Université de Strasbourg
UFR de Mathématique et d'Informatique
License 2, semestre 4

Projet
Algèbre linéaire appliquée
À soumettre dans un git jusqu'au 24.11.2017

Idée generale

Ce projet est separé dans les parties differentes (les points de chaque étape ne sont pas encore forcement les points de la note finale, mais donnent une idée de l'importance d'un exercice) :

1. Utilisation de Git *(2 point)*.
2. Implementation de la structure matrix *(2 points)*.
3. Manipulations sur la structure *(19 points)* :
   - 3.1. Tests *(1.5 points)*
     - 3.1.1. Matrice carré *(0.5 point)*.
     - 3.1.2. Matrice symmetrique *(1 point)*.
   - 3.2. Operations *(10 points)*
     - 3.2.1. Calcul de la transpose *(0.5 point)*.
     - 3.2.2. Addition de deux matrices *(1.5 points)*.
     - 3.2.3. Multiplication de deux matrices *(2.5 points)*.
     - 3.2.4. Multiplication avec un scalaire *(0.5 point)*.
     - 3.2.5. Mettez un block d'une matrice par rapport à une autre matrice *(1 point)*. Exemple : pour les matrices

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ -1 & -4 & -3 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

     le resultat de `Matrix C = setMatrixBlock(A, 1, 2, B);` (si les indices commencent avec 0, l'appel est `Matrix C = setMatrixBlock(A, 0, 1, B);`) devrait être

$$\mathbf{C} = \begin{bmatrix} 1 & 10 & 20 \\ 2 & 30 & 40 \\ -1 & -4 & -3 \end{bmatrix}$$

     - 3.2.6. Retourner un block d'une matrice *(1 point)*. Exemple : pour la matrice

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \end{bmatrix}$$

     le resultat de `Matrix C = getMatrixBlock(A, 1, 2, 2, 2);` (si les indices commencent avec 0, l'appel est `Matrix C = getMatrixBlock(A, 0, 1, 2, 2);`) devrait être

$$\mathbf{C} = \begin{bmatrix} 3 & 5 \\ 5 & 1 \end{bmatrix}$$

     - 3.2.7. Calcul de la determinante *(3 points)* pour des matrices $\mathbb{R}^{nxn}$.
   - 3.3. Utilisation Pivot de Gauss *(7.5 points)*

3.3.1. Implementation de Pivot de Gauss *(3 points)*.
3.3.2. Calcul du rang d'une matrice *(1 point)*.
3.3.3. Calcul de la determinante *(0.5 point)*.
3.3.4. Resolution d'un system lineaire *(1 point)*.
3.3.5. Calcul de l'inverse *(1 point)* .
3.3.6. Calcul de L et U *(1 point)* .

Si vous arrivez pas de faire une des étapes, n'hésitez pas de les sauter et faire une des exercices suivantes. Cela est possible dans le cas de 3.1., 3.2. et dans 3.3. après avoir fini 3.3.1.. Pour chaque étape (ou chaque function), faites des commits dans le git (une explication se trouve en bas) avec des messages de commits clairs. Je vous encourage de travailler ensemble pour trouver une solution, mais il faut que vous écrivez votre code vous même, pas de copier/coller. Faites des commentaires (c'est mieux de les faire en anglais, mais en francais est bien aussi) qui expliquent votre code. Et ajoutez un ReadMe.md dans votre git qui explique ce que le git fait (un template comme orientation https ://gist.github.com/PurpleBooth/109311bb0361f32d87a2). N'oubliez pas de supprimer les matrices que vous introduisez. Testez toutes les parties avec les exemples listés dans l'exercice suivant et sortez les résultats sur la console.

Exercice 1 Exemples à tester avec votre implementation (dans le main)

Ajouter pour chaque étape un `print` qui explique ce que vous faites, s.v.p..

1. Definissez
$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ -1 & -4 & -3 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 1 & 4 & 2 \\ 2 & 5 & 1 \end{bmatrix}$$
2. Testez si $\mathbf{A}$ et $\mathbf{B}$ sont carrées et symmetrique (reference à 3.1.).
3. Calculez $\mathbf{A}^T, \mathbf{B}^T$ (reference à 3.2.1.).
4. Calculez lorsqu'ils sont définis (reference à 3.2.1.,3.2.2.,3.2.3.,3.2.4.) :
$$\mathbf{A} + \mathbf{B}, \mathbf{B} + \mathbf{A}, \mathbf{AB}, \mathbf{BA}, \mathbf{A}^T\mathbf{B}, \mathbf{B}^T\mathbf{A}, \mathbf{A} + \mathbf{A}^T, 5\mathbf{A}, 3\mathbf{B}$$
5. Testez si $\mathbf{A} + \mathbf{A}^T$ est symmetrique (reference à 3.1.).
6. Calculez `Matrix C = setMatrixBlock(A, 1, 0, B);` (reference à 3.2.5.).
7. Calculez `Matrix D = getMatrixBlock(A, 1, 1, 2, 2);` (reference à 3.2.6.).
8. Calculez det$(\mathbf{A})$ et det$(\mathbf{B})$ lorsqu'ils sont définis (reference à 3.2.7.).
9. Calculez le rang de $\mathbf{A}$ et $\mathbf{B}$ (reference à 3.3.2.).
10. Calculez det$(\mathbf{A} + \mathbf{A}^T)$ avec 3.3.3..
11. Resoudre $\mathbf{Ax} = \mathbf{b}$ (reference à 3.3.4.).
$$\mathbf{b} = \begin{bmatrix} 0 \\ -7 \\ 4 \end{bmatrix}$$
12. Calculez la matrice inverse $\mathbf{A}^{-1}$ de $\mathbf{A}$ (reference à 3.3.5.).
13. Calculez $\mathbf{L}, \mathbf{U}$ tel que $\mathbf{L}$ soit une matrice triangulaire inférieure, que $\mathbf{U}$ soit une matrice triangulaire supérieure, et que $\mathbf{A} = \mathbf{LU}$ (reference à 3.3.6.).

Untilisation de Git

Si vous avez des difficultés avec l'anglais des paragraphes suivantes, n'hésitez pas à me contacter.
Git [1] is one of the most famous SCM – source code management or revision control systems. When talking about SCM, a distant repository is called remote, while the repository on your computer is called local. Git is distributed, i.e. not centralized, it is based on standalone repositories that are local and communicate with remote repositories.

The most famous solution for a remote repository website is https://github.com/ and it often is used as reference when applying for a job position. Since it does not provide private repositories, we are going to use https://gitlab.com/.

Please go on github, make an account and use it to log into gitlab. Then create a new **private** project in gitlab, with the project name USER/ala_prénomNOMDEFAMILLE. Use Settings→Members to add me (chrijopa) to the members of the project as a master. Then use the command line to do the following :

```
# clone your repository
git clone https://gitlab.com/USER/ala_prénomNOMDEFAMILLE.git
# make a first commit (below is just an example)
cd ala_prénomNOMDEFAMILLE
touch main.cpp
git add main.cpp
git commit -m "First commit: added empty main.cpp"
# first push
git push -u origin master
# other pushs will work without additional parameters
git push
```

After the first push, you can use the following to commit your work :

```
# check which files have been changed
git status
# check the difference in the files
git diff
# to see all the commits and the changes in the commits
gitk
# add changed files to the index
git add changedFileName
# commit the index to the local repository
git commit -m "Commit message"
# get the changes performed on the remote repository
git fetch
# incorporate the changes in the local repository
git rebase
# push the local commits to the remote repository
git push
```

For more information, please refer to
https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf
for a cheatsheet containing all commands (that may be necessary for merges).

1. https://git-scm.com/

Structure matrice avec les operations

Implementez la structure suivante

```
// Faites s.v.p. beaucoup des commentaires dans votre code
// Vous pourriez aussi travailler avec des pointers pour le retour d'une matrice
typedef float E;
typedef struct matrix {
E *mat;
int nb_rows, nb_columns;
} Matrix;

// FAITES UN COMMIT PAR FONCTION SVP
Matrix newMatrix(int nb rows, int nb columns);
void deleteMatrix(Matrix A);
// Vous pourriez faire 0<=row<nb_rows ou 0<row<=nb_rows mais il faut d'abord faire un teste si la matrice a la ligne et la colonne existe
E getElt(Matrix A, int row, int column);
void setElt(Matrix A, int row, int column, E val);
print(Matrix);

// Puis les operations : Oubliez pas de faire des tests si les operations sont autorisees
int isSquare(Matrix A); // 0 si faux
int isSymetric(Matrix A); // 0 si faux
// Pour les fonctions suivantes utilisez de preference le set et le get (et pas une manipulation directe du pointer)
Matrix transpose(Matrix A); // Please introduce a new matrix inside of this function, if you want to have the transposed of your matrix, then the matrix name would be transposeD

// Pour les fonctions suivants retournez NULL ou une matrice de la taille 0 x0 si incompatible
Matrix addition(Matrix A, Matrix A);
Matrix multiplication(Matrix A, Matrix A);
Matrix mult_scalar(Matrix A, E alpha);
Matrix setMatrixBlock(Matrix A, int row, int column, Matrix B);
Matrix getMatrixBlock(Matrix A, int row, int column, int nb_rows, int nb_columns);
```

Determinant

Ecrivez une fonction determinant permettant de calculer le determinant d'une matrice carrée en utilisant l'algorithme récursif. Remarque : On écrira une fonction

`Matrix Extraction(Matrix A, int i, int j)`

qui extrait la matrice $A_{i,j}$ de dimension n-1. Aprés, utilisez la formule de developement par linie proposez dans le TD. À la fin votre algorithme devrait utiliser la formule standard pour calculer la determinante d'une matrice 2x2. La determinante d'une matrice 1x1 et le valeur de la matrice a_{11}.

Pivot de Gauss

L'objectif de cette partie est d'implémenter un pivot de Gauss en découpant chaque étape en fonction. Ecrivez une fonction

`Matrix pivotDeGauss(Matrix A, bool upperTriangularMatrix)`

permettant de transformer un système quelconque en système triangulaire supérieur équivalent (si `upperTriangularMatrix` est `true`). Pour `upperTriangularMatrix` egal `false`, le fonction devrait performer un pivot de Gauss complet qui transform les premiers colognes en matrice echeloner. Pensez à décomposer le processus en écrivant, les fonctions suivantes permettant de :

1. Multiplier la ligne i par un facteur k
2. Permuter la ligne i avec la ligne j
3. Additionner à la ligne i le résultat de la multiplication de la ligne j par un facteur k

Après écrivez les fonctions suivants :

```
// Le determinant avec la mthode optimise utilisant la forme triangulaire
// La forme triangulare est obtenu avec pivotDeGauss(Matrix,1)
E determinant2(Matrix A);

// Le rang en utilisant pivotDeGauss(Matrix,1)
int rang(Matrix A);

// La resolution du systeme lineaire
Matrix resolution(Matrix A, Matrix b);
// Cette fonction utilise setMatrixBlock pour declarer une nouvelle matrice
    C = (A|b) et apres elle utilise pivotDeGauss(C,0) pour retourner (I|x).
    Apres il faut utiliser getMatrixBlock pour ecrire x dans une nouvelle
    matrice et la retourner.

// L'inverse
Matrix inverse(Matrix A);
// Cette fonction utilise setMatrixBlock pour declarer une nouvelle matrice
    C = (A|I) avec la matrice d'identite I et apres elle utilise
    pivotDeGauss(C,0) pour retourner (I|A^(-1)). Apres il faut utiliser
    getMatrixBlock pour ecrire A^(-1) dans une nouvelle matrice et la
    retourner.

// La decomposition L et U
void decomposition_L_U(Matrix A, Matrix L, Matrix U);
// Cette fonction utilise setMatrixBlock pour declarer une nouvelle matrice
    C = (A|-I) avec la matrice d'identite I et apres elle utilise
    pivotDeGauss(C,1) pour retourner (U|L_hat). Finalement, la diagonale du
    resultat de L_hat devrait etre multiplier avec -1 pour obtenir L.
```