La simulation numérique: Traitement d'images

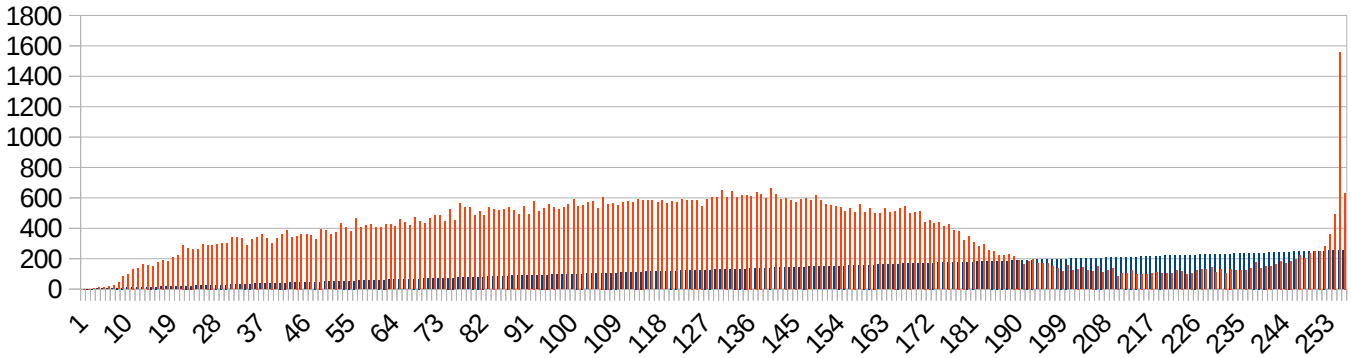
Série d'exercices N° 2

Dans la suite des exercices, utiliser le module PIL pour manipuler les images en Python.

```
>>> import PIL.Image as im #Voir annexe pour plus d'informations
>>> import numpy as np
```

Exercice 1.

L'objectif de cet exercice est de déterminer l'histogramme de l'image «Tigre en niveaux de gris»



On peut visualiser cet histogramme de deux façons différentes :

1. En sauvegardant l'histogramme dans un fichier puis en ouvrant le fichier avec un tableur et en réalisant un diagramme basique :

```
fichier_histo = open('histo.csv', 'w')
for i in range(256):
     fichier_histo.write('{},{}\n'.format(i,histo[i]))
fichier_histo.close()
```

2. Directement via matplotlib :

```
plt.plot(list(range(256)), histo)
plt.axis([0,255,0,5000])
plt.savefig('histo.pdf')
```

Exercice 2: Découpage et collage

1. Mettre un bandeau noir sur les yeux de tigre. (couper)
2. Échanger deux zones rectangulaires d'une image (copier – coller):
   Échanger les zones [[90, 190[[×[[120, 220[[et [[400, 500[[×[[80, 180[[de l'image tigre.
   Sauvegarder le résultat au format jpg.

Exercice 3: Effets sur les images

Pour une image en mode Niveaux de gris, chaque pixel est codé par un simple entier (entre 0 et 255) correspondant à un niveau de gris.

Si les entiers a,b, d, f, h, i sont tous compris entre 0 et 255, alors l'entier

$$q = (-2a - b - d + f + h + 2i)//8 + 128$$

(où // permet d'obtenir le quotient de la division entière) est également compris entre 0 et 255.

L'objectif est maintenant de transformer une image en niveaux de gris suivant le procédé ci-dessous.

a) Tout pixel de la première ligne de pixels, de la dernière ligne de pixels, de la première colonne de pixels et de la dernière colonne de pixels sera remplacé par un pixel noir.
b) Tout autre pixel sera codé par un niveau de gris calculé à partir des niveaux de gris de ses voisins.

a	b	c
d	e	f
g	h	i

Le pixel central, codé dans l'image initiale par le niveau de gris e, sera codé dans l'image finale par $q = (-2a - b - d + f + h + 2i)//8 + 128$.

Exercice 4: Floutage

Pour flouter une image, un procédé raisonnable consiste à remplacer chaque pixel par la moyenne des pixels sur un certain voisinage (typiquement, les 9, 25 ou 49 voisins...). Si on travaille en couleurs, on fait la même chose sur chaque composante RGB.
On va travailler avec *numpy*, qui offre ici trois avantages :

- On récupère un tableau au bon format (au passage, ses dimensions sont données par le champ *shape* du tableau) ;
- On peut « slicer » facilement un sous-tableau via t[i1:i2 , j1:j2], et on fait facilement la somme d'un tel sous-tableau via *numpy.sum*.
- Les calculs sont censés être rapides.

Écrire une fonction réalisant le floutage d'une image (noir et blanc, ou RGB).

Annexe: Manipulation des fichiers bitmap

Les données sont accessibles via la méthode **getdata()**, qui fournit un objet pouvant être transformé en liste. Pour le travail inverse, il s'agit de la méthode **putdata()**.
Si on travaille avec **numpy** (et c'est bien pratique...), on passera par des **array**.

Quoi	Comment
Importer la bibliothèque	`import PIL.Image as im`
Ouverture en lecture	`tigre = im.open('tigre.bmp')`
Sauvegarder une image	`tigre.save('tigre_modif.bmp')`
Si on préfère le jpg... ou le pdf...	`tigre.save('tigre_modif.jpg')` `tigre.save('tigre_modif.pdf')`
Extraire les données... et en faire une liste (linéaire)	`donnees = tigre.getdata()` `ldonnees = list(donnees)`
Mettre les données dans un tableau numpy	`tab = numpy.array(tigre)`
Créer une image depuis la liste de ses pixels	`tigre = im.new(tigre.mode,tigre.size)` `tigre.putdata(ldonnees)`
Pareil... depuis le tableau de ses pixels	`tigre = Image.fromarray(tab,mode='L')`