TP INFO 8 | Tableaux ndarray et images matricielles

Cette séance est dédiée au traitement numérique des images.

1. Le type ndarray de Python

À plusieurs reprises, nous avons manipulé des vecteurs et des matrices sous forme de listes (cf. par exemple un des exemples du TP sur les algorithmes gloutons). Cette implémentation est adaptée tant que le traitement des données matricielles n'est pas numérique.

Pour sommer des vecteurs de l'espace, le type `list` n'est pas vraiment adapté, la somme effectuant une concaténation et non une addition vectorielle.

Plus généralement, la structure de liste n'est pas adaptée au calcul matriciel.

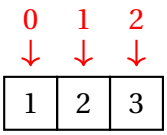
```
>>> v1,v2=[1,2,3],[3,2,1]
>>> v1+v2, 2*v1
[1,2,3,3,2,1], [1,2,3,1,2,3]
```

Dans ce cadre, il faut utiliser un autre type de Python appelé `ndarray`, littéralement *tableaux n-dimensionnels*.

```
>>> v1,v2=np.array([1,2,3]),np.array([3,2,1])
>>> v1+v2, 2*v1
array([4,4,4]),array([2,4,6])
```

Module à charger	numpy
Type d'un tableau	`ndarray`
Type des données	Tous les éléments doivent être du même type
Création d'un vecteur	`t=array([1,2,3])`
Création du 0 de $\mathbb{K}^n$	`t=zeros(n)`
Accès à la case d'indice i	`t[i]` (la numérotation commence à 0)
Taille du tableau t	`len(t)`

La numérotation des indices commence à zéro. On trouvera ci-contre la représentation du tableau `array([1,2,3])`.




```
>>> t=zeros(4)
>>> t
array([0.,0.,0.,0.])
>>> len(t)
4
```

```
>>> t[1],t[3]=-1,6
>>> t
array([0.,-1.,0.,6.])
>>> 2*t[1]+3*t[3]
16
```

Les tableaux `ndarray` ne sont pas dynamiques

Comme dans le cas des listes, on peut accéder aux coefficients d'un tableau en lecture et en écriture (i.e. on peut les modifier). En revanche, les opérations d'insertion et de suppression d'une case du tableau sont impossibles, sauf si l'on créée un nouveau tableau, de taille plus grande ou plus petite. Ceci nécessite la recopie en mémoire du tableau qui peut être coûteuse en temps si le tableau a très grande taille.

Plus généralement, le type ndarray permet de définir des matrices :

Création d'une matrice	`m=array([[1,2,3],[4,5,6]])`
Création du zéro de $\mathscr{M}_{n,p}(\mathbb{K})$	`m=zeros([n,p])`
Accès au coefficient d'indices `i,j`	`t[i,j]` ou `t[i][j]`
Dimensions	`t.shape()` (renvoie un tuple)
Ligne i de `t`	`t[i]` (sous forme unidimensionnelle)

Une matrice est donc définie au moyen de la liste de ses lignes, chaque ligne étant donnée sous forme d'une liste. Dans `t[i,j]`, l'indice `i` désigne la position dans la première liste, `j` la position dans la deuxième liste.

La numérotation commence à zéro.

	0 ↓	1 ↓	2 ↓
0 →	1	2	3
1 →	4	5	6

```
>>> t=array([[1,2,3],[4,5,6]])
>>> shape(t)
(2,3)
>>> t[0][1]
2
>>> t[1]
array([4,5,6])
>>> t[1][1]=-1
>>> t
array([[1,2,3],[4,-1,6]])
>>> t[1:,:2]
array([[4,-1]])
```

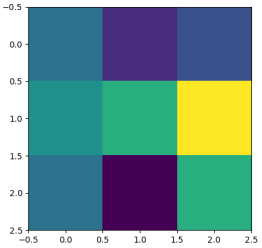
Attention aux copies de tableau, la même précaution que dans le cas des listes s'impose. Le slicing est valable dans ce cadre, pour chacun des indices. Comme pour les listes, il faudra utiliser la fonction `copy` de numpy afin d'effectuer des copies indépendantes de tableaux. On peut généraliser à des tableaux de dimension d quelconque avec la syntaxe `t[i1,i2,...,id]`.

2. Colorisation d'une matrice

Un mode très courant de représentation matricielle est la colorisation. On commence par choisir *une échelle de couleur* correspondant à l'intervalle $[m, M]$ où se trouvent les coefficients de la matrice : à m et M on associe deux couleurs et les valeurs intermédiaires sont associées à des dégradés. Ensuite, on représente la matrice comme un échiquier de mêmes dimensions, la case de coordonnées matricielles (i, j) étant remplie avec une couleur déduite de l'échelle.

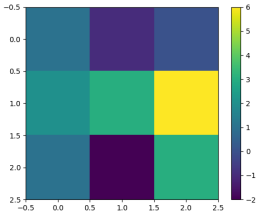
On peut afficher une matrice au moyen de la fonction `imshow`.

```
>>> A=np.array([[1,-1,0],[2,3,6],[1,-2,3]])
>>> plt.imshow(A)
```

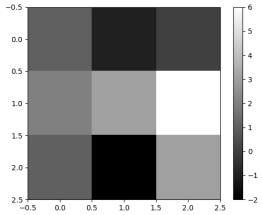


On notera l'affichage des coordonnées matricielles de part et d'autre de la figure obtenue. Ce mode d'affichage est intéressant pour des matrices de grandes tailles et permet de voir où sont localisés les coefficients les plus petits ou les grands par exemple.

L'échelle de couleur est choisie par défaut par cette fonction. On peut la modifier en choisissant par exemple une échelle de gris. On peut également demander l'affichage de l'échelle de couleur en utilisant la fonction `colorbar`.

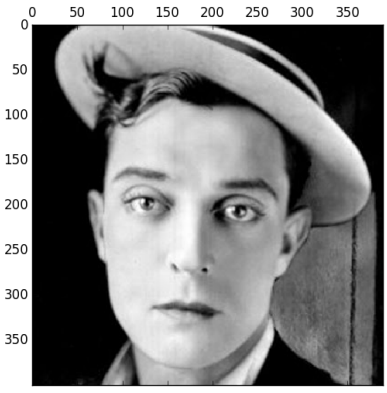


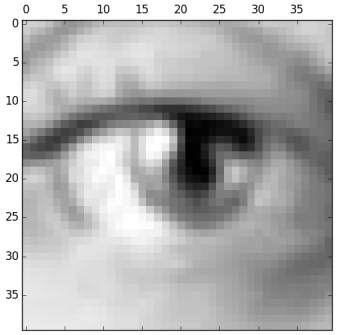
```
>>> plt.imshow(A)
>>> plt.colorbar()
>>> plt.imshow(A,cmap='gray')
>>> plt.colorbar()
```

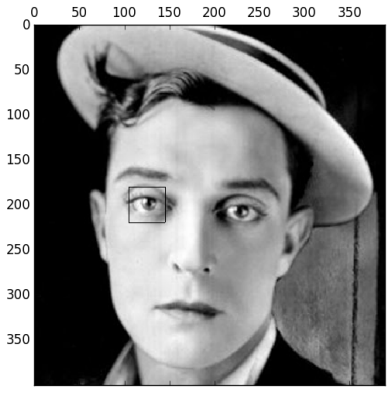


3. Les fichiers images

Il existe deux grands types d'images numériques : les images vectorielles et les images matricielles. Une vectorielle est définie par des objets graphiques élémentaires (comme des segments, des cercles, etc.) alors qu'une image matricielle est décrite par une matrice de pixels. Nous n'étudierons aujourd'hui que ce second type de codage. Il est fondé sur l'approximation d'une image par un quadrillage très finement maillé de cases monochromatiques, on pourra la représenter au moyen d'une matrice colorisée. Chaque case est appelé *un pixel*[1].







1. Abréviation de « *picture element* ».

3.1. Les images matricielles (bitmap)

Définition et résolution d'une image

⇛ La *définition d'une image matricielle* est le nombre de pixels qui composent l'image en hauteur (nombre de ligne de la matrice) et en largeur (nombre de colonnes) : 100 pixels par 300 pixels par exemple, ce qu'on écrit 200×450.

⇛ *La résolution d'une image* est définie par un nombre de pixels par unité de longueur de la structure à numériser (classiquement en ppp[2]). Ce paramètre est défini lors de la numérisation (passage de l'image sous forme binaire), et dépend principalement des caractéristiques du matériel utilisé lors de la numérisation. Plus le nombre de pixels par unité de longueur de la structure à numériser est élevé, plus la quantité d'information qui décrit cette structure est importante et plus la résolution est élevée. La résolution d'une image numérique définit le degré de détail de l'image. Ainsi, plus la résolution est élevée, meilleure est la restitution.

Cependant, pour une même dimension d'image, plus la résolution est élevée, plus le nombre de pixels composant l'image est grand. Le nombre de pixels est proportionnel au carré de la résolution, étant donné le caractère bidimensionnel de l'image : si la résolution est multipliée par deux, le nombre de pixels est multiplié par quatre. Augmenter la résolution peut entraîner des temps de visualisation et d'impression plus longs, et conduire à une taille trop importante du fichier contenant l'image et à de la place excessive occupée en mémoire.

Colorisation

⇛ *Codage par niveaux de gris.* Commençons par le cas d'une image en noir et blanc. La couleur attribuée ici à chaque pixel représente le niveau de l'intensité lumineuse, codé généralement sur un octet (256 valeurs). Par convention, la valeur zéro représente le noir (intensité lumineuse nulle) et la valeur 255 le blanc (intensité lumineuse maximale). Ces images sont représentées par une seule matrice, ie un tableau à deux dimensions.

⇛ *Images en couleurs.* Il existe plusieurs modes de codage informatique des couleurs, le plus utilisé pour le maniement des images est l'espace colorimétrique rouge, vert, bleu (RVB ou RGB - red green blue). Cet espace est basé sur une synthèse additive des couleurs, c'est-à-dire que le mélange des trois composantes R, V, et B à leur valeur maximum donne du blanc, à l'instar de la lumière.

Le mélange de ces trois couleurs à des proportions diverses permet de reproduire à l'écran une part importante du spectre visible, sans avoir à spécifier une multitude de fréquences lumineuses.

Chaque composante est définie par une intensité lunimeuse (256 valeurs possibles). L'image ci-contre résulte de la synthèse additive des niveaux RGB donnés ci-dessous. Ces images sont représentées par trois matrices, ie un tableau à trois dimensions. À ces trois matrices s'ajoutent souvent une quatrième, la brillance (*glossiness*).

2. Points par pouces, dpi (dot per inch in english).



3.2. Les différentes normes en vigueur

Il existe plusieurs normes (ou encore formats) d'images : `png`, `jpeg`, `gif`, etc. Ces aspects n'étant pas au programme, nous nous contenterons d'un bref aperçu.

Le choix d'une norme correspond à une manière de stocher le tableau correspondant à l'image. Afin de gagner en taille, les données subissent des compressions :

⇛ *une compression sans perte :* les données sont codées de manière bijective (cas de `png` par exemple) ;

⇛ *une compression avec perte :* certains éléments sont perdus (cas de `jpeg`).

3.3. Manipulation des images sous Python

Le module `PIL` permet de manipuler de manière fine les images. Nous nous contenterons d'utiliser les possibilités, plus restreintes, offertes par le module `matplotlib`. Ce module permet de manipuler des images au format `png`[3]. La fonction `imread` renvoie un tableau à trois dimensions.

```
>>> tab=plt.imread('/home/kaczmarek/pyzo2015a/hokusai.png')
>>> plt.imshow(tab)
```

```
>>> np.shape(tab)
(404, 600, 3)
>>> tab[0,0,1]
0.92941177
```

3. Ce n'est pas un obstacle en pratique : il suffit d'utiliser un logiciel quelconque de traitement d'image (*gimp*, *photoshop*, etc.) pour effectuer une conversion, par exemple de `jpeg` vers `png`.

Dans notre exemple, le tableau obtenu est tridimensionnel car le codage RGB nécessite trois matrices bidimensionelles. On peut alors transformer l'image en modifiant le tableau obtenu :

```
tab=plt.imread('/home/kaczmarek/pyzo2015a/hokusai.png')
for i in range(0,261):
    tab[i,391,0]=.25
    tab[i,391,1]=.25
    tab[i,391,2]=.25
plt.imshow(tab)
plt.axis('off')
```

Images numériques sous Matplotlib

Utilisation de `Matplotlib` pour le format `png` :

- *Récupération du tableau numérique :* `tab=plt.imread('chemin')`
- *Affichage d'un tableau numérique dans une fenêtre :* `plt.imshow(tab)`
  On peut obtenir une conversion en noir et blanc par `plt.imshow(tab,cmap='gray')`; `cmap` signifiant *colormap*.
- *Sauvegarde d'un tableau sous la forme d'un fichier png :* `plt.imsave('chemin',tab)`

Appel d'une fonction avec un chemin

Pyzo ne considère pas a priori le dossier courant comme dossier de travail. Il convient donc d'écrire le chemin absolu du fichier avec des / entre les dossiers (et non pas des \). Il y a d'autres manières de faire, doubler tous les ou simplement rajouter la lettre r devant la chaîne de caractères pour ne pas prendre en compte les caractères d'échappement.

4. Exercices

Vous testerez vos fonctions au moyen des images `ck.png` et `seagulls.png` disponibles sur le site suivant

`https://laurentkaczmarek.fr/info.html`

Ces deux images sont en noir et blanc mais contiennent quatre matrices. Après extraction au moyen de `imread`, on pourra récupérer la première des quatre matrices au moyen de l'instruction

```
t=t[:,:,0]
```

C'est cette matrice qui sera l'argument de toutes les fonctions du TP. On obtiendra alors la taille $p \times q$ de l'image (p lignes et q colonnes) au moyen de

```
p,q=np.shape(t)
```

1 — *Transformations élémentaires* f

1. Écrire une fonction `negatif` prenant en argument un tableau et renvoyant son négatif.


2. Écrire une fonction `symetrique` prenant en argument un tableau et renvoyant son symétrique par rapport à son axe vertical médian.


3. Écrire une fonction `rotation` prenant en argument un tableau et qui renvoie le tableau tourné de $+\frac{\pi}{2}$.


4. Écrire une fonction `contraste` prenant en argument un tableau et renvoyant un tableau où le contraste est plus prononcé.


2 *Introduction au filtrage linéaire des images* f

Un filtre linéaire est une application qui à une matrice A associe une matrice image A′ dont les coefficients sont linéaires en ceux de A. Nous n'étudierons que des filtrages obtenus au moyen d'un « *filtre* » 3×3. On considère une matrice $A \in \mathscr{M}_{p,q}(\mathbb{R})$ et $F \in \mathscr{M}_3(\mathbb{R})$.

α	β	γ
δ	ϵ	ι
κ	λ	μ

		P	Q	R		
		S	T	U		
		V	W	X		

$$T' = \alpha P + \beta Q + \gamma R + \delta S + \epsilon T + \iota U + \kappa V + \lambda W + \mu X$$

La matrice image A′ est définie de la manière suivante : la première et la dernière lignes de A′ sont celles de A, de même pour les colonnes. L'image T′ d'un coefficient T, s'exprime en fonction des coefficients de A au moyen du filtre F comme la figure ci-dessus l'indique. On laissera la première ligne, la première colonne, la dernière ligne et la dernière colonne iinchangées.

1. Écrire une fonction `filtrage` qui prend en argument un tableau t et un filtre F au format 3×3 et qui renvoie le tableau filtré t'.

2. Visualiser l'image obtenue en appliquant à l'image `seagulls.png` le filtre de cache $\frac{1}{3}\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix}$.

   Interpréter le résultat obtenu.

3. Même question avec les filtres de Sobel : $\frac{1}{4}\begin{pmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{pmatrix}$ et $\frac{1}{4}\begin{pmatrix}-1&0&1\\-2&0&2\\-1&2&1\end{pmatrix}$.

On pourra utiliser librement les syntaxes suivantes :

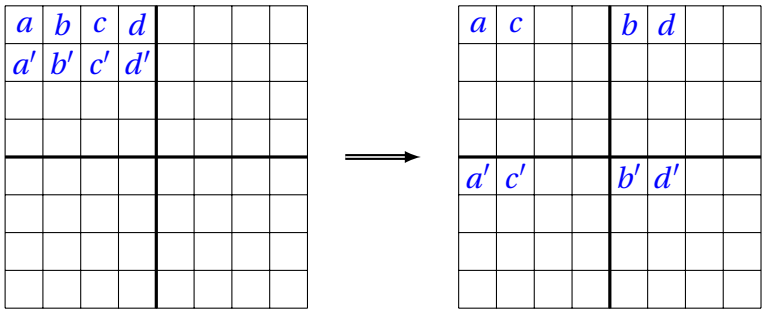
⇛ Le produit `t1*t2` de deux tableaux `ndarray` de mêmes tailles produit le tableau dont les coefficients sont les `t1[i][j]*t2[i][j]`.

⇛ La somme des coefficients d'un tableau bidimensionnel `t` s'obtient par `sum(sum(t))`.

3 *La transformation du Photomaton ff*

La transformation du photomaton « réduit » la taille d'une 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. On découpe l'image initiale en paquets carrés de 2×2 pixels 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.

Cette transformation est bijective. Chaque pixel est envoyé sur un autre selon le principe suivant :



On considère une image au format $p \times q$ avec p et q pairs.

1. Écrire une fonction `imagePixelPhotomaton(i,j,p,q)` qui renvoie sous la forme d'une liste les coordonnées matricielles (i', j') *dans l'image transformée* du pixel de coordonnées matricielles (i, j) *dans l'image initiale*.
2. En déduire une fonction `photomaton` qui prend en argument un tableau et renvoie la transformée du photomaton de celui-ci.
3. Écrire un script Python pour visualiser les transformations successives de l'image `seagulls.png` jusqu'à revenir à celle-ci. Quelle est la période de la transformation du photomaton pour cette image ?
4. 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` qui prend en argument un tableau codant une image et renvoie la période de la transformation du photomaton pour celle-ci.
5. Quelle est la période de la transformée du photomaton de l'image `ck.png` ?

4 — *Coloriage ff*

On considère une image en noir et blanc codée par un tableau `numpy` de taille 100×100. On se donne un pixel de coordonnées (i, j) puis on applique l'algorithme suivant :

a. Si le pixel est noir, alors on ne fait rien.

b. S'il est blanc, alors on le colorie en noir puis, pour chacun des voisins du pixel (gauche, droite, haut et bas – en cas d'existence), on revient à l'étape a).

1. Décrire ce que réalise cet algorithme.
2. Écrire une fonction récursive `coloriage(i,j)` prenant en argument un couple d'indices valables (i, j) et effectuant « *le coloriage* » de l'image selon cet algorithme à partir du pixel (i, j).
3. Créer une image contenant un carré en noir puis tester la fonction `coloriage` pour un pixel (i, j) à l'intérieur puis à l'extérieur du carré.

5. Indications

1

Pour augmenter le contraste, il suffit d'appliquer aux coefficients une même fonction $f : [0,1] \to [0,1]$ croissante qui varie brusquement de 0 à 1 au voisinage de $\frac{1}{2}$.

2

On peut utiliser le slicing pour extraire une sous-matrice de taille trois centrée aux indices (i, j) via la syntaxe `t[i-1:i+2,j-1:j+1]`.

3

Les périodes demandées valent respectivement 8 et 4140.

4

L'algorithme colorie en noir la composante connexe du pixel (i, j).