m} c_j k, \omega_j) - z_i$$

$$= \left(k(\omega_1, \omega_1), k(\omega_1, \omega_1), \dots, k(\omega_1, \omega_1)\right) \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix} - z_i$$

Lorsque l'on généralise cette formule pour tout i nous obtenons :

$$\left\| \begin{pmatrix} k(\omega_1, \omega_1), & k(\omega_1, \omega_2) & \dots & k(\omega_1, \omega_m) \\ \vdots & \ddots & & \vdots \\ k(\omega_m, \omega_1) & k(\omega_m, \omega_2) & \cdots & k(\omega_m, \omega_m) \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix} - \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix} \right\|_2^2$$

Avec $c^* = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix}$ et $z = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix}$.

Ainsi le problème précédent s'écrit matriciellement sous la forme :

$$\|KC - Z\|_2^2$$

Avec K la matrice de Gram.

Nous calculons donc le vecteur solution grâce à la méthode des moindres carrés

$$C = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix}$$

Le problème se résume donc à :

$$c^* = \arg\min \sum_{i=1}^{m} (< R, \phi(\omega_i) > - z_i)^2$$

$$c^* = \arg\min \|KC - Z\|_2^2$$

Ce problème admet une solution c^* unique garantie au sens classique des moindres carrées lorsque la matrice de Gram K est définie positive.

De la même manière que dans la partie 1, nous pouvons maintenant reconstruire le plan qui approxime le mieux la surface décrite par l'ensemble de points donné.

Nous admettons que la fonction solution s'écrit :

$$f_K(x, y) = \left\langle R *, \phi\left(\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}\right) \right\rangle = \sum_{i=1}^{m} c_i k\left(\omega_i, \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}\right)$$

Avec $R* = \sum_{i=1}^{m} c_i k(\omega_i)$

On suppose que l'on cherche une solution de la forme d'un plan :

$$f_K(x,y) = \alpha x + \beta y + \gamma = ax + by + d$$

On réécrit $(a \quad b \quad d)\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \backsim (a^* \quad b^* \quad d^*)\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$

Ainsi $f_K(x,y) = \left\langle c_k^*, \phi\left(\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}\right)\right\rangle$

On peut appliquer le théorème de Mercer, ce qui nous donne :

$$\left(k(\omega_1,\cdot), k(\omega_2,\cdot), \dots, k(\omega_m,\cdot)\right)\begin{pmatrix} {c_1}^* \\ {c_2}^* \\ \vdots \\ {c_m}^* \end{pmatrix}$$

$$= R^* = \sum_{i=1}^{m} {c_i}^* k(\omega_i,\cdot)$$

Ainsi :

$$f_K(x,y) = \left\langle R^*, \phi\left(\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}\right)\right\rangle$$

$$= \left\langle \sum_{i=1}^{m} {c_i}^* k(\omega_i,\cdot)\,, \phi\left(\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}\right)\right\rangle$$

$$= \sum_{i=1}^{m} {c_i}^* k\left(\omega_i, \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}\right)$$

R^* est la solution des moindres carrées à noyaux.

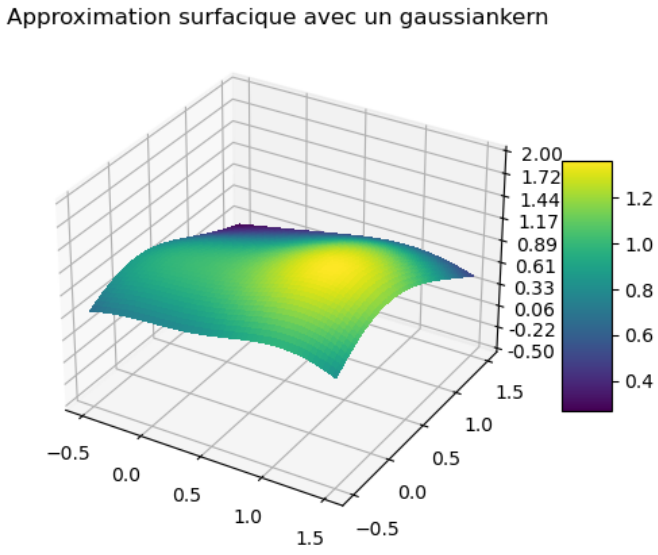
Appliquons maintenant cela à la même matrice P que précédemment afin de voir en quoi la méthode a noyau diffère.

Premièrement nous testons cette méthode avec le noyau Gaussien. Pour rappel, le noyau Gaussien est une fonction a deux variables et une constante de la forme :

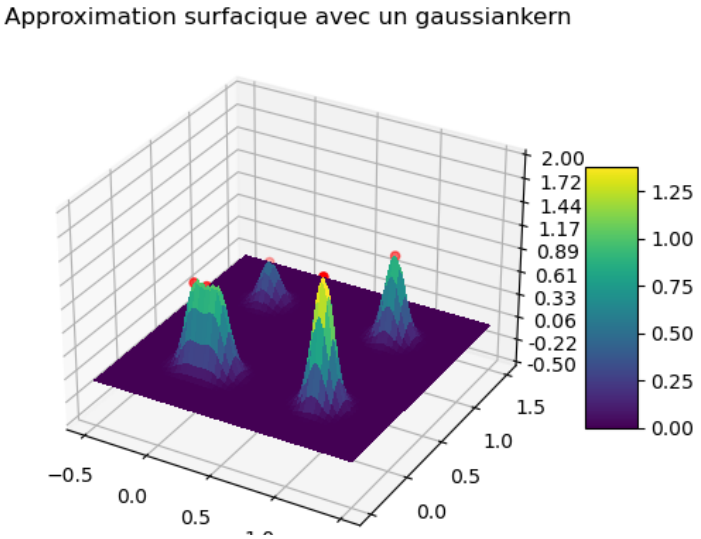
$$Gkern(x,y,c) = exp(\frac{1}{c^2}) * \|x-y\|^2$$

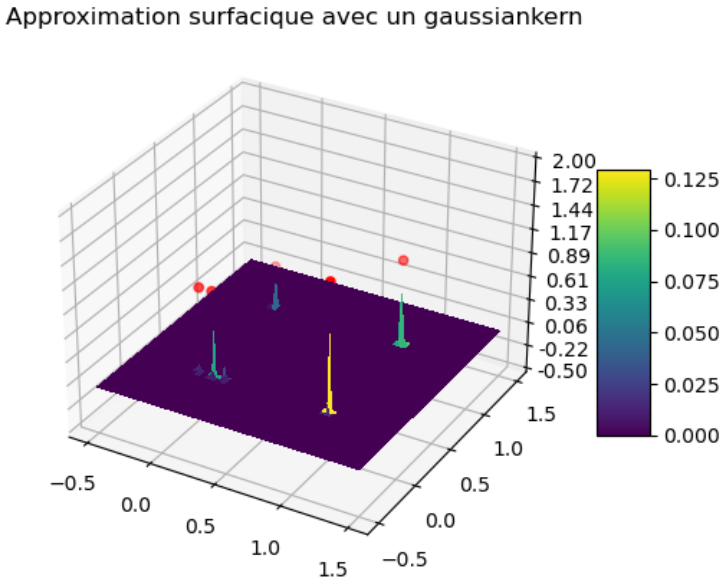
Avec c une constante à choisir permettant d'avoir plus ou moins de précision lors de l'approximation.

Dans un premier temps nous testons différentes valeurs de la constante c arbitrairement afin de voir l'effet sur la surface :



$c = 1$ *avec une erreur de* $8.3 * 10^{-13}$



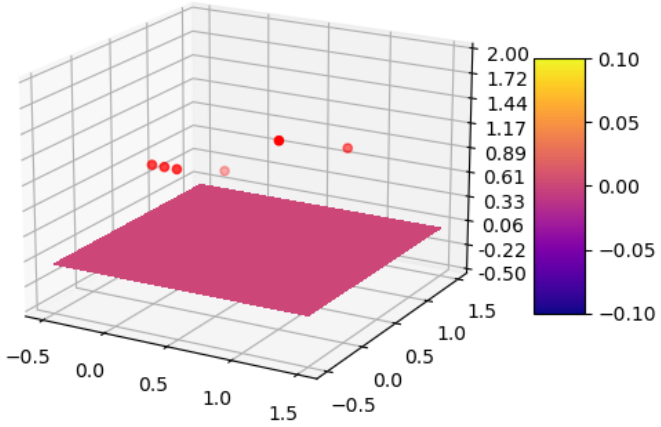


$$c = 10^{-2} \ avec\ une\ erreur\ de\ 4.44 * 10^{-16}$$

On voit que lorsque c réduit, l'approximation est de plus en plus précise, en n'atteignant les valeurs demandées que sur une zone très réduite. Nous avons donc codé une fonction permettant de déterminer la valeur de c permettant de minimiser l'erreur. Ainsi, on apprend qu'avec $c = 10^{-5}$ il est possible d'obtenir une approximation parfaite, c'est-à-dire une erreur nulle.

Nous obtenons dans ce cas le graphique suivant :

$$c = 10^{-2} \ avec\ une\ erreur\ de\ 4.44 * 10^{-16}$$

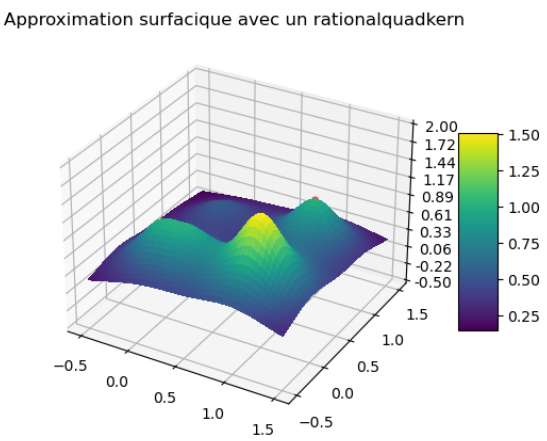


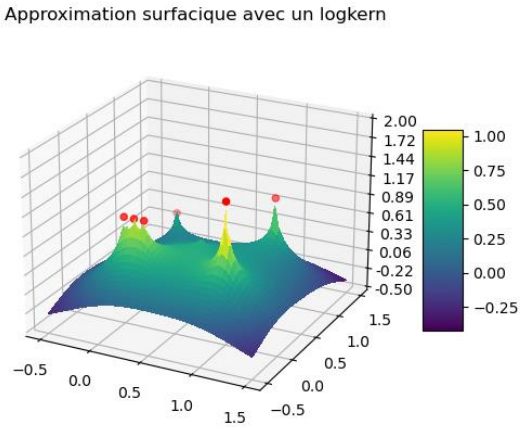
$$c = 10^{-5} \ avec\ une\ erreur\ de\ 0$$

Cette surface à première vue plane est en fait un plan ou aux coordonnées de chaque point de la matrice P, un Dirac apparait. Du fait du coté infinitésimal de cet objet mathématique, nous ne pouvons l'apercevoir à l'écran. Il est donc théoriquement possible d'approcher une surface parfaitement du point de vue mathématique, mais physiquement ces résultats ne sont pas utilisables notamment du fait de leur non-dérivabilités en certains points.

Le noyau Gaussien permet donc d'avoir une grande précision à condition de déterminer les bonnes constantes. Nous remarquons tout de même que dans certains cas, avoir le plus de précision ne garantis pas un graphique utilisable, dans le cas où nous voudrions reconstituer une plaine montagneuse à partir de quelques données géographiques, le cas des Dirac est difficilement envisageable.

Pour pallier ce problème, il est possible de modifier les constantes des fonctions noyaux ou bien d'en utiliser d'autres. Il y a par exemple le noyau quadratique rationnel ou encore le noyau logarithmique qui donnent d'assez bons résultats :

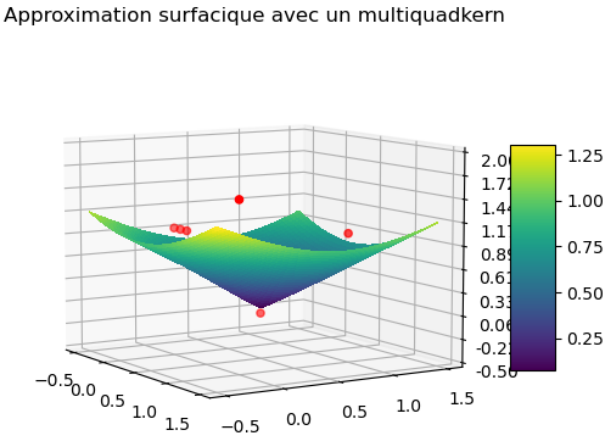


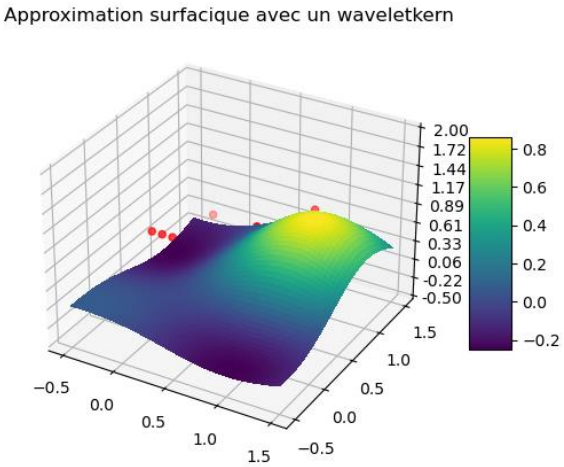


Le noyau quadratique rationnel permet d'avoir presque les mêmes résultats que le noyau gaussien en termes de précision mais la surface est plus « dérivable ». Le noyau logarithmique, dans les conditions du graphique permet une erreur nulle sans discontinuités à la base de la surface contrairement au noyau gaussien.

D'autres noyaux peuvent être utilisés, mais sont respectivement moins performants sur cette matrice (le noyau multi quadratique) soit complexes à utiliser car dépendant de 2 constantes (le noyau wavelet).

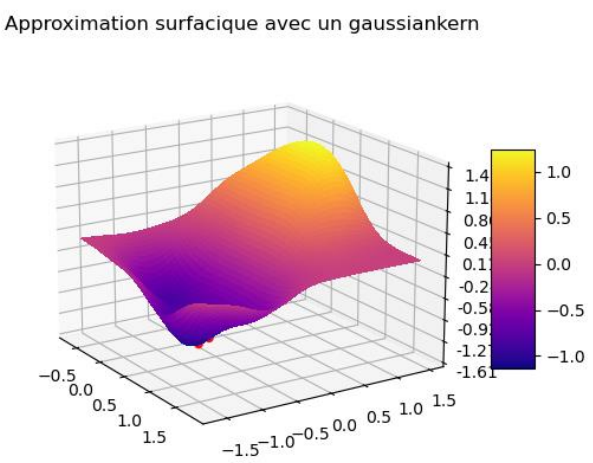
Nous avons pour ces deux noyaux les résultats suivants :

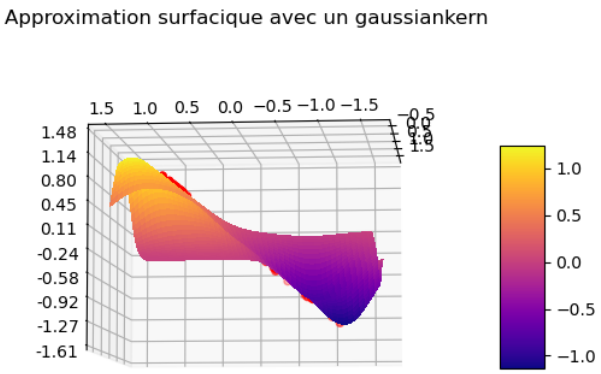




Pour ce qui est de la question défi, le fait de demander le noyau renvoyant l'erreur la plus faible est vite répondu puisque le noyau gaussien permet toujours d'avoir une erreur nulle en renvoyant des Dirac. Cela dit nous avons essayé d'approximer par une surface réaliste en gardant le plus de précision possible.

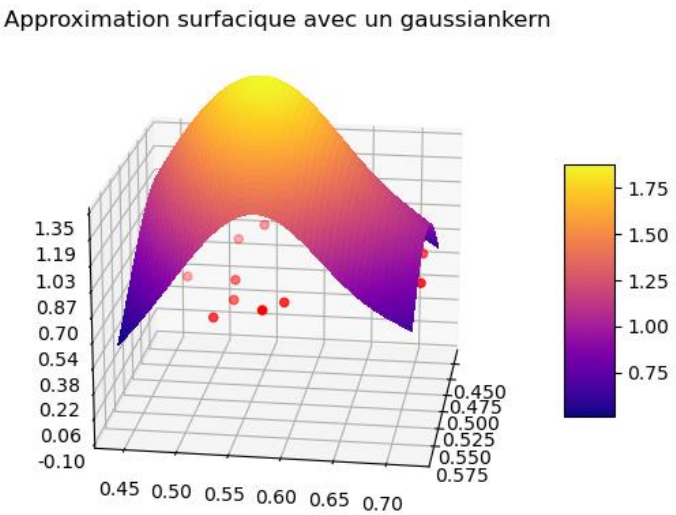
Nous obtenons :





Sous ces deux angles on peut voir que la surface est collée aux points rouges et donc minimise l'erreur (3*10**-4)

Enfin après avoir récupéré quelques données géographiques, nous avons essayé de reconstruire une carte d'Andorre, voici le résultat avec 12 points répartis équitablement sur la carte :



Exercice 2 : partitionnement d'image avec ACP et k-moyennes

Le partitionnement d'image, la catégorisation de données ou la régression de nuage de points ont en commun une solution simple à voir à l'œil nu. Comme sur une image de parking ou notre cerveau différencie très facilement une voiture du reste du cliché, la question est de traduire cette intuition numériquement.

Nous avons écrit un programme d'apprentissage non superviser qui partitionnement des images pour en extraire les masques des différents objets que le code reconnaitra. Comme nous n'indiquons pas à l'avance le nombre ou la forme des partitionnements de l'image, nous utilisons les algorithmes d'analyse en composantes principales et de K-moyennes que nous expliquerons en détail dans des parties spécifiques.

Partie 1 : L'ACP le retour du retour du retour…

Nous avons vu et utilisé l'algorithme de l'analyse en composante principal plusieurs fois déjà, le but est toujours le même. A partir d'un tableau de valeur X, trouver les directions d'analyse les plus pertinente, qui met en avant les catégories des différentes instances du tableau.

Dans notre cas, nous travaillons sur une image et nous cherchons à vectoriser l'image tout en conservant le maximum d'information sur les pixels. Le tableau que l'on créer pour appliquer l'ACP est de la forme :

Tableau 1 : extrait de la matrice X, description d'un pixel sur une image

Indice i de la ligne du pixel	*Indice j de la colonne du pixel*	*Intensité de rouge r du pixel*	*Intensité de vert v du pixel*	*Intensité de bleu b du pixel*	*Moyenne de rouge rm autour du pixel*	*Moyenne de vert vm autour du pixel*	*Moyenne de bleu bm autour du pixel*

L'information de chaque pixel est un vecteur de $\mathbb{R}^8$ et l'ACP va nous simplifier l'information en un vecteur de $\mathbb{R}^2$.

Pour cela il nous faut faire la décomposition en valeur singulière de la matrice de covariance du *tableau 1 :*

$$C = X^T X = UDU^T$$

La partie la plus gourmande en calcule de cet algorithme et la SVD de la matrice C. Or cette matrice sera toujours de taille (8,8) indépendamment du nombre de pixel de l'image. Pour construire les composantes principales de X, on utilise les k premières colonnes de U.

Il est aussi possible de calculer U en diagonalisant la matrice de covariance C :

$$C = X^T X = PDP^{-1}$$

Or C est symétrique donc la matrice P est orthogonal, de plus on fait la décomposition SVD de X :

$$U\Sigma U^T (U\Sigma U^T)^T = PDP^T$$

$$U\Sigma\Sigma U^T = PDP^T$$

On identifie que $U = P$, donc que les vecteurs propres de la base de diagonalisation de C sont aussi les directions principales de X.

On sait donc qu'il est possible d'obtenir les directions principales de X sans trop de calcule grâce a la matrice de covariance. La sortie de notre algorithme acp_img nous retourne une matrice de taille *(nb_pixel, 4)* de la forme :

Tableau 2: extrait d'une ligne du résultat de la fonction acp_img

Indice i de la ligne du pixel	*Indice j de la colonne du pixel*	*Composante principale 1 y1*	*Composante principale 2 Y2*

Nous cherchons maintenant à séparer au mieux composantes principales des pixels de l'image d'origine en utilisant l'algorithme des K-moyennes. Tout en conservant une trace de la position des pixels dans l'image, pour pouvoir la reconstruire par la suite.

Partie 2 : L'algorithme des k-moyennes

Nous avons fait une ACP de rang 2 sur la matrice X, d'où les 2 composantes $y1$ et $y2$, ce qui nous permets de facilement visualiser dans un plan la projection des différents pixels de l'image. Mais l'algorithme des k-moyennes utilises des notions de norme et de distance pour faire de l'apprentissage non superviser et séparer les pixels en n groupe. Ces outils mathématiques sont valables dans $\mathbb{R}^2$ comme dans $\mathbb{R}^m$, on n'est donc pas obligé de se cantonner à une ACP de rang 2.

Le meilleur moyen de trouvé un compromis est d'utiliser la règle de kaiser, qui indique le nombre de composantes principales utile à retenir en fonction des valeurs singulières associer. La pratique sur les images tests, que nous verrons dans la partie 3, nous donne toujours des valeurs de kaiser de 1, 2 ou 3.

On traduit la règle de kaiser en nombre de catégorie que l'algorithme des K-moyenne va créer. C'est un algorithme itératif qui essaie au mieux de minimiser la distance de tous les pixels aux barycentres associer aux catégories.

La sortie de l'algorithme S est une matrice de taille (nb_pixel, 5) de la forme :

Tableau 3: extrait d'une ligne de S

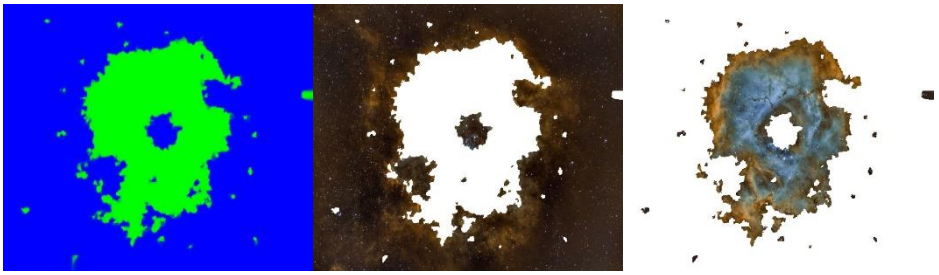
Indice i de la ligne du pixel	*Indice j de la colonne du pixel*	*Composante principale 1 y1*	*Composante principale 2 Y2*	*Groupe auquel appartient le pixels*

C'est grâce a la dernière colonne que l'on va pouvoir créer un masque de l'image d'origine avec seulement une catégorie de pixels. Si l'ACP et les poids ont bien été paramétré, le partitionnement de l'image doit faire ressortir des formes des contours ou des objets présents sur la photo.

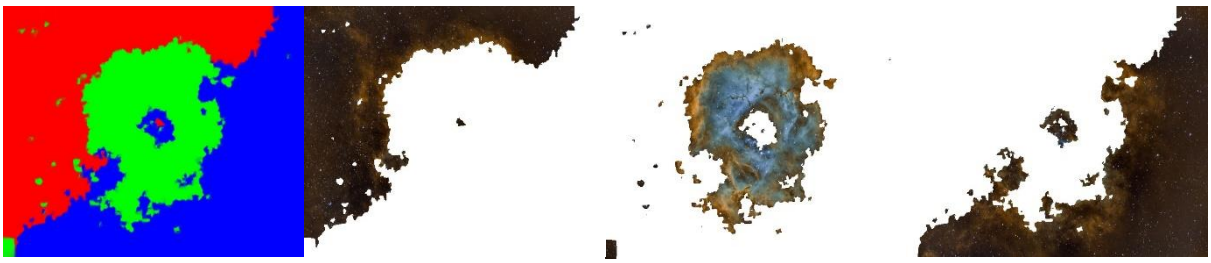
Nous verrons que si la théorie est simple, la mise en pratique est plus complexe. Le choix des paramètres de X est primordial pour définir les directions principales et donc classifier les pixels. Une fois les algorithmes programmés et fonctionnels, c'est le choix de ces paramètres qui est le plus dure et qui sort du cadre purement mathématique. Nous allons voir et commenter nos petites expériences de photographie.

Partie 3 : L'algorithme de partitionnement

Une fois que l'on connait pour chaque pixel sont groupe, ce que l'on représente visuellement par des couleurs, on peut donc créer autant de masque qu'il y a de catégorie. Ce qui nous donne les résultats suivant haut en couleurs :

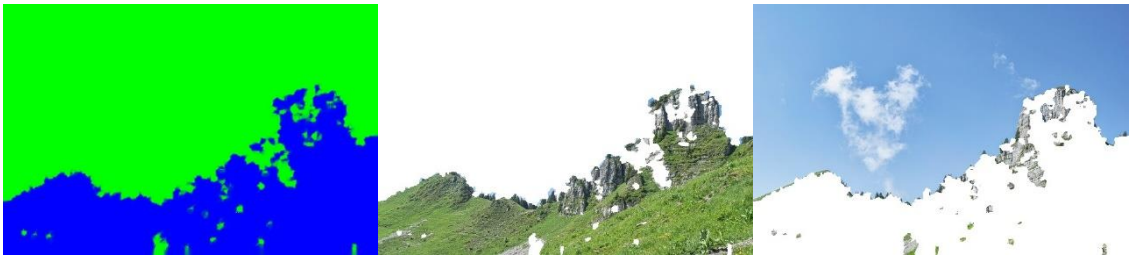


Cette image est celle d'une nébuleuse, on arrive bien à voir le détour de la nébuleuse par rapport à l'espace en arrière-plan. La règle de kaiser qui impose de découper l'image en 2 semble bien fonctionner, si l'on force un découpage de l'image en 3 groupes on trouve ceci :



Le résultat reste propre et la nébuleuse a bien été séparer, mais l'espace est maintenant en 2 parties. Si l'on s'intéresse aux raisons de ce partitionnement, on remarque que les couleurs des pixels sont relativement identiques. La matrice de données X contient des informations de couleur et de position et les directions principales ont dû privilégier les variations de coordonnées i et j. C'est pourquoi la séparation se fait sur cette diagonale, en haut les indices les plus faible et en bas les plus élevés.

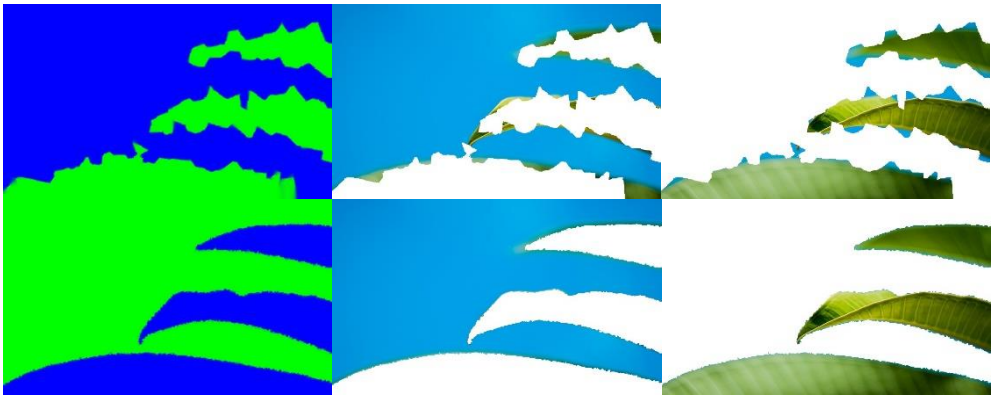
C'est la que rentre en compte les poids que l'on applique aux différents paramètres, il est possible de mettre l'accent sur une couleur bien spécifique que l'on cherche à séparer. Sur la photo suivante les poids associer aux couleurs bleu et verte ont été doublé par rapport au reste :



Pour réussir ce partitionnement nous avons dû mettre l'accent sur les couleurs verte et bleu qui sont un bon discriminant entre le ciel et la montagne. Le problème se trouve sur le masque le plus a droite, qui

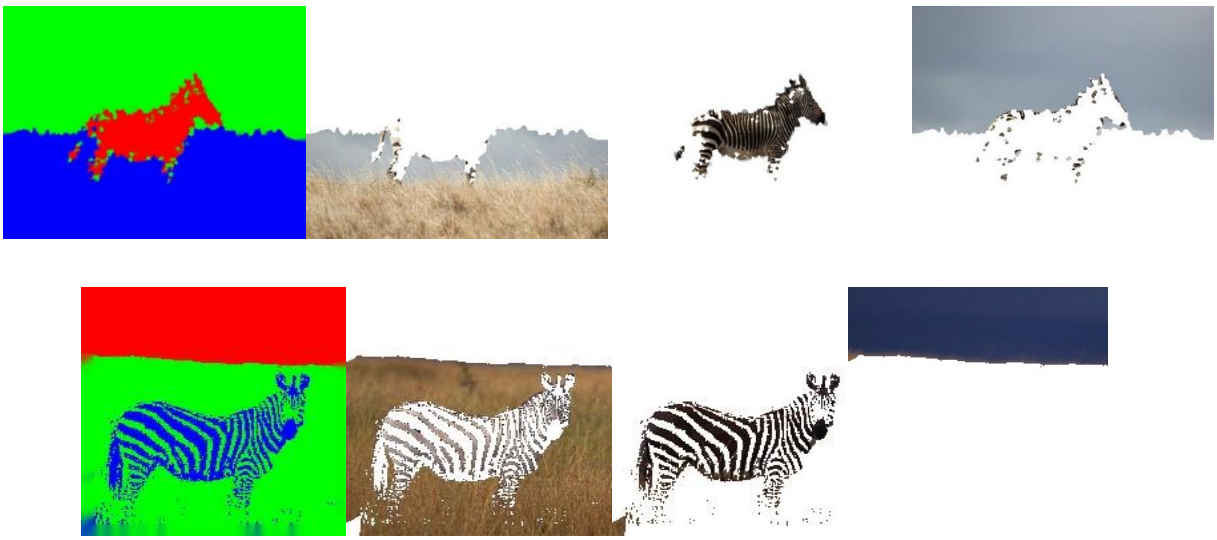
a cause du poids sur les couleurs classifie tous les couleurs les plus claires comme appartenant au ciel. C'est là qu'il faut s'attarder pour trouver les paramètres idéaux qui sépareront au mieux l'image.

Un autre paramètre qui influe sur la qualité du découpage est le nombre de pixel que l'on choisie sur l'image pour faire l'ACP. On montre pour une même image la différence entre 500 et 50000 pixels :



On voit bien que la détection et la réparation des feuilles de l'arrière-plan ne pause aucun problème dans les 2 cas, mais c'est sur la précision des bordures que la différence va se jouer. Il faut garder à l'esprit que pour une image de 640 par 420 pixels, prendre uniformément 50000 pixels revient juste à 5% de l'image total. Et la puissance de calcule nous limite dans la précision, indépendamment de la bonne séparation ou non des formes comme nous avons vu plus haut.

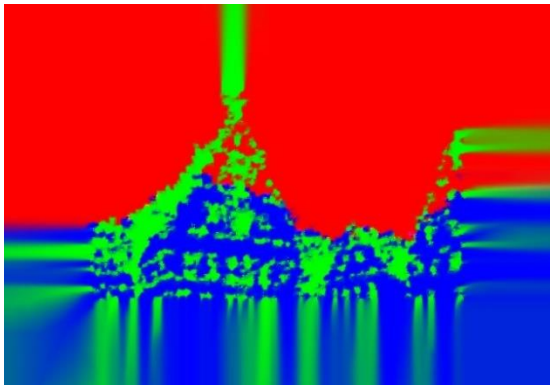
Parmi les paramètres qui décrivent un pixel il y a une moyenne des pixels environnant qui traduit l'idée de lissage de l'image. On peut voir l'influence de largeur de la zone de lissage sur le partitionnement d'une image de zèbre :



Ce ne sont certes pas les mêmes photos mais les conditions se ressemble, la différence se fait sur la zone de moyenne autour de chaque pixel. Dans le deuxième cas la moyenne a été fait sur seulement les 8 pixels autour et cela n'a pas été assez large pour englober les rayures du zèbre et donc être considéré comme un seul groupe.

Dans l'autre cas la moyenne a été fait sur un rayon de 10 pixels autour et le résultat est celui attendu. Notons tout de même une perte de précision sur les contours extérieurs du zèbre.

L'astuce du lissage de l'image avec un moyenne implique que les points en périphérie de l'image n'ont pas cette moyenne par manque de pixel autour. Pour palier ce problème, le choix uniforme des pixels se fait en excluant les bords de l'image. La fonction *inpaint* de cv2 rend un masque plein à partir d'un masque ponctuel et n'ayant pas d'information sur les contours de la photo, l'algorithme derrière improvise ! Lorsque l'on pousse à l'extrême ce paramètre, la fonction donne ce jolie mais totalement inutilisable résultat :



Voici d'autre résultat toujours issue de nos algorithmes, mais moins concluant du point de vu du partitionnement. A noté que plus une image est complexe, au sens charger avec beucoups de détail et de couleur différente et plus l'analyse sera ardu :