Exponentiation rapide

La procédure proposée au concours centrale calcule a^n. On peut la réécrire de façon plus explicite de la façon suivante :

```
expo_rapide2:= proc(a , n) # a est la base et n l'exposant, la procédure calcule a^n
local  N , base, exposant;
N:=1 ; base:= a; exposant:= n;
while exposant <> 0 do
if type(exposant , odd) = true then N:=N*base ; exposant:=exposant – 1 fi;
else base:=base^2 ; exposant:=iquo(exposant,2); # iquo(m,2) = reste dans la division de l'entier m par 2
od;
N;
end;
```

1) La quantité $N*base^{exposant} = a^n$ est un <u>invariant</u> de la boucle. Vérifier que cette égalité est satisfaite à chaque passage dans la boucle.

2) Ecrire une version récursive expo_rapide3 de l'algorithme d'exponentiation rapide en utilisant la méthode suivante :

$$a^{exposant} = \begin{cases} (a^2)^{exposant/2} & \text{si exposant pair} \\ a * (a^2)^{(exposant-1)/2} & \text{si exposant impair} \end{cases}$$

3) Adapter l'algorithme expo_rapide2 pour réaliser une version du produit A^n où A est une matrice :
Sous Maple, pour rentrer une matrice, on procède de la façon suivante :
> with(linalg); charge le package linalg contenant des commandes propres à l'algèbre linéaire (chap. 11)
> A:= matrix([[ligne1] , [ligne2] , ... , [ligne n]]);
par exemple matrix([[1,2,3],[4,5,6]]); donne la matrice $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$
Le produit de deux matrices A et B se calculent par evalm(A&*B).

4) Version universelle de expo_rapide : écrire une procédure expo_rapide4 (a , n, loi) qui calcule a^n dans le corps $\mathbb{R}$ ou $\mathbb{C}$ lorsque loi = p , qui calcule A^n dans l'anneau des matrices lorsque loi = mat et qui calcule $a^n = a \circ a \circ ... \circ a$ lorsque a est une application et loi = o. On pourra s'aider de la procédure suivante :

```
produit:= proc(a , b , loi) # loi : p (produit ordinaire dans ℝ ou ℂ) , mat (produit matriciel) , o (composition des
applications, a et b étant des fonctions)
local y;
if loi = p then expand(a*b)
elif loi = mat then evalm(a&*b)
elif loi = o then y:=a(b(x));unapply(y,x);
else print(`opération non reconnue, désolé `)
fi;
end;
```

Utiliser cet algorithme pour calculer F^{10} où $F(x,y,z) = (2x – y + z , 3y – 2z , – x + 2y – z)$.

5) Hörner version récursive et non récursive.

Commandes utiles sur les polynômes (rappel : un polynôme est vu comme une expression; dans les commandes ci-dessous, la variable est t)

> collect(*polynôme* , *t*) ; # écrit un polynôme dans la base canonique (1, *t*, t^2, …).
> expand(*expression*) ; # pour développer une expression.
> coeffs(*polynôme* , *t*) ; # retourne la suite des coefficients du polynôme (pas forcément dans l'ordre !)
> coeff(*polynôme*, *t* , *k*) ; # retourne le coefficient de t^k dans *polynôme*, $k \geq 0$.
> degree(*polynôme*, *t*) ; # retourne le degré de *polynôme*.

Ecrire une procédure Horner1(*P*,*x*) qui calcule $P(x)$ de façon récursive par la méthode de Hörner.

Ecrire une procédure itérative Horner2(*P*,*x*) qui calcule $P(x)$ par la méthode de Hörner : si $P = \sum_{k=0}^{n} a_k t^k$, utiliser que $P(x) = (((a_n x + a_{n-1}) x + a_{n-2}) x + ...) + a_0$; à l'étape i, on a $P(x) = s_i x^{n-i} + R_i$ avec $R_i = \sum_{k=0}^{i-1} a_k x^k$. Etablir d'abord la relation de récurrence entre les s_i puis écrire la procédure qui calcule les s_i successifs.
Exercice : s_0, ... , s_{n-1} sont les coefficients dans le quotient de la division euclidienne de P par $X - x$, le reste étant $P(x)$.

6) Un exercice sur les polynômes pour finir ...

On considère $F : \mathbb{R}_3[X] \to \mathbb{R}[X]$, $P \mapsto (t^2 - 1)P''(t) + (-3t + 2)P'(t) + 4P(t - 1) - P(t)$.
P et $F(P)$ seront des expressions polynomiales en la variable t et on utilisera la commande diff(*P*,*t*) … ; $P(t - 1)$ désigne la composée des polynômes $t - 1$ et P et s'effectue à l'aide de la commande subs. Il faut développer à l'aide de expand avant d'appliquer la commande coeff.
Construire la matrice de F relativement à la base canonique de $\mathbb{R}[X]$ (voir syntaxe question 3).
Déterminer Ker F soit en résolvant un système (commande solve), soit à l'aide de la commande kernel(matrice).

On considère $G : \mathbb{R}_3[X] \to \mathbb{R}_4[X]$, $P \mapsto t^2P''(t) + 2t(t + 2)P'(t) - 2(4t - 1)P(t + 1) + P(t)$.
Déterminer $G(P)$ où P est dans $\mathbb{R}_3[X]$ (on pourra écrire *P*:= sum(*a*[*k*] * *t*^*k* , *k* = 0..3))
Construire la matrice de G relativement à la base canonique de $\mathbb{R}[X]$.
Prouver que G est injective et en déduire que Im g est un hyperplan de $\mathbb{R}_4[X]$.
Déterminer une équation de cet hyperplan. Deux méthodes :

a) soit résoudre $G(P) = \sum_{k=0}^{4} b_k t^k$ (ne garder que 4 équations sur 5 puis remplacer dans l'équation laissée de côté, les a_k par leur expression en fonction des b_k ...)

b) soit chercher l'équation de l'hyperplan sous la forme $\sum_{k=0}^{4} \alpha_k b_k = 0$ en remarquant que les vecteur colonne de la matrice de G doivent satisfaire l'équation. Commande éventuellement utile : transpose(*matrice*) inverse les lignes et les colonnes d'une matrice ...