TD n°3 : corrigé

PTSI B Lycée Eiffel

16 octobre 2012

Premiers exercices

1. t :=array[1..10,1..10] : for i to 10 do for j to 10 do t[i,j] :=10*(i-1)+j : t ;
2. diagonale :=proc(t : :array,n : :integer)
   local d : :real,i : :integer ;
   d :=0 ;
   for i to n do d :=d+t[i,i] end do ;
   print(d) ;
   end proc ;
3. magique :=proc(t : :array,n : :integer)
   local s : :real, a : :real, i : :integer, j : :integer, z : :boolean ;
   z :=true ; s :=0 ;
   for j to n do s :=s+t[1,j] end do ;
   for i from 2 to n do a :=0 ; for j to n do a :=a+t[i,j] end do ; if a<>s then z :=false end do ;
   for j to n do a :=0 ; for i to n do a :=a+t[i,j] end do ; if a<>s then z :=false end do ;
   a :=0 ; for i to n do a :=a+t[i,i] end do ; if a<>s then z :=false ;
   a :=0 ; for i to n do a :=a+t[i,n-i+1] end do ; if a<>s then z :=false ;
   print(z) ;
   end proc ;

Un peu de polynômes

1. derive :=proc(t : :list,n : :integer) ;
   local u : :list,i : :integer ;
   for i to n do u[i] :=(i+1)*t[i+1] ;
   print(u) ;
   end proc ;
2. prodpoly :=proc(t : :list,u : :list,n : :integer,p : :integer)
   local v : :list,i : :integer,j : :integer ;
   v :=list(1..n+p+1) ;
   for i to n+p+1 do v[i] :=0 end do ;
   for i to n+1 do for j to p+1 do v[i+j-1] :=v[i+j-1]+t[i]*u[j] end do ;
   print(v) ;
   end proc ;
3. evalpoly :=proc(t : :list,n : :integer,x : :real)
   local p : :real, a : :real, i : :integer ;
   a :=1 ; p :=t[1] ;
   for i to n do a :=a*x ; p :=p+t[i+1]*a ;
   print(p) ;
   end proc ;

4. La vérification rigoureuse du fait que l'expression est égale à $P(x)$ se fait par récurrence sur le degré de P. Si P est de degré 0, l'expression se résume à $p_0 = P(x)$, c'est bon. Supposons que ça marche pour tout polynôme de degré n, alors pour un polynôme de degré $n+1$, $p_0 + x(p_1 + x(p_2 + x(\cdots + x(p_{n+1})))) = p_0 + xQ(x)$, où $Q(x) = p_1 + p_2x + \cdots + p_{n+1}x^n$ est un poltnôme de degré n auquel on peut appliquer l'hypothèse de récurrence. On a donc $p_0 + xQ(x) = p_0 + p_1x + p_2x^2 + \cdots + p_{n+1}x^{n+1} = P(x)$, le résultat est démontré. L'algorithme de Hörner est très facile à programmer :

   ```
   horner := proc(t : :list,n : :integer,x : :real)
   local p : :real, i : :integer ;
   p :=t[n+1] ;
   for i to n do p :=p*x+t[n+1-i] end do ;
   print(p) ;
   end proc ;
   ```

   La procédure « normale » utilise deux multiplication et une addition à chaque étape, l'algorithme de Hörner n'utilise qu'une multiplication et une addition, il est donc plus rapide.

Coefficients binomiaux

Vous avez déjà du croiser dans votre scolarité antérieure les coefficients binomiaux $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, nous en reparlerons en cours de maths un de ces jours. Pour l'instant, notre objectif est simplement de les calculer de façon efficace.

1.
   ```
   binom :=proc(n : :integer,k : :integer)
   local f1 : :integer, f2 : :integer, f3 : :integer, i : :integer ;
   f1 :=1 ; f2 :=1 ; f3 :=1 ;
   for i to k do f1 :=f1*i end do ;
   f2 :=f1 ;
   for i from k+1 to n do f1 :=f1*i ; f3 :=f3*(i-k) ;
   print(f1/(f2*f3)) ;
   end proc ;
   ```
2.
   ```
   binombis :=proc(n : :integer)
   local t : :list, i : :integer, k : :integer ;
   t :=list(0..n) ;
   t[0] :=1 ; for k to n do t[k] :=0 ;
   for i to n do for k from n-i to n do t[n-k] :=t[n-k]+t[n-k-1] end do ; end do ;
   print(t) ;
   end proc ;
   ```

   Le fait de remplir les cases du tableau dans cet ordre n'est pas du tout anodin : en commençant par remplir la ligne suivante du triangle de Pascal par la droite, même si on efface au fur et à mesure du remplissage les coefficients de la ligne précédent, on ne les efface que lorsqu'on les a déjà utilisés pour les deux calculs de la ligne du dessous. Ainsi, on utilise un seul tableau mais on n'a pas besoin de variables auxiliaires. Cette méthode utilise $i+1$ additions à chaque étape, soit au total $\sum_{i=1}^{n} i+1 = \frac{(n+1)(n+2)}{2} - 1$ additions. La première méthode utilise $k+2*(n-k)$ mutliplications, et encore une multiplication et une division, soit $2n-k+2$ opérations. C'est beaucoup moins qu'en utilisant le triangle de Pascal, mais les multiplications sont beaucoup plus gourmandes en temps de calculs que les additions, et surtout ces opération porteront sur

des entiers énormément plus gros (pour calculer par exemple $\binom{10}{5}$, qui vaut 252, la deuxième méthode ne manipulera aucun entier plus grand que 252, alors que la première ira calculer $10! = 3\ 628\ 800$ et faire une division avec). En pratique, la deuxième méthode est nettement plus efficace. Qui plus est, elle donne tous les coefficients binomiaux $\binom{n}{k}$ et pas un seul comme la première méthode.