**Codage de source**  
  
Codage de source.Le but du codage de source peut être de compresser l'information répétitive du langage, sa redondance. Pour toute langue, on peut considérer l'entropie d'un message, c'est-à-dire la quantité d'information transmise. Ceci donne lieu au théorème du codage de source.  
  
**Codage de canal**  
  
Le but est d'ajouter de l'information redondante à un message pour compenser le bruit sur le canal de communication. Ceci donne lieu au théorème du codage de canal et c'est à celui-ci qu'on doit l'origine de la théorie des codes.  
  
  
**Codage de données**  
  
7 bits pour un caractère  
"L' American Standard Code for Information Interchange" (ASCII) s'est donc ingénié à coder chaque caractère d'une machine à écrire sous la forme d'une combinaison de 7 bits. En décimal, ça nous donne des valeurs comprises entre 0 et 127.  
  
Comme la base binaire (0 ou 1), si elle est très commode pour un calculateur électronique, l'est beaucoup moins pour le cerveau humain, nous allons utiliser une autre base qui, si elle n'est guère plus "parlante", offre tout de même l'avantage d'aboutir à une écriture beaucoup plus compacte. Cette base devra être une puissance de 2, la plus courante étant la base hexadécimale, parce que chaque "digit" hexadécimal va représenter une combinaison de 4 bits :  
  
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111  
0 1 2 3 4 5 6 7 8 9 A B C D E F  
  
Mais on peut aussi utiliser de l'octal, sur trois bits.  
  
Pourquoi pas la base 10 à laquelle nous sommes habitués depuis notre plus tendre enfance ? Parce que, malheureusement, 10 n'est pas une puissance de 2 et qu'un "digit" décimal ne représente donc pas toutes les combinaisons que l'on peut faire avec un groupe de n bits. 4 c'est trop (hexadécimal) et 3 c'est pas assez (octal). Plus mathématiquement, on ne peut pas trouver de valeur entière de n telle que 10=2n. Essayez donc de résoudre n=Log(10)/Log(2).  
  
Certains ont mis en oeuvre un codage appelé BCD (Binary Coded Decimal). Le principe est simple : chaque "digit" décimal (de 0 à 9) est codé sur un quartet. Certaines combinaisons de bits sont donc impossibles.  
  
•9 va donner 1001  
•10 donnera 0001 0000 et non pas 1010  
Mais revenons à notre code ASCII ; 7 bits sont-ils suffisants ? Oui et non...  
  
D'abord, dans une machine à écrire, il n'y a pas que des caractères imprimables. Il y a aussi des "caractères de contrôle", comme le saut de ligne, le retour chariot, le saut de page, la tabulation, le retour arrière... Tous ces caractères doivent aussi être codés pour que l'ordinateur puisse efficacement piloter une imprimante.  
  
De plus, pour transmettre convenablement un texte, il faudra quelques sémaphores pour indiquer par exemple quand commence le texte, quand il finit...  
  
Enfin, suivant les langues, même lorsqu'elles exploitent l'alphabet latin, certaines lettres sont altérées différemment. L'anglais n'utilise pas d'accents mais la plupart des autres langues les exploitent plus ou moins parcimonieusement.  
  
Au final, si 7 bits suffisent généralement pour une langue donnée, éventuellement en faisant l'impasse sur certains symboles peu usités comme [ ou ], nous ne pourrons pas coder l'ensemble des caractères nécessaires pour la totalité des langues utilisant l'alphabet latin.  
  
La norme iso-646 définit un code ASCII sur 7 bits. Ce code, parfaitement adapté à l'anglais US, l'est moins pour les autres langues. Nous assistons donc à la création d'une multitude de "dialectes ASCII", où certains caractères sont remplacés par d'autres suivant les besoins locaux. Les lecteurs les plus "anciens" se rappelleront peut-être des configurations hasardeuses de certaines imprimantes pour arriver à ce qu'elles impriment en français lisible...  
  
  
ASCII US défini par la norme iso-646 Les "caractères" sur fond bleu sont les caractères non imprimables.  
Pour bien lire le tableau, il faut construire le code hexadécimal en prenant d'abord le digit de la ligne, puis le digit de la colonne. Par exemple, la lettre "n" a pour code hexadécimal 6E  
  
  
Comme vous le constatez, il n'y a aucune lettre accentuée dans ce codage. Ce dernier a donc été joyeusement "localisé" pour satisfaire aux exigences des divers pays utilisant l'alphabet latin. Cette situation aboutit rapidement à une impasse, les fichiers ainsi construits n'étant plus exportables dans d'autres pays. De plus, vous constaterez aisément que l'ajout de caractères supplémentaires (le "é", le "ç", le "à" etc.) implique obligatoirement la suppression d'autres caractères (le "[", le "]", le "#" etc.). Ceux qui ont quelques notions de programmation comprendront à quel point c'est facile d'écrire du code avec un jeu de caractères amputé de ces symboles. Dans la pratique, les programmeurs sont condamnés à utiliser un clavier US.  
  
Pour un bit de plus  
Avec les avancées de la technique, le huitième bit qui servait pour le contrôle de parité, contrôle rendu de plus en plus inutile, va être utilisé pour coder plus de caractères. Deux fois plus, finalement.  
  
Ainsi, le codage "iso-latin-1", également connu sous le nom de "iso-8859-1" propose à peu près le codage suivant  
  
Comme vous pouvez le constater ici :  
•Les codes ASCII de 0 à 7F (127 en décimal) demeurent inchangés,  
•les codes supérieurs (ceux qui ont le bit 7 à 1) représentent quelques symboles supplémentaires, ainsi qu'une panoplie de lettres accentuées qui satisfont aux exigences des langues de l'Europe de l'Ouest.  
  
  
Pourquoi "à peu près" ? Le codage ci dessus est une interprétation de la norme iso-8859-1 par notre cher Microsoft qui a un peu bricolé pour ajouter quelques symboles de plus, dont celui de l'euro... La conséquence en est qu'une fois de plus, Windows n'est compatible qu'avec lui-même. Fort heureusement, nous verrons qu'il demeure possible d'adopter un codage plus officiel avec les applications communicantes, mais avec des limites. Notez que si l'on peut reprocher à Microsoft de ne pas suivre les normes, il faut aussi reprocher aux normes d'êtres imparfaites et assez peu réactives.  
  
Pour ajouter à la complexité, la norme iso-8859 définit pas moins de 15 versions différentes, pour satisfaire à tous les besoins mondiaux. A titre d'information, la norme iso-8859-15 devrait pouvoir être utilisée pour l'Europe de l'Ouest avec plus de "bonheur" que l'iso-8859-1.  
  
Finalement, ce bit de plus ne fait que déplacer le problème sans toutefois l'éliminer, nous ne disposons toujours pas d'un système normalisé universel.  
  
**Codage de la parole**  
  
Introduction  
  
I) Analyse du signal de parole  
  
A. Analyse du signal de parole  
1. Analyse du fichier froid.wav  
2. Signal stationnaire  
3. Analyse spectrale de fichiers  
  
II) Quantification scalaire  
  
A. Quantification scalaire uniforme  
1. Etude du fichier SQuantifU.m  
2. Quantification scalaire du fichier froid2.wav  
3. Quantification scalaire du fichier froid2.wav avec un gain quasi égal au gain maximum  
4. Quantification scalaire du fichier froid.wav  
B. Quantification scalaire logarithmique  
1. Etude des fonctions lin2mu et mu2lin  
2. Modification du fichier SQuantifU.m  
3. Courbe du signal quantifié  
4. Relation entre le rapport S/B et la puissance du signal d'entrée  
5. Débit d'une quantification linéaire  
6. Quantification logarithmique du fichier froid.wav  
7. Débit minimum acceptable  
C. Analyse LPC et quantification scalaire prédictive  
1. Analyse prédictive uniforme de froid2.wav  
2. Influence du gain et du débit sur le rapport S/B  
3. Quantification prédictive du fichier froid.wav  
  
III) Modélisation du signal de la parole  
  
A. Analyse LPC  
1. Comparaison de réponses en fréquences  
2. Variation de la longueur de l'analyse  
3. Variation de l'ordre du filtre  
B. Introduction à la synthèse  
1. Synthèses successives du signal  
2. Introduction au principe de quantification vectorielle  
C. Influence de la quantification des coefficients du filtre de synthèse  
1. Reconstitution correcte du signal  
2. Quantification des coefficients LSP  
  
IV) Etude du codeur CELP32  
  
A. Etude du schéma de principe  
B. Etude algorithmique  
1. Etude globale  
2. Analyse LPC  
  
**Codage GSM**  
  
Les codes cycliques.  
  
Les codes cycliques sont des codes linéaires en blocs. Plaçant à l'entrée du codeur un bloc de k bits, celui-ci renvoie un bloc de n bits (avec bien sûr n > k). En utilisant des codeurs systématiques, on peut faire en sorte que les k premiers bits de sortie soient les mêmes que les bits entrés, le codeur se contentant alors de rajouter (n-k) bits dits bits de contrôle de parité.  
  
Dans le cas particulier des codes cycliques C(n,k,d), les mots du code doivent être stables par permutation circulaire. On peut alors modéliser le codage mathématiquement à l'aide de polynômes, à coefficients dans le corps de base (Z/2Z si on travaille sur des éléments binaires, mais GF(2^m) de façon plus générale), définis modulo le polynôme D^n - 1.  
  
Chaque mot du code est alors représenté par un polynôme c(D) de degré < n correspondant à la séquence de ses bits, de même pour les séquences de bits d'information représentés par des polynômes i(D) de degré < k.  
  
Tous les mots du code peuvent être obtenus à partir d'un polynôme générateur g(D) en multipliant celui-ci par un polynôme de degré < k (le polynôme d'information en général). g(D) est un polynôme de degré r = n-k divisant D^n-1, c'est à dire que D^n - 1 = g(D).h(D), où h(D) est le polynôme de parité.  
  
Les bits de parité, appelé dans ce cas CRC pour Cyclic Redundant Checksum, sont obtenus en prenant le reste de la division euclidienne de D^r.i(D) par g(D), ce qui donne un polynôme de degré n-k. Les bits correspondants à ce polynôme constituent le CRC.  
  
On obtient alors c(D) = D^r.i(D) + n(D), où n(D) est le CRC correspondant à i(D).  
  
Lors de la réception d'un mot y(D), on calcule le reste de la division euclidienne de y(D) par g(D). Si ce reste est nul, y(D) est un mot du code et sera donc accepter, sinon il y a erreurs.  
  
GSM utilise, pour les 50 bits de la classe Ia du canal de parole, un code cyclique C(53,50,2)de polynôme générateur G0 = 1+D+D^3, ce qui protège les bits de la classe Ia par 3 bits de parité.  
  
Revenir au système de codage cyclique des bits de la classe Ia dans GSM.  
  
Aller au système de codage convolutionnel des bits de la classe I dans GSM.  
  
Les codes convolutionnels.  
  
Les codes convolutionnels fonctionnent sur le principe des registres à décalage par blocs. On code une trame de k bits par un bloc de n bits, mais le codage fait intervenir également la donnée des L-1 trames précédentes. Une fonction logique permet de faire le calcul de n bits de sortie à partir des (k \* L) bits dans les registres. Une fois les calculs faits, les trames sont décalées par blocs de k bits, une nouvelle trame est introduite à l'entrée du codeur et on peut recommencer le calcul.  
  
On peut représenter le calcul de la fonction logique par polynômes : A chacun des n bits de sortie sont associés k polynômes. On dispose ainsi d'une matrice de polynômes [Gij(D)] ( 0 £ i £ i-1, 0 £ j £ n- 1). Le coefficient d'indice r de Gij(D) représentant la contribution du ième bit du rième registre sur le jème bit de sortie. Les polynômes Gij(D) sont donc de degré < L. Ceci vaut si les contributions des différents bits sont additives.  
  
La valeur des bits de sortie peut alors se calculer de façon formelle à l'aide de calcul de produits de polynômes.  
  
Les caractéristiques du code sont alors :  
  
rendement du code par le rapport n/k,  
  
La longueur de la fenêtre L,  
  
Les polynômes générateurs [ Gij(D) ]  
  
La distance caractéristique du code, qui permet de déterminer le nombre d'erreurs corrigibles.

<https://infoforall.fr/fiches/fiches-act020.html>

<https://www.grafikart.fr/tutoriels/encodage-utf8-1000>