TD1 : Inversion de matrice

Vendredi 4 février 2011

Le but de ce problème est de programmer un algorithme pour l'inversion de matrices réelles, connu sous le nom d'élimination de Gauss-Jordan (ou pivot de Gauss). C'est l'occasion de revenir sur les pointeurs et l'allocation de mémoire, et à ce titre il est indispensable de bien travailler les deux premières parties. La troisième, plus difficile, est un peu plus optionnelle. Dans la suite, A, désignent une matrice de réels et a_{ij} le coefficient de la ie ligne et je colonne.

Représentation mémoire d'une matrice

1. Proposer un type C pour représenter les matrices de réels, à la manière de ce qui a été vu en cours pour les vecteurs. Les matrices seront de dimension $m \times n$ (lignes/colonnes).
2. Écrire une fonction `create_matrix` permettant de créer une matrice, dont les coefficients ne sont pas initialisés.
3. Écrire une fonction `free_matrix` permettant de libérer la mémoire allouée à une matrice.
4. Écrire une fonction de copie de matrice, de prototype

   ```
   matrix copy_matrix(matrix m);
   ```

5. Soit c une constante réelle, on souhaite disposer de constructeurs produisant des matrices courantes :
   - la matrice pleine, $a_{ij} = c$ pour tous i, j,
   - la matrice diagonale, $a_{ii} = c$ pour tout i et $a_{ij} = 0$ pour $i \neq j$,
   - les matrices triangulaires supérieures (resp. inférieures), telles que $a_{ij} = c$ si $i \leq j$, et $a_{ij} = 0$ sinon (resp. $a_{ij} = c$ si $i \geq j$, et $a_{ij} = 0$ sinon).

   Écrire une fonction pour chaque type de matrice, qui initialisera une matrice déjà existante.

Remarque : on dit qu'une fonction agit **en place** quand elle stocke le résultat d'un calcul dans ses arguments, plutôt que d'allouer de la mémoire.

Opérations usuelles

1. Écrire une fonction d'affichage des matrices `print_matrix`, en utilisant la fonction `printf` et notamment :
   - le caractère de tabulation `\t` pour aligner les colonnes de la matrice affichée.
   - le formatage des flottants, par exemple `%.2f` ou `%g`, pour limiter le nombre de décimales affichées.
2. Implémenter l'opération de transposition, définie par $B = A^T$ avec $b_{ij} = a_{ji}$. Le prototype attendu est

   ```
   void transpose_matrix(matrix A, matrix B);
   ```

   et la fonction doit mettre le résultat attendu dans le deuxième argument. Penser à vérifier que les matrices sont de dimensions compatibles !

   (Optionnel) Que se passe-t-il lors d'un appel du type `transpose_matrix(A,A)` ?
3. Dans le même esprit, implémenter l'addition de deux matrices. Le prototype attendu est

   ```
   void matrix_add(matrix A, matrix B, matrix C);
   ```

   (Optionnel) Que se passe-t-il lors d'un appel du type `add_matrix(A,B,A)` ?

4. Idem, écrire la multiplication de deux matrices. Rappel : les coefficients de $A \times B$ sont les c_{ij} définis par

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

   (Optionnel) Que se passe-t-il lors d'un appel du type `mult_matrix(A,B,A)` ? Quelle mesure faut-il prendre ?

5. On définit P^{kl} comme la matrice égale à la matrice identité sauf pour 4 coefficients : $p_{kk} = p_{ll} = 0$ et $p_{kl} = p_{lk} = 1$. La matrice P^{kl} est appelée *matrice de permutation* ; expliquez pourquoi en calculant AP_{kl} et $P_{kl}A$, où A est une matrice quelconque.

Élimination de Gauss-Jordan

Le pivot de Gauss consiste à transformer une matrice A en la matrice identité par une série d'opérations sur les lignes de A. L'astuce est que ces opérations reviennent à multiplier A à gauche par des matrices bien choisies. On obtient ainsi en n étapes l'égalité :

$$(J_n \dots J_2 J_1)A = I$$

où les matrices $J_1, \dots, J_n$ sont les transformations successives appliquées à A. Par conséquent on a $A^{-1} = J_n \dots J_2 J_1$. L'algorithme consistera donc à transformer A en la matrice identité, et à appliquer les mêmes transformations à la matrice identité pour calculer A^{-1}.
Notation : l_i représente la ie ligne de A.

1. Nous aurons besoin de l'opération qui échange (ou permute) en place deux lignes d'une matrice, dont le prototype est :

   ```
   void permutation(matrix m, int i, int j);
   ```

2. Proposer une fonction pour l'opération d'"élimination" qui remplace la ie ligne l_i de A par $l_i - a_{ik} l_k$, de prototype :

   ```
   void elimination(matrix m, int i, int k);
   ```

3. Le pseudo-code suivant permet de transformer une matrice A de dimension n en la matrice identité :

pour *k allant de 1 à n* **faire**
  si *il existe* $i \geq k$ *t.q.* $a_{ik} \neq 0$ **alors**
    Échanger les lignes l_i et l_k;
    $l_k \leftarrow \frac{1}{a_{kk}} l_k$;
    pour *i allant de 1 à n, et* $i \neq k$ **faire**
      $l_i \leftarrow l_i - a_{ik} \times l_k$
    fin
  sinon
    A n'est pas inversible, abandonner.
  fin
fin

   En vous inspirant de cet algorithme, implémentez une fonction qui inverse une matrice et dont le prototype est

   ```
   void inverse(matrix A, matrix B);
   ```

   La fonction écrira son résultat dans le deuxième argument et on veillera bien à ce que le premier argument ne soit pas modifié.

Références

– Page wikipedia sur le pivot de Gauss
  `http://fr.wikipedia.org/wiki/Elimination_de_Gauss-Jordan`