TP 2 : Les boucles avec Matlab

Dans ce TP, nous verrons l'utilisation des boucles (**for et while**) dans Matlab, et nous écrirons des programmes plus complexes pour avoir des comportements toujours plus intéressants.

Vous devez avoir terminé le TP1 avant de commencer ce TP.

1 La boucle for dans Matlab

1.1 La boucle for sur les vecteurs

Pour commencer, nous allons nous familiariser avec la boucle **for**. Commencez par écrire un programme très simple qui parcourt un vecteur ligne :

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

Ce programme est sensé afficher tous les éléments de **v** à travers la variable **i** dans la boucle.

Questions

1. Comment faire pour afficher les éléments de **v** du dernier au premier ?
   Indice : regardez la fonction ***flip****.*
2. Comment faire pour n'afficher que les éléments de **v** d'indice pair ?
3. Nous aimerions modifier l'affichage du précédent programme afin qu'il n'affiche pas le message *La variable i vaut*, mais plutôt *Voici ce que contient la case ....* On veut donc à chaque tour afficher le numéro de la case et son contenu.

1.2 La boucle for sur les matrices

Nous allons faire le même travail, mais avec une matrice. Voici un programme permettant de parcourir une matrice :

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

Ce programme affiche-t-il tous les éléments de **A** un par un ?

Questions

1. Comment faire pour afficher tous les éléments de **A** un par un à l'aide de deux boucles **for** ?
2. Comment faire pour afficher tous les éléments de **A** un par un à l'aide d'une seule boucle **for** ? Comment contrôler si on souhaite afficher d'abord les éléments par colonne, ou d'abord ceux par ligne ?
   Indice : Comment faire pour transformer la matrice ***A*** *en un vecteur ?*

1.3 Les performance de la boucle for

Parcours avec for

Nous allons voir maintenant que l'utilisation d'une boucle **for** ralentit considérablement un programme. Pour cela, nous allons essayer de réaliser la somme pondérée des éléments de deux vecteurs de même taille. On commence par déclarer les deux vecteurs et une variable qui servira au calcul :

```
v = rand(1,100000);
w = rand(1,100000);
s=0;
```

Ensuite, on utilise une boucle **for** afin de parcourir les deux vecteurs, et un indice pour parcourir l'autre :

```
i=1;
for k = v
       s = s + k*w(i);
       i=i+1;
end
```

> **Questions**
>
> 1. A l'aide des mots clefs **tic** et **toc**, mesurez le temps d'exécution de ce programme. La valeur de **s** que vous obtenez est-elle en accord avec le fait que la fonction **rand** réalise une distribution uniforme entre 0 et 1 ?
> 2. Réalisez une vectorisation de ce code, afin de vous débarrasser totalement de la boucle. Les performances obtenues sont-elles meilleures ?

La différence de performance pourrait venir du parcours que l'on fait, où l'on est obligé de ne parcourir qu'un seul des deux vecteurs, et utiliser une variable qui grandit de un en un pour parcourir l'autre vecteur. On pourrait parcourir les deux vecteurs en même temps, en les fusionnant dans une matrice de deux lignes :

```
for k = [v;w]
       s = s + k(1)*k(2);
end
```

> **Questions**
>
> Ce nouveau code est-il plus performant que l'ancienne version de la boucle **for** ? Est-il plus performant que la version vectorisée ?

Préallocation

Nous allons chercher à calculer, étant donné $a \in \mathbb{R}^+$, les termes de la suite

$$\begin{cases} u_0 = 1 \\ u_n = 0.5(u_{n-1} + a/u_{n-1}) \end{cases}$$

Pour ce faire, nous proposons ce code :

```
a=16;
lim = 1000000;

un = 1;
for k = [2:lim]
       un_1 = un;
       un = 0.5*(un_1 + a/un_1);
end
```

Pouvez-vous conjecturer la limite de la suite pour différentes valeurs de **a** ? Quel est le temps de calcul de votre programme ?

Afin de rendre l'étude plus intéressante, nous souhaitons stocker les valeurs successives de la suite dans un vecteur. Nous modifions ainsi notre code :

```
tic;
a=16;
lim = 1000000;
h(1) = 1;

for k = [2:lim]
       h(k) = 0.5*(h(k-1) + a/h(k-1));
end
toc;
```

Questions

1. **La toute première fois** que vous exécutez ce code, va-t-il plus vite ou moins vite que l'autre version du code ? L'écart est-il important ?
2. Si vous exécutez ce code une seconde fois, est-il plus rapide ?
3. Afin de comprendre pourquoi ce code était bien moins rapide la première fois, regardez ce qu'il se passe pour le vecteur **h** à chaque tour de boucle (quelle sera sa taille) ? Ceci explique-t-il pourquoi, la seconde fois, votre code était plus rapide ?
4. Pour éviter cela, comment allouer dès le départ une certaine taille au vecteur **h** ? Après avoir supprimé la variable **h** de la zone des variables, et avoir effectué les modifications nécessaires à votre code, testez votre nouvelle version : s'exécute-t-telle plus vite ?

Il est important d'éviter de faire grandir la taille de matrices ou de vecteurs dans une boucle, et de recourir à la pré-allocation mémoire des tableaux (avec des fonctions comme **zeros** ou **ones**) avant le début de la boucle. En effet, à la chaque agrandissement de tableau, Matlab peut se retrouver contraint d'allouer une nouvelle place et de recopier tout l'ancien tableau dans le nouveau.

1.4 Quelques exercices

Questions

1. A l'aide d'une boucle **for**, réalisez un programme qui réalise l'histogramme demandé au TP1, question 4.6. Comparez les performances de ce code avec le code vectorisé précédent.
2. Ecrivez une fonction de tri d'un vecteur. Pour ce faire, recherchez le plus grand élément du vecteur, puis placez-le à la fin du vecteur (vous pouvez utiliser la fonction **max**. Recommencez la même opération en ne recherchant le plus grand élément que sur les $n-1$ premiers éléments (où n est la taille du vecteur), etc. Comparez ensuite vos performances à celles de la fonction **sort**.

2 La boucle while dans Matlab

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 choisit un nombre entier au hasard entre 1 et 100 (à l'aide de **randi**), et doit le faire deviner à l'utilisateur. Pour ce faire, tant qu'il n'a pas trouvé le nombre, l'utilisateur doit entrer un nombre et le programme lui indique si ce nombre est plus petit ou plus grand.
2. 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.
3. 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.
4. On veut savoir combien faut-il, en moyenne, faire de lancers de dés 6 faces, afin d'obtenir un score de 20 ou plus. Pour ce faire, on lancera des dés virtuels avec la fonction **randi** jusqu'à avoir obtenu un score de 20. Nous répéterons cette expérience cent mille fois, et à chaque fois, nous enregistrerons le nombre de lancers de dés qu'il a été nécessaire de faire dans un vecteur qui servira d'histogramme.

   Ensuite, dans la zone des variables, vous pourrez visualiser votre vecteur sous forme d'un graphe en faisant un clic droit dessus, puis en choisissant l'option **plot**. Quel est le nombre de lancers qui est, statistiquement, celui qui réussit le mieux à cette épreuve ? Est-ce en accord avec la valeur moyenne d'un dé ?
5. Ecrivez un programme qui, étant donné un nombre **a** entre 0 et 1, recherche et affiche le premier élément d'un vecteur **v** (généré avec la fonction **rand**) qui est plus grand que **a**. Votre programme doit fonctionner même s'il n'y a aucun élément de**v** plus grand que **a**.

3 Quelques exercices

Questions

1. Réalisez un programme qui peut échanger deux colonnes d'une même matrice sans utiliser de variables intermédiaires.
2. Proposez une version vectorisée du programme qui, étant donné un nombre **a** entre 0 et 1, recherche et affiche le premier élément d'un vecteur **v** qui est plus grand que **a**.

   Ce programme est-il plus rapide si **a** est grand ou petit ?
3. Dans le même esprit, vectorisez le code permettant de savoir si un nombre est premier : ce code est-il toujours plus rapide que l'autre version (testez dans le cas d'un nombre bel et bien premier, et dans le cas d'un nombre non premier).
4. Toujours dans le même esprit, vectorisez la fonction pgcd vue pendant le cours.
5. Ecrivez un jeu de Mastermind. Un joueur choisit une combinaison de 4 chiffres entre 1 et 9, qu'il garde secrète, et qui ne contient aucun doublon. Un autre joueur doit deviner cette

séquence dans le bon ordre. Pour ce faire, il peut réaliser jusqu'à 6 propositions de 4 chiffres. On lui affiche clairement, après chaque proposition, quel chiffre est absent de la véritable combinaison, quel chiffre est bien placé, et quel chiffre est présent mais mal placé. Le jeu s'achève si les 6 propositions ont échoué ou si l'une d'elles a réussi.