Exercices avec Scilab Version 1.0

Mai 2001

Les logiciels de simulation en automatique. Quelques exercices d'automatique avec le logiciel Scilab.

Lucien Povy EUDIL 59655 VILLENEUVE D'ASCQ CEDEX

Lucien.Povy@eudil.fr

Résumé :

Regard sur quelques logiciels de simulation en automatique analogique, exercices d'automatique avec le logiciel Scilab.

Mots clés : Automatique. Systèmes linéaires. Logiciels de simulation. Scilab.

1 Les logiciels de simulation et l'automatique

Comme je vous l'ai signalé en cours, il existe à ma connaissance, trois logiciels de simulation permettant d'illustrer les enseignements d'automatique, de traitement du signal et même d'analyse numérique. Ces logiciels se nomment *Matlab*, *Octave*, *Scilab*. Ces trois logiciels s'installent sur des plateformes diverses à savoir *WIN??*, *MAC*, divers *UNIX*, dont *LINUX*.

1.1 Matlab

On trouvera des renseignements actualisés concernant *Matlab* sur le site internet : http://www.mathwork.com.

1.1.1 Avantages

Le plus connu, utilisé en écoles d'ingénieurs, dans les iut, les universités, dans de nombreux bureaux d'études industriels, c'est un logiciel complet, qui possède de nombreuses boîtes à outils, il est de plus interfaçable avec le logiciel de mathématique symbolique *Maple*.

1.1.2 Inconvénients

Il est très coûteux, car protégé par un copyright commercial, chaque boîte à outils est payante, toute mise à jour l'est aussi et ce logiciel doit posséder une licence par poste de travail ou par groupe de postes : *licence classroom.*

Vous n'avez pas accès au logiciel source bien que de nombreux programmes soient directement issus de bibliothèques mathématiques libres, comme par exemple la bibliothèque *NETLIB*, en particulier des programmes écrits en languages *fortran* ou en *c*, programmes que l'on obtient librement et gratuitement en se procurant le cdrom de cette bibliothèque : http://www.netlib.org.

1.2 Octave

Logiciel de simulation mathématique, d'algèbre linéaire, de simulation de fonctions, de simulation d'équations différentielles ..., il est accessible sur différentes plateformes, c'est un logiciel ouvert écrit en *c++*, il est couvert par une licence libre, licence *GPL*, et est gratuit. Le site internet d'*Octave* se nomme : http://www.che.wisc.edu.

1.2.1 Avantages

Logiciel non commercial, avec une licence *GPL* qui donne automatiquement accès au code source, vous pouvez l'essayer à l'*EUDIL* car il est installé sur un des serveurs.

1.2.2 Inconvénients

Encore en cours de développement, pas trop instable, je crois, c'est un logiciel en devenir. Il lui manque, malgré tout, un outil graphique de construction de systèmes complexes comme *simulink* qui existe dans *Matlab* ou *scicos* pour le logiciel *Scilab* ainsi qu'un outil intégré de tracé de courbes : vous devez utiliser le logiciel *gnuplot*. Autre inconvénient de taille, à la date du 1er mai 2001, il ne semble plus être développé.

1.3 Scilab

C'est un logiciel écrit par des français travaillant à l'*INRIA Institut National de Recherche en Informatique et Automatique* ; ce logiciel a une licence proche de la licence *GPL*, il est libre et gratuit, vous pouvez obtenir de l'aide par le réseau sur le site de l'*INRIA*, http://www-rocq.inria.fr/scilab. De même sur ce site vous trouverez un lien vers le site ftp de l'*INRIA*, où vous pourrez télécharger le logiciel, soit les sources, soit différents binaires pour diverses plateformes informatiques. La dernière version stable est maintenant la version *2.6*. Il existe aussi une version de développement mise à la disposition des curieux et téméraires.

1.3.1 Avantages

Logiciel complet, très proche de *Matlab* quant à sa syntaxe, voir les exercices proposés : comparaisons des instructions et de la progammation. C'est un produit que nous devons à un institut de recherche Français l'*INRIA*. Il possède de nombreuses boîtes à outils dont *scicos* analogue à la boîte à outils *simulink* de *Matlab*, son développement se poursuit.

1.3.2 Inconvénients

C'est un logiciel libre et gratuit, il ne vaut donc rien, le contribuable devant bien entendu débourser pour des logiciels complètement fermés ... et être à la merci des vendeurs. Enfin son *look* n'est pas terrible, il utilise une vieille bibliothèque graphique : il lui manque un *look* à la *WIN??*, bien que la dernière version de *Scilab* utilise maintenant des boutons en trois dimensions.

2 Mise en place de Scilab sur un PC–LINUX

Vous devez rapatrier le source de *Scilab* sur le site de l'*INRIA*, ftp://ftp.inria.fr/INRIA/Projects/Meta2/Scilab/distributions, l'archive se nomme **`scilab-2.6-src.tar.gz`** ; ou allez dans le répertoire **`unstable`** pour récupérer la dernière version, soit **`scilab-src-unstable-date.tar.gz`**. Dans un répertoire temporaire, **`/usr/temp`** par exemple, décompressez le fichier *Scilab* par la commande : `tar xvzf scilab-2.6.tar.gz`, la commande *Linux* **`tar`** va vous créer une archive **`scilab-2.6`**, puis déplacez cette archive dans le répertoire **`/usr/local`** ou dans le répertoire **`/opt`**. Vous pouvez maintenant effacer le fichier compressé que vous avez rapatrié. Un conseil : lisez le fichier **`README`** situé dans le répertoire principal du logiciel.

2.1 Les logiciels indispensables à la compilation et au fonctionnement de Scilab

Afin de compiler *Scilab* sur une station de type *Linux*, vous devez disposer d'un compilateur *c* et d'un compilateur *fortran77*, du logiciel *Xaw3d*, de l'interface graphique *Xwindow* pour la dernière version de *Scilab*, de *tcl et tk* nouvelle version, éventuellement du logiciel *pvm*. Faites attention avec la version de *Linux SUSE 6.2* et peut être avec d'autres distributions de *Linux*, de ne pas installer les deux compilateurs *c* à savoir *gcc* et *egcs* comme vous pouvez le faire avec cette distribution. Installez *gcc* avec *g77* qui est le compilateur fortran correspondant. Faites aussi attention à l'endroit où sont installées les bibliothèques *tcl et tk*, voir plus loin les options de configuration de *Scilab*.

2.2 Compilation du logiciel, installation

2.2.1 Compilation

Pour compiler *Scilab*, placez vous dans le répertoire racine de *Scilab*, par exemple **`/usr/local/scilab-2.6`** et exécutez dans un terminal *X* la commande : `./configure`, vous pouvez rajouter des options à **`configure`**, pour cela voyez le fichier **`README`** situé dans le répertoire racine de *Scilab*. Pour démarrer la compilation vous devez exécuter la commande **`make all`**, et après quelques dizaines de minutes, et quelques *warning !!*, vous devriez avoir un exécutable situé dans le répertoire **`SCI/bin`**, pour démarrer le logiciel de ce répertoire cliquez sur le script **`scilab`**, mais avant cela ouvrez avec un éditeur, *kedit* par exemple, ce fichier et corrigez la ligne huit afin de définir votre imprimante : **`PRINTERS="lp:nom_de_mon_imprimante"`**. Voici un exemple d'instructions exécutées dans un terminal permettant de configurer et compiler le logiciel.

```
cd /usr/local/scilab-2.6
./configure --with-xaw3d --with-tk --without-pvm
make all
cd tests
make tests
```

2.2.2 Installation

Quant à l'installation elle consiste à dire à *Linux* par la variable d'environnement **`PATH`** que le système doit rechercher le script de Scilab à savoir **`scilab`** dans le répertoire **`/usr/local/scilab-2.6/bin`** ou dans le répertoire

/opt/scilab−2.6 ; vous pouvez aussi faire lien sur votre bureau pointant vers ce script, ou créer un lien dans **/usr/bin** pointant vers ce script, par la commande, `ln -s /usr/local/scilab-2.6/bin/scilab /usr/bin/scilab`, en effet le répertoire **usr/bin** est toujours sur le chemin de recherche de *Linux*.

2.2.3 Quelles bogues connus, comment y remédier, quelques améliorations

Je ne parlerais ici que de quelques petits bogues ou imperfections que j'ai repéré dans certains programmes situés dans des sous répertoires du répertoire **SCI/macros**, ces bogues, ces imperfections, concernent des fonctions utilisées en automatique classique : vous trouverez dans le corps du document les remarques à ce sujet.

Vous trouverez de même dans un nouveau répertoire nommé **autoelem** quelques macros permettant de traiter les réponses fréquentielles des systèmes continus linéaires avec ou sans retard pur, avec des fonctions amenant, en plus du système un vecteur gain et un vecteur déphasage fonctions de la fréquence, que vous pourrez ajouter au répertoire **SCI/macros** (Voir Annexe B).

3 Exercices d'automatique avec Scilab : analyse d'un système

Ces exercices ont pour but de définir les systèmes linéaires continus et de mettre en évidence quelques propriétés de ces systèmes.

3.1 Démarrage de Scilab

Comme je l'ai signalé précédemment, après avoir installé *Scilab* dans son répertoire **SCI**, vous devez mettre le script **scilab** dans votre chemin de recherche, ou faire un lien symbolique entre **SCI/bin/scilab** et, par exemple, **/usr/bin/scilab** ou **/usr/bin/X11/scilab**, car les répertoires **/usr/bin** et **/usr/bin/X11** sont toujours sur votre chemin de recherche. Ainsi, en frappant dans un terminal *X*, **scilab**, une fenêtre *Scilab* apparaît. Dans cette fenêtre vous pouvez maintenant faire exécuter ligne après ligne un programme de simulation : ce que je viens de proposer s'applique bien entendu à un système de type *Unix*.

3.2 Les deux manières d'exécuter un programme Scilab

La façon la plus évidente de démarrer une session *Scilab* consiste, dans une fenêtre *Scilab*, après le prompt de *Scilab*, de frapper les lignes de commande les unes après les autres, en frappant la touche **entrée** après chaque ligne de programme. Ainsi la ligne correspondante est exécutée et affichée ; si l'on refuse l'affichage des résultats il suffit de mettre un **;** en fin de ligne de commande, ceci est utile quand on fait un calcul donnant de très nombreux résultats et que l'on ne souhaite pas remplir son écran d'une multitude de valeurs numériques.

On peut aussi par un **copier-coller**, introduire des lignes de programme dans une fenêtre *Scilab*. Cette méthode est très utile quand on veut faire exécuter les programmes exemples donnés dans les pages de manuel : bouton **help** de la fenêtre principale de *Scilab*.

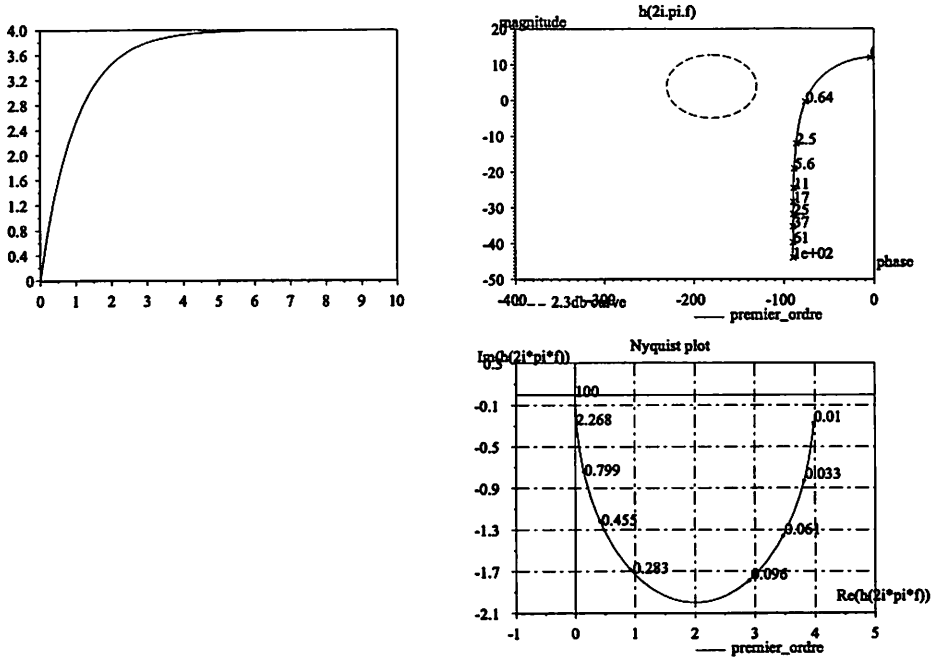
C'est en utilisant **le presse papier X** que les programmes *Scilab* apparaissent dans ce texte : vous pouvez par ce procédé, échanger dans les deux sens, des lignes de programme entre un éditeur, ou un formateur de documents, ici *Thot*, et une fenêtre *Scilab*.

Voici un exemple, réalisé par un étudiant, de programme permettant d'illustrer ceci.

```
tau=input('donner une valeur à tau :')
k=input('donner une valeur à K :')
s=%s;
den=1+tau*s;
num=k;
fr=num/den;
sys=syslin('c',fr)
temps=linspace(0,10,51);
h=csim('step',temps,sys);
xbasc()
// reponse indicielle unitaire
xsetech([0.,0.,0.5,0.5]);
plot2d(temps',h')
// black
xsetech([0.5,0.,0.5,0.5]);
black(sys,.01,100,'premier_ordre')
// nyquist
xsetech([0.5,0.5,0.5,0.5]);
nyquist(sys,.01,100,'premier_ordre')
```

Dans ce programme, par l'instruction **variable=input('??')** *Scilab* attend que l'utilisateur donne une valeur à la variable. Ce fichier texte attaché à un courrier électronique, a été exécuté ligne après ligne, dans une fenêtre *Scilab*.

Vous remarquerez l'instruction **xsetech** qui permet ici, dans une même fenêtre graphique d'afficher trois dessins : le premier situé en position (0, 0), coin haut gauche du graphe qui va être tracé, sera dans un cadre de dimensions (0.5, 0.5) : c'est la réponse indicielle. Le second graphique, le lieu de *Black* du système, est situé, pour son coin haut gauche, en position (0.5, 0) et a pour dimensions (0.5, 0.5). Quant au dernier graphique, je souhaite le mettre dans le dernier quadrant de mon cadre à savoir, en position : abscisse, 0.5 ; ordonnée , 0.5.



3.3 Définir un polynôme par ses racines, ses coefficients, valeur numérique d'un polynôme: nous faisons des mathématiques

Avant de définir un système linéaire, nous allons faire quelques exercices sur les polynômes et les fractions rationnelles.

Tout d'abord nous allons définir un polynôme et voir quelques instructions se rapportant aux polynômes.

Définir un polynôme par ses racines :

```
-->s=poly(0,'s')
 s  =
    s
```

Pour introduire dans *Scilab* une variable, ici **s**, nous devons la définir comme un polynôme du premier degré ayant pour racine **s=0**. Cette instruction est nécessaire pour introduire la variable **s**.

Une autre façon, plus rapide, d'introduire la variable utilisée dans la transformée de Laplace ou dans la transformée en z est de faire :

```
-->s=%s
 s  =
    s
-->z=%z
 z  =
    z
```

Par cette méthode, on utilise des variables réservées, que l'on ne peut détruire, variables précédées du signe **%**.

```
-->num=poly([-1,-2,-3],"s",'r')
 num  =
                    2   3
    6 + 11s + 6s + s
```

Voici un descriptif des variables réservées.

Le logiciel offre des constantes spéciales **%i**, **%pi**, **%e**, **%eps**. La constante **%i** est le nombre complexe pur $\sqrt{-1}$, **%pi** est le nombre $\pi = 3{,}1415927\ldots$, **%e** est la constante d'Euler $e = 2{,}7182818\ldots$, **%eps** représente la précision de la machine : ici $4{,}41E{-}16$. Quant aux symboles **%inf** et **%nan** ils représentent respectivement *l'infini* et *NotANumber*.

Enfin Scilab possède des booléens notés **%t** (ou **%T**), **%f** (ou **%F**) pour caractériser les deux symboles logiques **1** et **0** (true, false en Anglais).

Ces variables sont dites réservées car elles sont protégées, ne peuvent être détruites ni sauvées (par la commande **save**). Vous pouvez créer vos propres variables réservées par la commande **predef** : si vous avez dans votre propre répertoire (sous *Unix*) un fichier script **.scilab** vous pouvez mettre ces variables spéciales dans ce fichier.

Contairement à *Matlab*, il ne faut pas utiliser le caractère `%` pour faire un commentaire dans votre programme, les lignes de commentaires commencent par les deux caractères `//`.

Les racines de ce polynôme sont : `[-1,-2,-3]` c'est un vecteur ligne, on peut si l'on veut, rajouter `'r'` ou `"r"` comme indicatif pour rappeler que l'on définit un polynôme par ses racines : ce drapeau est facultatif.
Nous avons défini ici un vecteur ligne par la syntaxe :
`vecteurligne=[?,?,...,?]`, on peut remplacer la virgule par un espace.

```
-->num=poly([-1,-2,-3],"s")
 num  =
                    2    3
    6 + 11s + 6s + s
```

On obtient le même résultat. Profitons pour faire le calcul des racines de ce polynôme.

Attention : Quand on définit un polynôme par ses racines avec l'instruction précédente, on réalise l'opération : $(s-s_1)\,(s-s_2)\ldots$où s_1, $s_2\ldots$, sont les racines du polynôme.

```
-->racines=roots(num)
 racines  =

! - 1. !
! - 2. !
! - 3. !
```

Faire attention ici, *Scilab* retourne les racines sous forme d'un vecteur colonne. Un vecteur colonne est défini par la syntaxe :
`vecteurcolonne=[?;?;...;?]`

```
-->racines_en_ligne=racines'
 racines_en_ligne  =
! - 1.  - 2.  - 3. !
```

La mise sous forme transposée d'un vecteur se fait par le signe `'` ceci est aussi valable pour une matrice .

```
-->num=poly(racines_en_ligne,"s')
 num  =
                    2    3
    6 + 11s + 6s + s
```

Scilab est suffisamment futé pour reconstruire le polynôme demandé.

On peut aussi, tout simplement, définir un polynôme par son expression.

```
-->s=poly(0,'s")
 s  =
    s
-->num=6+11*s+6*s*s+s^3
 num  =
                  2   3
    6 + 11s + 6s + s
-->num1=(s+1)*(s+2)*(s+3)
 num1  =
                  2   3
    6 + 11s + 6s + s
```

Cette façon de faire, très naturelle, conduit à des résultats identiques.

```
-->A=[1,2,7;3 4,8;5 6,9]
 A  =
!   1.    2.    7. !
!   3.    4.    8. !
!   5.    6.    9. !
```

On définit une matrice, **A**, ligne par ligne, le point virgule sépare les trois blocs (vecteurs lignes), chaque élément de ligne est séparé de son suivant par un espace ou une virgule (on peut mélanger les deux), préférer la virgule à l'espace afin de ne pas faire d'erreurs de syntaxe : en fait une matrice est un vecteur colonne de vecteurs lignes (de même dimension).

Polynôme caractéristique de **A** :

```
-->nA=poly(A,'s")
 nA  =
                             2   3
  - 4.602E-16 - 40s - 14s + s
-->ncA=clean(nA)
 ncA  =
               2   3
  - 40s - 14s + s
```

Un coefficient est très petit, en dix moins seize, on lui affecte la valeur zéro, en utilisant l'instruction **clean**.

```
-->roots(nA)
 ans  =
! - 1.150E-17 !
```

```
!  - 2.4339811  !
!    16.433981  !
```

En faisant **roots** du polynôme caractéristique d'une matrice, on calcule les valeurs propres de celle–ci, et l'on obtient un vecteur colonne.

```
-->racinesncA=roots(ncA)
 racinesncA  =

!    0          !
!  - 2.4339811  !
!    16.433981  !

-->racinesnA=clean(roots(nA))
 racinesnA  =

!    0          !
!  - 2.4339811  !
!    16.433981  !
```

C'était un petit jeu !!!

Par l'instruction **spec**, on obtient, sans passer par le polynôme caractéristique de la matrice, les valeurs propres de celle ci.

```
-->valpropA=clean(spec(A))
 valpropA  =

!    16.433981  !
!  - 2.4339811  !
!    0          !
```

J'ai profité, comme je savais qu'une des valeurs propres était très petite, pour lui affecter la valeur zéro.

On va utiliser maintenant l'instruction **horner**, elle permet deux choses : soit de calculer la valeur numérique d'un polynôme pour une valeur de la variable, soit de transformer ce polynôme en changeant l'argument **s**, en un polynôme, en un rapport de polynômes, cette instruction s'applique aussi bien à des polynômes qu'à des fractions rationnelles : on verra plus loin la définition des fractions rationelles.

```
-->valpi=horner(nA,%i)
 valpi  =
    14. - 41.i
```

Dans cet exemple on calcule la valeur du polynôme pour **s=j**, (c'est le complexe pur), noté **%i**, constante réservée (voir début du la section).

```
-->valunsurs=horner(nA,1/s);
-->clean(valunsurs)
 ans  =
                  2
    1 - 14s - 40s
    -------------
           3
          s
-->valhomo=horner(nA,(1-s)/(1+s))
 valhomo  =
                    2     3
  - 53 - 29s + 57s + 25s
     --------------------
                  2   3
      1 + 3s + 3s +
s-->valquel=horner(nA,(1+s+s*s)/(2+s+3*s^2))
 valquel  =
                      2       3       4       5       6
  - 187 - 387s - 988s - 1095s - 1296s - 695s - 401s
     ------------------------------------------------
                     2     3     4     5     6
          8 + 12s + 42s + 37s + 63s + 27s + 27s
```

Très intéressant les trois dernières commandes avec *Scilab* : c'est déjà du calcul symbolique. On reviendra sur les fractions rationnelles (rapport de deux polynômes). Il existe une autre instruction permettant de calculer la valeur numérique d'un polynôme ou d'une fraction rationnelle : c'est l'instruction **freq** qui permet aussi de donner la valeur d'une expression, pour une valeur de la variable, ou pour un vecteur.

Remarque : L'instruction **horner**, peut aussi (depuis la version 2.5 de Scilab ?), donner une matrice d'expressions (nombres, polynômes, fractions rationnelles), un exemple :

```
-->s=%s;
-->H=[1+s;(1+s+s*s);1/(2+3*s+s*s)];
```

On définit un vecteur colonne constitué de polynômes et d'une fraction rationnelle.

```
-->f=[1+%i,(1-s)/(1+s),1/s,4];
```

On construit un vecteur ligne constitué de nombres et fractions rationnelles de la variable **s**, le résultat est une matrice de fractions rationnelles (**type 16**). Voir l'instruction **type** dans le manuel.

```
-->Mat=horner(H,f)
 Mat  =

!                                                  !
!  (2+i)        2          1 + s          5       !
!  ------     ------       ------         --      !
!                                                  !
!    1        1 + s          s            1       !
!                      2              2            !
!  (2+i*3)     3 + s      1 + s + s      21       !
!  --------  -----------  ----------     ---      !
!                        2       2                 !
!     1      1 + 2s + s       s          1       !
!                       2        2                !
!     1      1 + 2s + s        s         1       !
!  --------  -----------  ------------   ---      !
!                                    2             !
!  (5+i*5)     6 + 2s     1 + 3s + 2s    30       !
-->type(Mat)
 ans  =
    16.
```

Admirez la concision du résultat !

Vous noterez que *Scilab* ordonne un polynôme dans le sens des puissances croissantes.

Définir un polynôme par ses coefficients :

```
-->den=poly([1,3  2.5,1],'s",'c')
 den  =
                    2   3
    1 + 3s + 2.5s + s
-->coe=[1 3 2.5 1];ceprim=coe'
 ceprim  =
!   1.  !
!   3.  !
!   2.5 !
!   1.  !
-->dem=poly(ceprim,'s','c')
 dem  =
```

```
                    2   3
    1 + 3s + 2.5s + s
```

A l'avant dernière commande j'ai séparé deux instructions par un point virgule **;** le vecteur ligne ne s'affiche pas, seul le vecteur colonne **ceprim** est affiché (comme précédemment le vecteur coefficient peut être une ligne ou une colonne). Encore quelques instructions qui mettent en oeuvre des polynômes.

```
-->s=poly(0,"s")
 s  =
    s
-->num=poly([8 5 4 7 3 9 6 2 1 4],"s",'c")
 num  =
              2  3  4  5  6  7 8  9
     8+5s+4s+7s+3s+9s+6s+2s+s+4s
-->coe=coeff(num)
 coe  =
! 8.   5.   4.   7.   3.   9.   6.   2.   1.   4.!
-->long=length(coe)
 long  =
     10.
-->r1=coe(long:-2:1)
 r1  =
!   4.    2.    9.    7.    5. !
-->r2=coe(long-1:-2:1)

 r2  =
!   1.    6.    3.    4.    8. !
-->r=[r1;r2]
 r  =
!   4.    2.    9.    7.    5. !
!   1.    6.    3.    4.    8. !
```

Dans cette session, j'ai cherché à faire une table, qui est ici proche de la table de Routh, d'un système qui aurait **num** comme polynôme caractéristique ; si l'on voulait faire vraiment la table de Routh il faudrait tester les dimensions de **r1** et de **r2** et complèter le vecteur **r2** à droite, avec un zéro si nécessaire.

Quelques commentaires sur ce petit programme.

On définit un polynôme de la variable **s**, par ses coefficients (drapeau **'c'**, **'c"** ou **"c"** dans l'instruction **num=poly(...)**), puis on recherche les coefficients de ce polynôme, par l'instruction **coe=coeff(num)**, et enfin on cherche la dimension **length**, de ce vecteur coefficient.

Enfin on va construire deux vecteurs lignes, **r1**, **r2**, le premier, par l'instruction **coe(long:-2:1)**, va être constitué d'un vecteur ligne, en prenant les coefficients de deux en deux depuis le dernier (c'est le coefficient de degré le plus élevé du polynôme de départ). On pouvait tout aussi bien, sans calculer la longueur du vecteur coefficient, sortir le même résultat par l'instruction : **r1=coe($:-2:1)**. Le signe **$** représente le dernier élément du vecteur ligne **coe**. Quant au second vecteur, c'est aussi un vecteur ligne, dont les coefficients sont ceux du polynôme de départ, pris de deux en deux, depuis l'avant dernier coefficient. Avec ces deux vecteurs lignes, on construit une matrice **r**.

Voici quelques instructions relatives au polynôme étudié.

```
-->derivat(num)
 ans  =
                    2
   11 + 12s + 3s
```

Cette dernière instruction, calcule la dérivée, par rapport à la variable **s**, du polynôme considéré, l'instruction dérivée s'applique aussi bien à des polynômes qu'à des fractions rationnelles.

```
-->invr(num)
  ans  =


                1
 ---------------
                  2   3
 6 + 11s + 6s + s
```

On pouvait aussi écrire :

```
-->den=1/num
  den  =
                1
 ---------------
                  2   3
 6 + 11s + 6s + s
```

Dans les instructions qui vont suivre on cherche à déterminer les facteurs constituants le polynôme.

```
-->num1=poly([1,2,3,4,5],'s','c')
```

```
 num1  =
                2    3    4
    1 + 2s + 3s + 4s + 5s
-->[f1]=factors(num1)
 f1  =
      f1(1)
                                2
     0.4788911 - 0.2756645s + s
      f1(2)
                                2
     0.4176315 + 1.0756645s + s
```

Scilab a trouvé deux polynômes à racines complexes conjuguées, polynômes constituants le polynôme originel et retourne une liste.

```
-->[f2]=polfact(num1)

 f2  =
!   5                             !
!                                 !
!                               2 !
!   0.4788911 - 0.2756645s + s    !
!                                 !
!                               2 !
!   0.4176315 + 1.0756645s + s    !
```

Dans cette dernière instruction *Scilab* factorise le polynôme de départ en trois facteurs : l'un de degré zéro et deux facteurs du second degré : cette instruction donne un vecteur colonne, dont le produit des éléments est le polynôme de départ.

Vous trouverez une instruction non documentée, factorisant un polynôme, pas une fraction rationnelle, en une liste comprenant le coefficient de degré le plus élevé, et en des termes du premier et/ou second degré, de telle sorte que l'on retrouve le polynôme originel en réalisant le produit de tous ces facteurs. Cette instruction ce nomme **pfactors**, cette factorisation ressemble à la factorisation d'Evans, mais ne s'applique qu'à des polynômes : voici un exemple.

```
-->num=poly([1 2 8 4 7 3 9],'s','c');
-->[res,g]=pfactors(num)
 g  =

    9.
 res  =
```

```
        res(1)
                                  2
  0.1528161 + 0.2462539s + s
        res(2)
                                  2
  0.7599724 + 1.0995144s + s
        res(3)
                                 2
  0.9567326 - 1.012435s + s

-->num1=g*res(1)*res(2)*res(3)
 num1  =
                     2    3    4    5    6
  1 + 2s + 8s + 4s + 7s + 3s + 9s
```

Jouons avec des instructions qui concernent deux polynômes.

```
-->[x]=ldiv(num1,num,6)
  x  =
!   5.    !
! - 26.    !
!   104.  !
! - 366.   !
!   1209. !
! - 3852.  !

-->[r,q]=pdiv(num1,num)
  q  =
   - 26 + 5s
  r  =
                        2
     157 + 258s + 104s
```

L'instruction **ldiv** donne ici les six premiers coefficients de la division du polynôme **num1** par le polynôme **num**. Attention on divise le polynôme $5s^4+s^3+3s^2+2s+1$ par le polynôme $s^3+6s^2+11s+6$ et on ne conserve que six coefficients dans cette division. Le résultat est donc $5s-26+104/s-366/s^2+1209/s^3-3852/s^4$.

Quant à l'autre type de division, elle nous donne le quotient et le reste de la division de **num1** par **num** : division dans le sens des puissances croissantes de la variable.

3.4 La programmation avec Scilab

Avant de continuer l'étude des fractions rationnelles, je préfère introduire quelques notions de programmation. Comme tout logiciel scientifique *Scilab* permet, par des instructions spéciales de réaliser, en ligne, ou à l'aide d'un éditeur de texte, des programmes.

3.4.1 Les fonctions, les macros

Les fonctions sont des ensembles d'instructions *Scilab* qui sont exécutées dans un nouvel environnement, isolant ainsi les variables introduites dans ses fonctions, des variables des environnements originaux. Les fonctions peuvent être créées et exécutées de différentes manières. Par exemple les fonctions peuvent passer des arguments, on peut réaliser dans des fonctions des instructions conditionnelles et des boucles, on peut les appeler récursivement. Les fonctions peuvent être des arguments d'autres fonctions, elles peuvent être éléments de listes. La façon la plus simple de créer des fonctions est d'ouvrir un éditeur de texte, *Emacs* ou *kedit* par exemple, ou alors, les fonctions peuvent être créées directement dans *Scilab* en utilisant la primitive **deff** ; un exemple :

```
-->deff('[x]=foo(y)','if y>0  then,x=1;else,x=-1;end')
-->foo(5)
 ans  =
       1.
-->foo(-3)
 ans  =
     - 1.
```

Si l'on crée une fonction à l'aide d'un éditeur de texte, on peut charger cette fonction dans l'environnement *Scilab* par l'instruction **getf('chemin_de_la_fonction/nom_de_la_fonction')**. Ceci peut aussi être fait en cliquant sur le bouton **File operation** de la fenêtre principale de *Scilab*. Cette instruction, charge la ou les fonctions depuis le fichier **nom_de_mon_fichier** et compile ses fonctions. **La première ligne du fichier contenant le programme doit impérativement commencer par l'instruction :**

```
function[y1,...,yn]=jojo(x1,...,xk)
```

Ici les arguments **yi** sont les variables de sorties tandis que les variables d'entrées sont les **xi** .

On peut, depuis la version 2.6, dans la fenêtre principale de Scilab définir un progamme permettant de réaliser la fonction souhaitée ; voici un exemple :

```
-->function  [u]=creneau(t,T,T1)
-->u=bool2s(T<=t)-bool2s(t>(T+T1));
-->endfunction
-->//La fonction est comprise entre function et
-->//endfunction
```

Cette fonction réalise une fonction créneau et utilise une instruction **`bool2s`** qui sera commentée dans un des paragraphes qui suit.

3.4.2 La programmation

L'une des plus intéressantes fonctionalités de *Scilab* réside dans la possibilité de créer et d'utiliser des fonctions. Ceci permet de développer des programmes spécialisés qui peuvent ensuite être mis dans *Scilab*, d'une manière simple et modulaire, à travers l'utilisation de bibliothèques spécialisées (voir Annexe B). Dans ce chapitre nous allons traiter les sujets suivants.

Les outils de programmation.

- La définition et l'utilisation de fonctions.
- La définition d'opérateurs pour de nouveau type de données.

Scilab possède un nombre important d'outils de programmation, comprenant les boucles, les instructions conditionnelles, la sélection, et la création de nouveaux environnements. Les tâches les plus importantes de programmation peuvent être accomplies dans le cadre des fonctions. Voici les principaux outils de programmation.

Les opérateurs de comparaison

Il existe six méthodes pour faire la comparaison entre les valeurs d'objets dans *Scilab*. Un tableau décrit ces méthodes.

caractère	**signification**
== ou =	égal à
<	plus petit que
>	plus grand que
<=	plus petit ou égal à
>=	plus grand ou égal à
<>,~=	pas égal à

Ces opérateurs de comparaison sont utilisés dans les instructions conditionnelles.

Les boucles

Deux types de boucles existent dans *Scilab* : la boucle **for** et la boucle **while**. La boucle **for** voit sa progression indexée à un vecteur d'indices. Elle se termine obligatoirement par la commande **end**. Voici quelques exemples :

```
-->x=1;for  k=1:3,x=x+k,end
   x  =
      2.
   x  =
      4.
   x  =
      7.
```

La boucle **for** peut s'itérer en utilisant les éléments d'un vecteur, ou les colonnes d'une matrice.

```
-->x=1;for  k=[6,-2,1],x=x/k,end
 x  =
       0.1666667
 x  =
    - 0.0833333
 x  =
    - 0.0833333
```

On peut aussi faire itérer la boucle **for** à l'aide des éléments d'une liste.

```
-->l=list(1,[1 2;3 4],'jojo');
-->for k=l,disp(k),end
      1.
 !    1.    2  !
 !    3.    4. !
      jojo
```

L'instruction **disp** affiche (display) les éléments de la liste.

Quant à la boucle **while**, elle exécute d'une manière répétitive, une séquence d'instructions, jusqu'au moment où une condition est satisfaite.

```
-->x=1; while x<9,x=2*x,end
  x  =
      2.
  x  =
      4.
  x  =
```

```
      8.
 x  =
      16.
```

Les boucles **for** et **while** peuvent être arrêtées par l'instruction **break**

```
-->for k=1:3; for j=1:4;if k+j>4 then break;
else disp(k);
end;end;end
 1.
 1.
 1.
 2.
 2.
 3.
```

Les instructions conditionnelles

Deux types d'instructions conditionnelles existent dans *Scilab* : l'instruction **if-then-else** et **select-case**. L'instruction **if-then-else** évalue une expression et si elle est vrai, le programme exécute les instructions comprises entre l'ordre **then** et **else** (ou l'ordre **end**). Si l'expression est fausse, le programme exécute les instructions comprises entre **else** et l'ordre **end**. L'ordre **else** n'est pas obligatoire. Quant à l'ordre **elseif** il a le sens habituel, et est un mot clef reconnu par l'interpréteur. Un exemple :

```
-->x=1
  x  =
       1.
-->if x>0 then y=-x,else,y=x,end
  y  =
     -1.
-->x=-3
  y  =
     -3.
```

L'instruction conditionnelle **select-case** compare une expression à plusieurs expressions possibles et exécute les instructions qui suivent le premier cas qui rend vrai l'expression initiale.

```
-->x=1
  x  =
     1 .
-->select x,case 1,y=2*x,case -1,y=sqrt(x),end
  y  =
     2.

-->x=-1
```

```
 x =
   -1
 y =
    i
```

Il est possible d'introduire l'ordre **else** quand aucun cas n'est vrai.

3.5 Définir une fraction rationnelle : quelques propriétés, on fait encore des mathématiques

Dans ce paragraphe nous allons décrire quelques instructions relatives aux fractions rationnelles avant d'introduire la notion de système linéaire.

```
-->s=poly(0,'s')
 s  =
     s

-->num=6+11*s+6*s*s+s*s*s
   num  =
                   2   3
     6 + 11s + 6s + s

-->den=poly([-1,-2,3],'s')
  den  =
                3
   - 6 - 7s + s
-->n1=roots(num)
  n1  =
! - 1. !
! - 2. !
! - 3. !
-->d1=roots(den)
 d1  =
! - 1. !
! - 2. !
!   3. !
-->fr=num/den
  fr  =
    3 + s
    -----
  - 3 + s
-->simp_mode(%F)
```

```
-->fr1=num/den
  fr1  =
                      2   3
          6 + 11s + 6s + s
          ----------------
                        3
            - 6 - 7s + s
```

Dans cet exercice on définit facilement une fraction rationnelle, mais attention si les deux polynômes ont des racines communes, comme dans l'exemple choisi, par défaut *Scilab* simplifie la fraction rationnelle, sauf si vous lui dites par l'instruction **simp_mode(%F)**, de ne pas le faire. Pour revenir à la situation antérieure il faut, dans votre session Scilab, introduire l'instruction **simp_mode(%T)**. **%F** et **%T** (**%f**,**%t**) sont des variables logiques, false, true en Anglais : ceux sont des variables réservées.

```
-->nu=numer(fr1)
nu  =
                    2   3
      6 + 11s + 6s + s
-->de=denom(fr1)
de  =
                     3
        - 6 - 7s + s
```

Ces dernières instructions se passent de commentaires.

Voici une session simple donnant les éléments simples d'une fraction rationnelle.

```
-->s=%s;num=poly([-1,-2,-3],'s');
-->dem=poly([-1.5,-2.5,-3.5],'s');
-->fr=num/dem
  fr=
                           2   3
              6 + 11s + 6s + s
         -------------------------
                              2   3
         13.125 + 17.75s + 7.5s + s
-->elementsimp=pfss(fr)
  elementsimp  =


           elementsimp(1)
   - 0.9375
```

```
 -------
 3.5 + s
        elementsimp(2)
- 0.1875
 -------
 1.5 + s

        elementsimp(3)
 - 0.375
 -------
 2.5 + s
        elementsimp(4)
 1.
```

Nous voyons, par l'instruction **pfss** que *Scilab* propose une liste donnant les éléments simples d'une fraction rationnelle (attention à cette instruction quand le polynôme dénominateur a des racines multiples).

On trouvera dans l'aide en ligne ou dans le manuel de référence les différentes instructions traitant des fractions rationnelles.

Dans la boîte à outils **autoelem** je propose une nouvelle fonction :
bodfact : Cette fonction est une fonction mathématique, qui factorise un polynôme, une fraction rationnelle, un système SISO, sous forme dite de *Bode*, elle retourne un vecteur constitué de :

- K : le gain statique du système, en position, en vitesse, en accélération, ..., suivant le nombre d'intégrations (de dérivations) que possède le système (le polynôme, la fraction).

- L : le nombre de dérivations (intégrations si L est négatif) de l'expression.

- TN : un vecteur colonne comprenant des polynômes du premier et/ou second degré de la forme $(1+\tau_1 s)$ et / ou $(1+\tau_{2,1}s+\tau_{2,2}s^2)$ (s étant la variable symbolique) ; de plus on a pour un polynôme de degré deux, la relation :
  $\tau^2_{2,1}-4\tau_{2,2} < 0$. Ces polynômes caractérisent le numérateur.

- TD : un vecteur analogue à TN, et caractérisant le dénominateur.

Ce n'est pas à proprement parlé une fonction utile pour étudier les fractions rationnelles qui font l'objet de cette étude, mais cette fonction est utile pour la

nouvelle fonction **dbphifr** utilisée dans l'étude des réponses fréquentielles des systèmes, cette fonction est proche de l'instruction **pfactors**.
Voici un petit exemple permettant d'illustrer cette nouvelle fonction, par la même occasion et par l'instruction :
;getd("/home/lpovy/autoelem"); je charge dans *Scilab* l'ensemble du répertoire **autoelem**.

```
-->;getd("/home/lpovy/autoelem");
-->num=poly([-1  -3,-1+%i,-1-%i],'x')
  num  =
                      2    3    4
       6 + 14x + 13x + 6x + x

-->den=poly([0,0,-3,-7,2+%i,2-%i],'x')
  den  =
           2     3     4     5   6
       105x - 34x - 14x + 6x + x
-->fr=num/den;

-->[k,l,tn,td]=bodfact(fr)
 td  =

!   1 + 0.1428571x    !
!                     !
!                   2 !
!   1 - 0.8x + 0.2x   !

 tn  =

!   1 + x           !
!                   !
!                 2 !
!   1 + x + 0.5x    !

  l  =
    -2.

  k  =
     0.0571429

-->fr1=k*poly(0,'x')^l*prod(tn,1)/(prod(td,1);

-->fr-fr1
```

```
 ans  =

         0
         -
         1
```

3.6 Définir un système linéaire: enfin un peu d'automatique

La principale commande permettant de définir des systèmes linéaires, est l'instruction **syslin** qui permet de définir un système continu ou échantillonné, par des polynômes numérateur et dénominateur, par une fraction rationnelle, ou par un quadruplet **A,B,C,D** (équations d'état). Voici une session *Scilab* permettant d'illustrer cette instruction (c'est une *liste typée*).

```
Startup execution:
loading initial environment
-->s=poly(0,'s')
s  =
  s
-->num=poly([0,-1,-2],'s')
num  =
               2   3
       2s + 3s + s
-->dem=poly([-.5,-1.5,-2.5,-3.5],'s')
dem  =

                              2    3    4
        6.5625 + 22s + 21.5s + 8s + s
-->fr=num/dem
fr  =
                   2   3
          2s + 3s + s
     ----------------------------
                        2    3    4
    6.5625 + 22s + 21.5s + 8s + s
-->sys=syslin('c',fr)
sys  =
                        2   3
                2s + 3s + s
       ----------------------------
                          2    3    4
      6.5625 + 22s + 21.5s + 8s + s
```

Cette session met bien en évidence dans l'instruction **syslin** que l'on défini un système décrit par une transmittance, rapport de deux polynômes de la variable complexe **s**, que ce système est défini en temps continu : la présence du drapeau **'c'** dans l'instruction. De la même manière, on pourrait avec le drapeau **'d'** définir un système discret. Dans l'étude de la réponse

fréquentielle, je définirai un système par ses équations d'état. La suite de la session va nous permettre de voir les graphiques utilisés par *Scilab*.

Une remarque très importante :

Quand on a défini un polynôme numérateur **num**, un polynôme dénominateur **den**, on peut définir un système linéaire continu, par l'instruction : **sl=syslin('c',num,den)**, mais attention, **num** doit être du même type que **den** (*type 2*), et surtout pas une constante (*type 1*) sinon en utilisant certaines fonctions traitants des systèmes linéaires on peut générer un bogue.

Texte de la remarque (pour les spécialistes) de mon collègue Patrick.Sarri@eudil.fr:

La fonction **routh_t(sys,k)** du répertoire **.../macros/calpol** ne fonctionne pas correctement lorsqu'on définit un système linéaire avec **syslin()** à partir de polynômes qui sont eux mêmes, non pas définis en tant que type polynôme dans *Scilab* (*type 2*), mais en tant que scalaire (en l'occurrence polynôme constant).

Exemple :

```
s=%s;
num=0.5;
den=(1+s)*(1+3*s)*(1+0.25*s);
h=syslin('c',num,den);  k=poly(0,'k');  routh_t(h,k)
```

Dans ce cas la fonction **routh_t()** renverra une matrice de polynômes en **k** erronnée.

Solution :

Il faut soit toujours utilisé la fonction **poly()** pour définir ses polynômes, soit utiliser : **h=syslin('c',num/den)**, ou **h=num/den**, mais dans ce deuxième cas bien que **h** soit défini comme une *liste typée*, il contiendra un champ de domaine vide (**[]** et non pas **'c'** ou **'d'**), ce qui pourra poser des problèmes pour l'utilisation d'autres fonctions (tel que **kpure()**) qui nécessite un champ de domaine précis.

Ce problème, lié à la fonction contenue dans le programme **routh_t**, se répercute bien entendu sur toute fonction faisant une utilisation interne de celle–ci (fonction **kpure()**).

L'erreur provient en fait d'un petit (vraiment petit) bogue lié à la fonction **coeff()**. En effet cette fonction qui prend en argument un objet de type polynôme (*type 2*), renvoit le vecteur des coefficients du polynôme dans l'ordre des puissances croissantes. Le problème se pose lorsqu'on définit implicitement un polynôme constant (exemple **p=0.5**) sans passer par la fonction **poly()**

`p=poly(0.5,'s','c')`. Ainsi défini, ce polynôme élémentaire est en fait considéré par *Scilab* comme un objet du *type 1* et non du *type 2* (polynôme), ce qui n'est pas grave en soit, puisque toute fonction travaillant sur des polynômes devraient être en mesure de traiter ces cas élémentaires en les considérant comme des polynômes, en rendant le tout transparent pour l'utilisateur. Ceci est le cas je pense en général mais pas pour la fonction `coeff()`. L'utilisation suivante de la fonction `coeff()` engendre un résultat erroné :

```
p=0.5; co=coeff(0.5,0:3)
```

qui renvoit comme résultat

```
! 0.5   0.   0.5   0. !
```

au lieu de

```
! 0.5   0.   0.    0. !
```

D'où l'erreur rencontré dans la fonction `routh_t()` qui fait largement usage de `coeff()`. Cette petite erreur se répercute sur la fonction `routh_t()` et ensuite sur la fonction `kpure()`. Voilà pourquoi un petit bogue peut devenir grand comme par effet boule de neige.

En résumé (de la part de l'auteur de ce rapport) : utilisez l'instruction `sl=syslin('c',num/den)` et non pas `sl=syslin('c',num,den)`,sauf dans des cas bien précis, à savoir définir un système linéaire, par exemple `1(s)`, par l'instruction `sys=syslin('c',poly(1,'s','c'),1)`, ou encore si on ne souhaite pas simplifier une fraction rationnelle, si elle possède des pôles et zéros identiques, et dans ce cas on fera :

```
sys=syslin('c',num,den)
```

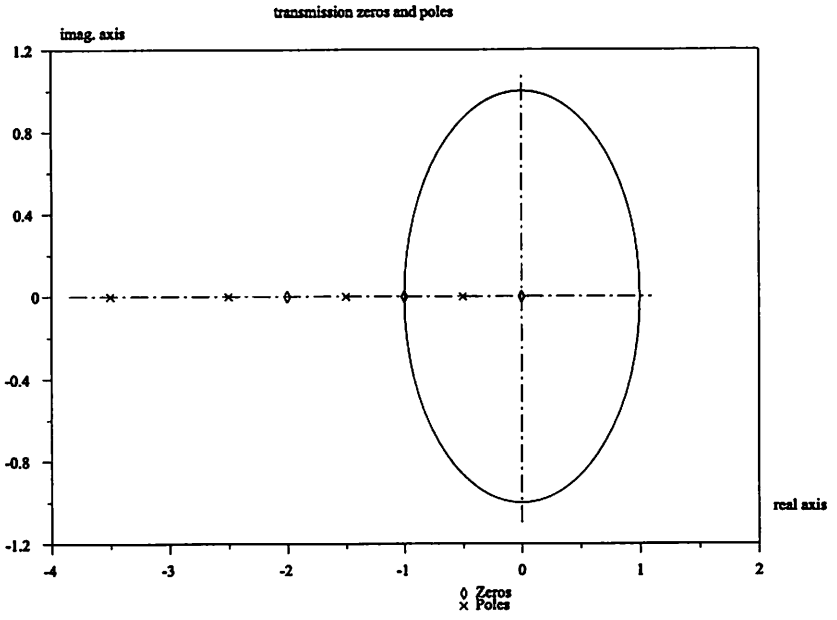
3.7 Les graphiques utilisés en automatique

Les principaux graphiques utilisés par *Scilab* permettent de placer dans le plan complexe les pôles et zéros d'un système, de visualiser les réponses temporelles et fréquentielles, ainsi que d'étudier l'évolution des pôles d'un système bouclé quand le gain de la chaîne d'action de ce système varie (*lieu d'Evans*).

3.7.1 Représentations des pôles et zéros d'un système

La suite de la session Scilab va nous permettre de visualiser les pôles et zéros du système précédemment introduit.

```
-->plzr(sys)
```



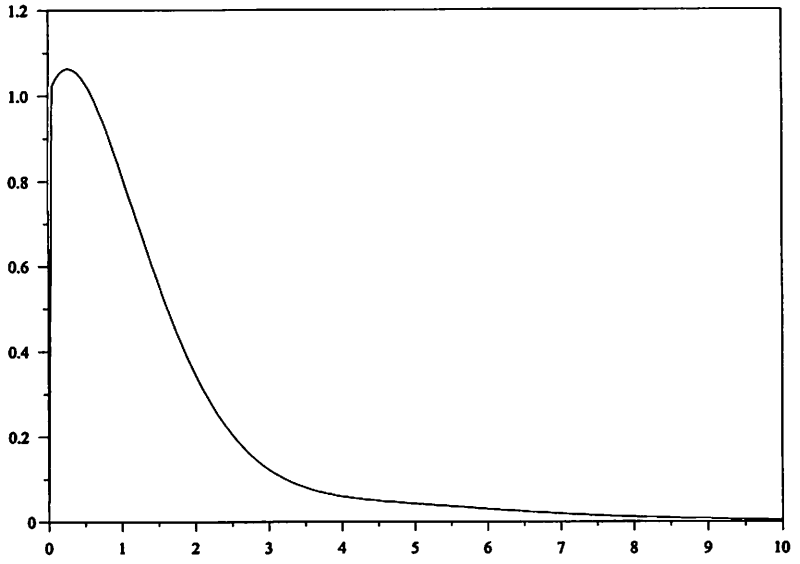
Vous remarquerez, qu'avec le programme donné par l'*INRIA*, l'instruction **`plzr`** était légèrement boguée car les deux légendes se superposaient, de même le diagramme n'était pas symétrique par rapport à l'axe réel : à vous de corriger le programme qui est situé dans le répertoire **`SCI/macros/xdess`**, il ce nomme **`plzr.sci`** ou recopier la correction proposée dans le chapitre *bogues de Scilab* dans le répertoire **`SCI/macros/xdess`** et en tant qu'administrateur, dans un terminal **`X`**, frappez, à partir de ce répertoire la commande **`make`**. Vous aurez ainsi, en redémarrant le logiciel, la macro corrigée. Cette méthode est valable pour toute correction ou modification de programmes **`.sci`**.
Le bogue a été corrigé avec la version 2.6 de *Scilab*.

3.7.2 Représentation temporelle: impulsionnelle, indicielle, à tout type de signal : instruction **csim**

La représentation temporelle d'un système se fait en utilisant la commande **csim**. Cette instruction utilise un programme *fortran* nommé **ode**, programme complexe permettant de simuler des équations différentielles, pour les curieux vous pourrez dans le manuel en ligne, découvrir les subtilités de ce programme. Pour ce faire, on doit d'abord, définir une échelle de temps et une graduation de ce vecteur temps en créant un vecteur **t**. Quand on créera ce vecteur ne pas oublier de mettre un **;** à la fin de l'instruction, sinon on se retrouve avec une quantité énorme de valeurs sur son écran.

```
Startup execution:
loading initial environment
-->s=%s;
-->n=poly([-1,-2],'s');
-->d=poly([-.5,-1+%i,-1-%i],'s');
-->fr=n/d;
-->sl=syslin('c',fr)
  sl  =
                    2
          2 + 3s + s
       ----------------
                    2   3
       1 + 3s + 2.5s + s
-->t=0:.05:10;
-->h=csim('imp',t,sl);
-->xbasc()
-->plot2d(t',h')
```

Par ces commandes on définit un système linéaire continu, de transmittance **fr**, puis une base de temps avec comme origine 0 et comme fin 10 secondes avec un pas d'échantillonnage de 0,05 seconde (c'est le vecteur ligne **t**). Enfin on simule la réponse impulsionnelle, présence du drapeau **'imp'** dans l'instruction **csim**. Les deux dernières instructions permettent d'une part, d'effacer la fenêtre graphique, puis de tracer sans aucune légende la courbe *h (t) = fonction (t)* . On pouvait tout aussi bien, dans la dernière instruction écrire : **plot(t,h)** car on ne trace qu'un simple graphique.



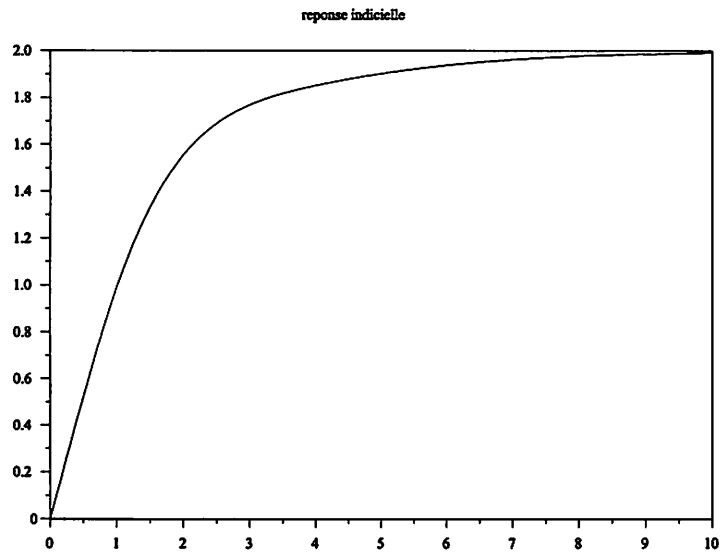
On peut par l'instruction **xtitle** rajouter un titre à la figure : par exemple dans la même session nous allons tracer la réponse indicielle.

```
-->y1=csim('step',t,sl);
-->xbasc()
-->plot(t,y1)
-->xtitle('reponse  indicielle')
```

Je voudrais faire ici une remarque sur la création d'une base de temps nécessaire à la simulation de système. Nous avons créé un vecteur temps, en partant d'une borne inférieure, en se donnant un pas, et ceci jusqu'une borne supérieure. Une instruction spéciale **linspace** peut réaliser aussi une échelle de temps : vous choisissez une borne inférieure, puis une borne supérieure, et enfin un nombre entier de graduations, (les bornes sont incluses dans ces graduations). Vous obtiendrez ainsi une échelle linéaire, d'une manière plus précise qu'en utilisant la première façon de procéder : il n'y a pas d'erreur cumulative. Quant à l'instruction **logspace** elle permet de découper une variable, suivant une échelle logarithmique. Une autre façon de procéder, consiste à créer un vecteur d'entiers, et à diviser chaque élément de ce vecteur par un nombre qui représente la quantification (le pas) demandée :

```
-->t=[0:10000]./1000;
```

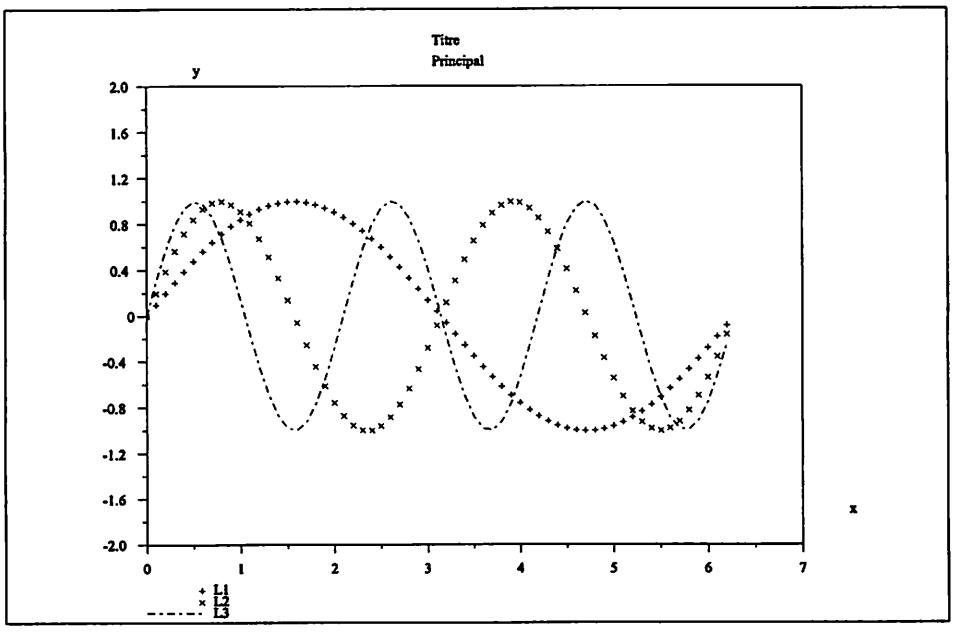
Vous utilisez ici la division élément par élément `./`, vous pouviez aussi utiliser la division normale car le diviseur est un scalaire.



3.7.3 *Exemple issu de la Demo de Scilab*

Voici se que vous allez obtenir en frappant l'instruction **`xtitle()`** dans une fenêtre *Scilab*.

```
Startup execution:
 loading initial environment
-->xtitle()
  Demo of xtitle
plot2d();
xtitle(['Titre';'Principal'],'x','y');
Demo of plot2d
x=0:0.1:2*%pi,
plot2d([x;x;x]',[sin(x);sin(2*x);sin(3*x)]',[-1,-2,3]-
,'151','L1@L2@L3',[0,-2,2*%pi,2]);
```



Dans cette démonstration, par l'instruction **plot2d()** on affiche une fenêtre graphique vide, puis par l'instruction **xtitle([...],...)** on met dans cette fenêtre le **Titre** en dessous **Principal** puis **x** sur l'axe des x et **y** sur l'axe des y. Quant à la démonstration sur la commande **plot2d** qui suit, vous remarquerez la syntaxe :

plot2d(x,y,[style,strf,leg,rect,nax])

x,y : deux matrices de même taille **[nl,nc]** **nc** donne le nombre de courbes et **nl** le nombre de points sur chaque courbe.

nc : le nombre de courbes.

nl : le nombre de points sur chaque courbe par exemple :

x=[1:10;1:10]' ,y=[sin(1:10);cos(1:10)]'

style : c'est un vecteur réel de taille **(1,nc)**. La façon de tracer la courbe numéro **j** est défini par le nombre **j**.

– Si **style[i]** est négatif la courbe est tracée en utilisant un caractère alphanumérique spécial portant le numéro d'identification de **style[i]**.

– Si **style[i]** est strictement positif une ligne pleine ou pointillée de numéro d'identification (ou de couleur) **abs(style[i])** est utilisé.

– Quand on désire tracer une courbe seulement, l'information **style**, peut avoir la dimension **(1,2)** : vecteur de composantes **[style,pos]** ou **style** est utilisé pour spécifier le type de tracé et **pos** est un entier prenant une valeur de **1** à **6**, valeur spécifiant la position à utiliser pour la légende (ceci est utile quand un utilisateur souhaite tracer de nombreuses courbes, dans la même

fenêtre, en appelant plusieurs fois la fonction **plot2d** et en mettant une légende sur chaque courbe).

strf : c'est une chaîne de caractères de longueur 3 **xyz**

x : des légendes sont affichées quand x prend la valeur 1.

y : ce caractère contrôle le cadre.

y=0 : les bornes courantes sont utilisées (données par le précédent appel).

y=1 : l'argument **rect** est utilisé pour spécifier les bornes du tracé,

rect=[xmin,ymin,xmax,ymax].

y=2 : les bornes du tracé sont calculées en utilisant les valeurs minimales et maximales de x et y.

y=3 : même chose que pour **y=1**, mais produit une échelle proportionnelle.

y=4 : même chose que pour **y=2**, mais produit une échelle proportionnelle.

y=5 : même chose que pour **y=1**, mais les bornes et la façon de graduer les axes sont différentes afin d'obtenir une meilleure graduation: ce mode est utilisé quand le bouton de zoom est activé.

y=6 : même chose que pour **y=2**, mais les bornes et la façon de graduer les axes sont différentes afin d'obtenir une meilleure graduation : ce mode est utilisé quand le bouton de zoom est activé.

z : contrôle l'affichage des informations relatives au cadre entourant le dessin.

z=1 : des axes sont tracés : le nombre de graduations peut être spécifié par l'argument **nax** : c'est un vecteur à quatre entrées, **[nx,Nx,ny,Ny] nx(ny)** représente le nombre de sous–graduations sur l'axe des x (y), **Nx(Ny)** est le nombre de graduations sur l'axe des x (y).

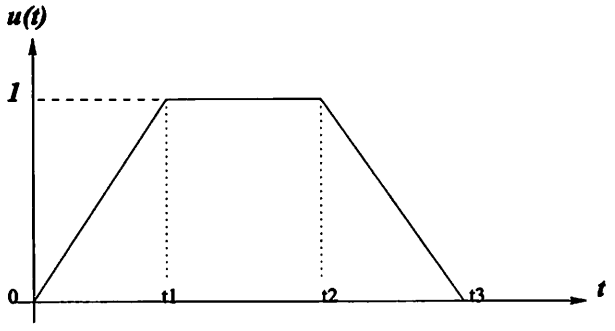
z=2 : le dessin est seulement contenu dans un cadre rectangulaire.

autre valeur on ne dessine aucun cadre autour de la courbe.

3.7.4 Création par Scilab d'un signal d'entrée

Le but de cette section est préparer un programme permettant de générer par *Scilab* un signal d'entrée pour un système donné. Je voudrais ici signaler le document mis à la disposition des francophones par le *Professeur Bruno Pinçon (Bruno.Pinçon@iecn.u–nancy.fr)*, document relatif à l'enseignement de l'analyse numérique, ce document est librement distribué. Je me suis très fortement inspiré d'un des programmes de ce document (section 3.5.5).

On cherche à réaliser un signal d'entrée *u (t)* constitué d'une rampe montante, d'un palier, puis d'une rampe descendante pour que le signal revienne à zéro.



signal d'entrée u(t)

Voici un exemple de programme optimisé ne nécessitant aucune boucle **for**.

```
function  [u]=signentre(t,t1,t2,t3)
tinft1=(t<=t1)
tinft2=(t<=t2)
inter_0_t1=bool2s(tinft1)
inter_t1_t2=bool2s(~tinft1&tinft2)
inter_t2_t3=bool2s(~tinft2&(t<=t3))
u=inter_0_t1.*(t/t1)+inter_t1_t2.*(1.)+inter_t2_t3.-
*(t3-t)/(t3-t2)
```

Un exemple de programme Scilab manipulant l'instruction csim

J'ai créé une fonction créneau, **u(t)**, valant **1** de l'instant **0** à l'instant **t1** puis revenant à **zéro**.

```
function  [u]=creneau(t,t1)
u=bool2s(t<=t1)
```

Voici un programme utilisant ce créneau comme signal d'entrée.

```
-->s=%s;
-->sl=syslin('c',1/(1+s));
-->t=linspace(0,10,101);
-->t1=.5;
```

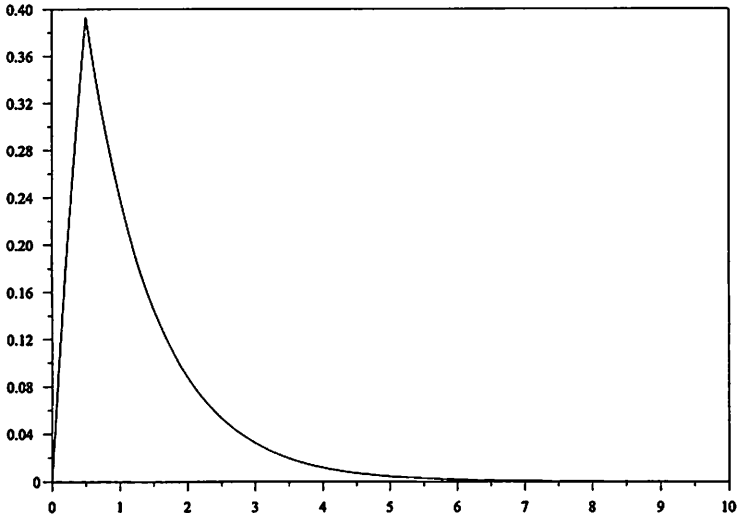
```
-->;getf("/home/lpovy/creneau.sci");
-->y=csim(creneau,t,sl);
-->plot(t,y)
```

La grandeur **t1** est le paramètre donnant la durée du créneau. Ne voulant pas le définir dans la fonction, afin de conserver toute la généralité au problème, je donne ici à **t1** dans le programme principal, (*variable globale*), la valeur **0.5**. On charge la fonction **creneau**, puis on exécute la simulation par l'instruction **csim**.

Attention : il faut dans **csim** donner le nom de la fonction d'entrée, ici **creneau** et pas le résultat, que j'ai appelé **u**. En effet **u** est de *type 1* (*scalaire*), tandis que **creneau** est de *type 13* (**function**), et **csim** accepte pour entrée, un argument de type **function** ou **list**. Voici le même programme en utilisant une *liste*.

```
-->s=%s;
sl=syslin('c',1/(1+s));
-->t=linspace(0,10,101);
-->;getf("/home/lpovy/creneau.sci");
-->ul=list(creneau,.5)
 ul  =
        ul(1)
[u]=function(t,t1)
        ul(2)
     0.5
-->y=csim(ul,t,sl);
Warning redefining function: uu
-->plot(t,y)
```

La *liste* **ul** est constituée de deux éléments : la fonction, et le paramètre **t1**.



L'utilisation de l'instruction **bool2s** permet de convertir une matrice de booléens en matrices de réels : le booléen **vrai** donnant le réel **1**, le booléen **faux** donnant le réel **0**.

De même on a utilisé l'opération **.*** qui représente la multiplication élément par élément pour réaliser la fonction demandée. Vous vérifierez que ce programme est nettement plus rapide que celui fait à partir de boucles **for** : utilisez pour cela la fonction **timer()**.

On pouvait aussi faire l'étude de la réponse de ce système du premier ordre en définissant la fonction d'entrée en ligne de commande, voici un programme exemple.

```
-->s=%s;sl=syslin('c',1/(1+s));
-->t=linspace(0,10,101);
```

```
-->deff('u=creneau(t,t1)','if  t<=t1
then,u=1;else,u=0;end')
-->t1=.5;
-->y=csim(creneau,t,sl);
-->plot(t,y)
```

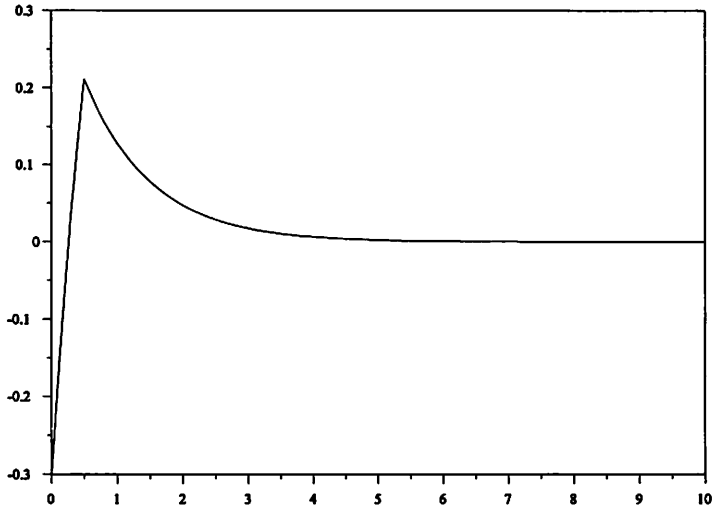
Vous remarquerez, que dans ce cas l'argument de sortie **u** est une chaîne de caractères, de même que **creneau**. On ne peut donc, tracer le signal d'entrée **u(t)**.

Conseil : Lisez attentivement le manuel sur la fonction **csim**.

Mon collègue déja cité, (*Patrick.Sarri@eudil.fr*) propose une nouvelle fonction de simulation permettant d'utiliser un vecteur **u** comme entrée, (à une valeur de **t**, on associe une composante du vecteur **u**, **t** et **u** ont même dimensions : à **t(i)**, on associe **u(i)**).

On peut aussi introduire des *conditions initiales* avec l'instruction **csim**, remplacez **y=csim...** par l'instruction suivante :

```
-->y0=-.3
-->y=csim(ul,t,sl,y0);
ou
-->y=csim(ul,t,sl,-.4);
```



3.8 Représentation fréquentielle

Cette partie de l'exposé a pour but d'étudier la réponse permanente d'un système soumis à une entrée sinusoïdale de fréquence variable.

Attention : Scilab emploi comme argument la *fréquence*, alors que les automaticiens préfèrent généralement utiliser comme argument, la *pulsation.*

Nous allons, dans ces exercices, présenter les principales instructions permettant l'étude de la réponse fréquentielle d'un système ; dans le paragraphe suivant je ferai un petit rappel sur les systèmes à déphasage minimaux et non minimaux, ainsi qu'un rappel sur la relation de *Bayard–Bode.*

```
Startup execution:
loading initial environment
-->x=poly(0,'x')
  x  =
     x
-->fra=(x+1)/(x^3+x^2+x+2)
  fra  =
            1 + x
        -------------
                 2   3
        2 + x + x + x
-->rep=freq(fra('num'),fra('den'),[1,2,3,5,10])
  rep  =
! 0.4   0.1875   0.0975610   0.0382166   0.0098921 !
```

Cette instruction **freq** a pour but de calculer, pour différentes valeurs de la variable (ici **x**), les valeurs de la fraction rationnelle **fra**, faites attention cette commande est valable pour une fraction rationnelle ou un quadruplet (A,B,C,D), modèle d'état d'un système, dans ce cas l'instruction calcule la valeur du scalaire **C*inv(x(k)*eye-A)*B+D** ou **x(k)** est la kième valeur du vecteur argument.

Un autre exemple, dans la même session *Scilab*, pour illustrer cette instruction :

```
-->rep=freq(fra('num'),fra('den'),[%i,2*%i,3,5,10*%i])
       rep  =
                column 1 to 4
    !   1. + i    - 0.35 + 0.05i    0.0975610    0.0382166 !
                column 5
    ! - 0.0101020 + 0.0000101i !
```

J'ai ici, panaché, des arguments réels et complexes, on a donc des résultats qui sont réels et complexes.

```
-->rep=freq(fra('num'),fra('den'),[1:5])
   rep  =
 !  0.4    0.1875    0.0975610    0.0581395    0.0382166 !
```

Vous remarquerez que cette instruction ressemble à l'instruction **horner** mais utilise un vecteur comme donnée d'entrée (**horner** aussi finalement). Cette instruction s'applique à une fraction rationnelle qui ne représente pas forcément un système linéaire.

Une instruction très importante pour l'étude de la réponse fréquentielle d'un système est l'instruction **repfreq**, elle s'applique aux systèmes linéaires.

```
--> A=diag([-1,-2]);B=[1;1];C=[1,1];
--> Sys=syslin('c',A,B,C)
--> frq=0:0.2:1;
--> [frq1,rep] =repfreq(Sys,frq)
 rep  =
          column 1 to 3
!   1.5     0.7462050 - 0.7124703i     0.3305398 -
0.5871213i !
          column 4 to 5
!   0.1755529 - 0.4548199i     0.1064100 - 0.3631223i !
          column 6
!   0.0707044 - 0.2997358i !
 frq1  =
!   0.    0.2    0.4    0.6    0.8    1. !
```

Nous voyons dans cet exemple que l'instruction **repfreq** renvoie deux vecteurs lignes, un vecteur fréquence ici **frel** et un vecteur complexe **rep** donnant les valeurs réelles et imaginaires du nombre complexe **sys(%i*2*%pi*frq)**.

Profitons de cet exercice pour déterminer la fonction de transfert d'un système à partir des équations d'état (attention à la simplification de pôles par des zéros).

```
-->A=diag([-1,-2,-3]);B=[1;1;1];C=[1,0,0];
-->sys=syslin('c',A,B,C);
-->gp=ss2tf(sys)
 gp  =
       1
     -----
     1 + s
-->simp_mode(%F)
-->gp=ss2tf(sys)
 gp  =
```

```
                    2
          6 + 5s + s
        ---------------
                   2   3
       6 + 11s + 6s + s
-->sys1=tf2ss(gp)
```

Cet exemple illustre le passage d'une représentation d'état à la fonction de transfert du système (instruction **ss2tf**). Le système possède deux zéros : **s=-3,s=-2** et trois pôles : **s=-1,s=-2,s=-3**. *Scilab*, par défaut simplifie l'expression, sauf si on lui impose la non simplification : présence de l'instruction **simp_mode(%F)**, l'opération inverse, passage de la fonction de transfert aux équations d'état, se fait par l'instruction : **tf2ss**.

De même on peut obtenir par l'instruction **dbphi** à partir du vecteur complexe **rep** le module, en décibels, et l'argument, en degrés, de ce vecteur.

```
-->[db,phi]=dbphi(rep)
 phi  =
         column 1 to 5
! 0. - 80.956939 - 85.440135 - 86.963211 - 87.721475 !
         column 6
! - 88.176834 !
 db  =
         column 1 to 5
! 0. - 16.072235 - 22.011613 - 25.518228 - 28.011667 !
         column 6
! - 29.947396 !
```

Le malheur, avec l'instruction **dbphi**, c'est que l'on perd le vecteur fréquence et cette instruction faisant appel à l'instruction **phasemag** peut donner, pour la phase, un décalage de phase de −360° ou +360° suivant les versions de *Scilab* et suivant l'exemple traité.

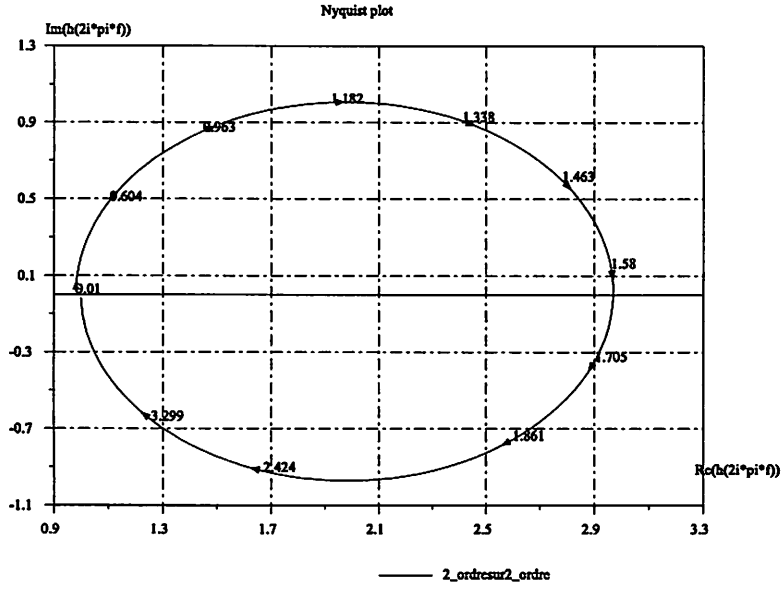
Une autre instruction utile pour la réponse fréquentielle est **phasemag** qui calcule le gain et la phase d'un vecteur complexe : cette commande cherche à obtenir une représentation continu de la phase afin d'éviter les discontinuités sur la courbe de phase (suivez mon regard...). Attention, cette instruction par le caractère de contrôle **'c'**, donne une représentation continue de la phase entre $-\infty$ et +360°, aux dires de la documentation, bien qu'en étudiant le programme source, on s'aperçoit que l'auteur de ce programme, compare le maximum de phase par rapport à zéro et décale la phase d'un nombre entier de termes de valeur 360°; si ce caractère de contrôle est **'m'** on a une représentation entre −360° et 0° : je reviendrais sur cette instruction lors de l'étude des lieux de *Bode* et *Black* d'un système.

Note au 20.03.2000 : les instructions précédentes peuvent être avantageusement remplacées par une seule instruction nommée **`dbphifr`** ; je donnerai ce nouveau programme en annexe avec la boîte à outils **`autoelem`**.

3.8.1 Représentation de Nyquist

La représentation de *Nyquist* est une courbe donnant en abscisse la partie réelle de $G(j\omega)$ et en ordonnée la partie imaginaire de $G(j\omega)$. Attention *Scilab* n'utilise pas la variable ω mais la fréquence à savoir f pour graduer le lieu de *Nyquist*.

```
-->xbasc()
-->s=%s;
-->sys=syslin('c',(s^2+2*.9*10*s+10^2)/(s^2+2*.3*10.1-
*s+10.1^2))
 sys  =
                              2
                   100 + 18s + s
                   -----------------
                                 2
                 102.01 + 6.06s + s
-->nyquist(sys,.01,100,'2_ordresur2_ordre')
```



Pour plus de renseignements sur le lieu de *Nyquist* faite appel au manuel : cette instruction ne pose aucun problème.

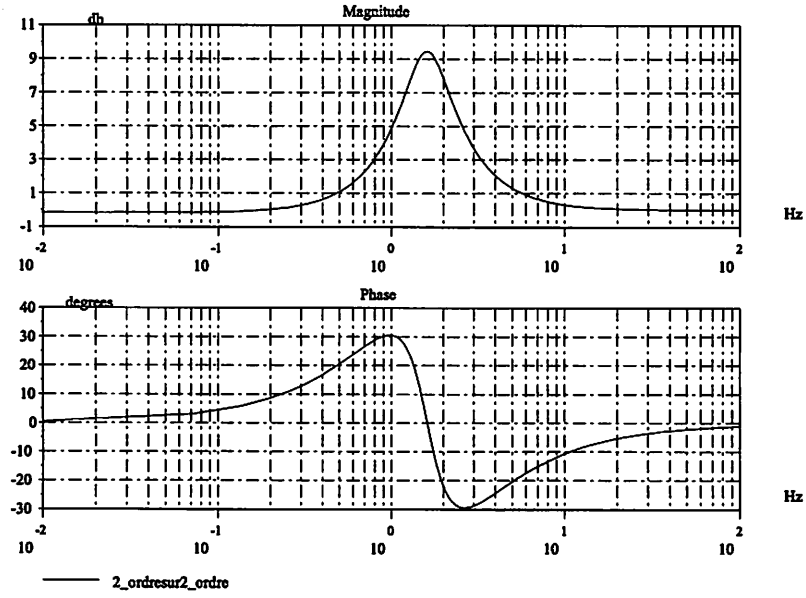
3.8.2 Représentation de Bode

La représentation de *Bode* est constituée de deux courbes : la première appelée courbe de gain, est la représentation de *20 log (| G (jω) |)* comme fonction du *log (ω)*. La seconde courbe, est le graphe de la phase φ comme fonction du *log (ω)*, (je rappelle que la phase φ représente le déphasage entre le signal de sortie sinusoïdal et le signal d'entrée, lui même sinusoïdal).

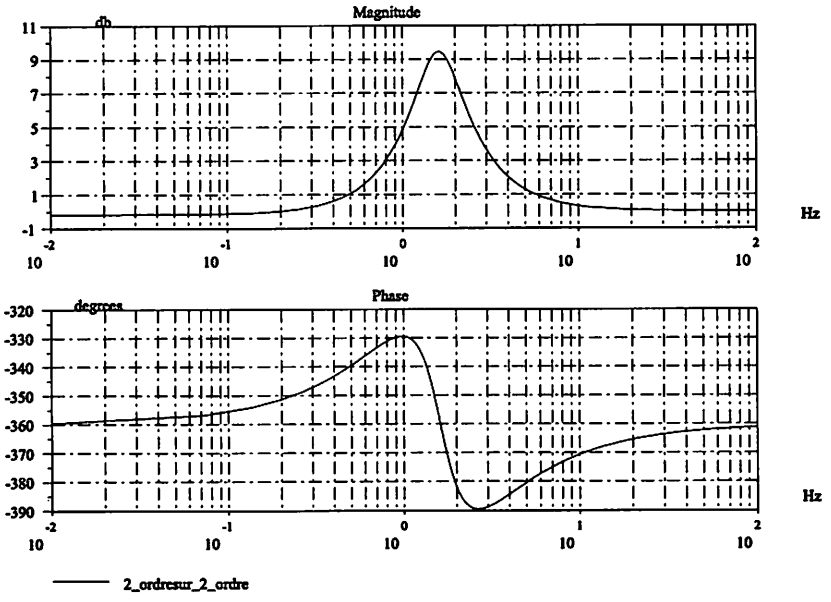
Comme pour le lieu de *Nyquist*, *Scilab* n'utilise pas l'argument pulsation mais la fréquence pour le tracé du lieu de *Bode*. Voici une session :

```
-->s=%s;
-->fr=(s^2+2*.9*10*s+10^2)/(s^2+2*.3*10.1*s+10.1^2);
-->sys=syslin('c',fr)
 sys  =
                          2
            100 + 18s + s
          -----------------
                            2
          102.01 + 6.06s + s
-->xbasc()
-->bode(sys,.01,100,'2_ordresur_2_ordre')
```

Si l'on ne souhaite pas tracer les deux courbes, gain et phase, on peut utiliser l'instruction **`gainplot`** qui ne donne que la courbe de gain du système.



Vous remarquerez que le lieu de *Bode* ainsi dessiné, correspond bien à un système (circuit) avance–retard de phase et que normalement, avec se système, quand la fréquence augmente, le déphasage commence par croître, en étant positif. Pour obtenir ce résultat, j'ai du corriger l'instruction **`phasemag.sci`**, instruction située dans le répertoire **`SCI/macros/auto`** du logiciel, sinon, on obtenait une courbe de *Bode* (et de *Black*), en phase, décalée de −360°, ce qui n'est pas logique avec un tel système : je propose dans la rubrique *bugs* une correction possible. Pour ne plus avoir ce problème utilisez la bibliothèque **`autoelem`** (Annexe A).



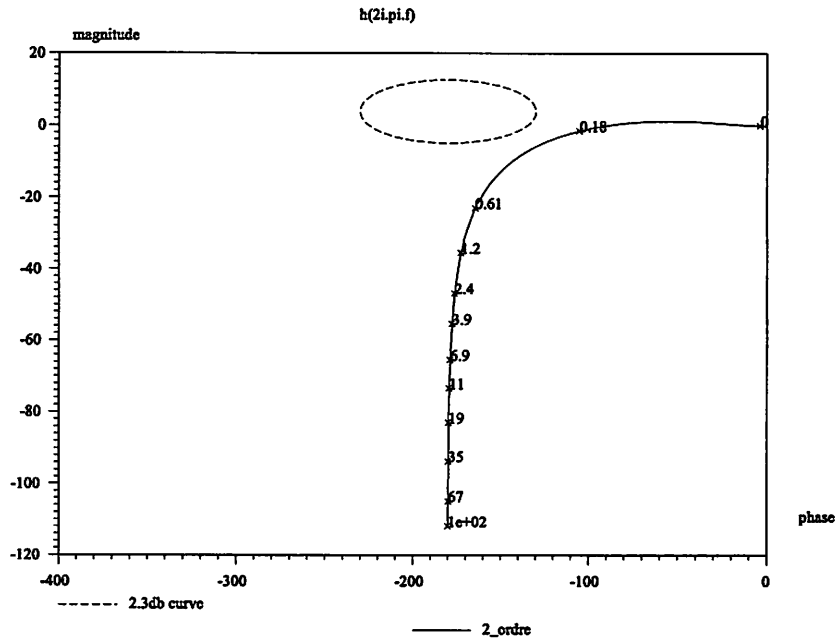
Une remarque à ce sujet ; cette correction influence les deux programmes : **bode** et **black**.

3.8.3 Représentation de Black

La représentation de *Black*, permet de synthétiser sur un même graphe les courbes de gain et phase. En abscisse, on définit le déphasage φ en degrés, et en ordonnées, le gain en décibels du système. Voici une session *Scilab* permettant de mettre en oeuvre l'instruction **black**.

```
-->sys=syslin('c',1/(s^2+s+1));
-->xbasc()
-->black(sys,.01,100,'2_ordre')
```

Vous remarquerez, que la commande **black** affiche en plus de la courbe demandée, le lieu iso-gain *2,3db*, valeur utile pour la synthèse d'un système.

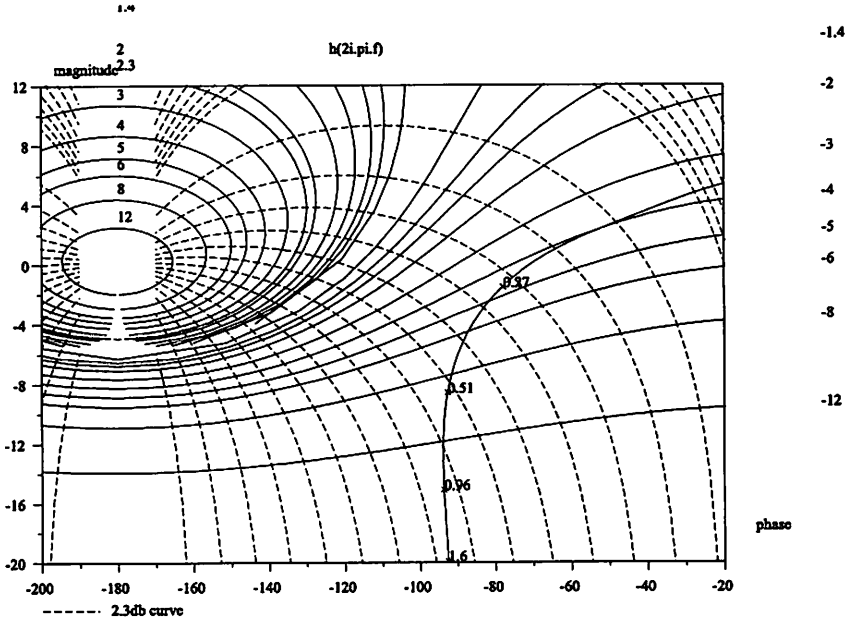


3.8.4 L'abaque de Black

L'abaque de *Black* est constitué d'un ensemble de deux réseaux de courbes orthogonales, lieux des points, en boucle fermée, où le gain est constant (valeur affichée sur une courbe de gain), et où la phase est constante.

```
-->s=%s;
-->fr=(2+3*s+s*s)/(1+3*s+2.5*s*s+s^3);
-->sl=syslin('c',fr)
  sl  =
                      2
          2 + 3s + s
        ----------------
                      2    3
       1 + 3s + 2.5s + s
-->xbasc()
-->black(sl,.01,100)
-->chart(list(1,0,2,3))
```

La représentation graphique est donné à la figure qui suit :



La première instruction graphique consiste à ouvrir une fenêtre graphique vide, **xbasc()**, puis à demander à *Scilab* de tracer dans cette fenêtre le lieu de *Black* du système précédemment défini, et enfin à placer sur ce lieu de *Black*, l'abaque de *Black*. Attention sur la figure de ce document je n'ai reproduit qu'une partie du lieu, pour ce faire j'ai utilisé le bouton **zoom** de la fenêtre graphique.

Enfin, voici quelques instructions qui peuvent être utiles pour déterminer quelques caractéristiques de systèmes bouclés : on verra cela plus profondément au chapitre concernant la synthèse des systèmes.

```
-->G=g_margin(sl)
 G  =
   Inf
--> P =p_margin(sl)
 P  =
  - 70.528779
```

On voit bien dans l'exemple choisi que la marge de gain (**g_margin**)est infinie, ce qui est logique, étant donné que la phase reste comprise entre 0 et $-\pi$.

Quant à l'instruction **p_margin**, elle donne le vecteur phase du système à une pulsation, pour laquelle le gain vaut 1 : vous ferez attention que cette instruction peut vous renvoyer un vecteur à plusieurs composantes (s'il y a

plusieurs fréquences à donner un gain de 1). De même ce n'est pas la définition habituelle du mot *marge de phase* : vous pouvez obtenir la marge de phase en faisant :

```
-->MP=180*ones(P)-abs(P)
 MP =
  109.47122
```

Enfin une dernière instruction **freson** permet de déterminer la (ou les) fréquences pour lesquelle le gain est maximal (résonance). Cette instruction renvoit un vecteur colonne, alors que partout dans le logiciel la fréquence est donnée sous forme vecteur ligne. De même attention si vous avez une indétermination pour la fréquence nulle (au moins un pôle et un zéro non simplifié à l'origine) **freson** vous renvoit une erreur. La solution ? faire **sys=tf2ss(sys)** puis **sys=clean(ss2tf(sys))** : c'est le problème de la simplification d'un pôle par un zéro.

```
-->freson(sl)
    ans  =
      0.
```

3.8.5 Rappel de cours, diagrammes asymptotiques de Bode : systèmes à déphasage minimal et non minimal

Après de nombreuses !!! années d'enseignement, je me suis aperçu que beaucoup d'étudiants n'avaient pas assimilé la relation de *Bayard–Bode*, relation liant la courbe de gain à la courbe de phase pour des systèmes à déphasage minimaux. Nous allons donc présenter les principaux résultats et définir la notion de système à déphasage minimal.

3.8.5.1 Systèmes à déphasage minimaux

On dira qu'un système est à déphasage minimal s'il ne possède ni pôle ni zéro dans le demi plan droit du plan complexe : système stable ne possédant pas de zéro dans le demi plan droit. Dans ces conditions il existe une relation liant la courbe de phase à la courbe de gain. Cette relation dite de *Bayard–Bode* est donnée par la relation suivante :

$$\varphi(\omega) = 2\omega/\pi \int_{-\infty}^{+\infty} \frac{[Log\,A(\alpha) - Log\,A(\omega)]}{\alpha^2 - \omega^2} d\alpha$$

Dans cette relation, $\varphi(\omega)$ représente le déphasage en fonction de la pulsation, et $A(\omega)$ est le gain en fonction de cette même pulsation.

Si l'on construit un diagramme, en ayant factorisé le système avec un terme constant (gain statique, en position, en vitesse, etc, suivant le cas), avec un terme dérivateur ou intégrateur pur muni du bon exposant, avec des termes du premier et / ou second degré pour le numérateur et le dénominateur de la forme : $(1+\tau_1 s)$ et / ou $(1+\tau_{2,1}s+\tau_{2,2}s^2)$ (s étant la variable symbolique, avec $\tau^2_{2,1}-4\tau_{2,2} < 0$ et $\tau_{2,2} > 0$), en ayant pour chacun de ces termes tracé des diagrammes réduits aux asymptotes (comportement aux basses et hautes fréquences des systèmes élémentaires), en ayant sommé algébriquement ces diagrammes, il existe alors, pour les *systèmes à déphasages minimaux*, une relation simple liant la courbe de phase asymptotique à la courbe de gain asymptotique. Cette relation se résume en une phrase : *à une pente de'x' fois 20 db par décade pour la courbe de gain correspond un déphasage de 'x' fois +90°* .

Il est bien évident que le logiciel de simulation devra respecter cette relation, pas de saut intempestifs, pas de décalages de *360°* dans un sens ou un autre, étant donné que *par convention*, en automatique on considère qu'un gain seul *k*, sans constante de temps, positif,est représenté en Bode par la droite *20 log (k)* et par la droite *0°*. Pour un gain négatif, la courbe de Bode est *20 log (|k|)* et *−180°*; et non *+180°* (diagramme de Black, et position du point *−1* pris par convention à *0 db*, *−180°*).

3.8.5.2 Systèmes à déphasage non minimaux, comment s'en sortir ?

D'abord il faut respecter la *factorisation de Bode* ; comme dans le cas précédent. On a donc des valeurs de τ_1 et / ou de $\tau_{2,1}$ qui peuvent être négatives. En multipliant le numérateur et le dénominateur de la transmittance par les termes $(1-\tau_1 s)$ ou par des termes de la forme $(1-\tau_{2,1}s+\tau_{2,2}s^2)$ ce qui ne change rien pour la réponse fréquentielle, on mettra ainsi en évidence, dans la transmittance un *déphaseur pur*. Un exemple mathématique éclaire le raisonnement.

Soit $G(s) = (1-s+s^2) / s$, on peut écrire cette expression sous la forme :

$G(s) =[(1-s+s^2) / (1+s+s^2)]\ \ [(1+s+s^2) / s]$; nous voyons apparaître pour le premier terme de la transmittance un déphaseur pur ; quant au second terme, il est à déphasage minimal. Le raisonnement que j'ai tenu pour deux zéros situés dans le demi plan droit s'applique aussi à deux pôles, la seule différence concerne la stabilité du système.

Remarque : je ne parlerais pas dans cet exposé des systèmes à retard pur, qui ne sont pas à déphasage minimal, mais dont le modèle mathématique n'est pas une fraction rationnelle.

3.8.6 Etude de la stabilité d'un système

Un problème très important en automatique est l'étude de la stabilité des systèmes bouclés, ou non.

L'étude algébrique des systèmes peut être faite en utilisant les deux instructions **routh_t** et **kpure**, voici deux exemples :

```
-->s=%s;den=poly([-1,6,-4,3,1+%i,1-%i],'s')
  den  =
                      2     3     4    5   6
     144 - 36s - 82s + 92s - 13s - 6s + s
-->routh_t(den)
  ans  =
!   1.          - 13.         - 82.      144.  !
! - 6.            92.         - 36.      0.    !
!   2.3333333   - 88.           144.     0.    !
! - 134.28571     334.28571     0.       0     !
! - 82.191489     144.          0.       0.    !
!   99.016309     0.            0.       0.    !
!   144.          0.            0.       0.    !
```

Dans ce cas, vous travaillez avec un polynôme sensé être le polynôme caractéristique d'un système linéaire (bouclé ou non).

```
-->sl=syslin('c',.5/den);
-->table=routh_t(denom(sl));
```

Dans ce contexte la table de Routh ne se justifie pas, en effet, comme on connait le dénominateur de la fonction de transfert du système, on peut donc, par l'instruction **roots** trouver directement, les racines de ce polynôme.

Dans le cas d'un système bouclé, c'est autre chose, car dans le cas d'un bouclage avec un gain **k**, l'instruction **routh_t(sl,k)** donne la table formelle, voici un exemple :

```
-->s=%s;num=poly([5,1],'s','c');
-->den=poly([0,-2,-3,-3],'s');
-->sl=syslin('c',num/den);
   sl  =
                        5 + s
                  ----------------
                          2    3   4
                 18s + 21s + 8s + s
-->k=poly(0,'k');//on definit la variable k
```

```
-->table=routh_t(sl,k)
  table  =
!   1                          21          5k   !
!   8                          18 + k      0    !
!   150 - k                    40k         0    !
!                  2                             !
!   2700 - 188k - k            0           0    !
!                  2     3                       !
!   108000k - 7520k - 40k      0           0    !
-->[KL,PL]=kpure(sl)
  PL  =
!   1.9813434i !
! - 1.9813434i !
  KL  =
    13.405773
```

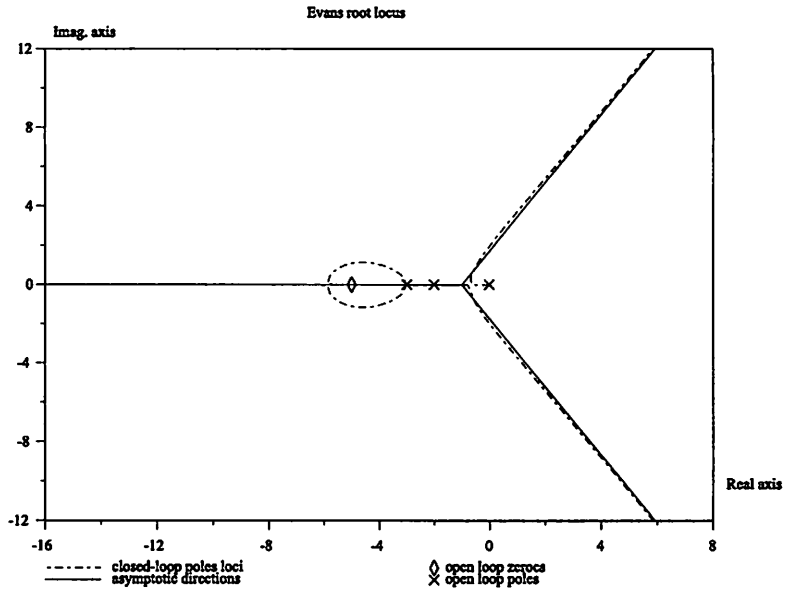
Dans cet exemple nous définissons un système bouclé à retour unitaire, dont la chaîne d'action est constituée de la mise en cascade d'un amplificateur de gain **k** positif, et du système linéaire de transmittance **sl**. *Scilab* construit la table de Routh formelle (table dépendant du gain **k**).

La table proposée tient compte de la correction proposée à la section 2.2.3. De même l'instruction **kpure**, corrigée, donne s'ils existent, les valeurs limites de **k**, ici **KL**, et les valeurs des pôles (imaginaires purs conjugués) **PL** du système bouclé pour cette valeur limite de **k**.

On donne à titre de complément, par l'instruction **evans**, le tracé du lieu des pôles du système bouclé. Dans l'instruction **evans** on donne comme argument d'entrée la transmittance de la boucle ouverte (sans le gain), *Scilab* se chargeant de construire le système bouclé, avec un gain **k** dans la chaîne d'action.

Attention, si le numérateur est une constante, (*type 1*), ne pas utiliser l'écriture **routh_t(syslin('c',num,den),k)** : à cause de la remarque faite à la section 3.6 .

```
-->evans(sl)
```



4 Synthèse d'un système linéaire par la méthode fréquentielle

Les exercices qui vont suivre ont pour but de réaliser la synthèse d'un réseau correcteur de stucture donnée, (avance de phase, retard de phase, retard–avance de phase, P.I.D etc...) afin que le système bouclé à retour unitaire constitué de la mise en série d'un système et du réseau correcteur choisi ait certaines propriétés.

4.1 Principe de la commande

Le but que se fixe l'automaticien en élaborant un système bouclé, consiste au choix d'un réseau correcteur, qui mis en cascade avec la procédé, permet de réaliser ainsi un système bouclé, ayant des performances nettement supérieures au système originel. Avant toute étude du système bouclé, il faut donc connaître les qualités et défauts du procédé.

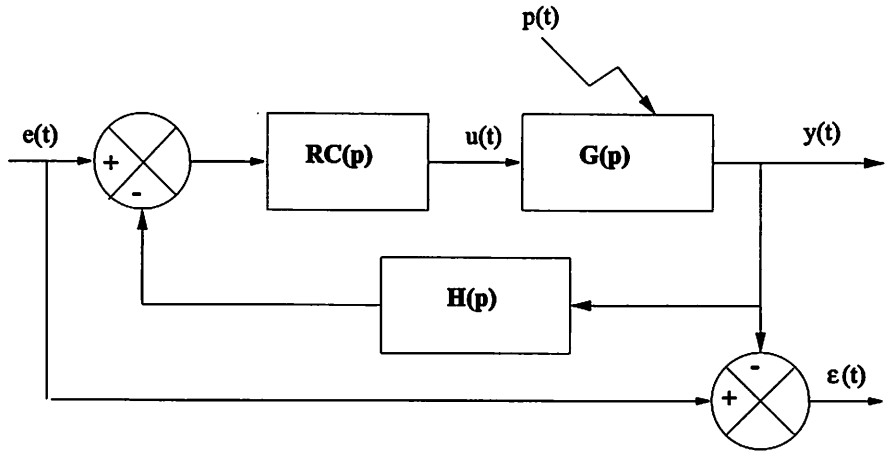
On peut, et cette liste n'est pas exhaustive, découvrir que le procédé est (ou n'est pas) :

- lent, son inertie est trop importante.
- mal amorti, il oscille trop longtemps sous l'effet d'une perturbation, ou d'un changement de point de consigne.
- il a tendance à dériver, sa sortie évoluant, alors que l'entrée reste constante : ceci est le signe de la présence d'une ou plusieurs intégrations.
- est instable, le correcteur devant donc, le rendre stable, avant de le corriger.

Du point de vue de l'automaticien, une structure de réseau correcteur, et les valeurs des paramètres de celui ci doivent permettre :

- d'améliorer la stabilité si nécessaire, ainsi qu'améliorer le comportement statique et dynamique du procédé.
- d'obliger le système à suivre au plus près la consigne désirée (même si celle ci évolue au cours du temps) : problème de poursuite. Il faut aussi que le réseau correcteur puisse annuler, autant que faire ce peut, les perturbations qui ne manquent pas d'influencer la dynamique du procédé : rejet des perturbations.
- Il sera donc nécessaire d'étudier, à tout instant, erreur de l'asservissement : différence entre ce que l'on souhaite avoir et ce que l'on a réellement.

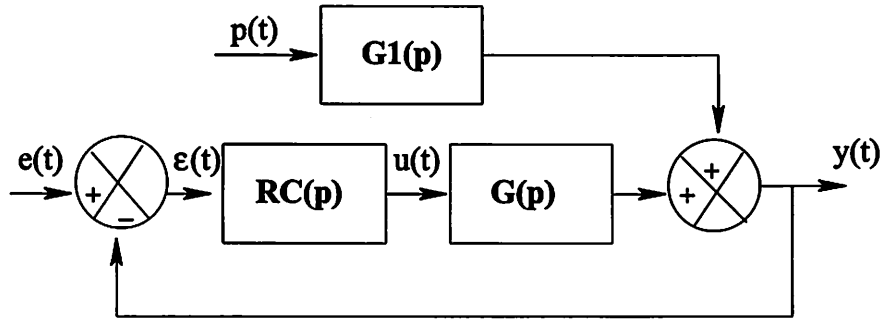
On peut sur la figure qui va suivre faire un schéma représentant un procédé bouclé avec un réseau correcteur.



On voit apparaître sur ce schéma, l'erreur de l'asservissement, différence entre la consigne et la sortie. Dans la suite on définira la précision statique, étude de l'erreur en régime permanent, et la précision dynamique, caractérisant cette erreur au cours du temps.

4.1.1 Précision statique

Quand on étudie la précision en régime statique d'un système, pour une entrée de type échelon de position, de vitesse ou d'accélération, on recherche la limite de ε *(t)* quand $t \rightarrow \infty$. Si l'on considère les pertubations, symbolisées par un signal *p (t)*, le procédé bouclé peut être représenté par la figure :



on peut exprimer l'erreur de l'asservissement par la relation :

ε (p)=e (p) / (1+RC (p) G (p))−p (p) G1 (p) / (1+RC (p) G(p))

En vertu du théorème sur les limites dans la transformée de Laplace l'étude de l'erreur permanente revient à faire l'étude de $p\varepsilon(p)$ quand $p \rightarrow 0$, et ceci pour un signal d'entrée donné (donc pour une expression de $e(p)$).

- **Erreur statique due à la consigne.**

  Seul l'original du premier terme intervient, on a donc le tableau suivant :

  (K est le gain statique en position de la chaîne d'action, K1 le gain en vitesse, et K2 le gain en accélération).

nombre de pôles à l'origine	0	1	2	>2
erreur en position	1/(1+K)	0	0	0
erreur en vitesse	∞	1/K1	0	0
erreur en accélération	∞	∞	1/K2	0

- **Erreur statique due à la perturbation.**

  Dans cette situation, en vertu du principe de superposition, on peut mettre la consigne à zéro, et l'on montre facilement, par l'étude du deuxième terme intervenant dans $\varepsilon(p)$, que l'erreur statique due à cette perturbation, diminue en augmentant le nombre de pôles à l'origine (en amont du point d'application de la perturbation), ainsi qu'en augmentant le gain statique de la chaîne d'action.

4.1.2 Précision dynamique

Le comportement dynamique d'un système peut être entièrement caractérisé par sa réponse impulsionnelle (ou aussi par sa réponse indicielle).

L'étude temporelle étant généralement plus complexe on préfère se ramener à une étude fréquentielle (généralement dans le plan de *Black*), et comparer le comportement du système par rapport à des systèmes connus : du premier et/ou second ordre.

- **Comparaison avec un premier ordre.**

  Que la boucle ouverte $RC(p)\,G(p)$ ait pour modèle une transmittance k/p ou $k/(1+\tau p)$ la boucle fermée sera toujours du premier ordre (seule

l'erreur permanente différera : et sera nulle dans le premier cas et vaudra $1/(1+k)$ dans le second). Mais si l'on considère l'erreur par rapport à cette réponse permanente, on sait, que cette erreur est inférieure à 5% pour un temps $t > 3\tau_1$ (τ_1 constante de temps de la boucle fermée).

Si l'on se ramène maintenant dans le domaine fréquentiel, on remarque que la rapidité de réponse est directement liée à la bande passante du système bouclé, bande passante que l'on caractérise par la pulsation de coupure (à *3db* ou *6db*, la cassure du lieu de *Bode* se faisant en un point $\omega = 1/\tau_1$). Vous remarquerez ceci en changeant dans le modèle p en $\tau_1 p$, et dans ce cas vous changez t en t/τ_1 et ω en $\tau_1\omega$.

- **Comparaison avec un second ordre.**

  Afin d'obtenir un erreur permanente nulle pour une réponse à un échelon on choisit pour boucle ouverte du système de référence (à retour unitaire) $RC(p)\,G(p) = k/(p(1+\tau p))$. La boucle fermée vaut donc: $W(p) = k/(k+p+\tau p^2)$ et est un système du second ordre de pulsation naturelle $\omega_n = (k/\tau)^{1/2}$ et de coefficient d'amortissement $\xi = 1/(2(k\tau)^{1/2})$.

  Si le coefficient $\xi > 1$ les pôles de $W(p)$ sont réels et la réponse indicielle est semblable à celle d'un système du premier ordre (sauf pour $t = 0$).

  Quand $\xi < 1$ on peut constater les faits suivants :

  La valeur du premier dépassement $D1$ de la réponse indicielle constitue un bon indicateur de l'amortissement.
  Ce premier dépassement, ainsi que le temps de pic, (valeur du temps pour ce premier pic), valent :

  $$D1 = e^{-\pi\xi/\sqrt{1-\xi^2}}, \quad T1 = \frac{\pi}{\omega_n\sqrt{1-\xi^2}}$$

  A ξ fixé, les oscillations sont d'autant plus rapides que la pulsation naturelle est grande.

  A ω_n fixé, les amplitudes des oscillations s'amortissent d'autant plus rapidement que ξ est grand. Mais attention si ξ est trop grand, la réponse peut être très lente.

  Par une étude complète du système du second ordre on montre, qu'à ω_n fixé, le temps de réponse à 5% passe par un minimum pour ξ voisin de 0,7, ce dépassement valant alors 4%.
  Quant à la réponse fréquentielle, on la caractérise par le facteur de

résonance Q, qui pour un système du second ordre vaut :

$$\frac{1}{2\xi\sqrt{1-\xi^2}}$$

Pour un facteur de résonance $Q=1,3$ soit $Q=2,3db$, on a une valeur de ξ de $0,42$, valeur donnant un dépassement $D1$ de 30%. Cette valeur pour Q est souvent prise comme référence.

Pour un facteur de résonance $Q=1,3$ soit $Q=2,3db$, on a une valeur de ξ de $0,42$, valeur donnant un dépassement $D1$ de 30%. Cette valeur pour Q est souvent prise comme référence.

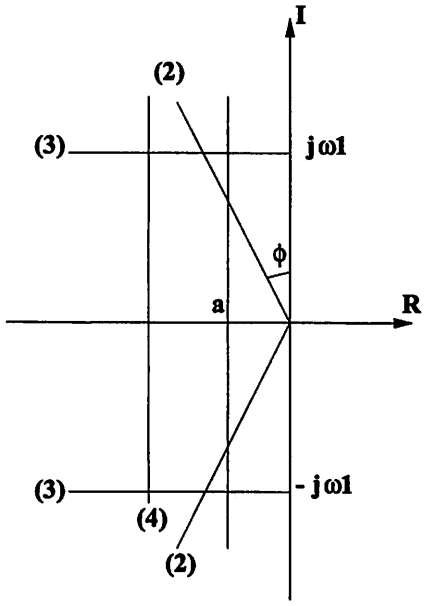
De même, comme la pulsation de résonance vaut $\omega_r = \omega_n (1-2\xi^2)^{1/2}$, cette pulsation est un indicateur de la rapidité de la réponse indicielle : plus ω_n et donc ω_r sont élevés et plus le temps de réponse est court.

Attention: pour avoir une bonne précision statique il faut soit rajouter un ou plusieurs intégrateurs dans la chaîne d'action et / ou augmenter le gain statique de cette même chaîne, ceci a pour conséquence de déstabiliser ou de rendre plus instable le système bouclé : dilemme **stabilité–précision.**

4.2 Cahier des charges

A partir des remarques que l'on vient de faire, on peut définir un cahier des charges type.

- Le système bouclé devra être stable et précis en régime statique : mais attention au fameux dilemme stabilité–précision.
- Il faudra le régler afin qu'il soit rapide, mais attention à ne pas saturer les actionneurs ; par simulation on vérifiera l'amplitude de la commande : étude du signal $u(t)$. Ceci pourra être fait par analogie avec un système du second ordre.
- Toutes ses considérations permettent de faire un dessin permettant de placer les pôles ainsi que les pôles dominants (ceux qui sont le plus près de l'axe imaginaire), dans une région du plan complexe : le dessin ci–dessous présente cette région.



En résumé lors de la synthèse d'un système asservi on cherchera, pour réaliser un bon asservissement, à avoir :

- un système stable avec des marge de phase et marge de gain suffisantes.
- avoir un gain en boucle ouverte assez grand.
- avoir une fréquence de résonance élevée.
- lié à la remarque précédente on devra donc essayer, autant que faire ce peut, d'augmenter la bande passante, mais attention cette augmentation a pour effet de ne pas atténuer les bruits.

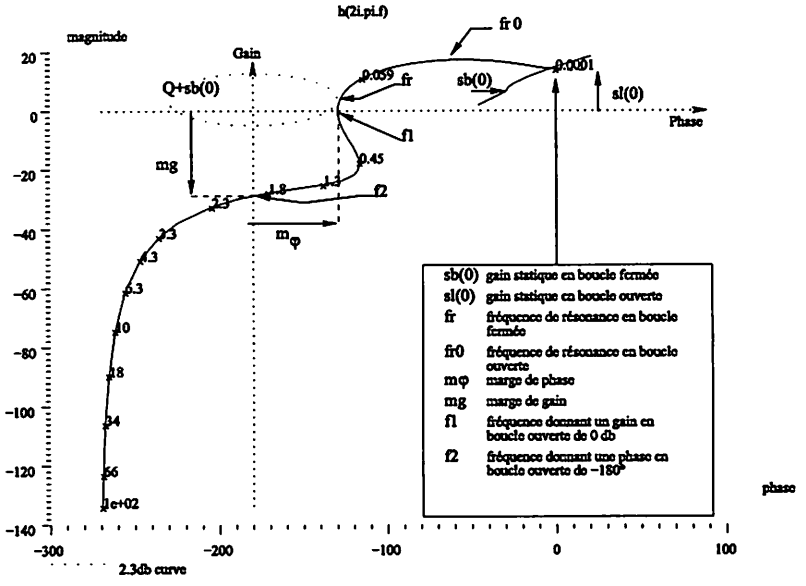
4.3 Synthèse d'un réseau correcteur par la méthode de Black

Nous avons dans la section, *exercices d'automatique avec Scilab : analyse d'un système*, introduit la représentation dans le plan de *Black* d'un système, ainsi que l'abaque de *Black*. Ce qu'il faut retenir de cette section peut se résumer par les points suivants :

1. Ramenez votre système bouclé, à un système à retour unitaire.
2. Tracez le lieu de *Black* de la chaîne d'action de ce système à retour unitaire.
3. A partir du tracé de la courbe de *Black* de la boucle ouverte, on déduira les propriétés essentielles de la boucle fermée. Un exemple illustre ces faits.

```
-->s=%s;
-->sl=syslin('c',5*(1+s)/(.1*s^4+s^3+15*s^2+3*s+1))
sl=
```

```
          5 + 5s
 ---------------------
              2   3       4
 1 + 3s + 15s + s + 0.1s
-->black(sl,.0001,100)
```



Si nous appelons `sb(s)`, dans l'exemple choisi, la transmittance de la boucle fermée, `sl(s)` étant le modèle de la boucle ouverte, la précision en régime statique vaut donc `sl(0)/(1+sl(0))`.

On cherchera donc la courbe gain=constante de l'abaque de *Black* passant par le point de fréquence nulle, et on aura ainsi le gain statique en boucle fermée : si le système, en boucle ouverte, possède une ou plusieurs intégrations, le point à la fréquence nulle, a pour coordonnées sur la courbe de *Black* : $(-\alpha\ 90°, +\infty)$, α étant le nombre d'intégrations de la chaîne d'action.

4.4 Synthèse d'un régulateur de structure donnée

On choisira une structure de régulateur permettant de vérifier le cahier des charges. Il convient donc:

- d'éloigner le lieu de *Black* de la boucle ouverte du point −1 : augmenter la marge de phase et de gain : avoir une marge de phase supérieure à 45°, et une marge de gain supérieure à 10 ou 12 db.
- augmenter le gain de la boucle ouverte, ou mettre le point à fréquence zéro à l'infini.
- augmenter la bande passante donc provoquer un tassement en fréquence vers les gains élevés : on diminue le temps de réponse de la boucle.
- Ceci peut être résumé par une avance de phase aux fréquences moyennes et un retard de phase aux basses fréquences.

La structure du correcteur devra donc réaliser les actions suivantes :

1. Action proportionnelle : translation verticale du lieu de *Black*.
2. Action dérivée : translation plus ou moins horizontale du lieu de *Black* et ceci vers la droite.
3. Action intégrale : remonter vers le haut les points du lieu de *Black* qui correspondent aux basses fréquences.

Pour réaliser ces contraintes on utilisera des réseaux correcteurs, à action proportionnelle, à avance de phase (action dérivée), à retard de phase (action intégrale) ou des combinaisons de ses réseaux.

4.4.1 Action proportionnelle

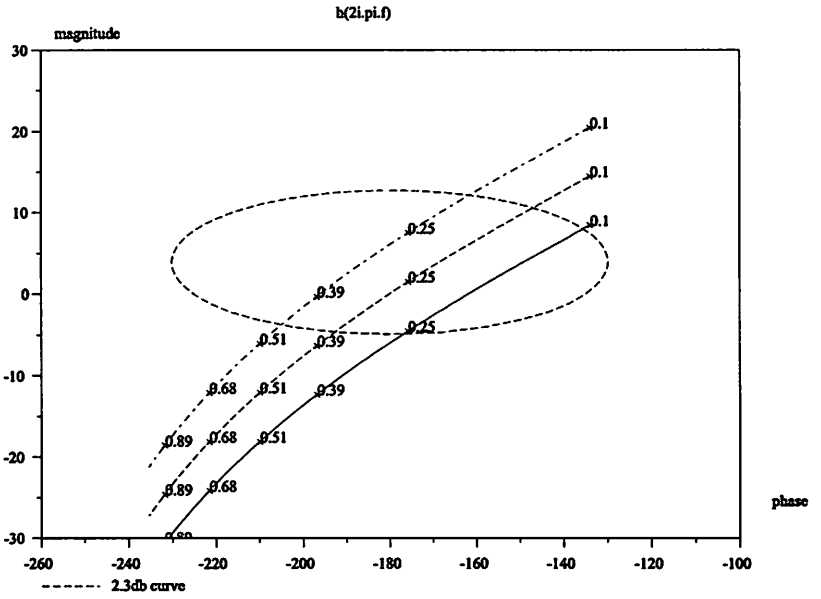
Le modèle du réseau est $RC(p) = K$, son action a pour effet de remonter ($K>1$) ou descendre ($K<1$) la courbe de *Black* de la boucle ouverte : reprenons un exemple de système à la limite de stabilité, avec *Scilab*, et traçons **`sl, 0.5*sl, 2*sl`**

```
-->s=%s;  den=s*(1+s)*(1+s/3);
-->sl=syslin('c',1/den);
-->;getd("/home/lpovy/autoelem");
-->[K,P]=kpurel(sl)
 P  =

!   1.7320508i !
! - 1.7320508i !
```

```
 K  =


    4.
//je me place a la limite de stabilite
-->sl=K*sl
 sl  =
                  4
    ---------------------------
                    2             3
     s + 1.3333333s + 0.3333333s
sl_05=.5*sl;  sl_2=2*sl;
-->black([sl_05;sl;sl_2],.1,10)
```

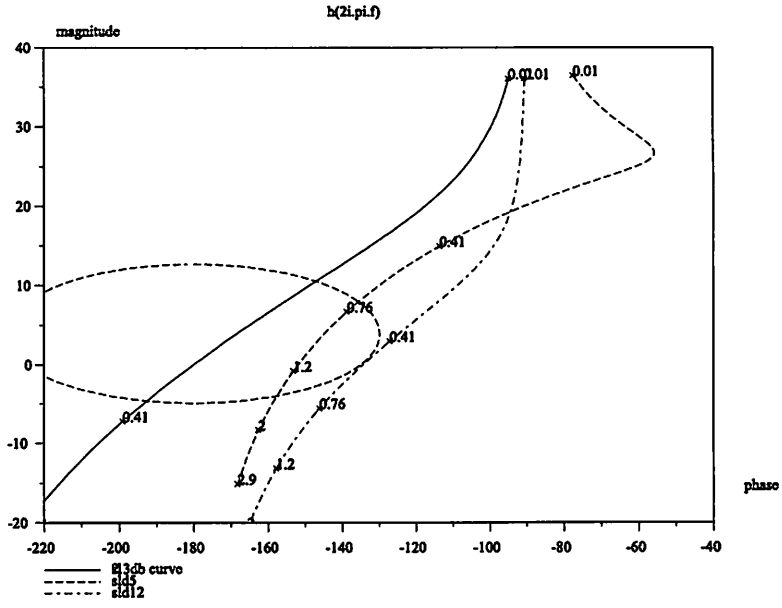


Nous voyons par exemple simple, l'effet d'un gain sur le lieu de *Black*.

4.4.2 Action proportionnelle et dérivée

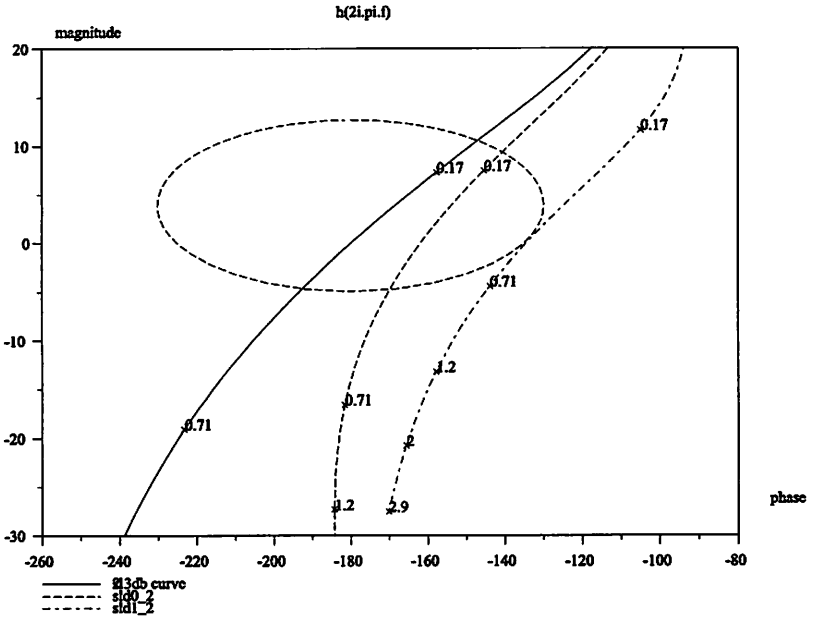
Le réseau correcteur théorique, (une explication sera donnée plus loin dans ce paragraphe), a pour modèle $RC(p) = K(1+\tau_d p)$, et en prenant $K=1$, ce qui ne change pas le raisonnement, on voit apparaître une avance de phase appréciable, mais attention au choix de τ_d. Il faut prendre pour τ_d une valeur supérieure ou proche de l'inverse de la valeur de la pulsation de résonance, (en boucle fermée), du système, avant correction : $\tau_d > 1/\omega_r$: l'effet d'avance de phase doit se faire suffisamment tôt, mais pas trop tôt. La valeur théorique de τ_d est facilement obtenue par le programme *Scilab* qui continue l'exercice précédent.

```
-->taud=1/imag(P(1))
 taud  =
    0.5773503
//valeur théorique que je ne prends pas :
//je prends approximativement deux et neuf fois cette
//valeur
-->sld5=(1+5*s)*sl;  sld1_2=(1+1.2*s)*sl;
-->black([sl;sld5;sld1_2],.01,3,['sl';'sld5';'sld1_2']
```



Reprenons ce même exemple en diminuant fortement τ_d: nous voyons l'effet sur le lieu de *Black* : j'ai conservé le lieu sans correction, le lieu avec $\tau_d = 1,2$ et avec $\tau_d = 0,2$. Vous pouvez tracer, `sltaud=(1+taud*s)*sl`, afin de comparer par rapport aux exemples précédents et suivants.

```
-->sld0_2=(1+.2*s)*sl;
-->black([sl;sld0_2;sld1_2],.01,3,['sl';'sld0_5';'sld-
1_2'])
```



Je vous conseille fortement de réaliser le programme suivant : (suite du même exercice).

```
-->sltaud=sl*(1+taud*s)
-->bblack([sl;sltaud],.01,3,['sl';'sltaud'])
-->sltaud2=sl*(1+2*taud*s)
-->bblack([sl;sltaud;sltaud2],.01,3,['sl';'sltaud';'s-
ltaud2'])
//on recherche avec la fonction dbphifr, les facteurs
//de résonance en boucle fermée, les déphasages et
```

```
//fréquences correspondantes.
-->[d,phi,f]=dbphifr(sltaud/(1+sltaud),freson(sltaud/-
(1+sltaud)))
-->[d2,phi2,f2]=dbphifr(sltaud2/(1+sltaud2),freson(sl-
taud2/(1+sltaud2)))
//puis faire les réponses indicielles des systèmes
//bouclés
-->t=linspace(0,10,101);
-->sbtaud=sltaud/(1+sltaud)
-->y1=csim('step',t,sbtaud);
-->[ymax,indmax]=max(y1)
-->sbtaud2=sltaud2/(1+sltaud2)
-->y2=csim('step',t,sbtaud2);
-->[y2max,ind2max]=max(y2)
-->plot2d([t;t]',[y1;y2]')
```

Cet exercice permet, avec deux choix du paramètre du réseau correcteur, d'étudier les fréquences de résonance et les facteurs de résonance en boucle fermée. Puis, par une simulation temporelle, réponse à un échelon unitaire, de déterminer l'amplitude du premier pic de la réponse ainsi que l'instant (l'indice)auquel ce produit ce premier pic. Vous voyez bien par cette petite simulation, en choisissant une valeur double de la valeur théorique du paramètre du P.D, que l'on obtient un dépassement de 21,6 %, et un coefficient de surtension Q, de 2,19 db.

On pourra avec ses deux exemples par les instruction **p_margin** et **g_margin** vérifier l'augmentation des marges de stabilité. De même, pour un réseau bien centré, on voit le tassement des fréquences vers le haut : augmentation de la bande passante.

4.4.3 Réseau correcteur à avance de phase

Le réseau correcteur à action dérivée étant physiquement irréalisable (présence du dérivateur pur), on utilise de préférence, le réseau à avance de phase. Le modèle de ce réseau est $RC(p) = (1+a\tau p) / (1+\tau p)$, avec $a > 1$.

Comme nous l'avons vu en cours, on constate que l'avance de phase maximale est indépendante de τ, et vaut : $\varphi_{max} = arcsin\,[(a-1)/(a+1)]$. Cette avance de phase maximale a lieu pour une pulsation : $\omega_a = 1/(\tau\sqrt{a})$. On pourra à l'aide de *Scilab*, tracer les différents lieux fréquentiels de ce réseau pour différentes valeurs de a. Voici un exemple de synthèse de ce type de réseau correcteur.

Exercice : *synthèse d'un réseau correcteur à avance de phase.*

On veut réaliser la synthèse d'un système à retour unitaire, dont la chaîne d'action est constituée d'un amplificateur de gain k, et d'une transmittance $G(p) = 1 / [p (1+p) (1+p/3)]$, comme le système est de classe 1, on cherche k pour avoir une erreur en vitesse de 40%, par *Scilab* déterminer la marge de phase, de gain, la pulsation de résonance de même que le facteur de résonance en boucle fermée (pour la valeur du gain trouvé). Faire une simulation des réponses impulsionnelle, indicielle, puis fréquentielle : on aura préalablement vérifié que le système bouclé est stable et trouvé la valeur du gain limite k et la pulsation d'oscillation omegal pour cette valeur limite du gain.

Instructions utiles pour réaliser cette première partie de l'exercice.

`poly ; syslin ; routh_t ; kpure ; p_margin ; g_margin ; freson ; horner ou freq ; dbphifr ; csim ; plot2d ; linspace ; bbode ; bblack ; nyquist.`

On place en cascade avec la chaîne d'action un réseau correcteur à avance de phase de transmittance $RC(p) = (1+a\tau p) / (1+ \tau p)$: et l'on cherche à rajouter, pour la valeur de k précédemment calculée, une phase supplémentaire de l'ordre de $55°$, donnez la valeur de a. Initialisez le réseau, c'est à dire, trouvez une première valeur à τ. Tracez les lieux de *Bode* du réseau correcteur, puis de la boucle ouverte corrigée.

Tracez sur le même graphique, pour la valeur de k choisie, les lieux de *Black* de la boucle ouverte du système non corrigé et du système corrigé.
Pour réaliser ceci on utilisera l'instruction **`bblack([sl1;sl2],f1,f2,['com1';'com2'])`**, on remarquera par la même occasion le petit bogue qui se manifeste par la superposition du premier commentaire désiré et du commentaire inclus dans l'instruction **`black`**, à savoir `2,3db`.

Retouchez les valeurs de τ, pour avoir, avec la valeur de k précédente, ou pour une valeur de k conduisant à une erreur plus petite, une marge de phase d'au moins $45°$, et un coefficient de surtension de $Q = 2,3\ db$.

Le réseau étant maintenant identifié, faire une étude temporelle de ce système bouclé : réponse indicielle ; de même faire l'étude temporelle du signal de commande $u(t)$, sortie du réseau correcteur, pour l'entrée précédente.

Instructions utiles pour réaliser cette deuxième partie de l'exercice.

Les mêmes instructions que dans la première partie, avec en plus l'instruction **`maxi`** permettant de déterminer le premier dépassement de la réponse indicielle.

4.4.4 Réseau correcteur à action proportionnelle et intégrale : P.I

Comme nous l'avons vu précédemment, l'introduction d'un intégrateur dans la chaîne d'action permet d'améliorer la précision en régime statique, (section 4.1.1) mais ce résultat est au détriment de la stabilité. En effet un intégrateur pur déphase le système de −90°. Pour cette raison on préfère introduire un réseau de type proportionnel et intégral (P.I).

Ce correcteur a pour transmittance : $RC(p) = 1+1/(\tau_i p)$; par une étude des courbes de *Bode* de ce réseau on peut remarquer qu'aux hautes fréquences ($\omega > 1/\tau_i$) ce correcteur n'introduit plus de déphasage et ne change pas le gain du système quand on met en cascade ce réseau et la boucle ouverte du système à étudier. Afin d'éviter l'effet déstabilisant de l'intégrateur, on prendra $1/\tau_i < \omega_r$. Cette pulsation ω_r étant la pulsation de résonance de la boucle fermée avant introduction du réseau.

Par un réglage satisfaisant, ni la pulsation de résonance ω_r ni le facteur de résonance Q ne sont modifiés, seul la précision statique est améliorée.

Voici un exemple de détermination d'un réseau P.I, (la boucle ouverte a pour transmittance $G(p) = 1/[p(1+p)(1+0,25p)]$].

```
-->s=%s;
num=1;den=s*(1+s)*(1+s/4);sl=syslin('c',num/den)
   sl  =
                1
    ----------------
                2       3
     s + 1.25s + 0.25s
```

Deux fonctions sont maintenant chargées : **bblack.sci** et **dbphifr**, (une explication sera donnée à la fin du rapport).

```
-->;getf("/home/lpovy/autoelem/bblack.sci");
-->;getf("/home/lpovy/autoelem/dbphifr.sci");
```

Vous pouvez, à ce stade tracer la courbe de *Black* de **sl** et déterminer la fréquence de résonance, le gain et la phase correspondante, de la boucle fermée par l'instruction : **dbphifr**.

```
-->sb=sl/(1+sl);
-->[db,phi,frb]=dbphifr(sb,freson(sb))
  frb  =
0.1299495
```

```
 phi  =
- 76.236457
db  =
3.0914789
```

On détermine ici la transmittance de la boucle fermée.

Par cette instruction, nous obtenons la fréquence de résonance **frb**, on peut ainsi déterminer une première valeur pour τ_i soit : τ_r.

```
-->tor=1/(2*%pi*frb)
tor  =
1.2247449
```

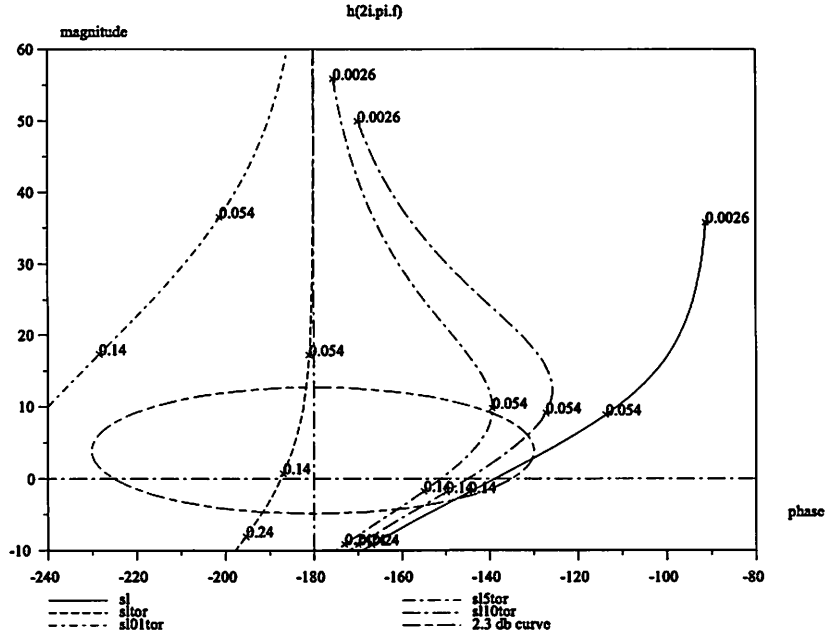
On peut maintenant, pour plusieurs valeurs de τ_i construire différents réseaux correcteurs : par exemple, $RC(p) = 1+1/\tau_i p$, puis $RC(p) = 1+1/(0{,}1\tau_i p)$, et enfin $RC(p) = 1+1/(5\tau_i p)$, pour voir l'influence de τ_i sur les caractéristiques de la boucle fermée.

```
-->rctor=syslin('c',1+1/(tor*s))
rctor  =
      1 + 1.2247449s
      --------------
        1.2247449s
-->sltor=sl*rctor
   sltor  =
                  1 + 1.2247449s
      -----------------------------------
                2             3            4
      1.2247449s + 1.5309311s + 0.3061862s
-->rc01tor=syslin('c',1+1/(.1*tor*s))
 rc01tor  =
      1 + 0.1224745s
      --------------
        0.1224745s
-->sl01tor=sl*rctor
   sl01tor  =


                  1 + 0.1224745s
      -----------------------------------
                2             3            4
      0.1224745s + 0.1530931s + 0.0306186s
```

D'une manière analogue, on peut construire par exemple **sl5tor,sl10tor** et avec la fonction **bblack** afficher sur un même graphe les cinq lieux de Black.

```
-->sl5tor=sl*(1+1/(5*tor*s));
-->sl10tor=sl*(1+1/(10*tor*s));
-->com=['sl','sltor','sl01tor','sl5tor','sl10tor'];
-->bblack([sl;sltor;sl01tor;sl5tor;sl10tor],.001,.3,com)
```



Ce graphe est suffisamment clair et met bien en évidence que pour $\tau_i = 1/\omega_r$, on a rendu instable un système qui était de toute façon stable à l'origine (courbes **sltor,sl01tor**. La valeur de τ_i est beaucoup trop faible, et une valeur normale de τ_i doit être de l'ordre de 5 à 10 fois ($1/\omega_r$). On remarquera de même que l'adjonction d'un réseau correcteur bien placé, permet, tout en conservant des marges de stabilité quasiment identiques, par l'introduction dans la boucle ouverte d'un intégrateur, d'avoir, dans ce cas, non seulement une erreur permanente nulle pour une entrée en échelon, mais d'avoir aussi une erreur permanente nulle pour une entrée en rampe : le système est de classe deux.

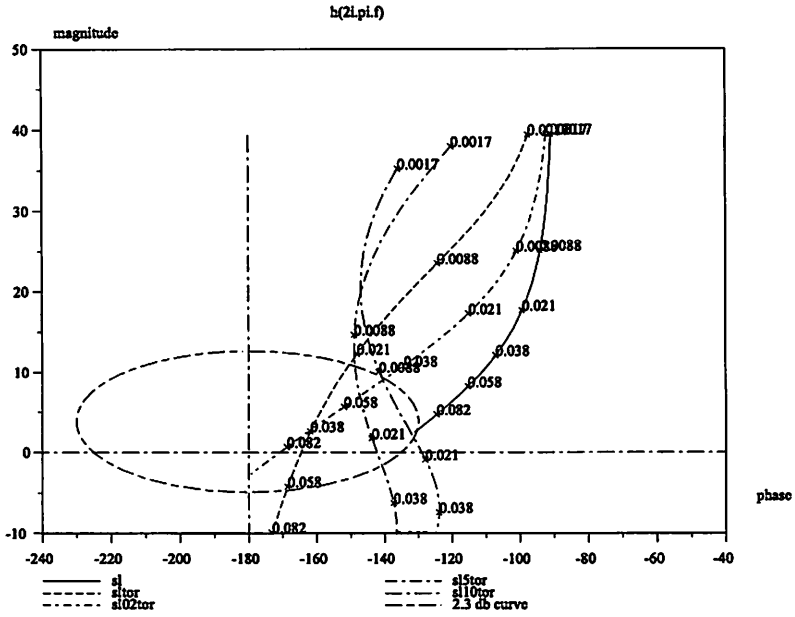
Ce réseau améliore, et comment ! la précision d'un système bouclé.

4.4.5 Réseau correcteur à retard de phase

Ce réseau est une approche du réseau de type P.I, et s'il n'introduit pas d'intégration dans la chaîne d'action il peut, s'il est bien placé, en conservant une bonne marge de stabilité, permettre d'augmenter le gain de la chaîne d'action et ainsi, d'augmenter sérieusement la précision en régime statique. Son modèle mathématique est : $RC(p) = (1+\tau p) / (1+b\tau p)$ avec $b>1$.

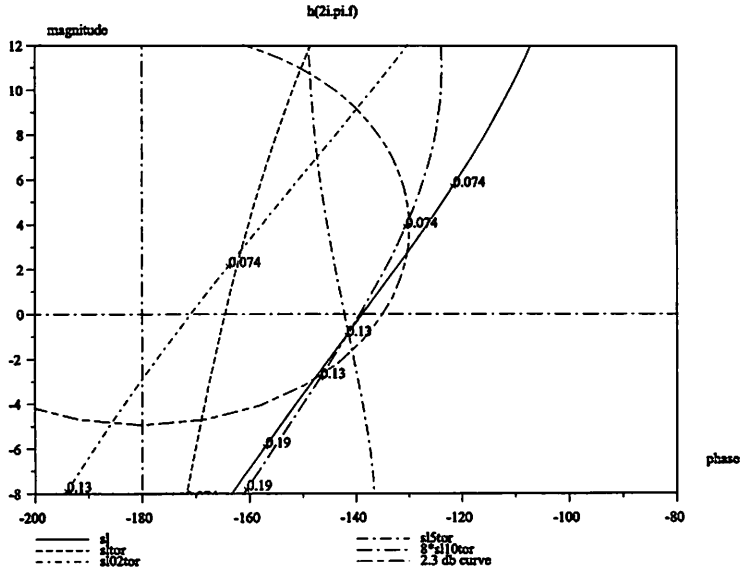
On pourra avec Scilab très simplement étudier les courbes de *Nyquist*, *Bode* et *Black* de ce réseau pour différentes valeurs de b (en prenant $\tau=1$) le produit $\tau\omega$ étant un nombre sans dimension. Comme précédemment on placera le nombre $1/\tau << \omega_R$. Le seul effet déplaisant de ce réseau est la diminution de la bande passante : voici un exemple avec *Scilab* de placement d'un tel réseau.

```
-->s=%s;sl=syslin('c',1/(s*(1+s)*(1+s/4)));
-->;getf("/home/lpovy/autoelem/bblack.sci");
-->;getf("/home/lpovy/autoelem/dbphifr");
-->sb=sl/(1+sl);
-->[db,phi,frb]=dbphifr(sb,freson(sb));
-->tor=1/(2*%pi*frb)
 tor  =
    1.2247449
-->rctor=syslin('c',(1+tor*s)/(1+10*tor*s))
 rctor  =
    1 + 1.2247449s
    --------------
    1 + 12.247449s
-->sltor=sl*rctor;
-->rc02tor=syslin('c',(1+0.2*tor*s)/(1+10*0.2*tor*s));
-->sl02tor=sl*rc02tor;
-->rc5tor=syslin('c',(1+5*tor*s)/(1+10*5*tor*s));
-->sl5tor=sl*rc5tor;
-->rc10tor=syslin('c',(1+10*tor*s)/(1+10*10*tor*s));
-->sl10tor=sl*rc10tor;
-->com=['sl','sltor','sl02tor','sl5tor','sl10tor'];
-->bblack([sl;sltor;sl02tor;sl5tor;sl10tor],.001,.3,c-
om)
```



Dans cet exemple nous voyons que l'adjonction d'un réseau à retard de phase avec $\tau = 1/\omega_r$ rend le système nettement moins stable : courbe **sltor**, afin de montrer l'influence de la valeur de τ, j'ai fait tracer à *Scilab* le lieu de *Black* un système **sl02tor** dont la valeur de τ est cinq fois plus faible : on rend encore moins stable la boucle fermée. Par contre en donnant à τ une valeur dix fois plus importante, et **augmentant le gain de la chaîne d'action d'un facteur huit,** on conserve **le même degré de stabilité** tout en **augmentant la précision** en régime statique, pour une **entrée en rampe,** de ce facteur **huit.**

Le réseau améliore, s'il est bien placé, la précision du système bouclé.



Voici les deux instructions qui permettent de tracer le bon lieu de *Black.*

```
-->com=['sl','sltor','sl02tor','sl5tor','8*sl10tor'];
-->bblack([sl;sltor;sl02tor;sl5tor;8*sl10tor],.001,.3-
,com)
```

4.5 Réglage d'un P.I.D par la méthode du pivot

Parmi tous les types de réseaux correcteurs utilisés industriellement, le réseau de type P.I.D est sans doute encore, et pour longtemps, le plus utilisé. Son modèle mathématique théorique est :

$RC(p) = K(1 + 1/(\tau_i p) + \tau_d p)$

En réalité, le terme dérivé, dans sa réalisation pratique a pour modèle le facteur suivant : $\tau_d p / (1+\alpha \tau_d p)$ avec α prenant une valeur proche de un dixième.

Quand on fait l'étude théorique de ce réseau, par exemple en traçant son lieu de *Bode*, on remarque qu'il existe une pulsation particulière, $\omega_1 = 1 / (\tau_i \tau_d)^{1/2}$ où le réseau n'avance ni ne retarde la phase, en ce point, le gain est de *0db*.

C'est ce point, qui une fois convenablement choisi permettra, de basculer le lieu de *Black* non corrigé, vers la gauche pour des pulsations inférieures à ω_1, et vers la droite pour des pulsations supérieures à ce pivot : en effet par une étude des lieux de *Bode* du P.I.D, on montre facilement que la phase de ce réseau est négative (comprise entre −90° et 0°) pour des fréquences inférieures à $\omega_1/(2\pi)$ et est positive (comprise entre 0° et 90°) pour des fréquences supérieures à cette valeur. Quant au gain, il décroit jusqu'à 0db (pour $\omega=\omega_1$) puis recroit quand la fréquence augmente. Vous ferez, avec *Scilab* l'étude du lieu de *Bode* du P.I.D théorique pour vérifier ces remarques.

Le choix du point de pivotement est donc très important, en effet l'introduction du réseau P.I.D a pour but de :

− Améliorer la précision en régime statique (introduction de l'intégrateur) sans pour cela réduire la stabilité.

− Améliorer la stabilité : marge de phase de gain, amortissement ζ.

On choisit le point de pivotement, si on souhaite un amortissement élevé, ordre de grandeur 0,7, on devra choisir ce point au dessus de l'axe 0db, légèrement à gauche de la verticale −90° (20° à gauche). Afin de rendre la méthode plus simple dans une première approche, on fixera $\tau_i = 4\tau_d$: avec cette valeur, le P.I.D a un zéro double.

Voici un exemple de session Scilab mettant en évidence ces considérations.

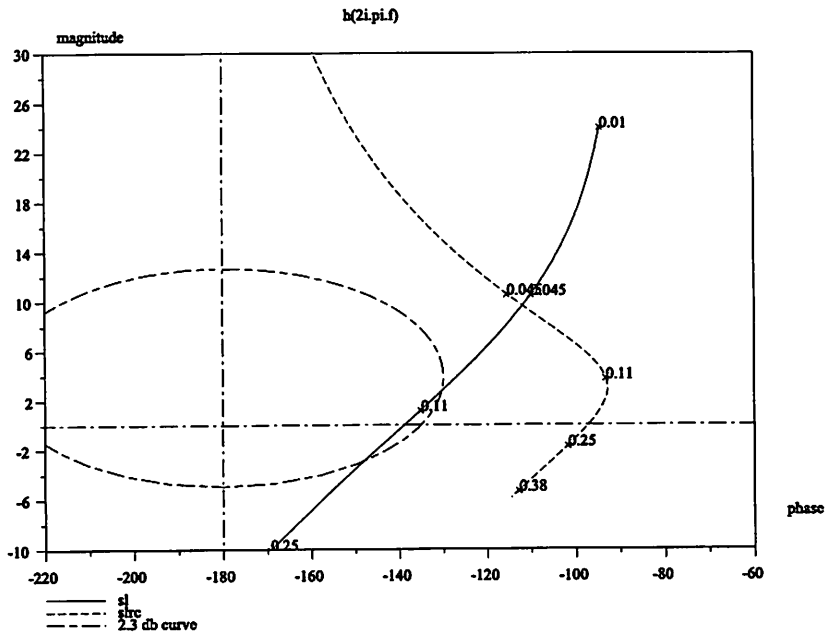
```
-->s=%s;sl=syslin('c',1/(s*(1+s)*(1+s/4)));
-->;getd("/home/lpovy/autoelem");
-->bblack(sl,.01,.4)
```

Je ne mets pas dans le texte le lieu de *Black* : il a déjà été tracé à l'exercice précédent, (courbe `sl`). On remarque qu'une fréquence de l'ordre de 0,05 hertz correspond aux remarques faites précédemment.

En première approche on a donc $\tau_i = 1/(0{,}05\pi)$. Le réseau a donc pour modèle :

$RC(p) = 1 + 1/(6{,}37\,p) + 1{,}59\,p \qquad (\tau_d = \tau_i/4)$

```
-->toi=1/(%pi*.05)//toi=2/(2*%pi*fp)
-->tod=toi/4       //tod=1/(2*(2*%pi*fp))
-->rc=syslin('c',(1+toi*s+toi*tod*s*s)/(toi*s))
 rc  =
                              2
1 + 6.3661977s + 10.132118s
 ---------------------------
          6.3661977s
-->slrc=sl*rc
 slrc  =
                                      2
          1 + 6.3661977s + 10.132118s
    ------------------------------------
              2             3             4
    6.3661977s + 7.9577472s + 1.5915494s
-->bblack([sl;slrc],.01,.4,['sl','slrc'])
```



Nous voyons, par cet exemple très simple, l'intérêt de cette méthode. J'ai pris ici une fréquence correspondant à une phase en boucle ouverte de l'ordre de −110°. On pouvait par le programme qui suit affiner ce point de pivotement.

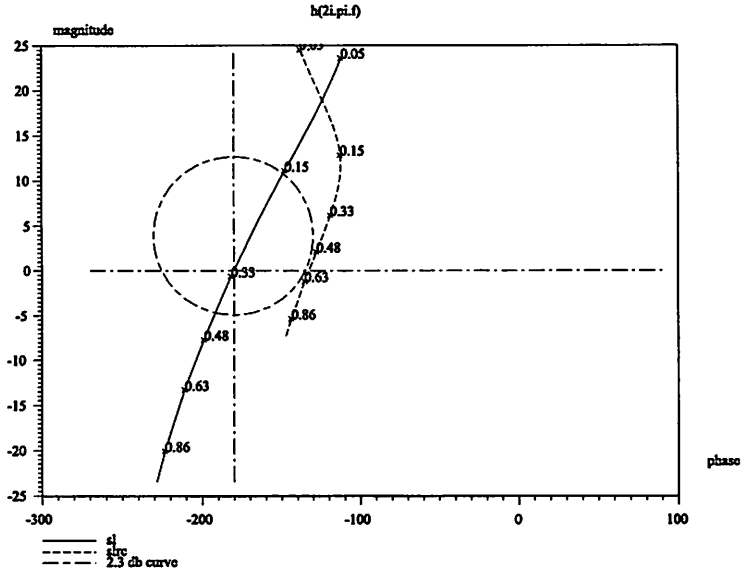
```
-->s=%s;sl=syslin('c',1/(s*(1+s)*(1+s/4)));
-->[d,p,f]=dbphifr(sl,.04,.1,.005);
-->omegp=2*%pi*(f(find(p<-109&p>-111)))
 omegp  =
          column 1 to 5
!   0.2724205    0.2755750    0.2787660    0.2819940
0.2852593  !
          column 6 to 9
!   0.2885625    0.2919039    0.2952840    0.2987032  !
-->omegp=omegp(5)
 omegp  =
    0.2852593
-->rc=syslin('c',(1+1/((2/omegp)*s)+(1/(2*omegp))*s))
 rc  =
                                  2
    1 + 7.0111642s + 12.289106s
    --------------------------
            7.0111642s
-->sl1=rc*sl
 sl1  =
                                       2
        1 + 7.0111642s + 12.289106s
    ----------------------------------
              2            3           4
    7.0111642s + 8.7639552s + 1.752791s
-->bblack([sl;sl1],.01,10,['sl','sl1'])
```

Cette méthode est beaucoup plus précise pour le choix du point de pivotement, on obtient ce point par les deux instructions des lignes trois et quatre du programme.

Cette fois, en augmentant le gain de la chaîne d'action et en nous plaçant à la limite de stabilité pour le système sans réseau correcteur (*Klimite=5*), par la même méthode, en choisissant pour fréquence de pivot $f_1 = 0{,}08$ hertz on obtient les résultats suivants:

```
-->sl=5*sl
   sl  =
```

```
                5
      -----------------
                2       3
      s + 1.25s + 0.25s
-->toi=1/(%pi*.08)
   toi  =
      3.9788736
-->tod=toi/4
   tod  =
      0.9947184
-->rc=syslin('c',(1+toi*s+toi*tod*s*s)/(toi*s))
   rc  =
                                    2
      1 + 3.9788736s + 3.9578587s
      --------------------------
              3.9788736s
-->slrc=sl*rc
   slrc  =
                                        2
         5 + 19.894368s + 19.789294s
      ---------------------------------
                2           3           4
      3.9788736s + 4.973592s + 0.9947184s
-->bblack([sl;slrc],.05,1,['sl','slrc'])
-->[db,phi,fr]=dbphifr(slrc/(1+slrc),freson(slrc/(1+slrc)))
   fr  =
     0.5401616
  phi  =
   - 59.703174
   db  =
     1.9266743
```



Nous voyons facilement qu'avec ce réseau, on a stabilisé le système (le facteur de résonance est maintenant très proche de 2,3db, en réalité 1,93db) et bien sur amélioré la précision : la boucle ouverte possède deux intégrateurs. On peut même, maintenant se permettre d'augmenter légèrement le gain de la chaîne d'action afin d'avoir un facteur de résonance de 2,3db.

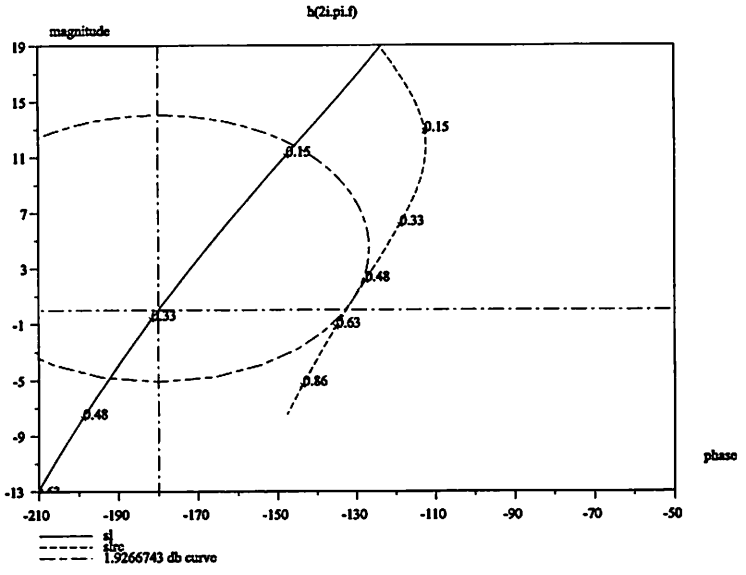
Remarque : Ne plus tenir compte de celle ci.

Avec la nouvelle fonction nommée **bblack** on peut tracer le lieu gain=constante que l'on souhaite afficher : voici un exemple.

```
-->slrc1=list([sl;slrc],1.9266743)
slrc1  =
     slrc1(1)
!           5                                  !
!   ------------------                         !
!             2       3                        !
!   s + 1.25s + 0.25s                          !
!                                              !
!                                        2     !
!       5 + 19.894368s + 19.789294s            !
!   ---------------------------------------    !
!              2           3             4     !
```

```
!    3.9788736s + 4.973592s + 0.9947184s   !
      slrc1(2)
     1.9266743
-->bblack(slrc1,.05,1,['sl','slrc'])
```

On crée une liste constitue des deux systèmes étudiés et du nombre ici 1,92... caractérisant la valeur du facteur de résonance que l'on souhaite afficher, on obtient le graphe suivant :



Je voudrais faire ici une remarque sur cette méthode du pivot et la méthode de synthèse dite méthode de Ziegler et Nichols.
Par la méthode de Ziegler et Nichols, quand on travaille en boucle ouverte, on détermine expérimentalement un modèle de la boucle ouverte, (ref bibliographique [3] pages 245 à 247), modèle caractérisé par deux paramètres : a et T_R et c'est à partir de ces deux paramètres que l'on détermine τ_i avec $\tau_i = 4\tau_d$.
On remarque facilement que si $\omega_1 = 1 / (\tau_i \tau_d)^{1/2}$ est la pulsation du pivot on a :

$\omega_1 = 1/T_R$, en effet on a $\tau_d = 0,5/\omega_1$ et $\tau_i = 2/\omega_1$ par la méthode du pivot et l'on obtient ces mêmes valeurs par le critère de Ziegler et Nichols, en ayant $\omega_1 = 1/T_R$.

4.6 Réglage d'un réseau retard–avance de phase

Après avoir étudié le réseau P.I.D, il est tout naturel introduire le réseau retard–avance de phase qui tout en ayant des effets comparables, peut être plus facilement réalisé.

Je rappelle que le but de ce type de réseau est de faire un retard de phase aux basses fréquences, et une avance de phase aux moyennes fréquences. De même on cherchera à augmenter le gain de la chaîne d'action afin d'améliorer la précision.

Comme c'est la mise en série d'un réseau à retard de phase et d'un réseau à avance de phase, son modèle mathématique est donc :

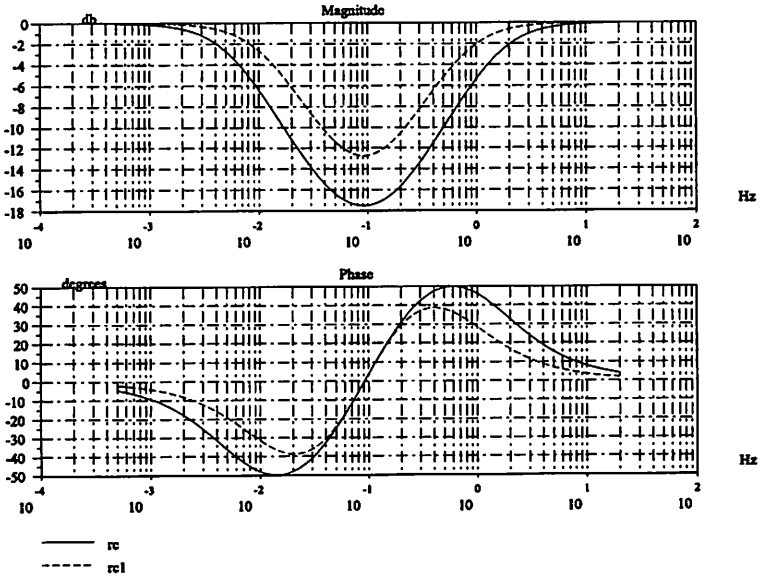
$RC(p) = [(1+\tau_2 p)(1+\tau_3 p)] / [(1+\tau_1 p)(1+\tau_4 p)]$. Si on veut que le gain en hautes fréquences reste égal à 1, on doit avoir la relation : $\tau_1 \tau_4 = \tau_2 \tau_3$. De même on cherchera à avoir une avance de phase la plus grande possible, en restant dans des limites raisonnables (influence de la dérivation sur les signaux bruités), il est donc normal de prendre le rapport des constantes de temps égal à environ 10 (avance de phase maximale de l'ordre de 55° : voir la section 4.4.3).

On fera de même avec le réseau à retard de phase, dans ces conditions on aura une courbe de Bode de ce réseau, pour le gain symétrique par rapport à $1/(\tau_2 \tau_3)^{1/2}$ avec $\tau_1\tau_4 = \tau_2\tau_3$, et une courbe de phase symétrique par rapport à ce point (en ce point le réseau n'introduit pas de déphasage, mais contrairement au P.I.D, il introduit une atténuation : voir la Figure46).

```
-->s=%s;
-->rc=syslin('c',((1+s)*(1+3*s))/((1+.1*s)*(1+30*s)))
   rc  =
                    2
        1 + 4s + 3s
       -------------
                      2
     1 + 30.1s + 3s
```

Tout en gardant le même point de symétrie, traçons un autre réseau où l'on remplace, τ_2 en $2\,\tau_2$, τ_3 en $\tau_3 / 2$ etc..., on a donc Figure46 :

```
-->rc1=syslin('c',((1+2*s)*(1+1.5*s))/((1+.2*s)*(1+15*s)))
      rc1  =
                        2
                 1 + 3.5s + 3s
                 -------------
                              2
                 1 + 15.2s + 3s
-->bode([rc;rc1],.0005,20,['rc','rc1'])
```



A partir des remarques précédentes, on choisit un point, sur la courbe de *Black* de la boucle ouverte où le déphasage ne changera pas par l'introduction du réseau, puis, on cherche une pulsation où l'on doit appliquer l'avance de phase maximale, cette valeur vaut pratiquement $1 / (\tau_3 \tau_4)^{1/2}$, car à cette pulsation le réseau à retard de phase n'agit que très peu. Bien sur par symétrie, le minimum

de déphasage introduit par le réseau retard de phase se produit à une pulsation valant $1 / (\tau_1 \tau_2)^{1/2}$.

Appelons *f1, f2, f3* les fréquences où les déphasages introduits par le réseau sont respectivement : minimum, nul et maximum on a donc :

$(\tau_2 \tau_3)^{1/2} = 1/(2\pi f2)$, $(\tau_3 \tau_4)^{1/2} = 1/(2\pi f3)$, *et* $\tau_1 = 10\,\tau_2$, $\tau_3 = 10\,\tau_4$

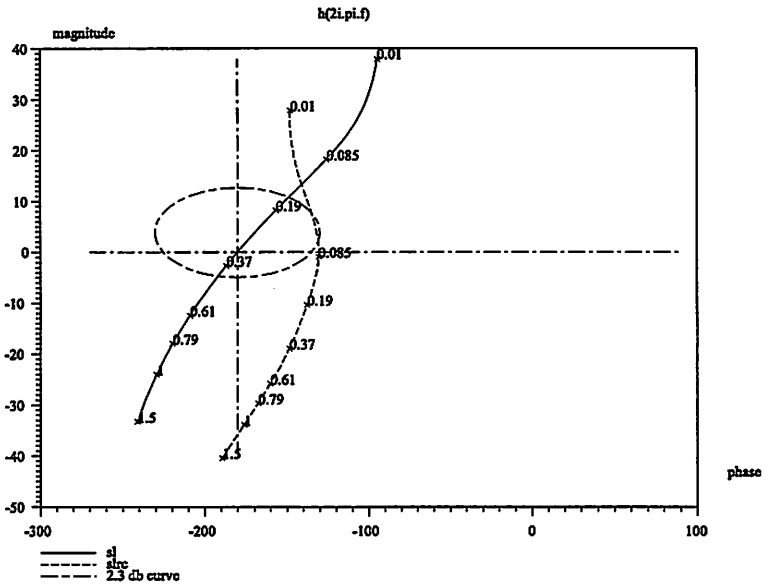
En prenant les logarithmes de ses fonctions on obtient un système de quatre équations à quatre inconnues.

Dans l'exemple choisi, si f2 = *0,1*hertz, point correspondant à un déphasage en boucle ouverte de −120° environ (pseudo pivot), si l'on prend *f3* aux environs de 1hertz (endroit ou l'on désire l'avance de phase maximale), en utilisant la méthode proposée, on trouve, l'ensemble des valeurs de τ. Vérifions ceci avec *Scilab.*

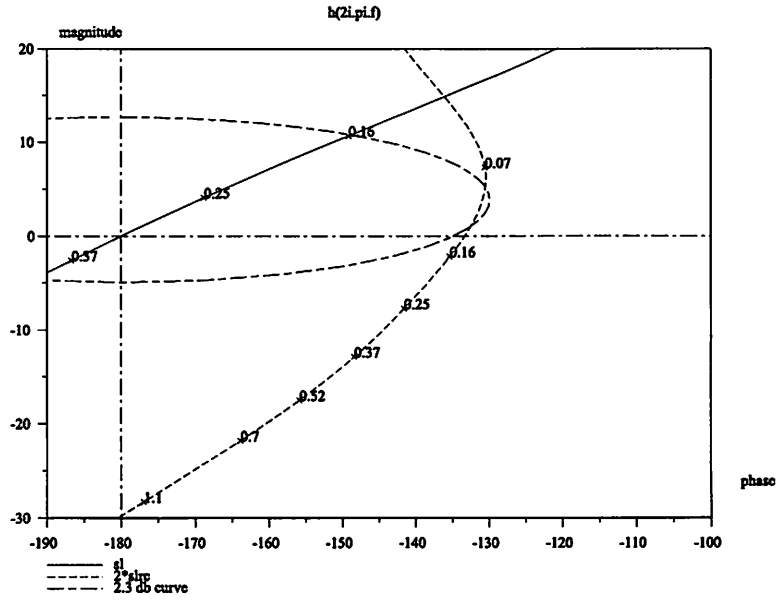
```
-->;getf("/home/lpovy/autoelem/bblack.sci");
-->;getf("/home/lpovy/autoelem/dbphifr");
-->s=%s;sl=syslin('c',5/(s*(1+s)*(1+s/4)));
-->bblack(sl,.08,1.5)
-->A=[1,-1,0,0;0,1,1,0;0,0,1,-1;0,0,1,1];
-->C=[log(10);-2*log(2*%pi*.1);log(10);-2*log(2*%pi*1-
)];
```

Les logarithmes des constantes de temps vérifient l'équation : $A\,X=C$.

```
-->X=inv(A)*C;
-->tau=exp(X)
 tau  =
!   50.329212 !
!   5.0329212 !
!   0.5032921 !
!   0.0503292 !
-->num=(1+tau(2,1)*s)*(1+tau(3,1)*s);
-->den=(1+tau(1,1)*s)*(1+tau(4,1)*s);
-->rc=syslin('c',num/den)
rc  =
                                2
   1 + 5.5362133s + 2.5330296s
   ---------------------------
                                2
   1 + 50.379541s + 2.5330296s
-->slrc=sl*rc;
-->bblack([sl;slrc],.01,1.5,['sl','slrc'])
```




```
-->mg=g_margin(slrc)
   mg  =
       35.795429
-->mphi=180-abs(p_margin(slrc))
 mphi  =
    49.601046
-->bblack([sl;2*slrc],.01,1.5,['sl','2*slrc'])
```

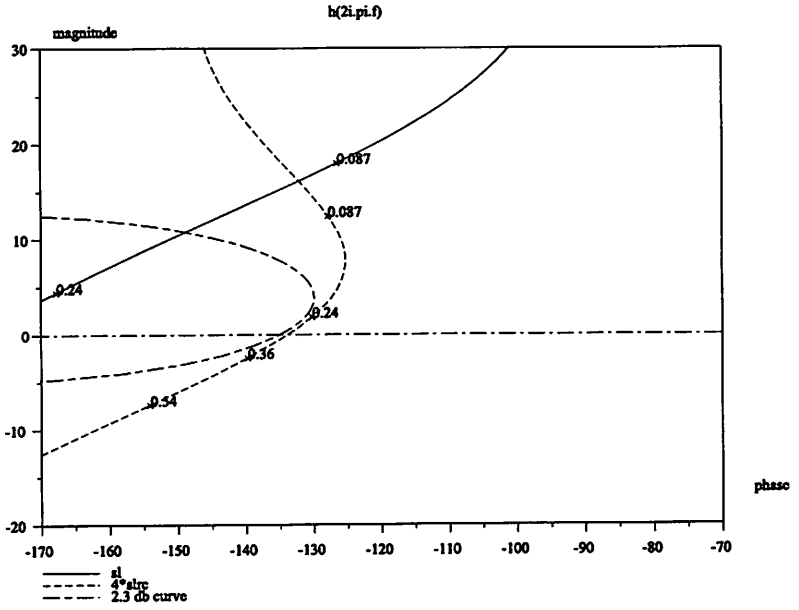


Avec ce réseau, le système n'a pas une marge très très importante, (bien que l'on ait multiplié le gain par deux), on va donc retoucher les deux fréquences, *f2* et *f3* : on prend *f2 = 0,09* hertz : on remonte légèrement le point de pseudo pivotement, et on remonte plus fortement le point où l'avance de phase est maximale en prenant *f3 = 0,5* hertz : on obtient ainsi le programme *Scilab* :

```
-->C=[log(10);-2*log(2*%pi*.09);log(10);-2*log(2*%pi*-
.5)];
-->X=inv(A)*C;
-->tau=exp(X)
 tau  =
!   31.067415 !
!   3.1067415 !
!   1.0065842 !
```

```
!   0.1006584 !
-->num=(1+tau(2,1)*s)*(1+tau(3,1)*s;
-->den=(1+tau(1,1)*s)*(1+tau(4,1)*s);
-->rc=syslin('c',num/den)
 rc  =
                                2
    1 + 4.1133257s + 3.127197s
    --------------------------
                                2
    1 + 31.168073s + 3.127197s

-->slrc=sl*rc;
-->bblack([sl;4*slrc],.01,1.5,['sl','4*slrc'])
```



Par cette retouche on a pu ainsi augmenter le gain de la chaîne d'action, **il est quatre fois plus important**, tout en conservant une bonne marge de phase, de gain et un bon facteur de résonance.

Par cette méthode on peut avec un tel logiciel de simulation, faire le choix le plus adéquat du réseau correcteur.

Une dernière remarque :

N'oubliez pas de simuler les réponses temporelles du système bouclé et les réponses de l'actionneur : sortie du réseau correcteur.

5 Conclusion

Le but de ce document est d'initier l'étudiant à l'automatique de base, en éclairant à l'aide du logiciel de simulation qu'est *Scilab* le premier cours d'automatique. Pour un étudiant, il serait souhaitable de refaire les exercices proposés. Dans le futur je pense proposer un autre document permettant d'aborder les concepts plus modernes de l'automatique.

Ce document a été réalisé grâce au formateur de texte *Thot*, logiciel vous permettant une exportation de votre document en *Latex*, en *Html* et bien sur vous proposant une impression en *Postscript* qui lui même peut être transformé en fichier *.pdf* par la commande *Linux* (à faire dans un terminal):

```
ps2pdf mon_fichier.ps mon_fichier.pdf .
```


Annexe A Programmes spécifiques

Nous allons donner dans ces annexes des programmes spécifiques permettant d'étudier les systèmes à modèles continus possédant ou non des retards purs.

A.1 Programmes d'algèbre et d'automatique

Ces programmes (macros) écrit en language *Scilab*, traitent de problèmes d'algèbre et de problèmes spécifiques à l'automatique.

La fonction **`bodfact.sci`** : cette fonction algébrique, factorise un polynôme, une fraction rationnelle *f (s)*, un système linéaire, sous la forme classique de *Bode* : en retournant un ensemble de vecteurs *[K,L,TN,TD]*.

- *K* : C'est la valeur de l'expression $s^{-L} f(s)$ pour $s = 0$: gain statique du système ou gain statique en vitesse ...
- *L* : C'est le nombre entier, positif, négatif ou nul, de zéros (ou pôles) nuls de *f (s)* (nombre de dérivations ou d'intégrations de cette fonction).
- *TN* : Vecteur colonne dont chacun des éléments est un polynôme du premier et/ou second degré de la forme $1+\tau_1 s$ ou $1+\tau_2 s+\tau_3 s^2$ (cas de racines complexes conjuguées).
- *TD* : Même chose que pour *TN* mais ce vecteur concerne le dénominateur.

La fonction **`dbphifr.sci`** : cette macro est spécifique à l'automatique, et elle remplace avantagement les deux fonctions **`phasemag.sci`** et **`dbphi.sci`**. Cette macro peut être utilisée avec des polynômes (type 2), des fractions rationnelles, des systèmes continus, échantillonnés, linéaires (type 16), avec retard pur ou élément fonction de la fréquence en cascade pour les modèles continus (type 15). Vous verrez le manuel de cette fonction pour la syntaxe.

Vous trouverez ces deux fonctions dans le répertoire **`.../autoelem`**.

A.2 Programmes de tracé de lieux

Afin de pouvoir tracer les divers lieux fréquentiels pour des systèmes particuliers, à retard pur par exemple, j'ai retouché les trois diagrammes à savoir *Bode, Black, Nyquist*, les nouvelles fonctions se nomment :
`bbode.sci`, **`bblack.sci`**, **`nnyquist.sci`**, **`ggainplot.sci`**, et **`cchart.sci`** et sont aussi situées dans le répertoire **`.../autoelem`**.

Annexe B Comment faire pour utiliser votre bibliothèque

Le but de ce dernier exposé est de pouvoir d'une manière temporaire ou définitive exploiter des *macros* que vous avez testées avec *Scilab.*

B.1 Utilisation temporaire

Pour une utilisation temporaire de macros Scilab il suffit de faire : `;getd('chemin_du_repertoire/repertoire')` dans une fenêtre *Scilab.* Toutes les *macros,* (fichiers texte avec le suffixe `.sci`) situées dans **repertoire** seront compilées et disponibles à partir de cet instant. Mais l'inconvénient de cette méthode est qu'il faut, à chaque session *Scilab,* retaper cette commande afin d'avoir accès aux fonctions contenues dans cette bibliothèque.

B.2 Utilisation définitive

Un conseil : si vous êtes sur que vos programmes ne contiennent pas d'erreurs, vous pourrez les utiliser d'une manière définitive en exécutant les commandes suivantes.

1. En étant administrateur, vous devez créer un répertoire que je nomme `jojo` dans le répertoire : `SCI/macros` (`SCI` étant par exemple le répertoire `/usr/local/scilab-2.6`). Vous donnerez les mêmes droits à ce répertoire qu'aux répertoires situés dans `.../macros`.
2. Rapatriez toutes vos fonctions, et mettez les dans ce répertoire nouvellement créé.
3. Vous devez maintenant compiler toutes les fonctions :
   Dans la fenêtre de *Scilab* exécutez la commande :
   `genlib('jojolib','SCI/macros/jojo')`
4. Ouvrez avec un éditeur de texte (`emacs` par exemple) le fichier texte `scilab.star` fichier située dans `SCI` et à la ligne 58 inscrivez :
   `load('SCI/macros/jojo/lib')` en ouvrant à nouveau *Scilab* vous aurez maintenant accès à toutes les fonctions *Scilab* contenues dans le nouveau répertoire. Vous pouvez maintenant avec l'instruction **who** voir que la bibliothèque `jojolib` est à votre disposition. Cette façon de procéder s'applique aussi bien avec Linux que Windows95, 98, mais pas avec Windows Millenium semble t'il.

Annexe C Un peu de calcul matriciel

Le but de cette annexe est de proposer quelques exercices d'algèbre linéaire très utiles pour l'automaticien.

C.1 Vecteurs, matrices

Vecteurs de nombres

Dans Scilab on peut définir des vecteurs, des matrices de nombres, réels, complexes, booléens, des vecteurs et matrices de polynômes, de fractions rationnelles et même de chaînes de caractères.

```
-->v=[2,-3+%i,7]
v  =
!   2.   - 3. + i       7. !
```

Création d'un vecteur ligne : des crochets, les éléments de la ligne séparés par des virgules ou des blancs.

```
-->v'
ans  =
!   2.            !
! - 3. - i        !
!   7.            !
```

Calcul de vecteur transposé : le '.

```
-->w=[-3;-3+%i;2]
 w  =
! - 3.         !
! - 3. + i     !
!   2.         !
```

Un vecteur colonne : des crochets, les éléments de la colonne séparés par des points virgules.

```
-->v'+w
ans  =
! - 1. !
! - 6. !
!   9. !
```

Somme de deux vecteurs colonnes

```
-->v*w

ans  =

   16. - 6.i
```

```
Produit d'une ligne par une colonne, c'est un sca-
laire.
-->w'.*v
ans  =
! - 6.    10.    14. !
```

Produit élément par élément d'une ligne par une ligne : signe `.*`.

Les éléments constitutifs des vecteurs lignes sont séparés par une virgule ou un blanc (espace), tandis que pour un vecteur colonne, on utilise le point virgule. Ces signes différentient un vecteur ligne d'un vecteur colonne. La matrice vide est représentée par `[]`, elle n'a aucune ligne, aucune colonne. On notera que la transposition d'une matrice se fait en utilisant le signe `'`. Ce signe est aussi utilisé pour calculer le complexe conjugué d'un nombre. Les opérations de multiplication et de division sont obtenues par les signes `*` et `/` ; elles concernent les scalaires, les vecteurs, les matrices.

Quant aux signes `.*` et `./`, ils représentent la multiplication et division, élément par élément, utiles pour les vecteurs et matrices. Quand vous définissez un vecteur ligne faites attention à la position des blancs . L'exemple ci–dessous l'illustre.

```
-->v=[1 +3]//on pouvait faire v=[1,+3]
  v  =
 !   1.    3. !
-->w=[1 + 3]
  w  =
     4.
-->w=[1+3]
  w  =
     4.
```

Faites ici attention à la position des espaces !

On peut aussi construire un vecteur d'une manière incrémentale.

```
-->v=5:-.5:3
 v  =
!   5.    4.5    4.    3.5    3. !
```

Le vecteur résultat commence par la première valeur donnée, ici `5` et les éléments de la ligne sont incrémentés de `-.5`, jusqu'à la valeur `3`.

Si l'incrément est égal à un on peut se contenter de l'instruction:

```
-->i=1:5
 i  =
```

```
! 1.  2.  3.  4.  5. !
```

Deux instructions spéciales ones et zeros définissent des vecteurs constitués de uns ou de zéros.

```
-->v=[1 5 6]
 v  =
!   1.    5.    6. !
-->ones(v)
 ans  =
!   1.    1.    1. !
-->ones(v')
 ans  =
!   1. !
!   1. !
!   1. !
-->ones(1:4)
 ans  =
!   1.    1.    1.    1. !
-->3*ones(1:4)
 ans  =
!   3.    3.    3.    3. !
-->zeros(v)
 ans  =
!   0.    0.    0. !
-->zeros(1:5)
 ans  =
!   0.    0.    0.    0.    0. !
```

On remarquera que les deux instructions précédentes remplacent un vecteur quelconque, ici v , par un vecteur de même dimension constitué de uns ou de zéros.

Les matrices constantes

Les éléments d'une ligne de la matrice sont séparés par des virgules ou des espaces, les éléments d'une colonne par points virgules. La multiplication de matrices par des scalaires, des vecteurs ou par d'autres matrices ce fait d'une manière habituelle. L'addition et la soustraction se fait élément par élément, la multiplication et la division se font d'une manière habituelle et utilisent respectivement les opérateurs * et / . Ne pas confondre avec la multiplication élément par élément que l'on verra par la suite.

```
—>A=[2 1 4;5 -8 2]
 A =
! 2.  1.  4.!
! 5. -8.  2.!
—>b=ones(2,3)
 b =
! 1.  1.  1.!
! 1.  1.  1.!
```

On définit la matrice dont les éléments sont des uns. Les instructions ones et zeros (matrice de zéros)peuvent être utilisées avec des matrices rectangles.

```
-->A.*b
 ans  =
!   2.      1.     4. !
!   5.    - 8.     2. !
```

Ici on fait le produit élément par élément, de la matrice A par b.

```
-->A*b'
 ans  =
!    7.        7. !
!  - 1.      - 1. !
```

On notera que l'instruction ones qui possède ici deux arguments séparés par une virgule, crée une matrice de dimensions égales aux arguments. Ceci est aussi valable pour l'instruction zeros. On peut aussi construire une matrice de zéros ou de uns de même dimension qu'une matrice précédemment définie par les instructions :

```
-->b=ones(A)
-->c=zeros(A)
```

Si l'on doit créer une matrice (ou une instruction)de grande dimension, on peut écrire (cette matrice, cette instruction) sur plusieurs lignes en mettant à la fin de chaque ligne trois points.

```
-->U=[1 2 3 0 0;...
-->1 2 5 0 0;...
-->1 2 5 0 0]
 U  =
!   1.    2.    3.    0.    0. !
!   1.    2.    5.    0.    0. !
!   1.    2.    5.    0.    0. !
```

Une autre instruction intéressante est l'instruction eyes. Voici deux exemples. Par exemple la matrice unité de rang 4.

```
-->c=eye(4,4)
c  =
!  1.  0.  0.  0. !
!  0.  1.  0.  0. !
!  0.  0.  1.  0. !
!  0.  0.  0.  1. !
```

La matrice unité définie par l'instruction eyes(n,m) .

```
-->c=eye(3,2)
c =
!  1.  0. !
!  0.  1. !
!  0.  0. !
```

La matrice unité de même dimensions que la matrice A.

```
-->I=eye(A);
```

Définir une matrice diagonale dont les éléments sont les composantes d'un vecteur, ici le vecteur b

```
-->b=[2 1 4 5 -8 2]
 b  =
!   2.    1.    4.    5.  - 8.    2. !
-->B=diag(b)
 B  =
!   2.    0.    0.    0.    0.    0. !
!   0.    1.    0.    0.    0.    0. !
!   0.    0.    4.    0.    0.    0. !
!   0.    0.    0.    5.    0.    0. !
!   0.    0.    0.    0.  - 8.    0. !
!   0.    0.    0.    0.    0.    2. !
```

On peut extraire le vecteur, diagonale principale d'une matrice, en appliquant la même instruction.

```
-->c=diag(B)
 c  =
!    2. !
!    1. !
!    4. !
!    5. !
```

```
!  - 8.  !
!    2.  !
```

De même il existe deux instructions `triu` et `tril` (pour triangulaire supérieure et triangulaire inférieure), (upper, lower), pour extraire respectivement la partie triangulaire supérieure et la partie triangulaire inférieure d'une matrice. Attention, ne pas confondre ces deux instructions avec l'instruction `trianfml` qui effectue une triangularisation, en faisant une série de combinaisons linéaires sur les lignes.

Une matrice utile est la matrice `rand(?,?)` qui génère une matrice de nombres pseudo–aléatoires suivant une loi uniforme sur l'intervalle [0,1[.

```
-->ALE=rand(2,3)
 ALE  =
!   0.5442573    0.2312237    0.8833888  !
!   0.2320748    0.2164633    0.6525135  !
```

Une autre particularité du logiciel réside dans le fait que l'on peut utiliser les fonctions d'algèbre classique avec des matrices : on applique ces fonctions à chacun des éléments de la matrice.

Exemples :

```
-->ALE=rand(2,3);
-->SALE=sqrt(ALE)
 SALE  =
!   0.5546252    0.4632502    0.6013619  !
!   0.9658994    0.5591440    0.5405799  !
-->EALE=exp(ALE)
!   1.3601692    1.239367    1.4356764  !
!   2.5420266    1.367032    1.3394066  !
```

On a défini deux matrices dont les éléments sont les racines carrées et les exponentielles de chacun des éléments de la matrice de départ.

Mais il existe en plus dans certains cas, des fonctions de matrice comme l'exponentielle d'une matrice carrée. Le nom de ces fonctions se termine par un m.

```
-->ALE=ALE([1 2],[1 2])
 ALE  =
!   0.3076091    0.2146008  !
!   0.9329616    0.312642   !
```

J'ai extrait de la matrice une sous matrice carrée.

```
-->TM=tanm(ALE)
```

```
 TM  =
!   0.4007827    0.2599590 !
!   1.130153     0.4068794 !
-->EX=expm(ALE)
 EX  =
!   1.4988522    0.3024921 !
!   1.3150629    1.5059464 !
-->SQM=sqrtm(EX)
 SQM  =
!   1.1955973    0.1263459 !
!   0.5492800    1.1985604 !
```

Les matrices, vecteurs, de chaînes de caractères

Comme pour les vecteurs et matrices de scalaires on peut définir des vecteurs et matrices de chaînes de caractères.

On utilise pour cela des apostrophes simples ou doubles. De même qu'il existe des fonctions traitant des mtrices de scalaires, dans le logiciel, il existe des fonctions spéciales manipulant des matrices de chaînes de carctères.

```
-->A=['x','y";"z','w+v']
 A  =
!x  y     !
!         !
!z  w+v   !
-->AT=trianfml(A)
 AT  =
!z  w+v           !
!                 !
!0  z*y-x*(w+v)   !
-->x=4;y=-2;z=5;w=1;v=8;
-->EVAL=evstr(AT)
 EVAL  =
!   5.    9.  !
!   0.  - 46. !
```

On a défini une matrice de chaînes de caractères, puis on a réalisé une triangularisation de cette matrice, et enfin on évalue la valeur de cette matrice.

Vous verrez qu'il existe de nombreuses fonctions traitant les matrices

de chaînes de caractères, ceci permet de créer et manipuler des fonctions.

Les matrices, vecteurs, de polynômes et de fractions rationnelles

Comme nous l'avons vu au cours de ce document, *Scilab* est capable de manipuler des vecteurs, des matrices de polynômes et de fractions rationnelles. Je n'insisterais donc pas sur ce point.

C.2 Calcul matriciel

A faire

C.3 Application à l'automatique

A faire

Bibliographie

[1] SCILAB GROUP, "*Scilab reference manuel*",

[2] J–C. GILLE, M. PELEGRIN, *Théorie et technique des asservissements*, DUNOD, PARIS , 1956.

[3] P. BORNE, D. DAUHIN–TANGUY, J.P. RICHARD, F. ROTELLA, I. ZAMBETTAKIS, *Analyse et régulation des processus industriels*, tome 1, Régulation continue, EDITIONS TECHNIP, PARIS, 1993.

Les logiciels de simulation en automatique. Quelques exercices d'automatique avec le logiciel Scilab.