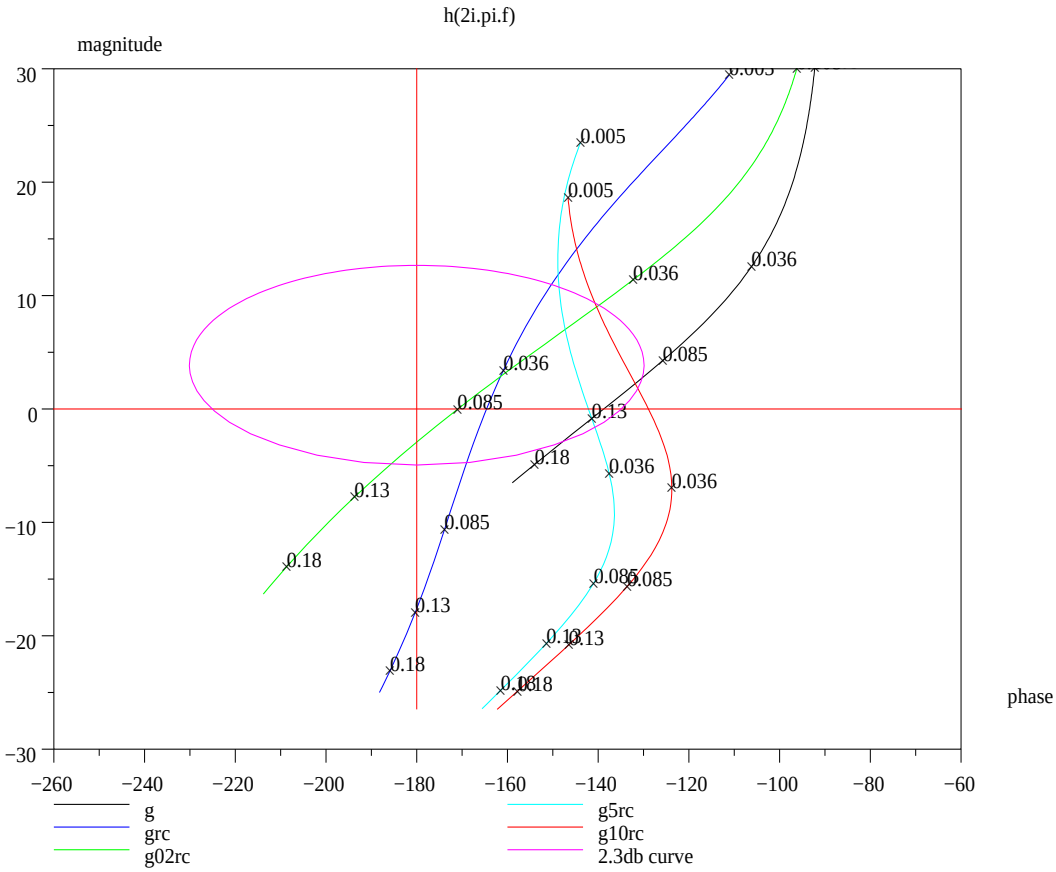
Les logiciels de simulation en automatique. Exercices d'automatique avec le logiciel SCILAB

LUCIEN POVY
lucien.povy@free.fr

Mars 2OO7. Version 2



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

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

Remarques : Ce texte a été écrit avec l'éditeur de documents LyX, logiciel libre et gratuit. Ce document et les programmes associés sont protégés par les licences GPL et LGPL, voir la conclusion en fin de rapport. Voici les logiciels qui ont été utilisés pour la simulation et la rédaction de ce document (sous Linux) :
Scilab versions 2.4 puis 2.5, 2.6 (version 2.7 depuis février 2003) et maintenant la version 4.1.
Latex2ϵ avec Lyx (dernière version : 1.4.3)
Xfig versions 3.2.4, 3.2.5 (logiciel de dessin vectoriel).
Gv depuis la version 3.5.8 jusqu'à la version 3.6.1 (visualisateur postscript et pdf), ou acrobat reader.

Avant propos

Ce livre a pour but de donner les principales méthodes utilisées dans le domaine de l'automatique analogique (étude des systèmes à modèles linéaires continus) et d'approfondir par des exercices de simulation cette discipline. Vous trouverez dans ce document un résumé du cours que j'enseigne (que j'ai enseigné) en première année dans le département IMA de l'EUDIL[1].

En écrivant ce polycopié j'ai voulu conforter avec un logiciel de simulation, ici Scilab (libre et gratuit), cet enseignement d'automatique. En effet il est impensable de demander aux étudiants d'acheter le logiciel professionnel Matlab qui est très cher, même si c'est le logiciel le plus connu et le plus utilisé.

Je voudrais dire ici que Scilab, bien qu'il soit libre et gratuit, n'est pas un logiciel bricolé : il est robuste car réalisé par des professionnels de l'informatique et de l'automatique, on le doit à un Institut de recherche français l'INRIA en collaboration avec l'ENPC[2]. Je pense que ce document peut être utile pour les débutants et même les experts en automatique analogique ; vous pouvez bien entendu le diffuser librement et gratuitement, cela va de soit (ceci s'adresse aux enseignants et étudiants), sous toute forme de support possible. De même, si vous souhaitez le compléter, le corriger, apporter vos propres remarques sur ce document et les méthodes de simulation qui lui sont attachées, vous voudrez bien m'en informer à l'adresse électronique donnée, je répondrais à tout courrier électronique relatif aux questions traitées dans ce document.

Il est souhaitable d'installer la bibliothèque de fonctions `autoelem` que je propose, afin de refaire les exercices qui sont dans le document. De même si vous avez une certaine compétence en langues (Anglais, Allemand, Espagnol ...) vous pouvez traduire, sans en faire commerce, ce polycopié. Merci d'avance à tous les utilisateurs de me donner le retour.

[1] L'EUDIL est mort, vive EPU de LILLE. Ecole Universitaire Polytechnique de Lille, cité Scientifique ; 59655 Villeneuve d'Ascq Cedex.

[2] INRIA : Institut National de Recherche en Informatique et Automatique, domaine de Voluceau-Rocquencourt ; BP 105, 78153 Le Cheynay.
ENPC : Ecole Nationale des Ponts et Chaussées, Cité Descartes ; 77455 Marne-la Vallée Cedex 02.

Table des matières

1 Les logiciels de simulation et l'automatique

Il existe à ma connaissance[1], 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 (pas tous), divers UNIX, dont LINUX.

1.1 Matlab

On trouvera des renseignements actualisés concernant Matlab sur le site internet : `http://www.mathworks.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.

De plus 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.no`.

[1] Il en existe d'autre, Xmath ... et quelques logiciels moins complets : comme Sirena. Je ne parle pas ici des logiciels de mathématique symbolique, comme Mathematica, Maple ou Mupad. Ce dernier est gratuit pour une plateforme de type Linux pour l'éducation (version d'essai pour Win$$).

Depuis que la version 2.5 de Mupad est sortie, on peut entre autre utiliser Scilab dans une fenêtre Mupad.

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++, couvert par une licence libre, licence GPL[2], et gratuit. Le site internet d'Octave se nomme : `http://www.octave.org`.

1.2.1 Avantages

Logiciel non commercial, avec sa licence GPL qui donne automatiquement accès au code source, vous pouvez l'essayer en allant sur le site de ce logiciel et en l'installant sur votre machine, il existe une version pour le système d'exploitation Window : je ne la connais pas.

1.2.2 Inconvénients

Ce logiciel n'a plus été développé pendant un certain temps mais une nouvelle équipe de développeurs a pris le relais : à essayer.

1.3 Scilab

C'est un logiciel écrit par des français travaillant à l'INRIA Institut National de Recherche en Informatique et Automatique en collaboration avec l'ENPC, Ecole Nationale de Ponts et Chaussées ; ce logiciel a une licence proche de la licence GPL est libre et gratuit, de plus vous pouvez obtenir de l'aide par le réseau sur le site de Scilab, `http://www.scilab.org`. De même sur ce site vous trouverez un lien vers le site ftp de Scilab, où vous pourrez télécharger le logiciel, soit les sources, soit différents binaires pour diverses plateformes informatiques. Vous trouverez aussi sur le site de Scilab, de nombreuses contributions vous permettant de traiter de nombreux problèmes. La dernière version stable du logiciel est maintenant la version 4.1 : il existe aussi une version de développement mise à la disposition des curieux et téméraires.

1.3.1 Avantages

Ce logiciel est très complet et est très proche de Matlab quant à sa syntaxe (voir les exercices proposés : comparaison des instructions et de la progammation), de même de nombreuses boîtes à outils sont disponibles dans le logiciel ou dans les contributions de divers auteurs. Son développement se poursuit toujours. De plus et ici je m'adresse aux étudiants et peut être à d'autres personnes : vous ne serez pas tenté de récupérer sur internet le logiciel professionnel précédent et vous mettre ainsi dans l'illégalité ...

[2] Vous trouverez le texte de cette licence à la fin du document : Annexe 4.

1.3.2 Inconvénients

C'est un logiciel libre et gratuit, il ne vaut donc rien[3], 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$$.

1.4 Mise en place de Scilab sur un PC-LINUX

Même si sur le site de Scilab vous pouvez trouver le logiciel Scilab sous différentes versions compilées, il me semble plus judicieux pour les utilisateurs de Linux, de compiler le logiciel et ainsi vous vous rendrez compte de la structure du logiciel et des problèmes que peut poser la compilation d'un gros logiciel.

1.4.1 Installation à partir d'un binaire

Vous devez rapatrier le binaire de Scilab sur le site de SCILAB, `http://www.scilab.org`, l'archive (la version binaire) se nomme `scilab-N°.X.bin.linux-i686.tar.gz` ; Depuis un répertoire temporaire, `/home/monrepertoire/temp` par exemple, décompressez dans un terminal X (xterm) le fichier Scilab par la commande :
`tar xvzf scilab-N°.X.bin.linux-i686.tar.gz`, la commande Linux `tar` va vous créer une archive `scilab-N°.X`.

Pour installer le logiciel passer sous le compte administrateur (`root`) et taper dans ce terminal les commandes : `cd scilab-N°.X` puis `make` : normalement le logiciel va être installé dans le répertoire `/usr/lib`. Un lien sera crée dans le répertoire `/usr/bin` qui est toujours dans votre variable d'environnement (votre PATH) ; de même vous pouvez créer un raccourci sur votre bureau : ce raccourci pointra vers le script `/usr/bin/scilab`. Vous pouvez maintenant effacer le fichier compressé que vous avez rapatrié. Un conseil : lisez le fichier `README_Unix` situé dans le répertoire principal du logiciel.

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

Si maintenant vous souhaitez compiler puis installer le logiciel Scilab vous devez avant tout vérifier si les logiciels suivants sont installés sur votre machine. Vous devez disposer d'un compilateur C (`gcc`) et d'un compilateur fortran77 (`g77`),[4] de l'interface graphique Xwindow, de `tcl` et `tk`, éventuellement du logiciel `pvm` et surtout du logiciel

[3] Remarque à prendre au premier degré bien sûr ! Mais il y a tellement de rumeurs sur les logiciels libres ! Et l'Éducation Nationale et la Recherche (que j'ai fréquenté pendant près de 40 ans) aimant tant les logiciels propriétaires, pas que ces organismes ... et pratiquement tous les organismes de l'état et vous aussi lecteur sans doute.

[4] Il arrive souvent que les bibliothèques de développement du serveur X soient manquantes sur votre linux. Avec un binaire précompilé, vous pouvez voir les bibliothèques dynamiques nécessaires : placez vous en mode terminal dans le répertoire `bin` de scilab et faites `ldd scilex`.

`xsltproc`[5] avec, si vous le voulez, `xmlto`. De même installez le compilateur java (logiciel `javac`).
Faites aussi attention à l'endroit où sont installées les bibliothèques `tcl` et `tk` (voir plus loin les options de configuration de Scilab : relisez le `README_Unix` situé dans le fichier `scilab-N°.X`).

1.4.3 Compilation du logiciel, installation

Compilation

Sous utilisateur habituel, pour compiler le logiciel Scilab, placez vous dans le répertoire racine de Scilab, par exemple dans le repertoire `/home/monrepertoire/tmp/scilab-N°.X` et exécutez dans un terminal X la commande : `./configure` ; vous pouvez rajouter des options à `configure`. Pour cela voyez (j'insiste) le fichier `README_Unix` 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`, (`SCI` est le répertoire `/home/monrepertoire/scilab-N°.X`). Pour démarrer le logiciel, de ce répertoire, cliquez sur le script `scilab` situé dans le répertoire `/SCI/bin`.
Voici un exemple d'instructions exécutées dans un terminal X permettant de configurer et compiler le logiciel.

```
cd /home/monrepertoire/tmp/scilab-4.1
./configure -prefix=/usr/local --without-pvm
make all
cd examples
make tests
```

Installation

Quant à l'installation elle consiste à passer sous mode administrateur (`su root`), à revenir dans le répertoire racine de Scilab , par la commande `cd ..`(pour revenir du répertoire `examples` au répertoire `SCI/scilab-4.1`) et à faire : `make install`.[6]

D'après la commande `./configure` précédente Scilab va être installé dans le répertoire `/usr/local/lib` et un lien (`scilab`) va être créer dans le répertoire `/user/local/bin`. Vous avez accès en tant qu'utisateur normal au logiciel en tapant dans un terminal la commande `scilab` (de même vous pouvez créer un raccourci sur votre bureau).

1.4.4 Quelles bogues connus, comment y remédier, quelques améliorations

Je ne parlerais ici que de quelques petits bogues ou imperfections que j'ai repérés dans certains programmes situés dans des sous répertoires du répertoire `SCI/macros`,

[5] Sinon vous n'aurez pas les pages de manuel
[6] On peut aussi désinstaller le logiciel en faisant depuis le répertoire racine de Scilab : `make uninstall`.

ces bogues, ces imperfections, concernent des fonctions utilisées en automatique classique : vous trouverez dans le corps du document des remarques à ce sujet.

Voici une petite liste des imperfections :

- fonction `p_margin()` dans `SCI/macros/auto`
- fonction `phasemag()` dans `SCI/macros/auto` à ne pas l'utiliser. Préférer la fonction nouvelle `dbphifr()` qui sera mise dans un nouveau répertoire par exemple `SCI/macros/autoelem` (voir ce qui suit)[7].
- fonction `horner()` dans un cas (correction dans le répertoire `autoelem`).

Vous trouverez de même, avec ce document, un nouveau répertoire nommé `autoelem` contenant quelques macros permettant de traiter les réponses fréquentielles des systèmes continus linéaires avec ou sans retard pur, avec des fonctions amenant en plus des vecteurs gain et phase du système, un vecteur gain et un vecteur déphasage fonctions de la fréquence, (ceux-ci sont donnés par une macro définie dans un fichier). Ce nouveau répertoire pourra être ajouter au répertoire `SCI/macros` (Voir Annexes 1 et 2).

1.4.5 Notes sur les logiciels LYX et XFIG

Le document que vous avez sous vos yeux a été écrit avec le logiciel **LYX** `http://www.lyx.org` . Je n'aurais jamais pu écrire un tel document avec un traitement de texte classique : il fallait utiliser un logiciel comme **SCIENTIFIC WORD** `http://www.ritme.com/fr/index.html` mais vu le prix pour une licence monoposte!!!

Je donne en même temps que le document au format `.pdf` le fichier source au format `.lyx` et au format `.tex` (que l'on obtient avec le logiciel **LYX**), ce qui vous permettra de complèter ce document à votre convenance, et si vous avez les moyens, de le traduire en une autre langue et de le diffuser. Ce logiciel s'installe facilement sur le système d'exploitation Linux et même sous Windows : allez sur le site de **LYX** . Enfin un site intéressant pour les francophones doit être mentionné `http://yann.morere.free.fr/lyx/configure.html` . Pour les utilisateurs du système Windows allez voir le site `http://wiki.lyx.org/Windows/Windows` .

Quant au logiciel de dessin vectoriel **XFIG**, il permet de retoucher et complèter les figures créées par Scilab, en effet toute figure scilab peut être sauvegardée au format `.fig`. Avec ce document je donne aussi les fichiers `.fig`, fichiers issues souvent de scilab qui ont été complètés pour une meilleure compréhension du document. Ce logiciel de dessin vectoriel semble « vieux » mais il est d'une efficacité remarquable vu son âge et sa petitesse : à consommer sans modération! `http://www.xfig.org`

[7] Depuis la version 2.7 de Scilab ce programme a été réécrit, mais pour certains systèmes il y a encore un bogue sur la phase, je justifierai cela au cours de l'exposé.

2 Exercices d'algèbre avec Scilab

Le but de ce chapitre n'est pas de refaire un cours d'automatique élémentaire, ni de développer les résultats acquis dans un cours d'algèbre des polynômes et des matrices, mais d'illustrer les principaux résultats d'algèbre qui seront nécessaires à l'analyse d'un système en utilisant un logiciel de simulation.

Nous admettrons comme connue la définition des systèmes à modèle linéaire continu ; de même les principaux résultats concernant l'outil mathématique permettant d'étudier ces systèmes, à savoir la transformée de Laplace monolatère, seront aussi admis. Dans ces conditions le modèle mathématique pourra être : soit une fonction de transfert (cas monovariable), une matrice de transfert (cas multivariable), ou des équations d'état. Dans tous les cas, mathématiquement, on est appelé à manipuler des polynômes (polynômes, vecteurs de polynômes, matrices de polynômes), des fractions rationnelles (sous forme de vecteurs et matrices éventuellement), des vecteurs et matrices de scalaires (si la modélisation sous forme de variables d'état a été préférée).

Dans ce chapitre nous donnerons des exemples de manipulations de vecteurs, matrices (de scalaires, de polynômes, de fractions rationnelles) et utiliserons les principales fonctions contenues dans le logiciel permettant de définir les systèmes linéaires continus, fonctions qui mettent en évidence les principales propriétés de ces systèmes.

2.1 Démarrage de Scilab

Comme je l'ai signalé précédemment, en frappant dans un terminal X, `scilab`,[1] 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. Pour ceux qui ne peuvent pas se passer des systèmes d'exploitations de la société Microsoft vous trouverez un exécutable `scilab.exec` (déjà compilé par Scilab.org) sur le site de cet organisme, de même vous voudrez bien lire le fichier texte `Lisez-moi-Windows.txt` associé au logiciel.

2.2 Les différentes 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 symbolisé par `--->`, à frapper les lignes de commande les unes après les autres, en frappant la touche `entrée` après chaque ligne de programme.

[1] Scilab peut être démarré avec plusieurs options par exemple : `scilab -l fr` pour démarrer Scilab en français.

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.

Une remarque sur les séparateurs d'expressions : il existe deux séparateurs d'expressions qui sont le point-virgule « ; » et la virgule « , » le premier comme on vient de le dire, annule l'affichage du dernier résultat de la commande qui précède (même s'il y a deux commandes) et est un séparateur d'expression. Quant à la virgule, elle sépare deux expressions mais les résultats sont tous les deux affichés.

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` ou `LyX`[2], et une fenêtre Scilab.

La seconde manière est d'utiliser un éditeur de texte, `emacs`, `kedit` ou `kwrite` pour sa coloration syntaxique (sur linux) ou `wordpad` (sur windows$$)[3] : vous allez créer ainsi un fichier texte, que vous nommerez `mon_progr.sce`, que vous ferez exécuter par la commande : `exec('path/mon_progr' [,mode])`[4].

Voici un exemple réalisé par un étudiant de programme permettant d'illustrer cela :

```
tau=input('donner une valeur à tau :')
k=input('donner une valeur a 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()
//instruction non obligatoire
//reponse indicielle unitaire
xsetech([0.,0.,0.5,0.5]) ; plot2d(temps',h')
//black
xsetech([0.5,0.,0.5,0.5]) ;
bblack(sys,.01,100,'premier_ordre')
//nyquist
xsetech([0.5,0.5,0.5,0.5]) ;
nnyquist(sys,.01,100,'premier_ordre')
```

[2] La première version de ce document à été écrite avec le logiciel `Thot`, mais comme ce logiciel n'est plus développé, je me suis mis au logiciel de formatage de document `LyX` qui permet d'écrire des gros documents à la `LaTeX2`ε (et fournir un fichier `.tex` entre autre) à consommer sans modération : on n'a plus à retenir ou à apprendre le language de balisage de `LaTeX2`ε.

[3] Depuis la version 2.7 de Scilab vous avez un petit éditeur avec Scilab.

[4] Le drapeau `mode`, prend une valeur numérique entière valant : 0, -1, 1, 2, 3, 4, 7 voir l'instruction `exec` ou `mode` dans l'aide ; vous pouvez aussi faire dans une fenêtre Scilab `apropos mode`.

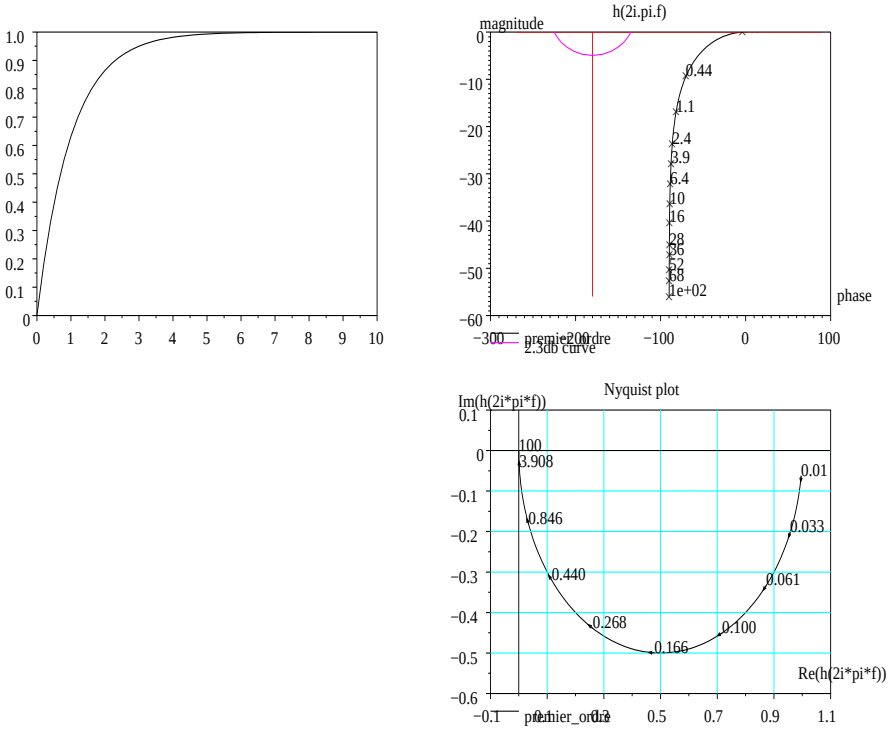


FIG. 2.1: Premier programme.

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 par des commandes copier-coller.

Vous remarquerez l'instruction `xsetech()` qui permet ici, dans une même fenêtre graphique[5], 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 (FIG. 2.1).

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. Ce programme sera commenté dans la suite de l'exposé.

[5] J'ai sauvé ce graphique sous un format `.eps`.

2.3 Définir un polynôme par ses racines, ses coefficients. Valeur numérique d'un polynôme

Avant de définir un système linéaire nous allons faire quelques exercices sur les polynômes et les fractions rationnelles : nous faisons des mathématiques.

2.3.1 Polynôme, racines d'un polynôme

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`. Attention ne confondez pas `'s'` avec le polynôme s : `'s'` est une chaîne de caractères[6], s un polynôme.

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 « `%` ». En plus des deux variables réservées précédentes, le logiciel offre des constantes spéciales réservées `%i, %pi, %e, %eps`. La constante `%i` est le nombre complexe pur $\sqrt{-1}$, `%pi` est le nombre $\pi = 3,1415927$, `%e` est la constante trigonométrique $e = 2,7182818$, `%eps` représente la précision de la machine : ici $2,22E-16$. Quant aux symboles `%inf` et `%nan`, ils représentent respectivement l'infini et NotANumber et le symbole `[]` est l'élément vide (rien).

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.

[6] Pour définir une chaîne de caractères on utilise la syntaxe 'machaîne' ou "machaîne" ou encore 'machaîne". On met les caractères entre des simples ou doubles quotes (on peut mélanger les deux).

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 « // ».

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

Les racines de ce polynôme sont : $\begin{bmatrix} -1 & -2 & -3 \end{bmatrix}$. 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=[-1,-3 -4]`, on peut **remplacer la virgule par un espace** pour séparer les éléments d'un vecteur ligne.

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

On obtient le même résultat. Profitons de ce programme 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 $(s - s_1)(s - s_2)$ ... où s_1, s_2 ... sont les racines du polynôme[7]. Vous remarquerez que Scilab ordonne un polynôme dans le sens des puissances croissantes.

```
-->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=[-1;-2;-3]` : la présence du « ; » est obligatoire.

```
-->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[8] .

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

[7] Dans ce cas les instructions `poly` et `root` sont des instructions inverses. Comme `poly` et `coeff` peuvent l'être quand on définit un polynôme par ses coefficients.

[8] Si on transpose une matrice A à coefficients complexes, l'opérateur « ' » réalise la transposition de la matrice conjuguée de A.

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).

Définissons maintenant le polynôme caractéristique de la matrice `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!
!  16.4339810!
```

En faisant `roots()`[9] 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.

2.3.2 Autres paramètres caractérisant un polynôme

Reprenons l'exemple précédent pour caractériser notre polynôme et introduire l'instruction `type()` : le type d'un objet.

```
-->s=poly(0,'s");num=poly([-1,-2,-3],"s",'r');
-->type(num)
 ans =
   2.
-->typ=typeof(num)
 typ =
   polynomial
-->type(typ)
 ans =
   10.
```

[9] Le programme `roots` calcule toutes les racines d'un polynôme de degré n. Mais attention si vous avez un polynôme avec une racine multiple ordre de multiplicité >3 vous aurez du mal à retrouver la racine multiple exactement : ceci est du à la précision de calcul de l'algorithme utilisé. Je reviendrais sur ce problème. (Vous pouvez aussi normaliser le polynôme en faisant un changement de variable adéquat).

Explicitons cette notion de type : la première instruction renvoie un entier, ici 2, qui caractérise un polynôme, un vecteur, une matrice de polynômes. La seconde instruction renvoie une chaîne de caractères suffisamment explicite. Quant à la dernière instruction elle renvoie l'entier 10 qui est du type chaîne de caractères. N'ayant pas défini la variable de retour, Scilab propose comme variable de retour `ans` (answer en anglais).

```
-->deg=degree(num)
 deg =
   3.
-->type(deg)
 ans =
   1.
-->varia=varn(num)
 varia =
   s
-->type(varia)
 ans =
   10.
```

Par les instructions `degree()`, `type()`, `varn()`, on détermine respectivement le degré du polynôme, le type du nombre degré, qui est ici un scalaire (un scalaire, un vecteur, une matrice de réels ou complexes est de *type*1) et la variable du polynôme qui est le caractère (chaîne) `s`, (*type*10) : on explicitera (en particulier pour les listes) cette notion de type dans la suite de l'exposé.

2.3.3 Vecteur de polynômes, matrices de polynômes

Comme pour les scalaires, Scilab sait reconnaître les vecteurs et matrices (pas forcément carrées) de polynômes. De même Scilab sait faire les opérations élémentaires (additions, soustraction, multiplication, élévation à la puissance, si cette opération est justifiée ...) sur ces vecteurs et matrices de polynômes ; voici un exemple :

```
-->s=%s ;
-->num=[1+s,3+s^2,s;6+s+s*s,4,7-s]
 num =
!                   2        !
! 1 + s        3 + s    s     !
!                             !
!          2                  !
! 6 + s + s    4        7 - s!
-->num^2
     !--error 20
first argument must be square matrix
```

Ici j'ai voulu élever une matrice au carré, elle était rectangulaire, donc l'opération n'était pas valide, Scilab me l'a signalé, il renvoie une erreur numérotée avec une explication.

```
-->num.^2
 ans =
!           2                    2   4     2         !
! 1 + 2s + s              9 + 6s + s     s          !
!                                                   !
!               2   3   4                          2!
! 36 + 12s + 13s +2s + s     16       49 - 14s + s  !
```

Dans cet exemple au lieu de faire opération puissance, j'ai avec l'opération `.^`, réalisé l'opération puissance (entière) élément de la matrice par élément de la matrice.

2.3.4 Valeur numérique d'un polynôme, changement d'argument

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 rationnelles.

```
-->s=%s ;nA=-40-14*s+s*s*s ;
-->valpi=horner(nA,%i)
 valpi =
   -40. - 15.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). Nous donnons un autre exemple permettant de transformer un polynôme, une fraction rationnelle, par changement de variable (polynôme, fraction rationnelle), en un polynôme ou une fraction.

```
-->valch=horner(nA,[1/s,(1-s)/(1+s),s^3])
 valch =
  column 1 to 2
!              3                                  2     3!
! 1 - 14s - 40s               - 53 - 137s + 103s + 27s  !
! -------------               ------------------------  !
!         3                                   2   3     !
!        s                          1 + 3s + 3s + s     !
  column 3
!                3     6   9!
! 7.322E-15 - 40s - 14s + s  !
! -------------------------  !
!              1             !
```

Très intéressante cette dernière commande avec Scilab : c'est déjà du calcul symbolique. On reviendra sur les fractions rationnelles (rapport de deux polynômes). Dans l'exemple choisi le polynôme `nA` vaut : $nA = -40s - 14s^2 + s^3$. Quant au deuxième argument

d'entrée de la fonction `horner()` c'est un vecteur ligne de polynômes et fractions rationnelles, vecteur de valeur :

$$\left[\begin{array}{ccc} \frac{1}{s} & \frac{1-s}{1+s} & s^3 \end{array} \right]$$

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.[10]

```
-->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.

On peut aussi avec l'instruction `horner()` faire un changement de variable, voici un exemple :

[10] Dans la dernière version de Scilab l'instruction `horner` ne marche plus avec l'exemple ci dessous : il faut supprimer la ligne 15 du programme horner (situé dans `SCI/macros/calpol`) pour que ce programme s'exécute. (Il semble que le deuxième argument de la fonction d'appel doit être un scalaire et non un vecteur ; le premier argument pouvant être une matrice ? contradiction entre la page de manuel et les commentaires de la fonction `horner.sci`)

```
-->s=%s ;
-->w=poly(0,'w')//on introduit une autre variable
 w =
  w
-->fr=1+s+s*s
 fr =
            2
   1 + s + s
-->frw=horner(fr,(w+1)/(w-1))
 frw =
        2
  1 + 3w
  ----------
            2
  1 - 2w + w
```

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.

```
-->result=freq(nA,1,[1,.1*%i,3])
 result =
! - 53.  - 5.401i  - 135. !
```

Vous ferez attention en utilisant l'instruction `freq()` : elle demande d'avoir deux polynômes (matrices de polynômes), numérateur puis dénominateur pour ses deux premiers arguments d'entrée : c'est pour cela que j'ai mis le scalaire `1` (Scilab l'accepte), comme deuxième argument. En toute rigueur (pour respecter le type des arguments d'entrée) je devais écrire :
`result=freq(nA,poly(1,'s','c'),[1,.1*%i,3])`.

2.3.5 Définition d'un polynôme par ses coefficients

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
```

Le troisième argument de l'instruction `poly( )` est obligatoire : c'est le caractère `'c'` ou `"c"`. 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` obtenu avec l'opérateur `'`, 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")//ou s=%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. !
-->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 des deux premières lignes de la table de Routh[11], 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 une matrice comprenant deux vecteurs lignes, `r1`, `r2`. Le premier, par l'instruction `coe(long:-2:1)`, va être constitué d'un vecteur ligne en

[11] Vous verrez cette table lors de l'étude de la stabilité des systèmes : paragraphe **5.1.3**

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 en exécutant les instructions : `r1=coe($:-2:1)` et `r2=coe($-1:-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`.

Une autre façon, illustrant les opérations puissance et le produit de deux vecteurs, l'un constitué de polynômes, l'autre de scalaires, est de faire le programme suivant :

```
-->S=s^(10:-1:0)
 S =

!  10 9  8  7  6  5  4  3  2     !
! s  s  s  s  s  s  s  s  s  s  1 !

-->a=[0 ;2 ;5 ;8 ;9 ;6 ;7 ;8 ;9 ;1 ;2] ;
-->r=S*a
 r =
          2    3    4    5    6    7    8    9
2 + s + 9s + 8s + 7s + 6s + 9s + 8s + 5s + 2s
-->type(r)
ans =
   2.
```

2.3.6 Instructions relatives aux polynômes

Voici quelques instructions relatives à un polynôme.

```
-->num=6+11*s+6*s^2+s^3
 num =
            2   3
6 + 11s + 6s + s
-->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 : l'instruction `factors()`renvoie une liste.

```
-->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, mais ne retourne pas le coefficient de plus haut degré avec cette syntaxe. Nous verrons dans la section relative aux fractions rationnelles les autres syntaxes de `factors()`.

```
-->[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 (parce que ce polynôme a deux couples de racines complexes conjuguées). Cette instruction donne un vecteur colonne,

vecteur constitué de polynômes dont le produit des éléments est le polynôme de départ.[12]

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[13], 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.

```
-->num=6+11*s+6*s^2+s^3;
-->x=ldiv(num1,num,6)
 x =
!      9. !
!   - 51. !
!    214. !
!  - 773. !
!   2598. !
! - 8367. !
```

[12] Avec la dernière version (4.1) de Scilab l'argument de retour est un vecteur ligne et non plus un vecteur colonne :

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

[13] On verra par la suite une autre factorisation dite de Bode, factorisation utile pour l'étude des réponses fréquentielles des systèmes. Cette instruction `bodfact` fait partie de la bibliothèque `autoelem` que je propose avec ce document.

```
-->[r,q]=pdiv(num1,num)
 q =
                    2   3
- 773 + 214s - 51s +9s
 r =
                      2
4639 + 7221s + 2589s
```

L'instruction `ldiv()` donne ici les six premiers coefficients de la division du polynôme `num1` par le polynôme `num`. Attention on divise le polynôme $1+2s+8s^2+4s^3+7s^4+3s^5+9s^6$ par le polynôme $6+11s+6s^2+s^3$, division dans le sens des puissance décroissantes. Si ces deux polynômes sont respectivement le numérateur et dénominateur d'une transmittance d'un système linéaire continu, alors sous certaines conditions, cette division donne les paramètres de Markov du système (voir le programme `pmark()` correspondant dans la bibliothèque `autoelem`).

On ne conserve que six coefficients dans cette division. Le résultat est donc :

$$9s^3 - 51s^2 + 214s - 773s^0 + \frac{2598}{s^1} - \frac{8367}{s^2}$$

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

2.3.7 Quelques autres fonctions

Equation de Bezout : cette équation concerne deux polynômes `p1` et `p2` et retourne un polynôme `thegcd`, le plus grand commun diviseur de deux polynômes $p1(s)$ et $p2(s)$. De même cette fonction renvoie une matrice (2, 2) `U` unimodale. Dans ce programme on calcule par la même occasion le plus petit commun multiple `lelcm` de $p1(s)$ et $p2(s)$: la fonction `lcm(p1,p2)`.

```
-->s=%s ;p1=(s+1)*(s+2)^3*(s*s+s+1)
 p1 =
                2      3      4     5   6
  8 + 28s + 46s + 45s + 26s + 8s + s
-->p2=(s+2)^2*(s*s+s+1)
 p2 =
         2    3   4
  4 + 8s + 5s + s
-->[thegcd,U]=bezout(p1,p2)

 U =
!  5.536E-18           1           !
!                                  !
!                                 2!
!  1 - 2.149E-17s    - 2 - 3s - s  !
```

```
 thegcd =
             2    3   4
  4 + 8s + 9s + 5s + s

-->U=clean(U)
 U =
   0                1
                     2
   1     - 2 - 3s - s
-->deter=det(U)
 deter =
  - 1
-->Racgcd=roots(thegcd)
 Racgcd =
! - 0.5 + 0.8660254i !
! - 0.5 - 0.8660254i !
! - 2.                !
! - 2.                !
-->Racp1=roots(p1)
 Racp1 =
! - 0.5 + 0.8660254i !
! - 0.5 - 0.8660254i !
! - 1.                !
! - 2.                !
! - 2. + 1.686E-07i   !
! - 2. - 1.686E-07i   !
-->Racp2=roots(p2)
 Racp2 =
! - 0.5 + 0.8660254i !
! - 0.5 - 0.8660254i !
! - 2.                !
! - 2.                !
-->clean([p1,p2]*U)
 ans =
!             2    3   4      !
! 4 + 8s + 9s + 5s + s     0 !
-->thelcm=p1*U(1,2)
 thelcm =
                2     3     4    5   6
   8 + 28s + 46s + 45s + 26s + 8s + s
-->lelcm=lcm([p1,p2])
//ici on calcule le plus petit commun multiple de p1 et p2
 lelcm =
                2     3     4    5   6
   8 + 28s + 46s + 45s + 26s + 8s + s
```

Enfin une dernière instruction mettant en oeuvre l'algorithme de Leverrier.

```
-->H=[s,s*s+2;1-s,1+s]

 H =
!              2!
! s       2 + s !
!                !
! 1 - s   1 + s !
-->invr(H)

 ans =
!                           2  !
!    1 + s            -2 - s  !
!  -----------     -----------!
!             3              3!
! - 2 + 3s + s   - 2 + 3s + s !
!                             !
!   - 1 + s            s      !
!  -----------     -----------!
!             3              3!
! - 2 + 3s + s   - 2 + 3s + s !
-->[NU,DE]=coffg(H)
 DE =

            3
 - 2 + 3s + s
 NU =
!                     2!
!   1 + s     - 2 - s !
!                      !
! - 1 + s           s !
-->RES=NU/DE;
```

Si `H` est une matrice de polynômes ou de fractions rationnelles, la fonction `invr(H)` calcule la matrice inverse de `H`. Quant à l'instruction `coffg(H)` elle donne le dénominateur commun `DE` et la matrice `NU` (numérateur) de la matrice inverse de `H`.

```
-->detr(H)
 ans =
            3
 - 2 + 3s + s
```

De même l'instruction `detr(H)` donne le polynôme déterminant de la matrice de polynômes ou de rationnels `H`.

2.4 La programmation avec Scilab[14]

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 (celui de Scilab par exemple), des programmes.

2.4.1 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('chemindelafonction/nomdelafonction')`[15] [16]. 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 `nomdelafonction` et compile la ou les 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`. N'oublier pas de faire un retour chariot[17], à la fin de votre fichier, pour que la dernière ligne de programme soit prise en compte.

[14] Cette section est une traduction pratiquement mot à mot du document Scilab : Introduction to Scilab que vous trouverez sur le site de l'INRIA.

[15] On peut aussi avec l'instruction `getd('chemindurépertoire/répertoire')` dans une fenêtre Scilab, charger et compiler l'ensemble des macros, fichiers texte avec le suffixe `.sci`, contenues dans le répertoire.

[16] Appeler votre programme avec un suffixe `.sce` et une fonction avec le suffixe `.sci`.

[17] Je ne sais pas si cela reste valable avec les dernières versions de Scilab : n'oublier donc pas de mettre l'instruction `endfunction` à la fin de votre macro maintenant.

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.

2.4.2 La programmation

L'une des plus intéressantes fonctionnalités 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 2). 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.

Ces opérateurs de comparaison sont utilisés dans les instructions conditionnelles.

caractère	signification
$==$ *ou* $=$	égal à
$<=$	inférieur ou égal à
$>=$	supérieur ou égal à
$<$	inférieur à
$>$	supérieur à
$<>$ *ou* $\sim=$	pas égal à

Associé à ces opérateurs de comparaison on trouve trois opérations sur les booléens :

- Opération logique et : `&` et `and()` voir le manuel pour la différence de syntaxe.
- Opération logique ou : `|` et `or()` voir le manuel pour la différence de syntaxe.
- Opération logique non : `~`.

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.

Pour votre gouverne personnelle allez voir dans le répertoire `macros` de Scilab l'ensemble des fonctions qui vous intéressent vous en apprendrez plus que dans n'importe quel document sur Scilab.

2.5 Définir une fraction rationnelle : quelques propriétés

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

2.5.1 Les fractions rationnelles

Une fraction rationnelle, (chaque élément d'un vecteur de fractions, d'une matrice de fractions rationnelles), est le quotient de deux polynômes de la même variable symbolique, ici, s. C'est une liste[18] typée[19] (*type*16).

```
-->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 =
```

[18] Dans Scilab la notion de liste est très importante, avec ce type de structure, on peut mélanger des objets de type différent : scalaire, matrices, polynômes, chaîne de caractères ... On peut de même définir des opérations avec ces structures (addition, division, concaténation ...). Voir les instructions `list`, `tlist`, `overloading` dans l'aide.

[19] Dans Scilab les objets ont un type. C'est un entier prenant les valeurs suivantes : 1, 2, 4, 5, 8, 10, 11, 13, 15 (pour une liste), 16 (pour une liste typée), 128 . Voir l'aide avec les instructions `type(x)` et `typeof(objet)`.

```
              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.

On pouvait aussi extraire le numérateur et le dénominateur de la fraction rationnelle en faisant :

```
-->nu1=fr1('num')
 nu1 =
               2   3
  6 + 11s + 6s + s
-->de1=fr1('den')
 de1 =
               3
  - 6 - 7s + s
```

Dans ce cas on fait appel à l'extraction du deuxième et troisième élément de la liste typée `fr1` (explication au paragraphe ci-dessous).

2.5.2 Matrices, vecteurs de fractions rationnelles vus comme des listes

Nous allons voir comment Scilab traite les fractions rationnelles et comment sont stockées les données relatives à une fraction rationnelle.

Nous reprenons l'exemple de la fraction précédente qui vaut :

$$fr1 = \frac{6 + 11s + 6s^2 + s^3}{-6 - 7s + s^3}$$

Scilab stocke cette fraction sous forme d'une liste typée (*type*16) : voici un exemple de session permettant d'extraire les éléments de cette liste typée `tlist` .

```
-->typ=type(fr1)
 typ =
   16.
-->typeof(fr1)
 ans =
 rational
-->elem1=fr1(1)
 elem1 =
! r num den dt !
-->elem12=fr1(1)(2)
 elem12 =
 num
//j'ai extrait le deuxième élément du vecteur elem1
-->type(elem12)
 ans =
   10.
-->type(fr1(1))
 ans =
   10.
-->elem2=fr1(2)
 elem2 =
              2   3
  6 + 11s + 6s + s
-->elem3=fr1(3)
 elem3 =
              3
  - 6 - 7s + s
-->elem4=fr1(4)
 elem4 =
   []
```

Nous voyons que cette liste comprend quatre éléments : le premier est un vecteur chaîne de caractères de dimension quatre où est stockée la lettre `r` pour rationnel puis la chaîne `num` puis la chaîne `den` et enfin le caractère `[]` (pas de définition pour le temps). Ensuite dans le deuxième et troisième éléments on stocke respectivement les vrais polynômes puis rien. Une autre syntaxe de cette fraction pouvait être :
`fr1=tlist(['r','num','den','dt'],num,den,[])`

Il est évident qu'avec ce type de structure on peut maintenant définir des opérations : extraction, insertion, addition etc... Bien entendu les concepteurs du logiciel ont prévus ces opérations élémentaires pour des rationnels.

Dans l'aide à la rubrique `rational` (faire dans la fenêtre Scilab : `apropos rational`), vous trouverez un exemple où le numérateur et le dénominateur sont des matrices de polynômes.

2.5.3 Quelques fonctions utiles pour l'étude des fractions rationnelles

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

Comme je l'ai dit à la section 2.3.6 la fonction `factors()` sait traiter les fractions rationnelles (et les systèmes linéaires). Voici un exemple de programme :

[20] Effectivement quand la fraction a des pôles multiples avec un ordre de multiplicité important (>3) vous pouvez dans certains cas avoir du mal à trouver tous les éléments simples : problème sur la précision du calcul des racines du dénominateur de la fraction.

```
-->s=%s;n=poly([0.2,2,5],'s');
-->d=poly([0.1,0.3,7],'s');
-->R=n/d
 R =
                     2   3
 - 2 + 11.4s - 7.2s + s
 ------------------------
                       2   3
- 0.21 + 2.83s - 7.4s + s
-->[tn,td,g]=factors(R)
 g =
  1.
 td =
    td(1)
- 0.1 + s
    td(2)
- 0.3 + s
    td(3)
- 7 + s
 tn =
    tn(1)
- 0.2 + s
    tn(2)
- 2 + s
    tn(3)
- 5 + s
-->[tn1,td1,g1]=factors(R,'c')
 g1 =
   1.
 td1 =
     td1(1)
0.1 + s
     td1(2)
0.3 + s
     td1(3)
7 + s
 tn1 =
     tn1(1)
0.2 + s
     tn1(2)
2 + s
     tn1(3)
5 + s
```

On voit apparaître par cette commande que `factors()` sort une liste comprenant respectivement le nombre g rapport des coefficients de plus haut degré du numérateur et dénominateur, puis des termes du premier et/ou second degré (cas de racines complexes conguguées) du numérateur et enfin des termes analogues pour le dénominateur. Avec la présence du drapeau `'c'` dans l'instruction, `factors()` retourne des termes du premier et/ou second degré dont les racines sont dans le demi plan gauche du plan complexe si à l'origine elles ne l'étaient pas (voir le manuel en ligne). Cette fonctions est utile en automatique, quand on étudie le lieu d'Evans d'un système : on a affaire à la factorisation d'Evans.

Dans la boîte à outils `autoelem` que je donne avec ce document je propose une nouvelle fonction nommée `bodfact()` . Cette fonction est une fonction mathématique, qui factorise un polynôme, une fraction rationnelle, un système SISO (Single Input Single Output), 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, le système : le nombre de pôles ou zéros à l'origine). C'est le réel $s^{-L}sl(s)$ pour $s = 0$ où $sl(s)$ est la transmittance du système, ou la fraction, ou le polynôme.
- L : le nombre de dérivations (intégrations si L est négatif) de l'expression, voir remarque précédente.
- TN : un vecteur colonne comprenant des polynômes du premier et/ou second degré de la forme $1 + \tau_1 s$ et/ou $1 + \tau_2 s + \tau_3 s^2$ (s étant la variable symbolique) ; de plus on a pour ce polynôme de degré deux, la relation : $\tau_2^2 - 4\tau_3 < 0$ (racines complexes conjuguées). Ces polynômes caractérisent le numérateur.
- TD : un vecteur analogue à TN 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`[21] 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`. Ce chargement n'est utile que dans le cas particulier où vous n'avez pas installé définitivement cette bibliothèque[22].

```
-->;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')
```

[21] Cette fonction remplace les deux fonctions `phasemag` et `dbphi`, `phasemag` est boguée pour l'automaticien (décalage de phase de 360° dans certains cas). Même dans certains cas avec la version 4.1 de Scilab !

[22] Vous trouverez dans l'Annexe 2 une façon de mettre définitivement votre bibliothèque de fonctions et les pages de manuel en ligne dans Scilab.

```
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
```

Je vais profiter de cet exemple pour expliciter les instructions `prod()`, `cumprod()`, `sum()`, `cumsum()`. Nous voyons par l'exemple précédent que `prod(tn,1)` fait le produit des lignes du vecteur colonne `tn`. On pouvait écrire `prod(tn,'r')` : `'r'` pour row. Quant au produit des colonnes d'un vecteur ligne on écrirait : `prod(vecteur,2)` ou `prod(vecteur,'c')` ; il en est de même pour l'instruction `sum()` qui fait la somme au lieu du produit. Ces deux instructions s'appliquent aussi à des vecteurs qu'à des matrices.

Les instructions `cumprod()` et `cumsum()` réalisent en plus des opérations cumulatives : un exemple extrait de l'aide.

```
-->A=[1,2;3,4]
 A =
! 1.  2. !
! 3.  4. !
-->cumprod(A)
 ans =
! 1.   6. !
! 3.  24. !
-->cumprod(A,'r')
```

```
 ans =
! 1.  2. !
! 3.  8. !
-->cumprod(A,'c')
 ans =
! 1.   2. !
! 3.  12. !
```

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

3.1 Fonction de transfert, matrice de transfert

Dans ce petit rappel de cours nous expliciterons la notion de fonction ou matrice de transfert isomorphe et donnerons les principales caractéristiques et formes que peut prendre cette matrice (fonction) de transfert.

3.1.1 Rappel de cours, définition de la fonction de transfert

La modélisation de connaissance, souvent nécessaire afin de comprendre le comportement dynamique d'un procédé, conduit dans la majorité des cas à un modèle liant les grandeurs de sorties (conséquences) aux grandeurs d'entrées (causes), modèle souvent représenté par un système différentiel linéaire ou non (système à constantes localisées).

Pour trouver ce modèle, lors de la description du procédé, on fait intervenir les variables temporelles d'entrée, les variables de sortie et des variables internes au procédé. Cet ensemble de variables sont reliées par des éléments passifs dissipant ou stockant de l'énergie, des éléments fournissant de l'énergie comme les amplificateurs de puissance, des éléments assurant la mise en forme et le traitement des signaux support de ces variables.

La modélisation de connaissance utilise souvent comme méthodologie le principe d'analogie. En effet dans les systèmes, on peut classer les variables en deux catégories :

- Les variables d'effort : tension électrique, force ou couple en mécanique, pression en hydraulique, température dans des procédés thermiques ...
- Les variables de flux : courant électrique, vitesse linéaire ou angulaire en mécanique, débit en hydraulique, entropie en thermique ...

Pour caractériser le stockage et la dissipation d'énergie, les lois fondamentales de la physique introduisent des paramètres (des constantes localisées) caractérisant ces stockages ou transformations d'énergie. Un exemple en électricité est la transformation de l'énergie électrique en chaleur à travers une résistance R. De même pour caractériser le stockage d'énergie potentielle on introduit l'élément capacitif caractérisé par le paramètre C, quant au stockage d'énergie cinétique c'est le paramètre L (self inductance).

Si l'on revient à la modélisation on peut trouver deux lois fondamentales en physique :

- La loi des mailles reliant les variables d'effort aux variables de flux (on introduit là les paramètres cités au dessus).
- La loi des noeuds (conservation de la masse ou des débits en mécanique des fluide), qui énonce qu'en un noeud il n'y a pas génération spontanée de matière.

Comme ces lois, en particulier la loi des mailles, font intervenir des équations intégro-différentielles[1], l'ensemble des relations liant les variables d'entrées les variables internes et les variables de sorties seront aussi des équations intégro-différentielles. Au sens mathématique, si ces équations sont linéaires ou linéarisables autour d'un point de fonctionnement statique, on pourra par élimination des variables internes trouver, en prenant la transformée de Laplace de ces équations, une relation linéaire liant le vecteur transformée de Laplace des grandeurs d'entrée au vecteur transformée de Laplace des grandeurs de sortie : on a ainsi pour noyau de la relation linéaire une matrice de transfert de dimension (m, l) le nombre m est le nombre de variables de sortie et l le nombre de variables d'entrée.

$$Y(s) = G(s)U(s)$$

avec $Y(s) = \begin{bmatrix} y_1(s) & y_2(s) & ... & y_m(s) \end{bmatrix}^t$, $U(s) = \begin{bmatrix} u_1(s) & u_2(s) & ... & u_l(s) \end{bmatrix}^t$ et $G(s) = \{g_{ij}(s)\}$

Le terme $g_{ij}(s)$ représente la fonction de transfert élémentaire liant la sortie i à l'entrée j : comme on est en linéaire, le principe de superposition s'applique et donc pour avoir le terme $g_{ij}(s)$ on met les autres entrées à zéro et l'on considère que la sortie numéro i.

3.1.2 Diverses représentations

Si l'on a un système monovariable (une entrée, une sortie) ou si l'on considère la sortie i liée à l'entrée j d'un système multivariable, le terme $g_{ij}(s)$ que l'on notera maintenant $g(s)$ est la transmittance isomorphe du système considéré. Cette transmittance pour un système linéaire classique[2] sera un rapport de deux polynômes de la la variable complexe s, généralement le degré du numérateur de cette transmittance est inférieur ou égal au degré du dénominateur (système propre), dans ces conditions on peut écrire :

$$g(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + ... + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + ... + a_1 s + a_0} = \frac{N(s)}{D(s)}$$

La première idée qui vient à l'esprit pour étudier cette transmittance est de rechercher les zéros du polynôme numérateur $N(s)$ (zéros de la transmittance) et le zéros du polynôme dénominateur $D(s)$ (pôles de la transmittance). Bien entendu un autre paramètre, en plus des pôles et zéros, est nécessaire pour reconstruire cette transmittance : c'est soit $g(0)$ si le système ne possède pas de pôles (ou zéros) nuls ou $g(+\infty)$ (s'il existe) ou encore $s^r g(s)$ pour $s = 0$ (si le modèle possède r intégrations : r pôles à l'origine).

Quand on connait les pôles on peut facilement décomposer cette transmittance en éléments simples et $g(s)$ vaut donc :

$$g(s) = \frac{b_n}{a_n} + \sum_{i=1}^{r} \sum_{k=1}^{n_i} \frac{c_{ik}}{(s - p_i)^k}$$

[1] C'est ici que des équations aux dérivées non entières ou des équations aux dérivées partielles peuvent apparaître.

[2] Sauf pour les systèmes non entiers, pouvant décrire certains systèmes à modèles représentés par des équations aux dérivées partielles par exemple (systèmes à constantes réparties), ou des modèles où une dérivée non entière apparaît.

où $b_n = 0$ sauf si $m = n$, p_i ($i \in [1, n]$) sont les pôles de la transmittance, n_i est l'ordre de multiplicité du pôle p_i. Quant à c_{ik} résidu élémentaire relatif au pôle p_i il vaut :

$$c_{ik} = \left\{ \frac{1}{(n_i - k)!} \frac{d^{(n_i-k)}}{ds^{(n_i-k)}} [(s - p_i)^{n_i} g(s)] \right\}_{s=p_i}$$

Vous reverrez avec intérêt le cours de mathématique sur la mise sous forme éléments simples d'une fraction rationnelle. Scilab réalisant cette opération par l'instruction (voir section 2.5.3) : `elementsimp=pfss(g)`.

Bien entendu à partir de cette mise sous forme éléments simples, on peut trouver facilement l'original de la fraction rationnelle et obtenir diverses réponses temporelles suivant le signal (ou plutôt la transformée de Laplace du signal) d'entrée[3]. Un petit exemple simple pour illustrer ceci.

Soit un système de transmittance isomorphe

$$g(s) = \frac{s+3}{s(s^2+2s+2)(s+2)(s+1)^2}$$

Cette transmittance s'écrit, car les pôles valent $\left[\begin{array}{cccccc} 0 & -1+j & -1-j & -2 & -1 & -1 \end{array}\right]$: un pôle simple à l'origine et en -2, un pôle double en -1 et deux pôles complexes conjugués, que je regroupe, en $-1+j$ et $-1-j$.

$$g(s) = \frac{c_1}{s} + \frac{c_2 s + c_3}{s^2+2s+2} + \frac{c_4}{s+2} + \frac{c_5}{(s+1)} + \frac{c_6}{(s+1)^2}$$

Calculons les résidus :

$c_1 = sg(s)$ pour $s = 0$; $c_1 = 3/4$

$c_4 = (s+2)g(s)$ pour $s = -2$; $c_4 = -1/4$

$c_6 = (s+1)^2 g(s)$ pour $s = -1$; $c_6 = -2$

Il reste à trouver les trois derniers coefficients. On multiplie $g(s)$ par s et on fait tendre la variable $s \to +\infty$ soit :

$c_1 + c_2 + c_4 + c_5 = 0$ ou :

$c_2 + c_5 = -1/2$

On donne maintenant deux valeurs à s. Par exemple si $s = 1$ on a :

$2c_2 + 2c_3 + 5c_5 = -1/4$

et par exemple si $s = -3$ (qui est un zéro pour $g(s)$) on a :

$-24c_2 + 8c_3 - 20c_5 = 35$.

Par la résolution de ce système de trois équations à trois inconnues c_2, c_3, c_5 on a l'ensemble des résidus, soit :

$c_1 = 3/4,\ c_2 = 1/2,\ c_3 = 3/2,\ c_4 = -1/4,\ c_5 = -1,\ c_6 = -2$

On obtient ainsi pour l'original de $g(s)$ (la réponse impulsionnelle du système)

$$\begin{aligned} h(t) &= \frac{3}{4} + \frac{\sqrt{5}}{2}\exp(-t)\sin(t+\phi) - \frac{1}{4}\exp(-2t) - (1+2t)\exp(-t) \\ \phi &= \arctan(\frac{1}{2}) \end{aligned}$$

[3] On peut facilement avec Scilab programmer cette méthode des résidus et retrouver l'original d'une expression `g(s)`.

Vous vérifierez plus tard que par l'instruction `pfss(g,100)` on obtient la mise sous forme éléments simples. Vous devez donner comme deuxième argument à cette fonction un nombre assez grand sinon vous n'obtenez pas le résultat (à cause du pôle double).

On peut en programmant la méthode précédente retrouver l'original (réponse impulsionnelle) d'une fonction de transfert.

Représentations d'Evans

Quand l'analyse et la synthèse des systèmes asservis sont effectuées par la méthode du lieu des pôles ou lieu d'Evans[4], on écrit la transmittance isomorphe du système sous la forme :

$$g(s) = \frac{k_e}{s^r}\frac{s^m + (b_{m-1}/b_m)s^{m-1} + ... + (b_0/b_m)}{s^{n-r} + (a_{n-1}/a_n)s^{n-r-1} + ... + (a_r/a_n)} = \frac{k_e}{s^r}g_e(s)$$

A cause des intégrations (ce qui est souvent le cas dans certains systèmes) on met à part les pôles à l'origine dans la transmittance : ainsi $s^r g(s)$ aura une valeur finie pour le régime permanent ($s \to 0$, i.e $t \to +\infty$).

Forme polynomiale d'Evans C'est la forme précédente que l'on nomme forme polynômiale d'Evans, elle vaut :

$$g(s) = \frac{N(s)}{D(s)} = k_e\frac{g_e(s)}{s^r}$$

avec :

$$g_e(s) = \frac{s^m + b'_{m-1}s^{m-1} + ... + b'_1 s + b'_0}{s^{n-r} + a'_{n-1}s^{n-r-1} + ... + a'_{r+1}s + a'_r}$$

Par exemple prenons

$$g(s) = \frac{s+1}{4s^5 + 3s^4 + 2s^3 + s^2}$$

on aura pour la forme polynômiale d'Evans :

$$g(s) = \frac{1}{4}\frac{1}{s^2}\frac{s+1}{s^3 + (3/4)s^2 + (1/2)s + (1/4)}$$

Forme factorisée d'Evans Quant à la forme factorisée d'Evans elle consiste à rechercher les pôles non nuls (p_i) et zéros (z_j) de $g(s)$, à mettre de coté les pôles nuls si ils existent, on obtient ainsi :

$$g(s) = \frac{k_e}{s^r}\frac{\prod_{j=1}^{m}(s - z_j)}{\prod_{i=r+1}^{n}(s - p_i)}$$

Une remarque importante : le paramètre k_e n'a aucune signification physique. Cette forme peut être obtenue par Scilab avec l'instruction `factors()`, voici un exercice (cette instruction retourne une liste).

[4] Méthode permettant de faire la synthèse d'un système bouclé quand le gain de la chaîne d'action varie.

```
-->s=%s ;
-->sl=syslin('c',(1+3*s+s*s+4*s^3)/(s+6*s^2+7*s^3+8*s^4));
//j'anticipe sur la définition d'un système
-->[fnum,fden,gain]=factors(sl)
 gain =
     0.5
 fden =
   fden(1)
     s
   fden(2)
     0.2038539 + s
   fden(3)
                                  2
     0.6131843 + 0.6711461s + s
 fnum =
   fnum(1)
     0.3231486 + s
   fnum(2)
                                  2
     0.7736379 - 0.0731486s + s
```

De même cette dernière expression peut être représentée géométriquement dans le plan complexe : voir le cours sur le lieu d'Evans d'un système bouclé.

Forme factorisée de Bode

Quand on traite l'analyse et la synthèse des systèmes bouclés par une méthode fréquentielle on doit à tout prix factoriser la transmittance isomorphe sous forme de Bode. Avant la factorisation on met l'expression de la transmittance sous la forme :

$$g(s) = \frac{k_b}{s^r} \frac{1 + (b_1/b_0)s + (b_2/b_0)s^2 + ... + (b_m/b_0)s^m}{1 + (a_{r+1}/a_r)s + (a_{r+2}/a_r)s^2 + ... + (a_n/a_r)s^{n-r}} = \frac{k_b}{s^r} g_b(s)$$

Puis on factorise le rationnel $g_b(s)$ en le mettant sous la forme :

$$g_b(s) = \frac{(1+\tau_1 s)(1+\tau_2 s)...[1 + (2\xi_m/\omega_{n,m})s + (1/\omega_{n,m}^2)s^2]...}{(1+\lambda_1 s)(1+\lambda_2 s)...[1 + (2\xi_n/\omega_{n,n})s + (1/\omega_{n,n}^2)s^2]...}$$

C'est cette factorisation que je propose dans la boîte à outil `autoelem` sous forme de la fonction `bodfact()`. L'expression de $g(s)$ ainsi donnée met en évidence le nombre k_b gain statique (en position, en vitesse ...) du système, suivant que r prend les valeurs $0, 1...$ r est le nombre de pôles à l'origine de $g(s)$.

De même on voit apparaître au numérateur et dénominateur des constantes de temps τ_i ou λ_i quand les pôles (ou zéros) sont réels, ainsi que des facteurs du second degré pour chaque paire de pôles (ou zéros) complexes conjugués : ceci correspond à des systèmes élémentaires du premier et second ordre qui seront étudiés dans une section particulière (section 4.3). Voici la factorisation de Bode du système précédent.

```
-->[k,l,tn,td]=bodfact(sl)
 td =
! 1 + 4.905474s               !
!                              !
!                            2 !
! 1 + 1.094526s + 1.6308312s  !
 tn =
! 1 + 3.0945515s              !
!                              !
!                            2 !
! 1 - 0.094551s + 1.2925944s  !
 l =
  - 1.//c'est le nombre de 'derivateurs'
 k =
    1.
```

Pôles, zéros, gain statique, constantes de temps, amortissement, pulsation naturelle

Comme nous venons de le voir, si la transmittance isomorphe est donnée sous la forme :

$$g(s) = \frac{N(s)}{D(s)}$$

alors les pôles sont les racines de l'équation $D(s) = 0$ et les zéros les racines de $N(s) = 0$. Quant au gain statique (en position, en vitesse ...) c'est la valeur de $s^r g(s)$ pour $s = 0$. C'est le comportement permanent ($t \to +\infty$) du sous système de transmittance $s^r g(s)$: c'est la valeur de k_b dans la factorisation de Bode.

Si l'on introduit les constantes de temps que l'on voit apparaître dans la factorisation de Bode, pour des zéros et pôles réels on a : $\tau_j = -\frac{1}{z_j}$ et $\lambda_i = -\frac{1}{p_i}$. Si les pôles (zéros) sont complexes, comme ils sont conjugués deux par deux, on peut calculer le coefficient d'amortissement ξ et la pulsation naturelle ω_n à partir de la partie réelle et de la partie imaginaire des deux pôles par les relations :

$$\xi = -\frac{R_i}{\sqrt{R_i^2 + I_i^2}} \quad et \quad \omega_n = \sqrt{R_i^2 + I_i^2}$$

avec R_i la partie réelle du pôle et I_i la partie imaginaire de ce même pôle p_i. Ce calcul est aussi valable pour les zéros de la transmittance.

3.1.3 La définition avec Scilab d'une fonction de transfert

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 (matrices de polynômes) numérateur et dénominateur, par une fraction rationnelle, ou par un quadruplet $\left[\begin{array}{cccc} A & B & C & D \end{array}\right]$ donnant les équations d'état. Voici une session Scilab permettant d'illustrer cette instruction (c'est une liste typée : *type*16).

```
Startup execution :
loading initial environment
-->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. Après l'étude de la réponse fréquentielle des systèmes, je définirai un système par ses équations d'état[5].

Nous allons maintenant voir la structure qui caractérise un système linéaire. La syntaxe de cette instruction quand on introduit un système linéaire continu (qui peut être multivariable) est :
`sl=syslin('c',N,D)` ou `sl=syslin('c',G)`, où `G` est une fraction rationnelle ou une matrice de rationnels. En fait, comme pour les rationnels un système linéaire est une liste typée et l'on peut, comme pour toute liste, extraire, affecter, additionner, multiplier, diviser (attention au sens que l'on donne à la division) soit les éléments de cette liste, soit deux ou plusieurs listes entre elles.
`sl=tlist(['r','num','den','dt'],N,D,dom)` ou

[5] Contrairement à Matlab qui utilise trois instructions différentes (zpk(), ss(), tf()), Scilab n'utilise qu'une seule instruction pour définir un système linéaire. Attention syslin() n'accepte pas des retards purs en cascade contrairement à Matlab : on peut contourner ce manque en créant une liste, voir la section 10.1.3.

`sl=tlist(['r','num','den','dt'],G(2),G(3),dom)`.
On retrouve la même représentation que dans la définition des fractions rationnelles, ce qui change par rapport à celles-ci, c'est la présence de la chaîne de caractères `'dt'` et la présence de la variable `dom` (valeur du temps).

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 en particulier si l'on veut récupérer la variable à partir de `num` (ou `den`)[6].

En résumé (de la part de l'auteur de ce rapport) : utilisez l'instruction `sl=syslin('c',num/den)` ou si l'un des termes, numérateur et/ou dénominateur est constant, faites d'abord par exemple, `num=0.5*poly(1,'s','c')` puis `sl=syslin('c',num,den)`. Un autre exemple est la définition de la transmittance `1(s)`, par l'instruction :
`sys=syslin('c',poly(1,'s','c'),poly(1,'s','c'))`. Si on ne souhaite pas simplifier une fraction rationnelle, quand elle possède des pôles et zéros identiques, on fera : `sys=syslin('c',num,den)`. On utilisera la même méthode pour définir un rationnel.

3.2 Les graphiques dans Scilab, réponses temporelles d'un système

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).

3.2.1 Les graphiques en deux dimensions avec Scilab

La suite de la session va nous permettre de voir les graphiques utilisés par Scilab, en particulier d'étudier les instruction `plot()`, `plot2d()`, `xsetech()`, `subplot()`[7]...

Exemple issu de la Demo de Scilab, plot, plot2d[8]

Voici se que vous allez obtenir en frappant l'instruction `xtitle()` dans une fenêtre Scilab (FIG. 3.1, FIG. 3.2) .

```
Startup execution : loading initial environment
```

[6] Une correction possible consiste à retoucher l'instruction `syslin` (voir section sur les bogues et imperfections) ou à remplacer la `constante` par `constante=poly(constante,'s','c')` si `s` est la variable de l'autre polynôme : cette correction sera faite, sans doute, dans la prochaine version de Scilab, la version 2.7 . C'est fait !

[7] Avec des titres, des légendes, en couleur et tout et tout.

[8] Depuis la version 2.6 de Scilab l'instruction `plot2d()` a un peu évoluée (voir l'exemple qui suit). De même l'instruction `plot2d1()` est obsolète : attention au diagramme de Bode.

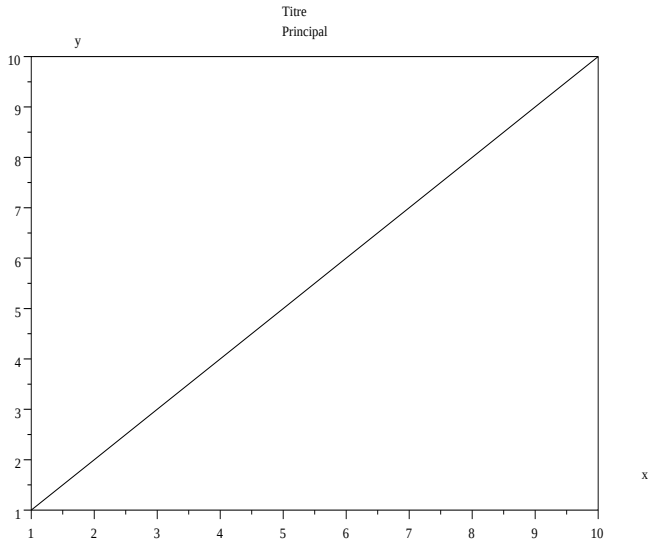


FIG. 3.1: Démo xtitle()

```
-->xtitle()
Demo of xtitle
x=(1:10)';
plot2d(x,x);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 contenant la première bissectrice dans un cadre ayant les valeurs maximales de l'abcisse et ordonnée, avec des graduations entières et des sous graduations. 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 : dans cette instruction on a défini un vecteur colonne de deux chaînes de caractères.

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)]'`

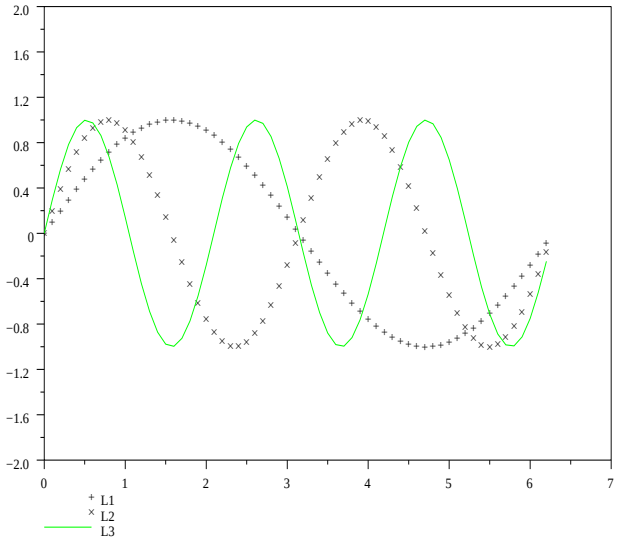


FIG. 3.2: Démo plot2d()

`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)[9] : 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`.

[9] Il semble que dans la version 4.1 de Scilab on ne puisse plus définir la position.

`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.

Depuis la version 2.6 de Scilab une nouvelle syntaxe pour l'instruction `plot2d()` est apparue :

```
-->plot2d(x,[sin(x) sin(2*x) sin(3*x)],..
-->[1,2,3],leg="L1@L2@L3",nax=[2,10,2,10],rect=[0,-2,2*%pi,2])
```

Par cette syntaxe on définit des mots clés qui permettent une plus grande lisibilité de l'instruction.

xsetech() et subplot()

La première instruction `xsetech()` apparaît dans ce document lors de l'exécution de notre premier programme à la section 2.2. Elle a pour but de créer des sous graphiques à l'intérieur d'une fenêtre. Un bel exemple d'un graphique spécialisé en automatique est le lieu de Bode constitué de deux sous graphiques : le lieu de Bode de gain et le lieu de Bode de phase. C'est avec cette instruction `xsetech()`, que les concepteurs de Scilab ont réalisé le programme `bode.sci` situé dans le répertoire `SCI/macros/xdess`.

```
[ffr,bds]=xgetech();
//magnitude
xsetech([0,0,1.0,hx*0.95]);
rect=[mini(frq),mini(d),maxi(frq),maxi(d)]
// just to fix the scales for xgrid
plot2d1("oln",mini(frq),mini(d),0,"051"," ",rect);
//avec la nouveauté de plot2d cette instruction est à revoir
// xgrid first
xgrid(4);
// now the curves
plot2d1("oln",frq',d',[1:mn],"000"," ",rect);
//avec la nouveauté de plot2d cette instruction est à revoir
if type(dom)==1 then [xx1,xx2]=xgetech();
val= xx2([2;4])';
```

```
plot2d1("oln",max(frq)*[1;1],val,5,"000"," ",rect);
//idem
end
xtitle('Magnitude ',' Hz','db');
```

La première instruction `xgetech()` donne l'échelle de la fenêtre graphique courante (ouvre la fenêtre 0 si aucun graphique n'a été tracé), avec pour `ffr` un vecteur ligne donnant les coordonnées du point haut gauche (abcisse, ordonnée), puis pour les deux nombres suivants de ce vecteur ligne, les proportions (largeur, hauteur) dans lesquelles le graphique est tracé. Quant au vecteur `bds`, il donne les dimensions d'un rectangle sous forme : $x_{min}, y_{min}, x_{max}, y_{max}$, rectangle inclus dans `bds`.

La seconde instruction `xsetech()` va permettre ici de découper la zone de dessin, peut être toute la fenêtre, en deux zones horizontales dans lesquelles le lieu de Bode gain sera tracé, puis en dessous le lieu représentant la phase. De même on tracera en couleur une grille en bleu cyan, nombre `4` de l'instruction `xgrid(4)`. Enfin un titre au premier sous graphique (courbe de gain) sera rajouté par l'instruction `xtitle()`.

L'instruction `subplot()` est une version édulcorée de `xsetech()` : elle découpe la fenêtre graphique courante en une matrice n, m de lignes et colonnes et dans chaque case correspondante on tracera une courbe ou un grahique 3d. Faites `help subplot` dans Scilab et copier l'exemple.

Les couleurs

Nous avons vu dans la syntaxe de l'instruction `plot2d()` que l'on pouvait gérer, par des paramètres optionnels, le style du tracé, les légendes, la taille du cadre, les graduations, etc. Cette syntaxe devient plus explicite en passant dans la liste d'appel, un mot clé, sous la forme : `plot2d(x,y,mot_clé=...)`. Ainsi, pour spécifier le style du tracé on peut utiliser la syntaxe : `plot2d(x,y,style=3)` et alors le tracé de la courbe se fera avec la couleur verte claire. Pour voir le numéro des couleurs par défaut dans Scilab, exécutez dans Scilab `getcolor()`. Si le mot clé `style` est négatif ou nul, alors le tracé est en noir sur fond blanc (normalement) avec une marque particulière, pour voir les marques de Scilab faites : `getlinestyle()`, par exemple `plot2d(x,y,style=-3)` donnera un tracé avec une étoile noire.

On faut si l'on veut définir un tracé de couleur (ou blanc) avec un fond de couleur (ou noir) définir le contexte graphique. Ce contexte est donné par l'instruction `xset()` dans le cas général. Appliqué au problème posé cela donnera les deux instructions :

```
-->xset('background',numéro_de_couleur);
-->xset('foreground',autre_numéro);
```

Voici un exemple.

```
-->xset("background",22)
-->xset("foreground",10)
-->x=linspace(0,8,101);
-->plot2d(x,tan(x),style=34)
```

Faites cet exercice pour visualiser le résultat.

Autres paramètres optionnels de plot2d()[10]

Pour règler le cadre du tracé, les axes et les graduations on utilise les mots-clés : `frameflag` avec comme option de `frameflag=rec`. Si `frameflag=0` alors on ne trace pas de cadre, la taille du cadre est donnée par le paramètre `rec`. Si l'on veut des graduations entières ou isométriques, le cadre peut être agrandi en donnant à `frameflag` la valeur 4. Quant aux axes et aux graduations elles peuvent être modifiées par le mot-clé `axesflag` associé à `nax` par exemple si `axesflag=3` on met l'axe vertical à droite. Voici un exemple que vous réaliserez :

```
-->xset("background",30)
-->xset("foreground",10)
-->plot2d([x;x;x]',[sin(x);sin(2*x);sin(3*x)]',style=[34,3,20],..
axesflag=3)
-->legends(['sin(x)';'sin(2x)';'sin(3x)'],[34,3,20])
```

Si vous changer la valeur de `axesflag` vous pouvez avoir une graduation à gauche, au milieu, à droite, amusez vous ! Enfin une dernière remarque : tant que vous ne fermez pas la fenêtre graphique (`bouton file, close` de `ScilabGraphic0`) par exemple, le contexte graphique ne change pas : pour ne pas le changer utiliser le `bouton file, clear`.

Conclusion, applications à l'automatique

Scilab a prévu de nombreux graphiques spécialisés dont voici la liste.

- `plzr(système)`. Position dans le plan complexe des pôles et zéros. Le système peut être multivariable, sous forme matrice de transfert, ou sous forme état.
- `evans(boucle_ouverte,gain)`. Lieu des pôles ou d'Evans d'un système bouclé quand le gain de la chaîne d'action varie.
- `sgrid(zéta,omegan[,couleur])`. Exemple : `sgrid(0.6,2,7)`, ou aussi `sgrid('new')`. Ce graphique peut être avantageusement associé avec la fonction `evans()`. Il donne dans le plan complexe, les courbes d'iso-amortissement et les courbes d'iso-pulsation naturelle.
- `zgrid()`. Pour les systèmes échantillonnés : lignes à amortissement constant et lignes à pulsation naturelle constante à l'intérieur du cercle de rayon unité du plan des `z`.
- `chart(...)`. L'abaque de Black.
- `black(...)`. La courbe de Black.
- `bode(...)`. Les courbes de Bode de gain et phase.
- `gainplot(...)`. La courbe de Bode de gain seule.
- `nyquist(...)`. La courbe de Nyquist.
- `m_circle()` ou `m_circle(gain)`. Isogains dans le plan de Nyquist (abaque de Hall).

Dans ce document nous verrons ces principaux graphiques, de même je propose des lieux fréquentiels adaptés dans la bibliothèque `autoelem` contenant des fonctions graphiques

[10] Lisez le manuel de `plot2d`, instruction : `apropos plot2d`.

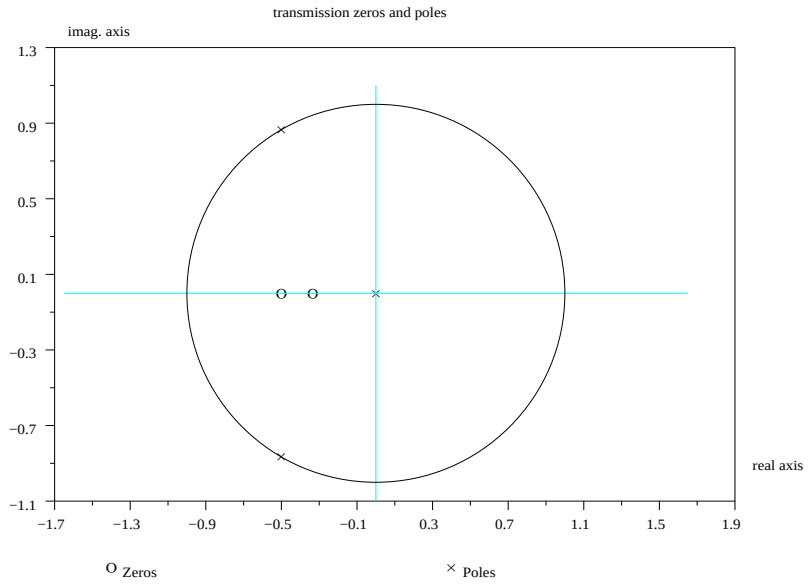


FIG. 3.3: Pôles et zéros

pouvant tracer les lieux fréquentiels des systèmes à retard pur entre autre (voir les explications et exemples dans les pages de manuel).

Voici ces lieux :

- `cchart(...)`. L'abaque de black modifiée.
- `bblack(...)`. La courbe de Black (avec retard ou autre).
- `bbode(...)`. Les courbes de Bode (avec retard ou autre).
- `ggainplot(...)`. La courbe de Bode de gain seule (avec retard ou autre).
- `nnyquist(...)`. La courbe de Nyquist (avec retard ou autre).

3.2.2 Visualisation 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 (FIG. 3.3).

```
-->s=%s ;
-->sys=syslin('c',((1+2*s)*(1+3*s))/(s*(s*s+s+1))) ;
-->plzr(sys)
```

Vous remarquerez que cette instruction trace en plus des pôles et zéros un cercle de rayon unité, cercle utile pour l'étude de la stabilité des systèmes échantillonnés.

3.2.3 Simulation temporelle : réponses impulsionnelle, indicielle, à tout type de signal

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 linéaires ou non, 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 ou instants`. 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.

La simulation d'un système : réponses impulsionnelle, indicielle

```
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 ;//très important le ;
-->h=csim('imp',t,sl) ;//très important le ;
-->xbasc()//pas forcement utile
-->plot2d(t',h')
```

Par ces commandes on définit un système linéaire continu de transmittance `sl`, 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`, vous remarquerez la syntaxe : `début:pas:fin`). Enfin on simule la réponse impulsionnelle (FIG. 3.4), présence du drapeau `'imp'` dans l'instruction `csim`. Les deux dernières instructions permettent d'une part d'effacer la fenêtre graphique, (si la fenêtre par défaut avait déjà été utilisée), 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 à l'aide de la fonction `plot()` et `xtitle("")`(FIG. 3.5).

```
-->y1=csim('step',t,sl) ;
-->xbasc()//inutile dans certains cas
-->plot(t,y1)
-->xtitle('reponse indicielle')
```

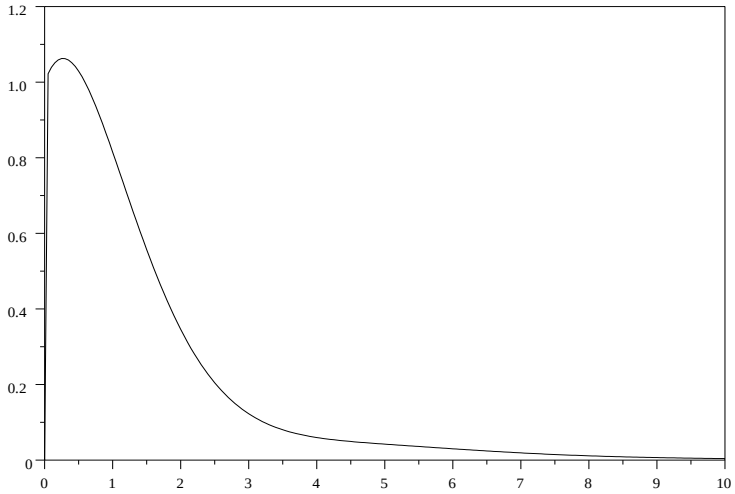


FIG. 3.4: Réponse impulsionnelle

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(début,fin,nombre de graduations)` 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. La troisième façon de procéder, consiste à créer un vecteur d'entiers et à diviser chaque élément de ce vecteur par l'inverse du 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.

Création par Scilab d'un signal d'entrée : simulation d'un système soumis à un signal quelconque

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-

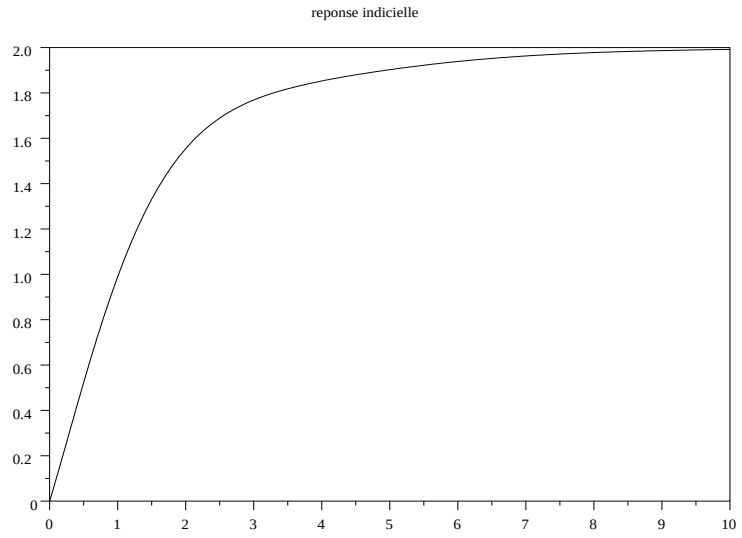


FIG. 3.5: Réponse indicielle

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) pour réaliser le signal d'entrée.

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 (FIG. 3.6).

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

```
function [u]=signentre(t,t1,t2,t3)
tinft1=(t<=t1)
//c'est un booléen donnant %t quand t<=t1, %f autrement.
tinft2=(t<=t2)
//idem
inter_0_t1=bool2s(tinft1)
//remplacement d'un booléen par les scalaires 0 ou 1
inter_t1_t2=bool2s(~tinft1&tinft2)
//idem, voir le signe ~ (pas égal à)
inter_t2_t3=bool2s(~tinft2&(t<=t3))
//idem
u=inter_0_t1.*(t/t1)+inter_t1_t2.*(1.)+inter_t2_t3.*(t3-t)/(t3-t2)
endfunction
//création du vecteur sortie
```

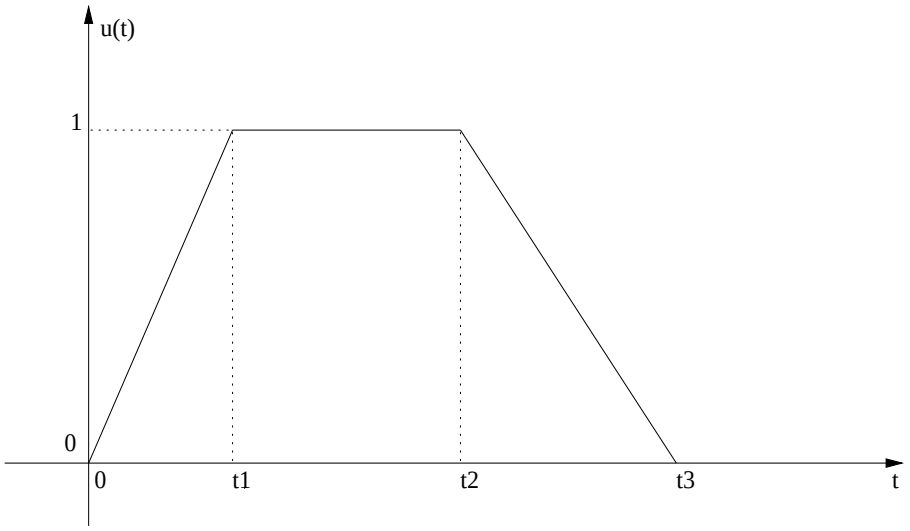


FIG. 3.6: Signal d'entrée

L'utilisation de l'instruction `bool2s()`[11] permet de convertir une matrice de booléens en matrices de réels. 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()`. Cet exemple de programme peut facilement être adapté à tout type de signal d'entrée défini par morceaux.

Un exemple plus simple 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 que je range dans le fichier texte `creneau.sci` et que je mets dans mon répertoire.

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

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`

[11] La fonction `bool2s` remplace le booléen `%f` par le scalaire `0` et remplace le booléen `%t` par le scalaire `1`. Ce programme est un bel exemple de vectorisation des calculs.

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`.

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 $u(t)$. Un conseil : lisez attentivement le manuel sur la fonction `csim`. Mon collègue déjà cité, (Patrick Sarri) propose une extension de cette 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)`[12].

3.2.4 Introduction des conditions initiales, utilisation de la bibliothèque ode

Les condition initiales

On peut aussi introduire facilement des conditions initiales avec l'instruction `csim()`, remplacez `y=csim()` par les instructions suivantes (suite du même exercice) en encapsulant le créneau et sa durée dans une liste `ul`.

[12]Cette nouvelle possibilité a été mise dans la version 2.6 de Scilab.

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

Je ne reproduis pas ici la réponse : vous réaliserez l'exercice avec Scilab.

La bibliothèque ode : résolution de systèmes différentiels

Avant de décrire les outils utiles à la résolution temporelle de systèmes dynamiques je voudrais signaler une série d'articles parus dans une revue grand public Linux Magazine, articles écrits par les concepteurs de Scilab, traitant entre autre de la résolution de systèmes différentiels linéaires et non linéaires[13]. Comme je l'ai signalé au début de cette section, l'instruction `csim()` est capable de simuler la réponse d'un système à tout type de signal, mais peut en plus donner l'évolution temporelle des diverses composantes d'état d'un système linéaire (modèle décrit par un système d'équations différentielles linéaires du premier ordre à coefficients constants) : cette instruction utilise un des programmes fortran contenus dans le package ODEPACK : `csim()` fait appel à la fonction `ode()`, méthode de résolution nommée méthode d'Adams. En explorant le programme `csim.sci` situé dans le répertoire `SCI/macros/auto` vous verrez la structure du programme.

Voici un exemple de programme Scilab permettant de simuler et de sortir dans l'espace d'état (plan de phase) la trajectoire d'un système du second ordre à commande nulle ($u(t) = 0, \forall t \geq t_0$), partant d'un vecteur d'état initial de valeur $x_0 = \begin{bmatrix} 1 & 1 \end{bmatrix}^t$ (FIG. 3.7). Attention le vecteur d'état n'est pas constitué dans le cas général, de $y(t)$ et de $\frac{dy(t)}{dt}$ mais d'une combinaison linéaire de ces deux variables et donc le vecteur d'état initial est une combinaison linéaire des composantes du vecteur conditions initiales.

```
-->s=%s ;
-->g=syslin('c',(1+s)/(1+s+s*s)) ;
-->m=500 ;T=30 ;
-->t=linspace(0,T,m) ;
-->deff('u=input(t)','u=0')
-->x0=[1 ;1] ;
-->[rep,x]=csim(input,t,g,x0) ;
//on sort la réponse et l'évolution du vecteur d'état.
-->plot2d(x(1,:)',x(2,:)',style=6,axesflag=3)
-->xgrid(4)
```

Dans le chapitre relatif aux variables d'état je reviendrais sur le plan de phase pour les systèmes linéaires d'ordre deux.

Nous allons afin de s'initier au programme `ode()`, refaire l'exercice sur le plan de phase, mais avant je vais faire un petit rappel sur les systèmes d'équations différentielles.

[13] Vous trouverez ces articles sur le site de Scilab, ou dans la revue Linux Magazine de mai 2000.

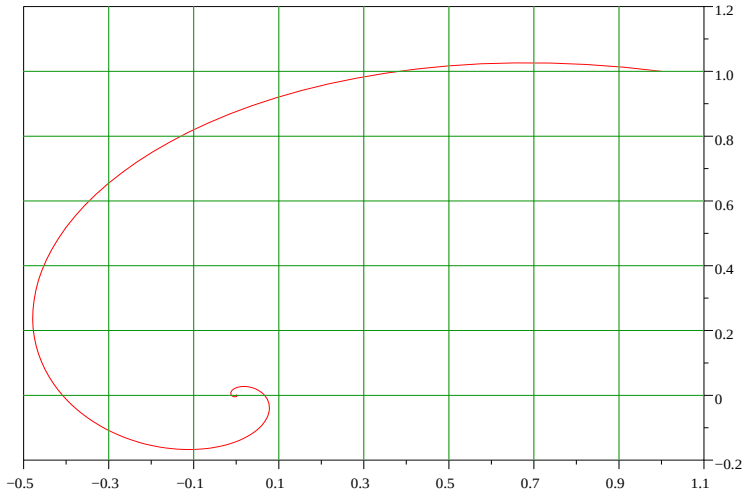


FIG. 3.7: Plan de phase

Système d'équations différentielles du premier ordre Soit un système d'équations différentielles du premier ordre (ou une équation différentielle d'ordre n mise sous forme d'un système d'équations différentielles d'ordre 1), avec un vecteur conditions initiales x_0.

$$\frac{dx}{dt} = f(t, x(t), u(t)) \; avec \; x(t_0) = x_0 \; et \; u(t) \; la \; commande$$

Ici $x(t)$ est un vecteur de R^n, f une fonction de $R \times R^n \rightarrow R^n$, et $x_0 \in R^n$. On supposera l'existence et l'unicité de la solution sur un horizon d'observation $\left[\begin{array}{cc} t_0 & t_1 \end{array} \right] = T$. Dans le cas de l'étude du système autonome ($u(t) = 0, \forall t \geq t_0$), on écrira le second membre du système différentiel f comme une fonction Scilab dans un fichier ou à la ligne de commande dans la fenêtre Scilab, avec la syntaxe suivante :

```
function f=mafonction(t,x)
//ici on code la fonction
endfunction
```

Reprenons la transmittance précédente qui vaut :

$$g(s) = \frac{1+s}{1+s+s^2}$$

qui correspond à l'équation différentielle suivante :

$$\frac{d^2y(t)}{dt^2} + \frac{dy(t)}{dt} + y(t) = u(t) + \frac{du(t)}{dt}$$

Le solveur `ode()` traitant les systèmes différentiels d'ordre 1, on met cette équation sous la forme de deux équations différentielles d'ordre 1. Ceci peut être fait en utilisant l'instruction Scilab : `ssg=tf2ss(g)` et on obtient ainsi le système différentiel (solution non unique, voir cours sur les équations d'état d'un système) :

$$\begin{aligned} \frac{dx_1(t)}{dt} &= -0,5x_1(t) - x_2(t) - 2u(t) \\ \frac{dx_2(t)}{dt} &= 0,75x_1(t) - 0,5x_2(t) + u(t) \end{aligned}$$

avec :

$$y(t) = -0,5x_1(t)$$

Une autre mise en équation, donnant pour la première composante du vecteur d'état la valeur de la sortie $y(t)$ peut être :

$$\begin{aligned} \frac{dx_1^*(t)}{dt} &= x_2^*(t) + u(t) \\ \frac{dx_2^*(t)}{dt} &= -x_1^*(t) - x_2^*(t) \end{aligned}$$

avec :

$$y(t) = x_1^*(t)$$

équations plus simples à simuler et plus lisibles (forme compagnon observable). C'est cette forme que l'on simulera avec le solveur `ode()`. Afin de comparer avec l'exercice précédent, seul le régime autonome ($u(t) = 0, \forall t \geq 0$) sera envisagé : voici la programmation Scilab (FIG. 3.8).

```
-->function [f]=mafonct(t,x)
-->f(1)=x(2)
-->f(2)=-x(1)-x(2)
-->endfunction
-->//on trace le champ de vecteurs
-->z=linspace(-1.5,1.2,10);
-->fchamp(mafonct,0,z,z)
-->//on résoud le système
-->x0=[1;1];//condition initiale
-->t=linspace(0,50,501);
-->[x]=ode(x0,0,t,mafonct);
-->plot2d(x(1,:)',x(2,:)',9,'000')
```

Nous allons, en nous inspirant (c'est plus que de l'inspiration !)[14] du document du Professeur Bruno Pinçon cité à la section 3.2.3.2, faire une petite animation permettant de visualiser les trajectoires de phase à partir de diverses conditions initiales. Voici un programme que je nomme `odeanim.sce` que je stocke dans mon répertoire :

[14] Voir aussi la revue Linux Magazine de mai 2000, que j'ai déjà cité.

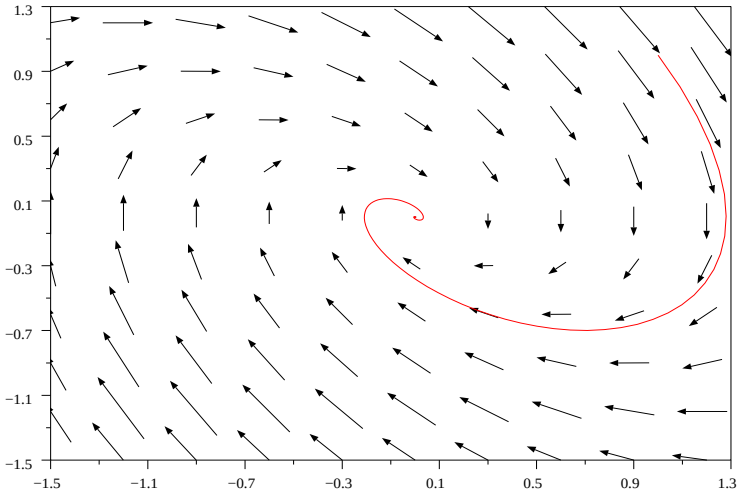


FIG. 3.8: Plan de phase avec ode

```
function [f]=mafonct(t,x);
f(1)=x(2);
f(2)=-x(1)-x(2);
endfunction
//c'était la programmation des équations
xxmin=-2;xxmax=2;yymin=-3;yymax=3;n=10;
xx=linspace(xxmin,xxmax,n);
yy=linspace(yymin,yymax,n);
fchamp(mafonct,0,xx,yy)
//le cadre et les lignes de champ
t=linspace(0,10,101);
coul=[2,3,4,5,6,7,15,16,18,22,21,27]; num=-1;
while %t
ret=xclick();
if ret(1)==0 then//bouton gauche de la souris
plot2d(ret(2),ret(3),-9,'000')
//abscisse et ordonnée du point choisi
x0=[ret(2);ret(3)];
x=ode(x0,0,t,mafonct);
num=modulo(num+1,length(coul));
plot2d(x(1,:)',x(2,:)',coul(num+1),'000')
elseif ret(1)==2 then
```

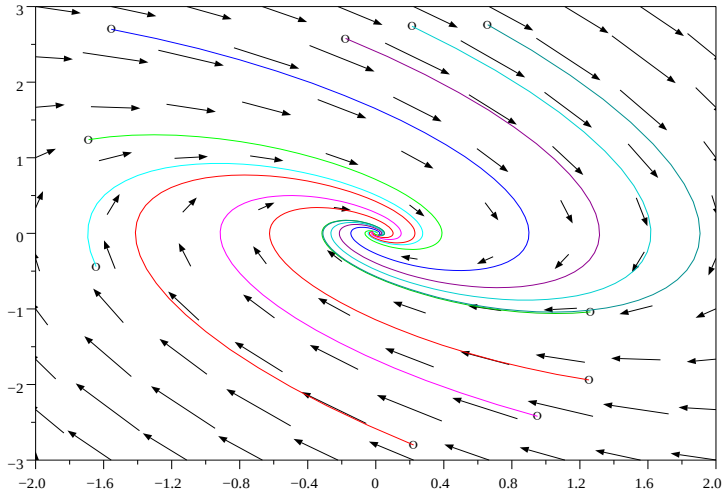


FIG. 3.9: Plan de phase animé

```
//si bouton droit alors on arrête l'animation.
break
end
end
```

`xclick()` renvoie un vecteur ligne de trois informations : le numéro de bouton de souris choisi, abscisse et ordonnée du point du graphique courant choisi. Pour exécuter ce programme, faire dans une fenêtre Scilab (FIG. 3.9) :

```
--> ;exec("/home/lpovy/odeanim.sce") ;
```

Si maintenant vous souhaitez simuler des équations non linéaires avec le programme `ode()`, vous n'aurez qu'à changer dans le programme précédent `mafonct` par `tafonct`, le cadre pour le tracé des lignes de champ, les couleurs et éventuellement le choix des boutons de la souris : exemple l'équation de Van der Pol :

$$\frac{d^2y(t)}{dt^2} - a(1 - y^2(t))\frac{dy(t)}{dt} + y(t) = 0$$

Vous reformulerez cette équation en un système différentiel et vous obtiendrez la fonction :

```
function [f]=vander(t,x,a) ;
f(1)=x(2) ;
f(2)=-x(1)+a*(1-x(1)*x(1))*x(2) ;
```

```
//ici on met a en paramètre
endfunction
```

Comme on a mis un paramètre `a` dans la fonction, on va créer une liste et appeler `ode()` de la manière suivante :

```
x=ode(x0,t0,t,list(vander,a))
//dans le programme principal a aura une valeur
```

Dernière remarque : les trajectoires de phase dans les deux programmes ne se ressemblent pas, la mise en équations n'est pas la même car les variables d'état sont différentes. Si on voulait comparer les deux méthodes (avec `csim()` et `ode()`), il faut définir le système directement dans le premier programme par ces équations d'état.

De même on peut avec l'instruction `param3d()` donner, dans un système d'axes 3d, la trajectoire.

```
xet('window',1)
x=ode([1 ;2],0,t,mafonct) ;
param3d(x(1,:),x(2,:),t,leg="x1@x2@t",flag=[1,4],ebox=[0,3,0,3,0,10])
```

Système d'équations différentielles d'ordre supérieur à un On considère le système mécanique décrit par la figure (FIG. 3.10) :

Soit une poulie de rayon r et de moment d'inertie $I = \frac{Mr}{2}^2$ par rapport à l'axe de rotation $0z$ (perpendiculaire à la figure), sur laquelle s'enroule un cable inextensible dont une des extrémités est liée au sol par l'intermédiaire d'un ressort de raideur k et dont l'autre extrémité est reliée à une masse m par l'intermédiaire d'un ressort identique au premier. On admettra que les frottements sont négligeables. Si θ et x représentent respectivement la rotation de la poulie et le déplacement de la masse m par rapport à la position d'équilibre du à la pesanteur, on a les deux équations différentielles suivantes, que l'on obtient par exemple à partir des équations de Lagrange : où $F(t)$ représente la force instantanée que l'on applique à la masse en mouvement m.

Afin d'étudier ces deux équations différentielles du second ordre couplées on peut poser :

$x_1 = x\,, x_2 = \frac{dx}{dt}\,, x_3 = \theta\,, x_4 = \frac{d\theta}{dt}$ et l'on obtient le système différentiel :

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{k}{m} & 0 & -\frac{kr}{m} & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{kr}{I} & 0 & -\frac{2kr^2}{I} & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{F(t)}{m} \\ 0 \\ 0 \end{bmatrix}$$

système que l'on pourra facilement intégrer à l'aide de de l'outil `ode()`.

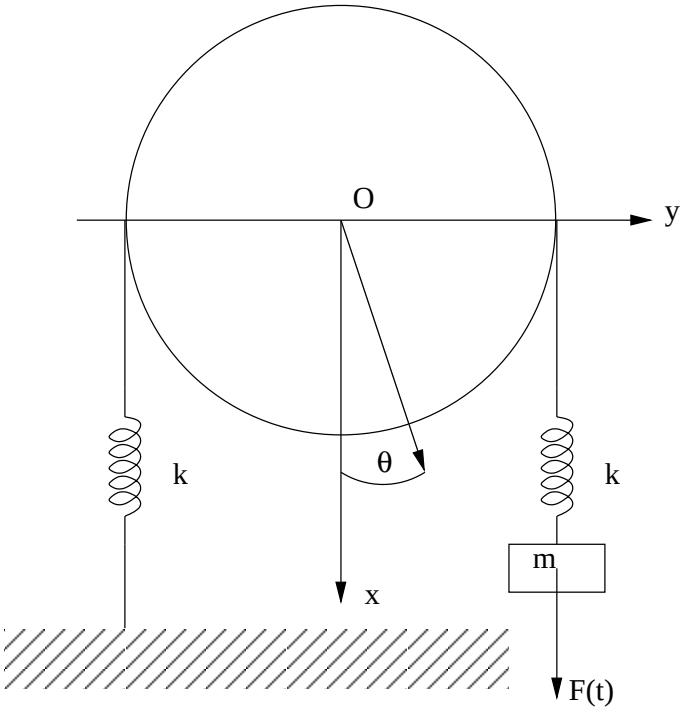


FIG. 3.10: Système différentiel

4 Représentation fréquentielle

4.1 Rappels sur la réponse fréquentielle : calcul de cette réponse

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.

Je rappelle que la réponse fréquentielle d'un système (vecteur de systèmes, car Scilab sait traiter les réponses fréquentielles d'un vecteur système), revient à calculer le nombre complexe $g(j\omega)$ transmittance isochrone, si $g(s)$ est le modèle transmittance isomorphe du sytème. Dans Scilab comme l'argument n'est pas la pulsation mais la fréquence, le nombre qui sera calculé est : $g(j2\pi f)$; f représente le vecteur (ou la matrice) fréquence. Ce nombre peut être représenté par la fraction :

$$g(j2\pi f) = \frac{N(j2\pi f)}{D(j2\pi f)}$$

où $N(s)$ et $D(s)$ sont respectivement le polynôme numérateur (degré m) et le polynôme dénominateur (degré n) de la transmittance isomorphe. Cette relation peut aussi s'écrire

$$g(j2\pi f) = \frac{N(j2\pi f)}{D(j2\pi f)} \frac{D(-j2\pi f)}{D(-j2\pi f)}$$

dans ces conditions on peut voir que cette quantité se transforme en sa conjuguée quand on change f en $-f$. Plus généralement comme $g(s)$ est un rapport de deux polynômes de la variable s on a : $g(s^\star) = g^\star(s)$. Ceci interviendra dans l'étude du lieu de Nyquist complet.

Nous allons dans les exercices qui vont suivre 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. La première instruction intéressante est l'instruction `freq()`[1].

```
Startup execution : loading initial environment
-->x=poly(0,'x')
```

[1] Cette instruction est une instruction mathématique et a déjà été vue dans un paragraphe précédent (section 2.3.3).

```
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 et 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 un vecteur ligne 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 et est semble t'il plus performante que l'instruction `horner()`.

Passons maintenant aux instructions spécifiques à l'étude des systèmes. Une instruction très importante pour l'étude de la réponse fréquentielle d'un système est l'instruction `repfreq()`[2], elle s'applique aux systèmes linéaires continus ou échantillonnés, avec modèle vecteur de transfert (système SIMO : Single Input, Multiple Output), ou un modèle d'état : quadruplet $\left[\begin{array}{cccc} A & B & C & D \end{array}\right]$.

```
-->A=diag([-1,-2]);B=[1;1];C=[1,1];
//On définit un modèle d'état pour le système.
-->Sys=syslin('c',A,B,C)
```

[2] Vous trouverez dans ma bibliothèque `autoelem` une instruction `rrpfreq()` semblable à `repfreq()`, mais pouvant traiter les réponses fréquentielles des systèmes à retard pur ou autre.

```
-->f=0:0.2:1 ;
-->[frq1,rep] =repfreq(Sys,f)
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 `frq1` et un vecteur complexe `rep` donnant les valeurs réelles et imaginaires du nombre complexe $Sys(j2\pi f)$. Le vecteur `frq1` est égal au vecteur `f` sauf pour des valeurs de fréquence donnant des discontinuités dans la (les) réponse(s). De même dans cette instruction on peut simplement donner les deux extrémités du vecteur fréquence, Scilab avec l'instruction `calfreq()` appelée par `repfreq()` ce chargeant de calculer d'une manière optimale (?) le vecteur fréquence (on peut aussi afficher les discontinuités). Voir le manuel de `repfreq()` et `calfreq()`.

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)
//passage d'une représentation d'état à la fonction de transfert
 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()`, par défaut la variable complexe choisie est pour un système continu s). Le système possède deux zéros : $s = -3$, $s = -2$ et trois pôles : $s = -1$, $s = -2$, $s = -3$ et 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(gp)`.

Revenons à la réponse fréquentielle d'un système.

De même on peut obtenir par l'instruction `dbphi()` à partir du vecteur complexe `rep` (voir avant dernier exercice), 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()` qui est boguée, peut donner pour la phase un décalage de phase de $-360°$ ou $+360°$, suivant les versions de Scilab et suivant l'exemple traité.

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

4.2 Les divers lieux

4.2.1 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)) ;
-->nnyquist(sys,.01,100,'2_ordresur2_ordre')
```

Pour plus de renseignements sur le lieu de Nyquist faites appel au manuel : cette instruction ne pose aucun problème particulier. Dans l'exemple (FIG. 4.1) j'ai utilisé la nouvelle fonction `nnyquist()` du répertoire `autoelem`, ceci n'est pas nécessaire pour le lieu de Nyquist (sauf pour les systèmes à retard pur).

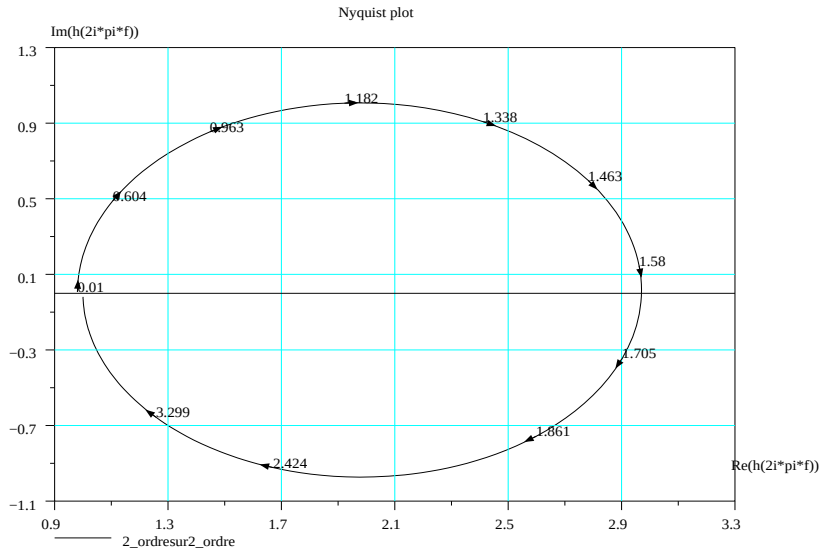


FIG. 4.1: Lieu de Nyquist

4.2.2 Représentation de Bode

La représentation de Bode est constituée de deux courbes : la première appelée courbe de gain (celui-ci exprimé en db), est la représentation de $g_{db} = 20\log|(g(j\omega)|$ comme fonction du $\log(\omega)$. La seconde courbe, est le graphe de la phase $\varphi(\omega)$, argument de $g(j\omega)$, comme fonction du $\log(\omega)$. Je rappelle que la phase $\varphi(\omega)$ (exprimée en degrés) 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 (FIG. 4.2). 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()//peut être inutile
-->bbode(sys,.01,100,'2_ordresur_2_ordre')
```

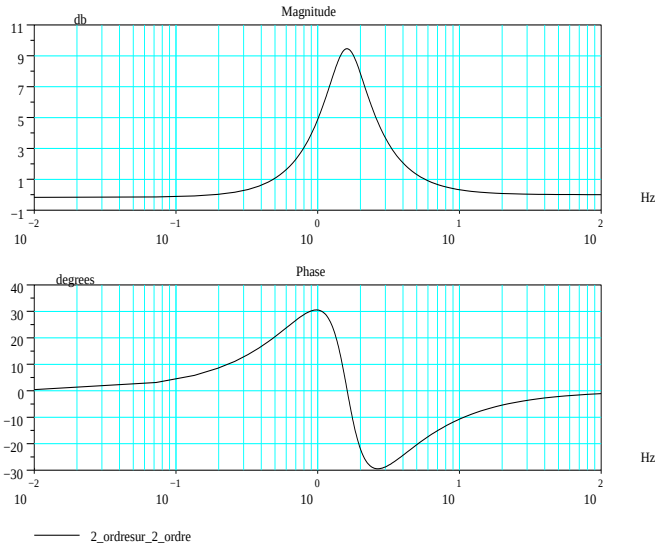


FIG. 4.2: Lieu de Bode avec bbode

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. On pourra avantageusement (à cause du bogue de `phasemag()`), utiliser `nnyquist()` et `bbode()` ou `bblack()` et `ggainplot()` pour le tracé des lieux.

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 justifier l'utilisation de `bbode` au lieu de `bode` faites les exercices suivants[3] :

```
sl=syslin('c',(100+6*s)/(100+6*s+s*s))
sl1=syslin('c',(100+6.1*s)/(100+6*s+s*s))
bode([sl;sl1],.01,100)
xset('window',1)
bbode([sl;sl1],.01,100)
```

Il n'y a pas de grosses différences entre les deux systèmes et pourtant regardez la différence sur les phases (FIG. 4.3, FIG. 4.4).

Voyez le résultat avec `bbode`.

[3] Cet exercice n'est valable qu'avec des versions antérieures à la version 2.7 de Scilab. Mais si votre sytème possède plusieurs intégrateurs, alors vous avez un bogue sur la phase avec les lieux de Bode et Black. Faire l'exercice suivant : `sl=syslin('c',1/(s*s+s*s*s))` ; `bbode(sl,.01,1)` puis `xset('window',1) bode(sl,.01,1)`.

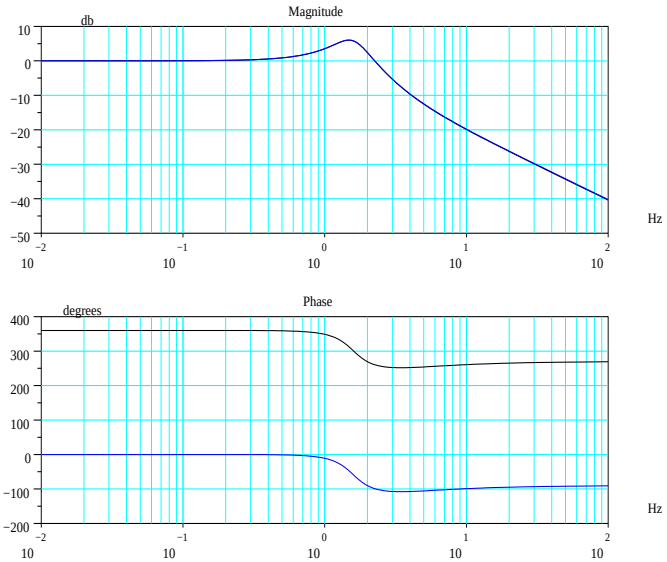


FIG. 4.3: Deux systèmes presque identiques avec bode

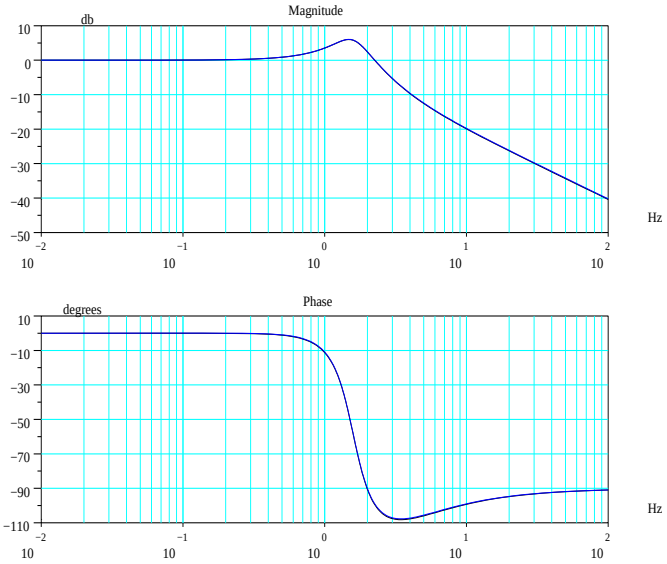


FIG. 4.4: Deux systèmes presque identiques avec bbode

4.2.3 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 g en décibels du système. Voici une session Scilab permettant de mettre en oeuvre l'instruction `bblack()`(FIG. 4.5). La même remarque s'applique avec le couple `bode bbode` et `black bblack`.

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

Vous remarquerez, que la commande `bblack()` affiche en plus de la courbe demandée, le lieu iso-gain 2,3db, valeur utile pour la synthèse d'un système ainsi que les axes $0\,db$, $-180°$.

4.2.4 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()
-->bblack(sl,.01,100)
-->chart([-6,-2,-.5,1,3,6,9,12,20],[-5,-50],list(1,0,2,3))
```

La représentation graphique est donné à la figure qui suit (FIG. 4.6) :

La première instruction graphique, `xbasc()` va ouvrir une fenêtre graphique vide (ce qui n'est pas obligatoire, car l'instruction qui suit à savoir `bblack()` ouvre automatiquement une fenêtre), puis va 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 même si j'anticipe un peu, voici quelques instructions qui peuvent être utiles pour déterminer quelques caractéristiques de systèmes bouclés : on étudiera cela plus profondément au chapitre concernant la synthèse des systèmes.

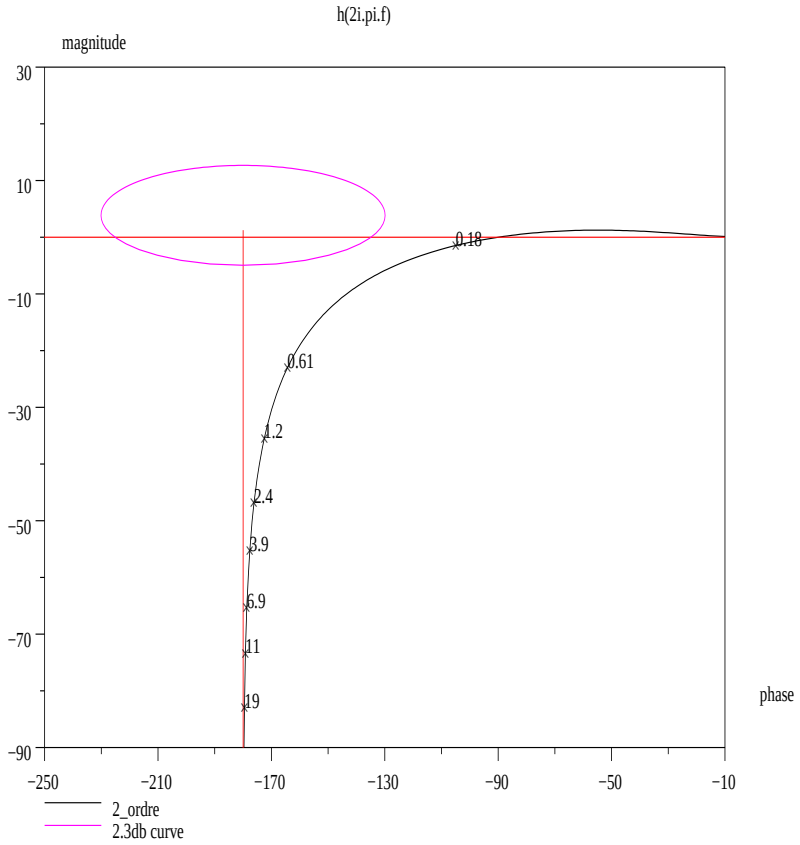


FIG. 4.5: Lieu de Black

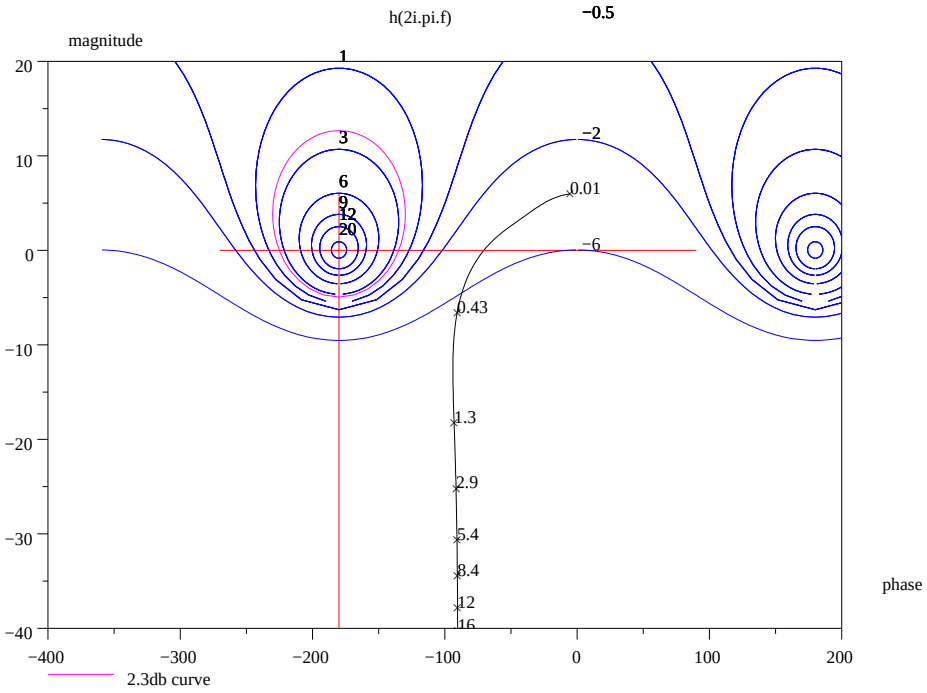


FIG. 4.6: Abaque de Black

4.2.5 Les caractéristiques d'un système à partir des lieux

```
-->sl=syslin('c',fr)/s
 sl =
                2
   2 + 3s + s
-----------------
      2      3   4
s + 3s + 2.5s + s
-->[mg,f1]=g_margin(sl)
 f1 =
  0.4090484
 mg =
  14.271982
-->[mp,f2]=p_margin(sl)
 f2 =
  0.1849030
 mp =
  - 151.64778
```

On voit bien dans l'avant dernier 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 $-\pi$; j'ai donc rajouté un intégrateur au système linéaire précédent en multipliant l'ancien système linéaire par le rationnel $\frac{1}{s}$ et obtenu, sans le redéfinir, un nouveau système linéaire continu. On voit donc apparaître la marge de gain (`mg`) et la fréquence correspondante (`f2`).

Quant à l'instruction `p_margin()`, elle donne le vecteur phase du système à une pulsation (fréquence) pour laquelle le gain vaut 1 (fréquence de coupure à $0db$) : vous ferez attention que cette instruction peut vous renvoyer un vecteur (colonne attention !) 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(mp)-abs(mp)
mp =
  28.352224
```

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 ?

```
-->fres=freson(sl)
!--error 9999
infinite gain at zero frequency
```

```
at line 8 of function freson called by :
fres=freson(sl)
```

Le résultat obtenu est normal car le système possède un intégrateur donc a un gain infini à la fréquence zéro. Réalisons maintenant un système bouclé à retour unitaire avec $sl(s)$ comme chaîne d'action :

```
-->sb=sl/(1+sl)
 sb =
                    2
         2 + 3s + s
  ----------------------
              2      3   4
  2 + 4s + 4s + 2.5s + s
-->fres=freson(sb)
 fres =
  0.1958009
```

Le système bouclé est facilement réalisé n'est ce pas. Utilisons la nouvelle fonction `dbphifr()`[4] afin de déterminer le facteur de résonance en boucle fermé (valeur `D`).

```
-->[D,P,FR]=dbphifr(sb,freson(sb))
 FR =
  0.1958009
 P =
  - 88.931411
 D =
  6.3927001
```

4.2.6 Rappel de cours, diagrammes asymptotiques de Bode : systèmes à déphasage minimaux et non minimaux

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.

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 l'équation suivante :

$$\varphi(\omega) = \frac{2\omega}{\pi} \int_{-\infty}^{+\infty} \frac{\ln A(\alpha) - \ln A(\omega)}{\alpha^2 - \omega^2} d\alpha$$

[4] Cette instruction renvoie trois vecteurs, `D` le vecteur gain en db, `P` le vecteur phase en degrés et retourne le vecteur fréquence (qui est une entrée pour la fonction).

Dans cette relation, φ représente le déphasage en fonction de la pulsation et A est le gain en fonction de cette même pulsation ($A = |g(j\omega)|$ ou $g(s)$ est la transmittance isochrone du système considéré).

Si l'on construit un diagramme en ayant factorisé (factorisation de Bode) le système avec un terme constant (gain statique, en position, en vitesse ... 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 $(1 + \tau_2 s + \tau_3 s^2)$ (s étant la variable symbolique, avec $(\tau_2^2 - 4\tau_3) < 0$ et $\tau_3 > 0$), 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 intempestif, 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 $+180°$ (voir aussi le diagramme de Black et position du point -1 pris par convention à $0\,db$ et $-180°$).

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 τ_2 qui peuvent être négatives dans les expressions $(1+\tau_1 s)$ et/ou $(1+\tau_2 s+\tau_3 s^2)$. En multipliant le numérateur **et** le dénominateur de la transmittance par les termes $(1-\tau_1 s)$ ou par des termes de la forme $(1-\tau_2 s+\tau_3 s^2)$, 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 :

$$g(s) = \frac{1 - s + s^2}{s(1 + s)}$$

on peut écrire cette expression sous la forme :

$$g(s) = \frac{1 - s + s^2}{s(1 + s)} \frac{1 + s + s^2}{1 + s + s^2}$$

soit encore :

$$g(s) = \frac{1 - s + s^2}{1 + s + s^2} \frac{1 + s + s^2}{s(1 + s)}$$

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 ce chapitre des systèmes à retard pur qui ne sont pas des systèmes à déphasage minimaux et dont le modèle mathématique n'est pas une fraction rationnelle.

4.3 Etude de systèmes simples

Comme lors de la synthèse d'un système bouclé on compare souvent ce système bouclé à un système du premier ou du second ordre, il est donc intéressant maintenant de donner les réponses temporelles et fréquentielles de ces systèmes élémentaires.

4.3.1 Le premier ordre

Le pôle

Un seul pôle caractérise un système du premier ordre il est noté : p_1. La transmittance isochrone a pour équation :

$$g(s) = \frac{k_b}{1+\tau s} = \frac{k_e}{s-p_1} \; avec \; \tau = -\frac{1}{p_1} \; et \; k_b = -\frac{k_e}{p_1} \geq 0$$

Je donne ici les deux formulations celle de Bode et celle d'Evans : τ est la constante de temps (supposée positive, système stable), p_1 le pôle, k_b le gain statique.

Les réponses impulsionnelle, indicielle

La condition initiale étant nulle, l'instant initial aussi, les réponses impulsionnelle et indicielle valent respectivement :

$$h(t) = \frac{k_b}{\tau}\exp(-\frac{t}{\tau}) \; et \; y_1(t) = k_b(1-\exp(-\frac{t}{\tau}))$$

Le gain statique k_b est la réponse indicielle permanente et pour $t = \tau$ la réponse $y_1(t)$ vaut 63% de la réponse finale. Enfin pour $t = 3\tau$ cette même réponse vaut 95% de la réponse permanente (temps de réponse à 5%). Vous pourrez avec Scilab tracer, en prenant $\tau = 1$, ce qui revient à prendre pour abcisse non pas t mais $\frac{t}{\tau}$, les réponses correspondantes (FIG. 4.7).

La réponse fréquentielle

Quant à la réponse fréquentielle (le lieu de Nyquist est un demi cercle), on a pour le gain en db et la phase en degrés :

$$\begin{aligned} g_{db} &= 20\log(k_b) - 10\log(1+\tau^2\omega^2) \\ \varphi &= -\arctan(\tau\omega) \end{aligned}$$

Pour la pulsation de cassure $\omega = \frac{1}{\tau}$ l'atténuation par rapport au gain statique est de $3db$ et le déphasage correspondant est de $-45°$. Si on cherche le diagramme

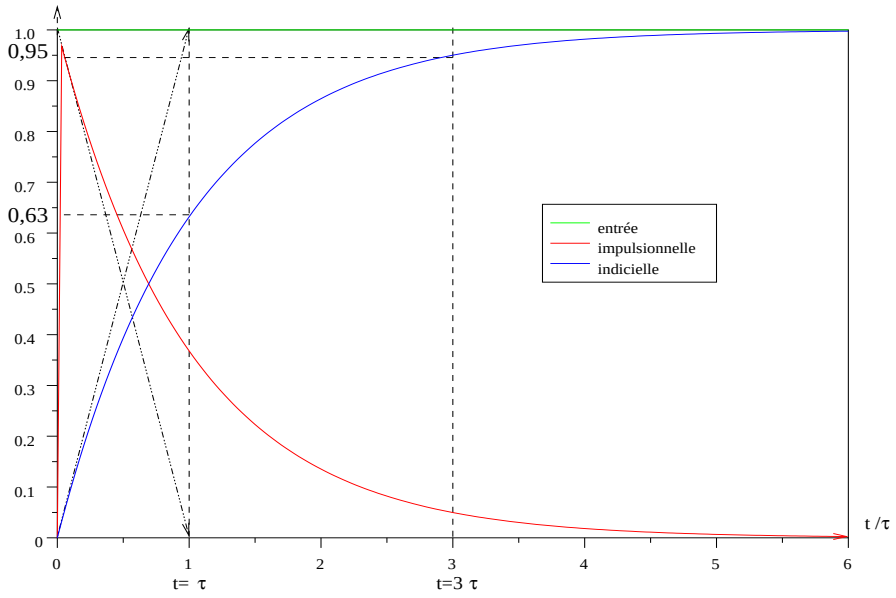


FIG. 4.7: Réponses d'un premier ordre

asymptotique de Bode, il est constitué pour la courbe de gain des deux droites $20\log(k_b)$ et $20\log(k_b) - 20\log\tau - 20\log\omega$ qui se coupent au point $[\omega = \frac{1}{\tau}, 20\log(k_b)]$, pour la courbe asymptotique de phase on a les deux horizontales $0°$ et $-90°$.

4.3.2 Le second ordre

Les pôles

On a l'habitude de mettre un système du second ordre sous la forme canonique suivante :

$$g(s) = \frac{k_b}{1 + 2\xi\frac{s}{\omega_n} + \frac{s^2}{\omega_n^2}}$$

Avec comme paramètres du système : k_b le gain statique, ω_n la pulsation naturelle et ξ le coefficient d'amortissement. Suivant la valeur de ξ on peut avoir des pôles réels ($\xi \geq 1$) qui valent : $p_{1,2} = -\xi\omega_n \pm \omega_n\sqrt{\xi^2 - 1}$ ou des pôles complexes conjugués ($\xi \leq 1$) qui dans ce cas valent : $p_{1,2} = -\xi\omega_n \pm j\omega_n\sqrt{1 - \xi^2}$ (FIG. 4.8). Dans le cas où ces deux pôles sont réels ($\xi \geq 1$), la transmittance est le produit de deux transmittances du premier ordre et on ce ramène au cas précédent.

Les réponses impulsionnelle, indicielle

Les condition initiales étant nulles, l'instant initial aussi, les réponses impulsionnelle et indicielle valent respectivement (FIG. 4.9, FIG. 4.10) :

$$h(t) = k_b\frac{\omega_n^2}{\omega_p}\exp(-\xi\omega_n t)\sin(\omega_p t) \; avec \; \omega_p = \omega_n\sqrt{1 - \xi^2}$$

$$y_1(t) = k_b[1 - \frac{\omega_n}{\omega_p}\exp(-\xi\omega_n t)\sin(\omega_p t + \theta)] \; avec \; \theta = \arccos(\xi)$$

On appelle la pulsation ω_p la pulsation propre qui vaut la valeur absolue de la partie imaginaire des pôles, $\omega_p = |I(p_{1,2})|$. Quant au facteur de l'exponentielle, c'est la partie réelle des pôles $-\xi\omega_n = R(p_{1,2})$: ces formules sont valables pour $\xi \leq 1$.

Si $\xi \geq 1$, on peut remplacer dans ces formules le sin() par sinh() et $\omega_n\sqrt{1 - \xi^2}$ par $\omega_n\sqrt{\xi^2 - 1}$.

Voici le programme donnant les pôles, et les réponses impulsionnelle et indicielle d'un ensemble de systèmes du second ordre pour $\xi = [\ 0,2 \quad 0,3 \quad 0,4 \quad 0,6 \quad 0,8\ ]$. De même on résume dans un tableau les principales propriétés concernant la réponse indicielle d'un second ordre oscillatoire.

```
//pôles et zéros
s=%s ;g=syslin('c',1/(s*s+s+1)) ;plzr(g)
//création des second ordre
den=(s*s+1)*ones(5,1)+2*s*[.2 ;.3 ;.4 ;.6 ;.8] ;
fr=poly(1,'s','c')*ones(5,1)./den ;
g=syslin('c',fr) ;
```

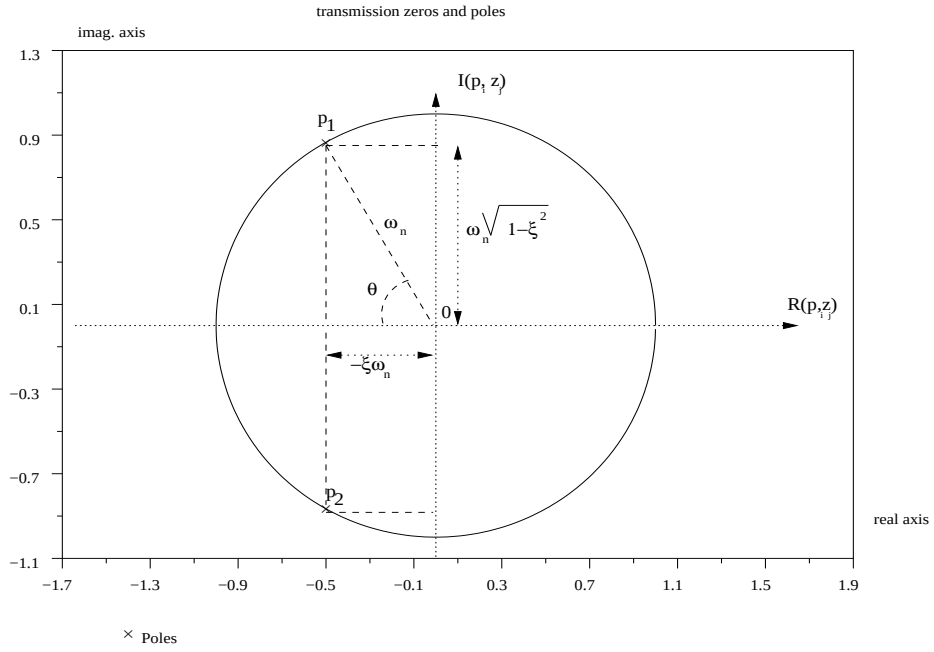


FIG. 4.8: Pôles d'un second ordre

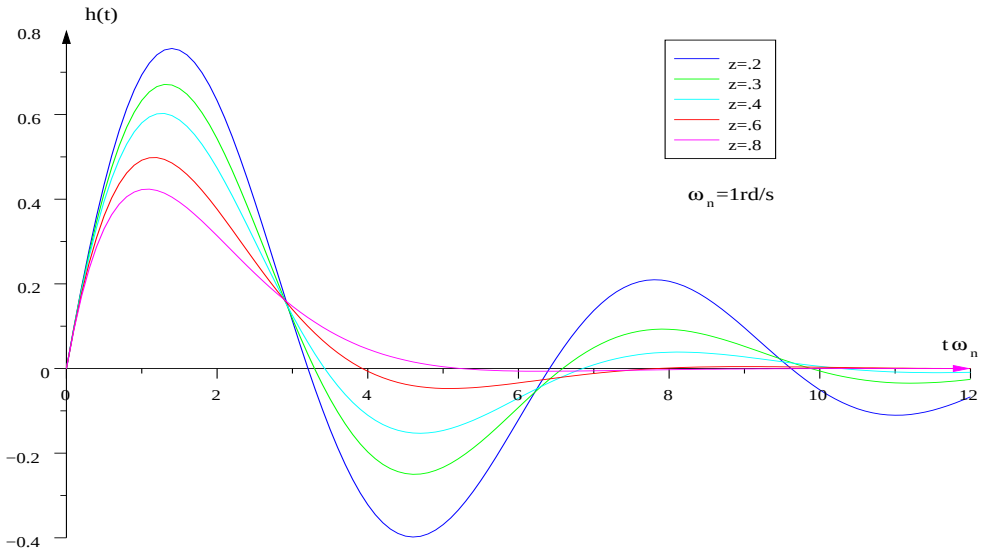


FIG. 4.9: Réponse impulsionnelle

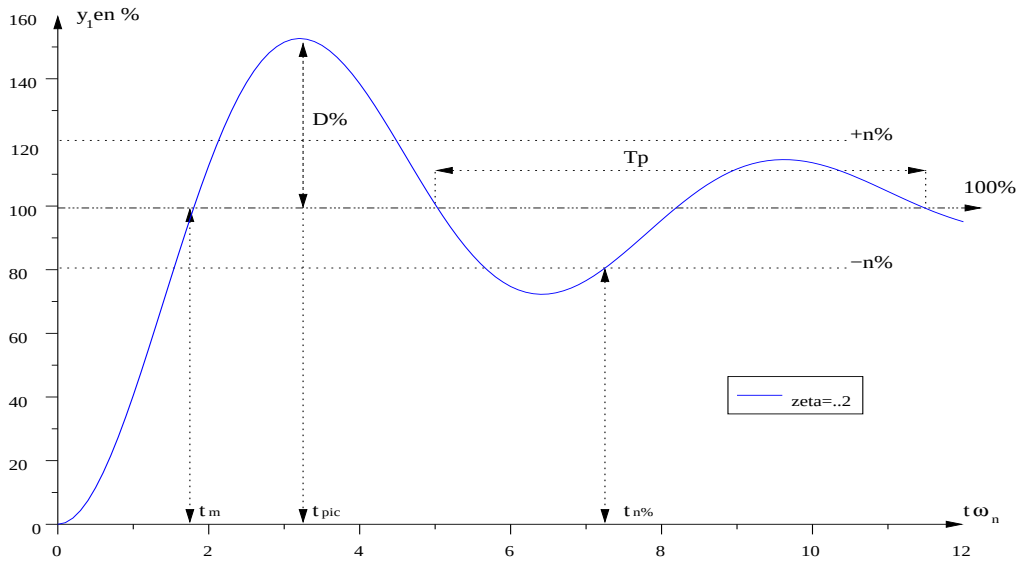


FIG. 4.10: Réponse indicielle

```
t=linspace(0,12,121);h=csim('imp',t,g);
plot2d(t',h',style=[2;3;4;5;6],axesflag=5)
legends(['z=.2';'z=.3';'z=.4';'z=.6';'z=.8'],[2,3,4,5,6])
y1=csim('step',t,g);xset('window',1)
plot2d(t',y1',style=[2;3;4;5;6],axesflag=5)
legends(['z=.2';'z=.3';'z=.4';'z=.6';'z=.8'],[2,3,4,5,6])
```

Nous donnons la réponse indicielle d'un seul système pour $\xi = 0,2$.

```
xset('window',2)
plot2d(t',y1(1,:)',style=2,axesflag=5)
legends('zeta=.2',2)
```

Enfin voici un tableau donnant les principaux résultats issus de la réponse indicielle :

Caractéristiques	Valeurs
Temps de montée	$t_m = \frac{1}{\omega_n\sqrt{1-\xi^2}}(\pi - \arccos\xi)$
Temps de pic	$t_{pic} = \frac{\pi}{\omega_n\sqrt{1-\xi^2}}$
Pseudo-période, période propre	$T_p = \frac{2\pi}{\omega_n\sqrt{1-\xi^2}}$
Pseudo-pulsation, pulsation propre	$\omega_p = \omega_n\sqrt{1-\xi^2}$
Dépassement	$D\% = 100\exp(-\pi\frac{\xi}{\sqrt{1-\xi^2}})$
Rapport de 2 maximas successifs	$\frac{D_1}{D_2} = \exp(2\pi\frac{\xi}{\sqrt{1-\xi^2}})$
Temps de réponse à n%, souvent n=±5	$t_{r\,n\%} \approx \frac{1}{\omega_n\xi}\ln(\frac{100}{n})$
Temps de réponse à 5%	$t_{r\,5\%} \approx \frac{3}{\omega_n\xi}$
Nombre d'oscillations complètes	$n \approx Q = \frac{1}{2\xi}$

La réponse fréquentielle

Nous reprenons l'exercice précédent et traçons les lieux de Bode (complet), de Bode (gain) (FIG. 4.11) et de Black (FIG. 4.12) par les instructions :

```
-->com=['z=.2';'z=.3';'z=.4';'z=.6';'z=.8'];
-->bbode(g,.01,1,com)
-->xset('window',1)
-->ggainplot(g,.01,1,com)
-->xset('window',2)
-->bblack(g,.01,1,com)
```

Comme nous l'avons fait pour la réponse indicielle nous allons donner sous forme de tableau, les principaux résultats relatifs aux réponses fréquentielles.

Caractéristiques	Valeurs
Pulsation de résonance	$\omega_r = \omega_n\sqrt{1-2\xi^2}$
Pulsation de coupure à $0db$	$\omega_{co} = \omega_r\sqrt{2}$
Facteur de résonance	$Q = \frac{1}{2\xi\sqrt{1-\xi^2}}$
Pulsation de coupure à$-6db$	$\omega_{-6} =$
Relation temps-fréquence	$\omega_{co}t_{r\,5\%} \approx \pi$

En posant $u = \frac{\omega}{\omega_n}$ on peut exprimer pour un système du second ordre dont le gain statique est égal à 1, dont la pulsation naturelle vaut ω_n et le coefficient d'amortissement ξ, le gain complexe

$$g(ju) = \frac{1}{1-u^2+2j\xi u}$$

ce qui donne pour le gain et la phase

$$|g(ju)| = \frac{1}{\sqrt{(1-u^2)^2+4\xi^2u^2}} \; et \; \varphi = -\arctan(2\xi\frac{u}{1-u^2})$$

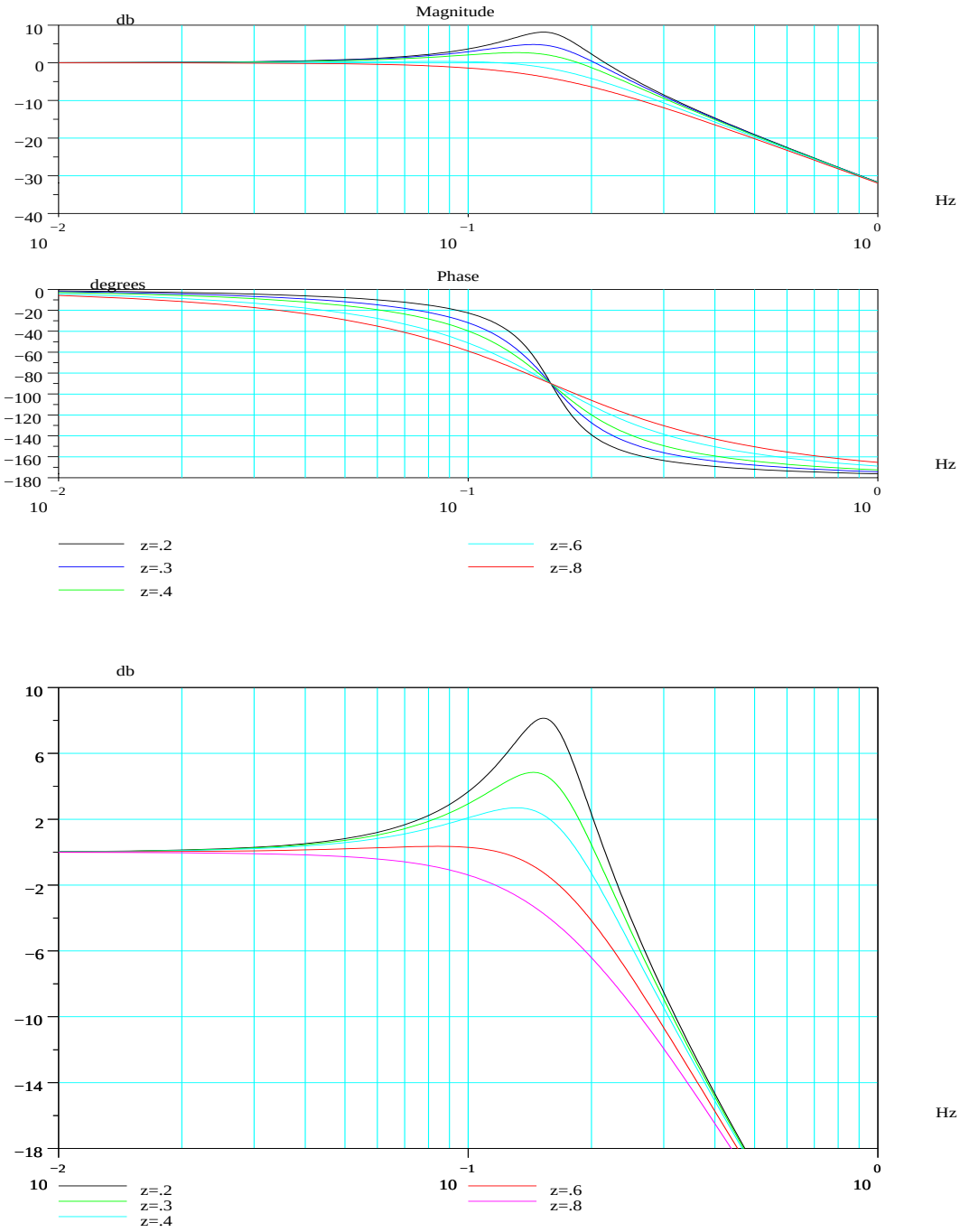


FIG. 4.11: Lieu de Bode, second ordre

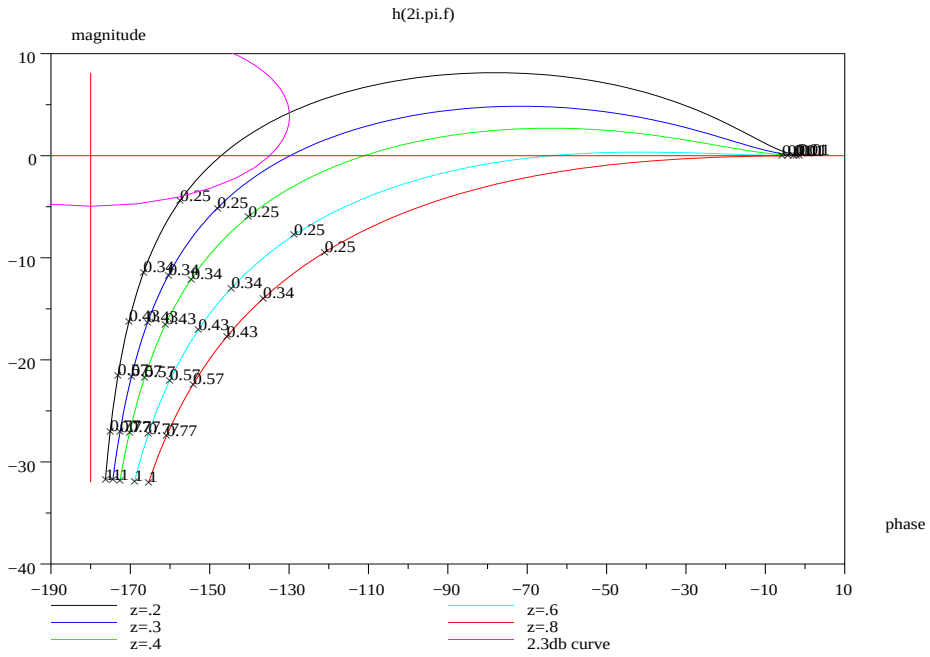


FIG. 4.12: Lieu de Black, second ordre

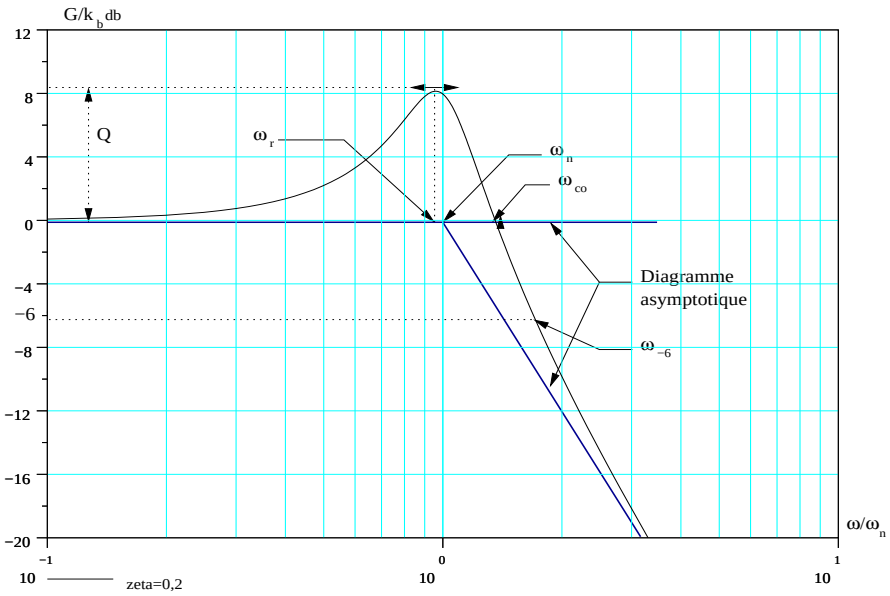


FIG. 4.13: Gain, second ordre

ou encore pour le gain en décibels

$$g_{db} = -10\log[(1-u^2)^2 + 4\xi^2 u^2]$$

Sur la figure précédente (FIG. 4.13) j'ai mis en abcisse le logarithme décimal de $u = \frac{\omega}{\omega_n}$ et non la fréquence.

5 Etude de la stabilité d'un système, marges de stabilité

5.1 Etude de la stabilité

5.1.1 Rappels sur la stabilité des systèmes

Pour un système à modèle linéaire, la stabilité est la propriété selon laquelle ce système écarté de sa position d'équilibre par une sollicitation extérieure (entrée ou perturbation), revient à cette position d'équilibre quand la sollicitation a cessé. L'étude de la stabilité se fait donc sur la base du régime transitoire ou du régime libre.

Pour tout système bouclé ou non, la **stabilité est une condition impérative**, c'est l'existence même du système qui est en jeu. Comme je le fais en cours à partir du théorème de convolution, on introduit la E.B.S.B stabilité : B.I.B.O stability pour les anglosaxons, entrée bornée, sortie bornée. La conséquence sur un système à modèle linéaire est la suivante : **Un système sera dit stable au sens strict s'il ne possède pas de pôle dans le demi plan droit du plan complexe**. Ceci revient à dire que pour un modèle d'état, la matrice d'état, notée généralement A aura **toutes ses valeurs propres dans ce même demi plan**. Ceci revient à dire aussi que sa réponse impulsionnelle sera décroissante en fonction du temps. Quant aux systèmes qui possèdent des pôles simples en nombre fini sur l'axe imaginaire pur, ou un pôle unique à l'origine, on dira que ces systèmes sont stables au sens large (déplacement d'un ou plusieurs pôles du demi plan gauche au demi plan droit par variation d'un paramètre du modèle).

5.1.2 Calcul des pôles d'un système

Si le modèle a tous ses paramètres constants et connus, la stabilité d'un système revient à chercher les valeurs des racines d'une équation (ou de plusieurs) de degré n. Ceci revient aussi (avec un choix de modèle d'état), à rechercher les valeurs propres de la matrice d'état A. Numériquement c'est exactement le même problème et cela peut être fait avec Scilab par les instructions `roots()` ou `spec()` avec les problèmes de précision que cela comporte dans le cas de pôles multiples.

5.1.3 Critère de Routh-Hurwitz

Historiquement ce problème de stabilité a été résolu par essentiellement deux mathématiciens (après 1850) Hurwitz (1895) et Routh (1877). Ces deux mathématiciens ont donné des critères très semblables, critères qui mettent en oeuvre les coefficients d'un équation

de degré n. A ce sujet, ces travaux viennent de l'étude de la régulation de la vitesse des machines à vapeur par l'utilisation du régulateur de Watt (1788), on lira l'article `http://fr.wikipedia.org/wiki/James_Watt`

Définition du critère de Routh-Hurwitz

C'est un critère algébrique permettant de calculer le nombre de racines à parties réelles positives (modèle instable) du polynôme caractéristique du modèle entrée-sortie du système. Ce polynôme noté $D(s)$ est de degré n.

$$D(s) = a_n s^n + a_{n-1} s^{n-1} + ... + a_1 s + a_0$$

Deux conditions doivent être tout d'abord vérifiées :

1. Tous les coefficients de $D(s)$ doivent exister.
2. Tous les coefficients de $D(s)$ doivent être de même signe.
3. Si ces deux conditions sont vérifiées, alors on construit la table de Routh :

ligne 1	$a_{1,1} = a_n$	$a_{1,2} = a_{n-2}$	$a_{1,3} = a_{n-4}$	$a_{1,4} = a_{n-6}$	...	...
ligne 2	$a_{2,1} = a_{n-1}$	$a_{2,2} = a_{n-3}$	$a_{2,3} = a_{n-5}$	$a_{2,4} = a_{n-7}$	...	...
ligne 3	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	...	...
ligne 4	$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	$a_{4,4}$	...	...
ligne 5	...	...	...	...	...	...

Les valeurs de la première colonne du tableau sont appelés les pivots, c'est le coefficient $a_{i,1}$ et le coefficient $a_{i,j}$ vaut :

$$a_{i,j} = \frac{a_{i-1,1} a_{i-2,j+1} - a_{i-2,1} a_{i-1,j+1}}{a_{i-1,1}}$$

La table de Routh associée au polynôme $D(s)$ d'ordre n comporte $n+1$ lignes, et la condition de stabilité s'énonce : **il y a autant de pôles instables pour le système que de changements de signe dans la première colonne de la table de Routh**.

Dans le cas particulier où un pivot est nul ou le tableau a une ligne de zéros, alors :

1. Si dans le calcul des éléments du tableau de Routh l'un des pivots est nul, les autres coefficients de la ligne étant tous non nuls, alors il existe au moins une racine de $D(s)$ dans le demi plan droit.
2. Si tous les coefficients de la ligne correspondante sont nuls, il existe un ou des couples de racines à partie réelle nulle, (le système est théoriquement un oscillateur oscillant à une pulsation dont la valeur est la valeur absolue de la partie imaginaire de ces deux racines).

Malgré cela on peut continuer à construire le tableau en remplaçant ce zéro par un nombre ε très petit quelconque (sauf dans le cas où tous les termes sont nuls), et l'on fera tendre ce petit nombre vers zéro.

Remarque : On peut facilement dans le particulier où une ligne est nulle déterminer les racines imaginaires pures du polynôme caractéristique : en prenant la ligne qui la

précède, on forme un polynôme en α^2 dont les coefficients ordonnés dans le sens des puissances décroissantes sont les coefficients de cette ligne et on résoud cette équation.

L'étude algébrique de la stabilité des systèmes peut être faite par Scilab 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). Dans cet exercice on a quatre changements de signe dans la première colonne du tableau de Routh, on a quatre racines à parties réelles positives : $\left[\begin{array}{cccc} 6 & 3 & 1+j & 1-j \end{array}\right]$.

```
-->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.

Nous allons faire deux exercices qui permettent de calculer les racines imaginaires pures dans le cas où le système est à la limite de la stabilité.

```
-->den=s^4+4*s^3+7*s^2+16*s+12

den =
              2    3   4
12 + 16s + 7s + 4s + s
-->tab=routh_t(den)

tab =
! 1.          7.     12. !
! 4.         16.      0. !
! 3.         12.      0. !
! 8.882E-16  0.       0. !
! 12.        0.       0. !
-->tab=clean(tab)
tab =
```

```
! 1.   7.  12. !
! 4.  16.   0. !
! 3.  12.   0. !
! 0.   0.   0. !
! 12.  0.   0. !
```

La ligne quatre possède un pivot très petit, les autres coefficients de cette ligne sont nuls. Scilab a remplacé ce pivot par un nombre très petit et continue le calcul : par l'instruction `clean()` on obtient la vrai table. On peut maintenant construire le polynôme donnant les racines imaginaires (il suffit de former le polynôme en α^2 dont les coefficients dans le sens des puissances décroissantes sont les termes de la ligne au dessus du pivot nul) ; ce polynôme vaut :

$$P(\alpha^2) = 3\alpha^2 + 12$$

soit en résolvant $P(\alpha^2) = 0$ on a : $\alpha = \pm 2j$. Le système oscille à la pulsation de $\omega = 2rd/s$. De plus dans le tableau de Routh il n'y a pas de changement de signe : les autres pôles sont stables. Vérifions ceci.

```
-->poles=clean(roots(den))
 poles =
! - 1. !
!  2.i !
! -2.i !
! -3.  !
```

Le deuxième exercice est plus un cas d'école car il s'applique à un système qui est de toute façon instable : tous les coefficients ne sont pas de même signe.

```
-->s=%s ;
-->den=s^6-5*s^5+11*s^4-25*s^3+34*s^2-20*s+24
 den =
                2     3     4    5   6
  24 - 20s + 34s - 25s + 11s - 5s + s

-->table=routh_t(den)
 table =
!  1.          11.          34.   24. !
! -5.         -25.         -20.    0. !
!  6.          30.          24.    0. !
!  1.110E-15   8.882E-16    0.    0. !
! 25.2        24.           0.    0. !
! -1.692E-16   0.           0.    0. !
! 24.           0.          0.    0. !

-->tab=clean(table)
 tab =
```

```
!  1.    11.    34.    24. !
! -5.   -25.   -20.     0. !
!  6.    30.    24.     0. !
!  0.     0.     0.     0. !
! 25.2   24.     0.     0. !
!  0.     0.     0.     0. !
! 24.     0.     0.     0. !
```

Vérifions ces résultats en calculant les racines de ce polynôme.

```
-->poles=clean(roots(den))
 poles =
!    i !
!   -i !
!  2.i !
! -2.i !
!  2.  !
!  3.  !
```

Il y a deux changements de signe dans la première colonne du tableau de Routh, donc deux pôles instables. De plus ce polynôme possède des racines imaginaires pures vérifiant l'équation, ligne trois du tableau :

$$P(\alpha^2) = 6\alpha^4 + 30\alpha^2 + 24 = 0$$

soit $\alpha_1 = 2j$, $\alpha_2 = -2j$, $\alpha_3 = j$, $\alpha_4 = -j$.

Critère de Routh-Hurwitz et les systèmes bouclés

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]=kpure1(sl)//préférer kpure1[1]
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).

De même l'instruction `kpure()` 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.

Vous verrez 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, on reviendra en temps utile sur cette instruction.

5.1.4 La stabilité d'un système bouclé par le critère de Nyquist-Cauchy

Le but de ce critère est de donner la **stabilité d'un système bouclé** à partir de la **connaissance de la boucle ouverte** seulement : c'est un **critère géométrique**.

Rappel du critère de Nyquist-Cauchy

Avant de rappeler le lemme de Cauchy qui permet de démontrer le critère de Nyquist-Cauchy, je voudrais définir dans quelles conditions ce critère peut s'appliquer.

Tracé de fonctions complexes d'une variable complexe Un exemple évident de ce tracé est le lieu de Nyquist d'un système. En effet le modèle d'un procédé est souvent un rapport de deux polynômes de la variable complexe s : pour tracer le lieu de Nyquist on remplace cette variable par le complexe pur $j\omega$ et on trace une courbe en coordonnées paramétriques $Real(g(j\omega)) = fonction1(\omega)$ *et* $Imag(g(j\omega)) = fonction2(\omega)$ et ceci pour $\omega \in [\ 0 \quad +\infty\ [$. Il est évident que l'on peut très bien à l'aide du logiciel Scilab

[1] kpure1() est une macros contenue dans le répertoire que je fournis autoelem, kpure1() considère que le gain limite est positif ou négatif, mais pas nul.

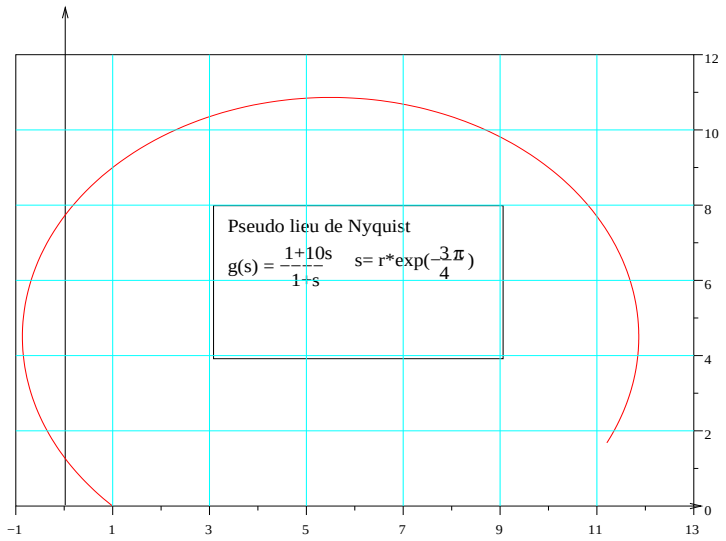


FIG. 5.1: Pseudo lieu de Nyquist

faire le tracé d'une courbe en paramétrique : voici un exemple de « pseudo-lieu » de Nyquist (FIG. 5.1). On se donne un système de transmittance :

$$g(s) = \frac{1+10s}{1+s}$$

et on désire tracer les transformés des points du plan complexe situés sur la quatrième bissectrice, ces points ont pour affixes $w = r\exp(\frac{3j\pi}{4})$ avec $r \in \left[\ 0 \quad 5\ \right]$. Le lieu de Nyquist étant les transformés des points ayant pour affixes $w = r\exp(\frac{j\pi}{2})$ avec $r \in \left[\ 0 \quad +\infty\ \right[$.

```
-->s=%s ;g=syslin('c',(1+10*s)/(1+s)) ;
-->r=linspace(0,5,501) ;
-->w=r*exp(%i*3*%pi/4) ;
-->z=horner(g,w) ;
//ou mieux z=freq(g('num'),g('den'),w) ;
-->x=real(z) ;y=imag(z) ;
-->plot2d(x(:)',y(:)',style=5,axesflag=3)
-->xgrid(4)
```

Une autre façon de procéder est d'utiliser la nouvelle utilisation de Nyquist (Bode, Black) du répertoire `autoelem`.

```
-->s=%s ;
-->uns=syslin('c',poly(1,'s','c'),poly(1,'s','c')) ;
```

```
-->g1=uns;//le système de transmittance 1(s)
-->g2=syslin('c',(1+10*s)/(1+s));
--> ;getf("/home/lpovy/LYX/pseudonyq.sci");
-->G=list([g1;g2],list(pseudonyq,0))
```

Commentaire : Au gain et à la phase de $g1(s)$ (ici $0\,db$ et 0°) on rajoute le gain et la phase de la fonction `pseudonyq.sci`, au gain et à la phase de $g2(s)$ de transmittance $g2(s) = \frac{1+10s}{1+s}$ on rajoute le gain et la phase donné par le retard pur de valeur $T = 0$ (idiot n'est ce pas) : en effet j'ai construit ces nouvelles fonctions pour étudier en particulier, les systèmes à retards purs.

```
 G =
  G(1)
! 1       !
! -       !
! 1       !
!         !
! 1 + 10s !
! -------!
! 1 + s   !
 G(2)
 G(2)(1)
[db1,ph1]=function(fr)
 G(2)(2)
  0.
-->nnyquist(G,.001,10,['pseudo nyquist';'nyquist'])
Warning :redefining function : Ti
```

Voici la fonction permettant de tracer le pseudo lieu de Nyquist (FIG. 5.2) :

```
function[db1,ph1]=pseudonyq(fr)
w=2*%pi*fr*exp(%i*%pi*3/4)
//c'est ici que l'on défini la trajectoire du point M.
//du plan complexe s
num=1+10*w
den=1+w
db1=-(20/log(10))*(log(abs(den))-log(abs(num)))
ph1=-(180/%pi)*(atan(imag(den),real(den))-atan(imag(num),real(num)))
endfunction
```

Vous allez me dire que le programme est plus compliqué, mais je peux comparer un lieu de Nyquist à un pseudo lieu : je reviendrais sur cette extension.

Fonctions analytiques, contours, transformations conformes En mathématique on dit qu'une fonction complexe $F(s)$ est analytique dans un domaine, si en tout point P d'affixe s_0 de ce domaine du plan complexe (s), la dérivée $\frac{dF}{ds}|_{s=s_0} = \lim_{s\to s_0} \frac{F(s)-F(s_0)}{s-s_0}$ existe et est unique.

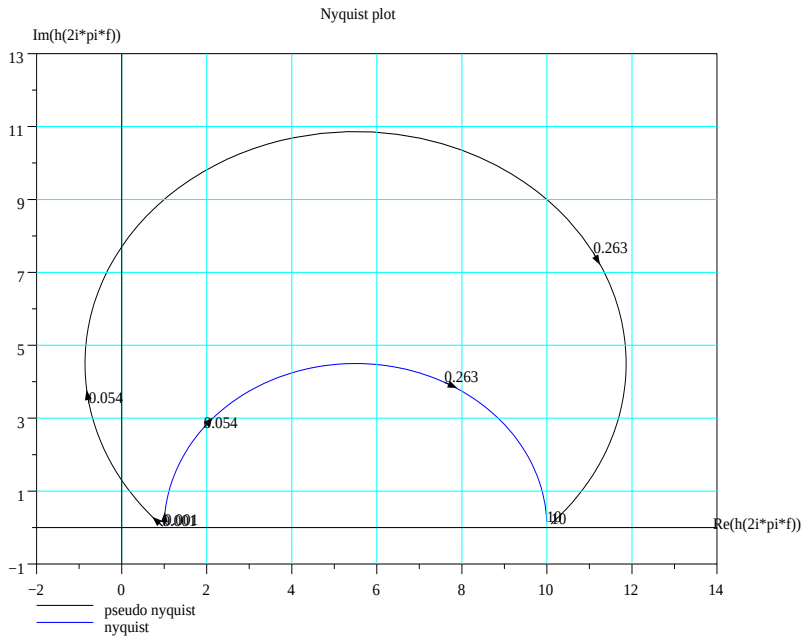


FIG. 5.2: Nyquist, pseudo Nyquist

Les fonctions de transfert des sytèmes physiques considérés ici sont analytiques dans le plan complexe, sauf aux pôles de ces fonctions de transfert. De même on admettra comme connu dans le plan complexe la notion de contours fermés sans point double.

De même on doit, enfin d'utilisé correctement le lemme de Cauchy qui va suivre, introduire la notion de transformation conforme. On dit qu'une transformation $F(s)$ de la variable complexe est une transformation conforme si :

- A un point d'affixe s_1 du domaine considéré correspond un et un seul point transformé d'affixe $F(s_1)$.
- La transformation est analytique sauf en un nombre fini de singularités.
- Le contour choisi doit éviter ces singularités.
- A un contour fermé du plan de la variable s correspond un contour fermé du plan transformé.
- Une transformation conforme conserve les angles des tangentes aux intersections de deux courbes.

Lemme de Cauchy On peut énoncer le lemme de Cauchy, vu en mathématiques, de la manière suivante :

Hypothèses : On se donne une transformation conforme $F(s)$ de la variable complexe s (donc analytique) et on se donne un contour C fermé sans point double englobant un domaine du plan complexe s, domaine D contenant P pôles et Z zéros de la transformation $F(s)$.

Lemme : Quand un point M parcourt le contour C dans le sens des aiguilles d'une montre, alors le point transformé M' par la transformation $F(s)$ fait un nombre de tours N autour de l'origine (du plan transformé), dans le sens inverse des aiguilles d'une montre, égal à :

$$N = P - Z$$

C'est ce lemme qui peut être utilisé pour étudier la stabilité d'un système bouclé.

Application à la stabilité d'un système bouclé, critère de Nyquist-Cauchy. Le problème que l'on se pose est l'étude de la stabilité d'un système bouclé dont la transmittance de la chaîne d'action est $G(s)$, celle de la chaîne de retour $H(s)$. La transmittance de la boucle vaut donc :

$$W(s) = \frac{G(s)}{1 + G(s)H(s)}$$

La stabilité de la boucle dépend donc des pôles de $W(s)$ qui sont les zéros de $1 + G(s)H(s)$. Mais les zéros dangereux instables, sont d'après la théorie de la stabilité, ceux situés dans le demi plan droit du plan complexe.

On va donc utilisé le lemme de Cauchy dans le sens $Z = P - N$ en se donnant :

- Un domaine du plan complexe où les **pôles de $W(s)$ ne doivent pas être situés**. Ce domaine contiendra donc tous les points situés dans le demi plan droit du plan complexe et sera entouré par un contour C_{ex} fermé sans point double.

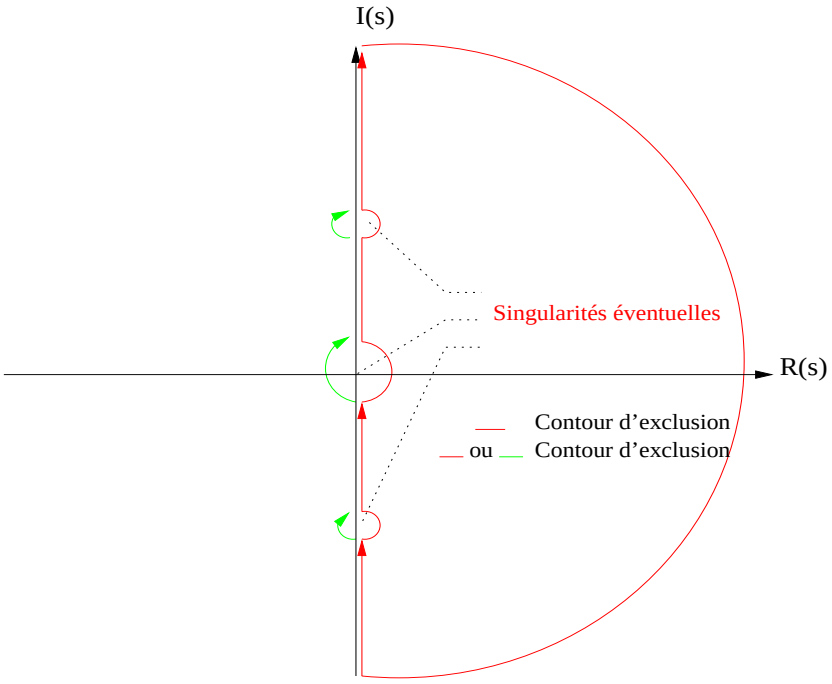


FIG. 5.3: Contour d'exclusion

- On tracera normalement les transformés des points de C_{ex} par la transformation $1+G(s)H(s)$, qui est la transformation $G(s)H(s)$ translatée de l'entier 1. Comme on ne souhaite pas faire cette translation, on comptera le nombre N de tours autour de -1.
- On comptera le nombre P de pôles de $1+G(s)H(s)$ qui sont les pôles de $G(s)H(s)$ contenus dans le domaine entouré par C_{ex} et on appliquera la relation $Z = P - N$ pour avoir le nombre de pôles instables de $W(s)$.

Contour d'exclusion C_{ex} et transformé de ce contour Pour englober l'ensemble du demi plan droit du plan complexe on l'entoure par la droite des nombres imaginaires purs (axe vertical), en prenant soin de contourner les singularités de $G(s)H(s)$ (par la droite ou la gauche) si elles sont sur cet axe : présence d'un intégrateur ou de paire(s) de pôles imaginaires purs conjugués sur l'axe vertical et on ferme le contour par un demi cercle de rayon infini (FIG. 5.3).

Quant au transformé par la transformation conforme $G(s)H(s)$ du contour d'exclusion ainsi choisi, il va être constitué de :

1. Du lieu de Nyquist de la boucle ouverte $G(s)H(s)$, points transformés de l'axe imaginaire positif .
2. Du symétrique par rapport à l'axe réel du lieu de Nyquist de la boucle ouverte

$G(s)H(s)$, en effet ce lieu est le transformé de l'axe imaginaire négatif et quand on change j en $-j$ dans la transmittance isochrone, on obtient le nombre complexe conjugué et ceci quelque soit la fréquence (parce que $G(s)H(s)$ est un rapport de deux polynômes).

3. Du transformé du grand demi cercle à l'infini qui est un point à distance finie (si le degré du numérateur de $G(s)H(s)$ est égal au degré du dénominateur), ou l'origine (si le degré du numérateur de $G(s)H(s)$ est inférieur au degré du dénominateur).
4. Enfin on cherche les transformés des demi cercles entourant les singularités de $G(s)H(s)$ situés sur l'axe imaginaire pur.
5. Puis on compte le nombre de tours que fait le point transformé autour de -1 et appliquons la relation $Z = P - N$. Un exemple explicite ceci.

Exemple Nous allons prendre un exemple de boucle ouverte possédant une singularité à l'origine. Soit

$$G_k(s) = \frac{k}{s(1+s)(1+\frac{s}{3})}$$

k est le gain d'un amplificateur et est supposé positif. Les trois pôles de la boucle ouverte sont $p_1 = 0$, $p_2 = -1$, $p_3 = -3$. Comme il y a une singularité sur l'axe vertical on contourne soit à droite (demi cercle rouge), soit à gauche (demi cercle vert) ce pôle à l'origine. Dans le premier cas $P = 0$ (pas de pôles de la boucle ouverte dans le domaine d'exclusion), dans le second cas $P = 1$, puis on réalise les opérations (FIG. 5.4) :

1. tracé du lieu de Nyquist de la boucle ouverte.
2. tracé du symétrique par rapport à l'axe réel de ce lieu.
3. tracé de la fermeture, deux cas sont à envisager :

– le contournement de l'origine se faisait par la droite, alors les points transformés de ce **demi cercle** de rayon infiniment petit, sont sur un **demi cercle** de rayon infiniment grand, pour démontrer cela posons $s = r\exp(j\varphi)$ et recherchons l'équivalent par $G_k(s)$ de ce demi cercle : $G_k(r\exp(j\varphi)) \approx \frac{k}{r}\exp(-j\varphi)$ qui est un demi cercle de rayon $\frac{k}{r}$ infiniment grand (φ varie de π), on a deux points de ce demi cercle, les points à l'infini sur le lieu de Nyquist et son symétrique, recherchons un troisième point : en prenant le transformé du point A ($\varphi = 0$), on obtient le point A_t, **la fermeture se fait par la droite**.
– si au contraire le contournement de l'origine se fait par la gauche, le transformé du petit **demi cercle** est encore un **demi cercle** et en recherchant le transformé de B ($\varphi = \pi$) on obtient le point B_t et **la fermeture se fait par la gauche**.

Il nous reste à compter le nombre de tours (N) que fait le point transformé autour de -1 et à appliquer $Z = P - N$.

Terminons l'exemple proposé, on voit que le lieu de Nyquist dépend du paramètre k et qu'en changeant ce paramètre on effectue une homothétie du lieu.

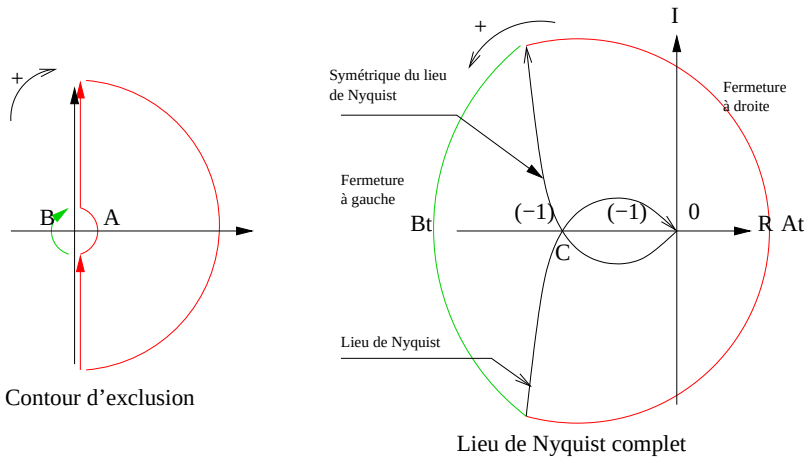


FIG. 5.4: Stabilité, exemple

- En prenant le premier contour d'exclusion, si la longueur OC est inférieure à 1 alors le point transformé ne tourne pas autour de -1 donc $N = 0$, comme $P = 0$ on a donc $Z = 0 - 0 = 0$; **la boucle fermée est stable**.
- Si l'on prend le deuxième contour d'exclusion on a ici $P = 1$ et l'on voit que $N = 1$, (en respectant les deux sens de rotation) donc $Z = 1 - 1 = 0$ on retrouve le même résultat.

On montrerait facilement que si la longueur OC est supérieure à 1 alors avec le premier contour on a : $Z = 0 - (-2) = 2$: **la boucle fermée est instable**.
Avec le deuxième contour on a $Z = 1 - (-1) = 2$ en effet ici $P = 1$.

Le critère du revers

Le critère du revers, utilisé dans 90% des cas est une version simplifiée du critère de Nyquist-Cauchy et ne s'applique **qu'aux systèmes bouclés dont la boucle ouverte est stable et à déphasage minimum** (dans certains ouvrages on dit que le système bouclé est régulier), en prenant comme contour d'exclusion un contour n'englobant pas les pôles éventuels de l'axe imaginaire du plan complexe.

Son énoncé est le suivant :
Pour un **système stable en boucle ouverte**, il sera **stable en boucle fermée**, si en parcourant le lieu de Nyquist dans le sens des fréquences croissantes **on laisse le point -1 à gauche**. Quand on raisonne avec le lieu de Black de la boucle ouverte, on remplace le mot gauche par droite.

5.2 Les marges de stabilité d'un système bouclé

Nous avons déja introduit à la section 3.3.2 la notion de **marge de phase** et de **marge de gain**. Ces deux notions sont issues de la stabilité des systèmes bouclés en utilisant le critère de stabilité de Nyquist-Cauchy, je reviendrai sur ces notions dans l'étude de la synthèse des systèmes bouclés par la méthode fréquentielle. En fait chiffrer la marge de stabilité d'un système bouclé revient à chiffrer **une distance qui sépare un lieu fréquentiel de la boucle ouverte (Nyquist ou Black) du point** -1.

5.2.1 Marge de stabilité absolue

On peut facilement avec Scilab introduire la notion de **marge de stabilité absolue** en utilisant l'instruction `horner()`.

Si $G(s)$ est la transmittance isomorphe de la chaîne d'action d'un système bouclé dont la chaîne de retour a pour transmittance $H(s)$, alors la transmittance du système bouclé vaut :

$$W(s) = \frac{G(s)}{1 + G(s)H(s)} = \frac{N(s)}{D(s)}$$

.

Si nous appliquons le critère de Routh-Hurwitz non pas au polynôme $D(s)$ mais au polynôme $D_\alpha(s) = D(s - \alpha)$, avec $\alpha > 0$, alors la condition : les racines de $D_\alpha(s) = 0$ dans le demi plan gauche, impliquera que les racines de $D(s) = 0$ sont situées à gauche de la droite d'abscisse $-\alpha$. Si l'on peut trouver une valeur de $\alpha > 0$ satisfaisant cette condition, alors cette valeur maximale pour α sera appelée **marge absolue de stabilité** et est notée m_a.

L'application du critère de Routh-Hurwitz au polynôme $D_\alpha(s)$ implique que toute racine de cette équation est à gauche de la verticale $-\alpha$ dans le plan complexe, et alors toute conséquence de perturbation appliquée à l'instant t_0 aura une décroissance aussi rapide qu'une exponentielle de la forme $\exp(-\alpha(t - t_0))$. Voici un exemple de programme Scilab.

On donne un système bouclé à retour unitaire de transmittance de chaîne d'action

$$G(s) = \frac{1}{s(1 + s)(1 + \frac{s}{3})}$$

En mettant en cascade avec $G(s)$ un amplificateur de gain k positif on obtient un système bouclé stable si $k \leq 4$. En prenant $k = 2$, on obtient pour marge de stabilité absolue $m_a = 0,1855394$, qui correspond aux pôles les plus à droite du plan complexe, voici le programme :

```
-->s=%s ;g=syslin('c',1/(s*(1+s)*(1+s/3))) ;
-->[kl,pl]=kpure1(g)
 pl =
!   1.7320508i !
! - 1.7320508i !
```

```
 kl =
  4.
-->//ici on prend k=kl/2
-->k=kl/2
-->//la boucle fermée
-->w=k*g/(1+k*g);
-->rw=roots(w('den'))
 rw =
! - 0.1855394 + 1.2723832i !
! - 0.1855394 - 1.2723832i !
! - 3.6289211              !
-->rmax=abs(max(real(rw)))
 rmax =
  0.1855394 //marge de stabilité absolue
```

Ce système bouclé de transmittance

$$W(s) = \frac{2G(s)}{1 + 2G(s)} = \frac{6}{6 + 3s + 4s^2 + s^3}$$

a deux pôles dominants complexes conjugués $p_{1,2} = -0,1855394 \pm 1,2723832j$, la valeur absolue du maximum de la partie réelle des pôles est bien m_a.

On peut maintenant envisager le **problème inverse**, à savoir par exemple trouver le gain k pour que le ou les pôles les plus à droite soient situés sur une verticale donnée. Reprenons le même exemple et on cherche k pour que les pôles dominants soient situés sur la verticale passant par $-0,2$ ($m_a = 0,2$).

```
-->ga=horner(g,s-.2)//changement de variable
-->ga=syslin('c',ga)//indispensable pour redéfinir la nouvelle

-->//boucle ouverte
-->[k1,p1]=kpure1(ga)
 p1 =

!   1.2328828i !
! - 1.2328828i !
 k1 =
  1.872//c'est la valeur du gain recherché

-->w1=k1*g/(1+k1*g)
 w1 =
        5.616
  -------------------
                2   3
  5.616 + 3s + 4s + s
-->rac=roots(w1('den'))//vérification
```

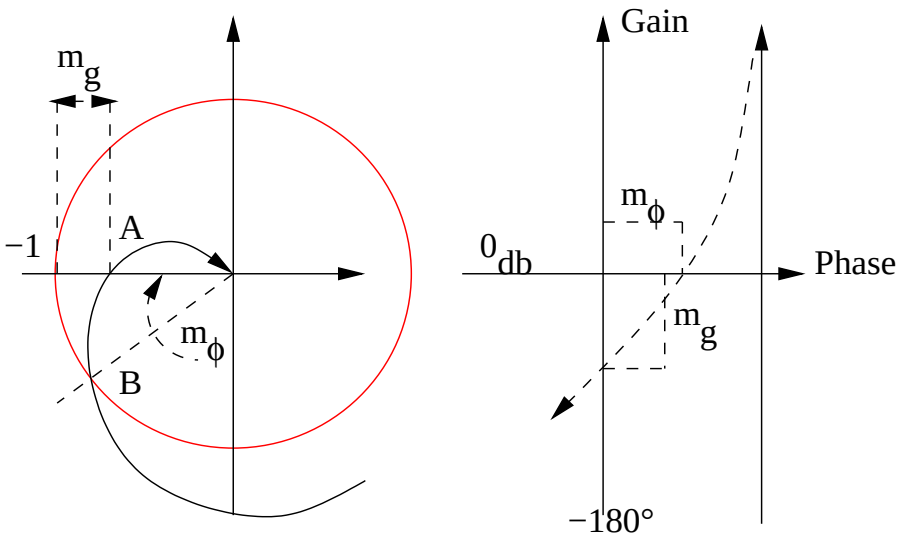


FIG. 5.5: Marge de phase, de gain

```
 rac =
! - 0.2 + 1.2328828i !
! - 0.2 - 1.2328828i !
! - 3.6               !
```

Dans cet exemple nous avons trouvé le gain de la chaîne d'action $k = 1,872$, donnant une marge de stabilité absolue $m_a = 0,2$.

5.2.2 Marge de phase, marge de gain

Comme nous l'avons dit en introduction, le chiffrage de la distance qui sépare un lieu fréquentiel du point -1, peut être fait dans le plan de Nyquist (ou Black), en calculant sur l'axe horizontal la distance qui sépare le point A au point -1. Cette distance, (homothétie pour le lieu de Nyquist, translation vers le haut pour le lieu de Black), exprimée en **décibels** est **la marge de gain**.

Quant à la **marge de phase** c'est la phase supplémentaire (retard de phase) qu'il faut rajouter (en négatif) à un lieu fréquentiel de la boucle ouverte pour faire passer ce lieu par -1 : rotation dans le sens des aiguilles d'une montre à faire au point B pour le faire passer par -1 (FIG. 5.5). Je ne fais pas d'exercice avec le logiciel Scilab pour illustrer ces deux notions, on verra cela lors des exercices de synthèse en utilisant la méthode de Black.

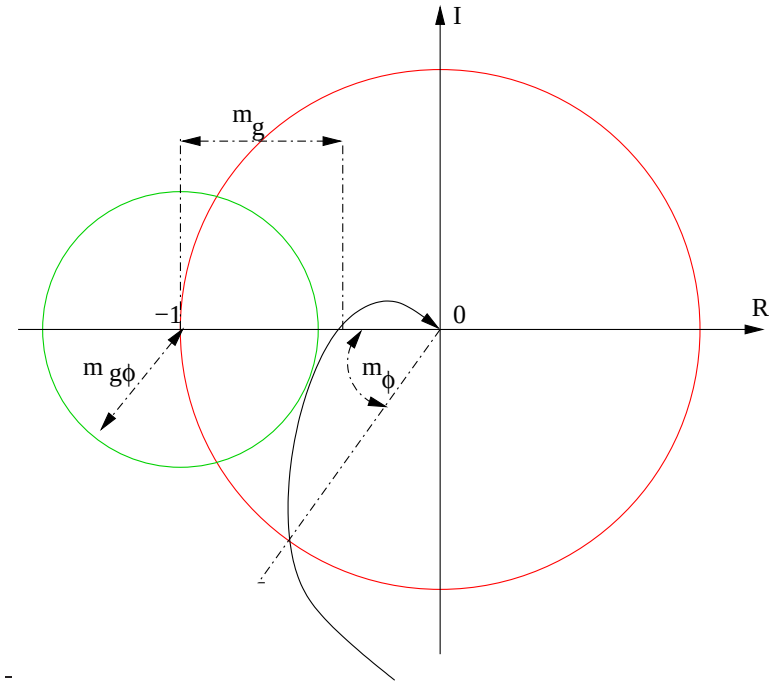


FIG. 5.6: Marge de gain-phase

5.2.3 Marge de gain-phase

L'inconvénient des marges de phase et de gain réside dans le fait que pour certains systèmes réels et en particulier quand on a négligé dans le modèle un petit retard pur, on obtient pour la boucle ouverte un lieu de Black moins horizontal qu'il ne l'est en réalité (plus la fréquence augmente et plus le déphasage devient important), dans ces conditions, imposer une bonne marge de phase ne veut pas dire que la marge de gain (qui est calculée pour une fréquence supérieure à la fréquence de coupure à $0\,db$) sera bonne même si on le croît. Il est donc intéressant de regrouper les deux marges précédentes sous la forme d'un seul nombre qui mesure la distance minimale entre le lieu de Nyquist de la boucle ouverte et le point -1. Ce nombre est la marge de gain-phase notée $m_{g\phi}$ (FIG. 5.6) ; c'est le rayon du plus grand cercle de centre -1 tangent au lieu de Nyquist de la boucle ouverte supposé situé à sa droite (système bouclé stable).

Calcul avec Scilab de la marge de gain-phase

Pour un système stable donné ceci est très simple en utilisant l'instruction `repfreq()` ou `rrpfreq()` : on calcule en fonction de la fréquence la distance du point courant du lieu de Nyquist avec le point -1 et l'on cherche la valeur minimale de cette distance, on obtient ainsi la fréquence correspondante et cette distance minimale. Cette fréquence et cette distance minimale n'ont pas de rapport avec la fréquence de résonance en

boucle fermée et avec le coefficient de surtension.

5.2.4 Marge de retard

Comme nous venons de le dire précédemment, le retard pur (quelquefois négligé), produit un déphasage qui augmente avec la fréquence et est source d'instabilité. En ramenant la marge de phase à la pulsation à laquelle elle a lieu, on introduit la marge de retard notée

$$m_r = \frac{m_\phi}{\omega_1} = \frac{\pi + \varphi_1}{\omega_1}$$

ω_1 et φ_1 sont respectivement la pulsation de coupure à $0\,db$ et le déphasage de la boucle ouverte correspondant. Ainsi plus cette pulsation de coupure à $0\,db$ sera importante moins le système sera tolérant vis à vis des retards purs. Il faut donc que la bande passante ne soit pas trop importante (à vous de trouver le bon compromis).

5.2.5 Cercles à gain constant, cercles à phase constante dans le plan de Nyquist : M et N cercles, abaque de Hall

Nous avons vu à la section 4.2.4 l'abaque de Black qui donne le gain (en *db*, la phase en degrés), d'un système bouclé à retour unitaire, connaissant le gain et la phase de la chaîne d'action. Et bien, l'abaque de Hall, est la présentation dans le plan de Nyquist de la transformation homographique $W(s) = \frac{G(s)}{1+G(s)}$, avec $G(s)$ la transmittance de la chaîne d'action et $W(s)$ la transmittance du système complet : c'est la transcription dans le plan de Nyquist de l'abaque de Black. Scilab donne par l'instruction `m_circle()` ou `m_circle(vecteurgain)` les courbes isogains mais ne donne pas, contrairement à l'instruction `chart()`, les courbes isophases. On verra, en faisant `apropos m-circle` dans Scilab, l'exercice proposé. Pour les curieux, vous verrez dans le livre [3, pages 278-279], la théorie sur l'abaque de Hall et la transformation homographique $W(s) = \frac{G(s)}{1+G(s)}$.

5.2.6 Nouveauté : utilisation des pseudo-lieux, pôles dominants sur une droite à amortissement constant[2]

Le problème que l'on peut se poser est le suivant : étant donné un système bouclé stable on cherche, par analogie avec un système du second ordre (dont l'amortissement est $\xi < 1$, pôles complexes conjuguées), à avoir deux pôles dominants sur des droites (droites OA, OB) passant par l'origine et faisant un angle ψ donné par rapport à l'axe vertical du plan complexe : **on fait du placement des pôles dominants**.

Ne voulant pas raisonner en temporel, on va rechercher les transformés de ces droites (en Nyquist ou Black) par la transformée $G(s)H(s)$: on aura donc affaire à des pseudo-lieux (voir section 5.1.4.1).

[2] A ma connaissance cette méthode n'a jamais été publiée et mise en oeuvre ; on trouverait peut être une méthode semblable dans les très vieux ouvrages d'automatique : si lecteur vous le savez, informez moi.

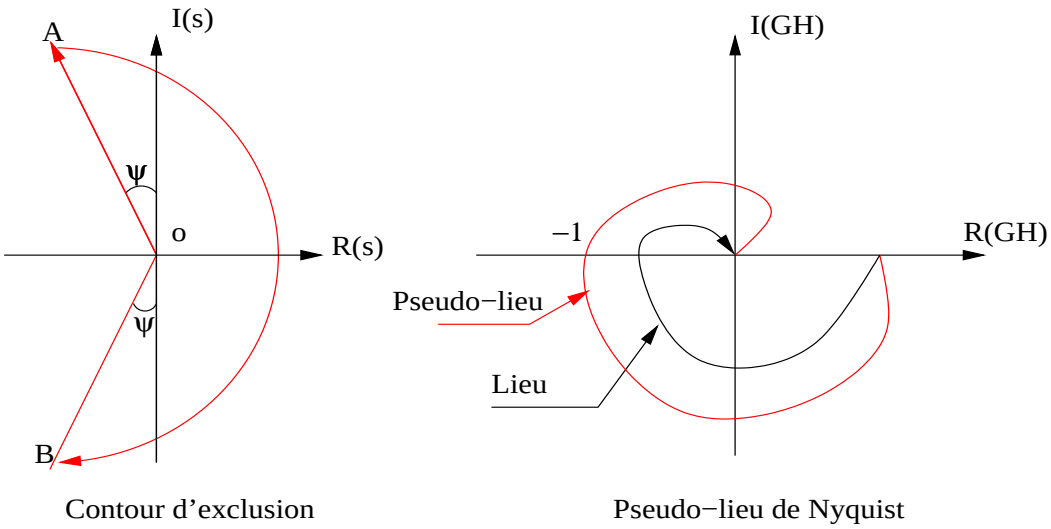


FIG. 5.7: Nouveau contour et pseudo-lieu

Il est évident qu'en règlant les paramètres de la boucle ouverte (gain, paramètres de réseaux correcteurs éventuels), de telle sorte que ce pseudo-lieu passe par le point -1, on placera les deux pôles dominants de la boucle fermée sur ces deux droites (utilisation du lemme de Cauchy), ce qui change dans cette méthode c'est le contour d'exclusion[3] (FIG. 5.7).

Mise en oeuvre de cette méthode Nous allons reprendre l'exercice vu à la section 5.2.1, de même je rappelle que pour un système du second ordre d'amortissement ξ, on a $\sin\psi = \xi$ car $\sin\psi = \frac{\xi\omega_n}{\omega_n} = \cos\theta$ (voir la figure 4.8). Voici la fonction (`pseudo.sci`) permettant de calculer le gain et la phase quand le point M décrit la droite OA (l'angle ψ vaut $\pi/6$).

```
function[db1,ph1]=pseudo(fr,tet,k)
w=2*%pi*fr*exp(%i*tet)
db1=(20/log(10))*(log(abs(k))-log(abs(w))-log(abs(1+w))-log(abs(1+w/3)))
//pour la phase mettre sous forme éléments simples
ph1=-(180/%pi)*(atan(imag(w),real(w))+atan(imag(1+w),real(1+w))+...
atan(imag(1+w/3),real(1+w/3)))
endfunction
```

Maintenant écrivons le programme principal :

```
-->;getf("/home/lpovy/SCI/pseudo.sci");
-->s=%s; g=syslin('c',1/(s*(1+s)*(1+s/3)));
```

[3] Je ne démontre pas, mais cela est facile, que si on change j en $-j$, le nouveau pseudo-lieu est symétrique par rapport à l'axe réel du pseudo-lieu.

```
-->bblack(g,.01,1)
```

Ici on a tracé le lieu de Black de $g(s)$.

```
-->ff=calfrq(g,0.03,.35);
-->[d,p]=pseudo(ff,%pi*4/6,1)
-->bblack(ff,d,p)
```

Un commentaire est ici nécessaire : par `calfrq()` on discrétise la fréquence au mieux, la transmittance $g(s)$ nous servant de référence, puis on calcule le gain et la phase pour un angle par rapport à l'axe réel positif, de $\frac{4\pi}{6} = \frac{\pi}{2} + \frac{\pi}{6}$ et pour $k = 1$, puis on trace le pseudo-lieu de Black pour voir le résultat.

```
-->//on calcule le gain et la phase classique
-->[d1,p1]=pseudo(ff,%pi*3/6,1);
-->bblack(ff,[d1;d],[p1;p])//vrai et pseudo-lieu
```

Ici on calcule le gain et la phase pour tracer le vrai et pseudo-lieu de Black ($k = 1$).

```
-->[bout,posx,posy]=xclick()
-->k=10^(-posy/20)//c'est le bon gain
-->gk=k*g;
```

Autre commentaire, par `xclick()` (après un bon zoom) je recherche sur l'axe -180° le point du peudo-lieu de Black (manière de connaître la marge de gain, car ici on ne peut pas utiliser `g_margin()`), puis je calcule le gain qu'il faut donner au système pour faire passer le pseudo-lieu par -1.

```
-->W=gk/(1+gk)//la boucle fermée
-->rbf=roots(W('den'))//les pôles de W
-->amort=abs(real(rbf(1)))/abs(rbf(1))
-->sinpsi=sin(%pi/6)
```

On compare l'amortissement (du aux deux pôles les plus à droite, dominants), et le $\sin\psi$.

```
-->[d2,p2]=pseudo(ff,%pi*4/6,k);
-->bblack(ff,[d2;d1;d],[p2;p1;p])
```

On trace le pseudo-lieu de Black du système corrigé avec cette valeur de k, le vrai lieu de Black pour $k = 1$, et le pseudo-lieu pour $k = 1$ (FIG. 5.8). La valeur de k trouvée assure un amortissement tel que $\xi = \sin\frac{\pi}{6}$. Je reviendrais sur cette méthode en la comparant avec la méthode de synthèse dite méthode d'Evans.

Remarque : Nous avons tracé ici les pseudo-lieux de Black du système de transmittance $g_k(s) = \frac{k}{s(1+s)(1+\frac{s}{3})}$, et ceci pour une valeur de $s = \frac{4}{3}\frac{\pi}{2}$. Ceci veut dire que ces pseudo-lieux, sont lieux de Nyquist, Bode et Black, du système de transmittance $g_{\alpha,k} = \frac{k}{s^{\alpha}(1+s^{\alpha})(1+\frac{s^{\alpha}}{3})}$ avec $\alpha = \frac{4}{3}$: l'échelle des graduations n'est pas la même, sur les lieux de Nyquist et de Black les lieux ont la même forme, mais pas en Bode. De même si $g_k(s) = \frac{k}{(1+s)}$ on a affaire à un système du premier ordre et tracer les pseudo-lieux de ce système revient à étudier fréquentiellement le modèle dit Coole et Coole ou explicite $g_{\alpha,k}(s) = \frac{k}{(1+s^{\alpha})}$, (voir l'étude de ce modèle au chapitre 10) ; on peut se

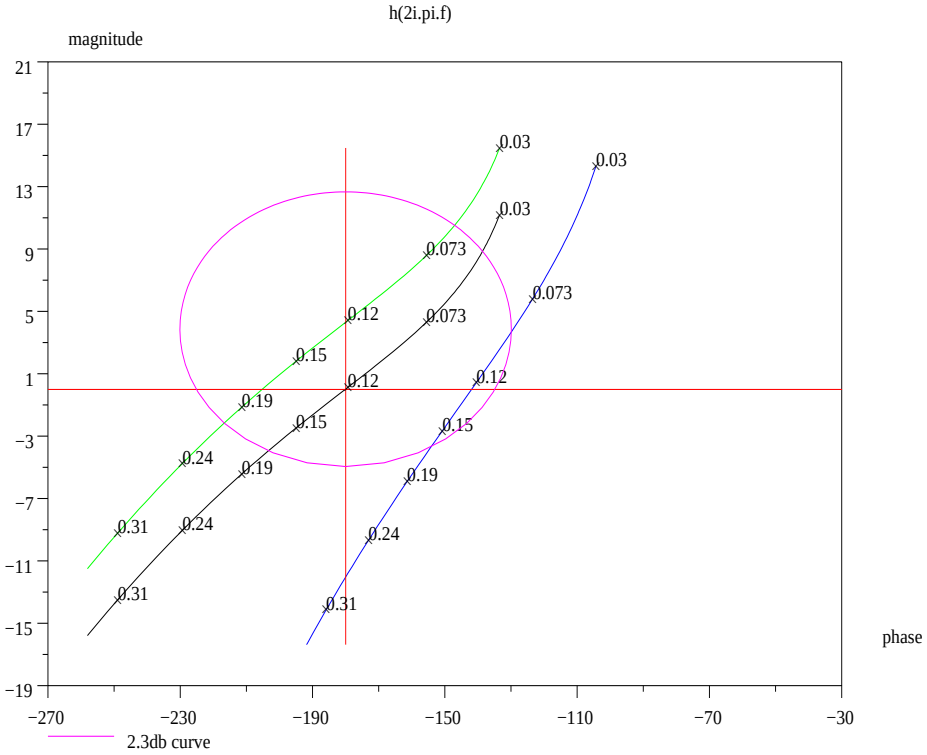


FIG. 5.8: Synthèse par les pseudo-lieux de Black

poser alors la question : quelle liaison y-a-t'il entre la réponse fréquentielle (obtenue avec les pseudo-lieux) et la réponse temporelle du modèle non entier de type Coole et Coole ?

6 Principe de la commande

6.1 Introduction

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 (la boucle ouverte).

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 (FIG. 6.1).

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 (ce qui revient à faire l'étude de l'erreur en régime permanent) et la précision dynamique, caractérisant cette erreur au cours du temps.

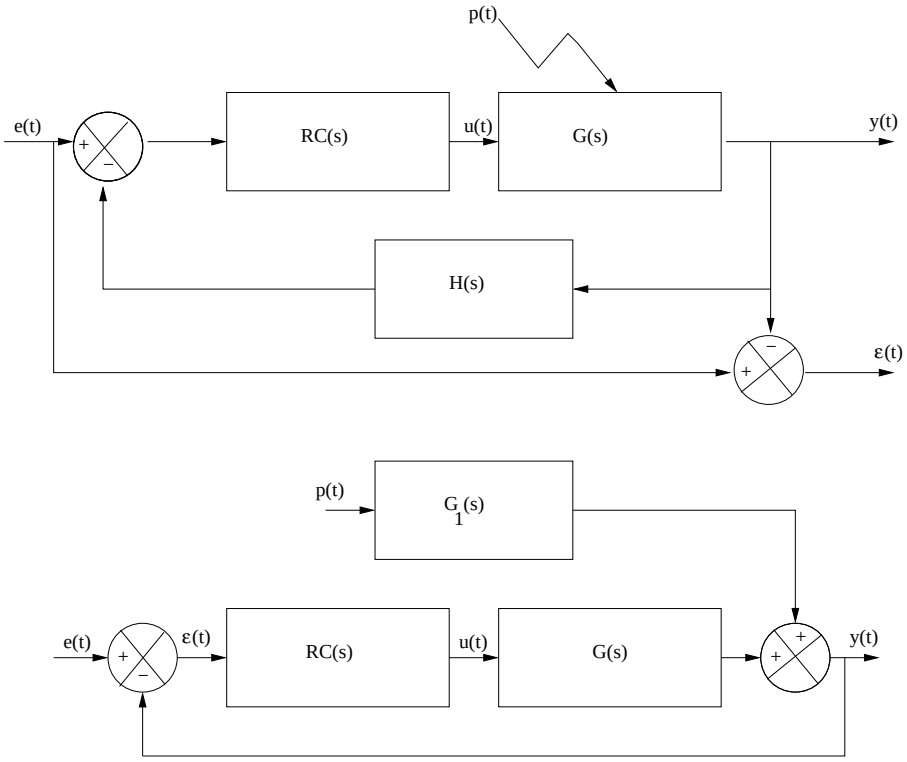


FIG. 6.1: Schéma d'un système bouclé

6.2 Précision

6.2.1 Précision statique

D'après le schéma précédent on déduit, d'après le principe de superposition, l'erreur de l'asservissement par la relation :

$$\varepsilon(s) = e(s)\frac{1}{1+RC(s)G(s)} - p(s)\frac{G_1(s)}{1+RC(s)G(s)}$$

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

- 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, K_1 le gain en vitesse, et K_2 le gain en accélération (gain apparaissant dans la factorisation de Bode).

nombre de pôles en 0	0	1	2	> 2
erreur en position	$\frac{1}{1+K}$	0	0	0
erreur en vitesse	∞	$\frac{1}{K_1}$	0	0
erreur en accélération	∞	∞	$\frac{1}{K_2}$	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 $\varepsilon(s)$, 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.

6.2.2 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 $RC(s)G(s)$ ait pour modèle une transmittance $\frac{k}{s}$ ou $\frac{k}{1+\tau s}$ la boucle fermée sera toujours du premier ordre (seule l'erreur permanente à un échelon différera : elle sera nulle dans le premier cas et vaudra $\frac{1}{1+k}$ 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 $t_5 = 3\tau_1$ (τ_1 constante de temps de la boucle fermée) : ce temps est aussi appelé **temps de réponse** (à 5 %).

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 $3\,db$ ou $6\,db$, la cassure du lieu de Bode se faisant en un point de pulsation $\omega = \frac{1}{\tau_1}$). Vous remarquerez ceci en changeant dans le modèle s en $\tau_1 s$, et dans ce cas vous changez t en $\frac{t}{\tau_1}$ et ω la pulsation en $\tau_1\omega$. En se fixant le temps de réponse à 5 % on se donne $\tau_1 = \frac{t_5}{3}$, donc la pulsation de coupure à $3\,db$ est donnée par $\frac{1}{\tau_1} = \frac{3}{t_5}$.

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) :

$$RC(s)G(s) = \frac{k}{s(1+\tau s)}$$

La boucle fermée vaut donc :

$$W(s) = \frac{k}{k + s + \tau s^2}$$

et est un système du second ordre de pulsation naturelle $\omega_n = \sqrt{\frac{k}{\tau}}$ et de coefficient d'amortissement $\xi = \frac{1}{2\sqrt{k\tau}}$ (voir section 4.3.2).

Si le coefficient $\xi > 1$ les pôles de $W(s)$ 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 D_1 % 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 respectivement :

$$D_1\% = 100\exp(-\pi\frac{\xi}{\sqrt{1-\xi^2}}) \; et \; t_{pic} = \frac{\pi}{\omega_n\sqrt{1-\xi^2}}$$

- A ξ fixé, les oscillations sont d'autant plus rapides que la pulsation naturelle est grande et à 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 $0,7$, le premier 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 :

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

Pour un facteur de résonance $Q = 1,3$ soit $Q_{db} = 2.3\,db$, on a une valeur de ξ de $0,42$, valeur donnant un dépassement $D_1\,\% = 30\,\%$. Cette valeur pour Q est souvent prise comme référence.

De même, comme la pulsation de résonance vaut $\omega_r = \omega_n\sqrt{1-2\xi^2}$, 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. Vous verrez dans les tableaux concernant les systèmes du second ordre (section 4.3.2) les différentes relations. Mais retenez que le temps de réponse à $5\,\%$ vaut $t_5 \approx \frac{3}{\xi\omega_n}$ (si on impose un temps de réponse à $5\,\%$ on se fixe le produit $\xi\omega_n$, et maintenant en s'imposant un amortissement, on determine tous les paramètres du second ordre). Une dernière remarque, ce produit $\xi\omega_n$est au signe près la partie réelle des deux pôles complexes conjugués du système du second ordre.

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.

6.3 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 règler 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-dessus présente cette région (FIG. 6.2). 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.

6.4 Méthodes de synthèse : pourquoi utiliser la méthode de Black

Nous avons dans les sections précédentes, analysé le comportement temporel et fréquentiel d'un système et introduit les différentes représentations, ainsi que l'abaque de Black. Ce qu'il faut retenir de ces sections peut se résumer par les points suivants :

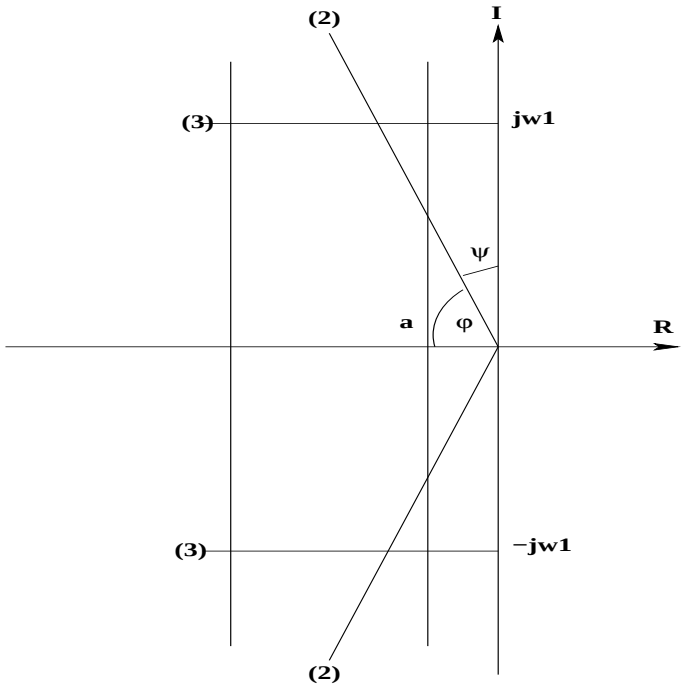


FIG. 6.2: Placement des pôles

1. On veut que le système bouclé soit précis, il faut donc se donner cette précision et de là on déduit, s'il faut ou non rajouter un ou plusieurs intégrateurs dans la chaîne d'action et/ou augmenter le gain statique (en position, en vitesse...) de cette même chaîne d'action.
2. On veut un système bouclé stable (obligation) avec des marges de stabilité.
3. On veut une dynamique déterminée, donc il faut placer les pôles dominants du système bouclé dans un certaine région du plan complexe.
4. On ne veut pas saturer les actionneurs et de plus ne pas consommer trop (?) d'énergie.

Nous avons vu que ceci pouvait être fait, d'une manière peut être un peu approximative, en prenant pour référence un système du premier et/ou second ordre dont on connaît parfaitement le comportement temporel et fréquentiel et que pour ces systèmes il y avait une liaison directe entre le comportement temporel et fréquentiel : du comportement fréquentiel on en déduisait le comportement temporel. On va donc plutôt travailler en fréquentiel car avec cet outil, on déduira les propriétés de la boucle fermée à partir des lieux de transfert de la boucle ouverte : la méthode de Black permet de faire cela d'une manière très élégante. On réalisera donc les opérations suivantes :

1. Ramener votre système bouclé, à un système à retour unitaire.
2. Tracer 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 (FIG. 6.3).

```
-->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
-->bblack(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))`.

6.4.1 Principe de mise en oeuvre

On choisira une structure de régulateur permettant de vérifier ce cahier des charges. Il conviendra 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, avoir un facteur de résonance en boucle fermée donné.

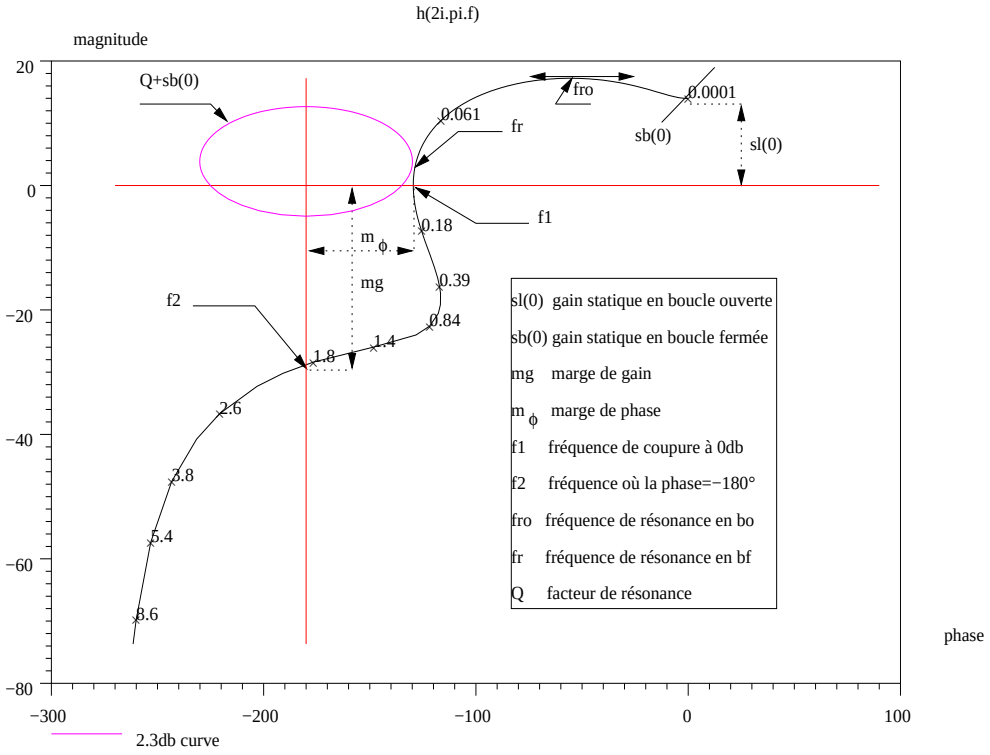


FIG. 6.3: Lieu de Black

- d'augmenter le gain de la boucle ouverte, ou mettre le point à fréquence zéro à l'infini (introduction d'un ou plusieurs intégrateurs dans la chaîne d'action).
- d'augmenter la bande passante donc provoquer un tassement en fréquence vers les gains élevés : on diminue le temps de réponse de la boucle. Rappelez vous que pour un système du second ordre, la pulsation de coupure à $0\,db$, ω_{co} est liée au temps de réponse à $5\,\%$ par la relation : $\omega_{co} t_{r\,5\%} \approx \pi$.
- Ceci peut être résumé par le choix d'un réseau correcteur apportant si nécessaire, une avance de phase aux fréquences moyennes et un retard de phase aux basses fréquences.

7 La correction des systèmes par la méthode de Black

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 ...) 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. Mais avant cela, nous allons voir à quel endroit de la boucle il est peut être intéressant de placer le ou les réseaux correcteurs.

7.1 Choix de la structure de réseau correcteur

Ce choix dépend beaucoup des informations qui sont disponibles, en qualité, sur le procédé. Le cas le plus courant est de disposer d'un capteur récupérant une image (plus ou moins bonne) de la sortie. C'est donc à partir de cette seule information, éventuellement traitée et mise sous un format normalisé, que l'on réalisera le bouclage.

Dans quelques cas particuliers on dispose d'informations supplémentaires pouvant caractériser le procédé : un asservissement de position par exemple, où l'on capte la position et la vitesse à l'aide d'un dispositif adéquat, ou d'un asservissement par moteur électrique où l'on peut ce permettre de mesurer le courant d'induit de moteur en plus de sa vitesse : quelques variables d'état du procédé sont accessibles.

7.1.1 Correcteur dans la chaîne d'action

C'est le cas le plus fréquent, on ne dispose que d'un capteur donnant une image de la sortie. Cette image combinée avec la consigne va nous permettre d'élaborer le signal erreur de la boucle (signal $\varepsilon(t)$). C'est ce signal (éventuellement amplifié et traité) qui sera l'entrée du réseau correcteur. Ce réseau élaborera une commande $u(t)$ que l'on amplifiera en tension et surtout en puissance et appliquera au procédé (amplificateur(s) et actionneurs).

7.1.2 correcteur dans la chaîne de retour

Un exemple classique de correcteur introduit dans la chaîne de retour est la correction tachymétrique d'un asservissement de position. Un exemple pour illustrer cette correction.

Soit un moteur à courant continu servant au positionnement (sortie $\theta(t)$) d'un objet, ce moteur est alimenté en courant, par un amplificateur de gain k, on dispose de plus en sortie d'une image de la position $\theta(t)$ par potentiomètre par exemple, de même

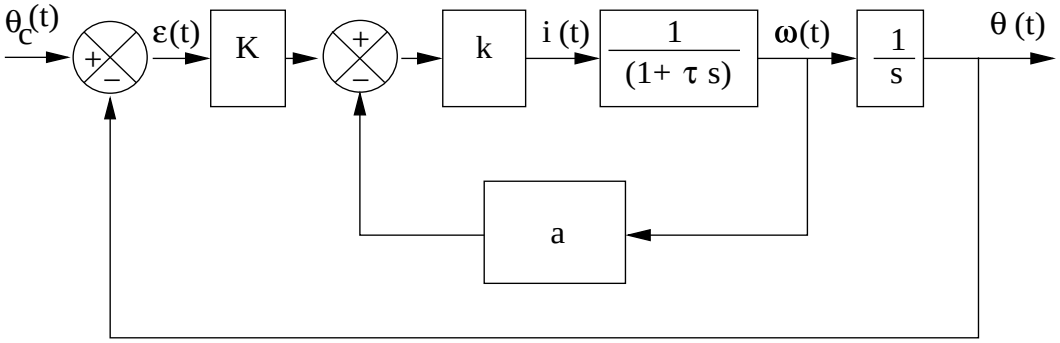


FIG. 7.1: Retour tachymétrique

en couplant une génératrice tachymétrique on a une image de la vitesse de rotation $\omega(t)$ de l'arbre moteur. On peut donc réaliser avec ces deux signaux le système bouclé suivant (FIG. 7.1) ; on rajoute un gain K dans la chaîne d'action afin d'avoir deux paramètres de réglage : K et a (taux de retour tachymétrique) : On admettra que la charge mécanique entrainée par le moteur est constituée exclusivement d'inertie J et de frottement visqueux f, alors $\tau = \frac{J}{f}$. D'après cette figure la transmittance du système bouclé vaut :

$$W(s) = \frac{1}{1 + \frac{1+ak}{Kk}s + \frac{\tau}{Kk}s^2}$$

Ce système bouclé est du second ordre avec une pulsation naturelle réglable par K et un amortissement, lui aussi réglable par K et a ; on peut faire du placement de pôles : voir placement de pôles par retour d'état[1].

7.1.3 Assemblage de correcteurs

Pour que l'on puisse honorer le cahier des charges, on devra choisir une structure de réseau correcteur capable de réaliser les actions suivantes :

- Action proportionnelle : translation verticale du lieu de Black.
- Action dérivée : translation plus ou moins horizontale du lieu de Black et ceci vers la droite.
- 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.

[1] On peut facilement ici, se fixer le temps de réponse à 5 % et par exemple la valeur du premier dépassement de la réponse indicielle, et en déduire les deux paramètres Kk et ak.

7.2 Action proportionnelle

Le modèle du réseau est $RC(s) = K$, 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,

$$sl = \frac{1}{s(1+s)(1+\frac{s}{3})}$$

et avec Scilab traçons : $sl(s)$, $0,5sl(s)$, $2sl(s)$ (FIG. 7.2).

```
-->s=%s ;den=s*(1+s)*(1+s/3) ;
-->sl=syslin('c',1/den) ;
-->[K,P]=kpure1(sl)
P =
!   1.7320508i !
! - 1.7320508i !
K =
  4.
//je me place à la limite de stabilité
-->sl=K*sl
sl =
             4
---------------------------
               2            3
s + 1.3333333s + 0.3333333s
-->sl_05=.5*sl ; sl_2=2*sl ;
-->bblack([sl_05 ;sl ;sl_2],.1,10)
-->legends(['k=0,5';'k=1';'k=2'],[1,2,3])
```

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

7.3 Action proportionnelle et dérivée

Le réseau correcteur théorique (une explication sera donnée plus loin), a pour modèle[2]

$$RC(s) = K(1 + \tau_d s)$$

et en prenant $K = 1$, ce qui ne change pas le raisonnement, on voit apparaître une avance de phase appréciable, (vous tracerez le lieu de Bode d'un réseau pour $\tau_d = 1$, par exemple) 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 (FIG. 7.3).

[2] Lors de l'étude du réseau P.I.D. on donnera l'explication en question.

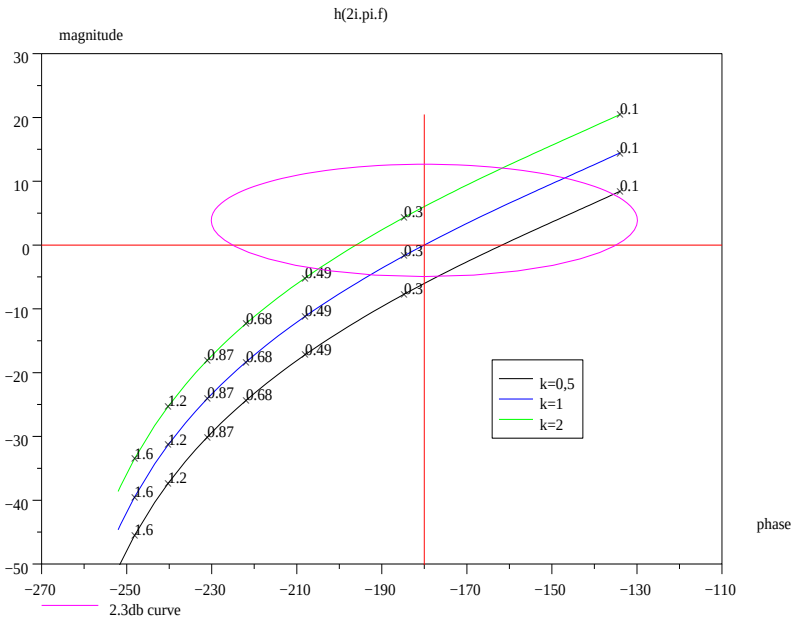


FIG. 7.2: Correction proportionnelle

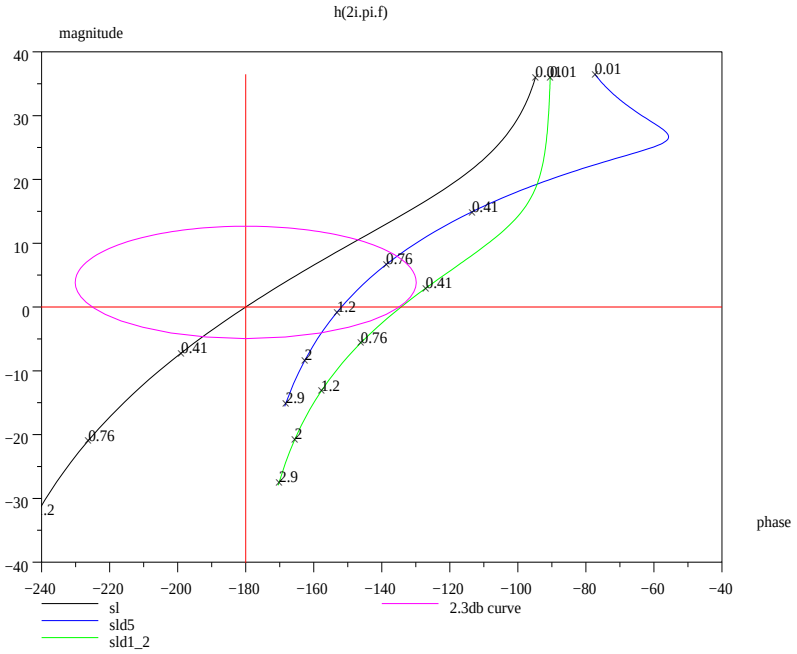


FIG. 7.3: Correction avec un réseau P.D.

```
-->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;
-->bblack([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, $sltaud(s) = (1 + \tau_d s)sl(s)$, afin de le comparer par rapport aux exemples précédents et suivants.

```
-->sld0_2=(1+.2*s)*sl;
-->bblack([sl;sld0_2;sld1_2],.01,3,['sl';'sld0_2';'sld1_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';'sltaud2'])
```

```
//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(sltaud2/(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 réseau proportionnel et dérivé, 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.

7.4 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

$$RC(s) = \frac{1+a\tau s}{1+\tau s} \; avec \; a > 1$$

Comme nous l'avons vu en cours, on constate que l'avance de phase maximale est indépendante de τ, et vaut : $\phi_{max} = \arcsin(\frac{a-1}{a+1})$. Cette avance de phase maximale a lieu pour une pulsation : $\omega_a = \frac{1}{\tau\sqrt{a}}$. 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 (je ne fais pas le corrigé de l'exercice, celui-ci faisant l'objet d'un TP).

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

$$G(s) = \frac{1}{s(1+s)(1+s/3)}$$

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_l et la pulsation d'oscillation ω_l pour cette valeur limite du gain.

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

`poly ; syslin ; routh_t ; kpure1 ; p_margin ; g_margin ; freson ; horner ou freq ; dbphifr ; csim ; plot2d ; linspace ; bbode ; bblack ; nnyquist.`

On place en cascade avec la chaîne d'action un réseau correcteur à avance de phase de transmittance

$$RC(s) = \frac{1+a\tau s}{1+\tau s} \; avec \; a > 1$$

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. On remarquera que pour une pulsation valant $\omega_a = \frac{1}{\tau\sqrt{a}}$ le réseau a un gain de $10\log(a)$[3]. 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'])`.

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.

7.5 Réseau correcteur à action proportionnelle et intégrale

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 6.2.1) mais

[3] En se plaçant à cette pulsation sur le lieu de Black de la boucle ouverte non corrigée, le point correspondant se déplace de ϕ_{max} vers la droite et $10\log(a)$ vers le haut, quand on rajoute le correcteur.

ce résultat est au détriment de la stabilité. En effet un intégrateur pur introduit un déphasage de -90° quelque soit la fréquence. Pour cette raison on préfère introduire un réseau de type proportionnel et intégral (P.I.).

Ce correcteur a pour transmittance :

$$RC(s) = 1 + \frac{1}{\tau_i s}$$

par une étude des courbes de Bode de ce réseau on peut remarquer qu'aux hautes fréquences ($\omega > \frac{1}{\tau_i}$) 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 $\frac{1}{\tau_i} \ll \omega_r$. 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, seule la précision statique est améliorée.

Voici un exemple de détermination d'un réseau P.I., la boucle ouverte a pour transmittance

$$G(s) = \frac{1}{s(1+s)(1+\frac{s}{4})}$$

```
-->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
```

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 et par instruction `dbphifr()`, nous obtenons la fréquence de résonance `frb` et le facteur de résonance : on peut ainsi déterminer une première valeur pour τ_i soit : `tor`.

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

On peut maintenant, pour plusieurs valeurs de τ_i construire différents réseaux correcteurs : par exemple,

$$RC(s) = 1 + \frac{1}{t_{or}s} \; puis \; RC(s) = 1 + \frac{1}{0,1t_{or}s} \; enfin \; RC(s) = 1 + \frac{1}{5t_{or}s}$$

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*rc01tor
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. Surtout **n'utiliser pas** la fonction `black()` donnée avec Scilab mais `bblack()`.

```
-->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 (FIG. 7.4) est suffisamment clair et met bien en évidence que pour $\tau_i = \frac{1}{\omega_r}$, 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 $\frac{1}{\omega_r}$. 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,

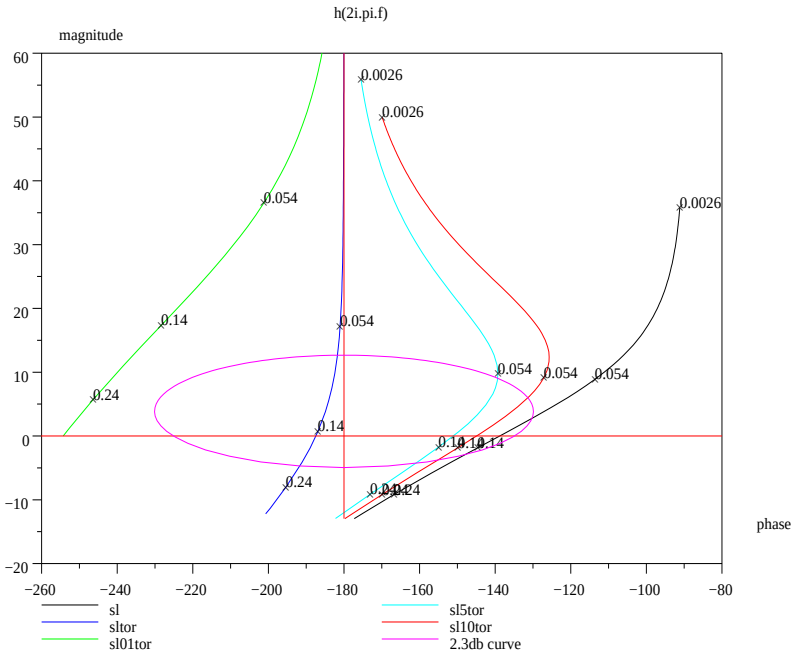


FIG. 7.4: Correction avec un réseau P.I.

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.

7.6 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 :

$$RC(s) = \frac{1+\tau s}{1+b\tau s} \; avec \; b > 1$$

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 $\frac{1}{\tau} \ll \omega_r$. 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 (FIG. 7.5).

```
-->s=%s ;sl=syslin('c',1/(s*(1+s)*(1+s/4))) ;
-->sb=sl/(1+sl) ;
-->[db,phi,frb]=dbphifr(sb,freson(sb))
 frb =
    0.1299495
phi =
- 76.236457
db =
  3.0914789
-->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)) ;
```

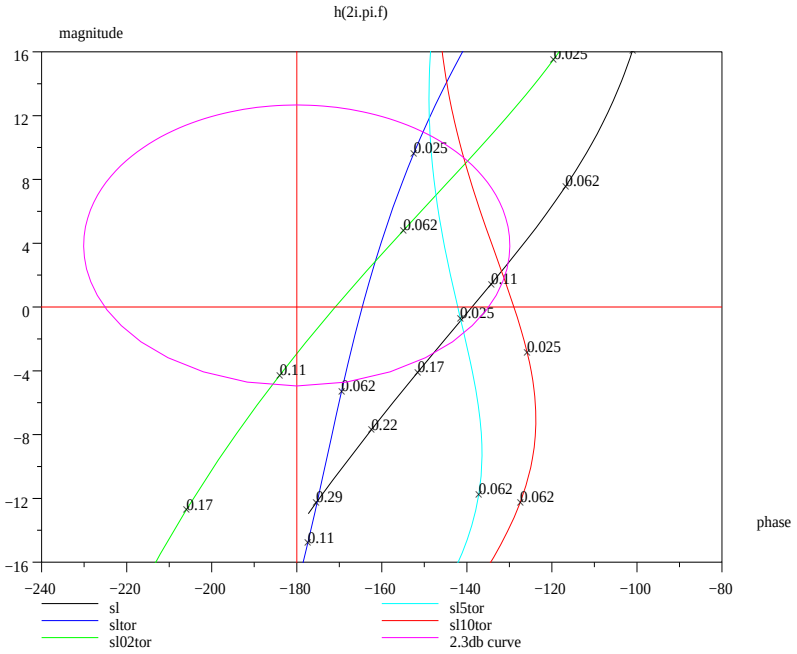


FIG. 7.5: Correction avec un retard de phase

```
-->sl10tor=sl*rc10tor;
-->com=['sl','sltor','sl02tor','sl5tor','sl10tor'];
-->bblack([sl;sltor;sl02tor;sl5tor;sl10tor],.001,.3,com)
```

Dans cet exemple nous voyons que l'adjonction d'un réseau à retard de phase avec $\tau = \frac{1}{\omega_r}$ 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 (FIG. 7.6).

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

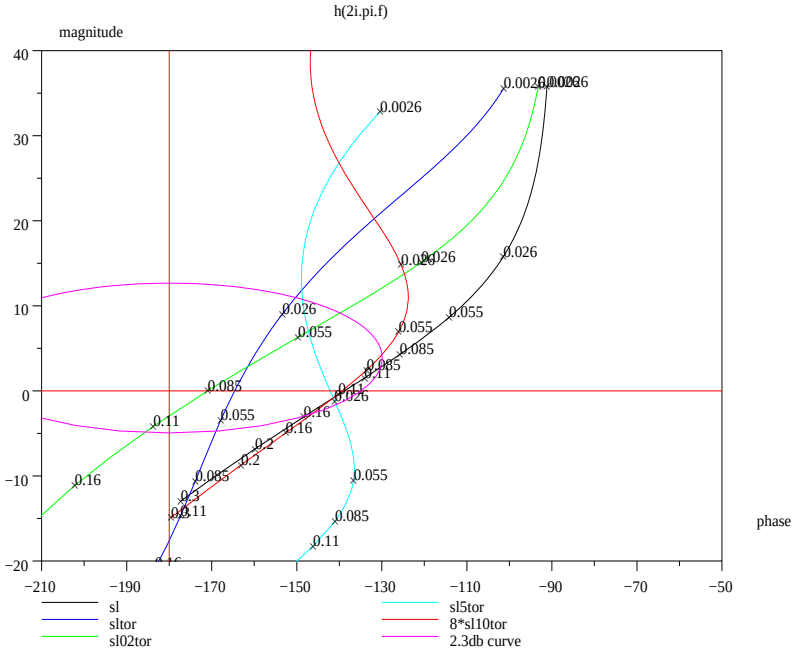


FIG. 7.6: Correction avec retard de phase et gain

7.7 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 :

$$RC(s) = K(1 + \frac{1}{\tau_i s} + \tau_d s)$$

En réalité, le terme dérivé, dans sa réalisation pratique étant impossible, a pour modèle le facteur suivant :

$$\frac{\tau_d s}{1 + a\tau_d s}$$

avec a prenant une valeur proche de un dixième (les deux points de cassure de ce terme séparés d'une décade au moins).

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, $\omega_p = \frac{1}{\sqrt{\tau_i \tau_d}}$ où le réseau n'avance ni ne retarde la phase, en ce point, le gain est de $0\,db$.

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_p, 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 $\frac{\omega_p}{2\pi}$ et est positive (comprise entre 0° et 90°) pour des fréquences supérieures à cette valeur. Quant au gain, il décroît jusqu'à $0\,db$ (pour $\omega = \omega_p$) puis recroît 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 Q.

Quant au choix du point de pivotement, si on souhaite un amortissement élevé, ordre de grandeur $0,7$, on devra prendre un point au dessus de l'axe $0\,db$, 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**. Si on exprime la transmittance du réseau en fonction de ω_p avec $\tau_i = 4\tau_d$ on obtient :

$$RC(p) = K(1 + \frac{\omega_p}{2s} + \frac{s}{2\omega_p})$$

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

```
-->s=%s ;sl=syslin('c',1/(s*(1+s)*(1+s/4))) ;
-->[kl,pl]=kpure1(sl)
pl =
!   2.i !
! - 2.i !
```

```
kl =
   5.
-->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 $\tau_i = \frac{1}{0,05s}$. Le réseau a donc pour modèle :

$$RC(s) = 1 + \frac{1}{6,7s} + 1,59s \, avec \, \tau_i = 4\tau_d$$

```
-->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 (FIG. 7.7). 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=mean(omegp)

//On prend la valeur moyenne
```

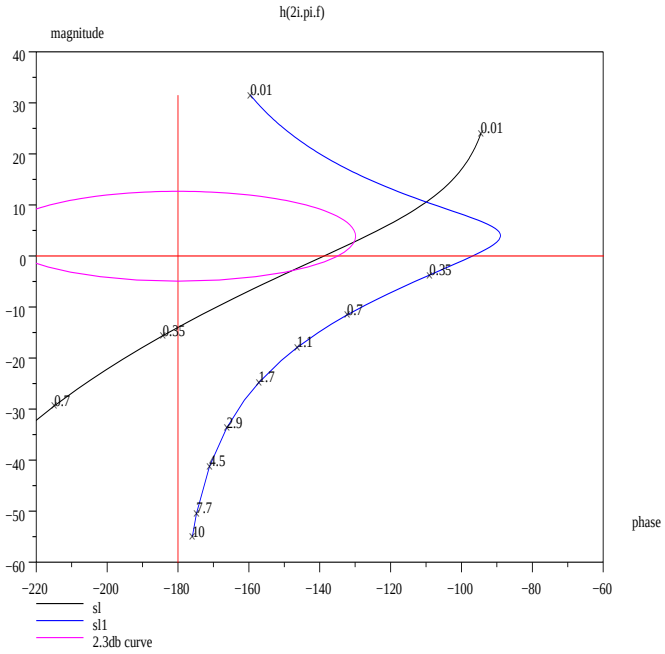


FIG. 7.7: Correction par la méthode du pivot

```
omegp =
0.2840349
-->rc=syslin('c',(1+1/((2/omegp)*s)+(1/(2*omegp))*s))

rc =

                            2
1 + 7.0413878s + 12.395285s
---------------------------
         7.0413878s
-->sl1=rc*sl
sl1 =
                            2
1 + 7.0413878s + 12.395285s
-----------------------------------
          2             3            4
7.0413878s + 8.8017347s + 1.7603469s
-->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 ($k_l = 5$ voir le début de l'exercice avec $\omega_l = 2\,rd/s$), par la même méthode, en choisissant pour fréquence de pivot $f_p = 0,08\,hertz$ 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 (FIG. 7.7), on a stabilisé le système (le facteur de résonance est maintenant très proche de $2,3\,db$, en réalité $1,93\,db$) 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 $2,3\,db$.

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 [4, pages 245-247], 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 $\omega_p = \frac{1}{\sqrt{\tau_i \tau_d}}$ est la pulsation du pivot on a : $\omega_p = \frac{1}{T_r}$, en effet on a $\tau_d = \frac{1}{2\omega_p}$ et $\tau_i = \frac{2}{\omega_p}$ par la méthode du pivot et l'on obtient ces mêmes valeurs par le critère de Ziegler et Nichols, en ayant $\omega_p = \frac{1}{T_r}$.

Peut-on faire un choix optimal de la pulsation du pivot ? On penserait normalement que le réglage du point de pivotement peut conduire à des résultats différents, non seulement sur la transmittance du réseau, mais surtout sur les caractéristiques dynamiques du système bouclé. Réalisons l'expérience suivante : en prenant la boucle ouverte précédente, avec un système à la limite de la stabilité, en choisissant différents points de pivotement pour des déphasages de -120°, -125°, -130°, -135°, on obtient les lieux de Black :

```
-->omegp=2*%pi*mean((f(find(p<-119&p>-121))))
omegp =
0.4398403
-->omegp1=2*%pi*mean((f(find(p<-124&p>-126))))
omegp1 =
0.5226819
-->omegp2=2*%pi*mean((f(find(p<-129&p>-131))))
omegp2 =
0.6070681
-->omegp3=2*%pi*mean((f(find(p<-134&p>-136))))
omegp3 =
0.7009965
-->rc=1+omegp/(2*s)+s/(2*omegp)
rc =
                            2
0.4398403 + 2s + 2.2735526s
---------------------------
           2s
-->rc1=1+omegp1/(2*s)+s/(2*omegp1)

rc1 =
                            2
0.5226819 + 2s + 1.9132097s
---------------------------
           2s
-->rc2=1+omegp2/(2*s)+s/(2*omegp2)
```

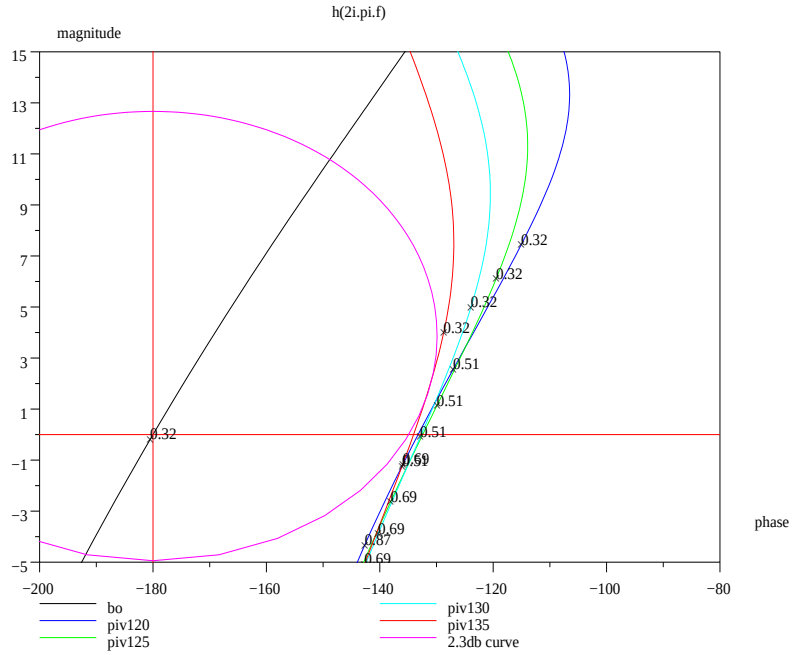


FIG. 7.8: Influence du choix du pivot

```
rc2 =
                              2
0.6070681 + 2s + 1.6472616s
---------------------------
            2s
-->rc3=1+omegp3/(2*s)+s/(2*omegp3)

rc3 =
                              2
0.7009965 + 2s + 1.4265407s
---------------------------
            2s
-->SL=sl*[1;rc;rc1;rc2;rc3];
-->com=['bo';'piv120';'piv125';'piv130';'piv135'];
-->bblack(SL,.1,1,com)
```

Nous voyons sur cet exemple que le point de pivotement n'a pas une influence prépondérante sur le résultat (FIG. 7.8). Comment le prouver ?, attention au fait qu'avec le P.I.D. on peut simplifier un pôle du système avec un zéro du correcteur.

7.8 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 (en électrique on le réalise facilement avec des quadripôles).

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 :

$$RC(p) = \frac{(1+\tau_2 s)(1+\tau_3 s)}{(1+\tau_1 s)(1+\tau_4 s)}$$

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 7.4).

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 $\frac{1}{\sqrt{\tau_2\tau_3}}$ 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) :

```
-->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 τ_2, τ_3 en $\frac{\tau_3}{2}$ etc..., on a donc (FIG. 7.9) :

```
-->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
-->bbode([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

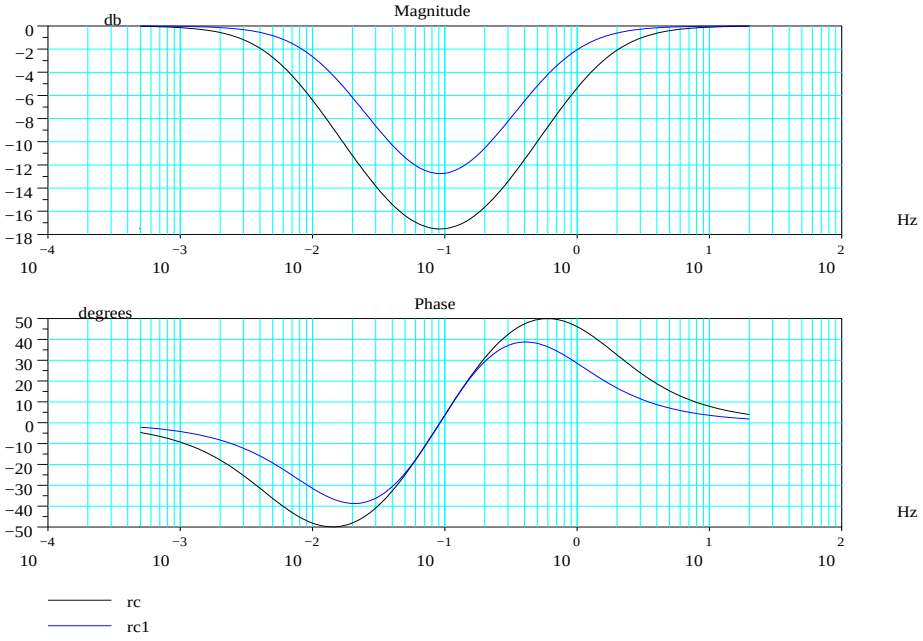


FIG. 7.9: Correcteur retard-avance

vaut pratiquement $\frac{1}{\sqrt{\tau_3\tau_4}}$, 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 $\frac{1}{\sqrt{\tau_1\tau_2}}$.

Appelons $f_1,\ f_2,\ f_3$ les fréquences où les déphasages introduits par le réseau sont respectivement : minimum, nul et maximum on a donc :

$$\sqrt{\tau_2\tau_3} = \frac{1}{2\pi f_2},\ \sqrt{\tau_3\tau_4} = \frac{1}{2\pi f_3},\ et\ \tau_1 = 10\tau_2,\ \tau_3 = 10\tau_4$$

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

Dans l'exemple choisi, si $f_2 = 0,1$ hertz, point correspondant à un déphasage en boucle ouverte de -120° environ (pseudo pivot), si l'on prend f_3 aux environs de 1 hertz (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 (FIG. 7.10).

```
-->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 X vérifient l'équation : $AX = C$.

```
-->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'])
```

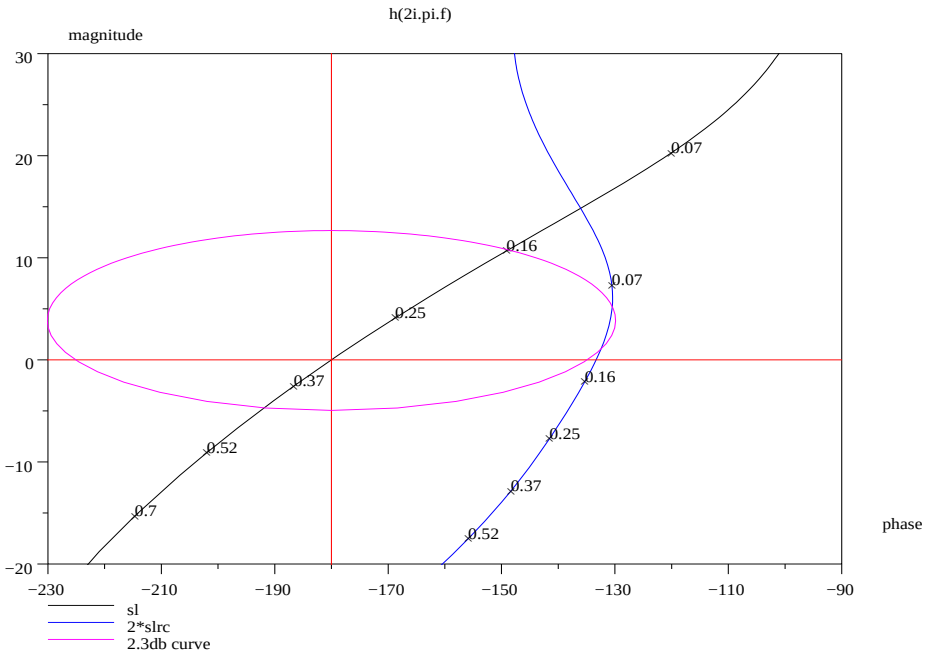


FIG. 7.10: Correction retard-avance de phase

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, f_2 et f_3 : on prend $f_2 = 0,09$ 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 $f_3 = 0,5$ 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 ; vous vérifierez ces constatations avec le logiciel.

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 (faire un exercice analogue à l'exercice de la section 7.4).

8 Synthèse par le lieu d'Evans

A rédiger avec la nouvelle contribution RLTOOL.

9 Représentation d'état des systèmes

A rédiger.

10 Utilisation du logiciel pour simuler les systèmes à retard pur et des systèmes exotiques

Je propose une nouvelle boîte à outils afin de simuler les systèmes (en boucle ouverte et boucle fermée) possédant un retard pur dans leur chaîne d'action, ainsi que quelques systèmes dont le modèle n'est pas un rapport de deux polynômes de la variable complexe s : je les nomme des systèmes exotiques ou non entiers ; c'est un premier essai de synthèse.

10.1 Transformation de quelques macros afin de simuler (en boucle ouverte) un système avec retard pur

La majorité des logiciels de simulation posent problème dès que l'on souhaite faire la simulation de systèmes dont le modèle mathématique n'est pas une fraction rationnelle ou un quadruplet $\left[\begin{array}{cccc} A & B & C & [D] \end{array}\right]$. Afin de palier en partie à cette lacune, nous allons proposer à partir des fonctions contenues dans le logiciel, un ensemble de macros, permettant d'introduire l'étude des systèmes à retard pur, et plus généralement l'étude des systèmes possédant dans leur chaîne d'action un élément dont le modèle n'est pas une fraction rationnelle en s : cas du retard pur dont la transmittance est représentée par l'opérateur $G_r(s) = \exp(-Ts)$. On peut aussi envisager d'autres types d'opérateurs (qui peuvent être des fractions d'une autre variable par exemple s^α avec α réel).

Dans cette partie nous nous intéresserons à la simulation de systèmes non bouclés qui possèdent en cascade avec un système classique de transmittance $G(s) = \frac{N(s)}{D(s)}$ ($N(s)$ est un polynôme de degré m et $D(s)$ un polynôme de degré $n \geq m$), un retard pur représenté par l'opérateur $G_r(s) = \exp(-Ts)$.

Les réponses peuvent être de deux types : soit temporelles, soit fréquentielles.

En ce qui concerne les premières, la création d'une fonction d'entrée retardée dans le temps de T associée à l'instruction `csim()` donne le résultat convenu.

Pour les réponses fréquentielles on transformera certaines macros afin quelles puissent accepter, en argument d'entrée, une liste dans laquelle on définit le système $G(s)$ et le retard T.

10.1.1 Exemple de programme Scilab pour simuler une réponse temporelle à commande retardée

Je rappelle la méthode déjà rencontrée dans ce document afin de générer un signal d'entrée.

Création par Scilab d'un signal d'entrée retardé par une fonction

Je crée un signal $u(t)$, nul jusque l'instant T puis valant 1 de T à $T+T_1$ et revenant ensuite à 0. Voici la fonction nommée `creneau.sce` (fonction stockée dans le répertoire `SYSR`) :

```
function [u]=creneau(t,T,T1)
u=bool2s(T<=t)-bool2s(t>(T+T1)) ;
```

et voici le programme principal :

```
-->s=%s ;sl=syslin('c',1/(1+s)) ;
-->t=linspace(0,10,101) ;
-->T=.5 ;T1=2 ;
--> ;getf("/home/lpovy/SYSR/creneau.sce") ;
-->y=csim(creneau,t,sl) ;
-->plot(t,y)
```

Les grandeurs T et T_1 représentent respectivement le retard pur et la durée du créneau. Ne voulant pas le définir dans la fonction, afin de conserver toute la généralité, je donne ici à T et à T_1 dans le programme principal (variables globales) les valeurs $0.5\,s$ et $2\,s$. 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 non le résultat, que j'ai appelé `u` . En effet `u` est de type 1 (scalaire), tandis que `creneau` est de type 13 (fonction compilée) et `csim` accepte pour entrée un argument de type : `function` ou `list` .

Création par Scilab d'un signal en utilisant la notion de liste (surcharge d'opérateur)

Cette deuxième façon de procéder utilise la notion de liste, manière élégante de surcharger un opérateur : ici une fonction[1].

```
-->t=linspace(0,10,101) ;
--> ;getf("/home/lpovy/SYSR/creneau.sce") ;
-->ul=list(creneau,.5,2)
ul =
    ul(1)
[u]=function(t,T,T1)
    ul(2)
0.5
```

[1] Voir l'article Scilab, de Serge Steer dans la revue Linux Magazine 14 de février 2000.

```
    ul(3)
2.
-->y=csim(ul,t,sl);
Warning redefining function : uu
-->plot(t,y)
```

La liste `ul` est constituée de trois éléments : la fonction, et les paramètres T et T1.

Création par Scilab d'un signal en ligne de commande

On peut aussi, sans définir un sous programme, créer le signal retardé à la ligne de commande : voici un exemple.

```
-->deff('u=creneau(t,T,T1)','if (t>=T&t<T1) then,u=1;else u=0;end')
-->s=%s;sl=syslin('c',1/(1+s));
-->t=linspace(0,10,101);
-->T=.5;T1=2;
-->y=csim(creneau,t,sl);
-->plot2d(t',y')
```

Comme précédemment, vous devez utiliser dans la fonction `csim` l'expression `creneau` et non pas le vecteur `u` , car ce vecteur est un scalaire.

Une autre façon de procéder est de créer dans la fenêtre Scilab la fonction par la syntaxe :

```
-->function [u]=creneau(t,T,T1)
-->u=bool2s(T<=t)-bool2s(t>(T+T1));
-->endfunction
-->//La fonction est comprise entre function et endfunction
-->s=%s;sl=syslin('c',1/(1+s));
-->t=linspace(0,10,101);
-->T=.5;T1=2;
-->y=csim(creneau,t,sl);
-->plot2d(t',y')
```

Cette dernière option n'est valable que depuis la version 2.6 de Scilab.

10.1.2 Exemple de programme Scilab pour simuler une réponse temporelle à sortie retardée

J'ai crée une petite fonction permettant de retarder d'une valeur de temps donnée un signal quelconque. Voici la fonction que je nomme `fr.sce` et que je mets dans `/home/lpovy/SCI`.

```
function [tt,fr]=fr(t,f,tr)
l=length(f);
i=max(find(t<tr));
fr=[zeros(1,i),f];
```

```
fr=fr(1:l);
trr=t(1:i);
tt=[trr,t+ones(t)*tr];
tt=tt(1:l);
endfunction
```

Enfin voici le programme principal :

```
//réponse à un échelon
-->s=%s;t=linspace(0,10,101);tr=4.99;
-->sys=syslin('c',1/(1+s+s*s));
-->f=csim('step',t,sys);
//chargement de fr.sci
--> ;getf("/home/lpovy/SCI/fr.sce");
//réponse retardée
-->[t1,fretard]=fr(t,f,tr);
//tracé des réponses retardée et non retardée
-->plot2d([t1;t]',[fretard;f]',style=[6,3],axesflag=3)
```

10.1.3 Réponse fréquentielle : les programmes à transformer

Dès que l'on aborde avec des logiciels de simulation les réponses fréquentielles des systèmes à retard pur, on doit dans le calcul de la réponse fréquentielle rajouter à la phase de $G(s)$ un terme en $-2\pi fT$ (f est la fréquence et T le retard). On doit donc tout d'abord définir un système retardé : ceci peut être fait en utilisant la notion de liste : l'opérateur `syslin` va être **surchargé** avec un vecteur de scalaires `T` . Voici un petit programme permettant ceci.

```
-->s=%s;s1=1/(1+s+s*s);s2=(1+s)/(1+s+2*s*s);
-->sl=syslin('c',[s1;s2])
-->T=[2;3];
-->sl=list(sl,T)
sl =
sl(1)
!     1       !
! ---------   !
!          2  !
! 1 + s + s   !
!              !
!   1 + s      !
! ---------   !
!           2!
! 1 + s + 2s  !

sl(2)
! 2.!
! 3.!
```

```
-->type(sl)
ans =
    15.
-->type(sl(1))
ans =
    16.
-->type(sl(2))
ans =
    1.
```

Cette façon de procéder est très élégante, mais va demander de transformer légèrement les principales fonctions intervenant dans l'étude des réponses fréquentielles d'un système, afin que ces fonctions **acceptent comme argument d'entrée une liste**. Nous voyons bien par l'instruction `list()` que le vecteur `sl` est une liste comprenant un vecteur constituée de deux systèmes, `sl(1)` et `sl(2)`, et d'un vecteur de scalaires `T` : à `sl(1)(1,1)`, premier système on associera la première composante du vecteur `T` , soit `sl(2)(1,1)` ou `sl(2)(1)` , ce qui représente la même chose. Le vecteur `T` peut être un vecteur ligne ou un vecteur colonne.

Il faut donc maintenant que les principales macros de Scilab traitant de la réponse fréquentielle, acceptent comme argument d'entrée une liste (*type*15) . Voici les principales fonctions concernées.

- `dbphifr` : Cette fonction est nouvelle et remplace deux fonctions anciennes à savoir `phasemag` et `dbphi`. Elle accepte comme argument d'entrée les transmittances classiques, et des listes : voir l'exemple précédent. Une remarque malgré tout ; afin de généraliser le problème et d'appliquer ces programmes à des systèmes peu courants, cette fonction accepte comme argument d'entrée une liste comprenant pour son premier élément, un vecteur colonne de systèmes linéaires, de polynômes, ou de fractions rationnelles, et dans ces deux derniers cas on considérera que l'on travaille en temps continu. Quant au deuxième terme de la liste, il peut être un vecteur de scalaires, et alors on a affaire à des systèmes possédants des retards fixes en cascade avec des transmittances classiques, ou c'est une seconde liste de même dimension que le vecteur sytème, liste pouvant mélanger des nombres (retards purs) et des fonctions. Dans ce dernier cas, la ou les fonctions considérées, sont définies dans des fichiers et chaque fonction renvoie pour un vecteur fréquence, défini par l'utilisateur ou calculé par la fonction `calfreq`, à condition qu'il existe dans la première liste un ou des systèmes, deux vecteurs de nombres représentants respectivement le gain et la phase de la fonction pour le vecteur fréquence donné.
  La syntaxe est la suivante :
  `[db,phi,fr[,splitf]]=dbphifr(sys,fmin,fmax[,pas])`
  `[db,phi,fr[,splitf]]=dbphifr(sys,vecteurligne)`
  L'écriture du vecteur ligne est : `vecteurligne=[.1,.2,.3]` par exemple.
  Attention si on utilise la syntaxe :
  `[db,phi,fr[,splitf]]=dbphifr(sys,fmin,fmax)`
  c'est l'instruction `calfrq()` qui discrétise la fréquence au mieux, en prenant le système non retardé pour référence.

- `rrpfreq` : je l'ai nommée `rrpfreq` mais elle peut remplacer la fonction `repfreq`, elle accepte une liste en entrée, elle peut aussi traiter les systèmes classiques, ou autre et donne la réponse fréquentielle (le nombre complexe) ; sa syntaxe est : `[rrpf,fr[,splitf]]=rrpfreq(sys,fmin,fmax[,pas])` `[rrpf,fr[,splitf]]=rrpfreq(sys,vecteurligne)`. La remarque précédente s'applique à cette instruction.
- Enfin, cinq fonctions nécessaires aux tracés des réponses ont été retouchées pour accepter directement des listes comme argument d'entrée. Voici ces fonctions : `bblack bbode ggainplot nnyquist`

Voici un exercice qui va nous permettre de mettre en évidence l'influence du retard pur sur un système du second ordre classique (ne pas oublier de charger le répertoire où sont situées les fonctions précédentes) :

```
-->s=%s ;sl=syslin('c',1/(1+s+s*s)) ;
-->SL=list([sl ;sl ;sl ;sl ;sl ;sl],[0 ;.1 ;.2 ;.5 ;.8 ;1]) ;
-->com=['T=0';'T=0.1';'T=0.2';'T=0.5';'T=0.8';'T=1'] ;
//ici faire un getd()
-->[D,P,F]=dbphifr(SL,[.1,.2,.3])
 F =
! 0.1 0.2 0.3 !
 P =
! - 46.07296 - 114.7432 - 143.56107 !
! - 53.27296 - 129.1432 - 165.16107 !
! - 60.47296 - 143.5432 - 186.76107 !
! - 67.67296 - 157.9432 - 208.36107 !
! - 74.87296 - 172.3432 - 229.96107 !
! - 82.07296 - 186.7432 - 251.56107 !
 D =
! 1.1857519 - 2.8206354 - 10.030795 !
! 1.1857519 - 2.8206354 - 10.030795 !
! 1.1857519 - 2.8206354 - 10.030795 !
! 1.1857519 - 2.8206354 - 10.030795 !
! 1.1857519 - 2.8206354 - 10.030795 !
! 1.1857519 - 2.8206354 - 10.030795 !
-->[REP,F]=rrpfreq(SL,[.1,.2,.3])
 F =
! 0.1 0.2 0.3 !
 REP =
  column 1 to 2
! 0.7952166 - 0.8255722i - 0.3024945 - 0.6563664i !
! 0.6854745 - 0.9187294i - 0.4562228 - 0.5605181i !
! 0.5649220 - 0.9973977i - 0.5812849 - 0.4294504i !
! 0.4354603 - 1.0603364i - 0.6698227 - 0.2713987i !
! 0.2991312 - 1.106553 i - 0.7162731 - 0.0962941i !
! 0.1580846 - 1.1353185i - 0.7177175 + 0.0848610i !
```

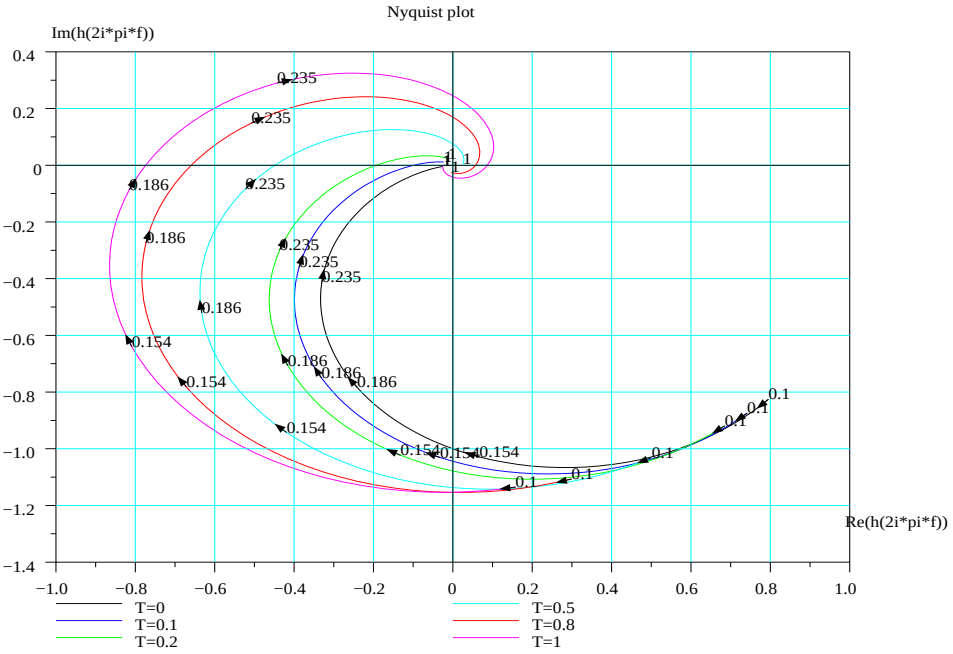


FIG. 10.1: Lieux de Nyquist avec retard

```
  column 3
! - 0.2535018 - 0.1871637i !
! - 0.3045996 - 0.0807002i !
! - 0.3129172 + 0.0370975i !
! - 0.2772866 + 0.1496849i !
! - 0.2027119 + 0.2412495i !
! - 0.0996669 + 0.2989313i !
-->nnyquist(SL,0.1,1,com)
-->xset('window',1)
-->bbode(SL,0.1,1,com)
-->xset('window',2)
-->bblack(SL,0.1,1,com)
```

Je reproduis ces trois lieux sur les figures ci-dessous (FIG. 10.1, 10.2, 10.3).

Une explication sur ces différentes commandes est nécessaire. Par la première instruction je construis une liste (*type*15) contenant deux vecteurs de même dimension : le premier vecteur colonne est constitué de cinq fois le le système linéaire `sl` (ici je ne cherche qu'à étudier l'influence d'un retard pur sur un système), quant au deuxième vecteur, c'est un vecteur de scalaires en ligne ou colonne de dimension cinq donnant respectivement

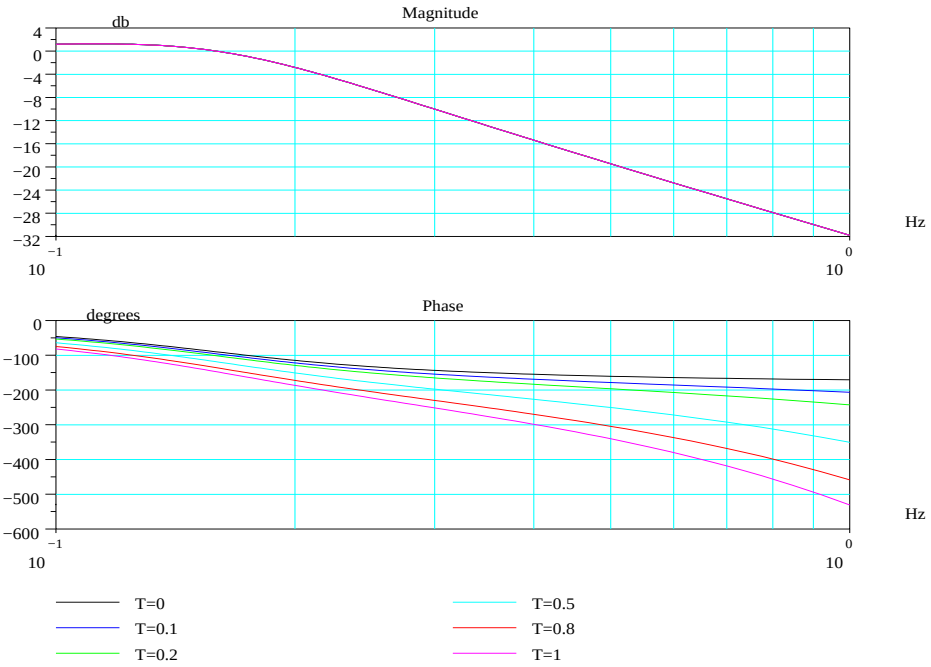


FIG. 10.2: Lieux de Bode avec retard

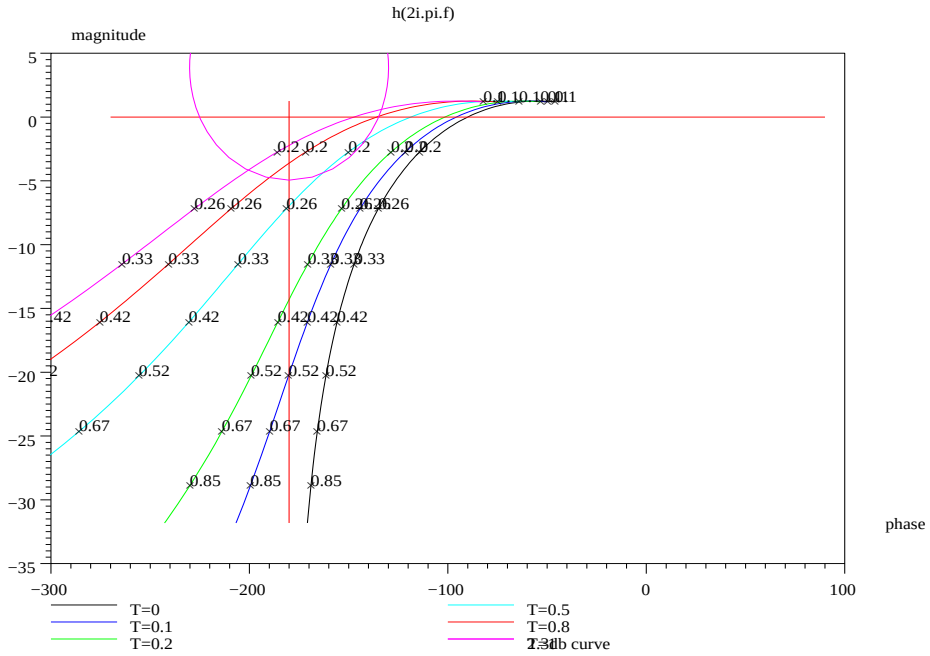


FIG. 10.3: Lieux de Black avec retard

la valeur du retard à associer à chacun des systèmes[2].

Par l'instruction `[D,P,F]=dbphifr()`, on calcule pour la liste précédente `SL` et pour trois fréquences, le gain, la phase et cette fonction retourne le vecteur fréquence. L'instruction `rrpfreq()` donnant la valeur du nombre complexe pour cette même liste.

Enfin, ayant légèrement modifié les graphiques fréquentiels pour accepter ce type de structure, on donne les lieux classiques du système retardé avec plusieurs retards.

10.2 Rappels de mathématiques

Depuis quelques années, on trouve dans la littérature scientifique, connus sous le nom de systèmes non entiers ou fractionnaires, des modèles de transfert qui font apparaître, non pas l'opérateur de Laplace s mais un opérateur de type s^α qui symbolise la dérivation non entière : donnons les deux définitions admises de cette dérivée.

Définition de Riemann-Liouville La dérivée d'ordre α ($\alpha < 1$) d'une fonction f continue, est par extension de la notion de dérivée, l'intégrale suivante :

$$D^\alpha(f(t)) = f(0)\frac{t^{-\alpha}}{\Gamma(1-\alpha)}u(t) + \int_0^t \frac{\theta^{-\alpha}}{\Gamma(1-\alpha)} D^1(f(t-\theta))d\theta$$

où $u(t)$ est la fonction échelon unitaire et où $D^1(f(t))$ est la dérivée première de la fonction $f(t)$.

Définition de Grunwald-Letnikov Cette deuxième définition de la dérivée non entière d'une fonction est donnée par une série et pourra donc être utilisée pour le calcul numérique de la dérivée.

$$D^\alpha(f(t)) = \lim_{h\to 0} \frac{1}{h^\alpha} \sum_{k=0}^{\infty} (-1)^k \binom{\alpha}{k} f(t-kh)$$

où $\binom{\alpha}{k} = \frac{\Gamma(\alpha)}{\Gamma(k)\Gamma(\alpha-k)}$ et où $\Gamma(\,)$ est la fonction gamma, extension de la fonction factorielle.

Si la fonction est causale et si h est petit, on peut calculer numériquement la dérivée par l'expression :

$$D^\alpha(f(t_m)) = \frac{1}{h^\alpha} \sum_{k=0}^{m} (-1)^k \binom{\alpha}{k} f(t_{m-k})$$

En appelant h le pas de discrétisation du temps, en notant que $f(t_i) = f(ih)$ et sachant que $\mu_{j,\alpha} = (-1)^j \binom{\alpha}{k}$ on a alors la récurrence suivante :

$$\mu_{0,\alpha} = 1$$

[2] On pouvait faire autrement en créant un vecteur de systèmes linéaires retardés avec un liste typée, par exemple SLR=tlist(['r','num','den','retard','dt'],N,D,RET,dom).

et

$$\mu_{j,\alpha} = (1 - \frac{\alpha+1}{j})\mu_{j-1,\alpha}$$

et ceci pour $j = 1 \cdots m$.

Ces deux définitions sont identiques si la fonction $f(t)$ est $n-1$ fois continûment dérivable et si l'opérateur $Q^n(f)$ est intégrable sur l'intervalle de temps $[0, t]$.

Quelques propriétés de la dérivation non entière C'est un opérateur linéaire, on peut analytiquement trouver les expressions de la dérivée $\alpha\, ieme$ de certaines fonctions relativement simplement, des exemples :

$$D^\alpha(\exp(zt)) = z^\alpha \exp(zt)$$

$$D^\alpha(\cos(\omega t + \varphi)) = \omega^\alpha \cos(\omega t + \varphi + \alpha\frac{\pi}{2})$$

$$D^\alpha(\sin(\omega t + \varphi)) = \omega^\alpha \sin(\omega t + \varphi + \alpha\frac{\pi}{2})$$

$$D^\alpha(\exp^{\frac{t}{\tau}} \sin \omega t) = R^\alpha \exp^{\frac{t}{\tau}} \sin(\omega t + \alpha\theta)$$

avec : $R = \mid \frac{1}{\tau} + j\omega \mid$ et $\theta = \arg(\frac{1}{\tau} + j\omega)$ $(j^2 = -1)$

Une autre propriété intéressante pour décrire un système par son transfert, est la transformée de Laplace de la dérivée non entière :

On montre que pour une fonction causale $f(t)$ on a :

$$L(D^\alpha f(t), s) = s^\alpha L(f(t), s)$$

ainsi on déduit facilement des modèles sous forme transfert.

Deux exemples :

$$G_{explicite}(s) = \frac{K}{1 + (\tau s)^\alpha}$$

$$G_{implicite}(s) = \frac{K}{(1 + \tau s)^\alpha}$$

ici α est un réel. Je donnerai par la suite deux modèles issues d'expériences concrètes, mais avant introduisons la notion d'équations différentielles généralisées.

Equations différentielles généralisées A partir de ces modèles on peut retrouver l'équivalent des équations différentielles ordinaires en les définissant comme des relations linéaires où intervient l'opérateur de dérivation non entière, par exemple, si l'on considère le modèle explicite, on obtient l'équation :

$$\tau^\alpha D^\alpha(y(t)) + y(t) = Ku(t)$$

et si $\alpha = 1$ on retrouve l'équation différentielle ordinaire du premier ordre.

Solutions analytiques de ces équations exotiques A partir des propriétés de la transformée de Laplace, avec une excellente table de transformées[9], on peut quelquefois trouver l'original $y(t)$ à partir de $u(t)$ et des conditions initiales (sur la fonction et ses dérivées non entières). Ces solutions analytiques (quand on les trouve) font généralement appel aux fonctions spéciales (fonction $\Gamma(\,)$, $Erf(\,)$, de Mittag-Leffler etc...).

10.3 Comment faire une simulation temporelle d'un système non entier ?

On entre ici dans le domaine de la recherche, il n'est pas question d'utiliser l'outil classique que l'on trouve dans les logiciels de simulation, à savoir le groupe de programmes fortran ODEPACK (qu'utilise l'instruction `csim()`), qui ne traite que des équations différentielles ordinaires (mises sous forme de systèmes différentiels ou sous forme d'équations d'état éventuellement). Mais avant donnons quelques exemples de systèmes à modèles « exotiques ».

10.3.1 Exemples de modélisation de système non entier : un système distribué[3]

Je rappelle d'abord le concept de système non entier en introduisant la notion de transfert en prenant un exemple classique, à savoir l'équation de la propagation de la chaleur (diffusion). Cet exemple a été traité, dans ma jeunesse (!!), dans le cours de Méthodes Mathématiques de la Physique par la transformée de Laplace.

Considérons une barre uniforme, isolée, constitué d'un matériau homogène dont le coefficient de conduction positif est noté $C = c^2$.

On chauffe cette barre (commande : $u(t)$ flux de chaleur) à une extrémité d'abscisse $x = l$: la température de la barre $T(x,t)$ mesurée par rapport à la température initiale T_0 (homogène) de la barre, vérifie l'équation de la chaleur à savoir

$$c^2 \frac{\partial T(x,t)}{\partial t} = \frac{\partial^2 T(x,t)}{\partial x^2}$$

avec les deux conditions (initiale et aux limites) suivantes

$$\begin{aligned} \frac{\partial T(0,t)}{\partial t} &= 0 \\ \frac{\partial T(l,t)}{\partial x} &= u(t) \end{aligned}$$

Dans tout l'exercice on prendra comme variable de sortie la température à l'autre extrémité de la barre : $x = 0$, (voir FIG. 10.4), on défini pour cette variable spatiale $x = 0$ un système noté SYS0 et pour une abscisse x quelconque on aura un système

[3] On lira avec intérêt le vieil article de G.JUMARIE sur la notion de pôles et résidues généralisés.[8].

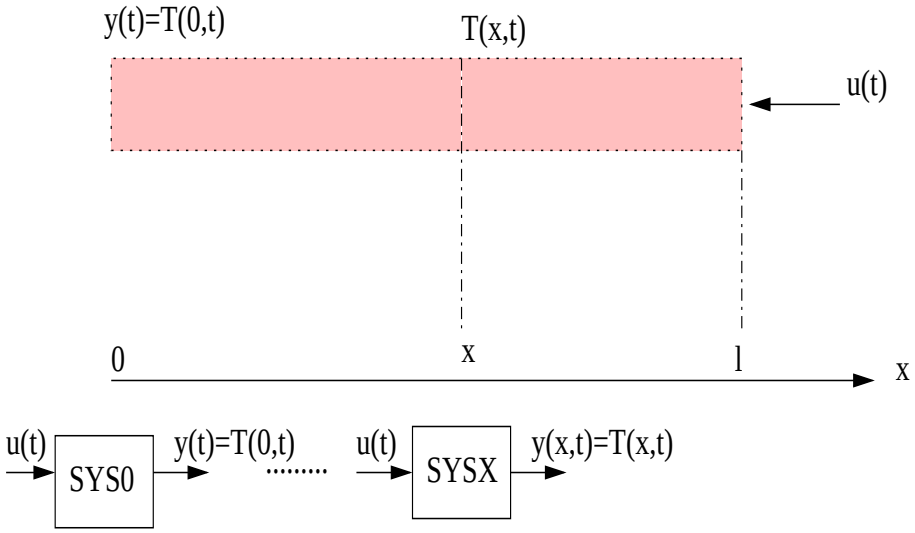


FIG. 10.4: Echauffement d'une barre.

noté SYSX. En prenant la transformée de Laplace de cette équation (la variable t est transformée en la variable s), on obtient :

$$\frac{\partial^2}{\partial x^2}T(x,s) = c^2 sT(x,s) - c^2 T(x,0)$$

équation différentielle linéaire par rapport à la variable x ($T(x,s)$ est la transformée de Laplace de $T(x,t)$). En intégrant cette dernière équation compte tenu des conditions aux limites on obtient :

$$T(x,s) = y(x,s) = \frac{\exp(cx\sqrt{s}) + \exp(-cx\sqrt{s})}{c\sqrt{s}(\exp(cl\sqrt{s}) - \exp(-xl\sqrt{s}))}u(s)$$

Pour une abcisse donnée x on a un transfert (fonction de transfert) qui vaut :

$$G(x,s) = \frac{\exp(cx\sqrt{s}) + \exp(-cx\sqrt{s})}{c\sqrt{s}(\exp(cl\sqrt{s}) - \exp(-cl\sqrt{s}))}$$

et comme cas particulier pour $x = 0$ on a :

$$G(0,s) = G_0(s) = \frac{2\exp(-cl\sqrt{s})}{c\sqrt{s}(1 - \exp(-2cl\sqrt{s}))}$$

cette fonction (de transfert n'est pas un rapport de deux polynômes de la variable complexe s) et fait de plus apparaître seulement la variable complexe $\sqrt{s}$: on peut dire que l'on a un système non entier.

Remarque : on trouvera dans les ouvrages de mathématique la solution temporelle exacte quand $u(t) = u_0$, réponse à un échelon. De même ce système étant causal on a donc, au sens des distributions, une relation entrée-sortie de la forme

$$y(0,t) = \int_0^t h_0(t-\tau)u(\tau)d\tau$$

avec $h_0(t)$[4] la transformée de Laplace inverse (au sens des distributions) de $G_0(s)$. Théoriquement cette dernière relation doit nous permettre de calculer la réponse à tout type de signal, connaissant $h_0(t)$ (théorème de convolution).

10.3.2 Exemples de systèmes non entiers, modèle à dérivée non entière

Modèle utilisé en résistance des matériaux Quand un système fait intervenir la physique des surfaces et des interfaces, quand on étudie des procédés mettant en oeuvre des phénomèmes viscoélastiques, quand on étudie la diffusion fractale, on est amené lors de la modélisation du système, à introduire la notion de dérivée non entière [6, 7] d'une variable (dérivation au sens de Riemann-Liouville) : voici un exemple de modélisation[5].

L'étude d'un matériau viscoélastique en régime harmonique a conduit Vinh[10] à exprimer le module de Young complexe $E(j\omega)$ sous la forme :

$$E(j\omega) = \frac{\sigma(j\omega)}{\epsilon(j\omega)} = E_0 \frac{\prod_{i=0}^{i=n}(1+(jT_i\omega)^{\alpha_i})}{\prod_{k=1}^{k=q}(1+(jT_k\omega)^{\alpha_k})}$$

Une formule simplifiée de cette expression, où $\sigma(j\omega)$ est la contrainte complexe, et $\epsilon(j\omega)$ la déformation complexe, peut être utilisée pour de nombreux matériaux, et l'on obtient :

$$E(s) = \frac{\sigma(s)}{\epsilon(s)} = E_0 \frac{1+(T_1 s)^{\alpha_1}}{1+(T_2 s)^{\alpha_2}}$$

ce modèle dit à quatre paramètres, peut être considéré pour les automaticiens, comme un système non entier et son étude fréquentielle par Scilab ne pose pas de problèmes particuliers[6] : voici un exemple de programme Scilab. L'exemple proposé dans l'article de M. SOULA et Y. CHEVALIER est le suivant :

$$E(s) = 1,005 \frac{1+\frac{1}{3,23}(\frac{s}{650})^{0,65}}{1+(\frac{s}{650})^{0,65}}$$

cas particulier du modèle plus général :

$$E(s) = E_0 \frac{1+\frac{1}{Z}(T_1 s)^{\alpha}}{1+(T_1 s)^{\alpha}}$$

[4] Cette réponse impulsionnelle est malgré tout une fonction au sens classique du terme : on peut trouver son expression à partir de $G_0(s)$ avec une bonne table de transformées de Laplace (A. ERDELYI, ..., TABLE OF INTEGRAL TRANSFORMS, MCGRAW HILL, NEW YORK, 1954).

[5] Cet exemple est extrait de l'article de M. SOULA et Y. CHEVALIER, ESAIM Vol. 5, 1998, pages 193-204.

[6] Je ne dirais pas la même chose pour l'étude des réponses temporelles!

cela ressemble à un réseau correcteur à retard de phase non entier : dans ce réseau on introduit un nouveau paramètre à savoir α. Je donnerais par la suite, le programme permettant d'étudier la réponse fréquentielle de ce modèle.

Modèle issue de l'électrochimie `www-ist.cea.fr/publicea/` Ce modèle est celui d'un capteur permettant de détecter de manière sélective et reproductible la présence d'un composé biochimique (protéine, ADN, molécule toxique).

Cette équipe de chercheurs a introduit un modèle dit de Randles (impédance complexe de Randles) qui a la forme suivante :

$$Z(j\omega) = R_e + \frac{R_t(j\omega)^{0.5} + \sigma\sqrt{2}}{C_d R_t(j\omega)^{1.5} + \sigma\sqrt{2}C_d(j\omega) + (j\omega)^{0.5}}$$

les paramètres intervenants dans cette équation sont des résistances (R_e, R_t) une capacité (C_d) et un paramètre de diffusion σ.

10.3.3 Utilisation de la transformée de Laplace

Comme je l'ai dit au paragraphe 10.1 il n'est pas question d'utiliser le groupe de programmes ODEPACK pour simuler les réponses temporelles d'un système non entier. L'idée qui vient naturellement à l'esprit quand on a modélisé un système non classique sous forme transfert est de rechercher chez les analystes numériciens des méthodes permettant d'inverser l'intégrale de Laplace, à savoir calculer d'une manière numérique l'équation :

$$f(t) = \frac{1}{2\pi j}\int_{a-j\infty}^{a+j\infty} f(s)\exp(st)ds$$

avec comme condition : $a > c$, c étant tel que $f(s)$ converge pour $R(s) > c$.

A ma connaissance il n'existe que quelques programmes fortran permettant d'inverser numériquement une transformée de Laplace (qui ne soit pas un rapport de deux polynômes, car alors on intègre un système différentiel).

Revue de détail (!) pour le calcul d'une réponse temporelle

Utilisation de la linéarité et de la convolution Mis à part les systèmes linéaires classiques, dans certain cas on peut avec l'aide d'une bonne table de transformée de Laplace inverse, et si l'on peut mettre la transformée de Laplace sous la forme :

$$f(s) = \sum_{k=1}^{k=N} f_k(s)g_k(s)$$

connaissant les originaux de $f_k(s)$ et de $g_k(s)$ (à savoir $f_k(t)$ et $g_k(t)$), alors on aura pour l'original de $f(s)$ l'expression :

$$f(t) = \sum_{k=1}^{k=N} f_k(t) \star g_k(t)$$

ou l'opérateur « $\star$ » représente la convolution entre les deux fonctions $f_k(t)$ et $g_k(t)$.

La mise en oeuvre numérique de la convolution pose un certain nombre de problèmes : la précision sur le résultat peut être très aléatoire.

Inversion de fonctions méromorphes Si la fonction $f(s)$ dont on recherche l'original est méromorphe (i.e n'a que des pôles) alors la théorie des résidus dit que l'inversion de l'équation de Laplace se résume à calculer l'intégrale le long d'un contour fermé comprenant la droite $[c - j\infty, c + j\infty]$ et un demi cercle situé à gauche du plan complexe, de rayon infini. Si pour $t > T$ on a $|\exp(sT)f(s)| < A|s|^{-k}$avec $k > 0$ l'intégrale le long du demi cercle est nulle et l'inversion de l'intégrale de Laplace revient au calcul des résidus relatifs aux pôles.

Inversion utilisant la formule de quadrature de Gauss Si $F(s)$est une transformée de Laplace ayant pour expression $F(s) = s^{-\mu}G(s^{-1})$ avec $\mu \geq 1$ et $G(s^{-1})$ un polynôme de degré N, alors on trouve facilement l'original $f(t)$ numériquement en utilisant n points de la formule de quadrature de Gauss, avec $2n - 1 \geq N$. On montre que si $F(s) = s^{-\mu}G(s^{-1})$ alors :

$$f(t) = \frac{1}{2\pi j}\int_{c-j\infty}^{c+j\infty} F(s)\exp(st)ds$$

soit si $u = st$:

$$f(t) = \frac{t^{\mu-1}}{2\pi j}\int_{ct-j\infty}^{ct+j\infty} \exp(u)u^{-\mu}G(\frac{u}{t})du$$

ou encore

$$f(t) \simeq t^{\mu-1}\sum_{k=1}^{N} A_k G(\frac{u_k}{t})$$

Cette approximation donne la valeur exacte de l'original chaque fois que $G(s)$ est un polynôme en s^{-1} de degré inférieur ou égal à $2N-1$: le problème d'analyse numérique consiste donc à trouver d'une part les valeurs de u_k et ensuite les nombres A_k.

Résolution numérique d'une équation différentielle non entière On peut comme on l'a dit dans le paragraphe précédent utiliser la deuxième expression pour la dérivée non entière et utiliser la somme :

$$D^{\alpha}(f(t_m)) = \frac{1}{h^{\alpha}}\sum_{k=0}^{m}(-1)^k \binom{\alpha}{k} f(t_{m-k})$$

avec h le pas de discrétisation du temps, en notant que $f(t_i) = f(ih)$ et sachant que $\mu_{j,\alpha} = (-1)^j \binom{\alpha}{k}$ on a alors la récurrence suivante :

$$\mu_{0,\alpha} = 1$$

et

$$\mu_{j,\alpha} = (1 - \frac{\alpha+1}{j})\mu_{j-1,\alpha}$$

et ceci pour $j = 1 \cdots m$. C'est ces équations qu'il faut programmer avec le logiciel sous forme d'une fonction écrite en C ou en fortran, ou une fonction en language Scilab.

10.3.4 Programmes de simulation fréquentielle des systèmes exotiques

Je donnerai ici quelques exemples de simulation des réponses fréquentielles avec la boîte à outils que je propose.

- Modèles $G_{implicite}$ et modèle $G_{explicite}$. On crée dans des fichiers le modèle gain phase de ces deux fonctions de transfert, ainsi que le modèle du premier ordre qui nous servira de comparaison. Voici les fonctions $G_{implicite}$et $G_{explicite}$.

```
function[dbexp,phexp]=Gexplicite(fr,a)
den=1+(2*%pi*fr*%i)^a
dbexp=-(20/log(10))*log(abs(den))
phexp=-(180/%pi)*atan(imag(den),real(den))
endfunction

function[db1pli,ph1pli]=Gimplicite(fr,b)
den=(1+2*%pi*fr*%i)^b
db1pli=-(20/log(10))*log(abs(den))
ph1pli=-(180/%pi)*atan(imag(den),real(den))
endfunction
```

- Modèle utilisé en résistance des matériaux nommé $G_{resismat}$ qui ici est comparé à un retard de phase classique.

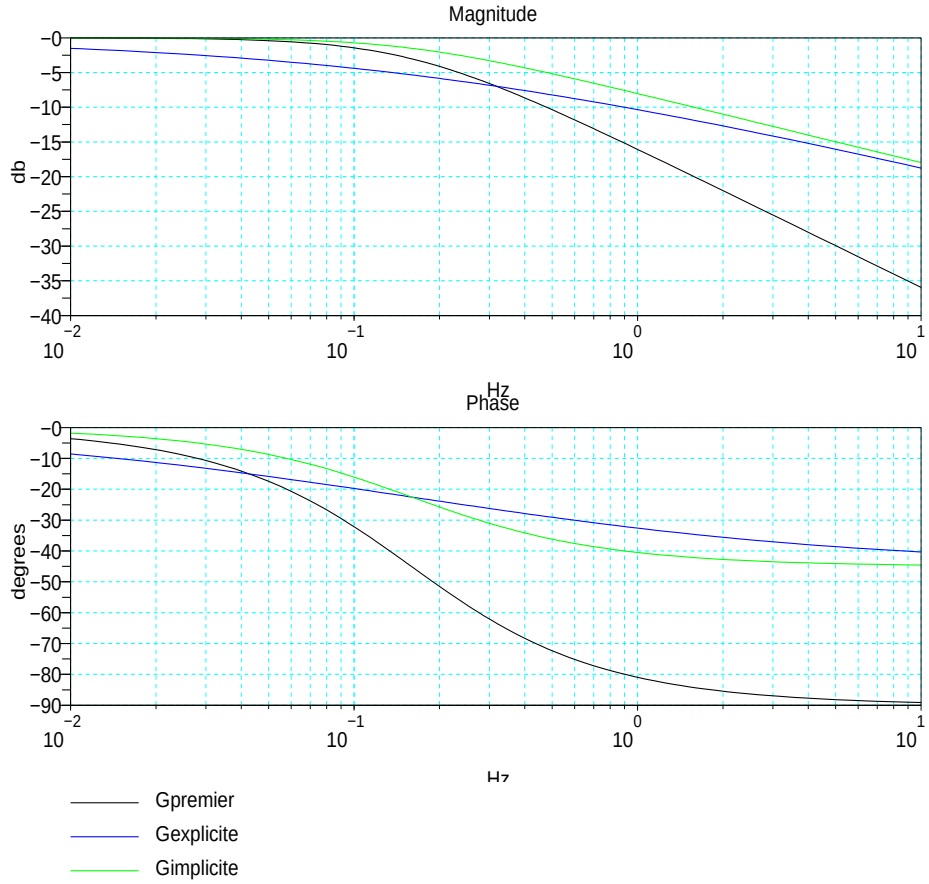
$$E(s) = 1,005 \frac{1 + \frac{1}{3,23}(\frac{s}{650})^{0,65}}{1 + (\frac{s}{650})^{0,65}}$$

Afin de travailler plus simplement je prendrais des variables reduites à savoir $\frac{E(s)}{1,005}$, $\frac{s}{650}$, ici $a = \frac{1}{3,23}$ et enfin $\alpha = 0,65$. Le modèle est donc de la forme $G_{resismat}(s) = \frac{1+a(s)^{\alpha}}{1+(s)^{\alpha}}$ (ce modèle sera comparé au modèle retard de phase $G_{retardphase}(s) = \frac{1+as}{1+s}$ $a > 1$).

```
function[dbmat,phmat]=Gresismat(fr,c,d)
den=1+(2*%pi*fr*%i)^c
num=1+d*(2*%pi*fr*%i)^c
//c ordre de dérivation, d le terme constant du numérateur
dbmat=(20/log(10))*(log(abs(num))-log(abs(den)))
phmat=(180/%pi)*(atan(imag(num),real(num))-atan(imag(den),real(den)))
endfunction.
```

Je ne donne pas les programmes issues du modèle électrochimique mais avec cette méthode ils ne posent aucun problème. Enfin le programme permettant de tracer les lieux.

```
-->s=%s ;Gpremier=syslin('c',1/(1+s)) ;

-->Guns=syslin('c',poly(1,'s','c'),1) ;

//je définis le système 1(s)

--> ;getf("/home/lpovy/SCILAB/AUTOENCOURSLYX/point-sce-sci/Gimplicite.sce") ;


Warning :redefining function : Gimplicite

--> ;getf("/home/lpovy/SCILAB/AUTOENCOURSLYX/point-sce-sci/Gexplicite.sce") ;

Warning :redefining function : Gexplicite

-->SL=list([Gpremier;Guns ;Guns],list(0,Gexplicite,Gimplicite))

//dans la seconde liste il y a 0

//(zéro seconde de retard pur en série

//avec un premier ordre) et deux fonctions

-->a=.5 ;b=0.5 ;

//variables globales valeurs de l'ordre de dérivation

-->com=['Gpremier','Gexplicite','Gimplicite']

-->bbode(SL,.01,10,com)
```



10.4 Essai de synthèse d'un système bouclé à retard

A vous de jouer ! pour ceux qui sont particulièrement compétents.

10.4.1 Approximation du retard pur : approximants de Padé

Une méthode classique (voir le logiciel Matlab) pour traiter les systèmes à retard pur (sur les entrées ou les sorties) consiste à approximer $\exp(-Ts)$ par une fraction rationnelle stable possédant des pôles et zéros symétriques par rapport à l'axe imaginaire : on réalise un déphaseur pur. Une bonne façon de trouver cette approximation est de

faire :

$$RTn = \left(\frac{1-\frac{T}{2n}s}{1+\frac{T}{2n}s}\right)^n$$
$$RTn = \left(\frac{2n-Ts}{2n+Ts}\right)^n$$

n prenant une valeur entière $1, 2, 3 \cdots$ Voyons avec Scilab les différences sur la réponse fréquentielle (on prend $T = 1$ ce qui ne change rien : car dans l'expression précédente seul le produit Ts apparaît). Voici un programme permettant de comparer les lieux de Bode d'un retard pur et des approximants de Padé pour $n = 1, 2, 3, 4, 5$.

```
-->s=%s ;
-->RT1=(2-s)/(2+s) ;RT2=((4-s)/(4+s))^2 ;
-->RT3=((6-s)/(6+s))^3 ;RT4=((8-s)/(8+s))^4 ;
-->RT5=((10-s)/(10+s))^5 ;
-->UNS=syslin('c',poly(1,'s','c')/poly(1,'s','c'))
//Création de la transmittance 1(s)
-->SL=list([UNS ;RT1 ;RT2 ;RT3 ;RT4 ;RT5],[1,0,0,0,0,0]) ;
```

Ici une remarque : à la transmittance $1(s)$ je rajoute un retard pur de 1 seconde, aux autres transmittances des retards nuls.

```
-->com=['exp(-s)';'n=1';'n=2';'n=3';'n=4';'n=5'] ;
-->bbode(SL,.1,1,com)
```

On peut voir ici que pour n supérieur ou égal à 4 on a, en phase, une bonne approximation dans la gamme de fréquence [0,1 , 0,5 Hz], (j'ai pris T=1 pour normaliser les résultats). On peut donc pour cette gamme de fréquences relatives faire la synthèse d'une manière classique. Ici c'est inutile de tracer les lieux de Black car le gain est toujours égal à un.

10.4.2 Etude de la stabilité des systèmes bouclés à retard [4]

Comme le proposent les auteurs du livre précédemment mis en référence, le système retardé considéré (régime libre) a pour équation :

$$\frac{dX}{dt} = AX(t) + BX(t-T), \, avec \, X \in R^n, \, T > 0.$$

On dira que $X(t)$ est le **vecteur d'état instantané**. L'état du système est le vecteur de fonctions $X_t(\theta) = \{X(t+\theta), -T \leq \theta < 0\}$. Quant au vecteur $X_0(\theta)$ c'est un vecteur contenant n fonctions définies et continues sur l'intervalle $\left[\ -T \quad 0\ \right]$. Pour les curieux on trouvera un modèle analogue, par le calcul opérationnel appliqué aux équations différentielles à argument retardé [9] (quand les coefficients de cette équation sont constants).

On montre qu'en étudiant l'équation caractéristique donnée par l'expression :

$$p(s,T) = det(sI - A - B\exp(-Ts)) = 0$$

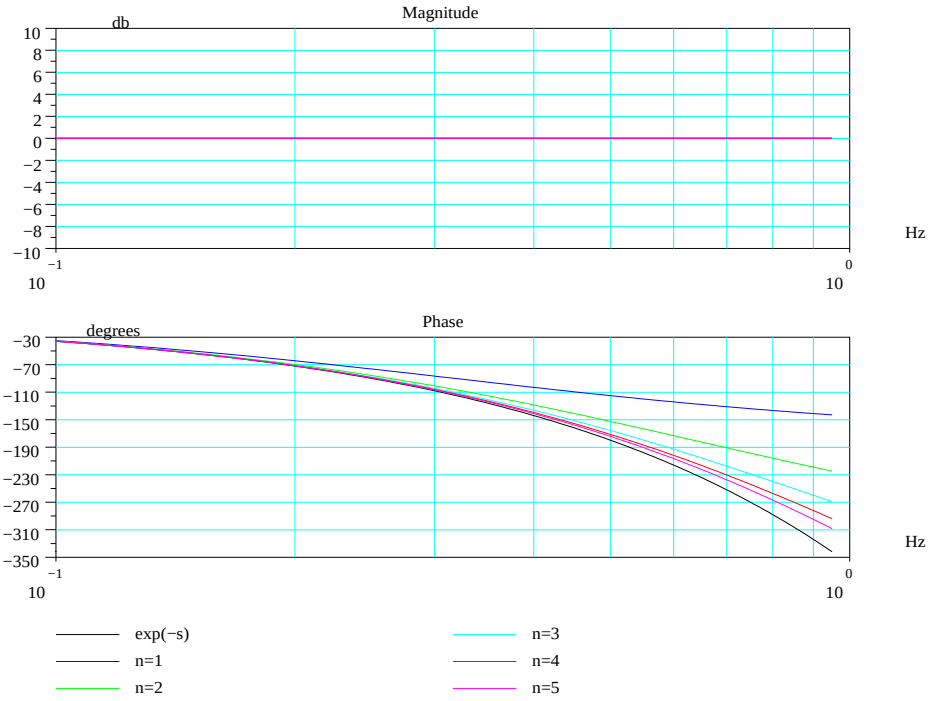


FIG. 10.5: Comparaison fréquentielle

le système retardé est stable si et seulement si son équation caractéristique ne possède pas de zéros dans le demi plan droit du plan complexe, même à l'infini.

Plus généralement, pour un système causal défini par une relation de convolution

$$y(t) = \int_0^t h(t-\tau)u(\tau)d\tau$$

ou encore par un transfert (section 9.2.1) de la forme

$$y(s) = G(s)u(s)$$

le système est $L_p - stable$, p entier positif, si pour toute entrée $u(t) \in L_p$, ($u(t)$ localement intégrable et $\int_0^\infty |u(t)|^p dt < \infty$), alors la sortie $y(t)$ est aussi dans L_p.

11 Conclusion provisoire

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. De même j'ai fait une petite introduction aux systèmes à retard et systèmes que j'ai appelé exotiques, le sujet reste ouvert et toute contribution et correction est souhaitable : vous connaissez mon adresse, vous pouvez aussi complèter ce document que je donne au format `.lyx` et au format `.tex`.

12 Annexes

12.1 Annexe 1 : Bibliothèque « autoelem »

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

12.1.1 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 classique.

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

1. `K` : C'est la valeur de l'expression $s^L f(s)$ pour $s = 0$: gain statique du système ou gain statique en vitesse ...
2. `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).
3. `TN` : Vecteur colonne dont chacun des éléments est un polynôme du premier et/ou second degré de la forme $1 + \tau_1 s$ ou $1 + \tau_2 + \tau_3 s^2$ (cas de racines complexes conjuguées).
4. `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 cette fonction dans le répertoire `.../autoelem`. Cette fonction ainsi que la fonction `rrpfreq.sci` utilise la fonction `bodfact.sci` pour faire le calcul du gain et de la phase ou pour calculer la réponse complexe pour `rrpfreq.sci`.

12.1.2 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 cinq diagrammes à savoir Bode, Black, Nyquist, Bode de gain seul : les nouvelles fonctions se nomment : `bbode.sci`, `bblack.sci`, `nnyquist.sci`, `ggainplot.sci`, et sont aussi situées dans le répertoire `.../autoelem`.

12.2 Annexe 2 : 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.

- **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.
- **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-N°.X`). 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, Windows2000 mais pas avec Windows Millenium semble t' il.
5. Pour utiliser le manuel en ligne (les fichiers `.cat` qui sont donnés avec la boîte à outils), mettre ces fichiers dans le répertoire `SCI/man/control` par exemple et rajouter au fichier texte `whatis` les noms des nouvelles fonctions avec la même syntaxe :
   ```
   abcd - state-space matrices @abcd
   bbode - Bode diagram @bbode
   ```

- **Troisième méthode**
  Vous pouvez enfin utiliser la méthode conseillée par l'organisation Scilab.org, car en plus du document que je donne au format .pdf vous trouverez aussi un grand répertoire (une boîte à outils) construit suivant les recommandations préconisées par ce site : le titre ce nomme « Guide for the toolboxes contibutions (general application) ».

12.3 Annexe 3 : Un peu de calcul matriciel

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

12.3.1 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 scalaire.

```
-->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.!
//mais
-->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)
an =
! 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 0.3126420 !
```

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 matrices de scalaires, dans le logiciel, il existe des fonctions spéciales manipulant des matrices de chaînes de caractères.

```
-->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.

Calcul matriciel

A faire

Application à l'automatique

A faire

12.4 Annexe 9 : Licences

Lucien Povy
lucien.povy@free.fr

12.4.1 Licence GNU GPL

Les programmes qui sont dans la boîte à outils **autoelem** sont distribués selon les termes de GPL v2.

GNU GENERAL PUBLIC LICENSE Version 2, June 1991.`http://www.gnu.org/licenses/licenses.fr.html#GPL`

12.4.2 Licence GNU FDL

Quant aux documents ils sont soumis à la licence GNU FDL. (Même site que pour la licence GNU GPL). Permission vous est donnée de distribuer, modifier, modifier des copies des pages des documents fournis, tant que ces notes sur les licences et le Copyright apparaissent.

Bibliographie

[1] SCILAB GROUP, Scilab reference manuel, INRIA.

[2] KUO, Automatic control systems PRENTICE HALL, New York, 1968.

[3] J-C. GILLE, M. PELEGRIN, P. DECAULNE, Théorie et technique des asservissements, DUNOD, Paris, 1956.

[4] P. BORNE, G. DAUPHIN-TANGUY, J.P. RICHARD, F. ROTELLA, I. ZAMBETTAKIS, Analyse et régulation des processus industriels, tome 1, Régulation continue, EDITIONS TECHNIP, Paris, 1993.

[5] J.J. DISTEFANO, A.R. STUBBERUD, I.J. WILLIAMS, Theory and problems of feedback and control systems, SCHAUM PUBLISHING CO, New York, 1967.

[6] A. OUSTALOUP, La Commande Crone, EDITIONS HERMES, Paris, 1991.

[7] A. OUSTALOUP, La dérivation non entière, théorie, synthèse et applications, EDITIONS HERMES, Paris, 1995.

[8] G. JUMARIE, Further applications of the non-stationary operational calculatus to the synthesis of closed-loop distributed systems, INT. J. SYSTEMS SCI, 1977, VOL. 8, NO. 12, 1337-1364.

[9] V. DITKINE, A. PROUDNIKOV, Transformations intégrales et calcul opérationnel, EDITIONS MIR, Moscou, 1978.

[10] T. VINH, Sur le passage du régime harmonique au régime transitoire viscoélastique, Extrait du mémorial de l'Artillerie Française, 3e Fasc, pp.725-776, 1967.