

SCILAB à l'École nationale des ponts et chaussées

<http://cermics.enpc.fr/scilab>

Calculs des zéros de polynômes

Les réponses

Licence de mathématiques Nouméa 2004

Jean-Philippe CHANCELIER

(avec le soutien du Ministère de l'Outre-Mer)

15 mars 2005 (dernière date de mise à jour)

Table des matières

1	Calcul des valeurs d'un polynôme et de ses dérivées par l'algorithme de Horner	1
1.1	Utilisation de l'algorithme de Horner pour la factorisation	4
1.2	La méthode de Horner backward	6
2	Recherche des zéros d'un polynôme	7

1 Calcul des valeurs d'un polynôme et de ses dérivées par l'algorithme de Horner

Soit $P(x)$ un polynôme de degré n :

$$P(x) \stackrel{\text{def}}{=} \sum_{i=0}^n a_{i+1}x^i$$

On cherche ici à évaluer la valeur du polynôme pour une valeur donnée y . Pour ce faire, on peut factoriser le polynôme $P(x)$ sous la forme :

$$P(x) = Q(x)(x - y) + R \tag{1}$$

où $Q(x)$ est un polynôme de degré $n - 1$ et R une constante scalaire. Sous cette forme on voit que $P(y) = R$ et que $P'(y) = Q(y)$. Le polynôme $Q(x)$ est bien sur différent du polynôme $P'(x)$ mais leur valeurs coïncident quand x prends la valeur y .

Soit b_i les coefficients du polynôme Q :

$$Q(x) \stackrel{\text{def}}{=} \sum_{i=0}^{n-1} b_{i+1} x^i \quad (2)$$

en développant l'équation (1) on obtient facilement une équation récurrente permettant de calculer les coefficients du polynôme Q :

$$b_i = b_{i+1}y + a_{i+1}, \quad b_n = a_{n+1}, \quad \text{pour } i = n-1, n-2, \dots, 0 \quad (3)$$

La valeur de R est donnée par b_0 . On notera que l'on peut initialiser la récurrence par $b_n = a_{n+1}$ ou quitte à rajouter un point par $b_{n+1} = 0$.

Cet Algorithme s'appelle algorithme d'Horner, il revient à évaluer la valeur du polynôme P en le factorisant sous la forme :

$$P(x) = ((\dots(a_{n+1} * x + a_n) * x + a_{n-1}) * x + \dots) * x + a_1$$

Nous avons noté que la dérivée de $P(x)$ au point y s'obtient en évaluant $Q(y)$. Il suffit donc de réappliquer l'algorithme d'Horner au polynôme Q pour évaluer $P'(y)$.

```
-->function [Q,R]=Horner(P,xi)                                ← Évalue le polynôme P au point xi
-->  [lhs,rhs]=argn(0);
-->  a=coeff(P);                                                ← Les coefficients du polynôme
-->  na=size(a,'*');
-->  b=ones(a);          ← Espace pour stocker les coefficient du polynôme Q et la valeur R=P(xi)
-->  b($)=a($);                                                ← Initialisation
-->  for i=na-1:-1:1
-->    b(i)= b(i+1)*xi+ a(i)                                    ← Les coefficients du polynôme Q
-->  end
-->  if lhs==2 then
-->    R = b(1);
-->    Q=poly(b(2:$),'x','coeffs');                            ← Construction de Q
-->  else
-->    Q= b(1)
-->  end
-->endfunction

-->P=poly(1:3,'x','coeffs');                                  ← P(x) = 1 + 2x + 3 x^2

-->xi=0.5;
-->[Q,R]=Horner(P,xi);

-->P2= Q*poly(xi,'x','roots')+R                                ← On vérifie le lien entre P, Q et R
P2 =
```

$$1 + 2x + 3x$$

```
-->if norm(coeff(P-P2)) > %eps then pause;end

-->if norm( R - horner(P,xi)) > %eps then pause;end    ← Contrôle avec la fonction Sci-
lab horner

-->[Pprimxi]=Horner(Q,xi);                                ← Calcul de  $P'(xi)$ 

-->if norm( Pprimxi - horner(derivat(P),xi)) > %eps then pause;end ← Contrôle avec les fonc-
tions Scilab horner et derivat
```

En fait, on peut calculer directement les valeurs $P(y)$ et $P'(y)$ sans passer par l'intermédiaire de la construction du polynôme Q , c'est à dire sans stocker les coefficients du polynôme Q . On écrit conjointement les deux algorithmes de Horner et pour faciliter la gestion des indices on initialise le deuxième avec $c_n = 0$ plutôt que $c_{n-1} = b_n$. Ainsi les deux vecteurs de coefficients à calculer auront même taille :

$$\begin{aligned} b_i &= b_{i+1}y + a_{i+1}, & b_n &= a_{n+1}, & \text{pour } i &= n-1, n-2, \dots, 0 \\ c_i &= c_{i+1}y + b_{i+1}, & c_n &= 0, & \text{pour } i &= n-1, n-2, \dots, 0 \end{aligned}$$

le calcul du couple (b_0, c_0) permet d'obtenir le couple de valeurs $(P(y), P'(y))$. En généralisant cette idée on peut calculer les p -dérivées successives du polynôme P en un point y . Posons $P_i(x) = P_{i+1}(x)(x-y) + R_i$ avec $P_0(x) = P(x)$. Par un algorithme de Horner on peut calculer en procédant comme dans (3) les valeurs $P_i(y)$ pour $i = 0, \dots, p-1$. Il est ensuite facile de voir par dérivation composées que :

$$P_0^{(i)}(y) = i!P_i(y), \quad \text{pour } i \geq 1 \quad (4)$$

et d'obtenir ainsi les dérivées successives de P en y .

```
-->function [val]=Hornerp(P,xi,p)                                ← Évalue  $P^{(i)}(xi)$  pour  $i = 0, \dots, p-1$ 
-->  V=zeros(p,1);
-->  a=coeff(P);
-->  V(1)=a($);                                                    ← Initialisation
-->  M=diag(xi*ones(1,p))+ diag(ones(1,p-1),-1);
-->  n=size(a,'*');
-->  for i=1:n-1
-->    V=M*V + [a(n-i);zeros(p-1,1)],                                ← Horner simultané
-->  end
-->  F=ones(p,1);
-->  for i=2:p-1 do F(i+1)=F(i)*i ; end                                ← Calcul des  $i!$ 
-->  val = V.* F;                                                  ← Les dérivées de P en y
-->endfunction
```

```

-->P=poly(1:8,'x','coeffs');
-->xi=1.5 ;
-->p=10;
-->val=Hornerp(P,xi,p);

-->Q=P;
-->for i=1:p-1, Q=[Q;derivat(Q($))];end          ← Les dérivées successives de P

-->if norm(val -horner(Q,xi)) > %eps then pause;end          ← Vérifions le résultat

```

1.1 Utilisation de l'algorithme de Horner pour la factorisation

Supposons que l'on ait estimé un zéro ξ du polynôme P , dans la décomposition de $P(x) = Q(x)(x - \xi) + R$ on doit avoir $R = 0$ et la méthode de Horner permet d'éliminer la racine ξ du polynôme $P(x)$ puisqu'elle donne les coefficients du polynôme Q . Mais si ξ est une racine de $P(x)$ on doit aussi avoir $R = b_0 = 0$ dans l'algorithme de Horner.

On peut donc utiliser la récurrence (3) à partir de $b_n = a_{n+1}$ (méthode forward) ou à partir de $b_0 = 0$ méthode backward. Si l'on cherche à éliminer successivement les racines d'un polynôme en éliminant à chaque fois la plus grande racine, la méthode backward donne une meilleure stabilité numérique.

On peut comparer les deux méthodes avec le polynôme $P(x) \stackrel{\text{def}}{=} \prod_{j=0}^{13} (x - 2^{-j})$ en utilisant la fonction `roots` de Scilab pour estimer les valeurs des racines.

```

-->function [Q]=deflate_forward(P,xi)          ← Élimine la racine xi méthode forward
-->  a=coeff(P);                                ← Les coefficients du polynôme
-->  na=size(a,'*');
-->  b=ones(a);          ← Espace pour stocker les coefficients du polynôme Q et la valeur R=P(xi)
-->  b($)=a($);                                ← Initialisation
-->  for i=na-1:-1:1
-->    b(i)= b(i+1)*xi+ a(i)                    ← Les coefficients du polynôme Q
-->  end
-->  Q=poly(b(2:$),'x','coeffs');              ← Construction de Q
-->endfunction

-->function [Q]=deflate_backward(P,xi)          ← Élimine la racine xi méthode backward
-->  a=coeff(P);                                ← Les coefficients du polynôme
-->  na=size(a,'*');
-->  b=ones(a);          ← Espace pour stocker les coefficients du polynôme Q et la valeur R=P(xi)
-->  b(1)=0;                                ← Initialisation
-->  for i=1:na-1
-->    b(i+1)= (b(i)-a(i))/xi                    ← Les coefficients du polynôme Q
-->  end
-->  Q=poly(b(2:$),'x','coeffs');              ← Construction de Q
-->endfunction

```

```
-->n=13;
-->r= 2.^(-[0:n]);
-->P=poly(r,'x','roots');
-->rr=roots(P)
rr =
```

```
! 0.0001221 !
! 0.0002441 !
! 0.0004883 !
! 0.0009766 !
! 0.0019531 !
! 0.0039062 !
! 0.0078125 !
! 0.015625 !
! 0.03125 !
! 0.0625 !
! 0.125 !
! 0.25 !
! 0.5 !
! 1. !
```

```
-->maxi(abs(rr'-r($:-1:1)))
ans =
```

```
4.441D-16
```

```
-->P1=P;
-->r3=zeros(r);
-->for i=1:n+1
--> r3(i)=maxi(real(roots(P1)));
--> P1=deflate_backward(P1,r3(i));
-->end
-->maxi(abs(r3-r))
ans =
```

← On élimine la plus grande racine

← Test du résultat

```
6.661D-16
```

```
-->P1=P;
-->r4=zeros(r);
-->for i=1:n+1
--> r4(i)=maxi(real(roots(P1)));
--> P1=deflate_forward(P1,r4(i));
-->end
```

← On élimine la plus grande racine

```
-->maxi(abs(r4-r))
ans =
```

← Test du résultat

1.2 La méthode de Horner backward

L'équation (3) permettant de calculer les coefficients du polynôme Q peut-être vu comme un système dynamique discret. La stabilité de ce système ou de celui obtenu pour le calcul simultané de P et de ses dérivées au point y dépend de $|y|$. Pour $|y| > 1$ le système est instable et on va donc pour des polynômes de grande taille amplifier les erreurs de calcul si l'on utilise (3). Mais on peut remarquer que $P(x)$ s'écrit aussi :

$$P(x) = x^n \tilde{P}(1/x), \quad \text{avec} \quad \tilde{P}(x) \stackrel{\text{def}}{=} \sum_{i=0}^n a_{i+1} x^{(n-i)} \quad (5)$$

Pour une valeur de y telle que $|y| > 1$. on se ramène à un système dynamique stable en utilisant l'algorithme de Horner pour l'évaluation de $\tilde{P}$ et de ses dérivées en $1/y$ et on utilise la relation (5) pour les relier aux valeurs de P et de ses dérivées en y .

```
-->n=13;
-->r= 2.^(-[0:n]);
-->P=poly(r,'x','roots');

-->xi=10000 ;
-->p=2;
-->val=Hornerp(P,xi,p);                                     ← P(xi) par une méthode forward

-->xii=(1/xi);
-->a=coeff(P);
-->Ptilde=poly(a($:-1:1),'x','coeffs');                     ← Construction de P̃
-->tval=Hornerp(Ptilde,xii,p);                                ← P̃(1/xi) par une méthode forward

-->n = degree(P);
-->back(1)= tval(1)*xi^n;                                     ← P(xi)
-->back(2)= n*xi^(n-1)*tval(1) - xi^(n-2)*tval(2);          ← P'(xi)

-->norm((back-val)./val)
ans =

4.614D-16
```

2 Recherche des zéros d'un polynôme

On utilise ici la méthode de Newton pour trouver les zéros d'un polynôme. Soit $P(x)$ un polynôme de degré n , on utilise pour trouver les zéros l'algorithme de Newton qui consiste à effectuer les itérations suivantes :

$$x_{k+1} = x_k - P(x_k)/P'(x_k) \quad (6)$$

Il faut pouvoir évaluer la valeur du polynôme et de sa dérivée au point courant x_k et nous avons vu qu'un algorithme d'Horner permet d'évaluer ces deux quantités.

La convergence de l'algorithme de Newton dans ce cas particulier est donnée par le théorème suivant :

Théorème 1 *Soit P un polynôme de degré n dont toutes les racines sont réelles. La méthode de Newton donne une suite strictement décroissante pour $x_0 > \xi_1$ où ξ_1 est la plus grande racine de P .*

On trouvera la preuve de la convergence dans [1]. Pour implémenter cet algorithme il faut trouver un majorant de ξ_1 . Plusieurs formules de majoration sont données dans la littérature. Par exemple :

$$|\xi_1| < \max \left\{ 1, \sum_{j=0}^{n-1} \frac{|a_{j+1}|}{|a_{n+1}|} \right\} \quad \text{si} \quad P(x) = \sum_{i=0}^n a_{i+1}x^i \quad (7)$$

$$|\xi_1| < \max \left\{ \frac{|a_1|}{|a_{n+1}|}, 1 + \frac{|a_2|}{|a_{n+1}|}, \dots, 1 + \frac{|a_n|}{|a_{n+1}|} \right\} \quad \text{si} \quad P(x) = \sum_{i=0}^n a_{i+1}x^i \quad (8)$$

```
-->function xn=newton(P,x,prec)
-->  while %t
-->    val=Hornerp(P,x,2);                                ← val=[P(x),P'(x)]
-->    xn= x -val(1)/val(2);
-->    if norm(xn-x) <= prec*norm(x) then
-->      break;
-->    end
-->    x=xn;
-->  end
-->endfunction

-->n=13;
-->r= 2.^(-[0:n]);
-->P=poly(r,'x','roots');                                ← Le polynôme test

-->a=coeff(P);
-->xd=max([1+abs(a(2:$-1)/a($)),abs(a(1)/a($))]) ;        ← Un majorant de la plus grande racine
```

```
-->y1=newton(P,xd,%eps)
y1 =

1.
```

En éliminant à chaque fois la plus grande racine trouvée on peut chercher toutes les racines du polynôme $P(x)$. On peut constater dans le code qui suit que la méthode d'élimination choisie (forward ou backward) n'est pas anodine !

```
y1 =

1.

-->y=y1; rnf= zeros(r); rnf(1)=y; Q=P;
-->for i=2:4
--> Q=deflate_forward(Q,y);
--> y=newton(Q,y,%eps);
--> rnf(i)=y;
-->end

-->y=y1;rnb= zeros(r); rnb(1)=y; Q=P;
-->for i=2:(n+1)
--> Q=deflate_backward(Q,y);
--> y=newton(Q,y,%eps);
--> rnb(i)=y;
-->end

-->if norm(abs(r-rnb)) > 10*%eps then pause;end
```

← Problème au dela de $i = 4$
← On élimine la plus grande racine
← Évaluation de la plus grande racine

← On élimine la plus grande racine

Plutôt que d'essayer de simplifier le polynôme P on peut utiliser la méthode de Maehly (1954) qui consiste à appliquer la méthode de Newton à la fraction rationnelle $P(x)/(x - \xi)$ plutôt que d'éliminer la racine ξ du polynôme $P(x)$. Ainsi si l'on a estimé les j premières racines du polynôme, notées ξ_i , on cherche la $j + 1$ -ème racine par l'algorithme de Newton :

$$x_{k+1} = x_k - Q(x_k) \quad \text{avec} \quad Q(x) \stackrel{\text{def}}{=} \frac{P(x)}{P^{(1)}(x) - \sum_{i=1}^j \frac{P(x)}{x - \xi_i}} \quad (9)$$

Pour initialiser l'algorithme, on peut utiliser la dernière racine trouvée (à epsilon près pour éviter une division par zéro).

```
-->function y=newton(P,x,r,prec)
--> while %t
--> val=Hornerp(P,x,2);
--> if r<>[] then
```

```

-->     w=sum(val(1)./(x-r));
-->     else
-->         w=0.0;
-->     end
-->     xn= x -val(1)/(val(2)-w);
-->     if norm(xn-x) <= prec*norm(x) then
-->         break;
-->     end
-->     x=xn;
--> end
--> y=xn;
-->endfunction

-->n=13;r= 2.^(-[0:n]);P=poly(r,'x','roots');
-->y=newton(P,10,[],%eps);

-->rn= zeros(r); rn(1)=y;

-->for i=2:n+1
-->    y=newton(P,y-100*%eps,rn(1:i-1),10*%eps);
-->    rn(i)=y;
-->end

-->if maxi(abs(rn-r)) > 10*%eps then pause;end

```

$$\leftarrow \mathbf{r} = \sum_{i=1}^j \frac{P(x)}{x - \xi_i}$$

Références

- [1] J. Stoer and R. Burlisch. *Introduction to Numerical Analysis*. Springer-Verlag, 1980.