Projet algèbre linéaire appliquée
L2 – 2013

Projet minicas

1. Le logiciel à fournir

Ce projet porte sur la réalisation d'un mini-interpréteur de commandes qui correspondent à des opérations classiques en algèbre linéaire. Les commandes pourront soit être saisies de manière interactive soit provenir d'un fichier. L'exécutable s'appellera `minicas` (l'acronyme CAS correspond à *Computer Algebra System*).

Ainsi, l'invocation à :

- `minicas` lance l'interpréteur, pensez à `ledit minicas` pour bénéficier des fonctionnalités habituelles des interpréteurs (retour sur les commandes précédentes par exemple)
- `minicas` *`fich`* ou `minicas` < *`fich`* exécute une à une les commandes présentes dans le fichier *`fich`* puis termine.

Pour `minicas` < *`fich`* , il est possible d'utiliser l'appel système `stat` pour connaître le type de fichier associé au descripteur 0 (voir `man stat`).

2. Les commandes

L'ensemble des commandes doit englober :

- la déclaration d'une variable de type flottante ou matrice de flottants (commande `matrix`)
- l'addition, soustraction, multiplication de matrices (cmd : `addition`, `sub`, `mult`)
- la multiplication par un scalaire (cmd : `mult_scal`)
- l'exponentiation (cmd : `expo`)
- la transposition (cmd : `transpose`)
- le calcul du déterminant (cmd : `determinant`)
- l'inversion d'une matrice (cmd : `invert`)
- la résolution d'un système d'équations linéaire donné sous la forme A.X = b (cmd : `solve`)
- le calcul du rang (cmd : `rank`)
- une commande `speedtest` décrite plus loin
- la commande `quit`

La syntaxe sera celle de la séquence suivante (le caractère '>' désigne ici le prompt) :

```
> a : 4
            4
> m1 : matrix([1, 2, 1], [a, 2, 6], [2, 3, 1])
             [ 1  2  1 ]
             [ 4  2  6 ]
             [ 2  3  1 ]

> m2 : matrix([1, 2], [3, 4], [5, 6])
             [ 1  2 ]
             [ 3  4 ]
             [ 5  6 ]
```

```
> m3 : mult(m1, m2)
             [ 12  16 ]
             [ 40  52 ]
             [ 16  22 ]

> m4 : invert(m1)
              [ -2.0   0.125    1.25 ]
              [  1.0   -0.125  -0.25 ]
              [  1.0    0.125  -0.75 ]

> det : determinant(m1)
            8

> b : matrix([4],[6],[5]);
              [ 4 ]
              [ 6 ]
              [ 5 ]

> x : solve(m1, b)
              [ -1 ]
              [ 2  ]
              [ 1  ]

> r : rank(m1)
            3

> m : toto(m1)
            toto : function not implemented

> quit
```

Les matrices sont des matrices numériques. Ainsi dans l'exemple précédent, la matrice `m1` comporte le symbole `a` qui sera immédiatement remplacé par sa valeur numérique. Une modification ultérieure de `a` ne change pas les valeurs de `m1`.

3. La commande speedtest

La commande `speedtest` aura la syntaxe suivante :

```
speedtest commande taille_min taille_max pas [ nb_secondes ]
```

avec la signification comme dans l'exemple suivant :

exemple : `speedtest mult 5 100 5 1`
- génère itérativement des couples de matrices de taille 5, 10, 15, …, 100 et effectuent la multiplication entre les matrices d'un couple.
- produit en sortie un fichier image contenant le graphique représentant le temps de calcul en fonction des tailles (dans une unité à définir qui doit figurer sur le graphique)
- si le temps de calcul dépasse 1 seconde le programme produit le graphique et termine.

4. Opérations additionnelles

Par ailleurs, il vous sera demandé d'implémenter une fonction parmi les 3 opérations qui suivent.

1. **Décomposition LU**

voir TD 2
commande : `decomposition`
exemple avec la matrice m1 ci-dessus

```
> decomposition(m1)

  L :       [ 1.0  0.0      0.0 ]
            [ 4.0  1.0      0.0 ]
            [ 2.0  0.16666 1.0 ]

  U :       [ 1.0   2.0   1.0       ]
            [ 0.0  -6.0   2.0       ]
            [ 0.0   0.0   -1.33333 ]
```

2. **Noyau**

Un algorithme simple pour trouver une base du noyau d'une application représentée par une matrice m de taille n x n consiste à :

- attribuer une variable x_i à la ligne i de m ;
- échelonner m ;
- soit, $j, j+1, \ldots, n$ les lignes à 0, on appelle $x_j, x_{j+1}, \ldots, x_n$ les variables libres. Affecter 1 à x_j et 0 aux autres variables libres, résoudre le système pour trouver les valeurs de $x_1, \ldots, x_{j-1}$. Le vecteur ainsi trouvé est un vecteur du noyau. Recommencer pour les autres variables libres.

Exemple :

$$m=\begin{pmatrix} 1 & 2 & 3 & 4 \\ -1 & 3 & 2 & 1 \\ 2 & 4 & 6 & 8 \\ 0 & 5 & 5 & 5 \end{pmatrix}$$

forme échelonnée $m'=\begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$ variables $x_1, \ldots, x_4$, variables libres x_3, x_4 .

$$m'.\begin{pmatrix} x_1 \\ x_2 \\ 1 \\ 0 \end{pmatrix}=0 \quad \Rightarrow \quad x_1=-1, x_2=-1$$

$$m'.\begin{pmatrix} x_1 \\ x_2 \\ 0 \\ 1 \end{pmatrix}=0 \quad \Rightarrow \quad x_1=-2, x_2=-1$$

D'où une base possible : $\begin{pmatrix} -1 \\ -1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -2 \\ -1 \\ 0 \\ 1 \end{pmatrix}$

Implémenter une commande `nullspace` qui produit une base du noyau à partir d'une matrice.

3. **Moindres carrés**

Voir cours

Ecrire une commande `least_`estimate qui prend en entrée une matrice *n x 2* et qui produit en sortie un vecteur de taille *2* contenant les coefficients de la droite ainsi qu'un vecteur de taille *n* contenant les résidus.