Introduction au traitement du signal avec Scilab

Les outils logiciels sont devenus un élément indispensable en traitement du signal, que ce soit pour l'apprentissage ou le développement de système, en particulier depuis l'apparition des techniques numériques.
Un signal numérisé peut être en effet vu comme une suite de valeurs, c'est à dire un vecteur. Un vecteur n'est qu'une matrice à une ligne ou une colonne.
Dans le cas de traitement d'images par exemple, les signaux étant à deux dimensions, on arrive directement à la notion de matrices. On pourrait trouver de nombreux exemples où le signal à traiter présente une dimension supérieure encore.

Il était donc naturel de se tourner vers des logiciels spécialisés en calcul matriciel, tel que Matlab (contraction de MAtrix LABoratory).
Nous utiliserons pour notre part un logiciel quasiment similaire, Scilab (pour SCIentific LABoratoty), qui présente l'avantage par rapport au précédent d'être libre et gratuit.

Scilab est développé par l'INRIA Rocquencourt (78). On peut le télécharger gratuitement avec sa documentation et de nombreuses applications, à l'adresse :

http://www-rocq.inria.fr/scilab/

Ce logiciel est utilisable pour toute application nécessitant un calcul matriciel, de la finance à la physique. Pour de nombreuses applications spécifiques telles que le traitement du signal, un ensemble de fonctions spécialisées, appelé « boîtes d'outils » (tools box) ont été mises au point.
Une interface graphique pour la simulation dynamique, nommé SICOS (pour SCIlab Connected Object Simulator), l'équivalent du Simulink de Matlab, est disponible avec Scilab.

Au cours de cette présentation, nous allons nous familiariser avec les fonctions de base de Scilab pour le traitement du signal.
Certaines fonctions de base de Scilab resteront cependant peu explicitées, on pourra alors se référer à la documentation générale (en anglais) fournie avec le logiciel. De nombreux documents (en français et anglais) sont disponibles sur Internet (taper scilab avec un moteur de recherche efficace, type Google par exemple), voir la bibliographie.

1. Installation

Scilab à été développé pour fonctionner sous Linux, mais fonctionne (presque) parfaitement avec windows. Nous utiliserons ici cette dernière option (même si ce n'est pas le choix le plus judicieux).

Une fois Scilab téléchargé (cas de la version 2.6) à l'adresse http://www-rocq.inria.fr/scilab/, le décompresser avec winzip, puis le lancer par un double clic sur le fichier « runscilab » dans le répertoire scilab_2.6/bin (tous ces détails sont donnés dans un fichier texte associé). On créera éventuellement un raccourci sur le bureau.

2. Règles pratiques

Avant toutes choses, il faut comprendre que Scilab est un langage interprété (comme le Basic) et non compilé (comme le C ou le Pascal), ce qui présente l'avantage de ne pas avoir à déclarer les variables, mais introduit une certaine lenteur (l'objectif n'est pas en général de faire du temps réel).

Les premiers programmes seront écris ligne par ligne directement dans l'éditeur de Scilab ; lorsqu'on a un peu l'habitude, il est cependant plus efficace d'utiliser un éditeur de texte annexe, comme le bloc

note de Windows ou WordPad, puis de faire ensuite des copier coller (attention le « ctrl V » ne fonctionne pas dans Scilab, on fera donc « clic droit » puis « paste »). On évitera d'utiliser Word, à moins de ne le configurer correctement (voir annexe 3) qui prend souvent des initiatives malheureuses (ajout de majuscules, de tabulation etc…).

Passons maintenant en revue quelques règles pratiques :
- pour chaque fonction, une aide est disponible en tapant « help Nom_de_la_fonction » à l'invite.
- pour une fonction dont on ne connaît pas le nom exact, on peut obtenir tous les noms réservés liés à la fonction en tapant « apropos Nom_de_la fonction » , par exemple « apropos filter ».
- de même la documentation complète en ligne est fournie avec le logiciel
- en cas d'erreur, il est inutile de réécrire les instructions, les flèches du clavier permettant de se déplacer dans le jeu des instructions précédemment entrées ; les touches de raccourci « Ctrl C » et « Ctrl V » pour le copier coller ne fonctionnent pas sous Scilab ; utiliser le menu contextuel accessible par un clic droit dans la fenêtre.
- Scilab fait la différence entre majuscules et minuscules.
- bien que des boucles (for, while etc) soit utilisable, Scilab est optimisé pour faire du calcul matriciel ; on préférera donc un produit matriciel à une boucle, afin de réduire le temps de calcul et l'utilisation des ressources.
- les signaux et variables créés sous Scilab peuvent être sauvegardés avec la commande « save » et rechargées avec « load » (voir les annexes pour la syntaxe exacte).
- il est fortement conseillé de définir en début de programme toutes les constantes de manière littéral, afin de pouvoir modifier simplement l'une d'elle par la suite, si nécessaire.

A la fin de ce document, on trouvera des annexes détaillant des points particuliers (première approche, affichage, sauvegarde etc…). On s'y réfèrera pour obtenir plus de détails.

3. Représentation d'un signal sinusoïdal dans les domaines temporel et fréquentiel.

Pour un système numérique, tel qu'un ordinateur, il ne peut y avoir que des signaux discrets. On pourra cependant voir un signal discret comme un signal continu si le pas d'échantillonnage (ou pas de calcul) est faible devant les autres grandeurs temporelles.

La première étape consiste donc à définir ce pas de calcul, ainsi que la durée de l'affichage (ou durée de calcul). On définit ainsi un vecteur temps, dont le nombre d'élément est égal au nombre de points à afficher, et dont l'incrément, d'un élément au suivant, est égal au pas de calcul. Il s'agit d'un vecteur ligne.

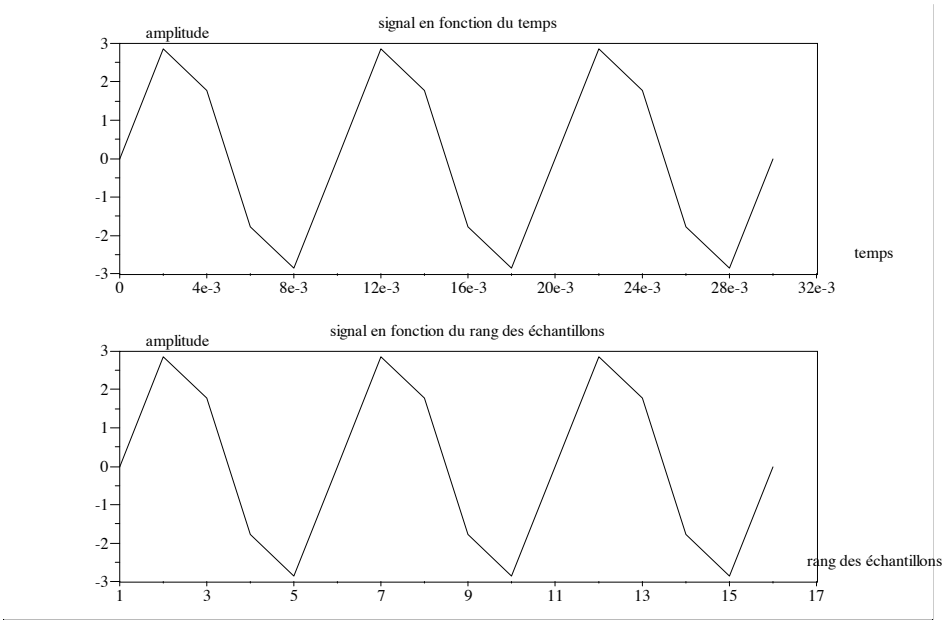
Dans l'exemple suivant, on affiche sur 16 points, par pas de 2ms, une sinusoïde d'amplitude 3 et de fréquence 100 Hz.
Entrer ces commandes (sans les commentaires).

```
// les commentaires sont précédés de deux barres
// on peut utiliser indifféremment majuscules et minuscule, mais « s » n'est pas « S »
// la séparation de la partie entière d'un nombre et des décimales se fait par « . » et non « , »
//
clear
nb_pts=16
// si on ne souhaite pas voir le résultat s'afficher, placer un « ; » à la fin de l'instruction
pas=2e-3
//
t=pas*(0 :1 :nb_pts-1)
// t est un vecteur ligne allant de 0 à nb_pts par pas de 1, multiplié par « pas »
// l'incrément étant par défaut unitaire, il peut être omis ici
//
```

```
//
amp=3 ; f=100 ;
s=amp*sin(2*%pi*f*t)
// les constantes prédéfinies (pi, j, e etc) sont précédées de « % »
//s est un vecteur ligne, de même nombre de points que t
//
// initialisation des paramètres d'affichage
xbasc() ; xset("font size", 4);
// affichage
plot2d(t,s) // affichage de s en fonction du temps
// commentaires divers
xtitle ("signal en fonction du temps","temps", "amplitude");
```

Le programme précédent nous permet d'afficher la sinusoïde en fonction du temps (voir premier graphe ci-après).
Dans la même fenêtre, on peut également afficher la sinusoïde en fonction du rang de l'échantillon, en modifiant la partie affichage du programme comme suit (pour les détails sur le fonctionnement de « xsetech( ) » voir l'annexe sur l'affichage) :

```
// initialisation de l'affichage
xbasc(); xset("font size",4);
// gestion du premier graphe dans la fenêtre, position (0,0), largeur 1, hauteur 1/2
xsetech([0, 0, 1,1/2]);
// affichage
plot2d(t,s)
// commentaires divers
xtitle ("signal en fonction du temps","temps", "amplitude");
//
// gestion du second graphe dans la fenêtre, position (0,1/2), largeur 1, hauteur 1/2
xsetech([0, 1/2, 1,1/2]);
// affichage
plot2d(s)
 // commentaires divers
xtitle ("signal en fonction du rang des échantillons","rang des échantillons", "amplitude");
```

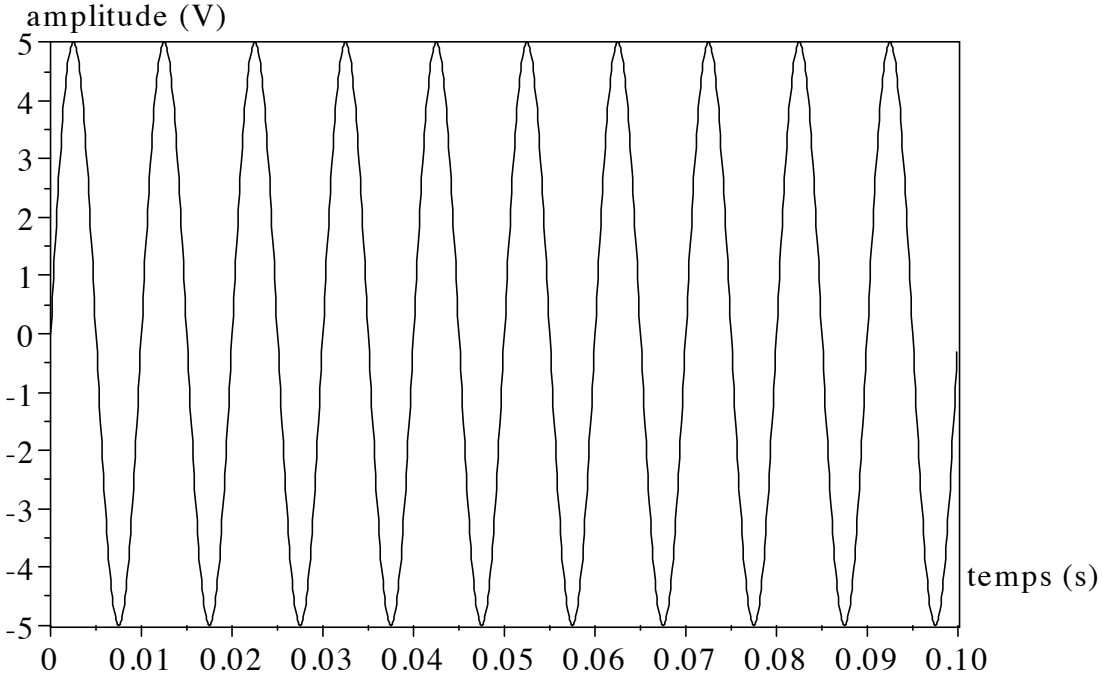


On peut noter à la vue de ces courbes, qu'un vecteur peut être représenté en fonction d'un autre (« s » en fonction de « t »), ou bien en fonction du rang de ses éléments.
Dans ce dernier cas, il est important de noter que **l'indice du premier élément est toujours « 1 » et non « 0 »**.

L'échantillonnage grossier de cette sinusoïde, nous a permis de voir s'afficher les valeurs des vecteurs t et s sans voir défiler une suite importante de nombre.

Affichons maintenant une sinusoïde, d'amplitude 5V crête, de fréquence 100 Hz, sur 1000 points, avec une période d'échantillonnage de 0,1 ms (soit 100 points par période et une fréquence fe d'échantillonnage de 10 kHz).

```
clear
//
// définition des constantes, nombre de points, période d'échantillonnage et fréquence du signal
N=1000 ; Te=0.1e-3 ; F=100 ;
//
// description du vecteur temps et vecteur signal
t=Te*(0:N-1);s=5*sin(2*%pi*F*t);
//
// affichage
xbasc() ; xset("font size",5);
plot2d(t,s); xtitle("","temps (s)","amplitude (V)")
```



Nous allons maintenant afficher la transformée de Fourier de ce signal. Scilab effectue non pas le calcul de la transformé de Fourier (calcul théorique analogique), mais, comme tout système numérique, la transformée de Fourier discrète. Une étude détaillée de cette dernière sera l'objet d'une prochaine séance. Pour l'instant, contentons nous de quelques informations :

- la transformation de Fourier discrète d'un signal « s » de N échantillons, se fait par la fonction « fft(s,-1) » (où « –1 » représente le signe de l'exposant de l'exponentielle dans l'expression théorique, la transformée inverse se faisant donc avec « 1 » comme paramètre),

- cette fonction renvoie un vecteur de même dimension N que « s »,
- les éléments de ce vecteur sont des nombres complexes, on pourra alors éventuellement utiliser la fonction « abs » pour n'en garder que le module,
- le premier élément (indice 1) correspond à la fréquence nulle,
- le dernier (indice N) à la fréquence d'échantillonnage fe diminuée d'un pas fe/N,
- une telle transformation est périodique de période fe (le signal temporel étant échantillonné).

Dans les cas relativement simples que nous allons étudier (signaux périodiques), pour passer de la transformation de Fourier discrète à la transformation de Fourier (au sens analogique), il faudra tenir compte des remarques suivantes :

- les signaux que nous étudions pour l'instant étant sensés être des signaux à temps continu, le spectre n'a de sens que de –fe/2 à fe/2,
- les signaux étudiés étant réels, on pourra limiter l'étude du module (qui est pair) de la transformée de 0à fe/2 ?
- si on souhaite également étudier la partie imaginaire (impaire pour un signal réel) de la transformée, il faudra considérer la bande de fe/2 à fe, qui correspond à la bande de –fe/2 à 0 du spectre théorique,
- pour se ramener aux amplitudes attendues pour un signal à temps continu, il faut diviser le résultat donné par la fonction « fft » par le nombre d'échantillons N.

A la suite du programme précédent, on peut donc écrire pour obtenir la transformée de Fourier :

```
// définition d'une variable fréquence pour l'affichage
// le premier point correspond au continu
// le dernier à la fréquence d'échantillonnage au pas près
f=1/(N*Te)*( 0 : N-1) ;
//
// calcul de la transformation
sf=fft(s,-1) ;
//
// initialisation de l'affichage
xbasc(); xset("font size",4);
//
// affichage du module brut de forme
xsetech([0,0,1,1/3]) ; plot2d(abs(sf)) ;
xtitle("module brut de la fft en fonction des indices","rang des indices","amplitude" );
//
// affichage du module corrigé de 0 à fe/2
xsetech([0,1/3,1,1/3]) ; plot2d(f(1:N/2), 1/N*abs(sf(1:N/2))) ;
xtitle("module corrigé en fonction de la fréquence de 0 à fe/2","fréquence (Hz)","amplitude (Vs)");
//
// zoom sur la raie, de la fréquence nulle à fe/5
xsetech([0,2/3,1,1/3]) ; plot2d(f(1:N/20),1/N*abs(sf(1: N/20))) ;
xtitle("module corrigé et zoomé","fréquence (Hz)","amplitude (Vs)");
```

Pour définir la variable fréquentielle « f » qui n'est indispensable que si on souhaite soigner l'affichage, il ne faut pas perdre de vue que la valeur f(0) (d'indice « 1 ») correspondra au continu, et la valeur f(N-1) (d'indice N) correspondra à fe-fe/N. L'expression de « f » vient alors d'elle-même.

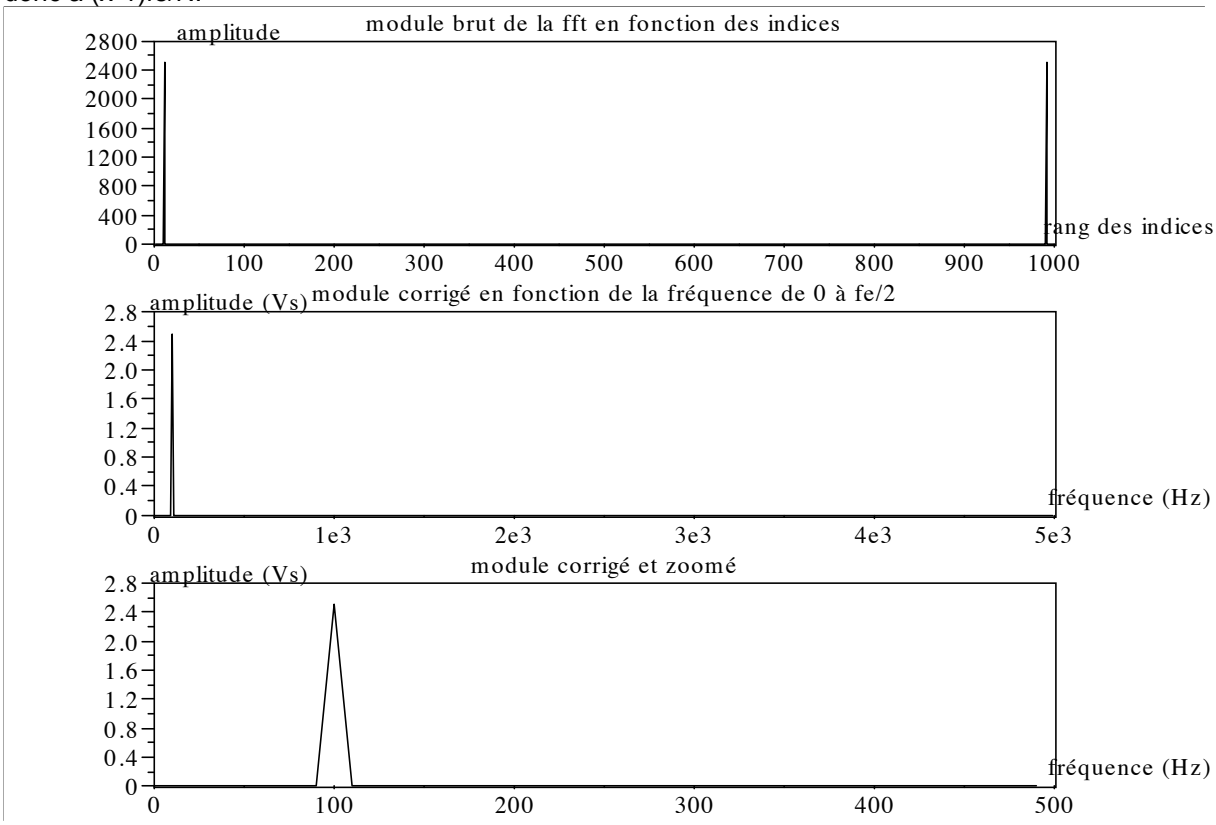
Le premier affichage met en évidence la périodicité du spectre, ainsi que le problème des amplitudes.

Avec le second affichage, on corrige le problème d'amplitude en divisant par N le module. L'amplitude est alors celle attendue, 2,5 Vs pour le spectre bilatéral d'un signal sinusoïdal de 5 V crête.
On limite la bande de fréquence de 0 à fe/2 en ne prenant que les éléments d'indices 1 à N/2 des deux vecteurs « f » et « sf » (voir annexe sur l'affichage).

Avec le troisième affichage, par la même méthode, on effectue un zoom sur les 20 premiers échantillons (soit du continu à fe/5). On peut remarquer que la raie s'affiche comme un triangle, ce qui

est du au choix du mode d'affichage. Pour obtenir une raie seule, on pouvait choisir la fonction « plot2d3 » (voir annexe sur l'affichage).

On peut noter que la raie se trouve sur l'élément de rang 11 de « sf » en tapant :
abs(sf(10))
ce qui renvoie la valeur 2500 (ce qui donne 2,5 une fois divisé par N) ; cette valeur correspond bien à 100 Hz puisque l'indice 1 correspond à la fréquence 0 et l'indice N à fe-fe/N ; l'indice x correspond donc à (x-1)fe/N.



Remarque importante : l'amplitude est exactement celle prévue. Ceci est dû au choix d'une fréquence à afficher multiple de fe/N. Ces différents problèmes seront étudiés ultérieurement.

Quelques améliorations concernant la présentation sont proposées en annexe 4.

Rappeler les spectres (parties réelle et imaginaire) des signaux suivant :
s1=sinωt s2=cosωt s3=sin(ωt+p/4) s4=e$^{j\omega t}$
Nous allons adopter maintenant une démarche inverse de la précédente : retrouver le signal temporel à partir de sa transformée de Fourier.
En gardant les mêmes nombre de points, fréquence d'échantillonnage et fréquence du signal, générer dans Scilab les spectres de ces différents signaux. On placera pour cela bout à bout des « 1 » et des « 0 » aux bons endroits en s'aidant de l'instruction « zeros(x,y) » qui génère une matrice de « 0 » de x lignes et y colonnes.
Attention au signal s4 qui n'est pas réel et ne vérifie donc pas les propriétés de symétrie paire du module et impaire de la phase.
Vérifier par affichage avec les instructions correspondantes la forme des modules –abs(S)-, partie réelle –real(S)- et partie imaginaire –imag(S)-.
Faire la transformation inverse et afficher le signal temporel pour en vérifier la forme. On notera en particulier pour le signal exponentiel que les parties réelle et imaginaire sont sinusoïdales, tandis que le module est constant.

Remarque : pour un signal complexe, sans autre précision, les fonctions d'affichages affichent la partie réelle.

4. Représentation d'un signal périodique quelconque

En vous inspirant du programme ci-après, écrire un programme permettant de visualiser un signal carré, puis un signal triangulaire sur quelques périodes. Effectuer l'analyse fréquentielle et vérifier qu'elle est bien conforme aux résultats attendus.

Pour générer le signal carré, on décrit d'abord un symbole +/-5 V par exemple, à l'aide de l'instruction « ones(x, y) » qui produit une matrice de « 1 » sur x lignes et y colonnes. On reproduit ensuite Nsymb fois ce symbole en multipliant sa forme transposée (matrice colonne) par une matrice de « 1 » de une ligne et Nsymb colonnes. Le résultat, une matrice de Nech_symb colonnes sur Nsymb lignes, est remis sous forme de vecteur ligne par l'instruction « matrix » (voir annexe).

```
clear
// définition des constantes :  nombre d'échantillons par symbole, nombre de périodes
// nombre total de points, fréquence d'échantillonnage
Nech_symb=10 ; Nsymb=12 ; N=Nech_symb*Nsymb ; Fe=1e3 ;
//
// variable temps
t=1/Fe*[0:N-1];
//
// écriture d'un symbole +/- 5
symb=5*[ ones(1, Nech_symb/2), -1*ones(1, Nech_symb/2)];
//
// répétion du symbole pour constituer le signal
S=matrix(symb' *ones(1, Nsymb), 1 , N);
//
// affichage
xbasc(); xset ("font size", 4);
plot2d2(t,S, rect=[0, -6, .12, 6]) , xtitle("signal carré", "temps (s)", "amplitude");
```

5. Modulation d'amplitude

Nous avons vu au chapitre précédent la manière de décrire une sinusoïde et d'obtenir son spectre. Appliquons maintenant ces notions sur des signaux un peu plus complexes.

Modulation sans porteuse

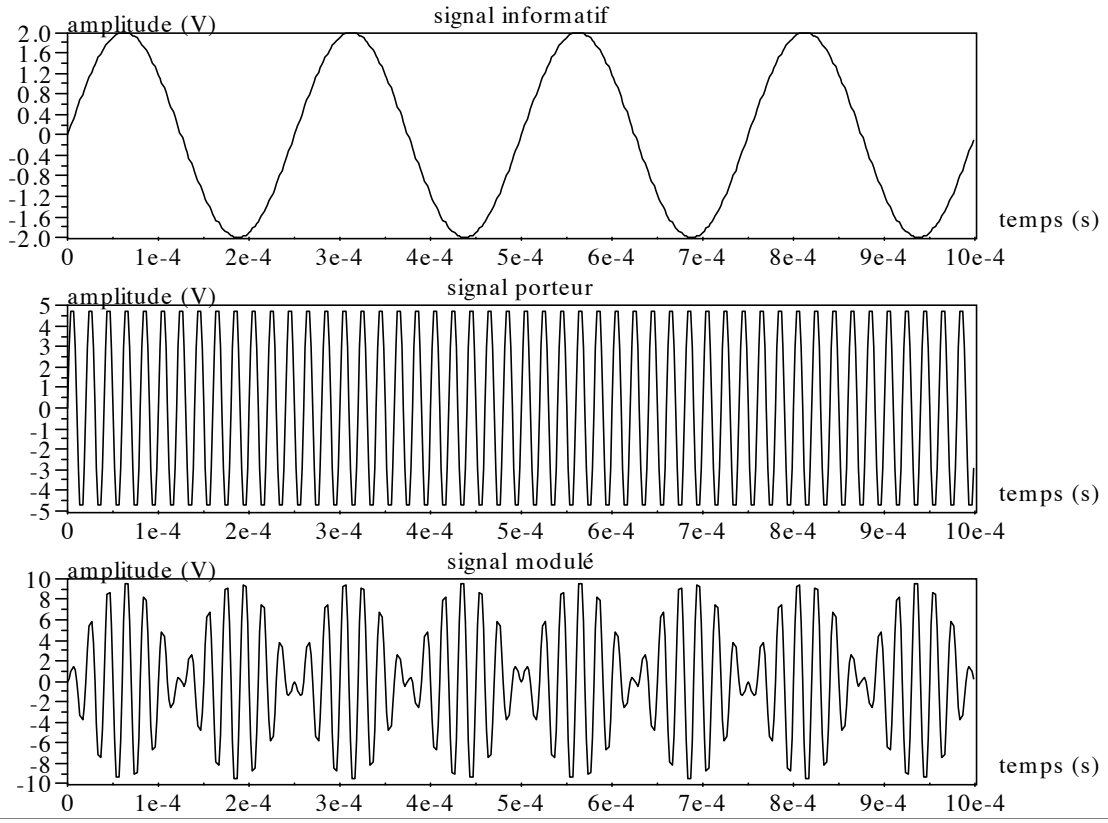
On souhaite moduler en amplitude avec porteuse supprimée, un signal de 5 V crête 50 kHz, par un signal informatif de 2 V crête 4 kHz.
Rappeler le spectre du signal obtenu modulé.

On se propose d'écrire un programme avec des calculs sur 500 points pour une fréquence d'échantillonnage de 500 kHz.
Pour réaliser l'opération de modulation, on utilise en électronique un multiplieur qui effectue à chaque instant le produit des deux signaux à ses entrées.
Pour réaliser la même chose avec Scilab, où on ne dispose pas de deux signaux, mais de deux vecteurs ligne pour les signaux informatif et porteur, il faudra générer un vecteur ligne correspondant au signal modulé, dont chaque élément est le produit des deux éléments de même position sur les vecteurs de signaux informatif et porteur. Il ne s'agit donc pas d'un produit matriciel classique, mais d'un produit « terme à terme », symbolisé par « .* » dans le logiciel.

Le programme ci-après permet l'affichage de l'évolution temporelle et spectrale du signal modulé.

```
clear
// définition des constantes
N=500 ; fe=500e3 ; fp=50e3 ; finf=4e3;
```

```
,,
// description des vecteurs temps, fréquence (pour l'affichage de la fft) et signal
t=(0:N-1)/fe ; f=(0:N-1)/N*fe ;
sinf=2*sin(2*%pi*finf*t); spor=5*sin(2*%pi*fp*t); smod=spor.*sinf;
//
// initialisation de l'affichage
xbasc() ; xset( "font size", 4);
//
// affichage du signal informatif
xsetech([0,0,1,1/3]) ; plot2d(t,sinf) ; xtitle("signal informatif","temps (s)","amplitude (V)");
//
// affichage du signal porteur
xsetech([0,1/3,1,1/3]) ; plot2d(t,spor) ;
xtitle("signal porteur","temps (s)","amplitude (V)");
//
// affichage du signal modulé
xsetech([0,2/3,1,1/3]) ; plot2d(t, smod) ;
xtitle("signal modulé","temps (s)", "amplitude (V)");
```

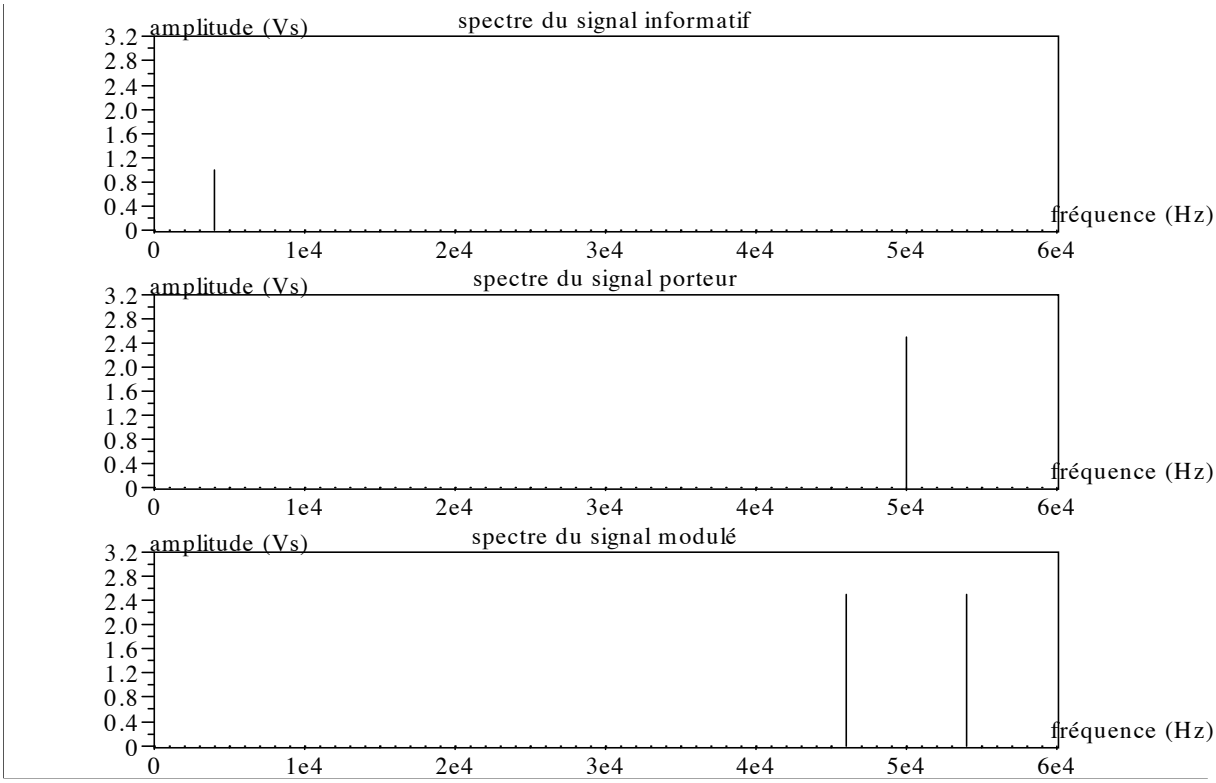


Le spectre des signaux peut être obtenu par le programme suivant :

```
// calcul des modules des fft
Sinf=1/N*abs(fft(sinf,-1)) ; Spor=1/N*abs(fft(spor,-1)) ; Smod=1/N*abs(fft(smod,-1)) ;
//
// initialisation de l'affichage
xbasc() ; xset( "font size", 4);
//
//
```

```
,,
// affichage du signal informatif
xsetech([0,0,1,1/3]) ; plot2d3(f(1: .12*N),Sinf(1: .12*N), rect=[0, 0, 6e4, 3]) ;
xtitle("spectre du signal informatif","fréquence (Hz)","amplitude (Vs)");
//
//
//
// affichage du signal porteur
xsetech([0,1/3,1,1/3])  ; plot2d3(f(1: .12*N),Spor(1: .12*N), rect=[0, 0, 6e4, 3])  ;
xtitle("spectre du signal porteur","fréquence (Hz)","amplitude (Vs)");
//
// affichage du signal modulé
xsetech([0,2/3,1,1/3]) ; plot2d3(f(1: .12*N),Smod(1: .12*N), rect=[0, 0, 6e4, 3])  ;
xtitle("spectre du signal modulé","fréquence (Hz)","amplitude (Vs)");
```

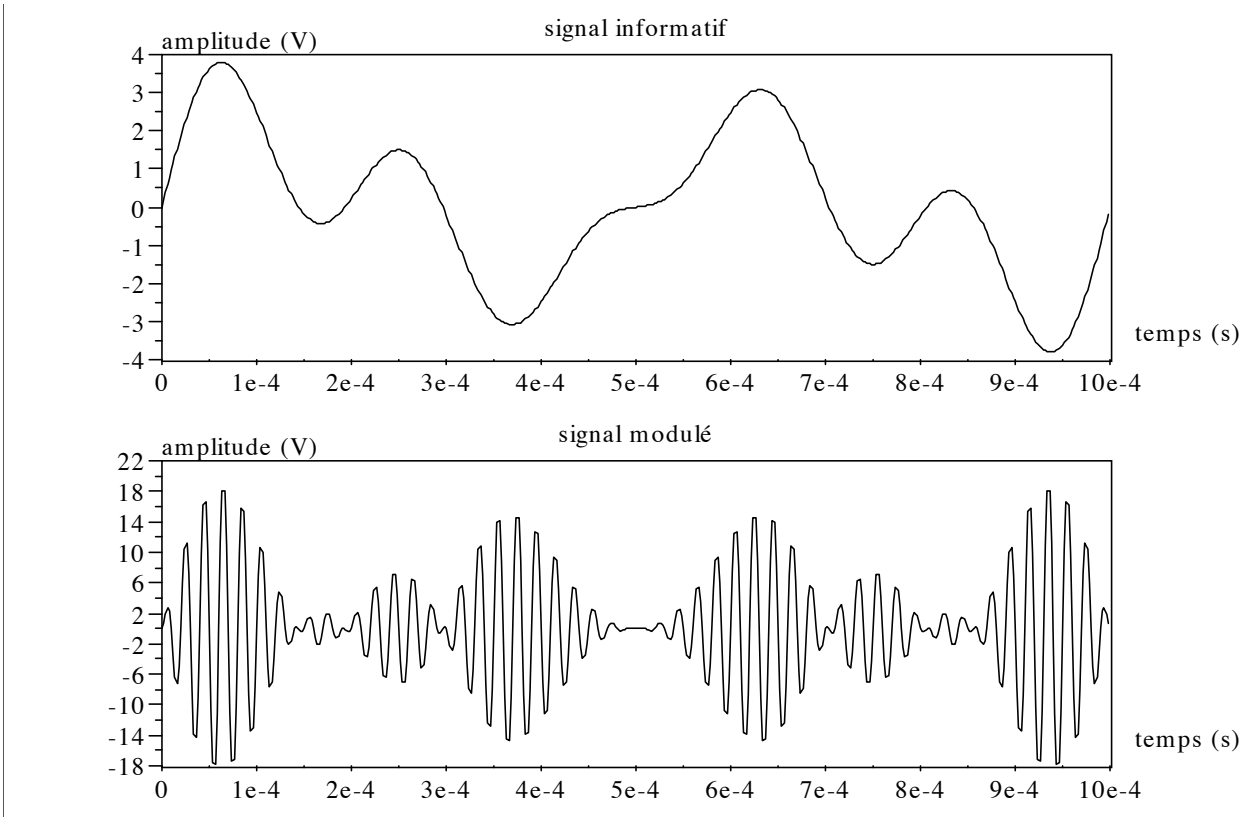
L'affichage a été réalisé avec la fonction « plot2d3 » qui permet d'afficher des raies, tandis que l'instruction « rect » permet d'avoir les mêmes échelles d'amplitude pour les trois signaux.



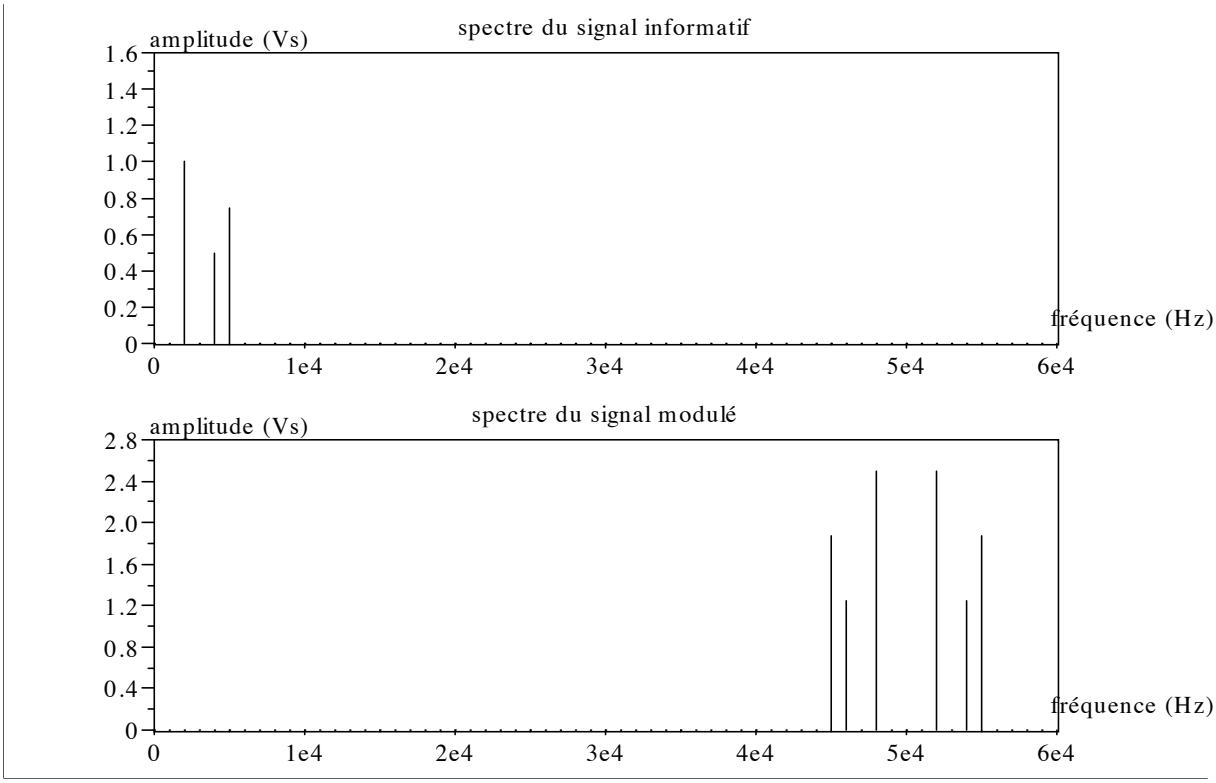
Remplacer maintenant le signal informatif par une somme de 3 sinusoïdes d'amplitudes et fréquences respectives (1V, 4kHz), (2V, 2kHz), et (1,5V, 5kHz). Visualiser les signaux informatifs et modulés, ainsi que le spectre des signaux modulés

```
clear ;
// définition des constantes
N=500 ; fe=500e3 ; fp=50e3 ; finf1=4e3; finf2=2e3; finf3=5e3;
//
// description des vecteurs temps, fréquence (pour l'affichage de la fft) et signal
t=(0:N-1)/fe ; f=(0:N-1)/N*fe ;
//
//
```

```
,,
sinf=2*sin(2*%pi*finf2*t)+1*sin(2*%pi*finf1*t)+1.5*sin(2*%pi*finf3*t);
spor=5*sin(2*%pi*50e3*t);
smod=spor.*sinf;
//
// initialisation de l'affichage
xbasc() ; xset( "font size", 4);
//
// affichage du signal informatif
xsetech([0,0,1,1/2]) ; plot2d(t,sinf) ; xtitle("signal informatif","temps (s)","amplitude (V)");
//
// affichage du signal modulé
xsetech([0,1/2,1,1/2]) ; plot2d(t, smod) ;
xtitle("signal modulé","temps (s)", "amplitude (V)");
```




```
// calcul des modules des fft
Smod=1/N*abs(fft(smod,-1)); Sinf=1/N*abs(fft(sinf,-1));
//
// initialisation de l'affichage
xbasc() ; xset( "font size", 4);
// affichage du signal informatif
xsetech([0,0,1,1/2]) ; plot2d3(f(1: .12*N),Sinf(1: .12*N), rect=[0, 0, 6e4, 1.5]) ;
xtitle("spectre du signal informatif","fréquence (Hz)","amplitude (Vs)");
//
// affichage du signal modulé
xsetech([0,1/2,1,1/2]) ; plot2d3(f(1: .12*N),Smod(1: .12*N))  ;
xtitle("spectre du signal modulé","fréquence (Hz)","amplitude (Vs)");
```



Afin de se familiariser avec les produits matriciels, on étudiera le programme ci-après, permettant de générer la somme des trois sinusoïdes pour le signal informatif.

```
clear
// définition des constantes
N=500 ; fe=500e3 ; fp=50e3 ; finf1=4e3; finf2=10e3; finf3=15e3;
//
// description du vecteur temps
t=(0:N-1)/fe ;
//
// les fréquences en vecteur colonne
frequence=[4e3; 10e3; 15e3];
//
// les amplitudes en vecteur ligne
amplitude=[1, 2, 1.5] ;
//
// calcul du signal modulé
sinf_2=amplitude*sin(2*%pi*frequence*t);
//
//affichage
xbasc(); plot2d(t,sinf)
```

Modulation d'amplitude avec porteuse

Proposer un programme permettant d'illustrer les propriétés temporelles et spectrales de la modulation d'amplitude avec porteuse. L'indice de modulation devra être facilement modifiable. Pour générer un signal continu, on pourra utiliser la fonction « ones() » (voir annexe 1).

Modulation en bande latérale unique

Pour obtenir notre signal modulé en BLU, nous allons utiliser dans un premier temps la méthode classique qui consiste à supprimer par filtrage, une des bandes du signal modulé en double bande. Cette méthode est en générale très difficile à mettre en œuvre dans un cas réel (pente du filtre nécessaire trop raide), mais notre but est ici simplement d'illustrer certaines fonctionnalités de Scilab.
Dans une seconde approche, nous nous contenterons d'afficher le signal à partir des équations données par la théorie.

Le programme suivant, à partir d'un signal informatif composé de trois sinusoïdes, va dans un premier temps calculer le signal double bande smod comme nous l'avons fait jusqu'à présent. On détermine ensuite la fonction de transfert du filtre à l'aide de l'instruction « analpf » qui calcule un filtre passe bas analogique en fonction des arguments fournis, à savoir l'ordre (ici 8), le type de filtre (ici un Chebycheff), l'ondulation dans la bande passante (ici 0,1, le second paramètre étant sans importance pour le type de filtre choisi), ainsi que la fréquence de coupure. Attention, l'ondulation n'est pas exprimée en dB (0.1 correspond à 0.9 dB d'ondulation).
Cette instruction nous renvoie en particulier (avec les notations du programme), la fonction de transfert G en variable de Laplace, le numérateur G(2) de cette fonction de transfert, et le dénominateur G(3).

A l'aide de la fonction « freq » on peut alors obtenir la valeur complexe pour tous les points désirés (dans le domaine fréquentiel), en précisant numérateur, dénominateur, ainsi que la variable (ici le vecteur f multiplié par $2\pi j$) pour laquelle le calcul doit être fait.

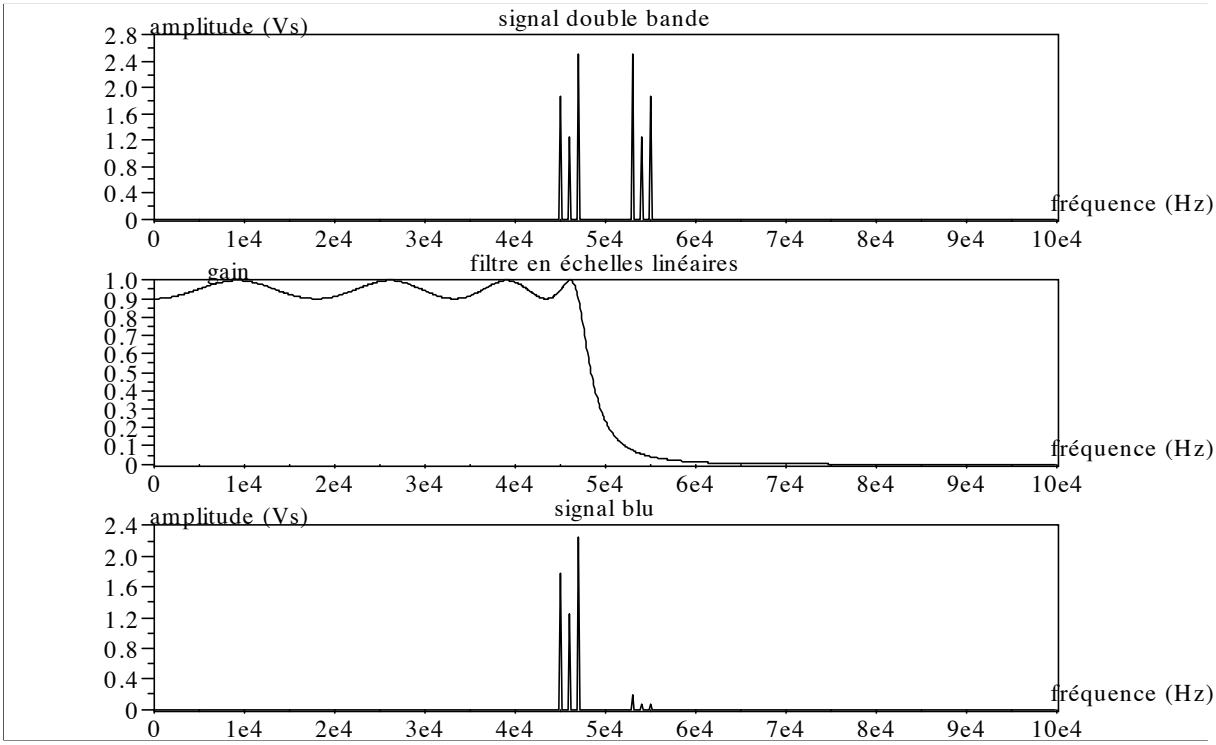
On trouvera plus d'information sur la manipulation des fonctions de transfert en annexe 4.

Pour effectuer le filtrage, on calcule la transformée de Fourier (opération « fft(x,-1) ») du signal double bande, qu'on multiplie par le gain complexe du filtre. Nous affichons alors les signaux dans l'espace fréquentiel en zoomant de 0 à fe/5 le signal et en effectuant la correction du nombre d'échantillons.
Le retour dans le domaine temporel, qui présente une petite subtilité, sera vu par la suite.

```
clear
// définition des constantes
N=5000 ; fe=500e3 ; fp=50e3 ; finf1=4e3; finf2=3e3; finf3=5e3;
//
// description des vecteurs temps, fréquence (pour l'affichage de la fft) et signal
t=(0:N-1)/fe ; f=(0:N-1)/N*fe ;
//
// définition des différents signaux informatif, porteur et double bande
sinf=2*sin(2*%pi*finf2*t)+1*sin(2*%pi*finf1*t)+1.5*sin(2*%pi*finf3*t);
spor=5*sin(2*%pi*50e3*t);
smod=spor.*sinf;
//
// calcul de la fonction de transfert du filtre et des valeurs de cette fonction sur la gamme de fréquence
G=analpf(8,'cheb1',[.1 0],2*%pi*(fp-finf2));
gain=freq( G(2), G(3), %i*2*%pi*f);
//
// filtrage par multiplication dans le domaine fréquentiel
Smod=fft(smod,-1);
Sblu=Smod.*gain;
//
//affichage dans le domaine spectral
xbasc() ; xset( "font size", 4);
//
xsetech([0,0,1,1/3]) ; plot2d(f(1:N/5),1/N*abs(Smod(1:N/5))) ;
xtitle("signal double bande","fréquence (Hz)","amplitude (Vs)");
//
xsetech([0,1/3,1,1/3]) ; plot2d(f(1:N/5), abs(gain(1:N/5))) ;
xtitle("filtre en échelles linéaires","fréquence (Hz)","gain");
//
```

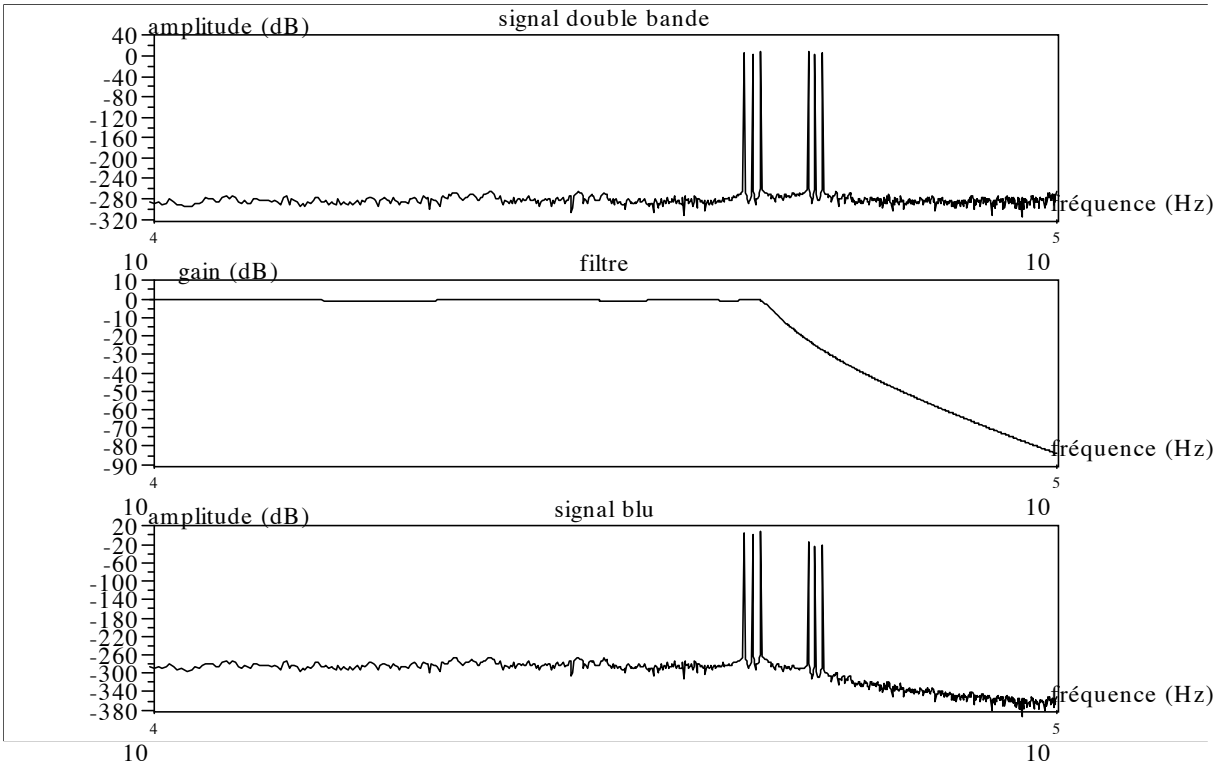
```
,,
xsetech([0,2/3,1,1/3]) ; plot2d(f(1:N/5), 1/N*abs(Sblu(1:N/5))) ;
xtitle("signal blu","fréquence (Hz)","amplitude (Vs)");
```

L'affichage est d'abord effectué dans le domaine spectral.



On peut afficher la même chose, mais avec les échelles semi-logarithmiques classiques. On notera l'utilisation de l'instruction « logflag » (voir annexe sur l'affichage ou l'aide en ligne sur « plot2d ») et la suppression de l'indice 1 du vecteur fréquence, indice qui valait 0 et dont le logarithme est indéfini. Afin de simplifier l'écriture des instructions d'affichage, tous les paramètres ont été décrits avant.

```
// définition des paramètres d'affichage de 10 kHz à 100 kHz
Nmin=N/50 ; Nmax=N/5; f_aff=f(Nmin:Nmax);
//
Smod_dB=20*log10(1/N*abs(Smod(Nmin:Nmax)));
gain_dB=20*log10( abs(gain(Nmin:Nmax)));
Sblu_dB= 20*log10(1/N*abs(Sblu(Nmin:Nmax)));
//
// affichage en dB
xbasc(); xset( "font size", 4);
//
xsetech([0,0,1,1/3]) ; plot2d(f_aff ,Smod_dB, logflag=["ln"] , rect=[1e4, -320, 1e5, 20]   ) ;
xtitle("signal double bande","fréquence (Hz)","amplitude (dB)");
//
xsetech([0,1/3,1,1/3]) ; plot2d(f_aff, gain_dB , logflag=["ln"] , rect=[1e4, -90, 1e5, 10]   ) ;
xtitle("filtre","fréquence (Hz)","gain (dB)");
//
xsetech([0,2/3,1,1/3]) ; plot2d(f_aff, Sblu_dB, logflag=["ln"] , rect=[1e4, -380, 1e5, 20]   ) ;
xtitle("signal blu","fréquence (Hz)","amplitude (dB)");
```

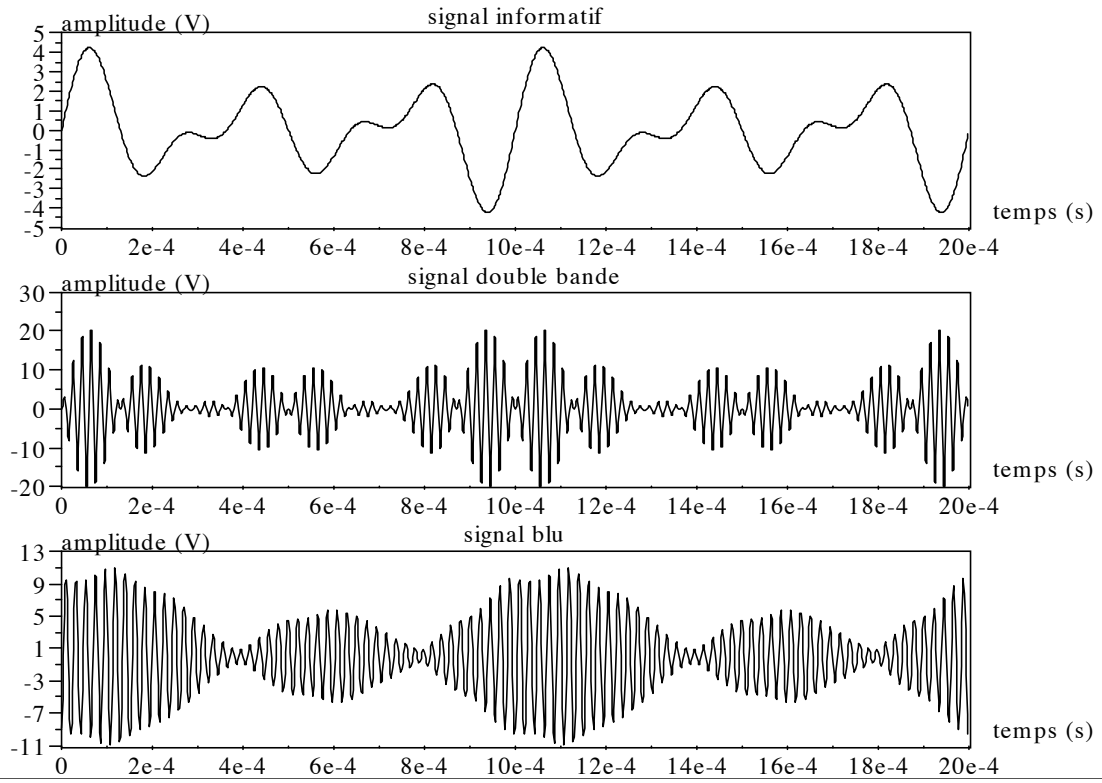


Afin de voir l'atténuation subie par la bande supérieure du signal BLU, on pourra utiliser le « 2DZoom » de la fenêtre d'affichage. On note alors une atténuation d'environ 25 dB de cette bande. Les valeurs exactes peuvent être obtenues en tapant 20*log10(abs((Sblu(x)/Sblu(y))) avec le couple (x,y) valant respectivement (d'après ce que nous avons vu sur l'affichage de la fonction « fft ») les valeurs (531, 471), (541, 461), (551, 451) correspondants aux différentes raies ; les atténuation sont alors de 21,5 dB, 25,1 dB et 27,2 dB.

Le retour dans le domaine temporel de « Sblu » se fait avec l'opération « sblu=fft(Sblu,1) ». Le signal fréquentiel doit cependant subir auparavant une modification.
Nous avons vu en effet que la fonction transformation de Fourier discrète dans Scilab (et en général) donnait un signal périodique, de période fe, qui était représenté sur l'interval 0 – fe par le logiciel.
On sait d'autre part (revoir l'exercice sur la transformée des signaux sinusoïdaux), que pour des signaux temporels réels, le module et la partie réelle de la transformée sont pairs, tandis que la phase et la partie imaginaire sont impaires.
Cette propriété introduit comme nous l'avons vu une symétrie par rapport à fe/2.
En conséquence, le filtrage effectué pour Sbu n'a de sens que de 0 à fe/2, le gain du filtre n'ayant pas été définit en respectant les symétries précédentes, ce qui n'était pas gênant pour ce que nous avions à afficher dans le domaine spectral.
Par contre pour calculer le signal « sbu » dans le domaine temporel, nous devons reconstituer la symétrie de « Sblu » en ne gardant dans un premier temps que la moitié des échantillons. Cette partie du spectre de 0 à (fe/2-fe/N) correspond aux échantillons de 0 à N/2.
Elle sera juxtaposée à cette même moitié dont l'ordre des éléments sera inversé (retournement du spectre pour obtenir la symétrie), tous les éléments étant conjugués (partie imaginaire impaire). On obtient ainsi la partie fe/2 – (fe-fe/N) correspondant aux éléments de (N/2 + 1) à 2 (l'indice 1 correspond au continu d'un côté et à fe de l'autre).
Cette opération se fait par « conj(Sblu(N/2+1 :-1 :2).
On calcule ensuite la transformée de Fourier inverse.

On peut maintenant afficher dans le domaine temporel, en réduisant le nombre de point afin de faire un effet zoom, ce nombre de point ayant été choisi important de manière à obtenir une meilleure résolution fréquentielle (ce détail sera abordé ultérieurement).

```
//
// retour dans l'espace temporel
Sblu=Sblu(1:N/2+1);
Sblu=[Sblu(1 :N/2), conj(Sblu(N/2+1 : -1 : 2))];
sblu=fft(Sblu,1);
//
// affichage dans le domaine temporel
xbasc() ; xset( "font size", 4);
xsetech([0,0,1,1/3]) ; plot2d(t (1:N/5) ,sinf(1:N/5) ) ;
xtitle("signal informatif","temps (s)","amplitude (V)");
//
xsetech([0,1/3,1,1/3]) ; plot2d(t(1:N/5) ,smod(1:N/5) ) ;
xtitle("signal double bande","temps (s)", "amplitude (V)");
//
xsetech([0,2/3,1,1/3]) ; plot2d(t(1:N/5) ,sblu(1:N/5) ) ;
xtitle("signal blu","temps (s)", "amplitude (V)");
```



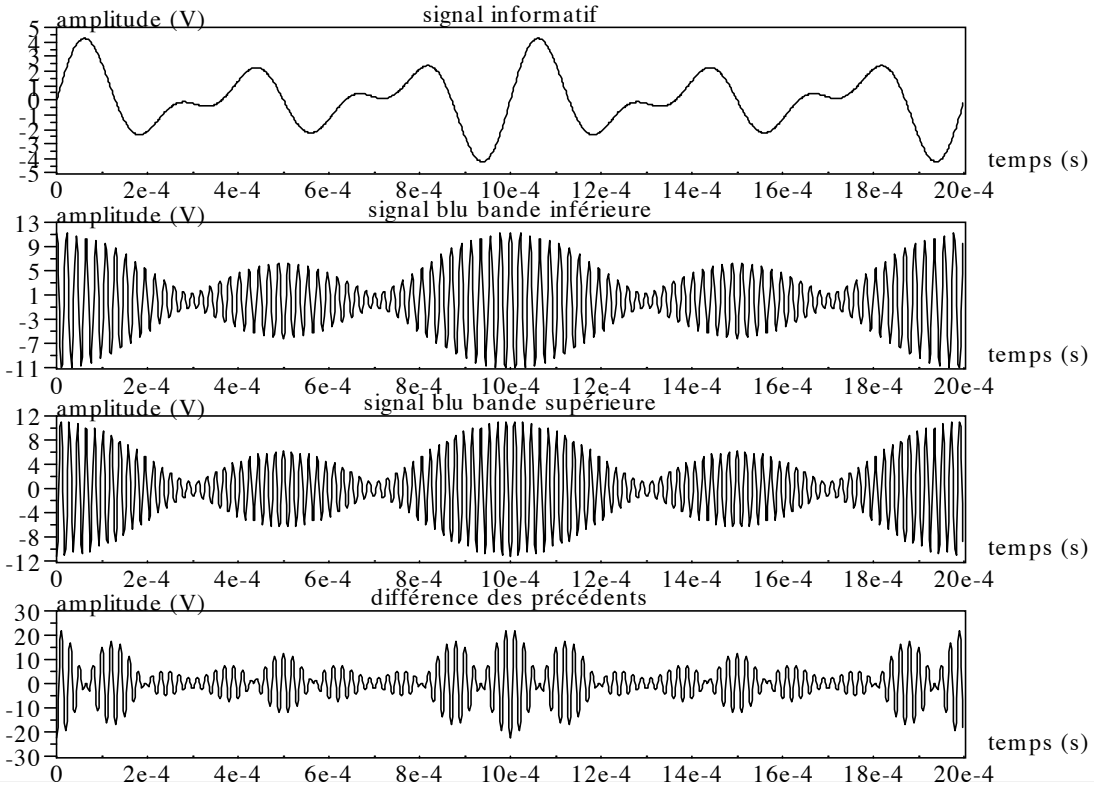
Remarques :

- un signal modulé en BLU donne un autre signal sinusoïdal, sa représentation est donc de peu d'intérêt,
- on pouvait s'affranchir du passage dans le domaine fréquentiel en effectuant un produit de convolution entre le signal double bande et la réponse impulsionnelle du filtre (c'est à dire la transformée de Fourier inverse de sa fonction de transfert). Cette approche sera vue ultérieurement.
- le signal temporel « sblu » peut être calculé plus simplement en faisant la transformée de Fourier inverse du produit Smod par le gain du filtre. Le spectre correspondant est celui d'une somme d'exponentielles complexes (les raies dans la bande supérieure à fe/2 sont très négligeables). En prenant alors la partie réelle ou imaginaire de la transformée inverse on obtient la somme des sinusoïdes à un terme de phase près (la partie réelle de e^{jx} est cosx et la partie imaginaire sinx).

On pouvait également générer directement, pour cet exemple très simple le signal modulé ; le programme ci-après synthétise les signaux de la bande inférieure et de la bande supérieure. Malgré

une apparente similitude, la différence entre les deux signaux modulés est très importante comme le montre le troisième chronogramme.

```
// définition des constantes
N=5000 ; fe=500e3 ; fp=50e3 ; finf1=4e3; finf2=3e3; finf3=5e3;
//
// description du vecteur temps
t=(0:N-1)/fe ;
//
// définition des différents signaux informatif, modulé bande supérieure et inférieure
s_inf=2*sin(2*%pi*(finf2)*t)+1*sin(2*%pi*(finf1)*t)+1.5*sin(2*%pi*(finf3)*t);
sblu_inf=5*cos(2*%pi*(fp-finf2)*t)+2.5*cos(2*%pi*(fp-finf1)*t)+3.75*cos(2*%pi*(fp-finf3)*t);
sblu_sup=-5*cos(2*%pi*(fp+finf2)*t)-2.5*cos(2*%pi*(fp+finf1)*t)-3.75*cos(2*%pi*(fp+finf3)*t);
s_dif=sblu_sup-sblu_inf;
//
xbasc() ; xset( "font size", 4);
xsetech([0,0,1,1/4]) ; plot2d(t (1:N/5) ,s_inf(1:N/5) ) ;
xtitle("signal informatif","temps (s)","amplitude (V)");
//
xsetech([0,1/4,1,1/4]) ; plot2d(t(1:N/5) ,sblu_inf(1:N/5) ) ;
xtitle("signal blu bande inférieure","temps (s)", "amplitude (V)");
//
xsetech([0,2/4,1,1/4]) ; plot2d(t(1:N/5) ,sblu_sup(1:N/5) ) ;
xtitle("signal blu bande supérieure","temps (s)", "amplitude (V)");
//
xsetech([0,3/4,1,1/4]) ; plot2d(t(1:N/5) ,s_dif(1:N/5) ) ;
xtitle("différence des précédents","temps (s)", "amplitude (V)");
```



Malgré une grande ressemblance, il ne faudra pas chercher à comparer la courbe théorique « sblu_inf » est la courbe obtenue par filtrage « sblu ». Le filtre introduit en effet un déphasage très important dans la bande utile.

Reprendre cet exemple de signal blu depuis le début en introduisant cette fois un filtrage idéal : on supprimera pour cela la partie du spectre « Smod » correspondant à des fréquences supérieures à 50 kHz en les remplaçant par une matrice ligne de zéros. Calculer ensuite le signal temporel « sblu » par transformée inverse et le comparer avec « sblu_inf » en affichant la différence des deux.

6. Modulations angulaires

Rappels théoriques

L'expression ci-après rappelle l'équation d'un signal modulé en fréquence :

$$u_m(t) = U_P \sin[2\pi (F_0 + K_F\ m(t))\ t\ ]$$

La fréquence instantanée vaut alors :

$$f(t) = F_0 + K_F\, m(t)$$

tandis que la phase instantanée vaut :

$$\varphi(t) = 2\pi F_0 t + 2\pi K_F \int m(t)\, dt$$

L'indice de modulation δ est défini comme l'écart de phase $\Delta\varphi$ autour de la phase $2\pi F_0 t$ de la porteuse.
Dans le cas d'un signal modulant sinusoïdal, soit $m(t)=U_i \sin(2\pi f_i t)$, l'intégration due au passage de la fréquence à la phase du signal, nous amène à l'expression :

$$\delta = \frac{K_F\ U_i}{f_i}$$

Comme on peut le constater, l'indice de modulation dépend de l'amplitude et de la fréquence du signal informatif. Lorsque plusieurs fréquences sont mises en jeu, on prend la plus élevée pour définir cet indice (qui sera alors d'autant plus faible).

L'expression d'un signal modulé en phase est la suivante :

$$u_m(t) = U_P\ \sin[2\pi\ F_0 t + K_P\ m(t)\ ]$$

La fréquence instantanée a pour expression :

$$f(t) = F_0 + \frac{1}{2\pi} K_P\ \frac{d\ m(t)}{dt}$$

tandis que la phase instantanée vaut :

$$\varphi(t) = 2\pi F_0 t + K_p\ m(t)$$

L'indice de modulation, toujours défini comme l'écart de phase autour de la phase de la porteuse, a pour expression dans le cas du signal informatif sinusoïdal précédent :

$$\delta = K_P\ U_i$$

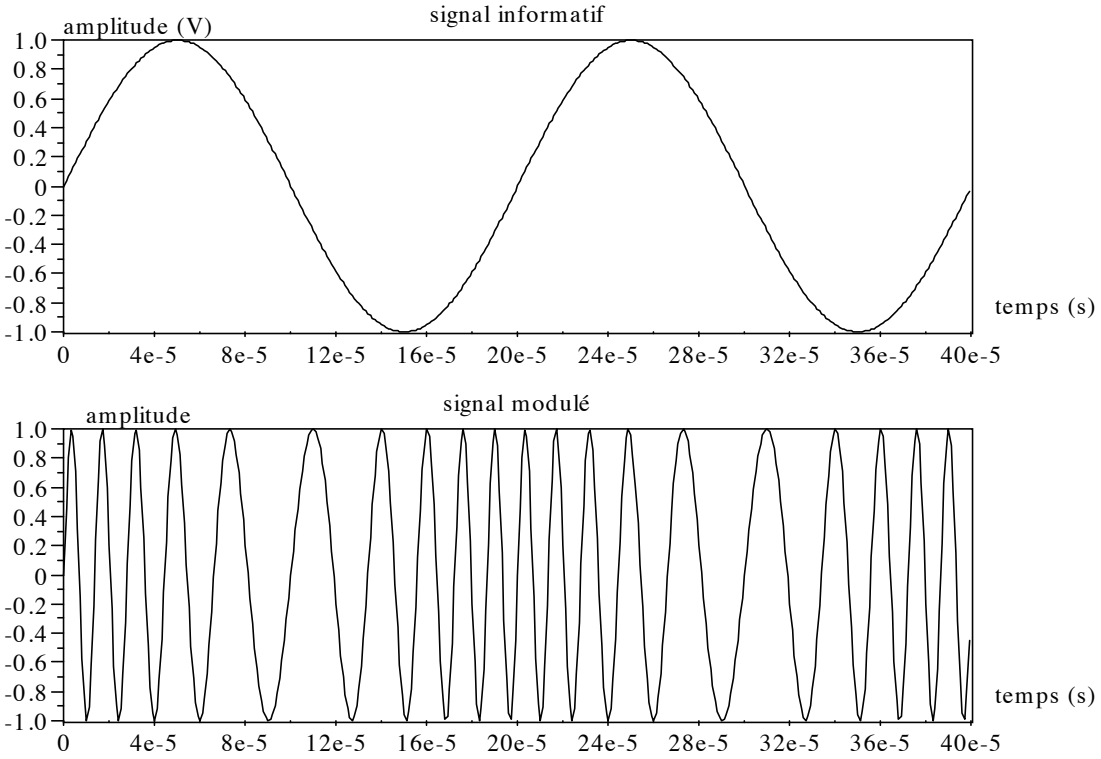
Ce bref rappel permet de mettre en évidence que pour faire une modulation de phase avec un modulateur de fréquence, il suffit de dériver d'abord le signal informatif.
De même, pour faire une modulation de phase avec un modulateur de fréquence, il faut d'abord intégrer le signal modulant.

Visualisation temporelle d'un signal modulé en phase

La modulation de phase est un peu plus simple à mettre en place avec Scilab que la modulation de fréquence, nous commencerons donc par elle.
La mise en œuvre découle presque directement de l'expression théorique comme le montre le programme suivant, où la porteuse a une fréquence de 50 kHz et le signal informatif sinusoïdal de 5 kHz pour un échantillonnage 1 MHz.

```
clear
// définition des constantes
N=400 ; fe=1e6 ; fp=50e3 ; finf=5e3; ind=5;
//
// description des vecteurs temps et signal
t=(0:N-1)/fe ;
//
```

```
,,
sinf=1*sin(2*%pi*finf*t);
smod= 1*sin(2*%pi*fp*t + ind*sinf);
//
// initialisation de l'affichage
xbasc() ; xset( "font size", 4);
//
// affichage du signal informatif
xsetech([0,0,1,1/2]) ; plot2d(t,sinf) ; xtitle("signal informatif","temps (s)","amplitude (V)");
//
// affichage du signal modulé
xsetech([0,1/2,1,1/2]) ; plot2d(t, smod) ;
xtitle("signal modulé en phase","temps (s)", "amplitude");
```



Comme on peut l'observer sur ces courbes (et le déduire de la théorie), la fréquence est maximale lorsque la dérivée du signal informatif est maximale (c'est à dire aux passages par zéro par front montant du signal).

Le signal modulé ne subit pas de discontinuité de phase, comme nous sommes souvent habituées à le voir dans les applications. Ceci est du à la nature sinusoïdale du signal modulant, les exemples étant plutôt donnés pour des signaux modulants numériques.
Cependant, une modulation de fréquence d'un signal analogique est souvent une modulation de phase également. En effet comme nous l'avons vu, l'indice de modulation, en modulation de fréquence diminue avec la fréquence du signal modulant ; il en va de même de l'immunité au bruit. Pour compenser, on fait alors subir une pré-accentuation fréquentielle avec une pente du premier ordre, à partir d'une certaine fréquence (2,1 kHz par exemple en radio-diffusion FM) au signal modulant ; ce qui revient à le dériver à partir de cette fréquence. La modulation est donc de phase pour les hautes fréquences, et de fréquence pour les basses.

Proposer un programme permettant d'illustrer la modulation de phase pour un signal numérique (Modulation à Déplacement de Phase –MDP- ou encore PSK pour Phase Shift Keying).
Deux phases seront possibles (0 ou π) selon que le bit est à 1 ou 0. On choisira donc en conséquence le rapport entre le débit d'information numérique et la fréquence porteuse.

Pour réaliser le signal modulant numérique, on pourra utiliser une technique assez similaire à celle utilisé pour la représentation d'un signal carré (à la différence près que cette fois le signal n'est pas périodique).
A partir d'une suite s_simb de symboles 1 et 0, établir le signal proprement dit, s_nrz , en « suréchantillonnant » le signal d'origine. Pour cela, on effectuera la multiplication par le vecteur s_simb, d'une matrice colonne de « 1 » dont le nombre de ligne est égal au coefficient de suréchantillonnage. Le résultat est ensuite mis sous forme de matrice ligne par la fonction « matrix » (consulter l'aide en ligne ou les annexes pour plus d'information sur les différentes fonctions).

Le programme ci-après donne l'exemple d'un signal de 8 symboles aléatoires, suréchantillonnés d'un facteur 16.

```
clear
// constante de suréchantillage
N=16;
// vecteur de symboles
s_symb=[1 0 1 1 0 0 1 0];
// vecteur temps
t=(0 : 8*N-1);
// signal modulant
s_nrz=matrix(ones(N,1)*s_symb, 1,  8*N);
xbasc(); plot2d(t,s_nrz, rect=[0,-0.5,8*N,1.5]);
```

Signal modulé en fréquence

La description d'une modulation de fréquence est un peu plus complexe sous Scilab. En effet, comme on peut le constater dans l'expression théorique, le signal modulant agit directement sur la fréquence, tandis que pour calculer le sinus nous avons besoins de la phase.
Il va donc falloir intégrer (numériquement) la fréquence pour avoir la phase.

Une première méthode retenue consiste à définir un vecteur ligne contenant les valeurs de la fréquence à chaque instant, et de calculer le vecteur phase dans lequel l'élément de rang i est la somme de tous les éléments de rang 1 à i+1 du vecteur fréquence, le tout pondéré par 2π et par le pas d'échantillonnage. Pour cela on met en place une boucle d'intégration.

Le programme suivant illustre ce principe pour un signal sinusoïdal, en calculant la sinusoïde par la méthode classique et par celle décrite plus haut.

```
clear
//
// définition des constantes et du temps
N=1000 ; fe=1e6 ;f0=1e3; t=1/fe*(0:N-1);
//
// définition du vecteur fréquence
fp=f0*ones(1,N);
//
// définition de la phase (vecteur colonne)
for i=1:N,
theta(i)=2*%pi/fe*sum(fp(1:i));
end;
//
// calcul du sinus par les deux méthodes
```

```
//
s1=sin(theta');
s2=sin(2*%pi*f0*t);
//
// affichage
xbasc();
xsetech([0,0,1,1/2]) ; plot2d(t,s1) ;  xtitle("méthode de la phase","temps (s)","amplitude (V)");
xsetech([0,1/2,1,1/2]) ; plot2d(t,s2) ;  xtitle("méthode classique","temps (s)","amplitude (V)");
```

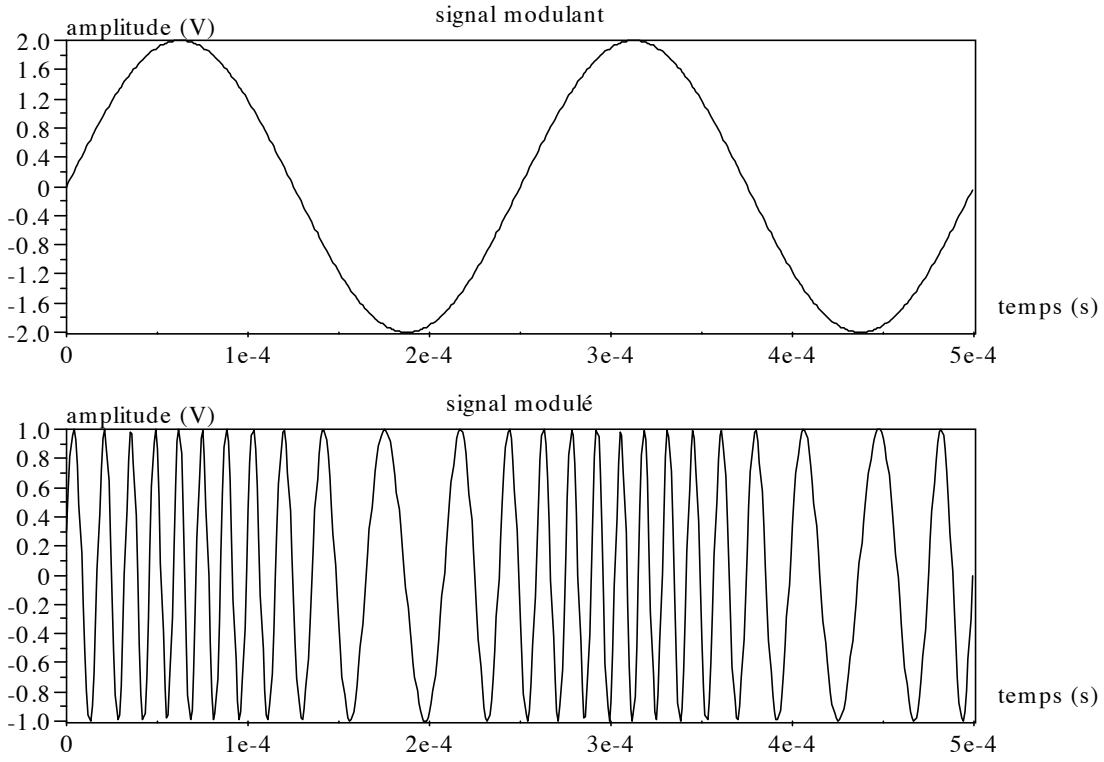
Comme on le constatera lors de l'essais, le temps de calcul est assez long (fonction de l'ordinateur utilisé), Scilab étant en effet comme on l'a précisé au début, optimiser pour le calcul matriciel et non pour les boucles de programmation. Aussi préférera t-on utiliser la seconde méthode.

Celle-ci consiste à générer le vecteur « thêta » en multipliant le vecteur fréquence « fp » par une matrice carrée de dimension N, composée de « 1 » sur la diagonale et au-dessus de celle-ci, et de « 0 » ailleurs. Pour créer cette matrice, on génère d'abord une matrice de « 1 » à l'aide de l'instruction « ones » puis on ne garde que la partie supérieure au moyen de l'instruction « triu » (voir aide ou annexe 1).
La taille de la matrice étant importante, il faut augmenter la mémoire disponible allouée à Scilab à une valeur supérieure à celle par défaut, au moyen de l'instruction « stacksize ».

```
clear
//
// augmentation de la taille de la mémoire allouée à Scilab
stacksize(1.5e6);
//
// définition des constantes et du temps
N=1000 ; fe=1e6 ;f0=1e3; t=1/fe*(0:N-1);
//
// définition du vecteur fréquence
fp=f0*ones(1,N);
//
// définition du vecteur phase
theta=2*%pi/fe*fp*triu(ones(N, N));
//
// calcul du sinus par les deux méthodes
s1=sin(theta);
s2=sin(2*%pi*f0*t);
//
// affichage
xbasc();
xsetech([0,0,1,1/2]) ; plot2d(t,s1) ;  xtitle("méthode de la phase","temps (s)","amplitude (V)");
xsetech([0,1/2,1,1/2]) ; plot2d(t,s2) ;  xtitle("méthode classique","temps (s)","amplitude (V)");
```

Le programme suivant permet d'afficher un signal modulé en fréquence par une tension sinusoïdale. La porteuse à une fréquence de 50 kHz, le signal informatif à une fréquence de 4 kHz et une amplitude de 2 V, l'indice de modulation des de 7 (afin de visualiser correctement les variations de fréquence).
La fréquence instantanée est un vecteur une ligne pour N colonnes, dont tous les éléments valent la fréquence porteuse, vecteur auquel on ajoute le vecteur correspondant au signal informatif, pondéré par le coefficient de modulation, qui dépend comme nous l'avons vu, de l'indice de modulation, de la fréquence et de l'amplitude du signal informatif.
On intègre ensuite cette fréquence pour obtenir la phase, puis le signal modulé est calculé à partir de cette phase.
Comme on s'en apercevra au déroulement du programme, les instructions de boucle sont lentes à exécuter, et on leur préfèrera lorsque c'est possible, des calculs matriciels, pour lesquels Scilab est optimisé.

```
clear; stacksize(1.5e6);
//
// définition des constantes et du temps
N=500 ; fe=1e6 ; fp=50e3 ; Finf=4e3; Uinf=2; t=1/fe*(0:N-1); ind=7;
//
// définition du signal informatif
sinf=Uinf*sin(2*%pi*Finf*t);
//
// définition du vecteur fréquence
f_inst=fp*ones(1,N)+ ind* Finf /Uinf*sinf;
//
// définition de la phase (vecteur colonne)
theta=2*%pi/fe*f_inst*triu(ones(N, N));
//
// calcul du signal modulé
smod=sin(theta);
//
// affichage
xbasc();xset("font size",4);
xsetech([0,0,1,1/2]) ; plot2d(t,sinf) ;  xtitle("signal modulant","temps (s)","amplitude (V)");
xsetech([0,1/2,1,1/2]) ; plot2d(t,smod) ;  xtitle("signal modulé","temps (s)","amplitude (V)");
```



Les courbes montrent que la fréquence du signal modulé est d'autant plus grande que l'amplitude du signal informatif est important, on a donc bien une modulation de fréquence (comparer par rapport à la modulation de phase).

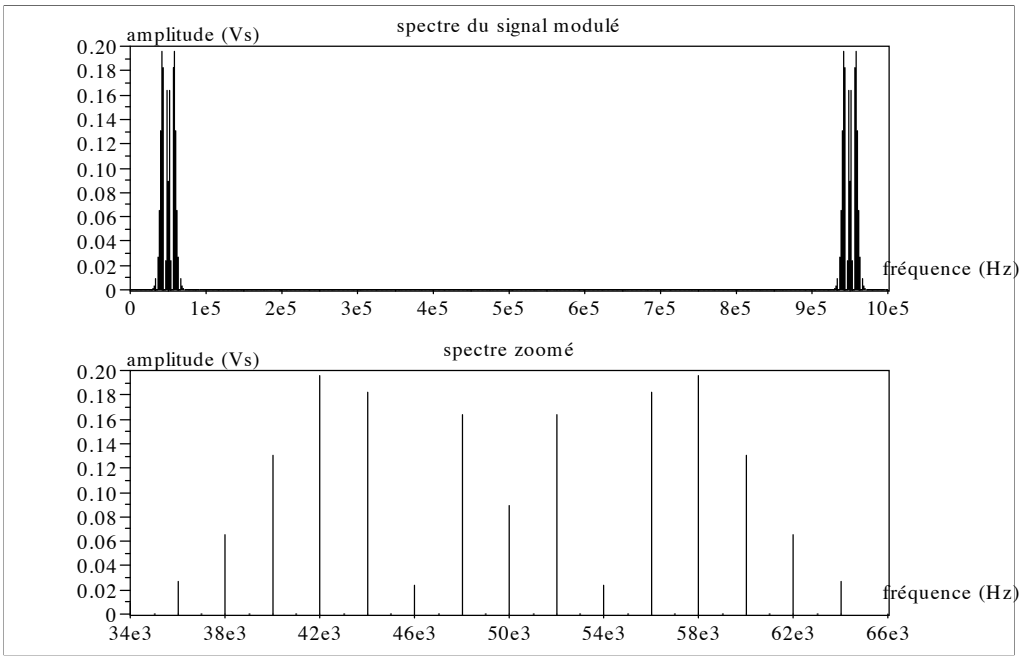
Proposer un programme permettant d'illustrer la Modulation à Déplacement de Fréquence (MDF ou Frequency Shift Keying –FSK-) en vous inspirant de la modulation à déplacement de phase. Comparer les résultats obtenus.

Observation spectrale

Un signal sinusoïdal de fréquence F_0, modulé en fréquence par une sinusoïde de fréquence F_i, est périodique, donc décomposable en série de Fourier. Le spectre est centré autour d'une raie à F_0, et composé de raies espacées de F_i .
Les coefficients sont donnés par les fonctions de Bessel, que l'on trouvera en annexe. Ces coefficients dépendent de l'indice de modulation.

Reprenons nos signaux analogiques modulés en fréquence, en augmentant le nombre de points visualisés. Cette opération permettra comme on le verra lors d'une prochaine séance, d'améliorer la résolution fréquentielle (qui est fonction du temps d'analyse).
Le programme suivant permet d'afficher le spectre du signal modulé, pour un indice de modulation de 5.

```
clear; stacksize(1.5e6);
// définition des constantes et du temps
N=1000 ; fe=1e6 ; fp=50e3 ; Finf=2e3; Uinf=2; t=1/fe*(0:N-1); ind=5;
//
// définition du signal informatif et de la fréquence instantanée
sinf=Uinf*sin(2*%pi*Finf*t);f_inst=fp*ones(1,N)+ ind* Finf /Uinf*sinf;
//
// définition de la phase et calcul du signal modulé
theta=2*%pi/fe*f_inst*triu(ones(N, N));
smod=sin(theta');
//
// définition de l'échelle de fréquence et calcul du spectre
f=fe/N*(0:N-1);
S=fft(smod,-1);
//
// affichage
xbasc(); xset("font size",4);
xsetech([0,0,1,1/2]); plot2d3(f,1/N*abs(S));
xtitle("spectre du signal modulé","fréquence (Hz)","amplitude (Vs)");
xsetech([0,1/2,1,1/2]); plot2d3(f(35:65),1/N*abs(S(35:65)));
xtitle("spectre zoomé","fréquence (Hz)","amplitude (Vs)");
```



Vérifier, à l'aide des courbes fournies en annexe, que les raies ont bien les amplitudes attendues.

Choisir un nouvel indice de modulation de manière à supprimer la raie à la fréquence porteuse ; vérifier à l'aide du programme.

Vérifier ces résultats pour une modulation de phase.

Bibliographie

Documentation de la « maison mère » : http://www-rocq.inria.fr/scilab/

Démarrer en Scilab par B. Ycart à l'adresse :
http://www.math-info.univ-paris5.fr/Enseignements/demarre_scilab/demarre_scilab.htm

Une introduction à Scilab, document « newdocA4.pdf » sur :
http://www.iecn.u-nancy.fr/~pincon/scilab/newdocA4.pdf

Annexe 1 : première approche

Les possibilités de Scilab sont très importantes, notre objectif est ici de passer en revue les fonctionnalités de base, ainsi que celles qui nous seront utiles pour le traitement du signal.

Initialiser
Avant de commencer un nouveau programme, il est conseillé d'initialiser toutes les variables du logiciel par l'instruction « **clear** » (en minuscules), à moins que l'on souhaite garder des variables précédemment définies.

Entrer un vecteur ligne
Plusieurs syntaxes possibles :

```
-->s=[0 1 2 3]
 s  =

!  0.    1.    2.    3. !
```

```
-->s=[0,1,2,3]
 s  =

!  0.    1.    2.    3. !
```

```
-->s=[0:1:3]
 s  =

!  0.    1.    2.    3. !
```

```
-->s=[0:3]
 s  =

!  0.    1.    2.    3. !
```

La troisième syntaxe se lit comme un vecteur allant de 0 à 3 par pas de 1, celui-ci étant 1 par défaut il peut être omis comme dans la quatrième syntaxe.
Dans le cas d'un incrément, les crochets peuvent être omis, ou remplacés par des parenthèses :

```
-->s=(0:3)
 s  =

!  0.    1.    2.    3. !
```

```
-->s=0:3
 s  =

!  0.    1.    2.    3. !
```

Un point virgule à la fin de la ligne évite l'affichage du résultat.

```
-->s=0:3;
```

Entrer un vecteur colonne
Plusieurs syntaxes possibles :

```
-->s1=[0;1;2;3]
 s1  =

!  0. !
!  1. !
!  2. !
!  3. !
```

```
-->s1=[0:3]'
 s1  =

!  0. !
!  1. !
!  2. !
!  3. !
```

```
-->s1=[0:1:3]'
 s1  =

!  0. !
!  1. !
!  2. !
!  3. !
```

```
-->s1=s'
 s1  =

!  0. !
!  1. !
!  2. !
!  3. !
```

Mettre des vecteurs bout à bout, former des matrices

```
-->s2=[0 1 2 3; 0 1 2 3]
 s2  =

!  0.    1.    2.    3. !
!  0.    1.    2.    3. !
```

```
-->s2=[s;s]
 s2  =

!  0.    1.    2.    3. !
!  0.    1.    2.    3. !
```

```
-->s2=[s1';s1']
 s2  =

!  0.    1.    2.    3. !
!  0.    1.    2.    3. !
```

Accéder aux éléments d'une matrice

```
-->s(0)
    !--error    21
invalid index
```

```
-->s(1)
 ans  =

   0.
```

```
-->s(1:2:4)
 ans  =

!  0.    2. !
```

```
-->s2(2,3)
 ans  =

   2.
```

Les deux premières syntaxes montrent que le premier indice d'une matrice est toujours « 1 » dans Scilab, et non « 0 ».
La troisième syntaxe permet d'accéder aux éléments de rang 1 à 4 de « s » par pas de 2.

Ci-dessous, la première écriture permet d'accéder à l'élément de s2 sur la deuxième ligne, troisième colonne, et la suivante à tous les éléments de la troisième colonne sur toutes les lignes :

```
-->s2(2,3)
 ans  =

   2.
```

```
-->s2(:, 3)
 ans  =

!   2. !
!   2. !
```

On peut noter qu'a chaque fois le résultat correspond à un élément unique, une matrice ligne ou une matrice colonne à qui on peut donner un nom (exemple s5=s1(1 :2 :4)).

Matrices particulières

```
-->zeros(2,3)
 ans  =

!   0.    0.    0. !
!   0.    0.    0. !
```

```
-->ones(3,2)
 ans  =

!   1.    1. !
!   1.    1. !
!   1.    1. !
```

```
-->eye(3,4)
 ans  =

!   1.    0.    0.    0. !
!   0.    1.    0.    0. !
!   0.    0.    1.    0. !
```

```
-->rand(3,2)
 ans  =

!   .0683740    .7263507 !
!   .5608486    .1985144 !
!   .6623569    .5442573 !
```

On précise à chaque fois, dans l'ordre le nombre de lignes et de colonnes souhaitées. La première écriture donne une matrice de « 0 », la seconde de « 1 », la troisième une matrice dont les éléments de la diagonale sont des « 1 » et les autres des « 0 », et la dernière donne des éléments aléatoire entre « 0 » et « 1 ».

Données particulières

Elles sont précédées du symbole « % ». Citons :
- le nombre π, soit **%pi**,
- l'imaginaire pur, soit **%i**,
- et très utiles pour lever certaines indéterminations, le plus petit nombre manipulable par Scilab ($2{,}22\ 10^{-16}$), soit **%eps**, ainsi que les type vrai, **%t**, et faux, **%f**.

Opérations particulières
On trouve évidemment :
- les opérations d'addition « **+** » (les matrices doivent alors avoir même dimension),
- de soustraction « **-** » (les matrices doivent alors avoir même dimension),
- de multiplication « ***** » entre matrices (le nombre de colonnes de la première matrice doit alors être égal au nombre de lignes de la seconde), de même que la multiplication de tous les éléments de la matrice par un nombre,
- de division « **/** »,
- la mise à puissance de 10, avec « **e** », par exemple 10e3 (soit 1000).

A ne pas confondre avec ces deux dernières les multiplications et divisions éléments par éléments, notées respectivement « **.*** » et « **./** ». Les matrices doivent alors avoir même dimension.
Observons l'exemple ci-après (à lire de gauche à droite) :

```
-->A=[0,1;2,3]
 A  =
!   0.    1. !
!   2.    3. !
```

```
-->B=[4,5;6,7]
 B  =
!   4.    5. !
!   6.    7. !
```

```
-->C=A*B
 C  =
!   6.    7.  !
!   26.   31. !
```

```
-->D=A .*B
 D  =
!   0.    5.  !
!   12.   21. !
```

On trouvera le même mode de fonctionnement avec la fonction élévation à la puissance et les symboles « **^** » et « **.^** ».

Citons également quelques opérations spécifiques :
- **size (A)** qui donne le nombre de ligne et colonnes de A,
- **length (A)** qui donne le nombre d'éléments de A,
- **matrix (A, m, n)** qui redimensionne A sur m ligne et n colonnes (la lecture se fait colonne par colonne),
- **sum(A)** qui renvoie un nombre correspondant à la somme de tous les éléments de la matrice,
- **sum(A,'r')** qui renvoie un vecteur ligne ('r' pour row) dont chaque élément est la somme des éléments de la colonne correspondante de A,
- **sum(A,'c')** qui renvoie un vecteur colonne ('c' pour column) dont chaque élément est la somme des éléments de la ligne correspondante de A,
- **triu(A)** qui ne garde que la valeur des éléments du triangle (« tri » pour triangle) supérieur (« u » pour up) de la matrice (diagonal comprise) et place des « 0 » ailleurs,
- **tril(A)** qui ne garde que la valeur des éléments du triangle (« tri » pour triangle) inférieur (« l » pour low) de la matrice (diagonal comprise) et place des « 0 » ailleurs,
- **A'** qui renvoie la matrice A transposée et conjuguée,
- **conj(A)** qui renvoie la matrice A conjuguée.

Fonctions particulières
Citons les principales :
- **sqrt( ),** la racine carrée,
- **sin( ), cos( ), tan( )** les sinus, cosinus et tangente,
- **asin( ), acos( ), atan( )** les fonctions inverses des précédentes,
- **exp( ), log( )** les fonctions exponentielle et logarithme népérien,
- **10^( ), log10( )** les fonctions puissance de 10 et logarithme décimal,
- **real( ), imag( )** les fonctions partie réelle et imaginaire d'un complexe,
- **abs( ), phasemag( )** les fonctions module et phase d'un complexe.

On peut noter d'après les exemples précédents, que les éléments d'une matrice sont notés entre crochet, tandis que les arguments d'une fonction sont notés entre parenthèse.

Annexe 2 : l'affichage

Afficher un vecteur « y » en fonction d'un vecteur « x » se fait de manière relativement simple dans Scilab grâce à l'instruction « plot2d(x,y) ». La gestion de l'affichage graphique peut être cependant très puissante (et donc relativement complexe). Nous nous contenterons ici de passer en revue les instructions qui nous serons les plus utiles.
Il faut noter que Scilab est un programme écrit pour fonctionner sous Unix, et l'affichage pose parfois quelques petits problèmes sous Windows si on souhaite utiliser des fonctions un peu complexes, de même que le « copier coller » qui s'autorise quelques décalages. Tous les exemples présentés sont cependant fait à partir de Windows.

Il est possible d'ouvrir plusieurs fenêtres d'affichage à la fois dans Scilab (qui à leur tour pourront contenir plusieurs graphiques), au moyen de l'instruction :
xset("window", x);

où « x » est le numéro de la fenêtre. Cette instruction ouvre crée la fenêtre et la rend active (une seule fenêtre active à la fois). Si une seule fenêtre est utilisée, ce qui sera en souvent notre cas, cette instruction est inutile.

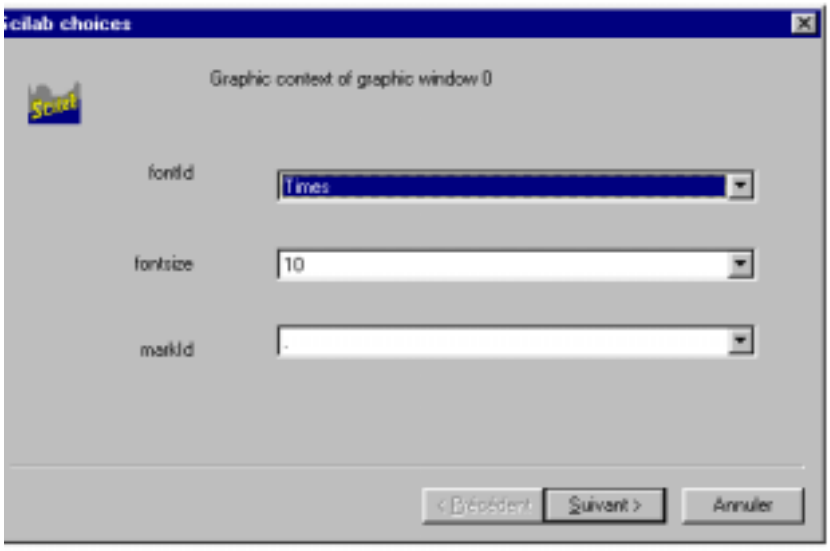
Attention de bien respecter **style des guillemets** (style « anglo-saxon »), ainsi que les **minuscules** !

Avant tout affichage, il est conseillé d'initialiser les paramètres d'affichage par l'instruction :
xbasc(x);

Sans cette instruction Scilab effectue le nouveau tracé sur le précédent. Si les parenthèses sont vides, c'est la fenêtre courante qui est initialisée.

Lorsque l'on souhaite configurer l'affichage de la fenêtre active, on peut utiliser l'instruction :
xset() ;

qui ouvre la fenêtre suivante :



Elle permet alors principalement de choisir la fonte du texte (**fontId**, dont la fonte « symbol » pour les lettres grecques, la taille de celle-ci (**fontsize**), le style des points d'affichage (**markId**) et leur taille (**markIdsize**), l'épaisseur des traits (**thickness**), l'utilisation ou non de la couleur (use color) et la teinte choisie (colors). Pour plus d'information on tapera « help xset » à l'invite.
On réinitialise toutes ces valeurs par la commande **xset("default")**.
Tous ces paramètres sont accessibles directement par lignes de code, comme par exemple le choix de la taille de la fonte (une taille de 4 convient bien en général) et l'épaisseur des traits (en pixels) :

xset("font size",4); xset("thickness",2);
Un titre, ainsi qu'une légende pour l'axe des abscisses et une pour l'axe des ordonnées peuvent être ajoutées au moyen de l'instruction :
xtitle("titre","axe des x","axe des y");

De la même manière il est possible d'ajouter du texte sur le graphique en précisant la position du point en bas à gauche du texte :
xstring(x_bg, y_bg, ["ligne 1"; "ligne 2" ; ….])

Attention : les expressions ne doivent contenir **ni guillemets, ni apostrophes**.

On peut également positionner des flèches sur le graphique, x_dn, y_dn, x_an et y_an représentant respectivement les coordonnées des positions de départ et d'arrivée de la flèche n (la flèche étant à l'arrivée) :
xarrows([x_d1 , x_a1 , x_d2 , x_a2 , …],[y_d1 , y_a1 , y_d2 , y_a2 , …]);

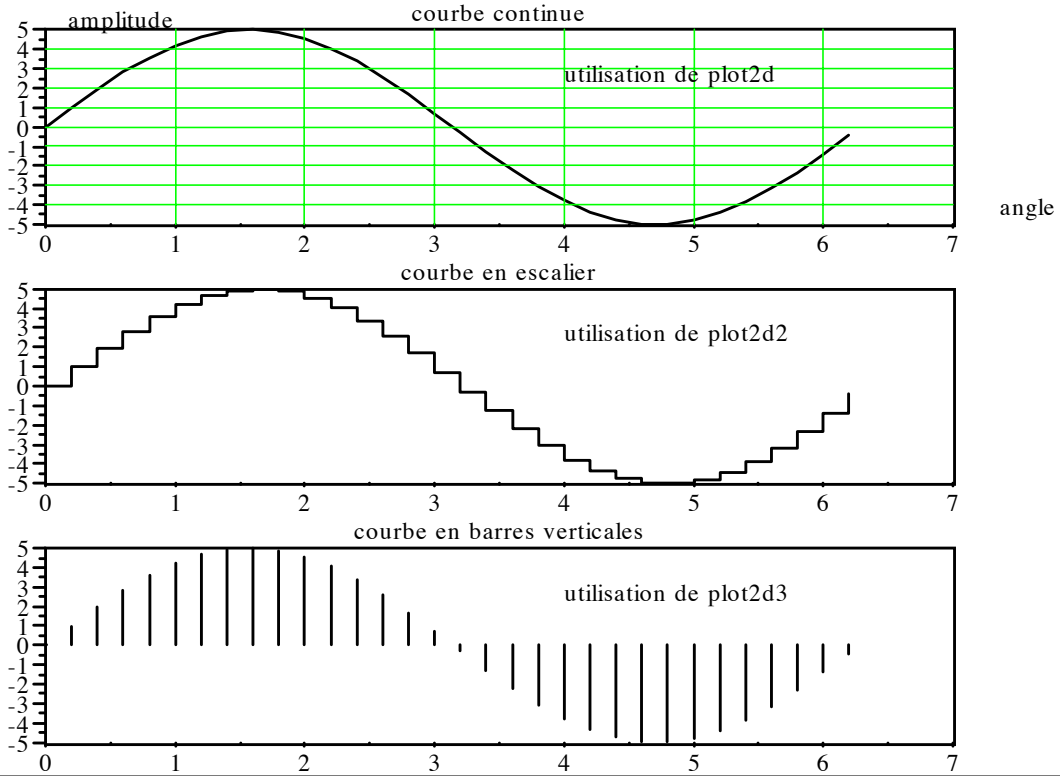
Il est également possible de placer une grille en précisant les style représenté par un entier n, à l'aide de :
xgrid(n) ;

Plusieurs graphes peuvent être tracés dans une même fenêtre au moyen de l'instruction :
xsetech(pos_x, pos_y, largeur, hauteur);

où la fenêtre est vue comme un carré dont les dimensions des côtés sont unitaires, et dont le point origine (0,0) se trouve en haut à gauche (le point (1,1) se trouve donc en bas à droite).
Dans cette fenêtre pos_x et pos_y représente les coordonnées en x et y du coin supérieur gauche du graphe concerné, tandis que largeur et hauteur représente sa taille, toujours dans le même système.

L'instruction « plot2d » permet d'afficher les éléments d'un vecteur en fonction de l'indice de chaque élément (exemple plot2d(x)) ou en fonction de l'élément de même rang d'un second vecteur (exemple plot2d(x,y)). Dans le premier cas, le premier indice est « 1 », dans le second cas, les vecteurs doivent avoir même taille.
Cette instruction est très puissante, on consultera l'aide en ligne pour plus d'information. Signalons simplement que plusieurs déclinaisons de cette instruction sont possibles, comme le montre l'exemple suivant :

```
clear
// définition d'un signal sinusoïdal
theta=[0:.2: 2*%pi]; s=5*sin(theta);
//
//
// initialisation de l'affichage
xbasc(), xset("font size",4); xset("thickness",2);
//
// affichage de quatre graphes dans une fenêtre
xsetech([0,0,1,1/3]);
plot2d(theta,s);
xtitle("courbe continue",”angle”,”amplitude”);
xset("thickness",1);xgrid(3); xset("thickness",2);
xstring( 4 , 2 , [“utilisation de plot2d”]);
//
xsetech([0,1/3,1,1/3]);
plot2d2(theta,s);
xtitle("courbe en escalier");
xstring( 4 , 2 , [“utilisation de plot2d2”]);
//
xsetech([0,2/3,1,1/3]); plot2d3(theta,s);
xtitle("courbe en barres verticales") ;
xstring(4 , 2 , [“utilisation de plot2d3”]);
```



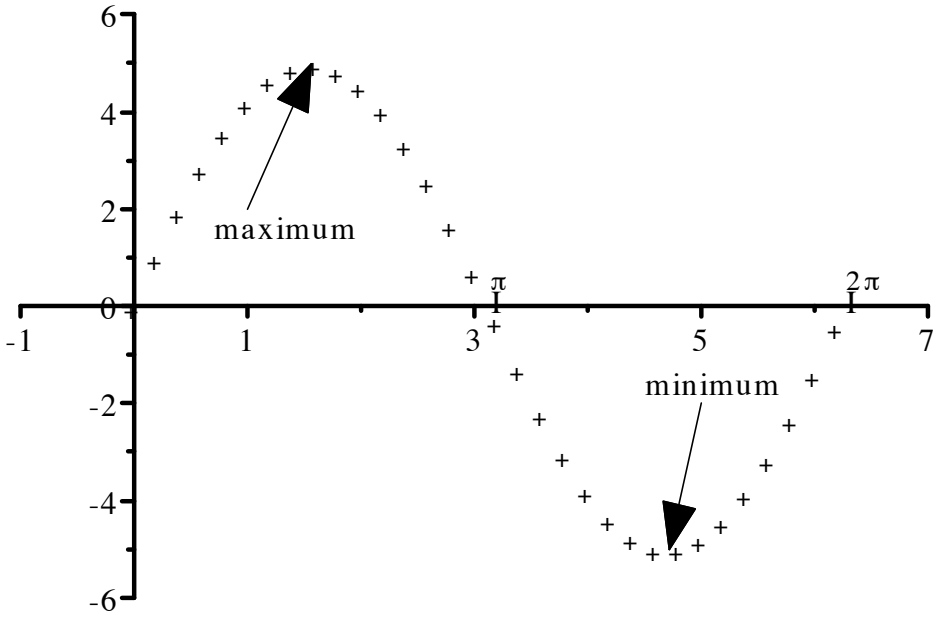
Remarques :
- les instructions plot2di nécessitent de préciser les abscisses, contrairement à plot2d (plot2d1(x) génère une erreur).
- lorsque la grandeur s affichée est complexe, les instruction plot2di affichent la partie réelle de s si on ne donne pas d'autre précision (comme « abs », « imag » ou « phasemag »).

Ces instructions peuvent recevoir des arguments supplémentaires ; citons :
- le style par **style(x)** qui permet de choisir la couleur de l'affichage si x est un nombre positif, ou la forme (croix, pointillés etc…) si x est un nombre négatif,
- les valeurs extrêmes de l'affichage par **rect=[xmin, ymin, xmax, ymax]**,
- les graduations par **nax=[nx, Nx, ny, Ny]** où Nx représente le nombre d'intervalles sur l'axe x et nx le nombre de sous intervalles.
- les choix d'échelles normales ou logarithmiques avec **logflag["hv"]**, où h et v représente les axes horizontal et vertical et prennent la valeur n (pour normal) ou l (pour logarithmique).
- la gestion des axes avec **axesflags=[n]** avec en particulier la valeur n=5 qui permet de tracer des axes passants par le point (0,0).

Le programme ci-après permet de donner un exemple d'utilisation, avec en plus utilisation de flèches, et commentaires en lettres grecques (fonte numéro 1) :

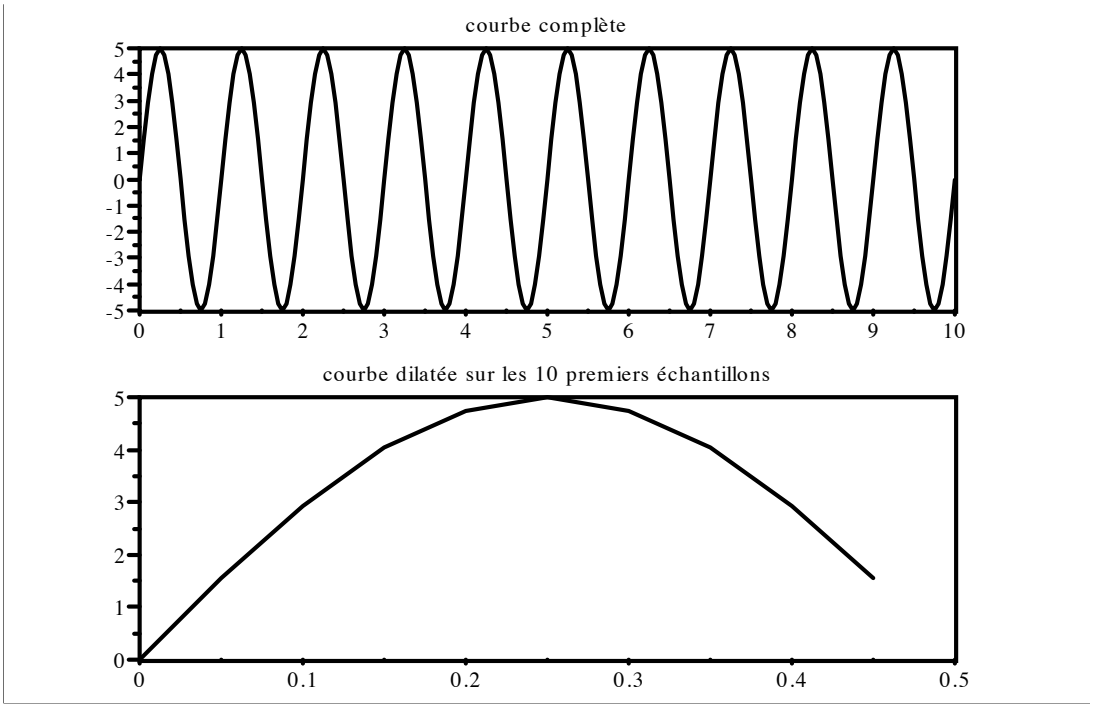
```
clear
// définition d'un signal sinusoïdal
//
theta=[0:.2: 2*%pi]; s=5*sin(theta);
//
// initialisation de l'affichage, fonte 2 et taille, épaisseur des traits et des marques
xset("window",0);xbasc(0);
xset("font", 2, 5); xset("thickness",3); xset("mark size", 18);
//
// affichage "point en croix", extension des échelles, gestion des graduation, placement des axes
plot2d(theta,s,style=-1,rect=[-1, -6, 7, 6],nax=[2,4,2,6], axesflag=[5});
```

```
,,
// affichage de flèches
xset("thickness",1);xarrows([ 1, 1.4, 5, 4.8],[ 2, 4.1, -2, -4.1]);
//
// affichage de commentaires sur la courbe
xstring(0.7 , 1.2 ,["maximum"]); xstring( 4.5 , -2 ,["minimum"]);
//
// affichage des valeurs π et 2π
xset("font", 1, 12); xstring( %pi, 0.2,["p"]); xstring( 2*%pi, 0.2,["2p"]);
//
// placement de graduation pour les valeurs précédentes
xset("font",2, 18); xstring( %pi, -0.3,["I"]); xstring( 2*%pi, -0.3,["I"]);
```



Les instructions nax et rect ont parfois un fonctionnement aléatoire sous windows. En particulier, l'instruction rect, bien pratique pour faire un zoom sur une courbe, laisse « déborder » la courbe en dehors des axes, ce qui, hormis l'aspect inesthétique, rend parfois les « copier coller » difficiles. Aussi si le zoom ne concerne que l'axe horizontal, on préférera utiliser la méthode de l'exemple suivant :

```
clear
// définition d'un signal sinusoïdal
theta=[0:.05:10]; s=5*sin(2*%pi*theta);
//
// initialisation de l'affichage
xbasc(), xset("font size",4); xset("thickness",3);
//
// affichage de quatre graphes dans une fenêtre
xsetech([0,0,1,1/2]); plot2d(theta,s); xtitle("courbe complète");
xsetech([0,1/2,1,1/2]); plot2d(theta(1:10),s(1:10));
xtitle("courbe dilatée sur les 10 premiers échantillons");
```



Ce que nous venons de voir n'est qu'un aperçu très succin des possibilités d'affichage de Scilab ; pour plus d'informations on consultera la documentation fournie avec le logiciel, l'aide, ainsi que les nombreux documents disponibles sur internet.

Annexe 3 : la sauvegarde

Le répertoire de travail

Scilab sauvegarde les données dans le répertoire de travail. Il s'agit par défaut du dossier C:\Scilab-2.6\bin si le logiciel à été installé par la procédure classique. Ce répertoire peut être modifié par la commande :

chdir c:\chemin\répertoire

On peut vérifier qu'il s'agit bien du répertoire souhaité en tapant à l'invite :

pwd

Sauvegarde du contexte

Toutes les variables peuvent être sauvegardées dans un fichier binaire au moyen de la commande :

save Nom_de_fichier

puis rappelées par la commande :

load Nom_de_fichier

D'autres possibilités existent avec les commandes **write** et **read**, ainsi que **mopen**, **mclose**, **mfprintf** et **mfscanf** dont le comportement est identique aux commandes du langage C de nom similaire (au « m » près).

Fichier d'instructions

Comme nous l'avons signalé au début, il est souvent plus simple, lorsque l'on a un peu l'habitude de Scilab, d'écrire les programmes dans un éditeur de texte à part, comme le bloc note de Windows, et de les envoyer dans Scilab par un copier coller.

Word peut également être employé efficacement, en créant des modèles et des macros permettant une configuration correcte évitant toute correction automatique : pas de remplacement des guillemets anglo-saxons par les guillemets français, pas de majuscules en début de phrase, pas de suppression de la seconde majuscule d'un mot, pas d'ordinaux et pas d'exposants. On trouvera tous ces réglage dans Outils / Correction automatique…
Les macros pourront également être utilisées pour automatiser l'écriture des lignes de commande d'affichage (xsetech, xtitle etc…) un peu fastidieuse à mettre en place.

Les instructions d'un programme peuvent être sauvegardées dans un fichier texte (extension .txt ou encore **.sce** par habitude pour les fichiers Scilab) et exécutées moyen de la commande :

exec Nom_du_fichier.extension

Annexe 4 : la programmation

Les boucles de programmation

Scilab offre la possibilité d'utiliser des boucles de programmation classiques, tels que : **for, while, if… then…else, select…case**. On trouvera plus d'information dans l'aide en ligne, et des exemples d'utilisation dans le présent document, chapitre sur la modulation de fréquence et annexe sur les fonctions de Bessel.
Cependant, comme nous l'avons vu, on pourra souvent (et on devra) remplacer la boucle par un calcul matriciel, beaucoup plus efficace avec Scilab.

Les fonctions

Beaucoup plus intéressantes, les fonctions permettent d'enregistrer dans un fichier texte à part, des lignes de programme souvent utilisées, et de les rappeler ensuite. La syntaxe est la suivante :

function [y1, y2,…yn] = nom_de_fonction(x1, x2…xm)

où les termes xi et yi représentent respectivement les arguments d'entrée et de sortie de la fonction. Suit alors la description de la fonction.

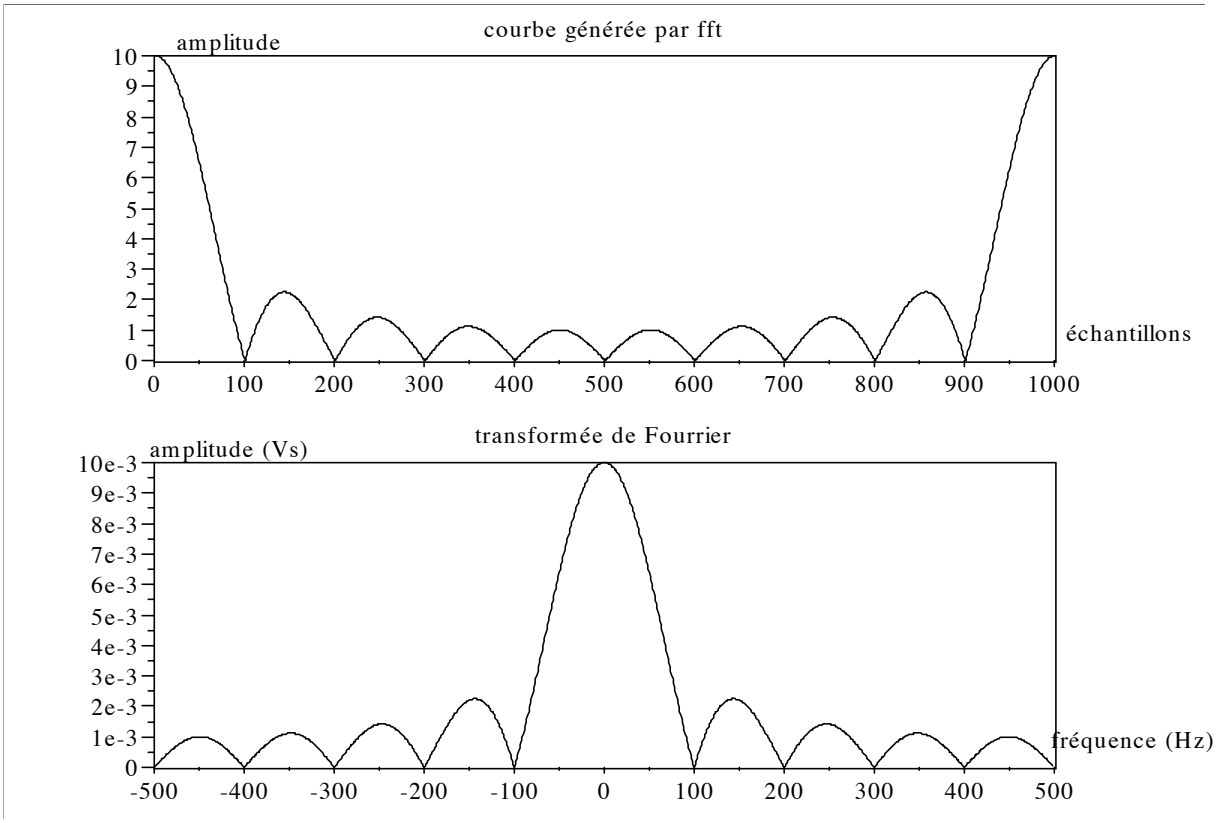
Dans l'exemple suivant, deux fonctions ont été enregistrées dans le fichier « mes_fonctions.txt » dans le répertoire de travail courant (voir annexe sur la sauvegarde). L'extension habituelle est plutôt « .**sci** », mais un « .txt » se génère plus naturellement avec le bloc note.
La première donne les éléments pour tracer une transformée de Fourier bilatérale sous sa forme habituelle pour les signaux à temps continu, de –fe/2 à fe/2 (à un échantillon près). On se réfèrera au chapitre sur l'observation spectrale pour plus d'information.
La seconde permet à partir de la description sur une période d'un symbole, de le reproduire N fois (voir chapitre sur l'affichage d'un signal quelconque).

```
// "tfcbi" calcul la fft, divise par le nombre d'échantillons,
// met sous forme -fe/2, fe/2, et génère le vecteur fréquence pour l'affichage
//
function[S, f]=tfcbi(s,fe)
N=length(s)
S1=1/N*fft(s,-1);
S=[S1(N/2+1:N), S1(1:N/2)];
f=fe/N*[-N/2 : N/2-1];
//
// "repet" répète un vecteur N fois (pour créer un signal périodique)
//
function[S]=repet(s,N)
S=matrix(s' *ones(1, N), 1 , N*length(s));
//
```

Le programme suivant utilise une des deux fonctions, chargées par « **getf(' ')** », afin de tracer la transformée de Fourier d'un signal porte.

```
clear;
//
// chargement du fichier texte des fonctions
getf('mes_fonctions.txt');
//
// définition des constantes
a=10 ;  N=1000 ;Te=1e-3 ;
//
// description du signal
```

```
,,
s1=[ones(1, a), zeros(1, N-a)] ;
//
// appel de la fonction
[S1, f]=tfcbi(s1,1/Te);
//
// calcul de la transformée avec seulement la fonction fft (pour comparaison)
S=fft(s1,-1);
// affichage
xbasc(); xset("font size", 4);
xsetech([0,0,1,1/2]); plot2d(abs(S)); xtitle("courbe générée par fft","échantillons","amplitude");
xsetech([0,1/2,1,1/2]); plot2d(f,abs(S1));
xtitle("transformée de Fourier"," fréquence (Hz)","amplitude (Vs)");
```



La transformée de Fourier d'une porte d'amplitude unitaire et de largeur aTe est un sinus cardinal d'amplitude a.Te et dont le lobe principal à une largeur de 2/(a.Te). On retrouve bien la forme souhaitée, ainsi qu'une échelle de fréquence correcte. Si l'amplitude semble également correcte, il s'agit simplement d'un choix judicieux de la valeur 1 pour NTe.

> Comme nous le verrons en effet par la suite, la transformée de Fourier étant calculé pour des valeurs discrètes (le choix de plot2d au lieu de plot2d3 permet simplement l'affichage d'un spectre continu), le signal temporel est donc considéré comme périodique. Il s'agit donc d'un calcul de coefficient de série de Fourier (à un rapport N près). Pour obtenir des amplitudes correspondant à la transformé, il suffit de déterminer la condition à vérifier en égalisant sur une fréquence quelconque (la fréquence nulle est la plus simple), la valeur du coefficient de la série et la valeur de la transformée.

Remarques :

- les variables internes des fonctions ne modifient pas le contexte général du programme,
- seul le premier argument de sortie de la fonction peut être utilisé ; nous aurions pu écrire par exemple « S1=tfcbi(s1,1/Te) », le vecteur « f » n'étant alors évidemment pas utilisable.

Annexe 5 : polynômes et fonctions de transfert

La fonction de transfert d'un quadripôle peut être décrite et sa réponse fréquentielle affichée, de différentes façons.

Les exemples ci-après décrivent un filtre passe bande du second ordre. On rappelle que la fonction de transfert d'un tel filtre a pour expression :

$$h = \frac{2\,z\,\frac{p}{\omega_0}}{1 + 2\,z\,\frac{p}{\omega_0} + \frac{p^2}{\omega_0^2}}$$

le zéro est nul au numérateur, tandis que si le coefficient d'amortissement est inférieur à l'unité, les pôles du dénominateur sont des complexes conjugués :

$$p_{1\ 2} = -\omega_0 (z \pm j\sqrt{1 - z^2})$$

Un polynôme peut être décrit dans Scilab à l'aide de la fonction « poly » par ses racines (première écriture), ou bien ses coefficients (seconde écriture avec l'argument 'c') en commençant par le poids faible. La fonction de transfert est ensuite obtenue en faisant des opérations sur les différents polynômes.

```
clear;
//
// définition des paramètres
w0=10; z=.3;
//
// définition des pôles et zéro
pole_d=-w0*(z+%i*sqrt(1-z^2)); zero_n=0;
//
// définition du numérateur et dénominateur
n=2*z/w0*poly([zero_n],'p');
d=1/w0^2*poly([ pole_d, pole_d' ],'p');
//
// fonction de transfert
h=n/d

 h  =
         .06p
    -----------------
    1 +  .06p +  .01p2
```

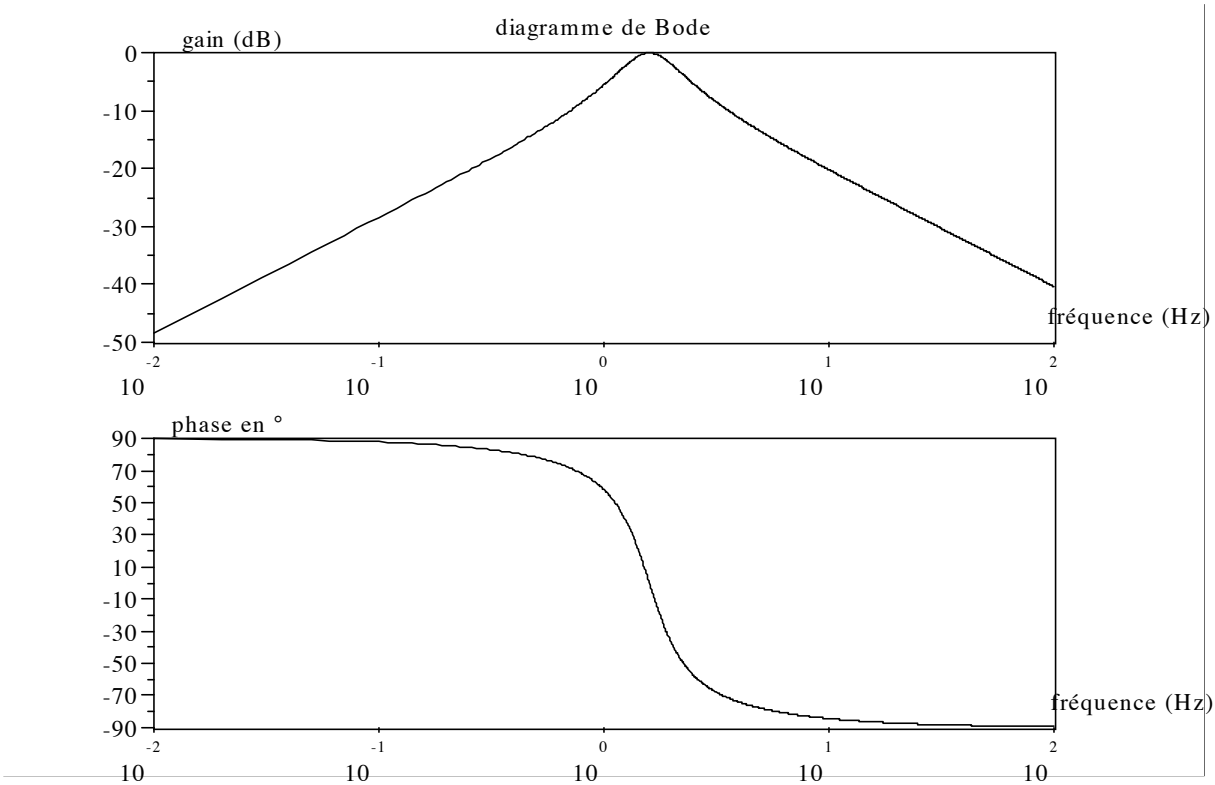
```
clear;
//
// définition des paramètres
w0=10; z=.3;
//
// définition du numérateur
// et dénominateur par les coefficients
n=poly([0,2*z/w0],'p','c');
d=poly([1, 2*z/w0, 1/w0^2],'p','c');
//
// définition de la fonction de transfert
h=n/d

 h  =
         .06p
    -----------------
    1 +  .06p +  .01p2
```

La fonction « roots() » permet de retrouver éventuellement les racines d'un polynôme.

```
-->roots(d)
 ans  =
! - 3. + 9.539392i !
! - 3. - 9.539392i !
```

L'affichage du diagramme de Bode peut être effectué en calculant les points souhaités à l'aide de la fonction « freq », à laquelle il faut préciser numérateur, dénominateur et domaine de variation de la variable (qui est complexe dans notre cas).

```
//
// définition de la plage de variation
f=[.01:.01:100];
//
// calcul des points
h=freq(n,d,2*%i*%pi*f);
//
// affichage
xbasc(); xset("font size",4);
xsetech([0,0,1,1/2]); plot2d(f,20*log10(abs(h)), logflag=["ln"]);
xtitle("diagramme de Bode","fréquence (Hz)"," gain (dB)");
xsetech([0,1/2,1,1/2]); plot2d(f,phasemag(h), logflag=["ln"]);
xtitle("","fréquence (Hz)","phase en °");
```



Il est également possible de décrire une variable (qui est vue soit comme un polynôme de racine nulle ou comme un polynôme de coefficients [0, 1]) et d'effectuer ensuite les opérations directement avec cette variable.

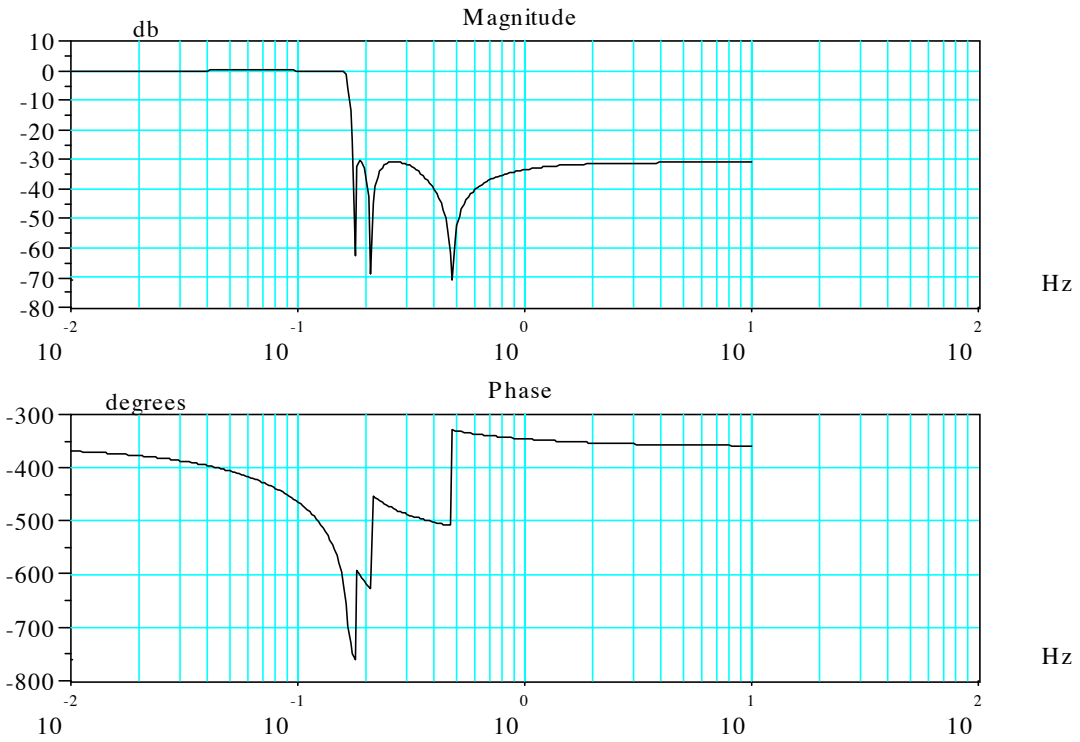
```
clear;
w0=10; z=.3;
p=poly(0,'p');
h=(2*z*p/w0)/(1+2*z*p/w0+(p/w0)^2)

 h =
        .06p
   -----------------
   1 +  .06p +  .01p2
```

```
clear;
w0=10; z=.3;
p=poly([ 0, 1],'p','c');
h=(2*z*p/w0)/(1+2*z*p/w0+(p/w0)^2)

 h =
        .06p
   -----------------
   1 +  .06p +  .01p2
```

Quelle que soit l'option choisie, on peut obtenir directement l'affichage dans le plan de Bode avec l'instruction « bode », si on a définit auparavant un système linéaire avec « syslin ». Cette dernière instruction permet l'utilisation de fonctions particulières, mais est surtout intéressante avec les variables d'état et les systèmes discrets.

L'exemple suivant trace la fonction de transfert d'un filtre passe bas de Cauer, de pulsation normalisée.

```
clear;
//
// définition de la variable p
p=poly(0,'p');
//
// définition de la fonction de transfert
h1=(1+0.1119*p^2)/(1+1.9234*p+1.808*p^2) ;
h2=(1+0.5788*p^2)/(1+0.4164*p+1.062*p^2) ;
h3=(1+0.8003*p^2)/(1+0.071*p+0.948*p^2) ;
h=h1*h2*h3 ;
//
// définition d'un système linéaire continu
h1=syslin('c',h);
//
// affichage de h1 pour des fréquences allant de 0,01 à 10 Hz par pas de 0,01
xbasc();xset("font size",4); bode(h1,.01,10,.01);
```



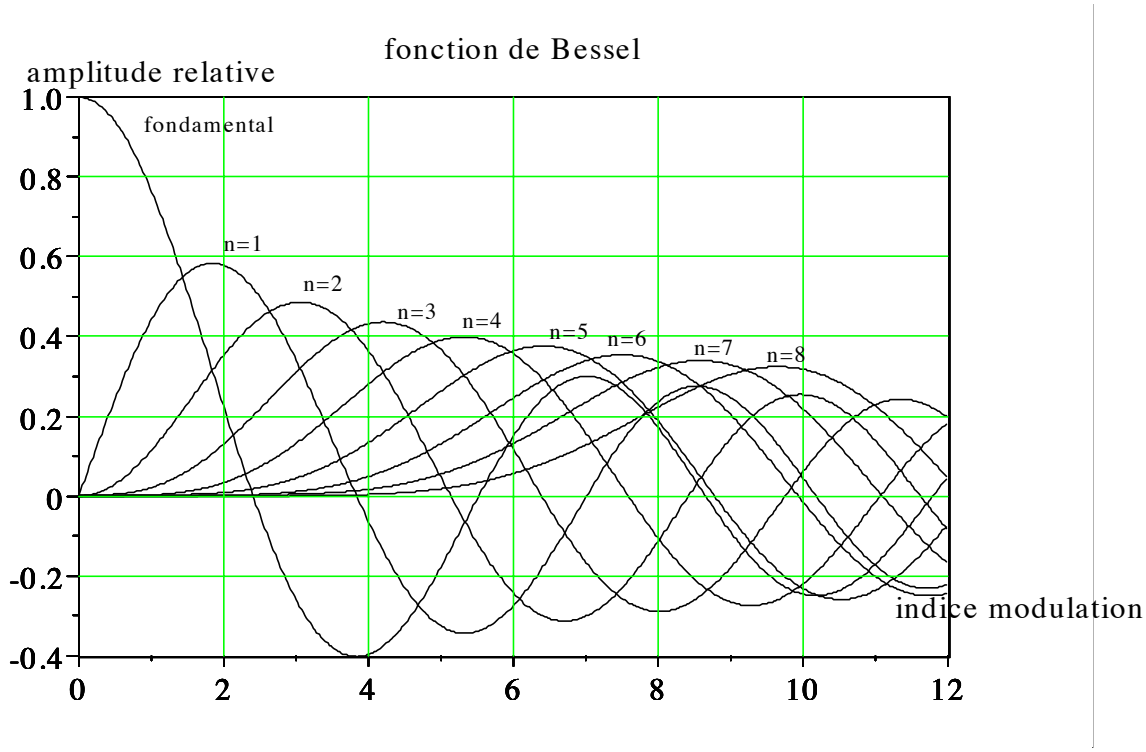
Remarque : les échelles de phase de la fonction « bode » sont parfois fantaisistes ; dans notre cas par exemple, la phase pour la fréquence nulle commence à -360° (au lieu de 0°, soit un tour de moins).

Annexe 6 : fonctions de Bessel

Elles permettent de déterminer l'amplitude des raies du spectre d'une modulation angulaire d'une porteuse de fréquence f_P par un signal sinusoïdal de fréquence f_i, et ont pour expression :

$$J_n(m) = \frac{1}{\pi}\int_0^{\pi} \cos(m \sin(\theta) - n\,\theta)\ d\theta$$

Les harmoniques sont de fréquences $f_P+n.f_i$, avec des amplitudes $J_n(m)$, fonction de l'indice de modulation m.



Le programme suivant a permis le tracé des ces fonctions :

```
clear ; stacksize(1.5e6);
//
// définition du nombre points, de l'indice de modulation maximal, du nombre d'harmonique
N=500; Mmax=12; harmo=8;
//
// définition de l'angle d'intégration
theta=%pi/N*[0:N-1];
//
// définition de la plage de variation de l'indice de modulation
m=Mmax/N*[0:N-1];
//
// définition d'une matrice contenant les différents indices
J=zeros(harmo+1,N);
//
// initialisation de l'affichage
xbasc(); xset("font size",5);
xtitle ("fonction de Bessel","indice modulation", "amplitude relative");
//
// calcul et affichage des fonctions pour 8 fréquences harmoniques et la porteuse
for n=0:harmo,
J(n+1,:)=  1/N*(sum (cos(m'* sin(theta) - n*ones(N,1)*theta),'c'))';
plot2d(m,J(n+1,:), rect=[ 0, -0.4, 12, 1]);
end;
```

```
,,
// placement de la grille
xgrid(3);
//
// changement de taille de fonte
xset("font", 2,4);
//
// placement des valeurs d'indices
xstring( 0.9, 0.9, ["fondamental"]); xstring( 2, 0.6, ["n=1"]); xstring( 3.1, 0.5, ["n=2"]);
xstring( 4.4, 0.43, ["n=3"]); xstring( 5.3, 0.41, ["n=4"]); xstring( 6.5, 0.38, ["n=5"]);
xstring( 7.3, 0.36, ["n=6"]); xstring( 8.5, 0.34, ["n=7"]); xstring( 9.5, 0.32, ["n=8"]);
```