MIAS 1 et MASS 1 – Programmation fonctionnelle

TP n° 10 : corrigé

```
; ================================================================
; Exercice 1 : Implantation de la methode de Newton

; derive :    Fonction,      Reel    --->            Fonction
;          f : x |-> f(x)  ,  dx     |-->    x |-> f(x+dx)-f(x) * 1/dx

(define derive
  (lambda (f dx)
   (lambda (x) (/ (- (f (+ x dx)) (f x)) dx))))


; assez-proche? :    Fonction,      Reel+, Reel   ---> Booleen
;                f : x |-> f(x)    err    x0     |--> vrai si |f(x0)| < err

(define assez-proche?
  (lambda (f err x0)
   (< (abs (f x0)) err)))


; ameliore :     Fonction,    Reel,  Reel   ---> Reel
;             f : x |-> f(x)   x0      dx    |--> x0 - f(x0)/f'(x0)
; en prenant pour f' la fonction derivee approchee donnee par la fonction
; scheme derive precedemment definie

(define ameliore
  (lambda (f x0 dx)
   (- y (/ (f x0)
           ((derive f dx) x0))) ))


; Pour trouver une racine approchee d'une fonction f par la methode de Newton
; en partant d'une solution approchee initiale init, on ameliore la solution
; initiale jusqu'a ce qu'elle soit assez proche d'une racine.

; newton :   Fonction, Reel, Rell, Rell ---> Reel
;                f     init   dx   err  |-->  x0   tel que |f(x0)|<err

(define newton
  (lambda (f init dx erreur)
   (if (assez-proche? f init erreur)
       init
       (newton f (ameliorer f init dx) dx erreur)) ))



; ================================================================
; Exercice 2 :

; 1/
; combine :          Fonction,                 Fonction          --->   Fonction
;           f: x1,...,xp |-> f(x1,...,xp)      g : x |-> g(x)    |-->  x1,...,xp |-> f(g(x1),...g(xp))

(define combine
  (lambda (f g)
   (lambda (l) (apply f (map g l)))))

; 2/
; a) Le produit scalaire de 2 vecteurs est egal a la somme
; des produits des coordonnees.
; si v=(a1 a2 ... ak) et w=(b1 b2 ... bk)
; alors v.w = a1 x b1 + a2 x b2 + ... + ak x bk

(define produit-scalaire
  (lambda (v w)
   (apply + (map * v w))))


; b) Le produit d'une matrice m par un vecteur v est un vecteur
; dont les coordonnees sont les produits scalaires de chaque
; vecteur-ligne de la matrice par le vecteur v.
; si m = (v1 v2 ... vk)
; alors m*v = ( v1.v v2.v ... vk.v )

(define (matrice-fois-vecteur m v)
   (map (lambda (l) (produit-scalaire l v)) m))



; ================================================================
; Exercice 3 : Un generateur de fonction recursive

; On definit localement une fonction recursive utilisant les quatre
; fonctions donnees en arguments et on renvoie  cette fonction

(define genrec
  (lambda (arret valarret calcul decrois)
   (define f
               (lambda (x)
                   (if (arret x)
                       (valarret x)
                       (calcul x (f (decrois x))))))
     f))


; Utilisation de la fonction genrec

(define longueur
   (genrec (lambda (l) (null? l))
           (lambda (l) 0)
           (lambda (l r) (+ 1 r))
           (lambda (l) (cdr l))))


(define renverse
   (genrec (lambda (l) (null? l))
           (lambda (l) '())
           (lambda (l r) (append r (list (car l))))
           (lambda (l) (cdr l))))


(define nbatt
   (genrec (lambda (l) (null? l))
           (lambda (l) 0)
           (lambda (l r)
               (if (not (pair? (car l)))
                   (+ 1 r)
                   (+ (nbatt (car l)) r)))
           (lambda (l) (cdr l))))
```