TP1 : Programmes simples avec Octave

Dans ce TP, nous verrons comment utiliser les fonctionnalités de base d'Octave (matrices, les tests conditionnels, les boucles) afin de faire des programmes.

Lors de ce TP, que vous devez lire attentivement, vous devrez écrire dans Octave toutes les lignes de code qui vous sont données afin de les tester, réaliser toutes les opérations demandées ou suggérées, et répondre à chaque question qui est posée. Vous devez noter (soit sur une feuille, soit dans un fichier) toutes les lignes de commande qui vous paraissaient importantes, ainsi que les réponses aux questions posées pour faciliter vos révisions futures. Aucun rapport ne sera à remettre, dans aucun des TPs.

Il est impératif, si vous ne terminez pas un TP pendant la séance de TP dédiée, de le terminer de votre côté avant la prochaine séance de TP.

1 Manipulation de matrices (20mn)

Comme vu en cours, l'opérateur **:** permet de récupérer des ensembles de lignes et/ou colonnes d'une matrice, et former une sous-matrice. Pour rappel :

```
>> disp(A)
   6 6 2 6 4 2
   7 2 0 3 3 4
   7 6 1 9 7 4
   4 0 7 0 7 6

>> B = A(2:3, 4:6);
>> disp(B)
   3 3 4
   9 7 4
```

On peut aussi récupérer l'intégralité des éléments d'une matrice, sous forme d'un vecteur colonne (les éléments sont placés selon leur numéro d'ordre dans la matrice), en faisant :

```
>> C = B(:);
>> disp(C)
   3
   9
   3
   7
   4
   4
```

Cependant, l'opérateur **:** peut être utilisé en dehors de toute matrice. Regardez par exemple ce code :

```
>> D = 2:8;
>> disp(D)
    2 3 4 5 6 7 8
```

L'opérateur **:** permet en réalité de générer un vecteur ligne de tous les entiers entre le nombre donné à gauche (2), et le nombre donné à droite inclus (8).

On peut rajouter un troisième paramètre pour contrôler le pas d'incrémentation entre les éléments générés. Par exemple, pour obtenir tous les entiers allant de 3 à 30 de 5 en 5, on ferait :

```
>> D = 3:5:30;
>> disp(D)
    3 8 13 18 23 28
```

Ce nouveau paramètre, qui vient s'intercaler entre le nombre contrôlant le début de la liste et le nombre contrôlant la fin de la liste permet de choisir le pas entre chaque élément de la liste résultante. Si aucun pas n'est précisé, alors un pas de 1 est utilisé par défaut.

L'utilisation du symbole **:** génère donc un vecteur d'indices qui est utilisé pour indiquer toutes les lignes que l'on souhaite récupérer dans la matrice **B**. On peut donc aussi utiliser un vecteur ligne afin de récupérer uniquement les numéros de colonnes/lignes de son choix dans une matrice, comme ci-dessous :

```
>> disp(A)
   6 6 2 6 4 2
   7 2 0 3 3 4
   7 6 1 9 7 4
   4 0 7 0 7 6

>> F = A([2 3], [1 3 4 6]);
>> disp(F)
    7 0 3 4
    7 1 9 4
```

Si l'on demande, par erreur, des indices de lignes ou de colonnes non entiers, Octave nous le signale par une erreur :

```
>> G = A(:, 1:0.1:6);
warning: non-integer range used as index
```

> **Questions**
>
> 1. Générez un vecteur ligne contenant tous les nombres réels entre 0 et 1 avec au plus deux chiffres significatifs après la virgule.
> 2. Ecrivez un script qui demande à l'utilisateur deux entiers **n** et **m**, et génère une matrice **M** de nombres aléatoires entre 0 et 100 de taille $\mathbf{n} \times \mathbf{m}$. A la suite de votre script, générez une sous matrice **N** avec uniquement les éléments de **M** situés à un indice de ligne et de colonne pair. Votre code doit fonctionner quel que soit la taille de M.
> 3. Ecrivez un script qui demande à l'utilisateur un entier **n** et génère un vecteur ligne **V** de taille **n** de nombres aléatoires entre 6 et 10. Ensuite, générez un vecteur **W** qui est le miroir de **V** (le premier élément de **V** est à la dernière place de **W**, le second élément de **V** est à l'avant dernière place de **W**, etc) sans utiliser le mot clef **flip**.

2 Le test conditionnel if (20mn)

Le mot clef **if** permet d'exécuter du code seulement si une condition est vraie :

```
x=-2;
if x<0
    disp('x est negatif')
end
disp('FIN')
```

Ce code affiche, si x possède une valeur négative, que x est négatif.

Le mot clef **else**, qui doit impérativement suivre un **if**, permet aussi d'exécuter du code si la condition du if n'a pas été respectée :

```
x=-2;
y=0;
if x<0
    disp('x est negatif')
else
    disp('x est positif')
end
disp('FIN')
```

Ici, le programme, en plus d'afficher que x est négatif s'il contient une valeur négative, affichera aussi que x est positif s'il contient une valeur positive.

Le mot clef **elseif** permet d'exécuter du code si la condition du **if** n'a pas été respectée, en y ajoutant une condition :

```
x=-2;
if x<0
    disp('x est negatif')
elseif x==0
    disp('x est nul')
else
    disp('x est positif')
end
disp('FIN')
```

Ici, le programme affichera que x est positif, nul ou négatif selon sa valeur.

> **Questions**
>
> 1. Ecrivez un script qui demande un nombre à l'utilisateur et affiche sa valeur absolue.
> 2. Ecrivez un script qui demande un nombre **x** à l'utilisateur entre 0 et 1. Si l'utilisateur se trompe et que **x** est négatif, on remplace sa valeur par zéro, et s'il est supérieur à 1, on remplace sa valeur par 1.
> 3. Ecrivez un script qui demande deux nombres à l'utilisateur, et affiche le plus grand des deux.
> 4. Ecrivez un script qui demande un nombre à l'utilisateur et affiche BINGO si ce nombre est entre 3 et 4.

3 Le test conditionnel sur une matrice (20mn)

Nous avons vu en cours qu'il est possible en Octave et Matlab de faire un test conditionnel sur chaque élément d'une matrice.

Par exemple, pour trouver tous les éléments d'une matrice **A** strictement supérieurs à 6, on fera :

```
>> A=[7 5 8 8 3;7 1 8 4 7;1 2 1 6 6;7 4 8 1 8];
>> disp(A)
         7 5 8 8 3
         7 1 8 4 7
         1 2 1 6 6
         7 4 8 1 8

>> F = A(A > 6);
>> disp(F)
   7
   7
   7
   8
   8
   8
   8
   7
   8
```

Ici, on récupère dans le vecteur colonne **F** les valeurs des éléments de **A** qui sont supérieurs à 6. En utilisant la fonction **find**, on peut récupérer les positions (sous forme de numéro d'ordre) des éléments de **A** supérieurs à 6 :

```
>> E = find(A > 6);
>> disp(E)
    1
    2
    4
```

```
   9
  10
  12
  13
  18
  20
```

Si on souhaite faire une opération uniquement sur ces éléments (par exemple, leur ajouter un), on fera :

```
>> A(A>6) = A(A>6) + 1;
>> disp(A)
          8 5 9 9 3
          8 1 9 4 8
          1 2 1 6 6
          8 4 9 1 9
```

Ici, tous les éléments de **A** supérieurs à 6 ont bien été augmentés de 1. On peut aussi faire :

```
>> A(E) = A(E) + 1;
```

Questions

1. Etant donné une matrice de nombres aléatoires entre -1 et 1 de 5 lignes et 7 colonnes, affichez combien d'éléments supérieurs ou égaux à 0 elle possède. Afin de générer une telle matrice, on pourra faire :

   ```
   A = rand(5,7)*2-1;
   ```

   Assurez-vous de bien comprendre pourquoi cette ligne de code permet de générer une matrice de nombres aléatoires entre -1 et 1, de 5 lignes et de 7 colonnes.
2. Réalisez, sur la matrice précédente, l'opération de valeur absolue sans utiliser la fonction **abs**.

4 La boucle for dans Matlab (30mn)

4.1 La boucle for sur les vecteurs

Commencez par écrire un programme très simple qui parcourt un vecteur ligne :

```
v = rand(1, 20);
for i=v
       disp(`La variable i vaut `);
       disp(i);
end
```

Ce programme parcourt, à l'aide de la variable i et de la boucle, les différentes colonnes de v. Comme v est un simple vecteur ligne, on se retrouve à parcourir les éléments de v.

Remarquez que, si le vecteur v n'était pas un vecteur ligne, mais un vecteur colonne, la boucle ne parcourant que les colonnes de v, nous n'aurions qu'un seul "tour de boucle" (avec un seul message s'affichant à l'écran, et tout un vecteur de valeurs apparaissant ensuite) :

```
v = rand(1, 20)';
for i=v
       disp(`La variable i vaut `);
       disp(i);
end
```

La boucle for, comme expliqué dans le cours (les parties supplémentaires à connaître, disponibles sur Moodle), s'écrit donc de la façon suivante :

```
for variable in matrice
...
end
```

La variable prendra tour à tour comme valeur les différentes colonnes de la matrice (ou du vecteur, un vecteur étant considéré comme une matrice spécifique).

Si on souhaite afficher les éléments d'un vecteur ainsi que leur position (un message du genre "La case 4 contient 67, La case 5 contient 53, ..."), il faut agir différemment en parcourant un vecteur qui contiendra les différentes valeurs de position des éléments du vecteur. Dans le cas du vecteur v, qui contient 20 éléments, les positions sont des entiers allant de 1 à 20. On commencera donc par générer un vecteur ligne e contenant tous les entiers de 1 à 20 à l'aide de la commande :

```
e = 1:20
```

Ensuite, on parcourt ce vecteur à l'aide d'une boucle for :

```
for i = e
```

La variable i prendra comme valeur les éléments de e, c'est à dire les entiers de 1 à 20. On peut donc utiliser i comme un index et faire, dans la boucle :

```
        disp(`La case `);
        disp(i);
        disp(`contient `);
        disp(v(i));
```

On peut tout résumer ainsi :

```
for i=1:numel(v)
        disp(`La case `);
        disp(i);
        disp(`contient `);
        disp(v(i));
end
```

> **Questions**
>
> 1. Comment faire pour afficher les éléments de **v** du dernier au premier ? *Indice : regardez la fonction* ***flip***.
> 2. Comment faire pour afficher les éléments de **v** du dernier au premier, en précisant leur position ?

4.2 La boucle for sur les matrices

Voici un programme réalisant une boucle sur une matrice :

```
A = rand(5, 7);
disp(A);
for i=A
        disp(`La variable i vaut `);
        disp(i);
end
```

Ce programme n'affiche pas chaque élément de A, mais chaque colonne de A. En effet, lorsqu'on parcourt une matrice ou un vecteur avec une boucle, ce sont en réalité les colonnes de l'élément que l'on parcourt. Si on souhaite parcourir chaque élément de la matrice A individuellement, on devra ensuite parcourir les éléments du vecteur i en le transposant :

```
for i=A
        for j=i'
                disp('j vaut')
                disp(j)
        end
end
```

Questions

1. Comment faire pour afficher tous les éléments de **A** un par un en précisant leur coordonnées dans la matrice ? Pour information, vous pouvez utiliser la fonction **printf**, comme en C, pour afficher un message composé sur une seule ligne.
2. Comment faire pour afficher tous les éléments de **A** un par un en précisant leur numéro de ligne et leur numéro de colonne ? On pourra commencer par récupérer le nombre de lignes et de colonnes de la matrice en faisant :

```
s = size(A);
nb_ligne = s(1);
nb_colonne = s(2);
```

   On peut également écrire :

```
[nb_ligne, nb_colonne] = size(A);
```

   On peut aussi, sur les versions plus récentes de Matlab et d'Octave, faire

```
nb_ligne = size(A)(1);
nb_colonne = size(A)(2);
```

3. Comment faire pour afficher tous les éléments de **A** un par un à l'aide d'une seule boucle **for** ? Comment afficher les éléments colonne par colonne, ou ligne par ligne ? *Indice : Il faut trouver un moyen de transformer la matrice* ***A*** *en un vecteur ligne. Regardez ce que produit ce code :*

```
B = A(:)
```

5 La boucle while dans Matlab (20mn)

La boucle **while** de Matlab permet de répéter un morceau de code tant qu'une condition n'est pas satisfaite. Par exemple, voici un programme permettant de demander un nombre positif à l'utilisateur, et de répéter ces instructions tant qu'elles n'auront pas été suivies :

```
a = input('Entrez un nombre positif : ');
while a<0
    a = input('Entrez un nombre positif : ');
end

disp(a);
```

La boucle **while** permet donc de répéter des instructions tant qu'une condition donnée est vraie. Si la condition devient fausse, alors la boucle s'arrête et le programme passe à la suite du code.

Testez ce code :

```
a=rand(1,1);
b=1;

while a<30
    b = b+1;
end
```

Si l'exécution de ce code vous parait longue, c'est normal : le code de la boucle **while** ne fait pas évoluer la condition qui restera pour toujours vraie. La boucle va donc tourner à l'infini. Pour interrompre votre programme, faîtes CTRL+C dans la fenêtre de commande. Il est donc très important, dans une boucle **while**, que le code de la boucle fasse évoluer la condition.

Questions

1. Ecrivez un programme qui calcule l'approximation de $\pi^2/8$ à l'aide de cette série convergente :

$$1^2 + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + ...$$

Votre programme doit arrêter son calcul lorsque la somme n'évolue plus beaucoup, c'est à dire que ce que l'on rajoute à la somme est inférieur à un millionième de la somme totale.

2. Ecrivez un programme qui, à l'aide d'une boucle **while**, teste si un nombre a est premier (il n'est divisible que par 1 et par lui-même). Pour ce faire, testez tous les nombres entre $a-1$ et 2, et voyez s'ils divisent a.

6 Vectorisation de code (60mn)

Dans Matlab, les boucles sont en général assez peu optimisées, ce qui signifie qu'elles s'exécutent lentement par rapport à ce que l'on pourrait obtenir dans d'autres langages de programmation. On cherche donc, en Matlab, à vectoriser son code le plus possible, c'est à dire à éviter des boucles et à appeler des opérations sur les matrices ou les vecteurs pour effectuer une opération de façon plus rapide.

Voici des exercices où vous devrez répondre à la question posée en proposant un code vectorisé, c'est à dire un code sans boucle...

1. Etant donné une matrice **A** de taille 3000 par 3000, générée avec des nombres aléatoires entre 0 et 1, calculez le pourcentage d'éléments de **A** qui sont supérieurs ou égaux à 0.5. Ces résultats sont-ils en accord avec le fait que **rand** réaliser une distribution uniforme de nombres entre 0 et 1 ?
2. Ecrivez un script qui, étant donné un nombre **n** donné par l'utilisateur, génère la matrice identité, sans utiliser les fonctions **eye** ou **diag**. Pour réussir cet exercice, tracez sur une feuille la matrice identité de taille 5x5, et regardez les numéros d'ordre des 1 : suivent-ils une certaine progression ?
3. Ecrivez un script qui demande une hauteur et une largeur à l'utilisateur, et génère une matrice aléatoire **A** de cette taille. Ensuite, construisez la matrice **B** qui est le miroir de **A** sans utiliser le mot clef **flip**. Par exemple, on doit avoir

```
>> disp(A)
   3 3 3 6 8 1
   9 5 4 6 3 6
   4 8 6 2 5 2
   2 6 7 8 3 6

>> disp(B)
   1 8 6 3 3 3
   6 3 6 4 5 9
   2 5 2 6 8 4
   6 3 8 7 6 2
```

4. Ecrivez un script qui génère une matrice carrée A avec des nombres aléatoires entre 0 et 1, et force à 0 tous les nombres sous la diagonale de la matrice, sans utiliser les fonctions **trilu** ou **trill**. Pour vous aider, voici une idée : construisez deux matrices B et C de la même taille que A : la première contiendra, pour chaque élément de A, son numéro de colonne, et la seconde contiendra son numéro de ligne. On aura donc $B(i,j) = i$ et $C(i,j) = j$, pour tout i et j. La fonction **repmat** vous sera utile pour construire B (C se construit facilement à partir de B).
5. Ecrivez un script qui demande une hauteur et une largeur à l'utilisateur, et génère une matrice aléatoire **A** de cette taille. Ensuite, construisez une matrice **M** qui est la plus grande matrice carrée extraite des éléments en haut à droite de **A**. Par exemple, on doit avoir

```
>> disp(A)
         9 8 5 5 1
         7 5 2 4 0
        10 4 8 9 9

>> disp(M)
         5 5 1
         2 4 0
         8 9 9
```

6. Ecrivez un script qui génère un vecteur ligne **V** de nombres aléatoires entre 0 et 10, possédant **7n** éléments. Puis, calculez dans un vecteur ligne **S** la somme partielle, par paquets de 7 éléments, du vecteur **V**. Le premier élément de **S** est donc la somme des 7 premiers éléments de **V**, le second élément de **S** est la somme des sept éléments suivants de **V**, etc. Par exemple :

```
>> disp(v)
    3 1 4 1 3 3 4 5 3 1 1 1 4 1

>> disp(s)
    19 16
```

*Les fonctions **reshape** et **sum** pourraient vous être utiles.*