

# Notes sur les programmes de la boîte à outils Autoelem

lucien.povy@free.fr

2016- 2017

## 1 Les différents programmes proposés.

Ces programmes sont inclus dans la boîte à outils **Autoelem Toolbox** et permettent l'étude temporelle et fréquentielle des systèmes continus **sans** ou **avec** retard. De même certains programmes permettent aussi de discrétiser ces systèmes et de comparer le comportement fréquentiel et temporel des systèmes continus avec leurs homologues discrétisés.

### Remarques très importantes :

Depuis la version 6 de Scilab on ne peut plus utiliser la boîte à outils « **iodelay toolbox** », j'ai donc récupéré dans cette boîte, les programmes essentiels à l'étude des systèmes à retard pur en entrée-sortie, et inclus ces programmes dans ma boîte à outils, merci à Serge Steer (INRIA).

De même le programme **freson.sci** étant corrompu (ne donne pas la fréquence de résonance pour certains systèmes), je l'ai corrigé, mais en attendant la version officielle corrigée, vous devez utiliser la fonction **ffreson.sci** qui est dans ma boîte à outils : la macro **m\_margin.sci** utilise ce programme. Dès la correction faite, vous pouvez éliminer **ffreson.sci** et corriger **m\_margin.sci**.

## 2 Le programme « **mroots.sci** » : racines multiples d'un polynôme.

J'ai cherché avec ce programme à déterminer les multiplicités des racines d'un polynôme  $P(x)$  de degré  $n$ , en utilisant le programme **roots.sci** proposé par Scilab, quand le polynôme a des racines multiples. Pour savoir si un polynôme a des racines multiples j'utilise la fonction Scilab **bezout.sci** : si le pgcd de  $P(x)$  et de  $\frac{dP(x)}{dx}$  vaut 1 alors les racines sont simples, et alors le programme **roots.sci** me les donnera.

### Préalables :

1. Dans le programme proposé, je **veux** utiliser des fonctions Scilab déjà programmées, et de préférence des fonctions compilées (en fortran ou c) pour la rapidité d'exécution.
2. Un peu de théorie : on sait que quand un polynôme  $P(x)$  a des racines multiples, si  $x_k$  est une de ces racines, alors  $x_k$  est aussi une racine de  $\frac{dP(x)}{dx}$  et éventuellement d'autres dérivées. En prenant un polynôme unitaire  $P(x) = \prod_{k=1}^{k=K} (x - x_k)^{m_k}$  qui a des racines multiples (et/ou des simples), alors  $\frac{dP(x)}{dx}$  et  $P(x)$  vérifient :

$$g(x) = \text{pgcd}(P(x), \frac{dP(x)}{dx}) = \prod_{k=1}^{k=K} (x - x_k)^{m_k-1}$$

$$U(x) = \frac{P(x)}{g(x)} = \prod_{k=1}^{k=K} (x - x_k) \text{ et } V(x) = \frac{\frac{dP(x)}{dx}}{g(x)}$$

$U(x)$  est la décomposition sans carré (voir l'algorithme de Yun ou de Tobey-Horowitz) de  $P(x)$ .

Maintenant formons le rapport  $R(x) = \frac{V(x)}{U(x)}$ , ce rapport vaut :  $R(x) = \sum_{k=1}^{k=K} \frac{m_k}{x - x_k}$ . Les racines  $x_k$  sont les racines simples de  $U(x)$  et donc les racines recherchées et les entiers (les multiplicités) valent  $m_k = ((x - x_k) \frac{V(x)}{U(x)})_{x=x_k}$  ou encore  $m_k = \frac{V(x_k)}{(\frac{dU}{dx})_{x=x_k}}$ . Ce résultat est classique et constitue un exercice souvent posé dans le cursus de maths du premier cycle universitaire.  $R(x) = \frac{V(x)}{U(x)} = \sum_{k=1}^{k=K} \frac{m_k}{x - x_k}$  est aussi la décomposition en éléments simples du rapport  $\frac{\frac{dP(x)}{dx}}{P(x)}$ .

3. En analysant l'équation de Bezout donnée dans Scilab, `[g,U]=bezout(P,dP)` où  $P$  est le polynôme de départ et  $dP$  sa dérivée première, on constate facilement que le terme  $U(2,2) = -P/g$  et  $U(1,2) = dP/g$ , de plus  $\det(U) = -1$ . Donc trouver les racines simples et/ou multiples de  $P(x)$  revient à rechercher les racines simples de  $U(2,2)$ , et les multiplicités sont obtenues par un calcul sur  $U(1,2)$ . C'est donc à partir de ces considérations que je propose le programme Scilab `mroots.sci`.
4. Si on applique brutalement la théorie proposée, on peut avoir, pour certains polynômes, des problèmes : si  $P(x)$  n'a que des zéros simples alors le programme `[g,U]=bezout(P,dP)` doit donner théoriquement  $g == 1$  mais ce n'est pas toujours le cas, en particulier sur l'exemple proposé par Wilkinson (voir aide sur le programme `roots.sci`), on obtient avec le premier programme `mroots`, qui appliquait brutalement la théorie, une racine multiple de multiplicité 20 située à l'isobarycentre des racines, (un exemple analogue proposé par Samuel Gougeon donnait ce résultat avec Scilab -Windows mais pas avec Scilab-Linux).
5. Pour éviter ce problème je propose un test sur la véracité des racines et de leurs multiplicités respectives, en calculant la somme et le produit des racines et en vérifiant, avec une précision donnée, que cette somme et ce produit sont respectivement le deuxième et dernier coefficient du polynôme.

**Remarques :**

1. Comme je viens de le dire si le terme  $g$  donné par l'équation de Bezout vaut 1 alors  $P(x)$  n'a que des racines simples et le programme `mroots` utilise le programme `roots` seulement.
2. De même dans le programme proposé je détermine d'abord les racines nulles (s'il y en a) à partir des coefficients du polynôme pour ne pas « polluer » le programme `roots(U(2,2))` par ces racines nulles et diminuer le degré du polynôme de départ. Le reste du programme est la présentation de la solution.
3. Bien sur on utilise le programme `roots` pour calculer les racines simples de  $U(x)$  : c'est à cet endroit que des problèmes de précision peuvent se poser en plus des remarques précédentes.

La syntaxe de cette fonction est : `[rm,r] = mroots(p[,flag])`

### 3 Le programme « `frac_decomp` » : décomposition d'une fraction rationnelle, d'un système linéaire, en éléments simples sur le corps des réels.

Cette fonction peut remplacer la macro Scilab `pfss.sci`. La méthode proposée n'est pas la même que la méthode utilisée par Scilab. On utilise ici le programme `mroots.sci` qui donne les racines et les multiplicités correspondantes, on commence par rechercher les polynômes élémentaires, du premier et second degré et les multiplicités correspondantes. En fonction de ces polynômes et des multiplicités, on calcule en résolvant un système d'équations linéaires, le résidu relatif à chacune des racines, le reste du programme consiste en une présentation sous forme de vecteur, des éléments simples (on donne aussi le polynôme quotient des deux polynômes numérateur et dénominateur). Dans la documentation de la fonction je donne un exemple illustrant le programme dans un cas particulier, et bien sûr, des exemples.

La syntaxe de cette fonction est : `[elm[,elm1]]=frac_decomp(g)`

### 4 Le programme « `bodfact.sci` » : factorisation de Bode.

Quand on fait la synthèse d'un système, par des méthodes fréquentielles on est amené à comparer un lieu de Black (ou de Bode) par rapport au point  $-1$  qui a pour déphasage  $-180^\circ$  et pour gain  $0db$ . Pour cela on devra mettre la transmittance sous forme « standard ou de Bode », en calculant bien le gain statique  $k$  (pas le gain pour  $s \rightarrow +\infty$ ), le nombre d'intégrations  $l$  (ou de dérivations), pôles ou zéros à l'origine, et les termes du premier et/ou second ordre du numérateur et dénominateur du système. De même ce programme donne aussi, pour un système continu retardé, le retard pur  $d$  en série avec un système continu : voici donc les équations de toute transmittance continu :

$$g(s) = \frac{k}{s^l} e^{-ds} \frac{1+(b_1/b_0)s+(b_2/b_0)s^2+\dots+(b_m/b_0)s^m}{1+(a_{l+1}/a_l)s+(a_{l+2}/a_l)s^2+\dots+(a_n/a_l)s^{n-l}} = \frac{k}{s^l} e^{-ds} g_b(s) \quad \text{ou} \quad g(s) = \frac{k}{s^l} e^{-ds} \frac{t_n(s)}{t_d(s)}$$

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

On a bien sur  $t_n(0) = 1$  et  $t_d(0) = 1$ .

En ce qui concerne les systèmes discrets ou échantillonnés ils se mettent aussi sous une forme standard, à savoir :

$$g(z) = \frac{k}{(z-1)^l} z^{-d} \frac{t_n(z)}{t_d(z)}$$

avec  $\frac{t_n(z)}{t_d(z)} = 1$  pour  $z = 1$ , en plus le polynôme  $t_d(z)$  est unitaire.

La syntaxe est : `[k, l, d, tn, td, fdn, fdd] = bodfact(g[,flag])`.

Les paramètres retournés sont donnés dans les formules précédentes, à part `fdn` et `fdd` qui sont les fréquences où les racines du numérateur et dénominateur sont sur l'axe imaginaire pur pour les systèmes continus sans retard, ou sur le cercle de rayon unité pour les systèmes échantillonnés, ces fréquences conduisent à des discontinuités sur les différents lieux fréquentiels.

## 5 Le programme « dbphifr.sci » : calcul des gains, des phases, des fréquences, pour des systèmes continus OU échantillonnés.

Compte tenu de la nouvelle fonction `bodfact.sci` je propose la fonction `dbphifr.sci` qui renvoie, pour des fréquences données ou calculées par les programmes `calfrq` ou `calfrqrd` (pour des systèmes à retard en entrée et/ou sortie), les fréquences, les gains, les phases, ceci permettant le tracé des divers lieux. De plus on peut tracer les **pseudo-lieux** qui correspondent à des arguments  $s = \omega \exp(j\theta)$  ( $\theta$  est un **angle** en degrés compris entre 0 et +180°) (voir le polycopié que je propose, avec des exemples pour des systèmes continus avec ou sans retard).

La syntaxe de cette fonction est :

`[fr, db, phi, splitf] = dbphifr(sys[,fmin[,fmax[,pas[,angle]]...])`.

**Remarques :** Pourquoi avoir écrit ce programme ?

Pour me dispenser d'utiliser les fonctions `repfreq.sci` ou `repfreqrd.sci`, qui dans des cas très rares, conduisent à des lieux de Black et Bode erronés (voir l'exemple). Ceci est dû à l'instruction `phasemag.sci` qui, bien que cherchant à avoir une courbe de phase sans sauts brusques, utilise la définition de `atan.sci` d'un complexe. C'est pour cela que dans `dbphifr.sci` je calcule **séparément** les gains et phases des **premiers** et **seconds** ordres et intégrateurs constituant la fonction de transfert (en tenant compte du signe du gain statique : d'où la mise sous forme de Bode `bodfact.sci` de la fonction de transfert). En effet, dans ce cas, pour un premier et/ou second ordre, le déphasage est dans l'intervalle  $[-\pi, +\pi]$ .

Mais prenez l'exemple simple suivant, on suppose que **Autoelem** est chargé, exemple avec Scilab-6.0.0.

```
-->s=%s;sl=syslin("c",1,s^3+s*s+0.2*s)//B0 stable avec 1 intégration.
//-->cacsdlib.black//On utilise le lieu de black de Scilab avec Scilab-5.
-->black(sl,0.1,10)//non OK
//on utilise ici repfreq.sci, pas de factorisation de Bode, erreur sur phase.
-->figure(1);
-->black(sl,0.01,10)//OK c'est toujours le même système, fmin a changé
//on utilise ici repfreq.sci, pas de factorisation de Bode,
//mais on démarre à des fréquences plus faibles alors cela marche.
On revient avec le lieu de black de Autoelem Toolbox (nom bblack.sci).
-->//Autoelemlib.black si on utilise Scilab-5...sinon utiliser bblack.
-->figure(2);
-->bblack(sl,0.1,10)//C'est OK
ou
-->[f,db,phi]=dbphifr(sl,0.1,10);//Autoelem Toolbox est chargé.
//on utilise ici dbphifr.sci, avec la factorisation de Bode (bodfact).
-->figure(3);
-->black(f,db,phi)//OK c'est toujours le même système.
```

Le passage du « bon » dessin au « mauvais » se fait à la fréquence de départ  $f_{min}=0.0712$  hz : la partie réelle de la réponse est négative, et on passe pour la partie imaginaire, de négatif à positif quand la fréquence de départ augmente : on traverse la **coupure** de la fonction `atan.sci` (en fait de `atan2`). Comme `phasemag` cherche à avoir un déphasage sans discontinuités, cela dépend bien du point de départ.

Pour le physicien il y a bien une erreur dans le premier cas, car pour la phase, il y a un décalage de  $+360^\circ$ . Le logiciel Matlab se sort de cette contradiction, en effet on peut rajouter à l'instruction Bode, (et Nichols) une option **PhaseMatching = 'on'**.

La syntaxe :

```
[fr,db,phi[,splitf]] = dbphifr(sys,fmin,fmax[,pas[,angle]])
[fr,db,phi[,splitf]] = dbphifr(sys,freq[,angle])
```

## 6 Les approximants de Padé.

### 6.1 Développement de Taylor, Maclaurin.

En analyse, la série de Taylor d'une fonction  $f$  (au voisinage d'un point  $a$ ), appelée aussi le développement en série de Taylor de  $f$ , est une série entière construite à partir de  $f$  et de ses dérivées successives en  $a$ .

Soit  $f$  une fonction indéfiniment dérivable d'une variable réelle ou complexe et  $a$  un point au voisinage duquel la fonction est définie. La série de Taylor de  $f$  en  $a$  est la série de fonctions suivante :

$$f(a) + \frac{f^{(1)}(a)}{1!}(x-a) + \frac{f^{(2)}(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \dots$$

ce qui s'écrit sous forme synthétique :

$$\sum_{n=0}^{+\infty} \frac{f^{(n)}(a)}{n!} (x-a)^n.$$

Si  $a = 0$ , la série est aussi appelée la série de Maclaurin de  $f$ .

Si je produis cette petite note pratique, c'est que j'ai été confronté à la précision des calculs en voulant produire un programme Scilab noté `padedelay.sci` pour approximer le retard pur en utilisant la méthode générale.

Plus généralement j'ai fait deux programmes Scilab, `pade.sci` et `padedelay.sci` donnant une fraction rationnelle  $\frac{P(x)}{Q(x)}$  de degré  $m$  pour  $P(x)$  et  $n$  pour  $Q(x)$  : pour cela il faut connaître au moins  $m+n+1$  coefficients dans le développement de Maclaurin de la fonction de départ que l'on veut approximer.

Je me suis aperçu, en voulant utiliser la méthode générale (que l'on verra plus avant), que l'approximant du retard pur (ou de  $e^{-x}$ ) était bien meilleur en cherchant une récurrence sur les coefficients des polynômes  $P(x)$  et  $Q(x)$  : c'est le programme `padedelay.sci`. Il est donc intéressant de connaître le développement en série de la fonction étudiée pour trouver cette récurrence : c'est pour cela que vous trouverez le développement en série de quelques fonctions usuelles au paragraphe suivant.

## 6.2 Développements en série de fonctions usuelles

### 6.2.1 Utilisation des tables de développement en série.

Ceci est extrait de l'encyclopédie en ligne Wikipedia.

#### Notations

Les nombres  $B_{2n}$  apparaissant dans les développements de  $\tan(x)$  et de  $\tanh(x)$  sont les nombres de Bernoulli. L'expression  $\binom{\alpha}{n}$ , apparaissant dans le développement de  $(1+x)^\alpha$ , est un coefficient binomial (généralisé) :  $\binom{\alpha}{n} = \frac{\alpha(\alpha-1)\dots(\alpha-n+1)}{n!}$ . Les nombres  $E_k$  dans le développement de  $\sec(x)$  sont les nombres d'Euler.

Nom de la fonction	Série	R
Exponentielle	$\exp(x) = \sum_{n=0}^{+\infty} \frac{x^n}{n!}$	$\infty$
Logarithme	$\ln(1+x) = \sum_{n=1}^{+\infty} \frac{(-1)^{n+1}}{n} x^n$	1
Série géométrique	$\frac{1}{1-x} = \sum_{n=0}^{+\infty} x^n$	1
Binôme	$(1+x)^\alpha = 1 + \sum_{n=1}^{+\infty} \binom{\alpha}{n} x^n$	1
Fonctions	$\sin(x) = \sum_{n=0}^{+\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$	$\infty$
trigonométriques	$\cos(x) = \sum_{n=0}^{+\infty} \frac{(-1)^n}{(2n)!} x^{2n}$	$\infty$
	$\tan(x) = \sum_{n=1}^{+\infty}  B_{2n}  \frac{4^n(4^n-1)}{(2n)!} x^{2n-1}$	$\frac{\pi}{2}$
	$\sec(x) = \sum_{n=0}^{+\infty} \frac{(-1)^n E_{2n}}{(2n)!} x^{2n}$	$\frac{\pi}{2}$
	$\arcsin(x) = \sum_{n=0}^{+\infty} \frac{(2n)!}{4^n(n!)^2(2n+1)} x^{2n+1}$	1
	$\arccos(x) = \frac{\pi}{2} - \sum_{n=0}^{+\infty} \frac{(2n)!}{4^n(n!)^2(2n+1)} x^{2n+1}$	1
	$\arctan(x) = \sum_{n=0}^{+\infty} \frac{(-1)^n}{2n+1} x^{2n+1}$	1
Fonctions	$\sinh(x) = \sum_{n=0}^{+\infty} \frac{1}{(2n+1)!} x^{2n+1}$	$\infty$
hypergéométriques	$\cosh(x) = \sum_{n=0}^{+\infty} \frac{1}{(2n)!} x^{2n}$	$\infty$
	$\tanh(x) = \sum_{n=1}^{+\infty} B_{2n} \frac{4^n(4^n-1)}{(2n)!} x^{2n-1}$	$\frac{\pi}{2}$
	$\arg \sinh(x) = \sum_{n=0}^{+\infty} \frac{(-1)^n (2n)!}{4^n(n!)^2(2n+1)} x^{2n+1}$	1
	$\arg \tanh(x) = \sum_{n=0}^{+\infty} \frac{1}{2n+1} x^{2n+1}$	1
	$W_0(x) = \sum_{n=1}^{+\infty} \frac{(-n)^{n-1}}{n!} x^n$	$\frac{1}{e}$

Cette table peut être complétée par d'autres résultats dans les livres connus.

### 6.2.2 Trouver un développement de Maclaurin en utilisant la fft quand la variable « z » décrit un cercle unité.

On peut montrer que si la fonction de la variable complexe  $f(z)$  est analytique dans l'entourage du disque unité du plan complexe, alors les coefficients de la série de Maclaurin qui permettent de calculer les approximants de Padé, entre autre, sont calculables en utilisant le programme `fft`.

Voici un petit programme Scilab réalisant ce développement.

Nous voyons bien que le résultat, après un petit nettoyage, donne bien les 13 premiers coefficients du développement de Maclaurin de  $e^{-x}$ .

Je propose un programme intégré au programme `pade.sci` nommé `maclau.sci` permettant de calculer le développement de Maclaurin d'une fonction ou d'une chaîne de caractères.

---

**Algorithme 1 Maclaurin par fft de  $\exp(-z)$ .**

---

```
L=1024; z = exp(2*i*pi*(0:L-1)/L); //Racines L iemes de l'unité.
f = fft(exp(-z))/L; //Pour la fonction exponentielle (retard en automatique).
f = f(1:13) //Je ne garde que 13 coefficients car je cherche un pade 6x6.
f =
1. + 1.837D-18i
- 1. + 1.257D-16i
0.5 - 1.174D-16i
- 0.1666667 + 6.684D-17i
0.0416667 - 2.367D-17i
- 0.00833333 - 6.534D-20i
0.0013889 - 3.475D-18i
- 0.0001984 + 1.851D-18i
0.0000248 - 2.455D-18i
- 0.0000028 + 7.855D-18i
0.0000003 + 8.980D-18i
- 2.505D-08 - 1.478D-18i
2.088D-09 - 4.985D-20i
tol = 1.e-14*norm(f);
f = clean(f,tol);
if norm(imag(f),%inf) < tol then
    f = real(f); //ne conserver que les valeurs réelles.
else
    error('impossible')
return
end
```

---

## 7 Le programme « `padedelay.sci` » : approximation d'un retard pur en continu.

Le but de ce programme est d'approximer un retard pur  $d$  afin de faire d'une manière classique l'étude des systèmes à retard pur (en entrée-sortie) avec un approximant de type Padé  $hpade_{m,n}(s) = \frac{P(s)}{Q(s)}$ .

Pour faire cela nous avons utilisé une récurrence sur les coefficients  $b_j$  du polynôme numérateur de degré  $m$   $P(x)$ , et  $a_j$  du polynôme dénominateur  $Q(x)$  de degré  $n$ . Voici ces récurrences pour un retard  $d = 1$ , on tiendra compte à la fin du retard  $d$  en utilisant le programme Scilab `horner`.

$$b_{j+1} = -\frac{m-j+1}{j(n+m-j+1)} b_j \quad b_1 = 1 \quad j \in [1, m].$$

$$a_{j+1} = \frac{n-j+1}{j(n+m-j+1)} a_j \quad a_1 = 1 \quad j \in [1, n].$$

Par la même occasion nous proposons, dans le programme trois autres approximants : approximants de Laguerre, de Kautz et de Padé du second ordre dont les transmittances sont respectivement :

$$Lag_n(s) = \frac{(1-\frac{s}{2n})^n}{(1+\frac{s}{2n})^n}, \quad Kautz_n = \frac{(1-\frac{s}{2n}+\frac{1}{2}(\frac{s}{2n})^2)^n}{(1+\frac{s}{2n}+\frac{1}{2}(\frac{s}{2n})^2)^n}, \quad Pade2_n = \frac{(1-\frac{s}{2n}+\frac{1}{3}(\frac{s}{2n})^2)^n}{(1+\frac{s}{2n}+\frac{1}{3}(\frac{s}{2n})^2)^n}.$$



La fonction `padedelay.sci` a pour arguments d'entrée les degrés  $n$  et  $m$  ainsi que le système  $hd$  à entrée-sortie retardée de  $d$ . En plus la présence d'un drapeau `flag`, à la place de `m` permet de choisir un approximant de Laguerre, de Kautz ou Padé du second ordre plutôt que de Padé.

La sortie de cette fonction renvoie le **produit de l'approximant de Padé du retard pur et du système originel non retardé** : cette fonction sait traiter une matrice de systèmes.

La syntaxe de cette fonction est : `hpade = padedelay(hd, np[, mp])`  
ou `padedelay(hd,np,flag)`

## 8 Le programme « pade.sci » : approximation d'une fonction.

Ce second programme que je propose, est basé sur la méthode classique permettant de trouver un approximant de Padé de toute fonction  $f(s)$  développable en série de Maclaurin.

Ce programme doit certainement être amélioré.

On se donne le développement de Maclaurin sous forme de vecteur ligne de dimension au moins  $m + n + 1$  : noté  $(d_0 \ d_1 \ d_2 \ \dots \ d_{m+n})$ .

Le polynôme correspondant a pour expression :

$D(s) = d_0 + d_1s + d_2s^2 + \dots + d_{n+m}s^{m+n}$  que l'on cherchera à évaluer au rapport de deux polynômes  $\frac{P(s)}{Q(s)}$  avec  $P(s)$  de degré  $m$  et  $Q(s)$  de degré  $n$ .

$P(s) = b_0 + b_1s + b_2s^2 + \dots + b_ms^m$  et  $Q(s) = 1 + a_1s + a_2s^2 + \dots + a_ns^n$  (on peut prendre  $a_0 = 1$  sans aucune ambiguïté).

Cette égalité conduit à traiter par blocs l'égalité matricielle suivante :

$D(s)Q(s) = P(s) + R(s)$  avec  $R(s) \simeq 0$  en égalant les termes de même degré.

On a donc les relations suivantes, pour représenter l'équation (2) j'ai pris  $m = n - 2$  :

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ \vdots \\ b_m \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ d_0 & 0 & 0 & \dots & 0 & 0 \\ d_1 & d_0 & \cdot & \dots & 0 & 0 \\ d_2 & d_1 & d_0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ d_{m-1} & d_{m-2} & d_{m-3} & \dots & d_0 & 0 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ \vdots \\ a_n \end{pmatrix} + \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ \vdots \\ \vdots \\ d_m \end{pmatrix} \quad (1)$$

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} d_m & d_{m-1} & \cdot & \cdot & d_0 & 0 \\ d_{m+1} & d_m & d_{m-1} & \cdot & \cdot & d_0 \\ d_{m+2} & d_{m+1} & d_m & d_{m-1} & \cdot & d_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & d_{m-1} \\ d_{m+n-1} & d_{m+n-2} & \cdot & d_{m+2} & d_{m+1} & d_m \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ \vdots \\ a_n \end{pmatrix} + \begin{pmatrix} d_{m+1} \\ d_{m+2} \\ d_{m+3} \\ \vdots \\ \vdots \\ d_{m+n} \end{pmatrix} \quad (2)$$

A partir de l'équation (2) on peut normalement trouver le vecteur  $(a_1 \ a_2 \ a_3 \ \dots \ a_n)$  puis reporter dans l'équation (1) et ainsi résoudre le problème. Seulement, dans l'équation (2) la matrice  $(n, n)$  est de Toeplitz et peut être **très mal conditionnée**, il faut donc trouver un algorithme général robuste pour résoudre ce problème, ou un algorithme d'inversion d'une matrice de Toeplitz lui aussi très robuste.

Après quelques expériences (?) et la lecture de nombreux articles récents (en particulier l'article de P. Gonnet, S. Guetel, L.N. Trefethen octobre 2011, ou l'article « Algebraic properties of robust Padé approximants » de B. Beckermann et A.C. Matos) il semble que la meilleure solution consiste à « normaliser » le vecteur des données  $(d_0 \ d_1 \ d_2 \ \dots \ d_{m+n})$  et d'inverser la matrice de Toeplitz (équation 2) par l'instruction Scilab `pinv` qui utilise la décomposition d'une matrice en valeurs singulières.

Quant au programme `pade.sci` lui même, il prend pour arguments d'entrée le degré  $m$  du numérateur  $P(s)$ , puis le degré du dénominateur  $n$  de  $Q(s)$  ainsi que le vecteur du développement  $D$  de Maclaurin, (ou la fonction à approximer ou la fonction sous forme d'une chaîne de caractères) et renvoie les vecteurs en ligne des coefficients du numérateur et dénominateur de la fraction rationnelle approximant de Padé.

Remarque : On trouve bien comme limite un `padé(7,7)` pour  $e^{-x}$  avec les mêmes coefficients que ceux trouvés par le programme `padedelay.sci`.

La syntaxe de cette fonction est : `[num, den] = pade(m, n, d)`

## 9 Le programme « `thiran.sci` » : approximation d'un retard pur en échantillonné.

Quand on cherche à faire l'étude des systèmes à retard, ou que l'on approxime les systèmes continus à retard par des systèmes échantillonnés on est amené à trouver une fraction rationnelle de la variable  $z$  pouvant décrire correctement le retard pur. La méthode proposée consiste à trouver un filtre numérique dont le gain est constant quelque soit la fréquence. Un des meilleurs filtres est le filtre de Thiran. Nous l'avons réalisé par le programme `thiran.sci` dont voici le principe .

Ce filtre est un filtre passe tout dont la transmittance est  $H(z) = \frac{D(1/z)}{D(z)} z^{-m}$ , ce filtre se calcule facilement par récurrence, en effet si  $D(z) = 1 + \sum_{n=1}^m d_n z^{-n}$  alors  $d_n = (-1)^n \binom{m}{n} \frac{(D1-m)_n}{(D1-1)_n}$  avec  $\binom{m}{n}$  le coefficient binomial et  $(D1-m)_n = (D1-m)(D1-m+1)\dots(D1-m+n-1)$ .

Cette récurrence donne :

$$d_{n+1} = d_n \frac{(m-n)(m-n-D1)}{(n+1)(n+1+D1)}$$

$D1$  étant le retard ramené à la période d'échantillonnage, ( $D1 = D/tsam$ ),  $d_0 = 1$  et  $0 \leq n \leq m-1$ ,  $tsam$  est la période d'échantillonnage.

J'ai profiter de ce programme pour proposer deux solutions : la première est l'algorithme décrit par les équations précédentes : on a suivant le retard un filtre avec de nombreux pôles et zéros, dépendants de la valeur du rapport retard sur période

d'échantillonnage. Avec la deuxième solution on recherche l'entier inférieur  $d1$  dans le rapport précédent, ce qui donne dans la transmittance un facteur en  $1/z^{d1}$ , le reste du retard étant alors inférieur à la durée d'échantillonnage, sera intégré au numérateur sous la forme  $1 + \frac{1-d_r}{1+d_r}z$ ,  $d_r$  étant le reste du retard ; le dénominateur quant à lui vaut :  $z^{d1}(\frac{1-d_r}{1+d_r} + z)$ . Ce choix se fera à l'aide d'un drapeau **flag** valant "yes" si on utilise la deuxième solution. Les deux autres entrées étant le retard et la période d'échantillonnage, la sortie de la fonction **thiran.sci** étant un système échantillonné approximant le retard.

La syntaxe de cette fonction est : `hz = thiran(d, dom[, flag])`

## 10 Le programme « m\_margin.sci » : marge de module des systèmes sans retard.

Lors de la synthèse par les méthodes fréquentielles on définit les marges de stabilité, **g\_margin**, **p\_margin** dans Scilab, mais quand on recherche une commande robuste, on est amené à calculer la « marge de module ou marge de gain-phase », distance minimale du lieu de Nyquist par rapport au point  $-1$ . Dans le cours que je donne on verra la définition exacte. Voici le principe que j'utilise pour calculer cette distance minimale.

Si  $h(s)$  est la transmittance de la boucle ouverte, le système ayant été ramené à un système à retour unitaire, la fonction  $si = \frac{1}{1+h(s)}$  est la fonction de sensibilité et  $invs_i = \frac{1}{si} = 1 + h(s)$  son inverse. Il suffit donc de rechercher la fréquence de résonance de  $si$  puis de calculer la réponse à cette fréquence de  $1 + h(s)$  donc de l'inverse de la fonction de sensibilité. Puis on calcule le **module** (et pas le module en db) de cette réponse particulière noté *mgp*. Le programme donne cette valeur de *mgp* et la fréquence correspondante *fmgp*.

La syntaxe de cette fonction est : `[mgp, fmgp] = m_margin(h)`

## 11 Le programme « g\_marginrd » : marge de gain des systèmes avec retard.

Ce programme qui est une copie presque conforme du programme scilab **g\_margin.sci** et sait traiter les systèmes à retard pur en entrée-sortie.

On commence par un changement de variable sur  $s$  pour se ramener à un retard pur égal à 1. Pour ce faire on fait le changement de variable  $s \rightarrow \frac{s}{d}$  : ( $d$  est le retard) ; par ce changement les lieux de Black et Nyquist restent identiques malgré ce changement de variable, seule la graduation du lieu change : j'ai normalisé la transmittance du système retardé.

Comme pour un système sans retard le problème consiste à trouver les racines (les pulsations) d'une équation qui n'est plus polynômiale mais à la forme :

$$Equa = B(w)\cos(w) - A(w)\sin(w) = 0$$

avec  $A(w)$  et  $B(w)$  deux polynômes en  $w$  :  $A(w)$  et  $B(w)$  sont respectivement les parties réelle et imaginaire de l'expression :

```
AA = horner(h.num,%i*w)*conj(horner(h.den,%i*w)).
```

Afin de trouver les racines nous initialisons en cherchant un approximant de Padé du quatrième ordre du retard, ainsi le résultat reviendra à trouver les racines d'un polynôme en  $w$ . Après avoir choisi la racine réelle la plus probable, (la plus probable est proche de la plus petite racine de l'approximation de Padé). On améliore la valeur de cette racine en utilisant l'algorithme de recherche d'un zéro d'une fonction : méthode de Newton-Raphson.

La syntaxe de cette fonction est : `[gm, fgm] = g_marginrrd(hd)`

## 12 Le programme « p\_marginrd » : marge de phase des systèmes avec retard.

La aussi c'est un programme qui ressemble étrangement au programme `p_margin.sci` de scilab, je le propose en remarquant que les lieux de Black du système avec et sans retard, se déduisent l'un de l'autre par une translation de la phase, variable, horizontale, de valeur  $-dw$  : ceci reste valable bien sûr pour la pulsation de coupure à  $0db$  : on a juste à rajouter l'expression  $-dw$  dans une ligne de programme de `p_margin.sci`. Si l'on nomme  $hd$  la transmittance du système avec retard,  $h$  celle du même système sans retard, on a pour la recherche des fréquences où le module vaut 1 :  $|hd(jw)| = |h(jw)||e^{-jw}| = 1$ .

La syntaxe de cette fonction est : `[phm, fr] = p_marginrrd(hd)`

## 13 Le programme « m\_marginrd » : marge de module des systèmes avec retard.

Contrairement au programme `m_margin`, pour un système avec retard en entrée/sortie, il est bien plus difficile de rechercher la marge de module de ce système : en effet il faudrait en toute rigueur calculer la fonction de sensibilité et son inverse pour trouver la marge de module.

Ainsi nous allons déterminer la distance minimale (trouver la pulsation sur la réponse fréquentielle) au point  $[-1, 0]$ .

Pour ce faire il faut trouver la pulsation  $\omega_0$  sur le lieu de Nyquist, pulsation telle que la normale au lieu en ce point, passe par le point  $[-1, 0]$ , puis calculer une distance.

On va donc procéder en deux étapes :

1. On détermine grossièrement une pulsation  $\omega_i$  proche de la pulsation recherchée, en assimilant le système étudié à un système classique en utilisant un approximant de Padé du quatrième ordre.
2. Enfin en choisissant une précision donnée, nous allons déterminer un point (donc une valeur à  $\omega$ ) sur le lieu de Nyquist donnant la distance minimale au point  $[-1, 0]$ .

La syntaxe de cette fonction est : `[mgp, fmgp] = m_marginrrd(hd[,f1,f2])`

## 14 Le programme « dscr\_sisord » : discrétisation d'un système à retard en entrée.

Si le retard est multiple de la période d'échantillonnage ( $d = kT_s$ ), alors la transmittance en  $z$  du système retardé vaut  $Gdelay(z) = z^{-k}G(z)$  si  $G(z)$  est la transmittance du système sans retard.

Dans le cas général, on aura  $d = kT_s + \alpha T_s$ , avec  $k$  le plus grand entier positif inférieur à  $d$  et  $0 \leq \alpha < 1$  : on recherchera la transmittance échantillonnée pour un retard de  $\alpha T_s$  notée  $G_\alpha(z)$  et l'on aura  $Gdelay(z) = z^{-k}G_\alpha(z)$ .

### Principe de calcul de $G_\alpha(z)$

Retarder le signal d'entrée du retard  $\alpha T_s$  revient à appliquer entre les instants  $nT_s$  et  $(n+1)T_s$  un signal constitué de deux paliers successifs, de  $nT_s$  à  $(n+\alpha)T_s$  le signal  $u_1(t) = u((n-1)T_s)$ , puis de  $(n+\alpha)T_s$  à  $(n+1)T_s$  le signal  $u_2(t) = u(nT_s)$ .

On va donc intégrer successivement les équations d'état du système continu avec comme signal d'entrée les deux paliers,  $u_1(t)$  puis  $u_2(t)$ .

Sur l'intervalle de temps de  $nT_s$  à  $(n+1)T_s$  le signal  $u_1(t)$  vaut  $u((n-1)T_s)$  de  $nT_s$  à  $(n+\alpha)T_s$ , puis 0(t) de  $(n+\alpha)T_s$  à  $(n+1)T_s$ . Quant au signal  $u_2(t)$  il vaut 0(t) de  $nT_s$  à  $(n+\alpha)T_s$  puis  $u(nT_s)$  de  $(n+\alpha)T_s$  à  $(n+1)T_s$ .

D'après ce raisonnement, on aura un système continu échantillonné et bloqué ayant une sortie  $y(t)$  et deux entrées  $[u_1(t), u_2(t)]$  que l'on intégrera sur l'intervalle de temps  $T_s$ ; remarquons que  $Z(u_1(t)) = k \frac{Z(u_2(t))}{z}$ .

### Calcul

Notons respectivement  $x(nT_s) = x_n$ ,  $x((n+1)T_s) = x_{n+1}$ ,  $x((n+\alpha)T_s) = x_{n+\alpha}$  on a :

$$x_{n+1} = e^{AT_s}x_n + \int_{nT_s}^{(n+1)T_s} e^{A(T_s-\mu)}Bu(\mu)d\mu$$

Ici l'entrée  $u(t)$  est constituée de deux paliers successifs  $[u_1(t), u_2(t)]$ , le premier terme de la somme étant la contribution à l'instant  $(n+1)T_s$  du vecteur d'état initial (instant  $nT_s$ ), le second terme sera constitué d'une somme qui dépendra linéairement de  $u_{n-1}$  et de  $u_n$ . Notons  $\Phi(T_s) = e^{AT_s}$ ,  $\Phi(\theta T_s) = e^{A(\theta T_s)}$ ,  $\Psi(T_s) = (\int_0^{T_s} e^{A\mu}d\mu)B$ ,  $\Psi(\theta T_s) = (\int_0^{\theta T_s} e^{A\mu}d\mu)B$  et comme  $\Phi(T_s) = e^{AT_s} = e^{A(1-\alpha)T_s}e^{A\alpha T_s}$ , (voir le paragraphe 9.2.2), avec ces notations et en intégrant sur ces deux intervalles de temps, le premier  $\alpha T_s$ , le second  $(1-\alpha)T_s$  nous avons :

$$x_{n+1} = \Phi(T_s)x_n + \Phi((1-\alpha)T_s)\Psi(\alpha T_s)u_{n-1} + \Psi((1-\alpha)T_s)u_n$$

Cette équation peut être représentée matriciellement (comme pour les systèmes sans retard) par l'équation :

$$\begin{bmatrix} x_{n+1} \\ u_n \end{bmatrix} = \begin{bmatrix} \Phi(T_s) & \Phi((1-\alpha)T_s)\Psi(\alpha T_s) \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_n \\ u_{n-1} \end{bmatrix} + \begin{bmatrix} \Psi((1-\alpha)T_s) \\ I \end{bmatrix} u_n$$

Maintenant prenons la transformée en  $z$  de la première équation, on a donc :

$$(zI - \Phi(T_s))x(z) = \left[ \frac{1}{z} \Phi((1-\alpha)T_s)\Psi(\alpha T_s) + \Psi((1-\alpha)T_s) \right] u(z)$$

Ainsi pour trouver la transmittance on réalisera les opérations matricielles :

$$G_\alpha(z) = \frac{G_1(z)}{z} + G_2$$

avec  $G_1 = C(zI - \Phi)^{-1}\Phi((1-\alpha)T_s)\Psi(\alpha T_s)$  et  $G_2 = C(zI - \Phi)^{-1}\Psi((1-\alpha)T_s)$ . Le logiciel Scilab permet de calculer facilement les expressions  $\Phi(?T_s)$  et  $\Psi(?T_s)$ , il suffit donc d'utiliser une partie de la macro `dscr.sci` sur différentes durées. Cette macro donne directement la transmittance du système échantillonné en utilisant les polynômes caractéristiques.

La syntaxe de cette fonction est : `sl_rd = dscr_sisord(sl,Ts)`

## 15 Le programme « `dpf_cd` » : calcul des gains, des phases, des fréquences pour des systèmes continus ET échantillonnés.

Si l'on veut comparer le comportement fréquentiel des systèmes continus (avec ou sans retard) et leurs homologues échantillonnés (par discrétisation par exemple), on doit pouvoir calculer à la fois les gains, les phases, les fréquences pour deux vecteurs de systèmes différents, à savoir les systèmes continus et les systèmes échantillonnés. On construit donc trois matrices de même dimensions donnant les fréquences, les gains, les phases pour deux types de systèmes et avec ces matrices on pourra tracer, entre autre, les différents lieux fréquentiels.

### Principe de l'algorithme :

On se donne deux vecteurs colonne de systèmes ; d'une part les systèmes continus (**avec** ou **sans retard**) `gc` et d'autre part les systèmes échantillonnés `gd`. Puis on se donne **la gamme** de fréquences `[fmin, fmax]` dans laquelle on va étudier ces systèmes (les continus et les échantillonnés) ou le vecteur fréquence sous la forme `[fmin:pas:fmax]` ou encore le vecteur fréquence `[f1,f2,...fn]`. Dans le premier cas avec les instructions `calfrq` ou `calfrqrd` on va discrétiser la gamme de fréquences en prenant les systèmes continus comme référence. Dans tous les cas on recherchera la fréquence de Nyquist des systèmes échantillonnés et on réalisera une homothétie sur les fréquences nécessaires au tracé des lieux échantillonnés pour que l'on ait un vecteur fréquence, puis une matrice fréquence, constituée de deux blocs, le premier pour les systèmes continus, le second pour

les systèmes échantillonnés et avec les vecteurs constituant cette matrice, on calculera les deux matrices de gain et phase à l'aide de l'instruction `dbphifr` : enfin avec ces trois matrices on peut maintenant facilement tracer les divers lieux.

La syntaxe de cette fonction est : `[fd,dd,pd] = dpf_cd(gc,gd,fmin[,fmax[,pas]])`

## 16 Le programme « `fdelay` » : retarder une matrice de données.

Cette fonction a pour but de retarder d'une valeur donnée de temps, un ensemble de signaux quelconques (matrice de données de dimensions  $k \times l$ ), retard scalaire  $d$ . On récupère l'incrément de temps en entrée, on compare cet incrément par rapport au retard, si cet incrément est un sous multiple du retard on décale les données du retard  $d$  : on ne change pas l'incrément. Sinon on accepte une discrétisation du temps en sortie non uniforme : le temps correspondant au retard est conservé et est compris entre deux valeurs de discrétisation.

La syntaxe de cette fonction est : `[fd,tt] = fdelay(f,t,d)`

## 17 Les lieux fréquentiels : `bode`, `black` ...

J'utilise les lieux fréquentiels de Scilab en rajoutant quelques éléments, à savoir des axes, des grilles ..., mais rien ne change sur le principe, à part la possibilité de tracer les pseudo-lieux et surtout les lieux des systèmes à entrée sortie retardés. Pour le tracé des lieux, j'utilise exclusivement la fonction `dbphifr.sci` ou trois matrices en entrée, à savoir `[fr,db,phi]` (explication au paragraphe 4).

## 18 Les fonctions commençant par %

Ces fonctions sont adaptées à la boîte à outils `iodelay toolbox`, et sont un complément à cette boîte à outils.