TP2 – Compression d'un signal

Initiation à Matlab (suite)

- `length(V)` retourne la taille d'un vecteur (ligne ou colonne) ou la plus grande dimension d'une matrice.
- `indices = find(V==0)` : le vecteur `indices` contient les indices `i` de la *matrice vectorisée* `W = V(:)` tels que `W(i)==0` (n'importe quelle expression booléenne peut être passée en paramètre à la fonction `find`).
- `[V_trie,indices] = sort(V,'ascend')` : si `V` a plus d'une ligne, alors `V_trie` est une version de `V` triée par ordre croissant, colonne par colonne ; sinon, le tri est effectué sur la ligne unique de `V` ; la matrice `indices` contient les indices correspondants ; pour `'descend'`, idem que `'ascend'` par ordre décroissant.

Exercice d'initiation à Matlab

Le script `exercice_Matlab` lit un texte en français, le stocke dans le vecteur `texte` et crée un alphabet ASCII de 128 caractères. Écrivez les fonctions `calcul_frequences`, `selection` et `tri` appelées par ce script :

- `calcul_frequences` calcule les *fréquences relatives* des caractères dans le texte passé en paramètre. Remarque : une boucle `for` est autorisée. Une fois cette fonction écrite, lancez `test_calcul_frequences`.
- `selection` sélectionne les caractères de fréquences strictement positives, et les fréquences associées.
- `tri` consiste à trier les caractères par ordre de fréquence décroissante (le script `exercice_Matlab` doit afficher dans cet ordre l'histogramme des caractères sélectionnés).

Vous remarquerez que le caractère le plus fréquent du texte donné en exemple est le caratère espace.

Codage de Huffman

Le niveau de gris d'une image numérique est un entier compris entre 0 et 255. Il est généralement encodé par une séquence de 8 bits appelée *octet*. La correspondance entre un entier et une séquence de 8 bits peut être a priori quelconque, du moment qu'il existe une bijection entre les deux ensembles. La convention usuelle est celle de la représentation en base 2, mais d'autres conventions sont possibles. 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 (c'est-à-dire de son nombre d'occurrences). 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 1 : codage de Huffman d'un texte

Lancez le script `exercice_1`. Ce script encode le texte du script `exercice_Matlab` par le codage de Huffman. Chaque caractère de fréquence non nulle reçoit donc un code binaire dont la longueur dépend de sa fréquence. Vous pouvez par exemple afficher les codes binaires `dico{23,2}` et `dico{13,2}` des caractères `dico{23,1}` (lettre minuscule `e`) et `dico{13,1}` (lettre majuscule `E`) : la comparaison des longueurs de ces codes confirme bien que le nombre d'occurrences du `e` est très supérieur à celui du `E` (cela se voit en lançant `exercice_Matlab`).

Écrivez la fonction `coeff_compression_texte`, 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 un texte dans sa version d'origine, sachant qu'en ASCII, les caractères sont encodés sur 8 bits, et le nombre de bits du même texte encodé par le codage de Huffman. Vous observez qu'il s'agit bien d'un *codage sans perte* : en décodant le texte encodé, le texte original est parfaitement retrouvé.

Exercice 2 : codage de Huffman d'une image

Dans le TP1, vous avez vu un moyen de décorréler les niveaux de gris des pixels d'une image. Or, la décorrélation permet d'améliorer les performances de la compression sans perte. Le script `exercice_2` vise à calculer le *gain en compression* qui peut être atteint par décorrélation préalable.

Écrivez la fonction `coeff_compression_image`, qui doit calculer le coefficient de compression obtenu avec le codage de Huffman d'une image. Pour cela, vous devez calculer le nombre de bits de l'image initiale sans compression, et le nombre de bits obtenu avec codage de Huffman (il est inutile d'effectuer l'encodage, qui est long). 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 par décorrélation est effectivement supérieur à 1. Remarque : une boucle `for` est autorisée pour écrire cette fonction.

Exercice 3 : codage arithmétique d'un texte (facultatif)

Le codage de Huffman est théoriquement optimal, en ce sens qu'un message est encodé avec un nombre de bits minimal. Toutefois, pour certaines distributions de caractères, ce codage est limité à cause du fait que chaque caractère est encodé par un nombre *entier* de bits. Par exemple, pour l'alphabet (s, u, v) et les probabilités $(0.9, 0.05, 0.05)$, chaque caractère est encodé au minimum sur un bit, malgré l'écrasante majorité de s.

Le *codage arithmétique* permet de traiter ce genre de cas en n'imposant pas un nombre entier de bits. Il est généralement plus performant que le codage de Huffman. Étant donné une loi de probabilité, chaque message est encodé de façon unique par un *intervalle* `[borne_inf,borne_sup[`. Tout nombre appartenant à cet intervalle encode alors le message original. Le dictionnaire est remplacé par la matrice `bornes`. Chaque caractère est encodé par un intervalle inclus dans $[0, 1[$, de longueur égale à sa probabilité (ces intervalles constituent donc une partition de $[0, 1[$). Dans l'exemple précédent, s est encodé par $[0, 0.9[$, u par $[0.9, 0.95[$ et v par $[0.95, 1[$. Par conséquent : `bornes = [0 0.9 0.95 ; 0.9 0.95 1]`.

Une fois cette matrice remplie, on encode un message en rétrécissant progressivement la taille de l'intervalle, de la manière suivante (le décodage s'effectue de façon similaire) :

1. Initialisation :
   ```
   borne_inf = 0;
   borne_sup = 1;
   ```
2. Pour chaque caractère du message (par ordre d'apparition), d'indice `j` dans l'alphabet, faire :
   ```
   largeur = borne_sup-borne_inf;
   borne_sup = borne_inf+largeur*bornes(2,j);
   borne_inf = borne_inf+largeur*bornes(1,j);
   ```
3. Choisir un nombre quelconque dans l'intervalle `[borne_inf,borne_sup[` : ce nombre encode le message !

Écrivez la fonction `partitionnement`, qui crée la matrice `bornes`, et la fonction `codage_arithmetique`, qui calcule `borne_inf` et `borne_sup` selon l'algorithme ci-dessus. Le script `exercice_3` compare le nombre de bits nécessaires pour encoder un texte avec l'algorithme de Huffman et avec le codage arithmétique, pour un message court. Testez d'autres messages : dans quels cas l'algorithme de Huffman est-il inadapté ?