

Scilab utile pour les TP de Signaux

Scilab est un logiciel open source gratuit de calcul numérique. Il peut être téléchargé à l'adresse : <http://www.scilab.org/>

Scilab comporte une aide en ligne efficace :

- un navigateur d'aide (dernier icône en haut à droite)
- **help** suivi d'un nom de commande
- **apropos** suivi d'un mot-clef

qu'il ne faut pas hésiter à utiliser.

- Lancer Scilab : double clic sur l'icône Scilab. Taper alors vos commandes en interactif ou...
- Créer son propre fichier de commandes (vivement recommandé, on ne perd pas tout son travail) en utilisant un éditeur : soit l'éditeur proposé dans la console Scilab (premier icône en haut à gauche SciNotes), mais vous pouvez aussi utiliser votre éditeur préféré. Sauver ce fichier avec un nom comportant l'extension **.sce** (**mon_programme.sce**).
- Le chemin d'accès :
il faut signaler à Scilab où aller chercher vos propres programmes si vous n'exécutez pas votre programme à partir de l'éditeur associé à Scilab. Pour cela, utiliser **chdir** (change directory)
exemple : **cd votre-repertoire** (vous pouvez vérifier avec la commande unix **pwd** que vous êtes bien à l'endroit désiré).
- Exécuter votre programme :
 - soit dans le menu de l'éditeur SciNotes.
 - soit en tapant **exec('mon_programme',0)** ; dans la fenêtre Scilab, si **mon_programme** se trouve dans le répertoire courant, soit en spécifiant le chemin complet **exec('etudiant/mon_programme',0)** ;

Quelques commandes Scilab utiles pour les Tp de Signaux

Généralités

- le signe **//** (double slash) permet de mettre des commentaires (non interprétés par Scilab)
- les instructions sont séparées par une virgule ou un retour à la ligne, sauf si l'on ne désire pas voir le résultat de l'instruction, auquel cas on utilisera un point-virgule.
- le signe **%** précède le nom des variables prédéfinies tels que **%i** (pour $\sqrt{-1}$), **%inf** (pour Infinity), **%pi** (pour 3.14...), **%e** (pour 2.718...), **%T** (pour la constante booléenne "true"="vrai"),...
- pour effacer une variable : **clear nom_de_la_variable** (**clear** pour toutes les effacer sauf les prédéfinies).
- pour faire le point sur les variables utilisées : **who**, **whos()**, **who_user**, ou utiliser le Navigateur de variables.

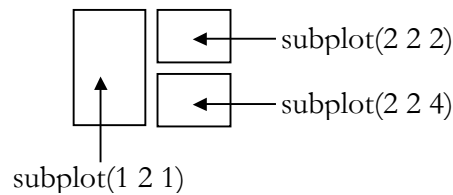
- module et argument du nombre complexe $z = a + \%i * b$:
`module=abs(z)` (si z est un vecteur `abs(z)` est un vecteur)
`argument=atan(imag(z),real(z))` exprimé en radian ou `argument=phasemag(z)` exprimé en degré.
- définition d'un vecteur :
`[1 2]` ou `[1,2]` est un vecteur ligne
`[1 2]'` ou `[1; 2]` est un vecteur colonne
- définition d'une matrice (vecteur de vecteurs) :
`[[1 2];[3 4]]` ou `[1 2; 3 4]`
- Si v est un vecteur `v(i)` est sa i^{e} composante, `v($)` est sa dernière composante ;
 si A est une matrice, `A(i,:)` est sa i^{e} ligne et `A(:,j)` est sa j^{e} colonne.
- taille d'un vecteur ou matrice : `size`.
`length` permet de connaître le nombre d'éléments.
- maximum (resp. minimum) des composantes d'un vecteur `max` (resp. `min`)
`max(v)` est la plus grande composante du vecteur v .
`[m,i]=max(v)` l'entier i est l'indice correspondant à ce maximum (ie. `v(i)=m`).
- pour créer une matrice m lignes n colonnes ne comportant que des 0 : `zeros(m,n)`
- pour créer une matrice m lignes n colonnes ne comportant que des 1 : `ones(m,n)`
- matrice identité n lignes n colonnes : `eyes(n,n)` ; si A est une matrice, `eyes(A)` est une matrice identité de même taille que A .
- génération d'un vecteur ligne dont les coordonnées sont linéairement espacées :
`t=linspace(t1,t2,N)` ; génère N points linéairement espacés entre $t1$ et $t2$ (attention au point-virgule s'il est omis, toutes les valeurs défilent à l'écran)
`t=[t1:delta:t2]` ; qui est équivalent à `t=[t1, t1+delta, t1+2*delta, ..., t2]` ;
- génération d'un vecteur ligne dont les coordonnées sont logarithmiquement espacées :
`om=logspace(d1,d2,N)` ; génère N points logarithmiquement espacés entre 10^{d1} et 10^{d2}
- les puissances de 10 s'écrivent indifféremment `10^5`, `1d5`, `1e5`.
- `log` est le logarithme népérien et `log10` le logarithme décimal.
- une fonction qui peut être utile : `find`
 exemple : `k=find(y<1)` donne tous les indices k tq $y(k) < 1$.

Figures

Les possibilités pour les fenêtres graphiques sont nombreuses. Nous nous limiterons ici à celles qui seront utiles dans ces TP, se référer à l'aide pour de plus amples renseignements.

- tracé de courbes : `plot2d`
 Soient y et t deux vecteurs de **même** dimension (cf `size`), `plot2d(t,y)` trace y (ordonnée) en fonction de t (abscisse). Attention `plot2d(y)` trace également une courbe (les composantes de y en fonction de leur indice).
 - pour mettre un titre à une fenêtre graphique : `xtitle('ô la jolie courbe')`
 - pour superposer deux courbes (donc ayant même abscisse t) comportant le même nombre de points : `plot2d(t,[y1;y2])'`
 - pour associer une légende à chacune des courbes : `plot2d(t,[y1;y2]','leg='courbe y1 @ courbe y2')` cette légende est alors affichée en dessous de la courbe. Pour une légende plus sophistiquée, utiliser `legend`.
 - pour choisir les échelles logarithmique ('l') ou linéaire ('n') : `plot2d(t,y,logflag="ln")` donne par exemple une échelle semi-logarithmique.

- pour obtenir les coordonnées de n points sur le graphe considéré : `[ti,yi]=locate(n)` puis cliquer gauche en n points de la courbe ou utiliser le mode `datatip` (4^e icône de la figure).
- pour "effacer" une courbe `clf()`
- pour ouvrir la fenêtre graphique $n^{\circ}n$: `scf(n)`
- pour fermer une fenêtre graphique : `xdel()`
pour fermer toutes les fenêtres graphiques : `xdel(winsid())`.
- si on désire tracer plusieurs courbes dans des repères différents mais sur la même fenêtre graphique : `subplot(mnp)` m , n et p étant respectivement le nombre de lignes, de colonnes, et le numéro de sous-division de la fenêtre, par exemple :



- si on désire exporter la figure au format pdf : `xs2pdf(gcf(),'courbe1.pdf')` ou utiliser le menu Scilab : Exporter vers ...

Définition d'un polynôme

Scilab est capable de gérer les polynômes (il existe un 'type' polynôme (voir la commande `typeof`). Une façon de procéder pour définir un polynôme est la suivante :

- choix de la variable du polynôme : `p=poly(0,'p');`
- définition du polynôme : `num= 2*p^2+3*p+5`

On peut alors définir simplement des produits et des fractions de polynômes : $(1-p)/(1+p)$ ou encore `num/(1+p)^2` si `num` a été préalablement défini. On peut également définir un polynôme à l'aide de ses racines (cf l'aide).

Si g est une fraction rationnelle `g('num')` et `g('den')` donnent respectivement les numérateurs et dénominateurs de g (ceci vaut aussi pour une fonction de transfert).

Fonction de transfert et réponses

- définition d'une fonction de transfert : `syslin`
on définit au préalable p comme étant la variable polynomiale, puis, par exemple :
`G=syslin('c',1/(p+3))` ('c' signifie qu'il s'agit d'une fonction de transfert d'un système à temps continu).
- produit de fonctions de transfert : `*`
Si $G_1(p)$ et $G_2(p)$ sont deux fonctions de transfert définies à l'aide de la commande `syslin` alors `G=G1*G2` est la fonction de transfert correspondant au produit des deux.
- réponses temporelles : `csim`
pour toutes ces réponses, il faut auparavant créer un vecteur temps (à l'aide de `linspace` par exemple).
— réponse indicielle : `y=csim('step',t,G);` calcule la réponse indicielle de la fonction de transfert $G(p)$ calculée aux instants déterminés par le vecteur t . On peut ensuite tracer cette réponse à l'aide de `plot2d(t,y)`.

- réponse impulsionnelle : `y=csim('impuls',t,G)`; calcule la réponse impulsionnelle de la fonction de transfert $G(p)$ calculée aux instants déterminés par le vecteur t .
- réponse à une entrée quelconque définie par un vecteur : soit *entree* le vecteur contenant les valeurs du signal d'entrée correspondant aux instants définis par t , alors la réponse temporelle correspondant à ce signal est : `y=csim(entree,t,G)`;
- réponse à une entrée quelconque définie à l'aide d'une fonction : soit *entreef* une fonction : (définie par exemple par `deff('u=entreef(t)','u=2*sin(om1*t)')`) alors la réponse temporelle correspondante est `y=csim(entreef,t,G)`.
- réponse en fréquences : **repfreq**
Soient $f_{min} = 10^{d1}$ et $f_{max} = 10^{d2}$ les fréquences (exprimées en Hz) minimales et maximales entre lesquelles on veut étudier la réponse en fréquences du système G . On définit le vecteur des fréquences *frq* à l'aide de `frq=logspace(d1,d2)`. On calcule alors $G(j2\pi frq(k))$ à l'aide de `repf=repfreq(G,frq)` (*repf* est alors un vecteur complexe de même dimension que le vecteur *frq*).
Si l'on veut seulement connaître la valeur de la transmittance isochrone en une pulsation précise *om0* : `repfreq(G,om0/(2*pi))` ou `horner(G,%i*om0)`
Si l'on désire calculer le module en décibel *db* et l'argument en degré *phi* de la réponse en fréquences on utilise la commande : `[db,phi]=dbphi(rep f)`;
- Tracé des lieux de Bode (module et phase) : **bode**
En utilisant les mêmes notations qu'au paragraphe précédent : `bode(G[,fmin,fmax])` (ceci est rapide mais présente l'inconvénient de ne pas donner accès aux valeurs des modules et phases) ou `bode(frq,db,phi [,comments])` ou `bode(frq, rep f [,comments])`.
Si on désire la pulsation en abscisse, utiliser le paramètre 'rad' :
`bode(G,fmin/(2*pi),fmax/(2*pi),'rad')`
- Tracé du lieu de Bode en amplitude : **gainplot**
La syntaxe est identique à celle de **bode**.
- Fréquence de résonance : **freson**

Divers

- afin de ne pas exécuter la totalité du programme (choix et affichage) :
`reponse = input('rentrez n pour faire la partie n°n');`
`disp('on commence la partie '+string(reponse));`
`if reponse==1 then`
`taper les commandes de la partie 1`
`elseif reponse==2 then`
`taper les commandes de la partie 2`
`end`