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

1 Principe, initialisation de l'algorithme

J'ai cherché avec ce programme à déterminer les multiplicités des racines d'un polynôme à coefficients réels $P(x)$ de degré n, et à initialiser les valeurs de ses racines en utilisant les programmes `roots.sci` et `bezout.sci` proposés 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, en c, ou c++) pour la rapidité d'exécution.
2. **Résultats de la théorie**.
   On connait le résultat suivant : quand un polynôme $P(x)$ a des racines multiples, vecteur $X_K = [x_1, \cdots, x_k, \cdots, x_K]$ de multiplicité $M_K = [m_1, \cdots, m_k, \cdots, m_K]$, si x_k est l'une de ses 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 :
   $$g(x) = pgcd(P(x), \frac{dP(x)}{dx}) = \prod_{k=1}^{k=K}(x - x_k)^{m_k - 1}$$
   $$U(x) = \frac{P(x)}{g(x)} = \prod_{k=1}^{k=K}(x - x_k) \; et \; V(x) = \frac{\frac{dP(x)}{dx}}{g(x)}$$
   $U(x)$ est la décomposition sans carré (voir l'algorithme de Yun ou de Tobey-Horowitz) de $P(x)$, les racines de $U(x)$ sont alors des racines simples.
3. 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 $\frac{\frac{dP(x)}{dx}}{P(x)}$.

4. En analysant l'équation de Bezout donnée dans Scilab, `[g,U] = bezout(P,dP)` où `P` est le polynôme unitaire de départ et `dP` sa dérivée première, on constate facilement que le terme `U(2,2) = - P/g = - Ux` et `U(1,2) = dP/g = Vx`, 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)` par l'intermédiaire de $R(x) = \frac{V(x)}{U(x)} = \sum_{k=1}^{k=K} \frac{m_k}{x-x_k}$. C'est donc à partir de ces considérations que je propose **l'initialisation** du programme Scilab `mroots.sci`.
   Remarque : On peut aussi retrouver les multiplicités à partir de $R(x)$ directement : si nous mettons sous forme d'éléments simples $R(x)$ en utilisant le programme Scilab `pfss(Vx/Ux)` on trouve les coefficients m_k dans les numérateurs des éléments simples.

5. En appliquant 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, polynôme de degré 20, (voir aide sur le programme `roots.sci`), on obtenait avec ma première version du 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).

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 : Comme je viens de le dire si le terme $g(x)$, pgcd 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.sci` seulement.

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.

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. Prenons un exemple où le polynôme P est constitué des facteurs : $s^3, (s-2)^3, (s+1), (s^2+s+1)^4, (s+2)^4$, $(s+6)$, $(s^2+1.5s+1)^3$, $(s^2+0.9s+0.5)^4$, le degré de ce polynôme est 34 et en utilisant l'initialisation de mon programme, on retrouve bien les multiplicités et les racines sont calculées avec une erreur de l'ordre de 10^{-4}. Avec le programme Scilab `nearly_multiples.sci` avec une précision de 0.1 on trouve assez bien des racines réelles, quant aux racines imaginaires, (non pures) certaines peuvent encore être regroupées : cela dépend beaucoup de la précision.

Le programme de comparaison.

```
--> s=%s; T=[s^3,(s-2)^3,(s+1),(s^2+s+1)^4,(s+2)^4, (s+6), (s^2+1.5*s+1)^3,
(s^2+0.9*s+0.5)^4]
```

```
--> P=prod(T)
--> RM=mroots(P,"y")//Initialisation de la solution.
--> [rac,m]=nearly_multiples(roots(P),2.e-1)
//Ce programme est dans ...scilab/modules/cacsd/macros
--> [rac,m]=nearly_multiples(roots(P),1.e-1)
```

2 Amélioration de la solution : recherche des racines, de multiplicité donné, par une méthode des moindres carrés

Dans de nombreux cas, la solution initiale obtenue semble bonne et donne des résultats largements supérieurs à ceux obtenus par le programme `roots.sci` seul (dans le cas de racines multiples). De même le vecteur multiplicité est correct et donne pour les nombreux exemples testés, un résultat nettement plus fiable que les programmes qui cherchent à homogénéiser les racines en groupes (« cluster »), et à faire la moyenne sur chacun de ces groupes : programme `nearly_multiples.sci`, par exemple de Scilab, que l'on peut tester sur l'exemple précédent.

Soit le polynôme $P(x) = \prod_{k=1}^{k=K}(x - x_k)^{m_k}$ unitaire de degré n qui possédent K racines x_k chacune de multiplicité m_k. A ce stade, on a donc le polynôme de degré n, $P(x)$, le vecteur des multiplicités $M_K = [m_1, ..., m_k, ..., m_K]^T$ tel que $\sum_1^K m_k = n$, que l'on considérera comme correct et le vecteur des racines correspondantes $Z = [z_1, ..., z_k, ..., z_K]^T$ **première** initialisation du vecteur des racines. Avec ces racines et ce vecteur multiplicité on construit le polynôme approché $P^*(x) = \prod_{k=1}^{k=K}(x-z_k)^{m_k}$. Si l'on calcule cette expression on récupère les coefficients du polynôme approché qui sont aussi les **fonctions symétriques élémentaires** des racines trouvées. On va ainsi, à partir des expressions des **fonctions symétriques élémentaires** calculées à partir des racines, résoudre un problème d'optimisation : trouver les $K < n$ valeurs des racines (z_k) minimisant l'écart quadratique entre les fonctions symétriques élémentaires réelles (données par les coefficients de $P(x)$) et celles calculées à partir des racines initiales (réelles et/ou complexes conjuguées) et ceci pour un vecteur de multiplicité donné (obtenu lors de l'initialisation).

On trouvera dans la littérature, une publication qui justifie de manière théorique et met en oeuvre ce principe avec le logiciel Matlab (**Auteur Z. ZENG, « COMPUTING MULTIPLE ROOTS OF INEXACT POLYNOMIALS » , Revue « Mathematics of computation »** Volume 74, July 22, 2004) ; je n'expose donc pas l'optimisation des racines par l'utilisation des fonctions symétriques élémentaires. Le programme que je propose utilise les fonctions Scilab programmées suivantes : `bezout.sci`, `roots.sci` et l'optimisation par la **méthode des moindres carrés non linéaire** : (Wikipédia, **Algorithme de Gauss-Newton**) et ne demande pas de faire des macros spécifiques pour initialiser le problème.

Remarques : Quand on programme le problème d'optimisation, on construit un vecteur différence D entre les fonctions symétriques élémentaires calculées à chaque itération et le « vrai » vecteur qui est le vecteur des coefficients du polynôme étudié

(au signe près). Ce vecteur différence est de dimension n, le degré du polynôme de départ, et dépend des K racines, avec $K < n$: c'est une transformation de $\mathbb{C}^{\mathrm{K}} \rightarrow \mathbb{C}^{\mathrm{n}}$. Puis on calcule la matrice jacobienne JcD de D qui est une matrice rectangle (n, K) (car le polynôme de départ a des racines multiples).
Voir `https://en.wikipedia.org/wiki/jacobian_matrix_and_determinant`.

A ce sujet on trouvera sur internet des exercices proposés à l'agrégation de maths, traitant des fonctions symétriques élémentaires et du calcul de la matrice jacobienne de ces fonctions : ces exemples et l'article précédemment cité ont été la source de mon raisonnement, car comme je l'ai signalé au début de mon exposé, je cherche à tout prix à utiliser des fonctions Scilab préprogrammées.

La syntaxe de cette fonction est :

```
[rm,bkerr,pjcnd,fkerr] = mroots(P[,tol[,iter[,flag]]])
```

3 Exemple simple illustrant la méthode proposée

Soit un polynôme de degré 5 possédant une racine triple en -1 et deux racines complexes conjuguées $-\frac{1}{2} \pm \frac{\sqrt{3}}{2} i$. Ce polynôme s'écrit : $p(s) = 1+4s+7s^2+7s^3+4s^4+s^5$.

Réalisons les programmes r=`roots(p)` puis `rinit=mroots(p,"y")` pour obtenir d'une part les racines de ce polynôme par Scilab, et d'autre part les racines et les multiplicités initiales par le programme `mroots(p,"y")`, drapeau `"y"` dans l'instruction `mroots` : on ne fait pas d'optimisation.

```
--> format(20)
--> r=roots(p)
r =
-0.49999999999999978 + 0.86602540378443393i
-0.49999999999999978 - 0.86602540378443393i
-1.00000505953435703 + 0.00000876385524891i
-1.00000505953435703 - 0.00000876385524891i
-0.99998988093129559 + 0.i
```

Nous avons quatre racines complexes et une réelle. Maintenant avec le programme `mroots(p,"y")` nous obtenons :

```
--> rinit=mroots(p,"y")//Le programme mroots sans optimisation.
rinit =
-0.999999999999995 + 0.i                              3. + 0.i
-0.49999999999999989 + 0.86602540378443682i           1. + 0.i
-0.49999999999999989 - 0.86602540378443682i           1. + 0.i
```

Nous trouvons bien cinq racines dont la réelle est de multiplicité trois et deux complexes conjuguées.

Maintenant utilisons le programme `mroots.sci` avec optimisation.

```
--> [rfinal,bkerr,pjcnd,fkerr]=mroots(p,1.e-14,3)
//Trois itérations seulement.
rfinal =
-1.                   + 0.i                             3. + 0.i
-0.50000000000000011 + 0.86602540378443849i             1. + 0.i
```

```
-0.50000000000000011 - 0.86602540378443849i          1. + 0.i
bkerr =
0.
pjcnd =
4.13072440814959396
fkerr =
0.
```

4 Problème plus complexe

Reprenons l'exemple proche du polynôme de la section 1 :

Le polynôme P est de degré 34 et est composé des termes élémentaires $s^3, (s-2)^3, (s+1), (s^2+s+1)^4, (s+2)^4$, $(s+6)$, $(s^2+1.5s+1)^3$, $(s^2+0.9s+0.5)^4$. Pour trouver les racines de P nous mettons le polynôme sous forme de ses coefficients : ce polynôme résultat (inexact) est $P1$ proche de P, puis programmer :

```
--> P1=s^3*(s-2)^3*(s+1)*(s*s+s+1)^4*(s+6)*(s+2)^4*..
(s*s+1.5*s+1)^3*(s*s+0.9*s+0.5)^4 ;
--> RAc=roots(P1)
RAc =
-6. + 0.i
2.0000032 + 0.0000056i
2.0000032 - 0.0000056i
....... ......
```

C'est à peu près n'importe quoi!!

```
--> format(16)
--> RACI=mroots(P1,"y") //Initialisation sans optimisation.
RACI =
0. + 0.i                              3. + 0.i
-5.9999999999998 + 0.i                 1. + 0.i
2.               + 0.i                3. + 0.i
-2.0000000041839 + 0.i                 4.+  0.i
-1.0000229796092 + 0.i                 1. + 0.i
-0.4999999770785 + 0.8660246420269i    4. + 0.i
-0.4999999770785 - 0.8660246420269i    4. + 0.i
-0.7499976077684 + 0.6614453600487i    3. + 0.i
-0.7499976077684 - 0.6614453600487i    3. + 0.i
-0.4500016559398 + 0.5454416934457i    4. + 0.i
-0.4500016559398 - 0.5454416934457i    4. + 0.i
--> [RACFINAL, bkerr, pjcdn, fkerr] = mroots(P1)//Avec optimisation
RACFINAL =
0. + 0.i                              3. + 0.i
-6. + 0.i                              1. + 0.i
2. + 0.i                              3. + 0.i
-2. + 0.i                              4. + 0.i
```

```
-1.0000000000007 + 0.i                 1. + 0.i
-0.5 + 0.8660254037844i                4. + 0.i
-0.5 - 0.8660254037844i                4. + 0.i
-0.7499999999998 + 0.6614378277661i    3. + 0.i
-0.7499999999998 - 0.6614378277661i    3. + 0.i
-0.45 + 0.5454356057318i               4. + 0.i
-0.45 - 0.5454356057318i               4. + 0.i
  bkerr =
  4.436358977D-14
  pjcdn =
  4616.6313404856
  fkerr =
  0.0000000004096
```

Bibliographie :

1. en.wikipedia.org/wiki/jacobian_matrix_and_determinant.
2. Z. Zeng, Computing multiple roots of inexact polynomials, Revue « Mathematics of computation » july 2004.
3. Madina Hasan : The computation of multiple roots of a polynomial using structure preserving matrix methods. PhD thesis Sheffield University England July 2011.
4. S.Gratton, A.S.Lawless, N.K.Nichols, Approximative Gauss-Newton methods for non linear least squares problems. Numerical Analysis report 9/04 .