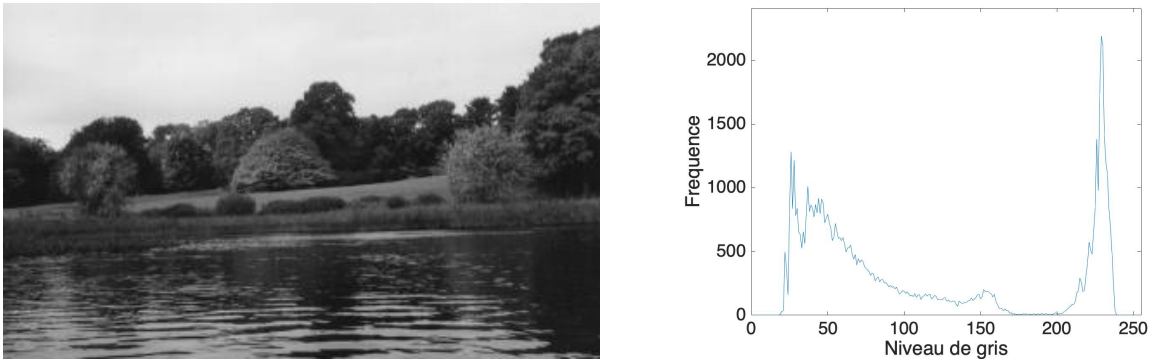
TP2 – Décorrélation et compression

Afin que vous n'ayez qu'un seul fichier à rendre pour ce TP, au lieu de créer un fichier pour chaque fonction que vous aurez à écrire, un fichier nommé `fonctions_TP2_proba` vous est fourni pour que vous complétiez les différentes fonctions qui y sont présentes.

Exercice 1 : mise en évidence des corrélations entre pixels voisins

Cet exercice constitue une illustration du cours de probabilités consacré à un couple de variables aléatoires. Ici, chaque pixel d'une image numérique est considéré comme une variable aléatoire. On s'intéresse alors à la corrélation entre pixels voisins.



(a) (b)

FIGURE 1 – (a) Exemple d'image en niveaux de gris. (b) Histogramme de l'image (a).

Le script Matlab de nom `exercice_1` affiche une image `I` interne à Matlab (cf. figure 1-a). Dans un premier temps, on cherche à connaître la répartition des différents niveaux de gris présents dans l'image `I` en affichant son histogramme (cf. figure 1-b). Pour ce faire, complétez la fonction `calcul_histogramme_image` appelée par ce script, d'en-tête :

```
[histogramme,I_min,I_max] = calcul_histogramme_image(I)
```

où `I_min` et `I_max` sont respectivement les valeurs minimales et maximales des niveaux de gris de l'image. Vous pouvez pour cela vous aider de la fonction `histcounts` (attention aux bornes !) afin d'éviter l'utilisation de boucles `for`. Le script affiche également les paires de niveaux de gris d'un pixel (de gauche) et de son voisin de droite sous la forme d'un nuage de points afin d'évaluer la corrélation entre eux. Écrivez ensuite la fonction `vectorisation_par_colonnes` d'en-tête :

```
function [Vg,Vd] = vectorisation_par_colonnes(I)
```

dont les deux paramètres de sortie `Vd` et `Vg` doivent être deux sous-matrices de `I` vectorisées, c'est-à-dire deux vecteurs colonnes, comprenant respectivement les voisins de droite et de gauche associés. Écrivez ensuite la fonction `calcul_parametres_correlation` prenant la forme :

```
function [r,a,b] = calcul_parametres_correlation(Vd,Vg)
```

qui calcule le coefficient de corrélation linéaire r des données, ainsi que les deux paramètres (a, b) de la droite de régression d'équation $y = ax + b$ (voir ci-dessous pour les rappels des différentes formules).
Remarque : Cet exercice doit être résolu sans boucle `for`, `while` ...

Rappels

- Moyenne : $\bar{x} = \dfrac{1}{n}\sum_{i=1}^{n} x_i$
- Variance : $\sigma_x^2 = \dfrac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \dfrac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{x}^2$
- Écart-type : $\sigma_x = \sqrt{\sigma_x^2}$
- Covariance : $\sigma_{xy} = \dfrac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) = \dfrac{1}{n}\sum_{i=1}^{n} x_i\, y_i - \bar{x}\,\bar{y}$
- Coefficient de corrélation linéaire : $r = \dfrac{\sigma_{xy}}{\sigma_x\,\sigma_y}$
- Équation de la droite de régression : $y = \dfrac{\sigma_{xy}}{\sigma_x^2}\,(x - \bar{x}) + \bar{y}$

Exercice 2 : décorrélation des niveaux de gris d'une image

La décorrélation des niveaux de gris consiste, par exemple, à soustraire au niveau de gris d'un pixel le niveau de gris de son voisin de gauche. Dans le script `exercice_2`, `I` (cf. figure 1-a) est remplacée par sa version décorrélée `I_decorrelee`, à l'aide de la fonction `decorrelation_colonnes` que vous devez compléter, et d'en-tête :

```
function I_decorrelee = decorrelation_colonnes(I)
```

Pour cela, initialisez `I_decorrelee` par duplication de `I`, conservez la première colonne et modifiez les autres colonnes de cette matrice.

Remarque : Il est nécessaire de conserver la première colonne de l'image d'origine dans `I_decorrelee`, sans quoi l'opération de décorrélation ne serait pas réversible (il serait impossible de recalculer l'image d'origine).

Principe du codage de Huffman

Le niveau de gris d'une image numérique est un entier compris entre 0 et 255. Il est généralement encodé avec 8 bits appelés *octet*. La correspondance entre un entier et ces 8 bits peut être a priori quelconque, du moment qu'il existe une bijection entre les deux ensembles. Avec un tel *codage à longueur fixe*, la taille d'une image est proportionnelle au nombre de pixels. La *compression sans perte* consiste à encoder les entiers autrement, de manière à utiliser moins de bits qu'avec un codage à longueur fixe. Son principe est celui du *codage à longueur variable* : les entiers les plus fréquents sont encodés sur un plus petit nombre de bits que les entiers les moins fréquents. L'*algorithme de Huffman* permet d'obtenir un codage optimal, qui est fonction des fréquences des différents entiers à encoder. Il doit donc disposer de la fréquence f_n de chaque entier n. Il construit un arbre binaire de manière récursive, à partir d'un ensemble initial d'arbres binaires. Chaque arbre binaire initial est constitué d'un seul élément, qui est un des entiers n à encoder, de fréquence f_n non nulle. La suite de l'algorithme consiste en l'itération suivante :

1. Classer les arbres binaires selon leurs fréquences.
2. Remplacer les deux arbres binaires de fréquences les plus faibles, par un nouvel arbre binaire dont la racine pointe sur les racines des deux arbres binaires supprimés. Affecter comme fréquence à ce nouvel arbre binaire la somme des fréquences des deux arbres binaires supprimés.
3. Retourner en 1 tant que le nombre d'arbres binaires est strictement supérieur à 1.

L'arbre binaire ainsi construit a pour nœuds terminaux l'ensemble des entiers à encoder. Pour connaître le code de Huffman associé à un entier, il suffit de descendre l'arbre, en partant de la racine, pour rejoindre cet entier. À chaque embranchement, on ajoute 0 si on passe à gauche et 1 si on passe à droite. Toute l'astuce du codage de Huffman réside dans le fait qu'un code ne peut pas être le préfixe d'un autre code : par exemple, les codes 001 et 00100 ne peuvent pas coexister. C'est grâce à cette propriété qu'un fichier encodé pourra être décodé.

Un générateur visuel d'arbre de Huffman est disponible à l'adresse `http://huffman.ooz.ie/`. Testez-le !

Exercice 3 : codage de Huffman pour des images

Le script `exercice_3` permet de comparer l'efficacité du codage de Huffman pour la compression entre l'image avant et après décorrélation. Dans un premier temps, il faut effectuer le codage de Huffman d'une image à l'aide de la fonction :

```
I_encodee = encodage_image(I)
```

Les différentes étapes de cette fonction consistent en le calcul des `frequences` d'apparition de chaque valeur de niveau de gris dans l'image `I` (en normalisant l'`histogramme` de l'exercice 1), la création d'un dictionnaire associant un code binaire à chaque niveau de gris, puis l'encodage de l'image à proprement parler, avec les fonctions suivantes qui vous sont fournies :

- `dictionnaire = huffman_dictionnaire(vecteur_min_a_max,frequences)`
- `I_encodee = huffman_encodage(I(:),dictionnaire)`

Pour la partie dictionnaire, il faut créer un vecteur allant de `I_min` à `I_max` par pas de 1 en première variable d'entrée, qui porte le nom `vecteur_min_a_max`. Vous remarquerez également qu'il est nécessaire de vectoriser l'image avant de pouvoir effectuer l'encodage. Une fois le codage effectué, écrivez la fonction `coeff_compression` d'en-tête :

```
function coeff_comp = coeff_compression(signal_non_encode,signal_encode)
```

qui doit calculer le *coefficient de compression* atteint par le codage de Huffman, défini comme le rapport entre le nombre de bits nécessaires pour encoder une image dans sa version d'origine (les pixels sont encodés sur 8 bits) et le nombre de bits de la même image encodée par le codage de Huffman. Il s'agit bien d'un *codage sans perte* : en décodant l'image encodée, l'image originale est parfaitement retrouvée. Écrivez ensuite la fonction `gain_compression`, qui doit calculer le rapport entre coefficients de compression après décorrélation et avant décorrélation, et qui prend la forme suivante :

```
function gain_comp = gain_compression(coeff_comp_avant,coeff_comp_apres)
```

En décorrélant l'image initiale avant de la compresser, vous constatez que le coefficient de compression augmente, ce qui signifie que le gain en compression obtenu après décorrélation est effectivement supérieur à 1.