Travaux pratiques

informatique commune

Transformations du photomaton et du boulanger

Durant cette séance nous allons nous intéresser à certaines transformations bijectives d'une image. Ce type de transformation déplace les points d'une image d'un endroit à un autre sans en ajouter ni en enlever aucun. De fait ceci revient à appliquer une permutation sur l'ensemble des points d'une image.
Une propriété remarquable de ces transformations bijectives est qu'elles reviennent toujours au point de départ après un nombre d'applications plus ou moins important : c'est une conséquence du fait que le groupe des permutations d'un ensemble fini est d'ordre fini donc tous ses éléments aussi. Par exemple, la transformée par symétrie de l'image par rapport à un axe vertical passant par son milieu est d'ordre 2, tandis que la transformée par rotation d'un quart de tour est d'ordre 4.

FIGURE 1 – Une image, sa transformée par symétrie verticale et sa transformée par rotation d'un quart de tour.

Nous allons nous intéresser plus particulièrement à deux transformations bijectives pour lesquelles le nombre d'étapes avant de voir réapparaître l'image initiale dépend des dimensions de celle-ci et peut être dans certains cas très grand.

Traitement d'image en PYTHON

Pour ce TP nous aurons besoin des deux bibliothèques suivantes : `numpy` et `matplotlib.pyplot` :

```
import numpy as np
import matplotlib.pyplot as plt
```

La seconde contient deux fonctions qui vont nous être utiles :

- la fonction `plt.imread` prend en argument une chaîne de caractère décrivant le chemin d'accès à un fichier image au format PNG et retourne un tableau NUMPY. Ce tableau a même dimension que l'image, et chacune de ses cases contient un triplet donnant les composantes RGB du pixel correspondant ;
- la fonction `plt.imshow` prend en argument un tableau NUMPY et affiche l'image associée à cette matrice (faire éventuellement suivre cette commande de l'instruction `plt.show()` si le mode interactif n'est pas activé).

Travail préliminaire
Récupérer à l'adresse `http://info-llg.fr/commun-mpsi/?a=tp` le fichier `picasso.png` et sauvegardez-le dans le répertoire courant. Exécutez ensuite le script suivant :

```
picasso = plt.imread('picasso.png')
plt.imshow(picasso)
```

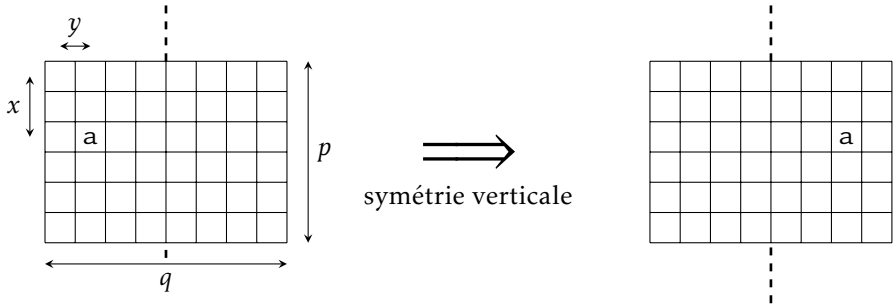
La première ligne convertit l'image `picasso.png` en une matrice NUMPY nommée `picasso` ; la seconde ligne affiche cette matrice-image à l'écran. Cette image, de taille 256×256, va nous servir de support pour expérimenter quelques transformations bijectives.

En guise d'échauffement, nous allons calculer les transformées de cette image, d'abord par une symétrie verticale puis par une rotation d'un quart de tour. Mais plutôt que de transformer l'image elle-même, nous allons calculer une nouvelle image contenant l'image transformée (c'est beaucoup plus simple). À partir d'une matrice-image vierge, il faudra recopier chacun des pixels de la matrice-image initiale à son nouvel emplacement dans la matrice-image vierge.

Question 1. Symétrie d'axe vertical

Dans le cas d'une symétrie d'axe vertical, l'image transformée à mêmes dimensions que l'image initiale. Pour créer la matrice-image vierge, on utilisera l'instruction `np.zeros_like(img)` qui prend en argument une matrice `img` et qui renvoie une matrice vierge de mêmes dimensions.

a) Étant donné un pixel a de coordonnées (x,y) dans l'image initiale, exprimer ses coordonnées (x',y') dans l'image résultat d'une symétrie d'axe vertical passant par le centre de l'image.



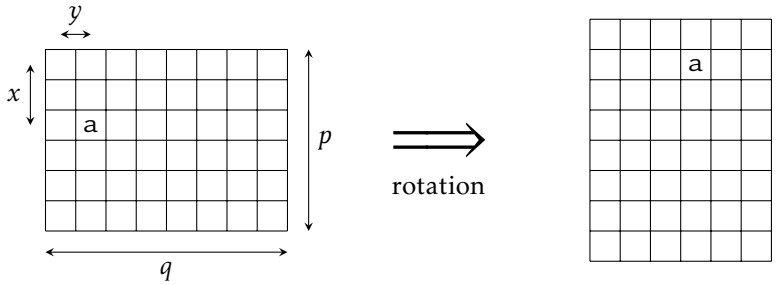
b) En déduire une fonction `symetrie(img)` qui prend en argument une matrice-image `img` et qui renvoie une nouvelle matrice-image résultat de la symétrie d'axe vertical. Les dimensions $p \times q$ de l'image initiale peuvent être calculées par les instructions `p = img.shape[0]` et `q = img.shape[1]`.

On testera cette fonction en faisant apparaître à l'écran le résultat de `symetrie(picasso)`.

Question 2. rotation d'un quart de tour

Pour une rotation d'un quart de tour, les dimensions de l'image résultat diffèrent de celles de l'image initiale lorsque cette dernière est rectangulaire. Pour créer la matrice-image vierge, on utilisera l'instruction `np.zeros((q, p))` pour créer une matrice vierge à q lignes et p colonnes.

a) Étant donné un pixel a de coordonnées (x,y) dans l'image initiale, exprimer ses coordonnées (x',y') dans l'image résultat d'une rotation d'un quart de tour vers la droite.

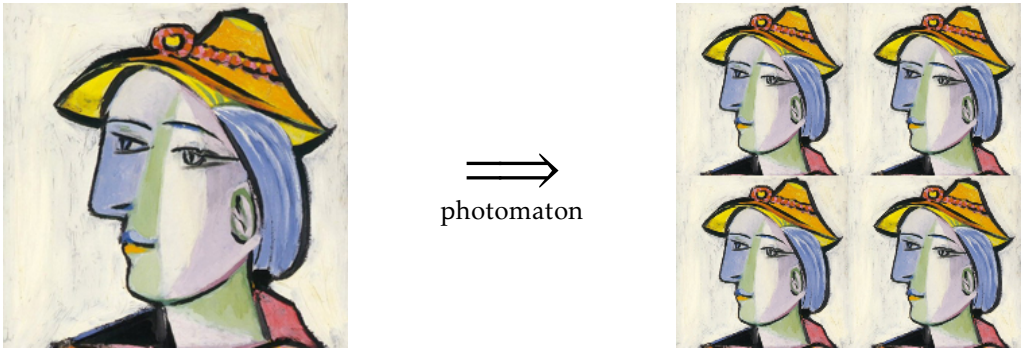


b) En déduire une fonction `rotation(img)` qui prend en argument une matrice-image `img` et qui renvoie une nouvelle matrice-image résultat de la rotation d'un quart de tour.

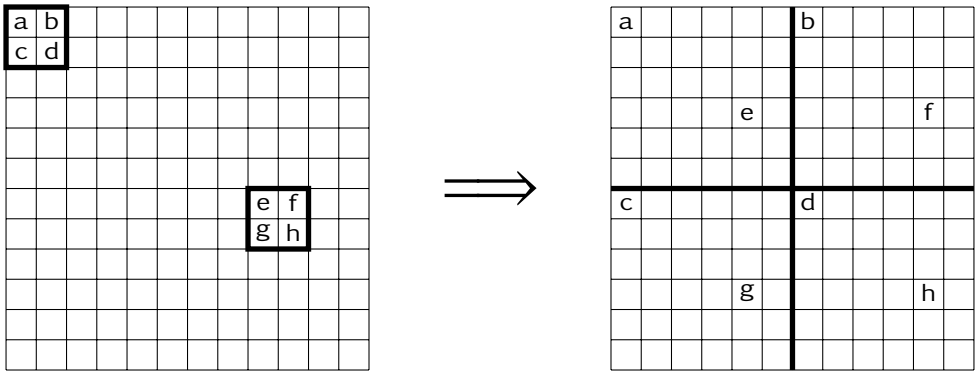
On testera cette fonction en faisant apparaître à l'écran le résultat de `rotation(picasso)` puis de ses itérées.

1. Transformation du photomaton

Cette transformation « réduit » la taille de l'image de moitié pour obtenir quatre morceaux analogues que l'on place en carré pour obtenir une image de même taille que l'image d'origine.



Pour que cette transformation soit bijective, on a découpé l'image initiale en paquets carrés de quatre pixels (2×2) puis pour chaque paquet carré de quatre pixels, on utilise celui en haut à gauche pour l'image réduite en haut à gauche, celui en haut à droite pour l'image réduite en haut à droite, etc.



On notera que pour que cette transformation soit définie il est nécessaire que les dimensions horizontale et verticale de l'image soient paires.

Question 3.

a) À partir des coordonnées (x, y) d'un pixel de l'image initiale, exprimer ses coordonnées (x', y') dans l'image résultat. Il sera nécessaire de distinguer quatre cas suivant la parité de x et de y.

b) En déduire une fonction `photomaton(img)` qui prend en argument une matrice-image et renvoie une nouvelle matrice-image résultant de la transformée du photomaton de l'image initiale.

Question 4. Rédiger un script Python pour visualiser les transformations successives de l'image `picasso.png` jusqu'à revenir à celle-ci. Quelle est la période de la transformation du photomaton pour cette image ?

Question 5. Plus généralement, si on considère une image de taille $p \times q$ il est possible de prouver que la période de la transformation du photomaton est le plus petit entier n pour lequel $p-1$ et $q-1$ divisent $2^n - 1$.
Rédiger une fonction `periode_photomaton(img)` qui prend en argument une matrice-image et retourne la période de la transformation du photomaton pour celle-ci.
Récupérer au même endroit que la précédente l'image `matisse.png`. Quelle est la période de sa transformation du photomaton ?

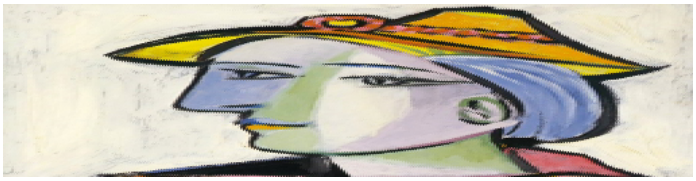
Question 6. On souhaite visualiser la 180e itération de l'image `matisse.png`, mais il n'est pas question de calculer les 179 images intermédiaires car le calcul serait trop long.

a) Rédiger une fonction `photomaton2(img, n)` qui prend en arguments une image et un entier n et qui retourne la n^e itérée de l'image par la transformation du photomaton, *en calculant celle-ci directement*.

b) Utiliser cette fonction pour visualiser la 180e itération de l'image `matisse.png`. Pouvez-vous expliquer le résultat obtenu ?

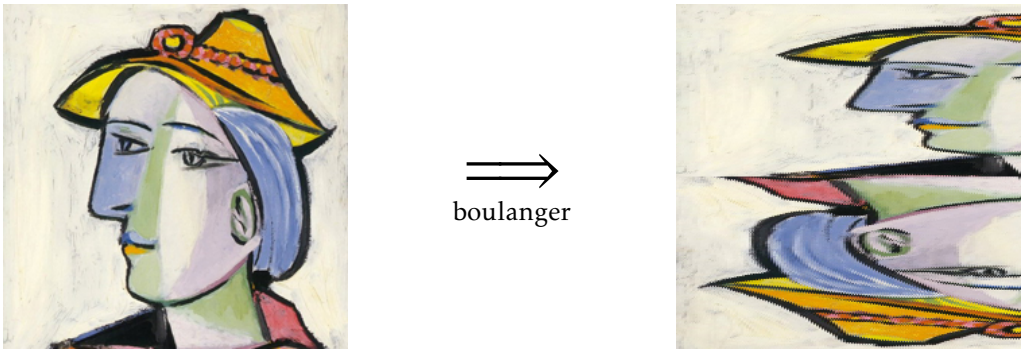
2. Transformation du boulanger

Cette transformation s'appelle ainsi car elle s'apparente au travail du boulanger qui réalise une pâte feuilletée. L'image est tout d'abord aplatie :

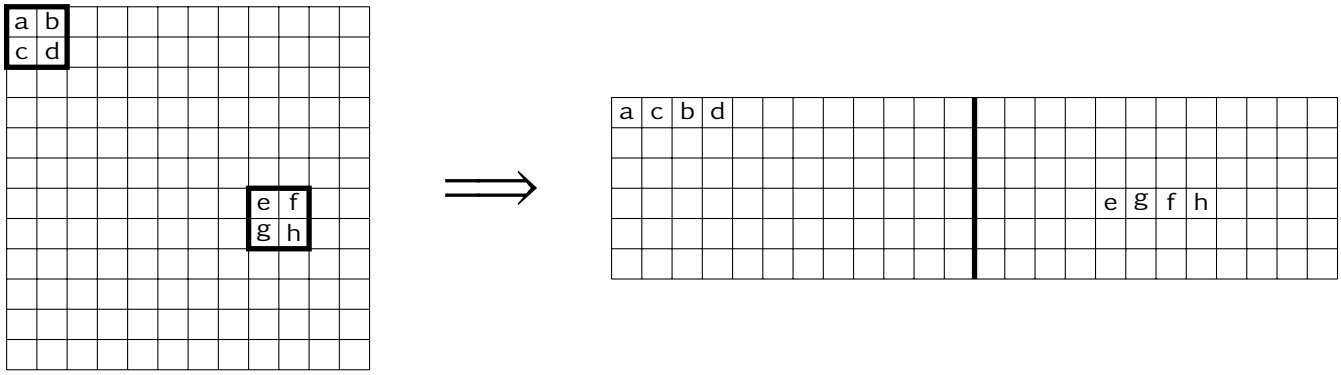


coupée en deux :

puis la partie droite est placée sous la partie gauche en la faisant tourner de 180°.



Pour que cette transformation soit bien définie, il est nécessaire là encore que les dimensions de l'images soient paires. L'image initiale est découpée en paquets carrés de quatre pixels puis ceux-ci sont entrelacés pour obtenir l'image intermédiaire aplatie :



Question 7. Rédiger une fonction `boulanger(img)` qui prend en argument une image et retourne la transformée du boulanger de celle-ci.

Question 8. Rédiger un script Python pour visualiser les transformations successives de l'image `picasso.png` jusqu'à revenir à celle-ci. Quelle est la période de la transformation du boulanger pour cette image ?

Et pour les plus rapides

Le calcul de la période de la transformation du boulanger est plus délicat à réaliser, et peut conduire à des valeurs très importantes. On l'obtient en déterminant tout d'abord la période de retour $r(x,y)$ de chaque pixel de coordonnées (x,y) puis en prenant le ppcm de toutes ces valeurs $r(x,y)$.

Question 9. Ecrire une fonction `periode_pixel(x, y, p, q)` qui prend en arguments les coordonnées (x,y) d'un pixel ainsi que la taille (p,q) de l'image auquel il appartient et qui retourne la période de retour de ce pixel à sa place initiale.

Question 10. En déduire une fonction `tableau_des_periodes(img)` qui calcule le tableau des périodes de tous les pixels d'une image. On optimisera le temps de calcul en observant que tous les pixels appartenant à une même orbite [1] ont une période identique.

Question 11. Rédiger enfin une fonction `periode_boulanger(img)` qui calcule la période de la transformation du boulanger d'une image. On pourra utiliser la fonction gcd du module `fractions` pour calculer le pgcd de deux entiers naturels. Quelle est la période de l'image `matisse.png` ?

Question 12. (difficile)
Rédiger une fonction `boulanger2(img, n)` qui calcule la n^e transformation du boulanger d'une image. Pour pouvoir utiliser de grandes valeurs de n sans que le temps d'attente soit rédhibitoire, il faudra calculer la position finale de chaque pixel (x,y) à l'aide du reste de la division euclidienne de n par $r(x,y)$, et traiter chaque pixel d'une même orbite dans le même temps.
Afficher l'itération de rang 67 911 de l'image `matisse.png`. Quel est le pourcentage de pixels situés à leur place initiale ?
Même question avec $n = 1\,496\,775\,000\,382\,576\,200$.

1. On appelle *orbite* d'un pixel l'ensemble des positions qu'il occupe durant les applications successives de la transformation du boulanger.