PROPOSITIONS POUR « améliorer » la fonction « freson.sci ».

Pour exécuter les instructions qui suivent, veuillez charger les trois programmes : « ffreson.sci », « hps_hms.sci » et « hpz_hiz.sci ». Les deux derniers sont aussi dans le programme « ffreson.sci ».

Problème posé :

On recherche la ou les fréquences de résonance de systèmes continu et échantillonné.
Ce problème consiste à rechercher les fréquences strictement positives conduisant à des extrema (maxima) du gain du système considéré.
On doit donc, si « h(s) » est la transmittance, calculer les zéros imaginaires purs strictement positifs de la fonction « derivat(h(s)*h(-s)).num » pour un système continu et les zéros différents de 1 (fréquence zéro) ou -1 (fréquence de Nyquist) de la fonction « derivat(h(z)*h(1/z)).num » pour un système discret ou échantillonné. De plus dans ce dernier cas les solutions possibles sont obligatoirement de la forme exp(%i*2*%pi*fp*Ts) (Ts période d'échantillonnage), donc complexes de module égal à 1.

UN EXEMPLE COMMENTE DE SYSTEME CONTINU QUI POSE ENCORE PROBLEME.

```
--> s = %s;num = s+1.4*s^2+1.4*s^3+0.4*s^4
 num  =

          2     3     4
  s +1.4s  +1.4s  +0.4s

--> den = 0.5+0.8*s+1.4*s^2+1.4*s^3+0.4*s^4

 den  =

                2     3     4
  0.5 +0.8s +1.4s  +1.4s  +0.4s

 --> h = syslin("c",num,den)//Le système

h  =

             2     3     4
      s + 1.4s + 1.4s + 0.4s
  -------------------------------
                2     3     4
  0.5 + 0.8s + 1.4s + 1.4s + 0.4s
```

REMARQUES :
On recherche les racines « si » de la fonction : « derivat(h(s)*h(-s)).num »; les valeurs recherchées ont la forme ± %i*abs(si).

PREMIERE METHODE (brutale). UTILISATION DE LA FONCTION « hps_hms.sci ».
Algorithme « hps_hms.sci » : calcul de hh(s)=h(s)*h(-s) sans passer par « horner.sci » : on utilise la **convolution**.

Cette fonction donne **obligatoirement** un résultat **pair** pour « h(s)*h(-s) ».

On fait d'abord « h*horner(h,-s) » pour comparer avec « hps_hms.sci ».

1 : Actuellement on utilise la fonction « horner » :

```
 --> format(15) ;
 --> hhh = h*horner(h,-s)
 hhh  =
    2           3      4           5      6           7      8
  -s + 2.220D-16s - 0.84s - 1.110D-16s - 0.84s - 1.110D-16s + 0.16s
  ---------------------------------------------------------------------
              2           3      4      6           7      8
   0.25 + 0.76s + 1.110D-16s + 0.12s - 0.84s - 1.110D-16s + 0.16s
```

On n’a pas un rationnel pair : termes très petits au numérateur et au dénominateur de degré impair.

```
 --> derh=derivat(hhh).num
 derh  =

                2       3           4         5           6         7           8         9           10
 -0.5s +1.665D-16s  -0.84s  +1.410D-16s  -2.2968s  -5.673D-16s  -5.5936s  -7.105D-16s  +0.0768s  -5.018D-16s
                  11          12
          +0.6144s   +5.329D-17s
```

le polynôme derh n’est pas un polynôme impair et a des petits termes pairs dont le dernier qui influence le nombre de racines.

2 : méthode proposé : la fonction « hps_hms » :

```
 --> hh=hps_hms(h)//vecteur ligne hh = [numérateur,dénominateur]

 hh  =

   2       4       6       8              2       4       6       8
 -s  -0.84s  -0.84s  +0.16s    0.25 +0.76s  +0.12s  -0.84s  +0.16s

 --> der= hh(2)*derivat(hh(1))-hh(1)*derivat(hh(2))//La dérivée.
 der  =


             3         5         7         9         11
 -0.5s -0.84s  -2.2968s  -5.5936s  +0.0768s  +0.6144s
```

Dans la nouvelle fonction « ffreson » on a pour « der » 11 racines (numérateur degré 11).

```
--> r=roots(der) //11 racines dont r1=0, et si rp est racine -rp l’est aussi.

   r  =

  -1.780444761511
   1.780444761511
   1.697197171247i
  -1.697197171247i
  -0.48171214486 + 0.44200935027i
  -0.48171214486 - 0.44200935027i
   0.48171214486 + 0.44200935027i
   0.48171214486 - 0.44200935027i
   0.698465432403i
  -0.698465432403i
   0.
```

Les racines à conserver sont en positions 3 et 9 (elles sont imaginaires pures positives).
Dans la fonction « freson » actuelle on a : 12 racines dont s=0.Le polynôme derh est de degré 12.

```
---> rh=roots(derivat(h*horner(h,-s)).num)
```

12 racines dont une très grande du à la valeur très faible du dernier terme de « derh ».

```
 rh  =

  -1.153D+16
  -1.7804448
   1.7804433
   0.0000008 + 1.6971979i
   0.0000008 - 1.6971979i
  -0.481713 + 0.4420075i
  -0.481713 - 0.4420075i
   0.481713 + 0.4420075i
   0.481713 - 0.4420075i
  -1.318D-11 + 0.6984654i
  -1.318D-11 - 0.6984654i
   0.
```

Normalement les racines à garder sont en positions 4 et 10 (elles devraient être

imaginaires pures) : elles correspondent à un max et un min.

```
 --> gainplot(h,0.01,10)
 --> freson(h)

   ans  =

        []
```

On ne trouve aucune solution à cause du nombre de racines, de la précision sur le calcul des racines et du seuil choisi (1.e-14), pour « freson.sci » de scilab-6.0.2.

```
--> ffreson(h)//La nouvelle fonction.
 ans  =

   0.1111642
```

Cette fonction utilise l'algorithme « hps_hms.sci » : calcul de hh(s)=h(s)*h(-s) par convolution.

REMARQUES : Le numérateur de la dérivée de hh(s) est une fonction impaire (s=0 est une racine évidente).
Si on élimine la racine s=0, (non utile pour la suite du problème et correspondant à la fréquence nulle), le reste sera une fonction paire.
On va donc construire un polynôme en $u=s^2$, rechercher les racines de ce polynôme, puis les racines carrées des valeurs trouvées de ces racines. Les racines imaginaires pures trouvées conduisent aux maxima et minima recherchés par la fonction « ffreson.sci ».

3 : AMELIORATION : Utilisation de « hps_hms.sci » et division par deux du degré du polynôme dont on recherche les racines, pour améliorer encore la précision sur le calcul des racines :rechercher les racines d'un polynôme de degré 5 au lieu de 10 ,et éliminer la racine évidente s = 0.
On travaille sur le polynôme numérateur de la dérivée de hh(s).

```
--> der=derivat(hh).num
 der  =

--> coe=coeff(der)
 coe  =

   0.  -0.5   0.  -0.84   0.  -2.2968   0.  -5.5936   0.   0.0768   0.   0.6144

--> coe(1:2:$)=[]

 coe  =

  -0.5  -0.84  -2.2968  -5.5936   0.0768   0.6144
```

Elimination de la racine s = 0 et des coefficients d'ordre impair de « coe » qui doivent être théoriquement tous nuls. Puis on construit un nouveau polynôme « DER ».

```
--> DER=poly(coe,"s","c")//Le polynôme en u=s² -->s=u.
 DER  =

                     2        3        4        5
   -0.5 -0.84s -2.2968s  -5.5936s  +0.0768s  +0.6144s

--> roots(DER)//racines en u=s² (5 racines).
 ans  =

   3.1699835
  -2.8804782
  -0.487854
```

```
  0.0366743 + 0.4258425i
  0.0366743 - 0.4258425i
```

```
 -> rac=sqrt(ans)//Les racines recherchées (voir manuel sqrt).
 rac  =
```

```
  1.7804448
  1.6971972i
  0.6984654i
  0.4817121 + 0.4420094i
  0.4817121 – 0.4420094i
```

Le vecteur « r » calculé précédemment vaut théoriquement : [rac;-rac;0].
Les racines recherchées qui doivent être imaginaires pures positives sont en position 2 et 3 dans le vecteur « rac » : un maximum et un minimum.
Comparons maintenant les résultats donnés par « freson » et « ffreson ».

```
--> freson(h)
 ans  =

    []

--> ffreson(h)
 ans  =

   0.111164226146

--> gainplot(h,0.001,1)
```

C'est avec cette méthode que je "corrige" la fonction « freson.sci », pour les systèmes continus. Le reste du programme consiste en la séparation des minima et des maxima et une mise en forme des résultats.

****
****

UN EXEMPLE DE SYSTEME ECHANTILLONNE QUI POSE PROBLEME.

Prenons l'exemple proposé dans la page du manuel relative à la fonction « freson.sci ».

REMARQUES :
Ici on doit trouver le racines de « der = derivat(hd*hd(1/z).num) » .
Dans l’exemple de la page du manuel, si la période d'échantillonnage vaut 0.04s alors on trouve une solution avec l’actuelle fonction « freson.sci », mais pas pour une période de 0,01s, les coefficients du numérateur de la transmittance sont très petits ce qui aura des conséquences sur le calcul des racines du numérateur de la dérivée de « h(z*h(1/z) ».

```
-->   s=%s;z=%z; h=syslin("c",s-1,(3+2*s+s*s)*(50+0.1*s+s*s)); format(15);

-->   hd=ss2tf(dscr(h,0.01))//Période 0,01s.

 hd  =
                                                2                3
    -0.00000016443 - 0.000000499867z + 0.000000489069z + 0.000000165336z
   ------------------------------------------------------------------------

                                                2                3   4
   0.979218964569 - 3.932443883701z + 5.92713314461z - 3.973906741777z + z

--> gainplot(hd,0.01,10)//pour visualiser la courbe de gain.
```

1 : Faisons un premier calcul avec la fonction « horner.sci ».

```
--> hdh=hd*horner(hd,1/z)
 hdh  =

                                          2               3               4               5
          -2.77631435D-14z - 2.22053297D-13z - 4.99509210D-13z - 2.22053297D-13z - 2.77631435D-14z
  ------------------------------------------------------------------------------------------------------------
                                                 2               3               4               5   6
  1.000000000265 - 5.989705152491z + 14.95884870591z - 19.93828709833z + 14.95884870525z - 5.989705151853z + z
```

On remarque bien que les coefficients des polynômes numérateur et dénominateur ne sont pas parfaitement symétriques.

```
--> coeden = coeff(hdh.den)
 coeden  =

   1.000000000265  -5.989705152491   14.95884870591  -19.93828709833   14.95884870525  -5.989705151853   1
```

```
coeden(1)#coeden($),...etc
```

Calculons le numérateur de la dérivée de hdh :

```
--> derh=derivat(hdh).num
derh  =

                                                 2               3               4               5
  -2.77631435D-14 -4.44106595D-13z +2.46810811D-13z  +0.000000000004z  -0.000000000007z  -6.32925647D-22z
                          6               7               8               9               10
          +0.000000000007z  -0.000000000004z  -2.46810812D-13z  +4.44106595D-13z  +2.77631435D-14z
```

La fonction « freson » actuelle calcule les racines de « derh ».

2 : Maintenant utilisons la fonction « hpz_hiz.sci » qui permet le calcul de « h(z)*h(1/z) » par convolution en respectant la symétrie des coefficients.

```
--> hdi =hpz_hiz(hd)
 hdi  =



       column 1
                                                 2               3               4               5
  -2.71861967D-14 -1.63063688D-13z -8.14157447D-14z  +5.43429097D-13z  -8.14157447D-14z  -1.63063688D-13z

                          6
          -2.71861967D-14z


        column 2
                                                2               3               4               5
  0.979218964569 -7.82376872868z +27.35825958644z  -54.68662321499z  +68.34582678531z  -54.68662321499z
                          6               7               8
          +27.35825958644z  -7.82376872868z  +0.979218964569z
```

L’algorithme respecte bien la symétrie.
Continuons le programme en tenant compte du terme « z^(degree(hd.den)-degree(hd.num)) »

```
--> hdi(1) = z^(degree(hd.den) – degree(hd.num))*hdi(1)
```

Puis,

```
--> der = hdi(2)*derivat(hdi(1)) – hdi(1)*derivat(hdi(2))
 der  =
                                                 2               3               4               5
  -2.66212394D-14 -3.19350111D-13z +0.000000000002z  +4.29057637D-13z  -0.000000000019z  +0.000000000048z
                          6               7               8               :   9               10
          -0.000000000048z  -2.58493941D-26z  +0.000000000048z  -0.000000000048z  +0.000000000019z
                         11              12              13              14
          -4.29057637D-13z   -0.000000000002z   +3.19350111D-13z   +2.66212394D-14z
```

Il y a une différence notable entre les deux polynômes « der » et « derh ». En particulier les degrés ne sont pas les mêmes (« derh » par « horner » est de degré 10, et « der » par « hpz_hiz.sci » est de degré 14).

A priori on ne trouvera pas les mêmes racines.
Malgré tout on va comparer les racines de « der » et « derh » :

```
--> ri=roots(der)
 ri  =

  -15.92773138497
  -3.730850636312
  -1.
   0.997510662761 + 0.070646339074i
   0.997510662761 - 0.070646339074i
   0.998908263207 + 0.042262213764i
   0.998908263207 - 0.042262213764i
   1.021089272931
   1.002378419766 + 0.017264731997i
   1.002378419766 - 0.017264731997i
   0.992752965805
   0.981899303401
  -0.26803538857
  -0.062783580149
```

Ici on a trois couples de racines imaginaires : c'est bon signe : deux max et un min entre les deux.

```
--> rh=roots(derh)

 rh  =

  -15.92773138497
  -3.730850636312
  -1.000000000002
   0.997508866847 + 0.0705538271i
   0.997508866847 - 0.0705538271i
   1.000003050142
   0.99905772408 + 0.043345496778i
   0.99905772408 - 0.043345496778i
  -0.26803538854
  -0.06278358014
```

Ici pas de chance on n'a que deux couples possibles.

Comme le montre la courbe de gain,on a deux maxima et un minimum entre les deux (positions 4,6,9 dans le vecteur « ri »).
Comme je l'ai dit précédemment,on ne conservera que les racines imaginaires positives de modules proches de 1 (ici problème de précision!). Puis on isolera les maxima des minima.
Les instructions pour réaliser cela sont évidentes. C'est à la fin du programme « ffreson.sci ».
Comparons les résultats avec la fonction « freson » (scilab-6.0.2) et la nouvelle fonction nommée « ffreson ».

```
--> freson(hd)
 ans  =

    []

--> ffreson(hd)//sans avoir éliminé z = 1 et z = -1.
 ans  =

   1.125298400643
   0.274097657806
```

Puis faire :

```
--> gainplot(hd,0,1,10)
```

3 : **Améliorations :**

Comme je l'ai signalé précédemment, les deux racines z = 1 et -1 qui correspondent aux fréquences f0 = 0 et f1 = 1/(2*Ts) (fréquence de Nyquist) sont à éliminer, ce que l'on fait directement en divisant le polynôme « der » par le terme (z-1)*(z+1), puis on recherche toutes les racines du polynôme modifié : on diminue de 2 le degré du polynôme et améliore ainsi la précision dans la recherche de ses racines.

```
-->der = der/(z*z-1) ;//ou :
-->[quo,rest] = pdiv(der,z*z-1) ;//puis faire un test, si vous le souhaitez, sur
les coefficients de « rest » qui sont ici de l’ordre de 1.e-26.
-->r = roots(quo) ;//Etc ...

Avec le programme « ffreson » amélioré on obtient :
--> fr=ffreson(hd)
 fr  =

   1.12537583190641444
   0.2687659884318471
```

Je reconnais que les exemples choisis, en particulier pour le modèle échantillonné, sont « limites » à cause de la petitesse des coefficients du numérateur de la transmittance.

Merci de me lire

lucien.povy@free.fr