Matrices creuses

1 Introduction

Dans de nombreux domaines de la physique, les calculs numériques demandent la résolution de systèmes d'équations de taille importante, comme en mécanique des fluides, lors du calcul d'un écoulement en un grand nombre de points. Ces calculs demandent alors la manipulation de matrices de grandes tailles qui comportent souvent un grand nombre de zéro. Si l'on stocke ces matrices traditionnellement, elles prennent beaucoup de place, on préfère alors utiliser une représentation dite *c*reuse de ces matrices, c'est à dire que l'on ne stocke que les valeurs non nulles.

On se propose ici de définir quelques opérations élémentaires sur les matrices réelles au format creux.

Les matrices seront représentées par des listes de vecteurs creux. Un vecteur creux est une liste de couples indice,flottant. L'indice indique le rang de la valeur réelle dans le vecteur. De plus, on choisit de placer les indices **en ordre croissant**, c'est utile pour simplifier les calculs.

Remarque 1 *Cette représentation est totalement identique celle des polynômes creux, l'indice représentant alors le rang du coefficient.*

On définit le type vecteur ainsi :
`type vecteur == (int*float) list ; ;`
La valeur `[ (2,2.4) ; (6,3.0)]` représente alors :

$$\begin{pmatrix} 0 \\ 2.4 \\ 0 \\ 0 \\ 0 \\ 3.0 \end{pmatrix}$$

De même, une matrice est définie par une liste de vecteurs accompagnés de leurs indices, chaque vecteur représentant une ligne de la matrice :
`type matrice == (int*vecteur) list ; ;`

Remarque 2 *Comme il est rare qu'une colonne soit complètement nulle dans une matrice, on pourrait utiliser une représentation pleine pour la matrice. Ce choix dépend en fait du type de matrice auquel on a à faire.*

Ainsi, la valeur `[ 1,[ 2,2.5 ] ; 3,[1,5.0 ;3,1.1] ]` représente la matrice

$$\begin{pmatrix} 0 & 2.5 & 0 \\ 0 & 0 & 0 \\ 5 & 0 & 1.1 \end{pmatrix}$$

2 Implémentation

Question 1 *Écrire une fonction qui additionne deux vecteurs :*
`add_vect : vecteur -> vecteur -> vecteur`

Question 2 *Écrire une fonction qui additionne deux matrices :*
`add_mat : matrice -> matrice -> matrice`

Attention : pour les deux fonctions précédentes, ne pas oubliez que les vecteurs résultats doivent rester creux, c'est à dire que les valeurs nulles doivent être supprimées.

Question 3 *Comparez les deux fonctions précédentes, trouvez une fonction* `add` *commune et réécrivez-les à l'aide de cette fonction commune.*

Indice : la fonction commune peut avoir le type suivant
`('a -> 'a -> 'a) -> 'a -> ('b * 'a) list -> ('b * 'a) list -> ('b * 'a) list`

Question 4 *Écrire une fonction qui calcule le produit scalaire de deux vecteurs :*
`scal : vecteur -> vecteur -> float`

La question suivante est un peu plus difficile et non indipensable pour la suite, vous pouvez la passer dans un premier temps.

Question 5 *Comparez attentivement les fonctions* `add_mat`, `add_vect` *et* `scal`, *et trouvez une généralisation de la fonction* `add` *qui permet d'écrire simplement les fonctions* `add_mat`, `add_vect` *et* `scal`.

Question 6 *Écrire une fonction qui extrait un élément d'un vecteur étant donné son indice :*
`element_v : int -> vecteur -> float`
Éssayez d'écrire ces fonctions avec la fonction `add` *de la question 5.*

Question 7 *Écrire deux fonctions qui extraient un élément ou un ligne dans une matrice :*
`ligne : int -> matrice -> vecteur`
`element : int -> int -> matrice -> float`
Éssayez d'écrire ces fonctions avec la fonction `add` *de la question 5.*

Question 8 *Écrire une fonction qui calcule la transposée d'une matrice :*
`transpose : matrice -> matrice`
On utilisera pour cela une fonction qui ajoute une colonne de rang i (représentée par un vecteur creux) à une matrice dont toutes les colonnes actuelles sont de rang strictement supérieur à i.
`add_col : int -> vecteur -> matrice -> matrice`

Question 9 *Écrire une fonction qui calcule le produit matrice * vecteur, le vecteur étant considéré comme une colonne :*
`prod_mv : matrice -> vecteur -> vecteur`

Question 10 *Écrire une fonction qui calcule le produit matrice * matrice :*
`prod_mv : matrice -> matrice -> matrice`