Introduction à la programmation et au traitement de signaux physiques numérisés
Un peu de théorie et de pratique

Mesure, Acquisition, Traitement et Analyse de Données 1
UE. MATAD1

Master de Physique - 1ère année
parcours Physique Appliquée

François Copie et François Anquez
Département de Physique,
Fac. S&T - Université de Lille
francois.anquez@univ-lille.fr
francois.copie@univ-lille.fr

2020

TABLE DES MATIÈRES

1

REPRÉSENTATION DES NOMBRES ET SIGNAUX DANS UN ORDINATEUR

L'utilité d'un ordinateur est de pouvoir lui faire faire le travail ennuyeux à notre place. En physique expérimentale, cela correspond souvent à programmer une série de transformations plus ou moins complexes d'un signal pour en extraire des informations. Dans le cadre du traitement numérique, ces opérations sont *in fine* traduites en une série d'opérations arithmétiques simples exécutées par le processeur. Bien que l'arithmétique des nombres binaires ou la logique puisse avoir des aspects intéressants, nous omettrons volontairement ces notions. Nous nous contenterons d'aborder comment sont représentés (codés) les nombres dans un ordinateur pour, ensuite, introduire quelques structures de données tels que les tableaux. Enfin nous introduirons la notion d'échantillonnage pour préciser comment sont représentés en machine les signaux temporels et les images.

1.1 Représentation des nombres

1.1.1 Généralités

Les données sont représentées en codage dit binaire dans un ordinateur. C'est-à-dire par une série de 0 et de 1, que l'on nomme «bits». Une série de bits représentent («codent») une information. Pour les données non numériques il existe de multiples façon de représenter les données. Nous n'aborderons pas ces représentations ici et nous nous limiterons au codage les plus utilisés pour les nombres.

Concernant les nombres ou les listes de nombres, ce qui nous intéresse est la possibilité d'effectuer des opérations et des séries d'opérations. Dans la plupart des cas une représentation en base 2 sera commode. En effet, les opérations arithmétiques seront naturellement réalisées par la machine dans cette représentation.

1.1.2 Entiers naturels

Considérons un nombre entier naturel[1] représenté sur N bits. Dans ce cas la série de bits correspond à la décomposition du nombre entier en base 2. Le plus petit élément de la base est 2^0 et le plus grand est $2^N - 1$. La série de bits est numérotée de la droite vers la gauche. Si on note x_i la valeur du bit i ($x_i = 0$ ou 1), le nombre entier, U ainsi représenté s'écrit donc :

1. En informatique on parle couramment d'entier non-signé (unsigned integer, uint).

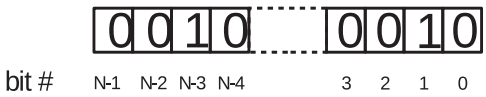


Figure 1.1 – Illustration de la représentation numérique d'un entier naturel. Le nombre de bits est N. Les bits sont numérotés de la droite vers la gauche.

$$U = \sum_{i=0}^{N-1} x_i^i \tag{1.1}$$

Sous le logiciel GNU-Octave, ce type de nombre est nommé : `uint8`, `uint16`, ... On s'en servira assez peu dans un premier temps. Ils s'avéreront utiles pour la représentation des images.

Exercices

- Donnez la valeur en représentation décimale des nombres suivant (codés sur 8 bits) : `00000001`, `00000010`, `00000111` ;
- Quel est la valeur en représentation décimale du plus grand nombre codé sur 8 bits ? sur N bits ?
- Déterminez la représentation binaire du nombre décimal 43. - *utilisez une succession de divisions par 2 du nombre décimal. Que faire des restes ?*

1.1.3 Entiers relatifs

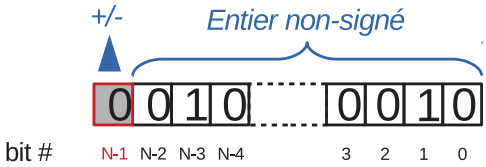


Figure 1.2 – Illustration de la représentation numérique d'un entier relatif. Le nombre de bits est N. Les bits sont numérotés de la droite vers la gauche. Le dernier bit (à gauche) est le bit de signe. Les autres bits codent pour un entier non-signé.

Pour les entiers relatif on introduit un bit de signe (bit $N-1$). Les bits de 0 à $N-2$ codent pour un entier naturelle. Un entier relatif, S, représenté sur N bits, s'écrit :

$$S = (-1)^{x_{N-1}} \times (\sum_{i=0}^{N-2} x_i \times 2^i) \tag{1.2}$$

où x_i est la valeur du bit i ($x_i = 0$ ou 1).

Sous le logiciel GNU-Octave, ce type de nombre est nommé : `int8`, `int16`, ... On notera que les plus utilisé sont les entiers codés sur 16 bits qui sont nommés `int`.

Exercices

- Donnez la valeur en représentation décimale des nombres suivant (codés sur 8 bits) : `00000001`, `10000010`, `00000111` ;
- Quel est la valeur en représentation décimale du plus grand nombre codé sur 8 bits ? du plus petit ?
- A priori le nombre binaire `10000000` n'éxiste pas. Que voudriez-vous en faire ?
- Déterminez la représentation binaire du nombre décimal -43.

1.1.4 Nombres réels : représentation à virgule flotante

On ne mentionnera ici que la représentation dite à virgule flotante. Celle-ci ressemble à la notation dite scientifique des nombres. Par exemple, plutôt que d'écrire $+0,000134$ on préfère $+0,134 \times 10^{-3}$. De manière générale, en base b, un nombre F dans cette représentation à virgule flotante, s'écrit :

$$F = s \times M \times b^e \tag{1.3}$$

où s est le signe, b est la base et e l'exposant. M est la mantisse qui représente un nombre entre 0 et 1 qui précise les chiffres significatifs. Dans notre exemple précédent ($+0,134 \times 10^{-3}$), on a : $s = 1$, $b = 10$, $e = -3$ et $M = 0,134$.

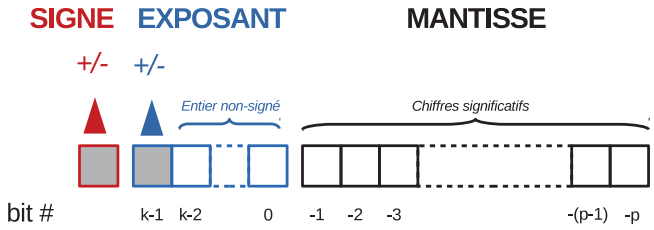


Figure 1.3 – Illustration de la représentation machine dite en virgule flotante d'un réel. Le nombre de bits est $N = 1 + k + p$. Le signe est codé sur 1 bit. L'exposant sur k bits et la mantisse sur p bits sont numérotés.

Dans un ordinateur le principe est le même, sauf que les nombres sont codés en format binaire.

- Le signe s est toujours codé sur un seul bit : $s \in \{-1; 1\}$.
- L'exposant est codé sur k bits : $e \in [-2^{k-1}; 2^{k-1}]$.
- La mantisse est codée sur p bits qui correspondent au p chiffres (d_i) après la virgule. N'oublions pas que l'ordinateur code en binaire (base 2) :

$$M = d_{-1}, d_{-2}, ..., d{-p}$$

$$M = d_{-1} \times 2^{-1} + d_{-2} \times 2^{-2} + ... + d_{-p} \times 2^{-p}$$

La norme *IEE754* spécifie deux formats de représentation en virgule flotante. Dans chaque cas le signe est codé par un seul bit. Pour les nombres dit à simple précision qui sont codés sur 32 bits au total : $k = 8$ et $p = 23$. L'exposant E n'a pas de bit de signe et on obtient des exposants négatifs ou positif par la convention suivante : $e = E - 127$. Le nombre réel simple précisions s'écrit alors : $F = (-1)^s \times M \times 2^{E-127}$. Pour les nombres dit à double précision qui sont codés sur 64 bits au total : $k = 11$ et $p = 52$. De manière analogue : $D = (-1)^s \times M \times 2^{E-1023}$. Dans le logiciel GNU-Octave ces types de nombres sont respectivement nommés `float` et `double`.

Exercices :

- Soit le réel représenté en binaire à virgule flottante de la manière suivante ($k = 3$ avec bit de signe, $p = 4$) : `0 011 0101`, donnez sa représentation décimale.
- Estimez le nombre de chiffres significatifs en base décimal des réels simple et double précision de la nomre *IEEE754*.

1.2 Quelques structures de données : Tableaux, vecteurs et matrices

La plupart des langages de programmation s'appuient sur des structures de données. Selon le «niveau » de langage et le type d'information à coder, les structures de données sont plus ou moins abstraites. Nous nous limiterons ici aux structures de données simples qui nous seront utiles pour l'analyse de signaux physiques. On distingue les langages dit de «bas niveau », dont les structures de donnée sont simple et le plus proche du «langage machine » des langages dit de «haut niveau», dont les structures de données peuvent être plus abstraites.

1.2.1 Tableaux

Il s'agit d'une structure de donnée simple que l'on trouve dans la plupart des langages de programmation. Il s'agit d'une liste ordonnée de nombres qui sont stockés successivement (en file indienne) en mémoire (sans séparation). Les conventions de numérotation varient. Prenons pour convention un tableau de p nombres chacun codés sur N bits. Il y a donc $p \times N$ bits au total. Pour être cohérent avec les notations du langage du logiciel GNU-Octave, nous numérotons les éléments du tableau [2] de 1 à p.

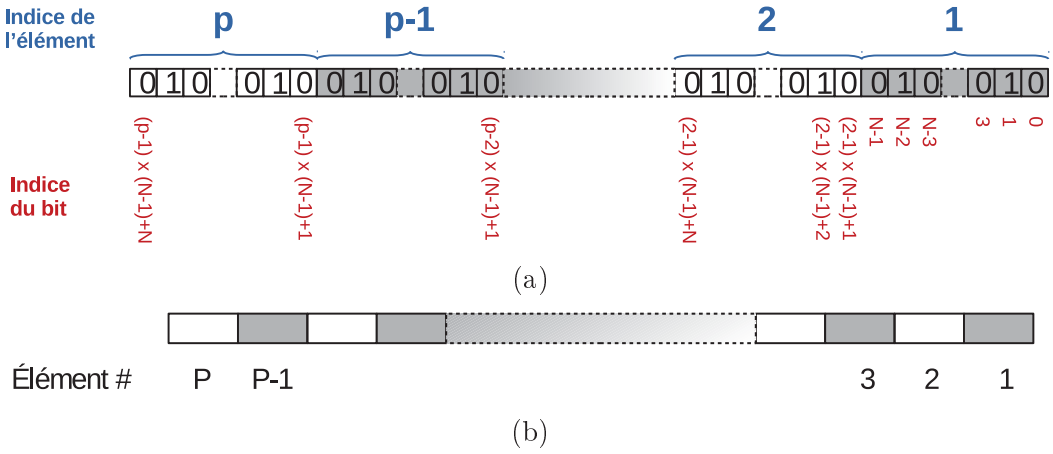


Figure 1.4 – (a) Illustration de l'architecture en mémoire des $N \times p$ bits d'un tableau de p nombres codés sur N bits. Les éléments du tableau sont numérotés de 1 à p alors que les bits sont numérotés de 0 à N. (b) Représentation abstraite d'un tableau de P éléments.

Le type `array` existe dans le logiciel GNU-Octave. Il nous semble cependant moins utile que le type par défaut (matrice).

Exercice : tableau 2D Imaginons que vous ayez des données (toutes de même type) stockées selon deux dimensions, par exemple un tableau avec N_l lignes et N_c colonnes. Vous disposez donc $N_l \times N_c$ données que vous souhaitez ranger dans un tableau. Imaginons maintenant que le langage de programation que vous utilisez ne dispose pas de structure «tableau 2D»mais uniquement de tableau 1D. Nommons l l'indice des lignes et c l'indice des colonnes.

- Imaginez un moyen de ranger vos données dans un tableau 1D. Faites un schéma abstrait.
- Soit k l'indice des éléments du tableau 1D, exprimez k en fonction de l et c.

1.2.2 Matrices et vecteurs sous GNU-Octave

Le logiciel GNU-Octave a été développé initialement pour travailler sur des matrices. Il possède un langage dit de «haut niveau». Ce langage est fait pour travailler sur des structures abstraites : les matrices sont donc le type par défaut pour stocker les données numériques dans ce logiciel. Ces matrices peuvent être de dimension supérieur à 2 et les vecteurs sont des types particuliers de matrices.

Les opérateurs mathématiques tel que le produit matriciel ou la transposition sont très facilement utilisable. Ce sont des opérations «pré-programmées». Par ailleurs le langage offre la possibilité d'utiliser d'autres opérateurs courants sur ces matrices tels que la multiplication terme à terme, l'accès à un sous-ensemble de la matrice ou des fonctions plus évoluées. Le logiciel peut donc être utilisé non seulement pour les opérations matricelles mais aussi pour d'autres applications en physique. Nous aborderons tout cela de manière pratique sous forme d'ateliers (cf. TP1, TP2 et TP3).

2. Alors que les bits sont quand à eux sont numérotés de 0 à $N-1$ (convention chosit au départ).

1.3 Echantillonnage

1.3.1 Echantillonnage régulier : définition

On considère un signal physique $S(t)$ qui dépend, par exemple, du temps t. On s'affranchit ici des unités de S qui peuvent être des volts, des ampères ou tout autre grandeur physique. L'échantillonnage régulier consiste en la mesure périodique des valeurs instantanées[3] de la grandeur physique.

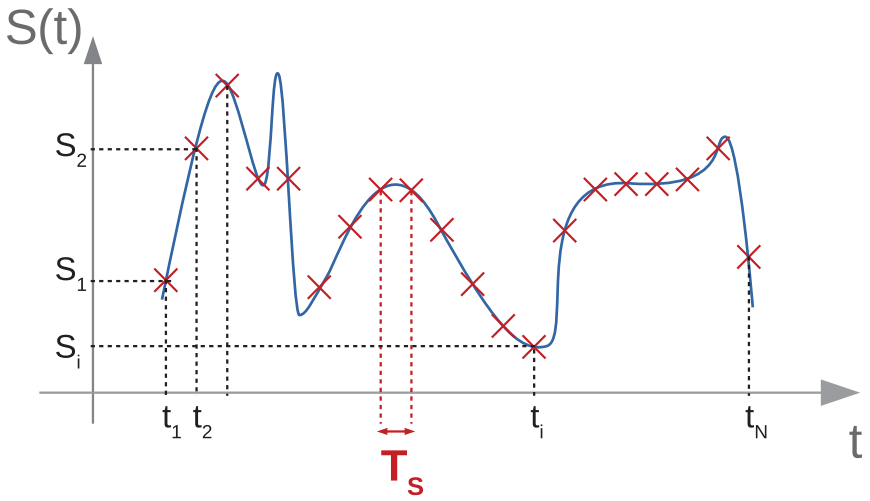


Figure 1.5 – Illustration de l'échantillonnage régulier d'une grandeur physique S dépendant du temps t.

On note T_S la période d'échantillonnage et $F_S = \frac{1}{T_S}$ la fréquence d'échantillonnage. Soit N le nombre de points de mesure effectués, la durée d'enregistrement s'écrit $\Delta t = (N-1) \times T_S$. On obtient donc deux séries de N valeurs : les mesures S_i aux dates t_i. L'interval de temps écoulé entre deux échantillons consécutifs est T_S et le temps s'écrit : $t_i = (i-1) \times T_S$. **On pourra, en machine, organiser ces données sous formes de tableaux ou matrices**.

1.3.2 Critère Nyquist-Shannon pour les signaux harmoniques

Qualitativement on voit bien dans la figure 1.5 que certaines variations rapides du signal ne seront pas capturées par l'échantillonnage. C'est, par exemple, flagrant à la figure 1.5 pour l'évolution du signal entre les échantillons #4 et #5 : l'échantillonnage «ne voit pas» le *pic* du signal réel. Il nous faut donc un critères pour connaitre les limites de l'échantillonnage.

De manière générale, le théorème dit de Nyquist-Shannon ou théorème d'échantillonage, donne une borne inférieur pour la fréquence d'échantillonnage, F_S, afin que la donnée de l'ensemble des echantillons, S_i, permette de reconstruire le signal sans ambiguité. Il faut pour cela que deux signaux différents ne donnent pas le même ensemble d'échantillons. Nous nous limiterons pour l'instant à une formulation limitée aux signaux périodiques :

Théorème 1.3.1. *Pour un signal harmonique, $S(t) = cos(2\pi ft + \phi)$, de fréquence f et de phase à l'origine ϕ et dépendant du temps t, et un échantillonnage régulier S_i de ce signal à la fréquence F_S, l'échantillonnage peut être considéré comme suffisant si : $F_S \geq 2 \times f$.*

3. En pratique la mesure n'est pas instantanée, mais on considèrera ici que le temps de la mesure est négligeable.

TP1 - PRISE EN MAIN DE GNU-OCTAVE : *APPRENDRE LES BASES DU LANGAGE PAR L'EXEMPLE*

Durant ces ateliers vous apprendrez les bases de la programmation avec le logiciel scientifique GNU-Octave. Vous utiliserez ce logiciel pour manipuler des données numériques échantillonnées (son, image, ...) et en faire l'analyse. Le logiciel GNU-Octave est libre et multi-plateforme, vous pouvez facilement l'installer sur un système linux ou windows. Notez qu'il existe une version en ligne de Octave qui peut être utilisée à l'adresse suivante : `https://octave-online.net/`.

Ce document est construit pour vous permettre d'aborder les bases du logiciel sans besoin d'autre documentation. Cependant, pour une utilisation plus poussée il vous est recommandé de compléter les informations de ce document en utilisant la documentation en ligne de Octave à l'adresse suivante : `http://octave.sourceforge.net/docs.html`.

Dans ce qui suit nous allons réaliser une série d'opérations qui vous permettront de comprendre les bases du langage. Nous aborderons comment faire des opérations simples sur les nombres, vecteurs et matrices. Nous apprendrons à accéder à certains éléments des vecteurs ou matrices et décrirons comment afficher et manipuler un graphique.

Démarrer avec GNU-Octave

La philosophie de GNU-Octave GNU-Octave est un logiciel de calcul scientifique. Comme Matlab, il est dédié au calcul numérique, c'est à dire qu'il n'est pas conçu pour réaliser du calcul formel. Contrairement à Maple ou Maxima, Matlab et GNU-Octave ne pratiquent pas de calcul symbolique : les valeurs numériques sont calculées.

La structure de base dans GNU-Octave est la matrice. Les opérations de base sur les matrices (produit, inversion, ...) sont pré-programmées. La syntaxe pour accèder aux éléments d'une matrice est simple. Les matrices peuvent être des vecteurs ligne ou colonne. Tout cela rend cette structure de données adaptée à l'analyse de signaux physiques. En outre, GNU-Octave pré-intègre un grand nombre d'opérations et de transformations mathématiques utiles à la physique.

GNU-Octave (comme Matlab) est aussi un véritable environement de programation dédié au calcul scientifique. GNU-Octave intègre un langage de programmation dit de «haut niveau ». C'est un langage interprété, par opposition aux langages compilés, qui ne nécessite pas la conversion préalable des commandes en langage machine pour leur exécution. En ce sens c'est un outil similaire

au langage Python muni de la librairie SciPy. On peut de manière dynamique créer des variables et leurs appliquer des opérateurs de manière interactive avec la console. Le langage de GNU-Octave permet aussi la plupart des structures algorithmiques telles que les boucles `for` et `while` ou les structures conditionnelles telles que `if` ou `switch`. Enfin, GNU-Octave offre la possibilité de créer des `function` pour construire sa propre librairie.

L'interface de GNU-Octave Octave dispose d'une interface graphique. Celle-ci est composée de plusieurs fenêtres ou consoles.

- Comme son nom l'indique la console de *«commande »* vous permet d'entrer les opérations à réaliser. C'est votre interface avec le logiciel.
- La console *«historique »* vous permet de relire les opérations déjà effectuées.
- La console *«espace de travail »* vous permet de visualiser les variables auquel vous aurez affecté des valeurs. Ces variables peuvent être affichées ou ré-utilisées. Les variables sont stockées en mémoire RAM.
- La console *«navigateur »* vous permet de charger ou d'enregistrer des fichiers. Le navigateur permet un accès vers «l'extérieur »du logiciel et d'accéder à des données pré-enregistrées ou issues d'autres programmes.
- La console *«éditeur »* est un éditeur de texte dédié à la création ou la modification de fichiers `.m`. Ce sont ces fichiers qui contiendront vos propres programmes et fonctions pouvant être interprétés par Octave.
- La console *«documentation »* vous permet de consulter l'aide. Cela vous permettra de connaître les syntaxes de fonctions précises et parfois complexes. C'est un outil essentiel pour la réussite dans ce cours mais aussi pour une utilisation dans votre future carrière. GNU-Octave est un outil puissant, vous aurez l'occasion de vous appuyer sur le travail d'une grande communauté pour vos propres applications. Il vous faudra pour cela bien utiliser l'aide.

Vous pouvez choisir de ne pas utiliser l'interface graphique. Le présent document est conçu pour que vous puissiez utiliser Octave sans cette interface graphique.

Lancement du programme avec interface graphique :

▷ ouvrez un terminal

▷ Entrez la commande : `octave --force-gui`

▷ Repérez les différentes consoles : «commande» ; «historique des commandes» ; «navigateur» ; «liste de variables».

▷ Ajustez les fenêtres à votre guise pour obtenir un environnement de travail qui vous semble confortable.

Lancement du programme sans interface graphique :

▷ ouvrez un terminal

▷ Entrez la commande : `octave --no-gui`

Quelques commandes utiles : Voici quelques commandes qui seront utiles pour la suite en mode console (sans interface graphique).

- variables utilisées : `whos` ou `who`
- répertoire courant : `pwd`
- enregistrement des commandes exécutées : `diary filename.txt`

La navigation dans l'arborescence de l'ordinateur se fait *via* des commandes similaires à unix.

- arborescence du répertoire courant : `ls`
- navigation dans les répertoires : `cd`
- création de répertoires : `mkdir`

Opérations simples et fonctions usuelles

Nouys allons ici commencer par les opérations de base d'Octave. Vous pouvez utiliser le logiciel comme une simple calculatrice. Nous en profiterons pour clarifier quelques options d'affichage.

▷ Entrez la commande : `2*3;`

▷ Entrez la commande : `2*3;`

Que constatez vous ? Dans la suite du document la présence de «;»sera important. En particulier, l'absence de «;»pourra conduire à un affichage interminable selon les situations...

▷ Entrez la commande : `1/3`

Attention ! Le séparateur décimal n'est pas une virgule mais un point...

▷ Entrez la commande : `cos(pi)`

▷ Entrez la commande : `cos(3.14)`

▷ Entrez la commande : `sin(pi)`

▷ Entrez la commande : `sin(3.14)`

▷ Entrez la commande : `2^2`

▷ Entrez la commande : `power(2,2)`

▷ Entrez la commande : `power(2,3)`

▷ Entrez la commande : `log(1)`

▷ Entrez la commande : `log(1)/log(10)`

▷ Entrez la commande : `exp(1)`

▷ Entrez la commande : `exp(log(1))`

▷ Entrez la commande : `mod(16,4)`

▷ Entrez la commande : `mod(17,4)`

Variables et types de données

Nous allons ici introduire la notion de variable. C'est un concept général en informatique et en programmation. On peut réserver un espace mémoire et stocker une ou plusieurs valeurs. De la même manière qu'il existe différents types de nombres (réels, complexes, entiers, ...), les variables peuvent être de différents types...

Préliminaires :

▷ Entrez la commande : `x=2*3;`

▷ Entrez la commande : `whos`
Que constatez vous [4] ?

▷ Entrez la commande : `x=2*3`
Que constatez-vous ?

▷ Entrez la commande : `x=3+4-1`

4. Pour les utilisateurs de l'interface graphique vous pouvez aussi vous reporter à la liste des variables de l'espace de travail.

▷ Entrez la commande : `x=x+1`

▷ Entrez la commande : `a=2`

▷ Entrez la commande : `y=power(x,a)`

▷ Entrez la commande : `whos`

▷ Entrez la commande : `clear x`

▷ Entrez la commande : `whos`
Que constatez-vous ?

▷ Entrez la commande : `y=power(x,a)` ?
Que constatez-vous ?

▷ Entrez la commande : `x=7`

▷ Entrez la commande : `y=power(x,a)` ?
Que constatez-vous ?

▷ Entrez la commande : `whos`

▷ Entrez la commande : `clear all`

▷ Entrez la commande : `whos`
Que constatez-vous ?

Les types de données

On distingue plusieurs types de données numériques.

- Les nombres réels double précision : `double`
- Les nombres réels simple précision : `single`
- Les entiers : `int8` ; `int16` ; `int32` ; `int64`
- Les nombres complexes `complex`

Octave prend aussi en charge des types de données non-numériques. Nous utiliserons assez peu ce types de données. A vous de vous documenter sur ceux-ci...

- Les caractères : `char`
- Les cellules : `cell`

Exemples

▷ Entrez la commande : `x=1`

▷ Entrez la commande : `whos`

▷ Entrez la commande : `x=1.0`

▷ Entrez la commande : `whos`

▷ Entrez la commande : `x=single(1)`

▷ Entrez la commande : `whos`

▷ Entrez la commande : `k=int16(1)`

▷ entrez la commande : `a=int64(1)`

▷ Entrez la commande : `whos`

▷ Entrez la commande : `i`

▷ Entrez la commande : `z=complex(1,2)`

▷ entrez la commande : `zz=complex(2,2)`

▷ Entrez la commande : `whos`

▷ Entrez la commande : `abs(z)`

▷ entrez la commande : `angle(z)`

Un peu de souplesse ... peu entrainer de la confusion Contrairement à certains langages Octave ne fige pas le type des variables. Cela peut être pratique à l'usage mais il faut faire attention... Cela demande une certaine maîtrise de ce que vous ferez...

- ▷ Entrez la commande : `x=1`
- ▷ Entrez la commande : `whos`
- ▷ Entrez la commande : `x=int8(1)`
- ▷ Entrez la commande : `whos`

- ▷ Entrez la commande : `clear all`
- ▷ Entrez la commande : `i`
- ▷ Entrez la commande : `2+i`
- ▷ Entrez la commande : `whos`
- ▷ Entrez la commande : `i=1`
- ▷ Entrez la commande : `2+i`
- ▷ Entrez la commande : `whos`

Vecteurs, matrices et opérations

Vous savez maintenant initialiser une variable et effectuer des opérations avec les fonctions et opérateurs mathématiques usuels. Nous allons aborder l'utilisation de vecteurs et matrices.

Vecteurs

Initialisation de vecteurs

- ▷ Entrez la commande : `x=[1,2,3,4,5,6]`
- ▷ Entrez la commande : `whos`
- ▷ Entrez la commande : `x=[1;2;3;4;5;6]`
- ▷ Entrez la commande : `whos`
- ▷ Entrez la commande : `x=[1,2,3,4,5,6]'`
- ▷ Entrez la commande : `whos`

- ▷ Entrez la commande : `y=[1:6]`
- ▷ Entrez la commande : `y=[1:2:13]`
- ▷ entrez la commande : `z=[1:0.1:2]`

Opérations sur les vecteurs

- ▷ Entrez la commande : `x*y`
- ▷ Entrez la commande : `y*x`
  Que constatez vous ?

- ▷ Entrez la commande : `y=y'`
- ▷ Entrez la commande : `whos`
- ▷ Entrez la commande : `x.*y`
  Que constatez vous ?

- ▷ Entrez la commande : `cos(z)`
- ▷ Entrez la commande : `power(y,2)`
  Que constatez vous ?

accédez aux éléments d'un vecteur

- ▷ Entrez la commande : `clear all`
- ▷ Entrez la commande : `x=[1:10]`
- ▷ Entrez la commande : `x(1,10)`
- ▷ Entrez la commande : `x(1,1)`
- ▷ Entrez la commande : `x(10,1)`
  Que constatez vous ?
- ▷ Entrez la commande : `x=x'`
- ▷ Entrez la commande : `whos`
- ▷ Entrez la commande : `x(10,1)`
  Que constatez vous ?

Matrices

Initialisation de matrices et opérations sur les matrices

- ▷ Entrez la commande : `M1=[1,2,3;1,2,3]`
- ▷ Entrez la commande : `M2=[1,2;1,2;1,2]`
  Que constatez vous ?
- ▷ Entrez la commande : `M1*M2`
- ▷ Entrez la commande : `M2*M1`
  Que constatez vous ?
- ▷ Entrez la commande : `M3=M2'`
- ▷ Entrez la commande : `M1*M3`
- ▷ Entrez la commande : `M1.*M3`
  Que constatez vous ?
- ▷ Entrez la commande : `ones(2,2)`
- ▷ Entrez la commande : `zeros(2,2)`

Accéder à des éléments d'une matrice ou d'un vecteur

- ▷ Entrez la commande : `y=[1:2:13]'`
- ▷ Entrez la commande : `x=[1:10]'`
- ▷ Entrez la commande : `M4=x*y'`
- ▷ Entrez la commande : `M4(1,1)`
- ▷ Entrez la commande : `M4(3,5)`
  Que constatez vous ?
- ▷ Entrez la commande : `M4(1:7,1)`
- ▷ Entrez la commande : `M4(1:end,1)`
  Que constatez vous ?
- ▷ entrez la commande : `M4(1:3,1)`
- ▷ entrez la commande : `M4(1:3,:)`
- ▷ Entrez la commande : `M4(1,:)`
- ▷ Entrez la commande : `M4(:,2)`

▷ Entrez la commande : `M4(:,1:2:7)`
Que constatez vous ?

▷ Entrez la commande : `x`

▷ Entrez la commande : `x(2,1)`

▷ Entrez la commande : `x(2)`

▷ Entrez la commande : `x(1,2)`
Que constatez vous ?

▷ Entrez la commande : `x(1:7)`

▷ Entrez la commande : `x(1:2:7)`

Algèbre matriciel

▷ Entrez la commande : `v1=[1;0]`

▷ Entrez la commande : `v2=[0;1]`

▷ Entrez la commande : `R=[0,1;1,0]`

▷ Entrez la commande : `R*v1`

Tracer de Graphiques

Dans cette section nous allons apprendre à afficher et éditer des graphiques.

Créer un graphique

▷ Entrez la commande : `clear all`
Que constatez vous ?

▷ Entrez la commande : `t=[0:0.001:3];`

▷ Entrez la commande : `f0=1`

▷ Entrez la commande : `w0=2*pi*f0`

▷ Entrez la commande : `signal=sin(w0*t);`

▷ Entrez la commande : `figure; plot(t,signal)`

▷ Entrez la commande : `plot(t,signal,'r')`

▷ Entrez la commande : `signal2=cos(w0*t)`

▷ Entrez la commande : `plot(t,signal2,'k')`
Que constatez vous ?

▷ Entrez la commande : `figure; plot(t,signal2,'.r')`

▷ Entrez la commande : `plot(t,signal2,'-.-c')` Que constatez vous ?

Afficher plusieurs fonctions

▷ Entrez la commande : `figure; plot(t,signal,'r',t,signal2,'k')`

▷ Entrez la commande : `hold on; plot(t,cos(2*w0*t),'b')`

Changer les échelles

▷ Entrez la commande : `axis([0,1,0,3])`

▷ Entrez la commande : `axis([0,3,0,1])`

▷ Entrez la commande : `axis([0,3,-1,1])`

TP2 - FONCTIONS ET ALGORITHMIQUE SOUS GNU-OCTAVE

Maintenant que vous connaissez les rudiments du logiciel, vous devez avoir envie d'écrire vos propres fonctions ! Ceci est assez simple avec le logiciel Octave. Il vous suffit pour cela de créer un fichier texte avec pour extension `.m`. Ce fichier peut-être créé dans votre éditeur de texte préféré ou dans l'éditeur intégré à GNU-Octave [5]. L'extension `.m` assure une compatibilité avec le logiciel *Matlab* (The MathWorks Inc.) qui est un équivalent commercial à Octave. Notez à ce propos qu'Octave est conçu pour être le plus compatible possible avec *Matlab*.

Insistons sur le fait que vous devrez indiquer au logiciel GNU-Octave le chemin d'accès vers les programmes que vous écrirez et appellerez. En général il suffit de faire de ce chemin d'accès le répertoire courant de GNU-Octave. Il vous faudra alors apprendre à naviguer dans l'arborescence de la machine pour appeler vos propres programmes [6].

Nous aborderons la description des scripts et des fonctions (`function`). GNU-Octave permet la défintion de `scripts` qui sont une successions de commandes que vous pouvez lister dans un fichier. Les scripts permettent aussi la construction de structures algorithmiques. La notion de fonction est très importante en programmaton. Nous la verrons plus en détail.

Notez enfin que le nom de fichier doit correspondre avec le nom de la fonction.

Mon premier script Octave

▷ Créez un fichier `script_hello_world.m` et ouvrez-le.

▷ Dans ce fichier entrez le texte suivant et enregistrez vos modifications.

```
printf ( " hello\n " );
printf ( " world\n " );
a=1
b=2
c=a+b
```

5. L'éditeur gedit par exemple comprend par défaut la coloration syntaxique adaptée au langage de GNU-Octave.

6. Notez que pour une utilisation avancée il est possible d'indiquer à GNU-Octave une série de répertoire dans lesquels vous archivez vos fonctions. Ceci se fait *via* la variable d'environnent `path`. Ce ne sera pas nécessaire dans le cadre de ce cours et nous vous laissons le soin de vous documenter sur ce point.

▷ Dans Octave, **placez vous dans le répertoire qui contient votre script,** entrez la commande : `script_hello_world`

Ma première fonction Octave

▷ Créez un fichier `hello_world.m` et ouvrez-le.

▷ Dans ce fichier entrez le texte suivant et enregistrez vos modifications.

```
function hello_world
printf ( " hello\n " );
printf ( " world\n " );
a=1
b=2
c=a+b
endfunction
```

▷ Dans Octave, **placez vous dans le répertoire qui contient votre script,** entrez la commande : `hello_world`

▷ Toujours dans Octave, entrez la commande : `whos`
Est-ce que les variables `a`, `b` et `c` existent ?

Entrées et Sorties d'une fonction

Un premier exemple : fonction avec sortie uniquement

▷ Créez un fichier `somme12.m` et ouvrez le.

▷ Dans ce fichier entrez le texte suivant et enregistrez vos modifications.

```
function [a]=somme12
b=1;
c=2;
a=b+c;
endfunction
```

▷ Dans Octave, **placez vous dans le répertoire qui contient votre script,** entrez la commande : `somme12`
Que constatez vous ?

▷ Toujours dans Octave, entrez la commande : `whos`
Est-ce que les variables `a`, `b` et `c` existent ? Quelle est la différence avec un script ? Pourquoi ?

▷ Toujours dans Octave, restez dans le répertoire qui contient votre script, entrez la commande : `d=somme12`
Que constatez vous ?

▷ Toujours dans Octave, restez dans le répertoire qui contient votre script, entrez la commande : `a=somme12`
Que constatez vous ?

Un second exemple : deux entrées, une sortie

▷ Créez un fichier `somme.m` et ouvrez le.

▷ Dans ce fichier entrez le texte suivant et enregistrez vos modifications.

```
function [a]=somme(b,c)
```

```
a=b+c;
endfunction
```

▷ Dans Octave, **placez vous dans le répertoire qui contient votre script,** entrez la commande : `somme(1,2)`. Que constatez vous ?

▷ Toujours dans Octave, entrez la commande : `whos`
Est-ce que les variables `a`, `b` et `c` existent ? Pourquoi ?

▷ Toujours dans Octave, restez dans le répertoire qui contient votre script, entrez la commande : `b=1; c=2; a=somme(b,c)`. Que constatez vous ?

▷ Toujours dans Octave, restez dans le répertoire qui contient votre script, entrez la commande : `d=4670; e=somme(a+d)`. Que constatez vous ?

Un troisième exemple : deux entrées, deux sorties

▷ Créez un fichier `somme_et_difference.m` et ouvrez le.

▷ Dans ce fichier entrez le texte suivant et enregistrez vos modifications.
```
function [s,d]=somme_et_difference(n1,n2)
s=n1+n2;
d=n1-n2;
endfunction
```

▷ Dans Octave, **placez vous dans le répertoire qui contient votre script,** entrez la commande : `somme_et_difference(1,2)`. Que constatez vous ?

▷ Dans Octave, **placez vous dans le répertoire qui contient votre script,** entrez la commande : `[ss,dd]=somme_et_difference(1,2)`. Que constatez vous ?

▷ Toujours dans Octave, entrez la commande : `whos`
Est-ce que les variables `s` et `d` existent ? Pourquoi ?

Boucle `for`

L'ordinateur n'est pas bien malin. C'est donc à lui qu'il faut faire faire les tâches répétitives... Une structure utile en programmation est la boucle dites `for`, c'est à dire «pour » en français. Cette structure permet de répéter une opération ou une série d'opérations un nombre N de fois.

Ecrire une fonction factoriel

Par exemple, pour calculer le factoriel d'un entier N on pourrait utiliser l'algorithme suivant :

1. attribuons la valeur N à la variable `resultat`
2. `pour` un entier j variant de 1 à $N-1$, multiplier `resultat` par j

Ce qui s'écrit en langage octave :
```
resultat=N
for j=1:N-1
  resultat=resultat*j
end
```

▷ Ecrivez une fonction qui prend un entier en entrée et renvoi en sortie le factoriel de ce nombre.

A vous de jouer...

▷ Ecrivez un programme qui prend un nombre entier N en entrée et qui renvoi la somme des N premiers entiers. Testez ce programme.

▷ Ecrivez un programme qui prend en entrée un tableau (1D) de nombres x_i et qui renvoi la moyenne, $< x_i >$ des x_i.

Structure `if`

Une structure utile en programmation est la possibilité de réaliser une opération si une condition est réalisée et une autre opération dans le cas contraire.

Ecrire une fonction valeur absolue

Elle pourrait s'écrire en langage Octave :

```
if x<0
  resultat=-x;
else
  resultat=x;
end
```

▷ Ecrivez une fonction qui prend un nombre en entrée et renvoi en sortie sa valeur absolue.

A vous de jouer...

...Aux dés. On cherche à réaliser un programme qui simule le lancé d'un dé à 6 faces. Pour cela nous utiliserons la fonction intégrée `rand`. La commande `x=rand` place dans la variable `x` un nombre pseudo-aléatoire uniformément distribué entre 0 et 1. Notons que les nombres générés par `rand` ne sont pas à strictement parler aléatoires. Ils sont en fait issus d'une série numérique dont la «mémoire »(correlation) entre des nombres consécutifs est extrêmement faible et ils apparaissent aléatoires si le nombre d'appel de la fonction n'est pas trop grand[7]. Si l'on souhaite générer la même séquence à chaque fois que le programme est lancé[8] il faut intialiser le générateur de nombres pseudo-aléatoires par une graine à l'aide de la commande : `rand("seed",0)` placée au début du programme.

▷ A titre préliminaire, utilisez la condition `if` et la fonction `rand` pour écrire un programme qui renvoit soit le nombre 0 ou le nombre 1 (et uniquement ces deux nombres) avec la même probabilité.

▷ Appellez 1000 fois ce programme, ranger les nombres dans un tableau et vérfier l'équiprobabilité.

▷ Ecrivez maintenant un programme qui renvoit de manière équiprobable les nombres 1, 2, 3, 4, 5 et 6 (et uniquement c'est 6 nombres). Pour cela vous vous reporterez à la documentation de GNU-Octave et utiliserez le teste de condition `elseif`.

▷ Appellez 1000 fois ce programme, ranger les nombres dans un tableau et vérfier l'équiprobabilité.

7. La période des générateurs le plus simple est de l'ordre de 2^{31}.
8. ce qui est utile si l'on souhaite un programme répétable et reproductible.

2

SÉRIES DE FOURIER

Avant d'entrer dans le vif du sujet de l'analyse spectrale nous allons rappeler ici la notion de série de Fourier. Elle trouve son origine au 18ème siècle pour l'interpolation de fonctions périodiques en astronomie, la description des cordes vibrantes et la *théorie de la chaleur* de Fourier (1822).

2.1 Définitions et Théorème de Dirichlet

Soit une fonction $f(t)$ dépendante du temps $t \in \mathbb{R}$ qui prend des valeurs réelles. Soit un réel T_0 tel que :

$$f(t + T_0) = f(t) \quad \forall t \in \mathbb{R} \tag{2.1}$$

Le fonction f et dite périodique de période T_0 sur $\mathbb{R}$. La grandeur $f_0 = \frac{1}{T_0}$ est nommée la fréquence.

Soit une fonction $f(t)$ périodique, considérons la décomposition suivante :

$$f(t) = \sum_{k=-\infty}^{\infty} c_k(f) \times e^{j2\pi \frac{k}{T_0} t} = c_k(f) \times e^{j2\pi k f_0 t} \tag{2.2}$$

C'est la série de Fourier complexe associée à la fonction f et les coefficients $c_k(f)$ sont les coefficients de Fourier.

La représentation du module du coefficient de Fourier, $|c_k(f)|$, en fonction de la fréquence $f = \frac{k}{T_0} = k \times f_0$ est le spectre en amplitude de la fonction f. La fréquence f_0 est nommée fréquence fondamentale et les fréquences $k \times f_0$ sont les harmoniques.

Théorème 2.1.1. *Pour toute fonction périodique de période T_0, continue en t et dérivable à gauche et à droite en t, la série de Fourier converge et il y a égalité entre la série et la fonction.*

2.2 Calcul des coeffcients

Les coefficients de Fourier complexes sont donnés par la relation :

$$c_n(f) = \frac{1}{T_0} \times \int_{t=0}^{T_0} f(t) \times e^{j2\pi n f_0 t} dt \quad \forall n \in \mathbb{Z} \tag{2.3}$$

Démonstration : Pour s'en convaincre, on montrera d'abord que l'intégrale sur une période du produit de deux fonctions de base $e^{j2\pi kf_0t} \times e^{j2\pi lf_0t}$ est nul si $k \neq l$. Ensuite il suffira de multiplier l'expression (2.2) par $e^{j2\pi\frac{n}{T_0}t}$ et d'intégrer.

Si f est une fonction paire, on montre que :

$$c_{-n}(f) = c_n(f) \quad \forall n \in \mathbb{Z} \tag{2.4}$$

Démonstration : A vous de jouer, quelques lignes...

Si f est une fonction impaire, on montre que :

$$c_{-n}(f) = -c_n(f) \quad \forall n \in \mathbb{Z} \tag{2.5}$$

Démonstration : A vous de jouer, quelques lignes...

2.3 Série de Fourier Réelle

On montre que :

$$f(t) = a_0(f) + \sum_{k=1}^{\infty} a_k(f) \times cos(2\pi kf_0t) + b_k(f) \times sin(2\pi kf_0t) \tag{2.6}$$

Démonstration :

- En utilisant les relation d'Euler, démontrez la relation (2.6) ;
- Donnez l'expression des coefficients a_n et b_n en fonction des c_n ;
- Donnez la définition intégrale des coefficients $a_n(f)$ et $b_n(f)$.

2.4 Exercices

Ces exercices sont préparatoires au TP3.

Signal carré

- Calculez les coefficients de Fourier complexes d'un fonction créneau périodique, de période T_0, d'amplitude A et dont la valeur moyenne est nulle. choisissez le signal pair pour cet exemple.
- Même question en prenant le même signal mais impair.
- Donnez l'expression des coefficients a_n et b_n ;

Signal triangle

- Calculez les coefficients de Fourier complexes d'un fonction triangle périodique, de période T_0, d'amplitude A et dont la valeur moyenne est nulle. choisissez le signal pair pour cet exemple.
- Même question en prenant le même signal mais impair.
- Donnez l'expression des coefficients a_n et b_n ;

TP3 - ALGORITHMIQUE SOUS GNU-OCTAVE : *APPLICATION À LA SYNTHÈSE DE SIGNAUX*

Phénomène de Gibbs : *Effet de troncature de la série de Fourier*

Dans cette partie nous allons reconstruire un signal périodique à partir de la série de Fourier. Pour cela nous utiliserons une boucle *for*. Évidement nous ne pourrons pas générer des sommes infinies. Nous nous intéresserons donc à l'effet de la troncature de la série sur le signal.

Signal Carré

- ▷ Définir la fréquence d'échantillonnage : `Fs=44100`
- ▷ Définir la période d'échantillonnage : `Ts=1/Fs`
- ▷ Créér un vecteur temps d'une durée de 0.5s : `temps=[0:Ts:0.5];`
  Vous pouvez visualiser le vecteur temps pour comprendre cette commande. Celle-ci sera importante tout au long des TP.
- ▷ En vous appuyant sur les exercices de 2.4, donner l'expression des coefficients c_n pour un signal créneau périodique, de valeur moyenne nulle, d'amplitude A et de fréquence f_0. On pourra choisir le signal pair ou impair pour faciliter les calculs.
- ▷ On prendra dans la suite $A = 1V$ et $f_0 = 440$Hz, entrez les commandes : `f0=440` et `A=1`
- ▷ Créer une fonction : `function [C0,CN_pos,CN_neg]=coef_fourier_carre(A,f0,NH)`
  Celle-ci renverra en sortie les coefficients c_n, et prendra en entrée l'amplitude `A`, la fréquence `f0` et le nombre d'hamorniques désirées `NH`.
- ▷ Créer une fonction : `function [signal]=somme_harmo(C0,CN_pos,CN_neg,temps)`
  Celle-ci renverra en sortie la somme des `NH` premières harmoniques du signal carré en fonction du `temps`, et prendra en entrée les coefficients de Fourier (`C0`, `CN_pos`, `CN_neg`), et le `temps`.
- ▷ Afficher sur un même graphique la première harmonique en noir, la somme des 5 premières harmoniques en rouge, la somme des 15 premières harmoniques en vert.

Comparaison

- ▷ Générez une série de vecteurs `C0_al`, `CN_pos_al`, `CN_neg_al` de manière aléatoire avec la fonction `rand`. Utilisez la documentation en ligne pour maitriser la taille des vecteurs.
- ▷ utilisez la fonction `somme_harmo` que vous avez créé pour générer un signal à partir de ces composantes aléatoires.

▷ Afficher sur un méme graphique la première harmonique en noir, la somme des 5 premières harmoniques en rouge, la somme des 15 premières harmoniques en vert.

Signal Triangle

▷ Afficher sur un même graphique la première harmonique en noir, la somme des 5 premières harmoniques en rouge, la somme des 15 premières harmoniques en vert, cette fois pour un signal triangle.

Synthèse de signaux sonnores

Dans cet exercice vous allez synthétiser des sons qui vous permettrons de construire une mélodie comme un synthétiseur ! Vous commencerez par générer des sons harmoniques (de la forme $s(t) = A \times sin(2\pi f_0 t)$). Ensuite nous utiliserons un modèle de corde vibrante pour générer des sons plus complexes, plus réalistes.

Préambule Une mélodie est composée de notes et de silences. Un même instrument joue les notes de manière séquentielle pour la plupart. Les noires correspondent à un temps et les blanches à deux temps. Nous prendrons ici un tempo de 110 (110 temps par minutes). C'est à dire qu'un temps dure environ 0.5s.

Il existe plusieurs gammes qui découpent l'octave en plusieurs notes. Voici quelques exemples de découpages et de fréquences associées :

note	ton juste (Hz)	Gamme Pythagoricienne (Hz)	Gamme Tempérée (Hz)
do	264,00	260,74	261,63
do#	275,00	278,44	277,18
ré	297,00	293,33	293,66
mi b	316,80	309,03	311,13
mi	330,00	330,00	329,63
fa	352,00	347,65	349,23
fa#	371,25	371,25	369,99
sol	396,00	391,11	392,00
sol#	412,50	417,66	415,30
la	440,00	440,00	440,00
si b	475,20	463,54	466,16
si	495,00	495,00	493,88
do	528,00	521,48	523,25

Synthétiseur sans timbre : *sons harmoniques* Vous commencerez par générer des sons harmoniques de la forme $s(t) = A \times sin(2\pi f_0 t)$.

▷ Définir la fréquence d'échantillonnage : `Fs=44100`

▷ Créer une fonction `function [signal]=son_harmo(f0,A,FS,DT)` qui à partir de la donnée de la fréquence de la note `f0`, d'une amplitude `A`, d'une fréquence d'échantillonnage `FS`, et d'une durée de note `DT`, renvoi `signal` correspondant à la note désirée.

▷ Entrez la commande : `signal_la=son_harmo(440,1,Fs,0.5);`

▷ Visualisez le signal en fonction du temps.

▷ Enregistrez votre signal à l'aide de la commande `wavwrite(signal_la,Fs,'monla.wav')`

▷ Écoutez votre note...

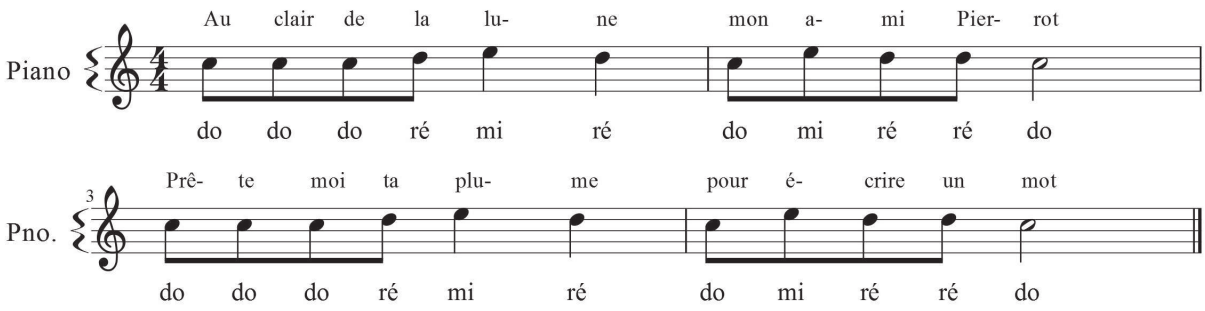
▷ Entrez la commande : `signal_do=son_harmo(275,1,Fs,0.5);`

▷ A l'aide de la commande `cat`, créer une variable signal qui comprendra les deux notes à la suite l'une de l'aute.

▷ Enregistrez votre signal à l'aide de la commande `wavwrite(signal_la,Fs,'monla.wav')`

▷ Écoutez votre son...

▷ Créez un script qui générera le signal correspondant à la mélodie dont la partition est ci-dessous. Vous enregistrerez ce signal au format `.wav`.
Attention une blanche correspond à deux temps soit le double de durée d'une noire et une croche correspond à un demi temps soit la moitié de la durée d'une noire...



Synthétiseur avec timbre : *sons complexes* Le timbre d'un instrument correspond à la présence d'harmoniques en plus de la fréquence fondamentale. Ces harmoniques sont dues à la manière dont l'instrument est stimulé ainsi qu'au cavité résonnantes qui le compose. Nous nous contenterons ici de simuler une corde sans résonateur.

On montre que la solution à l'état stationnaire pour la vibration d'une corde (non-élastique, tendue par une force de tension de norme T de chaque coté) s'écrit sous la forme d'une superposition de fonctions élémentaires appelées modes propre de vibration :

$$u(x,t) = \sum_{k=-\infty}^{+\infty} \{C_k.e^{j(k.2\pi f_0.t)}\} \times sin(k.\beta_0.x) \tag{2.7}$$

où f_0 et β_0 sont des constantes qui dépendent de la tension, de la longueur de la corde et de sa masse linéique. x est la position le long de la corde et t est le temps. La vibration produite par un point de la corde est donc du type :

$$\delta p(t) = \sum_{k=-\infty}^{+\infty} \{C_k.e^{j(k.2\pi f_0.t)}\} \tag{2.8}$$

Pour une corde pincée sans vitesse initiale, on montre que :

$$c_k = U_0 \times \frac{8}{\pi^2 k^2} \begin{cases} (-1)^{(k-1)/2}, & k = 2n+1 \\ 0, & k = 2n \end{cases} \tag{2.9}$$

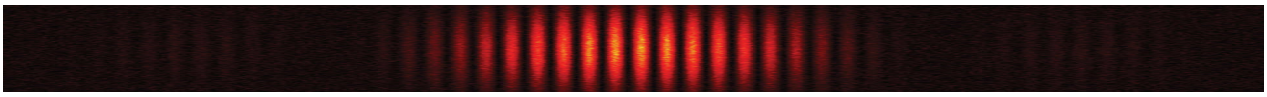
et que :

$$c_{-k} = c_k \tag{2.10}$$

où U_0 est une constante.

▷ Générer la mélodie de «Au clair de la lune » avec le timbre d'une corde pincée.

TP4 - IMAGES SOUS GNU-OCTAVE : *APPLICATION À LA MESURE D'INTERFRANGES*



Dans ce TP nous allons importer des données sous forme d'image sous GNU-Octave afin d'y appliquer un traitement numérique. Nous prendrons l'exemple d'une expérience d'interférences en optique pour aborder plusieurs situations typiques en traitement de signaux physiques.

Objectif : Ecrire un programme permettant d'analyser de manière systématique l'enregistrement d'un motif d'interférences optiques :

- ▷ Importer des données expérimentales sous forme d'image et les visualiser ;
- ▷ Repérer les maxima locaux d'un signal en écrivant une fonction ;
- ▷ Déterminer l'interfrange d'une acquisition ;
- ▷ Remonter à la distance physique entre les 2 fentes dans l'expérience ;
- ▷ Visualiser les résultats de manière claire et précise.

Pour réaliser ce TP, il est vivement conseillé de se référer à la riche documentation en ligne décrivant les fonctions d'Octave (`https://octave.org/doc/`, `https://octave.sourceforge.io/docs.php`)

L'expérience des fentes d'Young

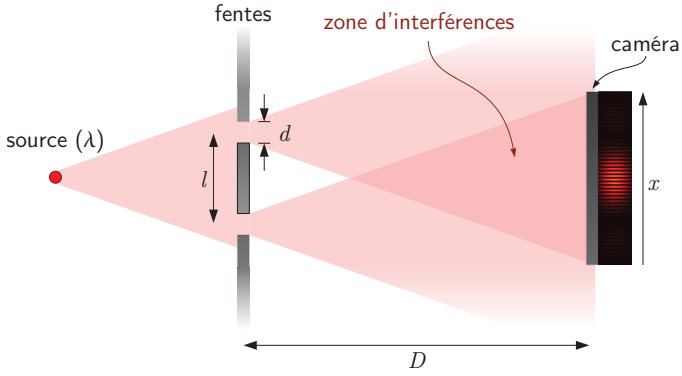
L'expérience des fentes de Young, réalisée pour la première fois en 1801 par Thomas Young, met en évidence de manière spectaculaire la nature ondulatoire de la lumière. Le principe, que vous connaissez bien, est le suivant : On éclaire à l'aide d'un faisceau de lumière monochromatique deux fentes étroites et proches l'une de l'autre. On observe sur un écran placé après les fentes une figure d'interférences où s'alternent franges brillantes et franges sombres. L'interfrange (distance séparant deux franges brillantes ou sombres) est directement lié à la distance séparant les fentes et aux paramètres expérimentaux. Le contraste des franges qui diminue lorsque l'on s'éloigne du centre de l'écran et lui lié à la figure de diffraction des ouvertures.
Par le calcul, on trouve relativement facilement que l'évolution de l'intensité de la lumière le long

de la figure d'interférence/diffraction prend la forme suivante :

$$I(x) = I_0 \cdot \mathrm{sinc}^2\left(\frac{\pi l}{\lambda D}x\right) \cdot \cos^2\left(\frac{\pi d}{\lambda D}x\right)$$

où x est la position sur l'écran selon l'axe perpendiculaire au franges, I_0 est l'intensité au centre de la figure, λ la longueur d'onde de la lumière, D la distance séparant les fentes de l'écran, l la largeur des fentes, et d la distance les séparant.

Dans ce TP, vous aurez à votre disposition un fichier contenant une figure d'interférence que l'on peut typiquement obtenir dans cette expérience à l'aide d'une caméra CCD. **A vous d'écrire un programme qui mesure précisément l'interfrange et remonte à la distance entre les fentes !** Pour les paramètres, vous prendrez les valeurs numériques indiquées dans le tableau ci-dessous.



Paramètres expérimentaux	
longueur d'onde (λ)	632nm
distance fentes-caméra (D)	15cm
taille d'un pixel	5μm

Import des données et visualisation

▷ Hors d'Octave créez un dossier `Programme_interferences` et un fichier `TP4_interferences.m`. Créez un sous dossier `data` et déplacez l'acquisition caméra `franges_interf.png` dans ce dernier.

▷ Dans Octave, placez vous dans le répertoire qui contient votre script.

La fonction permettant d'importer une image sous Octave est `imread()`.

▷ Dans votre script tapez la ligne de code suivante et compilez

```
acquisition = imread('data/franges_interf.png');
```

Que constatez-vous ?

Vous allez créer les variables x et y qui seront les coordonnées d'abscisse et d'ordonnée de l'acquisition.

▷ Dans votre script ajoutez les lignes de code suivantes et compilez

```
x = 1:size(acquisition, 2);
y = 1:size(acquisition, 1);
```

De cette façon x et y ont automatiquement les dimensions de l'image.

Pour visualiser les données dans Octave, vous allez créer un objet figure et utiliser la fonction `imagesc` pour afficher une version *en fausses couleurs* de la matrice `acquisition`.

▷ Dans votre script ajoutez les lignes de code suivantes et compilez

```
figure,
imagesc(x, y, acquisition)
colorbar
```

Que constatez-vous ? La commande `colorbar` affiche sur la droite de la figure le code couleur associant la couleur de chaque pixel à une valeur numérique.
Quelle est la valeur maximale de la matrice ? Pourquoi ?
Par défaut la barre de couleur associe les valeurs de la matrice de la plus faible à la plus grande à des couleurs allant du bleu au jaune, mais il existe beaucoup d'autre répartitions. De manière générale, la barre de couleur la plus adaptée dépendra de la nature des données à visualiser. Changez la colorbar en ajoutant la ligne suivante à votre code :

```
colormap(gray)
```

En programmation scientifique comme sur papier, **une figure doit contenir des axes avec des unités et un titre !** Dans octave ils sont définis respectivement avec les commandes suivantes que vous ajoutez à votre programme :

```
xlabel('Position horizontale (pixel)')
ylabel('Position verticale (pixel)')
title("Intensité lumineuse")
```

Détection des maxima d'intensité de la figure d'interférence

Votre objectif est de détecter de manière automatique les maxima d'intensité des franges d'interférence pour déterminer une valeur précise de celui-ci. Pour cela la méthode la plus simple est de sélectionner une coupe horizontale de la figure et repérer les maxima. Vous allez dans un premier temps programmer ceci pour vous rendre compte que cette méthode n'est pas viable. Nous verrons après comment y remédier.

Détection sur une ligne horizontale

▷ Vous allez tout d'abord tracer dans une nouvelle figure une coupe horizontale choisie au mileu de la figure d'interférence. Pour cela on défini la variable `position_verticale` qui contient la coordonnée verticale en pixel de la coupe que vous voulez tracer et dont vous définirez la valeur (`XXX` dans le code ci-après). Vous utiliserez ensuite la commande `plot()` pour tracer la figure.

```
position_verticale = XXX;
figure,
plot(x, acquisition(position_verticale, :))
xlabel('Position horizontale (pixel)')
ylabel('Intensité (u. a.)')
title(["Intensité le long de la ligne y = " num2str(position_verticale) " px"])
```

Lisez attentivement ce code pour bien comprendre comment fonctionne la fonction `plot()` (Quels sont ces arguments ?). La fonction `num2str()` est très utilise puisqu'elle converti une valeur numérique (ici la valeur contenue dans la variable `position_verticale`) en une chaine

de caractères. Cela permet d'afficher systématiquement la hauteur de la coupe au titre de la figure. Notez la façon dont 3 chaînes de caractères sont concaténées pour former le titre.

▷ Modifier la valeur de la variable `position_verticale` et relancez le code pour vérifier son bon fonctionnement.

▷ **A vous de jouer !**

Dans un nouveau fichier fonction `trouver_max_ligne.m`, créez une fonction `trouver_max_ligne` qui prend comme argument un tableau à une dimension (une coupe de la figure) et qui retourne un tableau contenant les indices (positions en pixel) des maxima locaux. Pour cela vous adopterez la stratégie suivante : la fonction balaye chaque valeur du tableau d'entrée en commençant par sa deuxième valeur et en terminant à l'avant dernière (boucle `for`). **Si** (boucle `if`) **la valeur testée est plus grande que** la valeur précédente **ET** la valeur suivante du tableau, alors il s'agit d'un maximum local. Vous stockez son indice dans un tableau qui sera la sortie de la fonction.

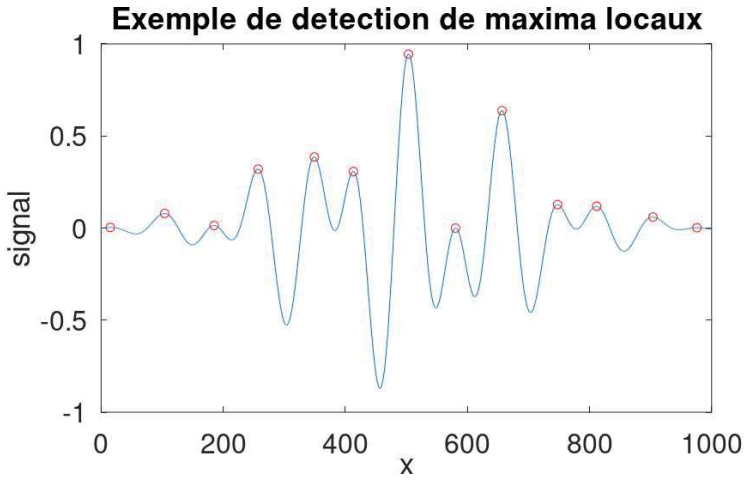
Il faut toujours tester ses fonctions sur des exemples simples dont vous pouvez confirmer facilement le résultat. Ici, vous pouvez créer dans un script séparé un tableau d'abscisse x :

```
x = linspace(0, 1000, 1000);
```

et un signal qui serait par exemple une gaussienne de largeur typique 25 centrée en 250. En appelant votre fonction avec comme argument votre signal `XXX` et comme variable de sortie `YYY` (A compléter vous même), vérifiez le bon fonctionnement en traçant le résultat dans une figure

```
figure,
plot(x, XXX), hold on
plot(x(YYY), sig(YYY), 'or')
```

Ce code trace votre signal `XXX` et superpose (`hold on`) le(s) maximum(s) détectés sous forme de cercle rouge (`'or'`).
Testez votre fonction dans plusieurs situations en modifiant le signal. Par exemple 2 gaussiennes d'amplitudes, de largeurs et de positions centrales différentes, une fonction cosinus, un produit de fonctions cosinus et sinus, etc...



▷ Une fois votre fonction `trouver_max_ligne` éprouvée, appliquez-la à la coupe horizontale choisie au milieu de la figure d'interférence et tracez le résultat comme précédemment.

Que constatez-vous ? Qu'est-ce-qui ne va pas ? Identifiez deux problèmes qui nuisent à la mesure de l'interfrange.

Détection sur un signal moyenné

En sélectionnant une seule ligne horizontale dans l'acquisition expérimentale, le signal à traiter est "bruité". Pour réduire l'impact de se bruit, on peut calculer un nouveau tableau à une dimension contenant la moyenne sur l'ensemble des lignes de la variable `acquisition`.

- ▷ Utilisez la fonction `mean()` pour créer une nouvelle variable contenant le profil d'interférence moyenné sur les lignes horizontales. Pour info, la fonction `mean()` prend deux arguments en entrée, un tableau à N dimensions et un entier indiquant la dimension selon laquelle la moyenne est réalisée.
- ▷ Appliquez votre fonction pour trouver les maxima du signal moyenné et vérifiez le résultat en le traçant dans une figure.
  Que constatez-vous ? Comment peut-on facilement se débarrasser des maxima superflus ?
- ▷ Copiez votre fichier fonction `trouver_max_ligne.m` et renommé le `trouver_max.m`. Modifiez cette fonction afin d'inclure un seuil de détection des maxima. La valeur du seuil sera un nouvel argument d'entrée de la fonction.
- ▷ Appliquez votre nouvelle fonction au signal moyenné et vérifiez que seul les maxima d'intérêt pour mesurer l'interfrange sont détectés.

Mesure de l'interfrange et estimation de l'incertitude

Une fois que les positions des maxima d'intensité sont correctement repérés, déterminer l'interfrange est assez trivial.

- ▷ Créez un nouveau fichier fonction `calcul_interfrange.m`. Ecrivez le code de cette fonction de telle sorte à ce qu'elle prenne en arguments d'entrée la variable x et le tableau contenant les indices des maxima locaux et que ses variables de sortie soient un tableau contenant les différentes valeurs mesurées de l'interfrange, la valeur moyenne de l'interfrange, et sa variance σ^2, le tout en pixel.
- ▷ Appliquez votre fonction à vos données et vérifiez la cohérence du résultat. *Est-ce-que mon programme donne un résultat qui semble cohérent avec la figure d'interférence qu'il traite ?*

Détermination de la distance physique entre les deux fentes

La dernière étape du traitement numérique consiste à convertir votre mesure de l'interfrange (en pixel) en une valeur physique : la distance réelle (en m) entre les fentes qui ont données lieu à la figure d'interférence. C'est la que le cours de physique intervient ! La formule qui relie cette distance, l'interfrange et les autres paramètres expérimentaux est simple est doit vous rappeler quelque chose !

- ▷ Ecrivez les quelques lignes de codes permettant de calculer la distance physique entre les fentes à partir des paramètres expérimentaux et de votre mesure de l'interfrange. Pour estimer le plus simplement l'incertitude sur cette distance, on rappel que si l'on a une relation du type $a = b * c$, alors $\frac{\Delta a}{a} = \frac{\Delta b}{b} + \frac{\Delta c}{c}$. Ici, on va considérer que la seule source d'incertitude provient de la mesure de l'interfrange que l'on prendra égale à σ.

Finalisation du programme

Quand on écrit un programme de traitement de données, le travail n'est réellement fini que lorsque l'on dispose d'un code qui, lorsqu'il est lancé, effectue sans erreur les opérations voulues et exporte de manière claire et sans fioritures les résultats. C'est une étape qui peut sembler optionnelle mais qui relève des bonnes pratiques indispensables en programmation et qui évite bien souvent des déconvenues.

▷ Créez un nouveau dossier qui contiendra tout les codes de votre programme ainsi que le ou les images à traiter. Dans ce dossier, créez un fichier qui sera le cœur de votre programme de traitement (`TP4_traitement_interferences_Votrenom.m`) et copiez-y vos fonctions. Mettez au propre votre code dans le fichier pour qu'il réalise les opération suivantes :

1. Déclaration des paramètres du traitement : chemin d'accès à l'image, seuil de détection des maxima, paramètres expérimentaux ;
2. Importation de l'image et création des variables x et y ;
3. Moyennage du signal ;
4. Détection de la position des maxima ;
5. Calcul de l'interfrange et de l'incertitude ;
6. Conversion en unité physique : distance entre les fentes ;
7. Visualisation des résultats.

Pour la visualisation des résultats, il est commode d'écrire un fichier séparé (`resultat.m`) qui contient le code traçant toutes les figures. Pour exécuter ce code, il suffit d'entrer le nom du fichier dans le programme principal à l'endroit où l'on souhaite qu'il soit lu.

▷ Pour ce TP, écrivez le code du fichier `resultat.m` pour qu'une fois exécuté soient tracés dans la même fenêtre et dans trois figures séparées : l'image brute, le signal moyennée avec les maxima détectés, un graphique illustrant la valeur de chaque interfrange mesuré en pixel avec la moyenne et l'incertitude. Utilisez la commande `subplot()` pour organiser les figures dans la fenêtre.

▷ Ajoutez une ligne de code faisant apparaitre une fenêtre de dialogue contenant les valeurs numériques de l'interfrange en pixel, de la distance entre les fentes en mm et les incertitudes. Cela se fait très simplement à l'aide de la fonction `msgbox()`.

▷ Compilez votre programme principal et vérifiez que toutes les opérations soient exécutées sans erreur.