Alexandre Brochard

Vianney Guilleminet

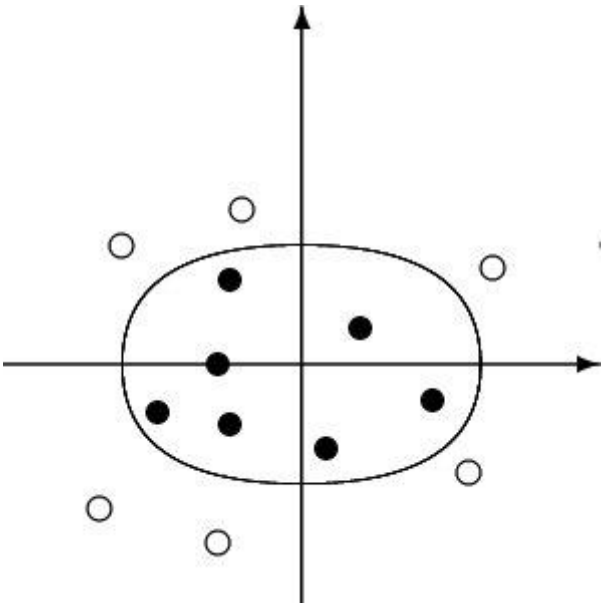
Hugo Le Dirach

Lucie Michelet

Chloé Pedoni

Bureau d'étude Ma325

Variations sur l'astuce des noyaux



Une représentation géométrique de l'astuce du noyau : (a) l'espace d'entrée d'origine ; (b) l'espace fonction de plus grande dimension, où les points sont linéairement séparés par l'hyperplan H. (Source : Support Vector Machines and Kernel Functions for Text Processing)

El Mahbouby – Mars 2022

Sommaire

Exercice 1 : Reconstruction de surface avec les moindres carrées à noyaux

1) a) On cherche à construire une surface 2D à partir de points donnés. On s'intéresse d'abords à des plans de forme linéaire. On peut écrire leur équation :

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

On suppose que $\gamma \neq 0$, cela élimine les plans parallèles au référentiel $(\vec{x}, \vec{y})$

On peut alors réécrire l'équation précédente :

$$\alpha x + \beta y + \gamma z + \delta = 0 \Leftrightarrow \alpha x + \beta y + \delta = -\gamma z$$

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

$$\Leftrightarrow ax + by + d = z$$

Avec $a = -\frac{\alpha}{\gamma}$, $b = -\frac{\beta}{\gamma}$ et $d = -\frac{\delta}{\gamma}$.

b) Nous souhaitons utiliser la méthode des moindres carrés pour résoudre cette équation et déterminer numériquement les coefficients a, b et d. Pour cela nous devons réécrire notre équation :

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

Sous la forme,

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

Avec : Z : = $(z_1, z_2, \cdots, z_m)^T \in \mathbb{R}^m$, X : = $(a, b, d)^T \in \mathbb{R}^3$

Et A la matrice s'écrivant : $A = \begin{pmatrix} x_1 & y_1 & 1 \\ ... & ... & ... \\ x_m & y_m & 1 \end{pmatrix}$ de taille (3, m).

On a,

$$\Leftrightarrow ax + by + d = z$$

$$\Leftrightarrow ax + by + d - z = 0$$

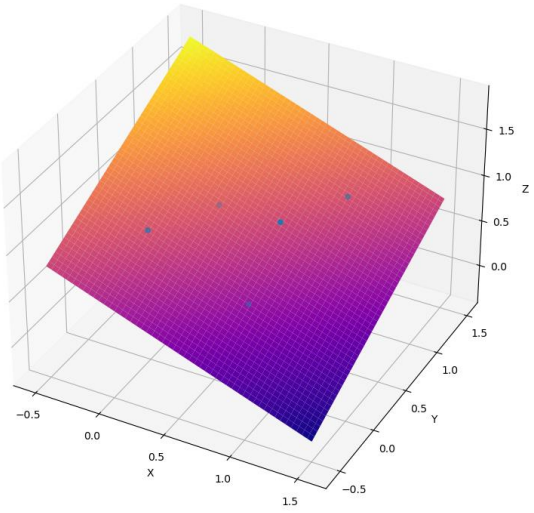
$$\Leftrightarrow \sum_1^m (ax_i + by_i + d - z_i)^2 = 0$$

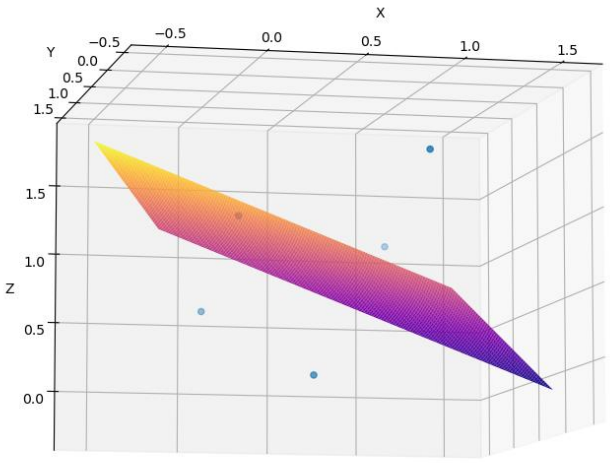
$$\Leftrightarrow \left\| \begin{pmatrix} ax_1 + by_1 + d - z_1 \\ ... \\ ax_m + by_m + d - z_m \end{pmatrix} \right\|_2^2 = 0$$

$$\Leftrightarrow \left\| \left(\begin{pmatrix} x_1 & y_1 & 1 \\ ... & ... & ... \\ x_m & x_m & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ d \end{pmatrix} - \begin{pmatrix} z_1 \\ ... \\ z_m \end{pmatrix} \right) \right\|_2^2 = 0$$

On résoudra numériquement par les moindre carré le problème : $X^* = arg\,min\|AX - Z\|_2^2$ afin de déterminer les coefficients a, b et d.

c) La résolution du système par la méthode des moindres carrés nous permet d'obtenir le plan-solution suivant :





On sait que l'algorithme des moindres carrés ne peut donner que le pan passant au plus près des différents points donnés. Cela n'est pas très satisfaisant puisque cela nous donne une imprécision de 89%

2) Nous souhaitons désormais améliorer notre méthode pour rendre la séparation des éléments plus juste. Pour cela nous utiliserons l'astuce des noyaux de Kernel. Il faut alors réécrire la forme de notre problème :

On note : ω : = (x, y, 1) ^{T} = A ^{T}, R = $\sum_{i=1}^{m} c_i k_{w_i}$ et c* = (c_1*, … , c_m*)^{T}.

$$\Leftrightarrow arg\ min \|AX - b\|_2^2$$

$$\Leftrightarrow \left\| \begin{pmatrix} w_1^T X - z_1 \\ \dots \\ w_m^T X - z_m \end{pmatrix} \right\|_2^2 = 0$$

$$\Leftrightarrow \sum_{i=1}^{m} \left(w_i^T X - z_i\right)^2 = 0$$

$$\Leftrightarrow \quad arg\ min \sum_{i=1}^{m} \left(w_i^T R - z_i\right)^2$$

On utilise les propriétés du théorème de Mercer et on obtient :

$$c^* = \arg\min \sum_{i=1}^{m} (\langle R, \phi(w_i) \rangle - z_i)$$

On cherche à écrire le problème précédent sous la forme matricielle :

$$\langle \phi(w_i), R \rangle - z_i = < \phi(w_i), \sum_{j=1}^{m} c_j k\left(w_j, \cdot\right) > - z_i$$

$$= \sum_{j=1}^{m} c_j \underbrace{k < k(w_i), k\left(w_j, \cdot\right) >}_{k\left(w_i, w_j\right)} - z_i$$

$$= (c_1 \quad \dots \quad c_m)\begin{pmatrix} k(w_i, w_1) \\ \dots \\ k(w_i, w_j) \end{pmatrix} - z_i$$

D'où :

$$\sum_{i=1}^{m}(\langle \phi(w_i), R\rangle - z_i)^2 = \left\| \begin{pmatrix} k(w_1, w_1) & \dots & k(w_1, w_m) \\ \dots & \dots & \dots \\ k(w_m, w_1) & \dots & k(w_m, w_m) \end{pmatrix} \begin{pmatrix} c_1 \\ \dots \\ c_m \end{pmatrix} + \begin{pmatrix} z_1 \\ \dots \\ z_m \end{pmatrix} \right\|_2^2$$

2.c

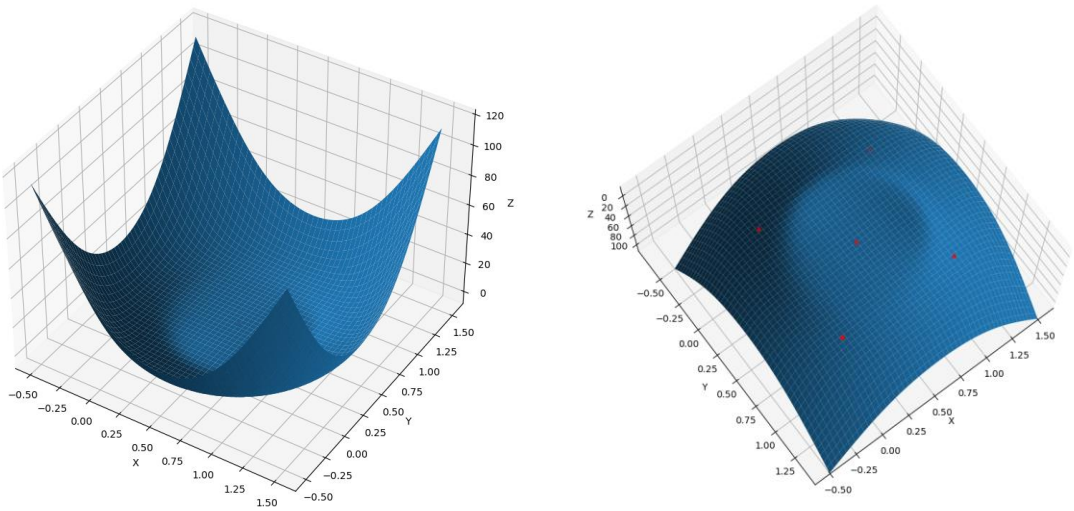
Finalement, nous pouvons réécrire une dernière fois le problème de la façon suivante :

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

$$= \sum_1^m c_i < k_{w_i}, \phi\left(\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}\right) >$$

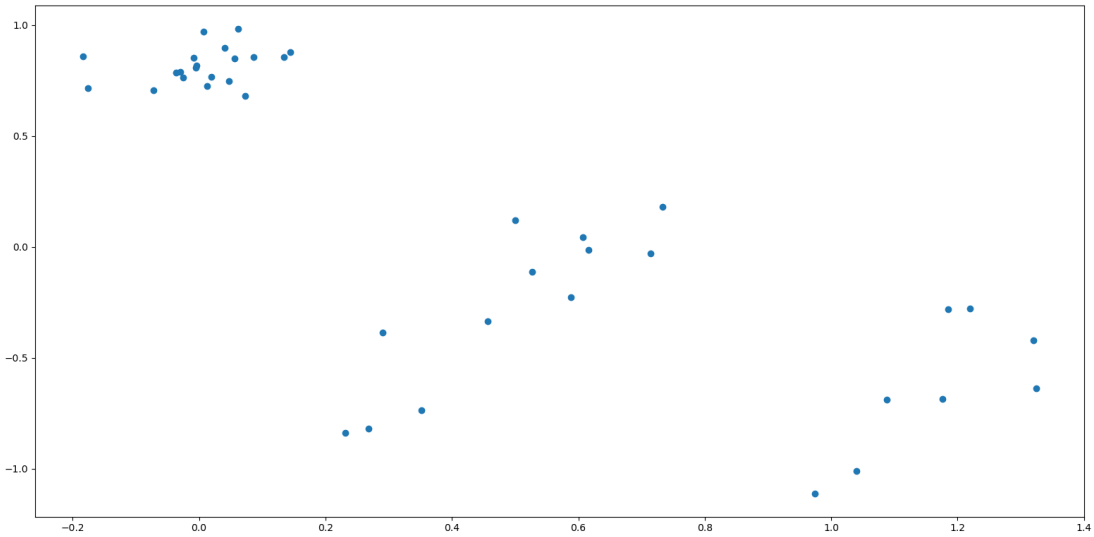
$$= \sum_1^m C_i k\left(w_i, \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}\right)$$

3) b) Les nouvelles fonctions utilisant l'astuce des noyaux appliquée aux moindres carrés permettent d'obtenir la solution suivante pour l'ensemble de points P :



On voit l'impact évident de l'utilisation des noyaux, puisqu'on trouve un plan courbé, solution très différente de celle trouvé précédemment. Cette fois ci l'imprécision est de $3.05 * 10^{-14}$, ce qui est très satisfaisant. Cependant, on observe l'apparition d'un temps de calcul conséquent. En effet la solution classique se calculait en moins d'une seconde, tandis que la solution avec noyau prend une dizaine de seconde.

c) Dans cette question on va chercher à appliquer la méthode des moindres carrés avec noyau sur l'ensemble de points suivants :



On cherchera le noyau permettant l'erreur de précision la plus faible possible.

4) On cherche maintenant à appliquer les fonctions MCKsurface () et VisualisationK () a d'autre données. Pour cela on génère grâce à la commande Python

```
P_test = np.random.randint(N, size = (10, 3))
```

Une matrice de N points avec des valeurs variant entre 0 et 5. On peut alors appliquer la méthode des moindres carrés à noyaux.

On peut alors voir que les noyaux essayés (polynomial, gaussien, multi-quadratique, Cauchy et logarithmique) ont des temps de calculs et des précisions différentes.

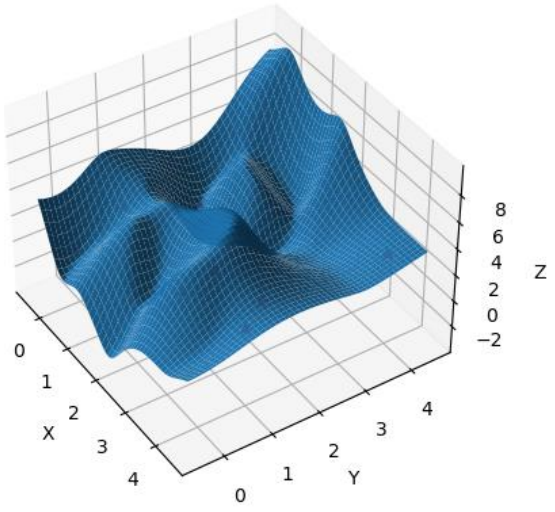
Par exemple pour une matrice P :

$$P = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 3 & 2 & 0 \\ 4 & 1 & 2 \\ 0 & 3 & 2 \\ 3 & 1 & 4 \\ 4 & 4 & 3 \\ 1 & 2 & 3 \\ 2 & 1 & 0 \end{pmatrix}$$

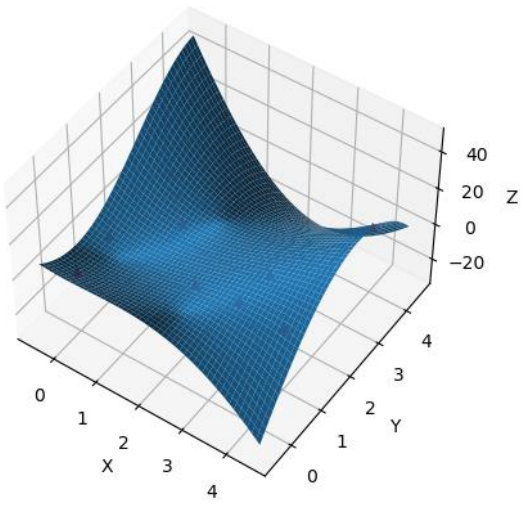
On obtient le tableau des précisions suivant :

Noyau	Polynomial	Gaussien	Multi-quadratique	Cauchy	Logarithmique
Précision	$1.11 * 10^{-11}$	$3.68 * 10^{-12}$	$3.48 * 10^{-9}$	$1.45 * 10^{-12}$	$4.03 * 10^{-14}$

On voit bien que même si les précisions sont toutes satisfaisantes, il y a un grand écart entre la plus petite, noyau multi-quadratique, et la plus grande, logarithmique. Cela se reflète bien sur les deux graphiques :



Noyau Logarithmique



Noyau Multi-quadratique

De plus en regardant le temps d'exécution, on peut établir que, de manière générale, la précision d'un noyau est contrebalancée par la rapidité d'exécution de l'algorithme.

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

Dans cet exercice on va créer un algorithme de partitionnement d'image, on rappelle auparavant l'algorithme de l'analyse en composantes principales (ACP).

Partie 1 : L'ACP le retour …

On a la matrice des données centrées réduites telle que $X_c = U\Sigma_r V^T$, de plus la matrice de covariance est de la forme

$$C = X_{rc}^T X_{r_c} = (U\Sigma_r V^T)^T (U\Sigma_r V^T)$$

Or $C = PDP^T$

On définit la première composante principale soit le vecteur centré v :

$$v = \arg\max(x_c\ v)$$

Ce vecteur est la direction principale. On a donc :

$$\arg\max(x_c\ v) = \arg\max(v^T C\ v)$$

On remplace ensuite C par son expression ci-dessus, soit :

$$\arg\max(v^T C\ v) = \arg\max(v^T\ PDP^T\ v)$$

On sait que P est orthogonal donc :

$$v^T v = v^T\ PP^T\ v \quad \leftrightarrow \quad (P^T v)^T\ P^T\ v$$

On pose $U = P^T v$ et on remplace :

$$\arg\max(v^T\ PDP^T\ v) = \arg\max(U^T DU)$$

On constate donc que les directions principales sont données par les vecteurs propres de U.

Par ailleurs nous avons aussi démontré que la diagonalisation est obtenue avec une décomposition SVD de la matrice de covariance C.

On compare le résultat des fonctions ACP et ACP pondéré sur différentes photos (masques globaux).

Photo bâtiment ACP

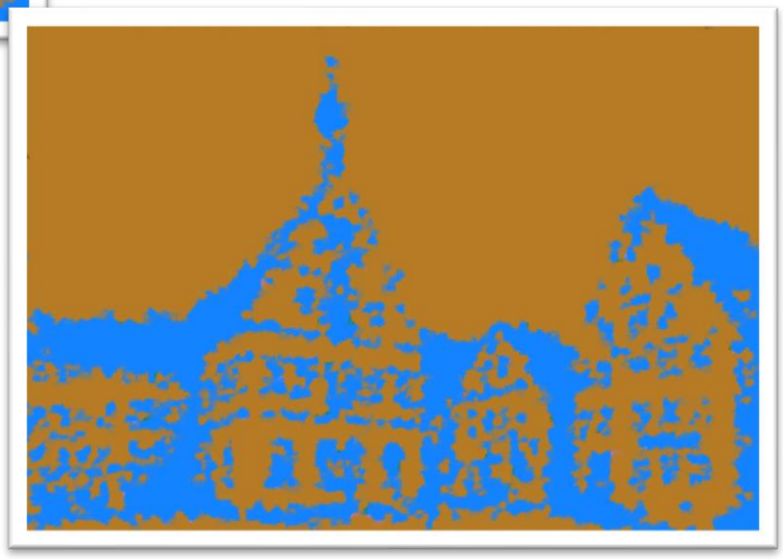


Photo bâtiment ACP pondéré

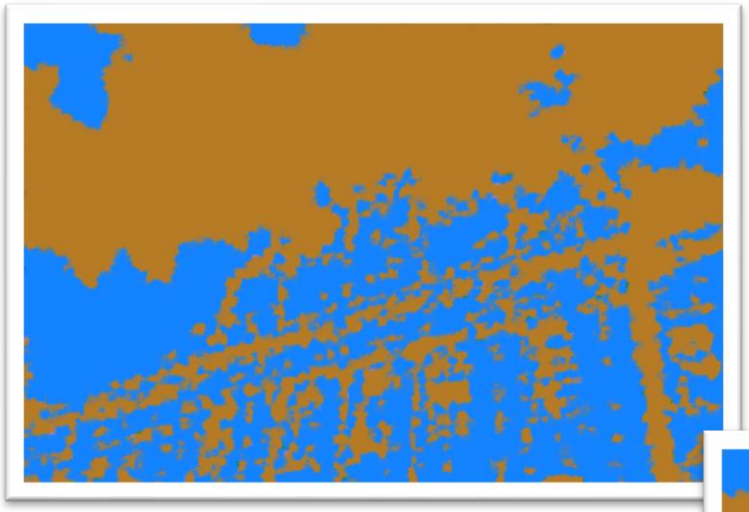


Photo immeuble ACP

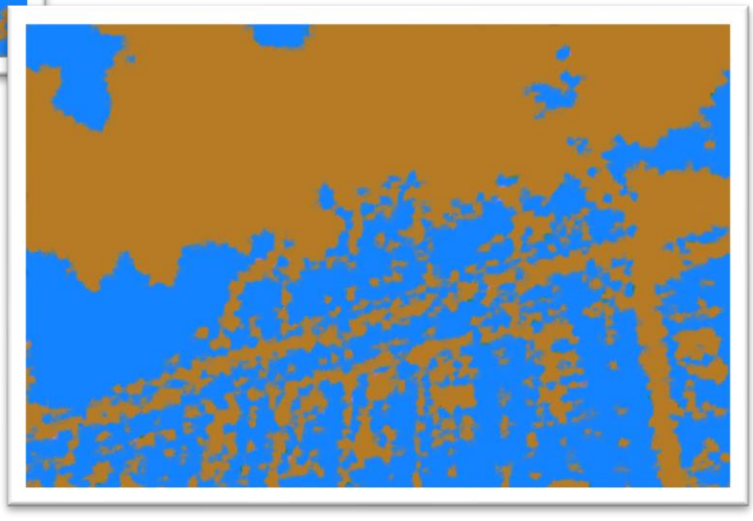


Photo immeuble ACP pondéré

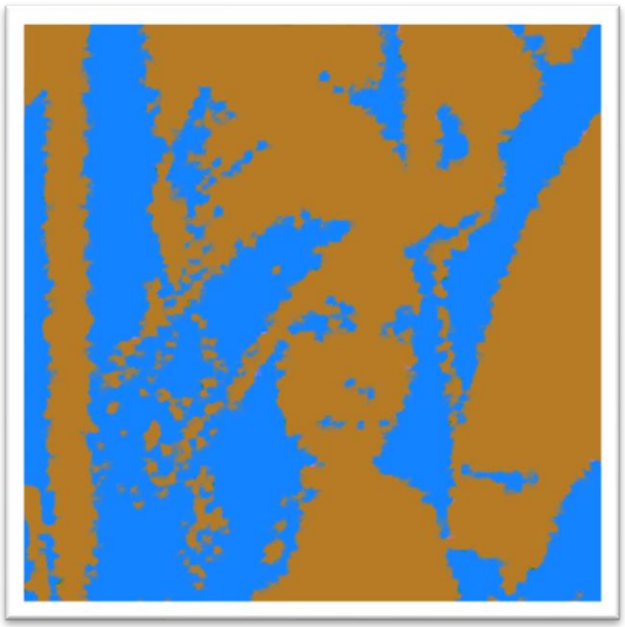


Photo Léna ACP

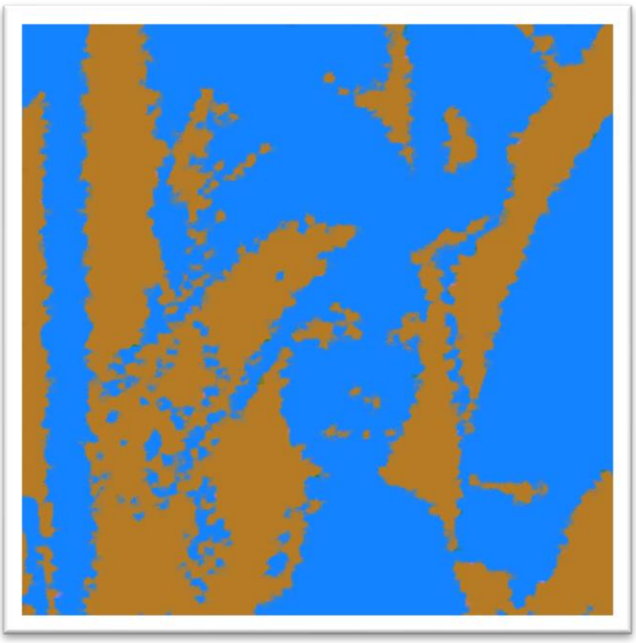


Photo Léna ACP pondéré

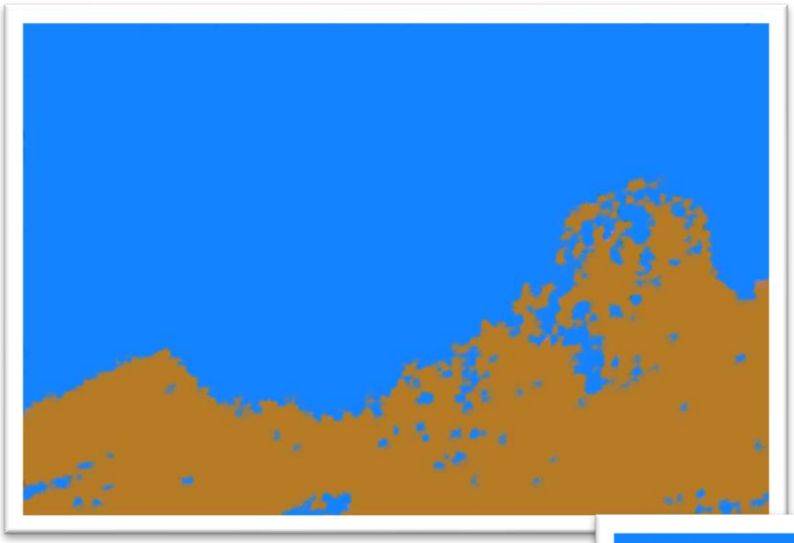


Photo montagne ACP

Photo montage ACP pondéré

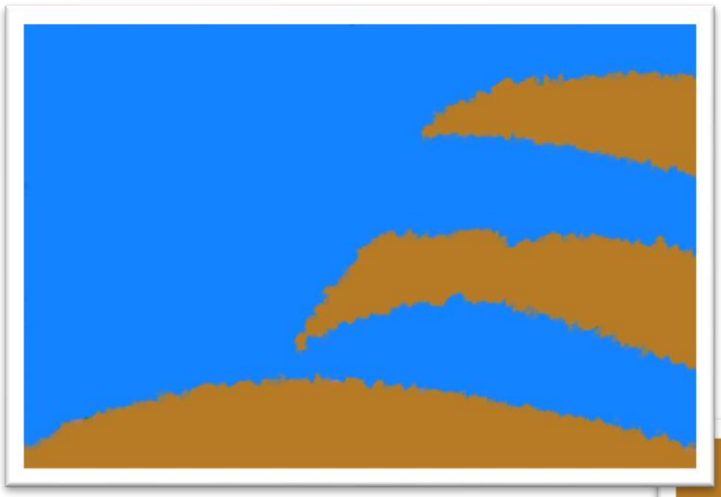


Photo feuille ACP

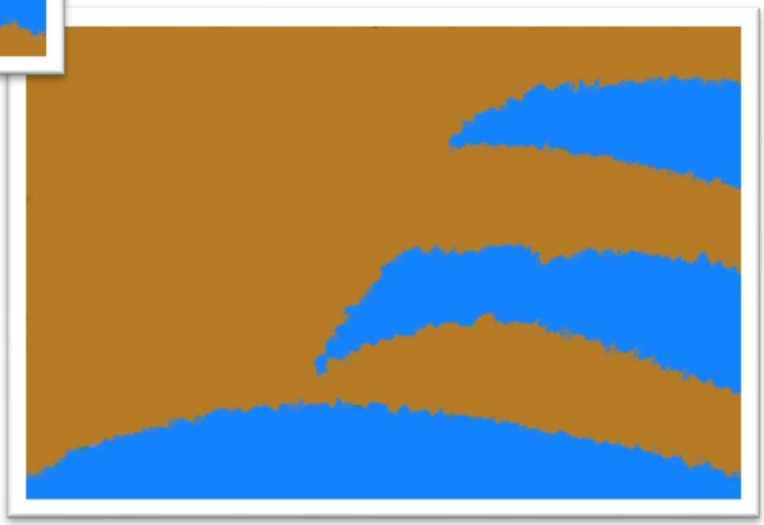


Photo feuille ACP pondéré

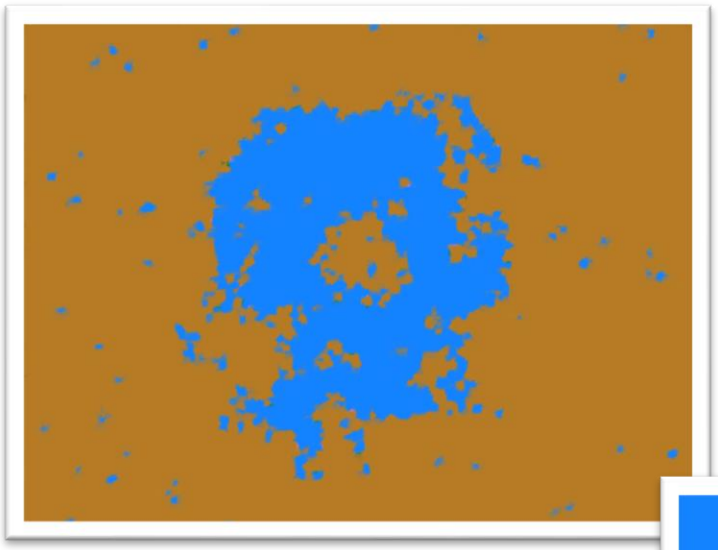


Photo nébuleuse ACP

Photo nébuleuse ACP pondéré

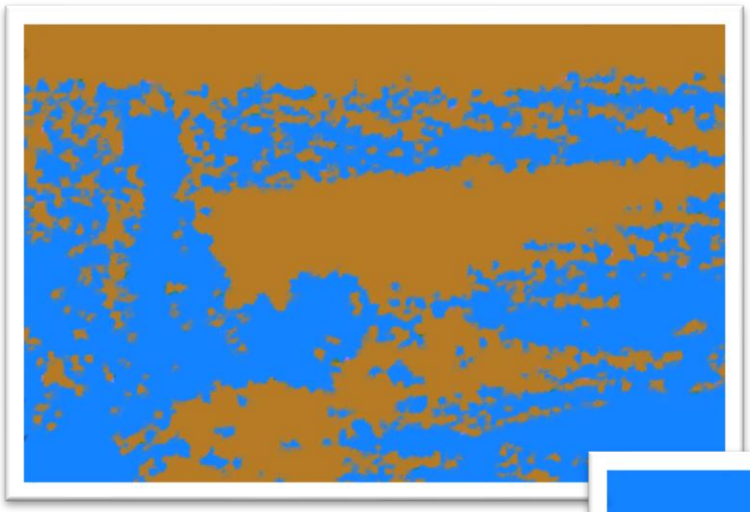


Photo paysage ACP

Photo paysage ACP pondéré

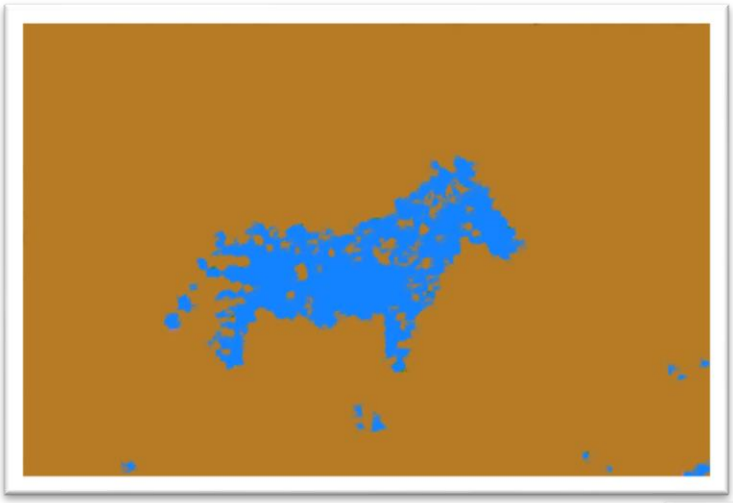
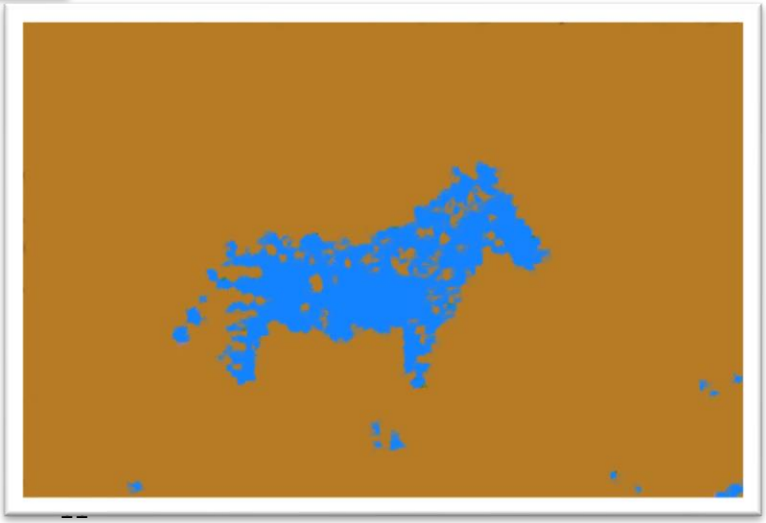


Photo zèbre ACP

Photo zèbre ACP pondéré



On peut regrouper nos résultats en deux groupes distincts. Le premier, où l'ACP pondéré renvoie exactement la même image que l'ACP classique, contient les photos du bâtiment, de l'immeuble, de la montagne et du zèbre. Dans l'autre groupe les couleurs sont inversées, on y met les photos de Léna, de la feuille, de la nébuleuse et du paysage.

Photos contenues dans le premier groupe

Photo de zèbre

Photo de bâtiment

Photo d'immeuble

Photo de montagne

Photos contenues dans le second groupe

Photo de feuille

Photo de Léna

Photo de paysage

Photo de nébuleuse

En étudiant les photos originales on note que ni la taille ni le poids de l'image n'influe sur la réaction de l'algorithme, pas d'homogénéité dans les groupes de ce côté-là. En revanche on remarque que dans le premier groupe il y a toujours plus de trois couleurs dominantes. Lorsque que l'algorithme choisis les pixels pour faire le remplissage de l'image il s'appuie sur leur couleur, celle-ci est déterminée car associée à un code chiffrés de 0 à 255. La couleur bleue ressortie par l'ACP correspond au ton clair de l'image, lorsque l'ACP pondéré inverse les couleurs c'est pour faire ressortir ces dernières.