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.

Correction

```
#! /usr/local/bin/python3.4
# -*- coding:utf-8 -*-

import numpy as np

a = np.random.rand(5,5)
print(a)
# matrice aléatoire d'éléments de [0,1]

# On utilise le fait que l'on peut sommer les booléens :
print(True+True)
# 2
print(False+False)
# 0

# On construit la matrice des booléens en comparant terme àterme a et .5
b = a <= .5
print(b)

# On somme tous les termes, ce qui donne le nombre de True
print(np.sum(b))

# On utilise un masque pour modifier les valeurs de a
a[a <= .5] = 0
a[a > .5] = 1
print(a)
```

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.

Correction

```
#! /usr/local/bin/python3.4
# -*- coding:utf-8 -*-

import numpy as np

a = np.array([[1, 2, 1, 1],
              [0, 1, 2, 1],
              [1, 0, 1, 2],
              [2, 1, 0, 1]])
b = np.array([0, 0, 1., 0])

print('Le déterminant du système vaut {} <> 0 donc unique \
solution'.format(np.linalg.det(a)))
# Le déterminant du système vaut -4.0 <> 0 donc unique solution

x = np.linalg.solve(a,b)
print(x)
#[-0.5  0.  -0.5  1. ]

print('Vérification : on doit trouver le vecteur nul : \
{}'.format(a.dot(x) - b))
# Vérification : on doit trouver le vecteur nul : [ 0.  0.  0.  0.]

b.shape = (4,1)
m = np.concatenate((a,b), axis=1)
print(m)
# [[ 1.  2.  1.  1.  0.]
#  [ 0.  1.  2.  1.  0.]
#  [ 1.  0.  1.  2.  1.]
#  [ 2.  1.  0.  1.  0.]]

m[2,:] = m[2,:] - m[0,:]
m[3,:] = m[3,:] - 2 * m[0,:]
print(m)
# [[ 1.  2.  1.  1.  0.]
#  [ 0.  1.  2.  1.  0.]
#  [ 0. -2.  0.  1.  1.]
#  [ 0. -3. -2. -1.  0.]]

m[2,:] = m[2,:] + 2 * m[1,:]
m[3,:] = m[3,:] + 3 * m[1,:]
print(m)
# [[ 1.  2.  1.  1.  0.]
#  [ 0.  1.  2.  1.  0.]
#  [ 0.  0.  4.  3.  1.]
#  [ 0.  0.  4.  2.  0.]]

m[3,:] = m[3,:] - m[2,:]
print(m)
# [[ 1.  2.  1.  1.  0.]
#  [ 0.  1.  2.  1.  0.]
#  [ 0.  0.  4.  3.  1.]
#  [ 0.  0.  0. -1. -1.]]

m[2,:] = m[2,:] + 3 * m[3,:]
m[1,:] = m[1,:] + 1 * m[3,:]
m[0,:] = m[0,:] + 1 * m[3,:]
print(m)
# [[ 1.  2.  1.  0. -1.]
#  [ 0.  1.  2.  0. -1.]
#  [ 0.  0.  4.  0. -2.]
#  [ 0.  0.  0. -1. -1.]]

m[1,:] = m[1,:] - .5 * m[2,:]
# ici, il est impératif de travailler en flottants

m[0,:] = m[0,:] - .25 * m[2,:]
print(m)
# [[ 1.   2.   0.   0.  -0.5]
#  [ 0.   1.   0.   0.   0. ]
#  [ 0.   0.   4.   0.  -2. ]
#  [ 0.   0.   0.  -1.  -1. ]]
```

```
m[0,:] = m[0,:] - 2 * m[1,:]

m[2,:] /= 4
m[3,:] /= -1
print(m)
# [[ 1.  0.  0.  0. -0.5]
#  [ 0.  1.  0.  0.  0. ]
#  [ 0.  0.  1.  0. -0.5]
#  [-0. -0. -0.  1.  1. ]]

print('La solution est {}'.format(m[:,4]))
# La solution est [-0.5 0. -0.5 1. ]
```

Exercice 3

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

Correction

```
#! /usr/local/bin/python3.4
# -*- coding:utf-8 -*-

import numpy as np
# Solution 1
m = np.zeros((8, 8), dtype = np.bool)
m[1::2, ::2] = True
m[::2, 1::2] = True
print(m)
# [[False True False True False True False True]
#  [ True False True False True False True False]
#  [False True False True False True False True]
#  [ True False True False True False True False]
#  [False True False True False True False True]
#  [ True False True False True False True False]
#  [False True False True False True False True]
#  [ True False True False True False True False]]


# Solution 2
m = np.zeros(64, dtype=np.int)
m[::2] = 1
m.shape = 8, 8
print(m)
m.dtype = np.bool
print(m)
```

Exercice 4 – Représentation informatique d'une couleur

Correction

```
#! /usr/local/bin/python3.4
# -*- coding:utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# question (a)
img = mpimg.imread('./theme_image_1_playmobil.png')

# plt.figure(1)
# plt.imshow(img)
# plt.show()

# question (b)

print(img.shape)
# (564, 1000, 3)
# il y a 564 pixels en hauteur et 1000 en largeur,
# et 3 composantes couleur pour chaque pixel.

print(img.dtype)
# float32

print(img[300,300,:])
# [ 0.11764706 0.24705882 0.41568628]

print(img.max())
print(img.min())
# 1.0
# 0.0

# question (c)
# En observant l'image, quitte àzoomer, on voit que le fond n'est pas uniforme.
# Pour répondre précisément àla question, dénombrons les valeurs différentes
# dans la zone en haut àgauche (strictement) du pixel (300,300)
zone = img[0:300,0:300,:]
# couleurs = []
# for i in range(300):
#     for j in range(300):
#         if list(zone[i,j,:]) not in couleurs:
#             couleurs.append(list(zone[i,j,:]))
# print(len(couleurs))
# 3734
# Il y a 3734 couleurs différentes sur les 90 000 pixels considérés.
# Le fond n'est donc clairement pas uniforme,
# même si, àl'oeil, il peut sembler uniforme.

# question (d)

playmobil = mpimg.imread('./theme_image_1_playmobil.png')
montagne = mpimg.imread('./theme_image_1_montagne.png')

# on choisit un pixel bleu de référence
reference_bleue = playmobil[300,300,:]

# pour décider de la valeur du seuil, correspondant àune distance "faible",
# on fait des essais.
seuil = .30

# Une première version sans utiliser toutes les fonctionnalités de numpy

# les pixels qui ont une couleur proche du pixel de référence sont remplacés
# par les pixels de montagne correspondants
# les autres pixels (qui correspondent donc aux playmobils) ne sont pas modifiés
n,p,q = img2.shape
for i in range(n):
    for j in range(p):
        if np.sqrt((playmobil[i,j,0]-pixel_ref[0])**2 + (playmobil[i,j,1]-pixel_ref[1])**2
                  + (playmobil[i,j,2]-pixel_ref[2])**2)< seuil :
            playmobil[i,j,:] = img1[i,j,:]
```

```
# plt.figure()
# plt.imshow(playmobil)
# plt.show()

# Une deuxième version en utilisant les opérations terme àterme de numpy

# on sélectionne les pixels àune distance inférieure au seuil
# en utilisant un masque
mask = np.sqrt( (playmobil[:,:,0] - reference_bleue[0])**2
             +(playmobil[:,:,1] - reference_bleue[1])**2
             +(playmobil[:,:,2] - reference_bleue[2])**2) < seuil
print(mask.shape)
# (564, 1000)
mask.shape = (564, 1000, 1)
masque_complet = np.concatenate((mask, mask, mask), axis=2)

playmobil_sans_bleu = playmobil * (1 - masque_complet)
# plt.figure(2)
# plt.imshow(playmobil_sans_bleu)
# plt.show()

montagne_decoupee = montagne * masque_complet
# plt.figure(3)
# plt.imshow(montagne_decoupee)
# plt.show()

images_superposees = playmobil_sans_bleu + montagne_decoupee
# plt.figure(4)
# plt.imshow(images_superposees)
# plt.show()

# question (e)
mpimg.imsave("theme_image_1_fake.png", images_superposees)
```

Exercice – Représentation informatique du son.

Correction

```
#! /usr/local/bin/python3.4
# -*- coding:utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt
import scipy.io.wavfile

# (a)
# Pour stocker 10 minutes de musique sur un CD :
# 2 voies (stéréo), 16 bits, 44100Hz
# donc 10*60*44100*16*2 = 846720000 bits
# soit 105840000 octets
# soit 103359 kO
# soit 100.94 MO

# (b)
for filename in ["seagull.wav", "sea.wav", "horn.wav"]:
    rate, data = scipy.io.wavfile.read(filename)
    print("Pour {} : ".format(filename))
    print("nombre de voies : data.ndim = {} donc le son est mono".format(data.ndim))
    print("fréquence : rate = {} Hz".format(rate))
    print("résolution : data.dtype = {} donc la résolution est 16 bits".format(data.dtype))
    print("durée : {} secondes".format(len(data)/rate))
    print("=====")

# Pour seagull.wav :
# nombre de voies : data.ndim = 1 donc le son est mono
# fréquence : rate = 22050 Hz
# résolution : data.dtype = int16 donc la résolution est 16 bits
# durée : 12.345714285714285 secondes
# =====
# Pour sea.wav :
# nombre de voies : data.ndim = 1 donc le son est mono
```

```
# fréquence : rate = 22050 Hz
# résolution : data.dtype = int16 donc la résolution est 16 bits
# durée : 37.53356009070295 secondes
# =====
# Pour horn.wav :
# nombre de voies : data.ndim = 1 donc le son est mono
# fréquence : rate = 22050 Hz
# résolution : data.dtype = int16 donc la résolution est 16 bits
# durée : 1.1014512471655329 secondes
# =====

# (c)
def affiche(filename):
    """filename désigne un fichier .wav
    représente graphiquement la première voie du signal sur un graphique"""
    rate, data = scipy.io.wavfile.read(filename)
    # on commence par ne conserver que la première voie si le son est stéréo
    if data.ndim > 1:
        data = data[:,0]
    x = np.arange(len(data))
    y = data
    plt.plot(x,y)
    plt.show()

# affiche("seagull.wav")

# On peut aller un peu plus loin, et avoir en abscisse des temps
# et en ordonnées des valeurs entre -1 et 1, en fonction de la résolution

def affiche(filename):
    """filename désigne un fichier .wav
    représente graphiquement la première voie du signal sur un graphique"""
    rate, data = scipy.io.wavfile.read(filename)
    # on commence par ne conserver que la première voie si le son est stéréo
    if data.ndim > 1:
        data = data[:,0]
    duree_du_son = len(data)/rate      # durée du son en secondes
    x = np.arange(0, duree_du_son, 1/rate)# les valeurs de temps,
                                       # espacées de 1/fréquence
    resolution = str(data.dtype)       # on regarde la résolution
    if resolution[-1:] == "8":         # dans le dtype
        y = data / 2**8
    else:
        y = data / 2**(int(resolution[-2:]))
    plt.figure()
    plt.plot(x,y)
    plt.grid()
    plt.title("Représentation du son de {}".format(filename))
    # plt.show()
    plt.savefig(filename[:-4]+".pdf")

# affiche("seagull.wav")
# $\includegraphics[width=.5\textwidth]{../input_exos_python/theme_son_2_fig_5.pdf}$
# affiche("horn.wav")
# $\includegraphics[width=.5\textwidth]{../input_exos_python/theme_son_2_fig_6.pdf}$
# affiche("sea.wav")

# (d)
def ajoute_silence (data, rate, duree_ms, debut=True ):
    """ Renvoie un tableau de son mono identique àdata
    où un silence de durée duree_ms a été ajout au début (à la fin)"""
    assert data.ndim == 1, "attention, le son proposé n'est pas mono"
    silence = np.zeros(((duree_ms/1000)*rate,), dtype=data.dtype)
    if debut:
        b = np.concatenate((silence, data), axis=0)
    else:
        b = np.concatenate((data, silence), axis=0)
    return b

rate, data = scipy.io.wavfile.read("sea.wav")
silence_sea = ajoute_silence(data, rate, 1000, True)
print("data.dtype = {}".format(silence_sea.dtype))
# data.dtype = int16
print("durée : {} secondes".format(len(silence_sea)/rate))
```

```
# durée : 38.53356009070295 secondes

scipy.io.wavfile.write("silence_sea.wav", rate, silence_sea)
# affiche("silence_sea.wav")
# $\includegraphics[width=.5\textwidth]{../input_exos_python/theme_son_2_fig_7.pdf}$
sea_silence = ajoute_silence(data, rate, 1000, False)
scipy.io.wavfile.write("sea_silence.wav", rate, sea_silence)
# affiche("sea_silence.wav")

# (e)
def mono_to_stereo(data, rate, duree_ms = 20):
    """Convertit un son mono en stéréo
    selon la méthode proposée"""
    assert data.ndim == 1, "attention, le son proposé n'est pas mono"
    voie1 = ajoute_silence(data, rate, duree_ms, True)
    voie2 = ajoute_silence(data, rate, duree_ms, False)
    n = len(voie1)
    voie1.shape = (n, 1)
    voie2.shape = (n, 1)
    data_stereo = np.concatenate((voie1, voie2), axis = 1)
    # print("Création d'un tableau de son de shape {}".format(data_stereo.shape))
    return data_stereo

rate, data = scipy.io.wavfile.read("sea.wav")
data_stereo = mono_to_stereo(data, rate)
# Création d'un tableau de son de shape (828056, 2)
scipy.io.wavfile.write("sea_stereo.wav", rate, data_stereo)

# (f)
def mixage(data, sound, rate, time, voie=0):
    """Ajoute àdata le son sound àpartir de l'instant time en secondes"""
    assert data.ndim == 2, "attention, le son proposé n'est pas stéréo"
    debut = time*rate
    data_mixee = np.copy(data)
    data_mixee[debut:debut+len(sound),voie] += sound
    return data_mixee

# (g)
rate, data = scipy.io.wavfile.read("sea.wav")
_, mouette = scipy.io.wavfile.read("seagull.wav")
_, klaxon = scipy.io.wavfile.read("horn.wav")
data_stereo = mono_to_stereo(data, rate)
data1 = mixage(data_stereo, mouette, rate, 4, 0)
data2 = mixage(data1, mouette, rate, 12, 1)
data3 = mixage(data2, 1.8*klaxon, rate, 20, 0) # je force un peu le son
data4 = mixage(data3, 1.8*klaxon, rate, 20, 1) # de la sirène de bateau

scipy.io.wavfile.write("son_mixe.wav", rate, data4)

# (h)
# D'une part, il faut rééchantillonner le signal.
# Comme les fréquences ne sont pas multiples l'une de l'autre,
# on peut supprimer régulièrement des valeurs, mais la qualité
# sonore est alors très altérée.
# On calcule donc plutot une moyenne pondérée des valeurs
# encadrant l'instant théorique du rééchantillonnage.
# $\includegraphics[width=\textwidth]{../input_exos_python/theme_son_2_fig_4}$
# Peut-on éviter une boucle ici, je ne pense pas.

def reechantillonnage(filename, newfilename, newrate):
    """Convertit le fichier filename, échantillonné avec une fréquence donnée
    en le fichier newfilename, échantillonné avec une fréquence newrate"""
    rate, data = scipy.io.wavfile.read(filename)
    newdata = np.zeros( (len(data)*newrate//rate,) , dtype=np.int16)

    for i in range(len(newdata)):
        t = i / newrate
        j = t * rate
        k = np.floor(j)

        newdata[i] = (j-k)*data[k] + (k+1-j)*data[k+1]

    scipy.io.wavfile.write(newfilename, newrate, newdata)
    return None
```

```
reechantillonnage("seagull.wav", "seagull_2.wav", 8000)

# D'autre part, il faut réduire la résolution du signal, pour passer en int8
# Mais attention, en lisant la doc de wavfile.write, on voit qu'il faut
# passer en uint8 et pas en int8

def reduction_resolution(filename, newfilename):
    """Réduit la résolution int16 du son du fichier filename
    en int8, dans le fichier newfilename"""
    rate, data = scipy.io.wavfile.read(filename)
    newdata=(128+(data/2**16)*2**8).astype(np.uint8)
    print(newdata.dtype)
    print(newdata.max(),newdata.min())

    scipy.io.wavfile.write(newfilename, rate, newdata)
    return None

reduction_resolution("seagull_2.wav", "seagull_3.wav")
```