Chapitre 8
Manipulation d'images

Simon Dauguet
simon.dauguet@gmail.com

13 mars 2025

Table des matières

1 Matrices et opérations matricielles

Une matrice $M \in \mathcal{M}_{n,m}$ (où $(n, m) \in \mathbb{N}^{*2}$) peut être représentée par un tableau contenant n tableaux (les n lignes) contenant m valeurs (les m colonnes) (entières ou flottantes). Dans toute la suite, on supposera qu'une matrice est un tableau de tableaux où tous les tableaux intérieurs ont même dimension.

1.1 Rappel sur les opérations usuelles

On rappelle les définitions de la somme et du produit matriciel pour n et m entiers non nuls fixés :

$$\begin{cases} \forall (M,N) \in \mathcal{M}_{n,m}(\mathbb{K})^2,\ \forall (i,j) \in [\![1,n]\!] \times [\![1,m]\!],\ (M+N)i,j = M_{i,j} + N_{i,j} \\ \forall (M,N) \in \mathcal{M}_{n,m}(\mathbb{K}) \times \mathcal{M}_{m,p}(\mathbb{K}),\ \forall (i,j) \in [\![1,n]\!] \times [\![1,p]\!],\ (MN)_{i,j} = \sum_{k=1}^{m} M_{i,k} N_{k,j}. \end{cases}$$

Remarque :
⚠On rappelle qu'en python, les indices commences à 0. Donc on devra prendre $(i,j) \in [\![0,n[\![\times [\![0,m[\![$.

Exemple 1.1 :
$mat = \begin{pmatrix} 4 & 5 & 6 \\ 0 & 1 & 8 \end{pmatrix} \in \mathcal{M}_{2,3}$ peut être définie sous python via l'instruction : `mat = [[4,5,6],[0,1,8]]`.
On alors les appels suivants :

```
>>> len(mat)      # nombre de lignes
2
>>> mat[0]        # Première ligne
[4,5,6]
>>> mat[0][2]     # Troisième élément de la première ligne
6
>>> mat[1][0]     # Premier élément de la deuxième ligne
0
>>> mat[1][2]     # Troisième élément de la deuxième ligne
8
```

Exercice 1 :
Définir en python la matrice $mat = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 6 & 7 & 8 & 9 \end{pmatrix}$.

1. Que vaut `len(mat)` ?
2. Que vaut `len(mat[0])` ?
3. Que vaut `mat[1]` ?
4. Que vaut `mat[1][2]` ?
5. Comment obtenir 2 ?
6. Comment obtenir 8 ?

Exercice 2 :
Proposer une fonction `somMatricielle(m1:list, m2:list) -> list`, qui renvoie la matrice somme de `m1` et `m2`. Si les dimensions de `m1` et `m2` sont incompatibles, un tableau vide est renvoyé. Préciser la complexité temporelle.

Exercice 3 :
Proposer une fonction `prodMatricielle(m1:list, m2:list) -> list`, qui renvoie la matrice produit

de `m1` et `m2`. On vérifiera que les matrices `m1` et `m2` sont carrées de même dimension, sinon un tableau vide est renvoyé. Préciser la complexité temporelle.

1.2 Copies de matrices

La gestion des matrices en python peut être un peu délicate. En particulier, il faut faire attention à la façon dont on fait des copies d'une matrice. Si on y prend pas grade, une matrice peut devenir une liste de plusieurs fois la même ligne. Et donc, changer une ligne va les changer toutes (c'est la même ligne qui apparaît plusieurs fois).

Pour s'assurer de créer une nouvelle matrice d'une taille donnée dont tous les coefficients sont indépendants les uns des autres, il faut utiliser la commende :

```
>>> m = [[0]*nbCol for k in range(0,nbLignes)]
```

Exemple 1.2 :

```
>>> m = [[0]*3]*2
>>> m
[[0,0,0],[0,0,0]]
>>> m[0][0] = 1
>>> m
[[1,0,0],[1,0,0]]
>>> mat = [[0]*3 for k in range(2)]
>>> mat[0][0] = 1
>>> mat
[[1,0,0],[0,0,0]]
>>> Mat = mat
>>> Mat[1][1] = 2
>>> mat
[[1,0,0],[0,2,0]]
>>> Mat = [mat[k] for k in range(len(mat))]
>>> Mat[1][1] = 3
>>> mat
[[1,0,0],[0,3,0]]
>>> newMat = [[0]*len(mat[0]) for k in range(len(mat))]
>>> newMat[0][1] = -25
>>> newMat
[[0,-25,0],[0,0,0]]
>>> mat
[[1,0,0],[0,3,0]]
>>> M = m[:]
>>> M[0][0] = 10
>>> m
[[10,0,0],[0,0,0]]
```

1.3 Autres opérations matricielles

1.3.1 Produit de Hadamard

Définition 1.1 (Produit de Hadamard) :
Le produit de Hadamard de deux matrices de même taille est définit comme le produit "naïf" éléments par éléments. Plus précisément, si $A, B \in \mathcal{M}_{n,p}(\mathbb{K})$, on définit le produit de Hadamard de A et B, noté $A \cdot B \in \mathcal{M}_{n,p}(\mathbb{K})$ par :

$$\forall i \in \{1, \dots, n\},\ \forall j \in \{1, \dots, p\},\ [A \cdot B]_{i,j} = [A]_{i,j} \times [B]_{i,j}$$

Exemple 1.3 :
Avec $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$ et $B = \begin{pmatrix} 1 & 1 \\ -1 & -1 \\ 1 & -1 \end{pmatrix}$, le produit de Hadamard est

$$A \cdot B = \begin{pmatrix} 1 & 2 \\ -3 & -4 \\ 5 & -6 \end{pmatrix}$$

Exercice 4 :
Faire une fonction `Hadarmard(mat1:list, mat2:list) -> list` qui effectue le produit de Hadamard des deux matrices, après avoir fait les vérifications sur la taille des matrices.

1.3.2 Produit scalaire canonique

Cette partie sera étudié plus en détails dans le chapitre de maths sur les espaces préhilbertiens.

Définition 1.2 (Produit scalaire canonique de deux matrices) :
Soit $A, B \in \mathcal{M}_{n,p}(\mathbb{K})$ deux matrices. On définit le produit scalaire canonique de A et B par

$$\langle A|B\rangle = \sum_{i=1}^{n}\sum_{j=1}^{p}[A]_{i,j}[B]_{i,j}$$

Exemple 1.4 :
Avec $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ et $B = \begin{pmatrix} -1 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix}$, alors

$$\langle A|B\rangle = -1 + 2 - 3 + 4 - 5 + 6 = 3$$

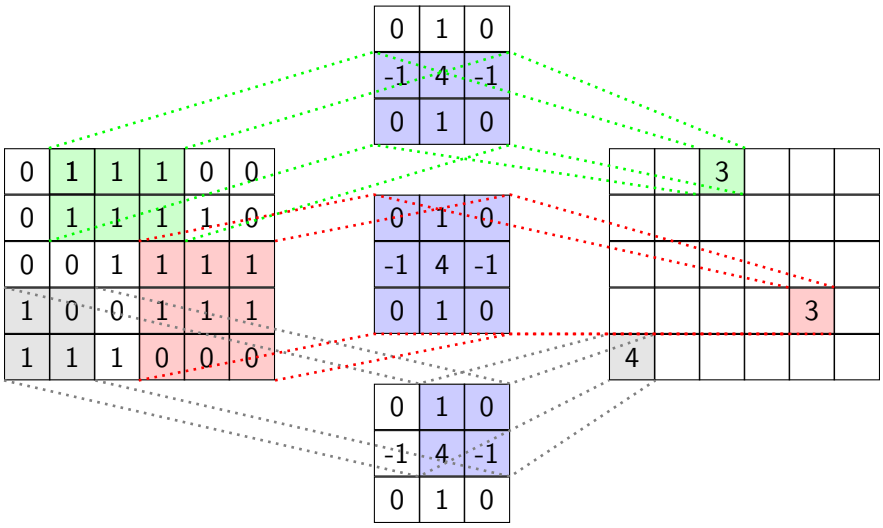
Exercice 5 :
Faire une fonction `prodScal(mat1:list, mat2:list) -> float` qui effectue le produit scalaire canonique des deux matrices, après avoir fait les vérification d'usage sur les tailles des matrices.

1.3.3 Produit de convolution matriciel

Définition 1.3 (Produit de convolution matriciel) :
L'opérateur de convolution (noté *) entre une matrice de convolution (ou de filtrage) et une matrice image consistera à renvoyer une matrice dont chaque coefficient i, j a été défini par le produit scalaire canonique entre deux sous-matrices centrées en i, j.

Il faut donc un coefficient "central" pour la deuxième matrice, quoi doit donc être carrée de taille impaire.

Exemple 1.5 :
Pour $mat = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$ et $conv = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 4 & -1 \\ 0 & 1 & 0 \end{pmatrix}$, la convolution $mat * conv$ se calculera :



Remarque :
Sur les bords de l'image, le produit de convolution est tronqué, menant éventuellement à des résultats incohérent. On pourrait laisser inchangés les pixels sur les bords pour lever le problème de l'incohérence.

Exercice 6 (Produit de convolution) :
Proposer une fonction `prodConv(mat1:list, mat2:list) -> list` qui effectue le produit de convolution des deux matrices (avec `mat2` de taille 3×3) en faisant au bords.

Remarque :
En mathématiques, le produit de convolution le plus classique est défini pour des fonctions continues. Si f et g sont deux fonctions continues sur $\mathbb{R}$ (et vérifiant d'autres hypothèses supplémentaires que nous ne discuterons pas ici et qui sont hors programme de maths en CPGE), on peut définir le produit de convolution de f et g, noté $f * g$ par

$$\forall x \in \mathbb{R},\ (f * g)(x) = \int_{-\infty}^{+\infty} f(x-t)g(t)dt.$$

Une version discrète existe pour les suites :

$$\forall n \in \mathbb{Z},\ (u * v)_n = \sum_{k=-\infty}^{+\infty} u_{n-k}v_k.$$

Le produit de convolution continue est notamment utile pour définir les transformées de Laplace utilisées en SI.

1.4 Manipulation hors programme : package `numpy`

La bibliothèque `numpy` hors programme (mais utilisé par Centrale) fournit un nouveau type d'objet : les `ndarray`. Ce sont de "véritables" matrices où le produit est bien définie (*i.e.* le produit de deux `ndarray` correspond bien au produit matriciel mathématique).

Il existe aussi le type `matrix` naturellement dans python, avec des opérations matricielle déjà définie. En revanche, le produit matriciel n'est pas le bon : c'est le produit terme à terme. Une méthode existe pour faire le produit matriciel mathématique. Cependant, le programme ne semble pas aller dans cette direction et semble demander à ce que tout soit fait "à la main". Nous n'utiliserons donc pas de `matrix` et seulement des listes de listes.

!!! ATTENTION !!!

⚠ Le package `numpy` est officiellement hors-programme. Ne l'utiliser que si le sujet vous l'y autorise **explicitement**.

Commandes de bases de **numpy**

Commande Python	Descriptions
`import numpy as np`	Importe la librairie **numpy** sous le nom **np**
`np.array(tab:list)`	Convertit une liste **tab** en tableau **ndarray**
`np.zeros(n:int)`	Crée un vecteur nul (type **ndarray**) de longueur n
`np.zeros((n:int, m:int))`	Crée une matrice nulle (type **ndarray**) de dimension $n \times m$
`np.ones((n:int, m:int))`	Crée une matrice (type **ndarray**) de dimension $n \times m$ emplie de flottant 1.0
`np.ones((n:int, m:int), int)`	Crée une matrice de dimension $n \times m$ emplie d'entiers 1
`np.linspace(a:float, b:float, n:int)`	Crée un **ndarray** contenant une subdivision régulière de $[a, b]$ de longueur n
`np.arange(a:float, b:float, h:float)`	Crée un **ndarray** des points entre a et b en ajoutant le pas h à caque fois : $a, a+h, a+2h, \ldots$
`np.cos([0,.2,.4,.6])`	Renvoie un **ndarray** contenant les images par $\cos()$ des valeurs
On suppose que **A** *et* **B** *sont des matrices* **numpy**	
`A.size`	Renvoie le nombre d'éléments de A (noter l'absence de parenthèses)
`A.shape`	Renvoie les dimensions de A (noter l'absence de parenthèses)
`3*A`	Multiplie tous les coefficients de **A** par 3
`A+B`	Matrice résultat de l'addition matricielle $A + B$
`A*B`	Matrice résultat des multiplications terme à terme (produit d'Hadamard)
`A**4`	Matrice résultat de la puissance 4 de chaque coefficient
`A.dot(B)`	Matrice résultat de la multiplication matricielle $A \times B$
`A[i][j]` ou `A[i,j]`	Coefficient $a_{i,j}$
`A[:,j]`	Colonne j de la matrice A
`A[1:3][4:7]` ou `A[1:3,4:7]`	Sous-matrice composée des lignes 1-2 et colonnes 4-5-6 de A

Remarque :
`B=A` n'effectue pas de copie de la matrice **A**, il faut faire `B=A.copy()`.

Le package **numpy** contient aussi un sous-module **linalg** de manipulations matriciel :

Commandes d'algèbre linéaire

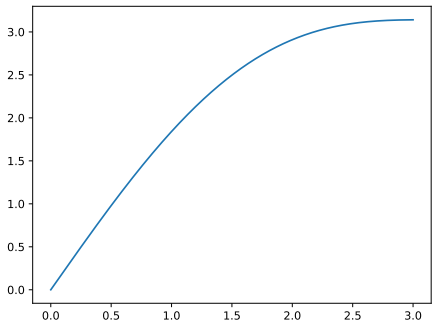
Commandes	Description
`numpy.dot(M,N)`	Produit matriciel de deux tableaux ou deux matrices ou deux listes de listes
`M.transpose()`	Transposée de la matrice ou tableau ou liste de listes `M`
`numpy.vdot(v,w)`	Produit scalaire des vecteurs (tableau ou matrice ou listes) `v` et `w`
`numpy.cross(v,w)`	Produit vectoriel des vecteurs (matrices ou tableaux ou listes) `v` et `w` de dimension 2 ou 3 seulement
`numpy.linalg.det(M)`	Déterminant de la matrice ou du tableau ou de la liste de listes `M` carrée
`numpy.linalg.inv(M)`	Inverse de la matrice ou tableau ou liste de listes `M`
`numpy.linalg.matrix_rank(M)`	Rang de la matrice ou tableau ou liste de listes `M`
`numpy.linalg.solve(A,B)`	Solution du système linéaire `AX=B` où `A` est une matrice ou tableau ou liste de listes et `B` est une matrice ou tableau ou liste de listes.

La liste n'est pas exhaustive. On pourra rajouter certaines fonctions que vous verrez en mathématiques en deuxième année (comme les valeurs propres, vecteurs propres etc). Tout ceci est caché dans le sous-paquets `linalg` de `numpy`. D'une façons générale, pour faire de l'algèbre linéaire, se référer à l'aide du sous-paquet `linalg`.

Remarque :
Le package `numpy` est surtout utile pour la façon dont il est fabriqué : il est fait pour simplifier la vie de l'usager.

Exemple 1.6 :

```
>>> import numpy as np
>>> f=lambda x : x+np.sin(x)
>>> X=np.linspace(0,3,1001)
>>> plt.plot(X,f(X))
[<matplotlib.lines.Line2D object at 0x720887ee3430
>>> plt.show()
```

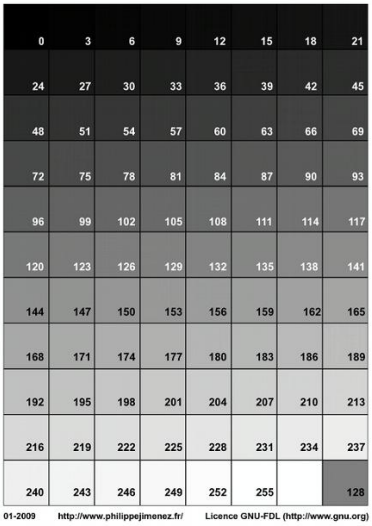


2 Manipulations d'images

2.1 Images en Python

Une image utilise de multiples méthodes d'encodage (bmp, jpg, png, ...), la plus simple consiste à considérer une image comme un tableau double entrée dans lequel chaque pixel est codé, *i.e.* une matrice de pixels.

- un pixel en nuance de gris est encodé sur un octet (0 représentant le noir et 255 le blanc)
- un pixel de couleurs est encodé par un triplet d'octets (Red, Green, Blue). Chaque valeur donne la quantité de rouge, de vert et de bleu dans le pixel.

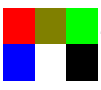


Dans ce cours, nous ne verrons que les images encodées en bitmap, et sauf exception, nos images seront des nuances de gris sur un octet.

Remarquons de suite que l'axe des ordonnées d'une image est inversée et pointe en bas afin d'être cohérent avec les lignes de la matrice la représentant.

Exemple 2.1 :

La matrice `m = [[ 255, 140, 140 ], [ 0, 128, 255 ]]` encode l'image  .

La matrice `m = [[ (255,0,0), (128,128,0), (0,240,0) ], [ (0,0,128), (255,255,255), (0,0,0) ]]` encode l'image  .

Remarque (Conversion d'images en tableaux) :
Les images ne sont pas directement des tableaux de nombres. Il est nécessaire de transformer les images en tableaux de nombres. La conversion n'est pas au programme. Seul la manipulation d'images vu comme des tableaux de nombres (et leurs effet sur les images une fois reconverties) sont au programme.

Par conséquent, pour la suite, on supposera que nous avons accès à une fonction `photo2matriceXX(nfichier: str` qui renvoie la matrice correspondante à la photo nommée `nfichier` et d'une procédure d'enregistrement `matrice2photoXX(mat: list, nfichier: str) -> None`, qui enregistre la matrice `mat` dans le fichier nommé `nfichier`. Selon les cas, XX vaudra NB pour les photos en nuance de gris et C pour les photos couleurs.

Ces fonctions sont disponibles dans le fichier `CPGE-traitementImages.py` qui se trouve dans l'espace d'échange et sur le site cahier-de-prepa.fr.

Exercice 7 :
On peut flouter une image en faisant la moyenne entre les pixels nord, sud, est et ouest pour déterminer la valeur du pixel au milieu. Ainsi on pourrait définir une matrice de convolution 3×3, $C = \frac{1}{4}\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$.

On considère l'image de matrice $mat = \begin{pmatrix} 250 & 200 & 150 & 100 \\ 200 & 150 & 100 & 50 \\ 150 & 100 & 100 & 0 \\ 100 & 50 & 0 & 0 \end{pmatrix}$.

Dessiner l'image définie par mat puis déterminer la matrice de l'image floutée.

2.2 Contraste

Définition 2.1 (Contraste d'une image) :
Le contraste est une propriété intrinsèque d'une image qui quantifie la différence de luminosité entre les parties claires et sombres d'une image.

Par conséquent, modifier le contraste d'une image, revient à augmenter l'écart entre les pixels les plus clairs, et les plus clairs les plus sombres (et les autres pixels également en proportion).

Plusieurs façons de faire sont possibles. Il faut faire un choix en fonction du rendu voulu.

Exercice 8 :
Une manière simple (et brutale) d'augmenter le contraste est de mettre en blanc les pixels qui ont une plus grande proportion de blanc que de noir, et réciproquement pour les pixels à dominance noire.

Proposer une fonction `contraste(mat: list) -> list`, qui implémente cet algorithme. Quel est le type d'algorithme mis en œuvre et préciser sa complexité temporelle si $mat \in \mathcal{M}_{n,m}$ (somme, seuil, etc.)

2.3 Stéganographie

Définition 2.2 (Stéganographie) :
La stéganographie (*steganós* : étanche ; *graphê* : écriture) est l'art de cacher l'écriture.

Une méthode classique consistant à encoder l'information cachée en retirant les détails de l'information visible se nomme Least Significant Bit (LSB).

Dans une image en nuance de gris, le gris de chaque pixel est défini par un nombre entre 0 et 255, soit sur 8 bits. Si on met à zéro les 4 bits de poids faibles, on ne peut coder que les multiples de 16 (0, 16, 32, $\ldots$ 240). Ce faisant, on ne peut plus distinguer le gris 160 du gris 166 mais l'œil ne remarque pas (trop) cette différence, on peut glisser l'information cachée dans ces 4 bits inutilisés.

considérant ces 4 pixels,

image stéganographiée

1010 1101	1011 1001
1010 1110	1011 0010

$\Longrightarrow$

image visible

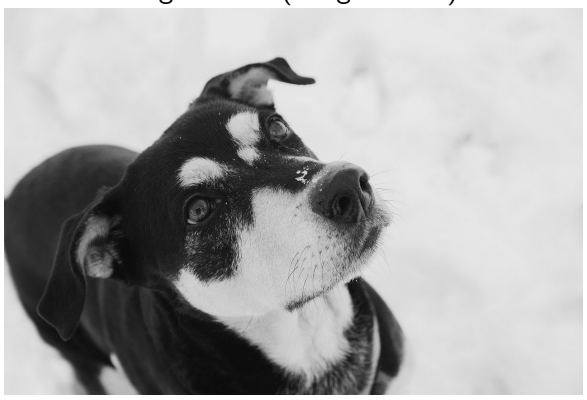
1010 0000	1011 0000
1010 0000	1011 0000

image cachée

1101 0000	1001 0000
1110 0000	0010 0000

Voici un exemple

original n° 1 (image visible)

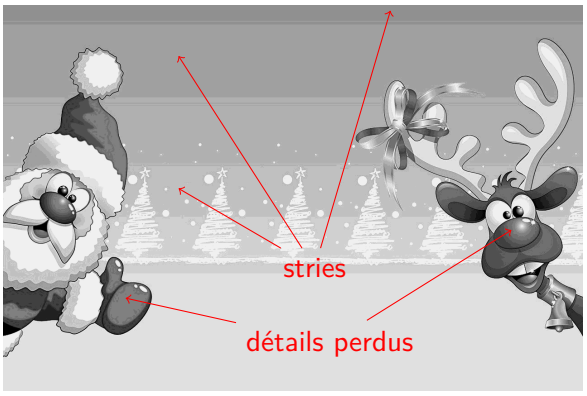


original n° 2 (image cachée)

image stéganographiée

image visible récupérée

image cachée récupérée



Remarque : on peut voir que le résultat n'est pas parfait, surtout dans les zones à faible dégradé. On pourrait améliorer le résultat en codant une image cachée de moindre dimensions que l'image visible, et n'utiliser que 3 bits par exemple.

considérant ces 4 pixels,

image stéganographiée

1010 1 101	1011 1 001
1010 1 110	1011 0 010

$\Longrightarrow$

image visible

1010 1000	1011 1000
1010 1000	1011 0000

image cachée

101	0 01	00
110	0 10	00

Exercice 9 :

1. proposer une fonction `bPoidsFortsSeuls(a:int) -> int`, qui renvoie les 4 bits de poids forts, (exemple : `bPoidsFortsSeuls(107)` renverra $96 = \overline{0110\ 0000}^2$ car $107 = \overline{0110\ 1011}^2$)
2. proposer une fonction `bPoidsFaiblesSeuls(a:int) -> int`, qui renvoie les 4 bits de poids faibles, (exemple : `bPoidsFaiblesSeuls(107)` renverra $11 = \overline{0000\ 1011}^2$ car $107 = \overline{0110\ 1011}^2$)
3. proposer une fonction `bPoidsFaiblesForts(a:int) -> int`, qui renvoie les 4 bits de poids faibles ramenés en poids forts, (exemple : `bPoidsFaiblesForts(107)` renverra $176 = \overline{1011\ 0000}^2$ car $107 = \overline{0110\ 1011}^2$)
4. proposer une fonction `bPoidsFortsFaibles(a:int) -> int`, qui renvoie les 4 bits de poids forts ramenés en poids faibles, (exemple : `bPoidsFortsFaibles(107)` renverra $6 = \overline{0000\ 0110}^2$ car $107 = \overline{0110\ 1011}^2$)
5. proposer une fonction `bInitiaux(a:int) -> tuple`, qui renvoie un tuple contenant les deux entiers formés par les 4 bits de poids forts et les 4 bits de poids faibles composants l'entier `a`. (exemple : `bInitiaux(107)` renverra le tuple `(96, 176)`).
6. Proposer une fonction `matrices2stegano(m1:list, m2:list) -> list`, qui vérifie que les deux matrices `m1, m2` sont de même dimensions (sinon renvoie un tableau vide), et renvoie la matrice résultante de la stéganographie des matrices `m2` dans la matrice `m1`.
7. Proposer une fonction `stegano2matrices(mstegano:list) -> tuple`, qui renvoie le tuple des deux matrices cachées par stéganographie dans la matrice `mstegano`.

2.4 Détection de contours

De nombreux algorithmes existent pour la détection de formes, nous verrons ceux mettant en œuvre un algorithme de convolution.

Un contour est la délimitation entre deux zones à fort contrastes. Donc, pour faire apparaître les contours, il suffit d'accentuer fortement les différences de contrastes et atténuer le reste.

Le produit de convolution est précisément fait pour ça. Le produit de convolution permet de modifier l'image en fonction d'un "filtre". En choisissant correctement la matrice de convolution, on peut alors augmenter les contrastes, et donc faire apparaître les contours.

2.4.1 Par gradient

Si on se place en une dimension, et l'on note $g(x)$ la valeur du pixel selon l'abscisse x, une grande différence de gris signifie une dérivée en valeur absolue élevée, supérieure à un seuil.

Le nombre dérivé $g'(x) = \lim_{h\to 0} \frac{g(x+h)-g(x-h)}{2h}$ s'écrit de manière discrète avec un pas de 1 par $2g'(x) \approx g(x+1) - g(x-1)$ (voir les développements limités).

Cette dérivée selon x sera calculée par le filtre 3×1 suivant `[-1/2, 0, 1/2]`. Pour une image en deux dimensions, on obtient les filtres suivants :

$$\begin{pmatrix} 0 & 0 & 0 \\ -1/2 & 0 & 1/2 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & -1/2 & 0 \\ 0 & 0 & 0 \\ 0 & 1/2 & 0 \end{pmatrix}$$

selon x selon y

On calcule une matrice des dérivées selon x, `matx` et une selon y, `maty`. Puis on calcule la matrice des normes 2 du gradient par $matg[i][j] = \sqrt{matx[i][j]^2 + maty[i][j]^2}$.

2.4.2 Par laplacien

Un contour peut correspondre à un extremum des nuances de gris, pour lequel on cherchera l'annulation de la dérivée seconde selon une direction (on utilisera souvent le laplacien qui est la dérivée seconde selon x et y : $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$).

Avec $g'(x) = g(x+1) - g(x)$, on obtiendra $g''(x) = g'(x) - g'(x-1) = g(x+1) - 2g(x) + g(x-1)$ soit le filtre 3×1 suivant `[1, -2, 1]`.

Pour un tableau en deux dimensions, le laplacien correspond à produit de convolution avec la matrice

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Exercice 10 :

On souhaite déterminer les contours d'une image à l'aide du filtre laplacien $\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$.

1. Proposer une fonction `decalageAutorise(i:int, n:int) -> list`, qui renvoie la liste des décalages autorisées `di` parmi $\{-1, 0, 1\}$, *i.e.* tel que $0 \leq i + di < n$.

2. Proposer une fonction `prod2conv(mat:list, mconv:list) -> list`, qui effectue le produit de convolution entre la matrice `mat` et la matrice de convolution `mconv`.
3. Proposer deux fonctions `contourLaplacien(maPhoto:str) -> None` et `contourGradient(maPhoto:str) ->` qui ouvrent le fichier `maPhoto.bmp`, fasse la détection de contour et enregistre l'image résultante dans le fichier `maPhoto-contoursGradient.bmp` et `maPhoto-contourGradient.bmp` en utilisant la méthode correspondante.

Mis en œuvre sur l'image du renne et du père Noël, le résultat des contours par gradient et laplacien sera

avec gradient

avec laplacien

Remarque :
On peut remettre "les couleurs à l'endroit" en inversant les niveaux de gris facilement.

Pour accentuer l'effet de contraster et donc mieux vois les contours, il est possible aussi d'appliquer un seuil.