Tronc Commun Maple
Recherche dichotomique, Approximation numérique

Corrigé

26 novembre 2006

1 Recherche dichotomique

Le problème consiste à déterminer si un élément figure dans un tableau. Si le tableau est quelconque, on ne peut guère faire autre chose qu'examiner toutes les cases du tableau. En revanche si l'on dispose d'un ordre sur le domaine des éléments du tableau, on peut faire mieux : on trie le tableau une fois pour toutes, puis pour déterminer l'appartenance d'un élément au tableau, on utilise le diviser pour régner ci-dessous :

```
dicho :=proc(t,m,i,j) # recherche m dans t[i..j]
    if j>i+1 then
        if t[floor((i+j)/2)]=m then RETURN(true)
        elif t[floor((i+j)/2)]<m then
            dicho(t,m,floor((i+j)/2)+1,j)
        else dicho(t,m,i,floor((i+j)/2)-1)
        fi
    else RETURN(evalb(t[i]=m or  t[j]=m))
    fi ;
end ;
```

On compare `m` avec le milieu du tableau : s'ils sont égaux on a gagné, sinon on recherche `m` dans la première ou la deuxième moitié du tableau.

Chaque appel récursif effectue un nombre borné d'opérations, et il y a $\Theta(\ln n)$ appels récursifs (la taille du tableau actif est divisée par 2 à chaque fois), donc on obtient du $\Theta(\ln n)$.

2 Polynome d'interpolation

Il faut calculer les nombres $c(0,k)$ pour k variant de 0 à n. On connaît $c(k,k)$ pour tout k, et la formule de récurrence de l'énoncé donne la valeur de $c(p,q)$ en fonction de celles de $c(p+1,q)$ et $c(p,q-1)$: on peut donc remplir ligne par ligne la table suivante, que ll'on appellera `T` dans le programme :

```
c(0,0) c(1,1) c(2,2) . . . . . c(n-1,n-1) c(n,n)
    c(0,1) c(1,2) c(2,3) . . . . . c(n-1,n)
        c(0,2) c(1,3)      . . . .    c(n-2,n)
                              . . .
                    c(0,n-1) c(1,n)
                         c(0,n)
```

- La première ligne, à laquelle on donnera l'indice 0 pour simplifier l'écriture, contient les termes $c(k,k) = f(x_k)$.
- La ligne d'indice k contient les termes $c(i, i+k)$ pour i variant de 0 à $n-k$, et chacun d'entre eux est obtenu par la formule de récurrence à partir des deux termes situés au-dessus de lui dans la table.
- Pour finir, on remplit toute la table `T` en plaçant le calcul du terme $c(i, i+k)$ à l'intérieur d'une boucle qui fait varier k de 1 à n.
- Il reste à constrire le résultat : c'est une somme que l'on obtient par accumulation dans une variable `res` initialisée à `T[0,0]`, en utilisant une variable auxiliaire `prod` dans laquelle on calcule successivement les polynomes $(x-x_0)\dots(x-x_i)$.

```
Newton := proc(f,S)
local T,i,k,n,prod,res;
    n :=nops(S)-1 ;
    for i from 0 to n do T[i,i] :=f(S[i+1]) od ;
    for k to n do
        for i from 0 to n-k do
            T[i,i+k] :=(T[i+1,i+k]-T[i,i+k-1])
                        /(S[i+k-1]-S[i+1])
        od
    od ;
```

```
    res :=T[0,0];
    prod :=1;
    for i to n do
        prod :=prod*(x-S[i]);
        res :=res+T[0,i]*prod;
    od;
    res
end;;
```

On remarque qu'il est possible de calculer `res` à l'aide de l'algorithme de Horner. On aboutit au programme suivant :

```
Newton := proc(f,S)
local T,i,k,n,prod,res;
    n :=nops(S)-1;
    for i from 0 to n do T[i,i] :=f(S[i+1]) od;
    for k to n do
        for i from 0 to n-k do
            T[i,i+k] :=(T[i+1,i+k]-T[i,i+k-1])
                        /(S[i+k-1]-S[i+1])
        od
    od;
    res :=T[0,n];
    for i from n-1 to 0 by -1 do
        res :=res*(x-S[i+1])+T[0,i];
    od;
    res
end;;
```

3 Calcul approché d'intégrales

3.2 Méthode des rectangles

```
Iapprox := proc(f,a,b,n)
local res,i,y_i;
    res :=0;
    y_i := a +((b-a)/(2*n));
    for i from 0 to (n-1) do
        res := res+f(y_i);
        y_i := y_i+(b-a)/n;
    od;
    evalf(res*(b-a)/n);
end;;
```

3.3 Méthode des trapèzes

```
Itrapez := proc(f,a,b,n)
local res,i,x,h;
    res :=0;
    x := a;
    h := (b-a)/n;
    for i from 1 to n do
        x := x+h;
        res :=res+f(x);
    od;
    res :=evalf(h*(res+(f(a)+f(b))/2));
end;;
```