TD – 02

Exercices d'applications

Savoirs et compétences :

Exercice 1

1. Définir une matrice aléatoire de flottants `a` de taille 50×50.
2. Compter le nombre de valeurs inférieures à 0.5.
3. Remplacer toutes les valeurs inférieures à 0.5 par 0, et celles strictement supérieures à 0.5 par 1.

Exercice 2

D'après exemple 3.15 p 28 « Algèbre linéaire », Robert C. Dalang, Amel Chaabouni.

On s'intéresse au système linéaire suivant :

$$\begin{cases} x_1 + 2x_2 + x_3 + x_4 = 0 \\ x_2 + 2x_3 + x_4 = 0 \\ x_1 + x_3 + 2x_4 = 1 \\ 2x_1 + x_2 + x_4 = 0 \end{cases}$$

1. Vérifier qu'il n'y a qu'une solution à ce système.
2. En utilisant `np.linalg.solve`, déterminer cette solution.
3. Vérifier le résultat obtenu en utilisant un produit matriciel.
4. Construire la matrice `m` de format 4×5, dont les colonnes sont successivement les colonnes de `a` et `b`.
5. Appliquer à `m` la méthode du pivot de Gauss pour résoudre « à la main » le système proposé.

Exercice 3

Créer une matrice 8×8, remplie de 0 et de 1 comme un échiquier.

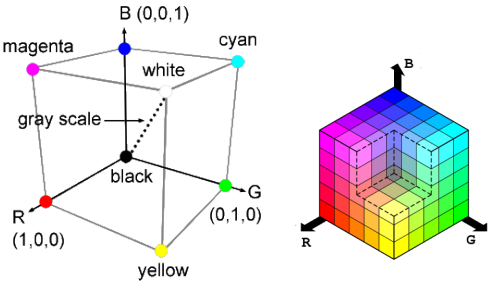
Exercice 4 – Représentation informatique d'une couleur

Commençons par parler de couleur. Il s'agit, comme toujours en informatique, de décider de la façon de stocker l'information « couleur ».

On peut choisir d'utiliser 1 bit pour stocker cette information. On distinguera alors uniquement deux couleurs. Par exemple du noir (0) et du blanc (1) ou alors du blanc (0) et du noir (1) ou pourquoi pas du rouge et du noir.

On peut choisir d'utiliser 1 octet pour stocker cette information. On distinguera alors $2^8 = 256$ couleurs différentes. Ces 256 valeurs différentes peuvent représenter des niveaux de gris différents (du blanc au noir ou alors du noir au blanc) ou pourquoi pas du blanc au rouge en passant par 254 nuances de rose.

Le plus souvent, on utilise 3 octets pour stocker une couleur. On distingue alors $256^3 = 16\,777\,216$ couleurs différentes. La norme RGB consiste à séparer ces trois octets et les penser comme un vecteur à 3 coordonnées dans l'espace $R \times G \times B$, où chaque coordonnée peut prendre $2^8 = 256$ valeurs.



On peut aussi utiliser 4 octets (trois octets pour la couleur et un pour la transparence).

On peut aussi utiliser des résolutions plus importantes, des compressions sans pertes.

Bref, il importe de se référer aux spécifications des fichiers que l'on utilise !

Représentation informatique d'une image

Il existe de nombreuses façons de représenter informatiquement une image. Certaines sont vectorielles (SVG, PS), compressées (JPG) d'autres matricielles (PNG). Nous ne nous intéressons ici qu'aux images au format PNG. Il s'agit d'un ensemble discret de points ayant chacun une couleur donnée. Ces points s'appellent des pixels. La résolution de l'image dépend du nombre de points (nombre de lignes et de colonnes), et aussi du nombre de bits servant à coder les couleurs.

Une image est stockée dans un tableau numpy, dont le `shape` est `(n,p,3)`, avec n le nombre de lignes, p le nombre de colonnes et 3 le nombre d'octets pour coder les couleurs.

En plus des bibliothèques habituelles

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

on charge la bibliothèque :

```
import matplotlib.image as mpimg
```

qui nous permet d'utiliser les instructions :

- `img = mpimg.imread(filename)` : le fichier PNG `filename` est converti en tableau numpy et nommé `img`.
- `plt.imshow(img)` : comme `plt.plot(...)`, l'affichage de `img` est préparé, et sera utilisé par `plt.show()` ou `plt.savefig(filename2)`.
- `mpimg.imsave(filename2, img)` permet d'enregistrer l'image (qui n'est pas un schéma python, n'a pas d'axes, de titre etc.)

R Avec `matplotlib.image`, les tableaux numpy sont de `dtype` `float32`, mais ils sont constitués uniquement de 256 valeurs flottantes uniformément réparties entre 0 et 1.

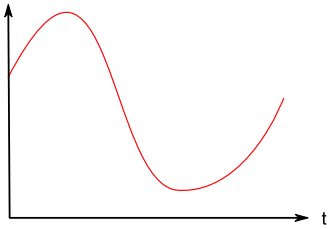
Manipulation d'images

1. Télécharger les deux fichiers PNG `playmobil.png` et `montagne.png` sur le site de la classe. Les ouvrir avec `matplotlib.image.imread` et les afficher.
   L'objectif de cet exercice est d'incruster le paysage de montagne derrière les Playmobil®.

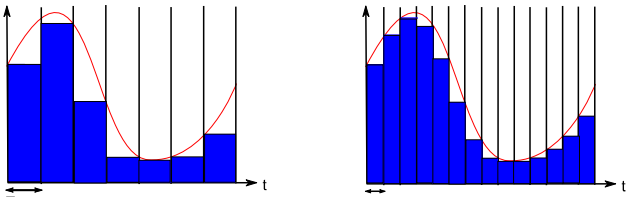
2. Déterminer pour l'image `playmobil.png` : la taille, le `dtype`, la couleur du pixel en position (300,300) et les valeurs max. et min. des éléments.
3. Est-ce que le fond bleu est de couleur uniforme ?
4. En sélectionnant par exemple les pixels dont la couleur est à une distance euclidienne faible d'un pixel de référence, « supprimer » le fond bleu de l'image `playmobil`.
   De même, « supprimer » de l'image `montagne` les pixels correspondant aux personnages.
   Construire alors l'image représentant les personnages devant la montagne.
5. Enregistrer cette nouvelle image au format PNG.

Exercice 5 – Représentation informatique du son.

Le son est une vibration de l'air, une suite de surpressions et de dépressions de l'air. L'enregistrement d'un son, c'est l'enregistrement de cette suite de variations de pression de l'air en fonction du temps : on parle de *signal analogique*.



Pour représenter informatiquement un tel signal, on discrétise l'information. On parle d'*échantillonnage* ou de *numérisation*. Le *signal numérique* est donc la suite des valeurs du signal pour une succession d'instants bien définis.



Lors de la numérisation, il y a une perte d'information plus ou moins grande selon les valeurs choisies pour :

- la *fréquence d'échantillonnage* ou *sample rate*. C'est le nombre de valeurs par seconde, exprimée en Hz. Le théorème de Shannon stipule que la fréquence d'échantillonnage doit être au moins le double de la fréquence maximale contenue dans le signal. Les fréquences les plus hautes (aiguës) perçues par l'oreille humaine sont de l'ordre de 20 kHz.
  On donne quelques valeurs courantes : DVD : 48 kHz, CD : 44100 Hz, radio : 22 kHz, téléphone : 8 kHz
- la *résolution* (quantification) ou *bit depth*. C'est le nombre de bits servant à coder les valeurs à un instant donné.
  Avec 8 bits, seules 256 valeurs différentes sont utilisées, tandis qu'avec 16 bits, ce sont 65536 valeurs différentes qui sont utilisées, ce qui permet de représenter un son beaucoup plus riche.
  On donne quelques valeurs courantes : DVD : 24 bits, CD : 16 bits, téléphone et radio : 8 bits
- le *nombre de voies* ou *channel*. Il peut y en avoir 1 (mono), 2 (stéréo), 6 (5.1) voire plus.

1. Quelle est la taille des données à stocker pour enregistrer 10 minutes de musique sur un CD ?

Manipulation de fichiers son avec Python

Il existe de multiples normes, et de nombreux formats de fichiers pour stocker des sons numérisés. Certains formats sont compressés, comme `.mp3` ou `.aac`. D'autres ne sont pas compressés comme `.wav` ou `.aif`.
Nous travaillons ici avec des fichiers `.wav`, que nous lirons et écrirons grâce aux deux fonctions `scipy.io.wavfile.read` et `scipy.io.wavfile.write`.

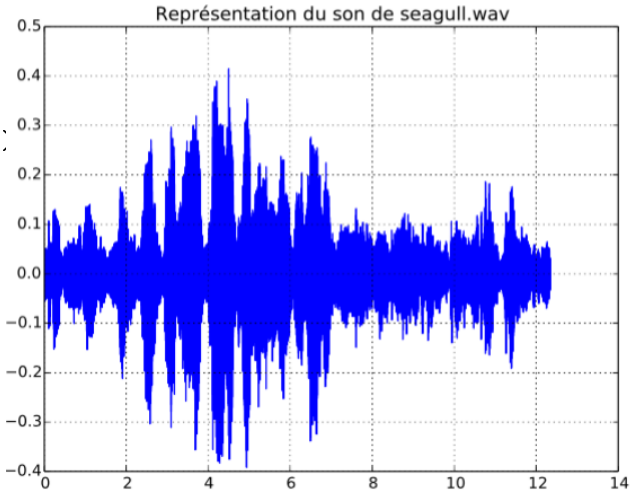
La fréquence d'échantillonnage correspond donc à `rate`.
La résolution est donnée par le `dtype` du tableau de données : `int8` ou `int16`.
Le nombre de voies est donné par le `shape` du tableau de données : `(n,)` ou `(n,2)`.

Les questions

2. Déterminer, pour les trois fichiers à télécharger sur le site de la classe, les caractéristiques du son (nombre de voies, fréquence, résolution). Préciser également la durée en secondes.
3. Définir une fonction `affiche(filename)` qui affiche la première voie du signal sur un graphique.
4. Définir une fonction `ajoute_silence(data, rate, duree_ms, debut=True)` qui prend en arguments un tableau numpy de son

mono, sa fréquence d'échantillonnage, la durée en ms du silence à ajouter, et un booléen indiquant si le silence doit être ajouté au début ou à la fin ; et qui renvoie le tableau numpy de son où un silence a été ajouté.

5. Définir une fonction `mono_to_stereo(data, rate, duree_ms = 20)` qui prend en argument un tableau numpy de son, sa fréquence d'échantillonnage ; et renvoie un tableau numpy de son correspondant à deux voies identiques, mais la seconde sera décalée de 20 ms.
6. Définir une fonction `mixage(data, sound, rate, time, voie = 0)` qui prend en argument deux tableaux de son (l'un stéréo, l'autre mono), la fréquence d'échantillonnage, un temps en secondes et le numéro d'une voie ; et renvoie un tableau numpy de son correspondant à `data` où a été superposé `sound` à partir de l'instant `time` s sur la voie `voie`.
   On ajoutera simplement les deux ondes sonores, sans se préoccuper de l'augmentation du gain et du risque de saturation.
7. Utiliser les fonctions précédentes pour créer un fichier stéréo, avec le bruit de la mer, un cri de mouette à gauche après 4 secondes, un cri de mouette à droite après 12 secondes, puis, un peu plus tard, une sirène de bateau sur les deux voies.
8. Dégrader l'un des fichiers son pour qu'il soit au standard de la téléphonie.




```
Help on function read in module scipy.io.wavfile:

read(filename, mmap=False)
    Return the sample rate (in samples/sec) and data
    from a WAV file

    Parameters
    ----------
    filename : string or open file handle
        Input wav file.
    mmap : bool, optional
        Whether to read data as memory mapped.
        Only to be used on real files (Default: False)
    Returns
    -------
    rate : int
        Sample rate of wav file
    data : numpy array
        Data read from wav file
    Notes
    -----
    * The file can be an open file or a filename.
    * The returned sample rate is a Python integer
    * The data is returned as a numpy array with a
      data-type determined from the file.
```

```
Help on function write in module scipy.io.wavfile:

write(filename, rate, data)
    Write a numpy array as a WAV file

    Parameters
    ----------
    filename : string or open file handle
        Output wav file
    rate : int
        The sample rate (in samples/sec).
    data : ndarray
        A 1-D or 2-D numpy array of either integer
        or float data-type.
    Notes
    -----
    * The file can be an open file or a filename.
    * Writes a simple uncompressed WAV file.
    * The bits-per-sample will be determined by
       the data-type.
    * To write multiple-channels, use a 2-D array
      of shape (Nsamples, Nchannels).
```