Introduction à Matlab pour l'agrégation de mathématiques :

Option probabilités et statistiques

Pour survivre :

Ouvrir une fenêtre de commande Matlab et taper **help** après l'invite de commande et appuyer sur **F1** qui est le raccourci clavier pour l'aide. Pour obtenir de l'aide sur une fonction, utiliser le moteur de recherche disponible ou taper dans la fenêtre de commande :

```
help nom_de_la_fonction
```

Naviguez à votre guise sur l'une ou l'autre des aides... Obligez vous à parcourir les noms des différentes fonctions disponibles après cette introduction afin de vous familiariser avec Matlab.

Premiers calculs :

La fenêtre de commande permet d'utiliser Matlab comme une calculatrice. Matlab est comme son nom l'indique (MATrixLABoratory) un outil performant pour le calcul matriciel, c'est aussi un langage de programmation dirigé vers les calculs scientifiques. Le principal type d'objet utilisé est une matrice rectangulaire réelle ou complexe. Les opérations de Matlab sont essentiellement des opérations matricielles. **En général, il faudra préférer une expression matricielle des calculs.**

Les matrices et leurs opérations élémentaires :

Il y a plusieurs syntaxes équivalentes pour saisir les matrices et aussi des petits pièges syntaxiques qui en découlent. Le ; à la fin de la définition d'une variable permet de ne pas afficher la variable. Entrez les lignes de commandes ci-dessous et modifiez les selon vos idées.

```
>> vec_ligne = [1,2,3]
>> vec_colonne = [1;3;4]
>> matrice_SDP = [2 1 1
1 2 1
1 1 2];
>> size(matrice_SDP)
>> length(matrice_SDP)
>> prod_scal = vec_ligne * matrice_SDP * vec_colonne
>> norm2 = vec_ligne * matrice_SDP * vec_ligne'
>> sin(vec_ligne .* vec_ligne)
>> 3./(vec_ligne ./ vec_ligne)
>> A=ones(2);A(2,:)=zeros(1,2)
>> A * (ones(2)-A) .* A == A * ((ones(2)-A) .* A)
>> ans ~= A
>> B= 3:3:36;reshape(B,3,4);diag(matrice_SDP)*B
```

```
>> matrice_SDP(3,:)=[]
>> A=cumsum(2 .* rand(100,10)>ones(100,10),1)./100; A(100,:)
>> var(1:3)
>> randn(3,3)
>> mean(exp(cos(1:10)))
>> s= linspace(1,10,300);
>> plot(s,cos(s),'ro')
>> who
>> clear all
>> who
>> clc
```

Pour accélérer votre utilisation de la fenêtre de commande, pensez aux raccourcis clavier comme le pavé de flèches qui permet de réécrire directement les commandes précédemment tapées. La touche "TAB" permet de compléter une amorce de mot parmi l'ensemble des noms en mémoire.

Exercice 1 *Rechercher la fonction qui donne la liste des valeurs propres d'une matrice (utiliser pour cela la fonction lookfor) et l'appliquer aux matrices suivantes :*

```
matrice_SDP = [2 1 1;1 2 1;1 1 2]
repmat(matrice_SDP,2)
```

Exercice 2 *Construire une matrice* $(10, 15)$ *de tirages indépendants de gaussiennes standards et en extraire la matrice des coefficients dont l'indice de ligne est pair et l'indice des colonnes est impair.*

Exercice 3 ***une astuce pour créer une matrice :*** *l'indexation multiple.* $z = [1 : 3]$, $a = z(ones(1, 4), :)$ *permet de créer une matrice de* 4 *lignes identiques à* z*. Utiliser la même astuce pour créer une matrice de colonnes identiques.*

Utiliser Matlab comme un éditeur de programmes

La plupart du temps, vous allez taper des lignes de commande dans l'éditeur pour ne pas avoir à les saisir à chaque utilisation. Avec l'éditeur, on peut écrire des scripts ou des fonctions. Vous pouvez utiliser un autre éditeur de texte que celui de Matlab, dans ce cas, il faudra enregistrer ce fichier avec l'extension .m dans le répertoire de travail. Pour connaître le répertoire de travail, on peut utiliser la commande *cd* dans la fenêtre de commande. Pour donner un nom à ces fichiers, utiliser des lettres non accentuées ou des chiffres. Eviter de donner le même nom à vos fonctions que des fonctions déjà intégrées dans Matlab sous risque de conflits.

Exercice 4 *Ouvrir un éditeur Matlab (File − > New − > M File) et inscrire*

```
tic
n=input('entrez un chiffre')
toc
disp(['Vous avez mis' num2str(toc) 'seconde(s) a peu pres pour vous decider'])
```

Enregistrer ce fichier sous le nom de votre choix, l'exécuter à partir de l'éditeur (onglet debug) et à partir de la fenêtre de commande en saisissant le nom du fichier sans son extension.

Fichiers fonctions :

L'éditeur vous permet d'enregistrer des fonctions que vous pouvez réutiliser dans vos scripts s'ils sont tous enregistrés dans le même répertoire courant. On enregistre le fichier sous le nom de la fonction. Par défaut, l'éditeur propose de l'enregistrer sous le nom de la première fonction définie. On peut définir d'autres fonctions à la suite de la première fonction mais celles-ci ne seront pas accessibles de l'extérieur, ce qui a un intérêt pour clarifier le code. (codage de fonctions qui servent uniquement à la fonction principale.)

```
function [r]=factoriel_3(n)
% c'est un simple calcul de factoriel
r=prod(1:n);
```

Vous pouvez appeler la fonction dans la fenêtre de commande, taper dans la fenêtre lookfor("simple calcul")...

Boucles itératives, fonctions logiques et boucle while :

La syntaxe de la boucle for est la suivante :

```
for <var>=<matrice>
<ins_1>;
<ins_2>;
<ins_3>;
...
<ins_n>;
end;
```

Un exemple d'itération sur une matrice :

```
z=repmat([1,2;1,2],2);
m=ones(4,1)
for i=z
m=m+max(cumsum(i),i)
end;
```

On remarque que l'itération se fait sur les colonnes de la matrice. La variable *var* prend donc successivement comme valeur chaque colonne de la variable *matrice*. A la différence de certains langages, l'indentation (alinéa dans les boucles après le for par exemple) n'est pas nécessaire. C'est une pratique qui permet cependant une meilleure lisibilité du code.

La syntaxe du groupement $if \ldots end$:

```
if <condition_1>
<ins_1>;
```

```
elseif <condition_2>
<ins_2>;
...
elseif <condition_n>
<ins_n>;
else
<final_ins>;
end;
```

Un exemple de simulation d'une variable aléatoire par mélange :

```
pile=floor(2*rand(1))
if pile
A=randn(1)
else
A=rand(1)
end;
```

Avec la structure if, il est utile de connaître quelques opérateurs relationnels et logiques : Outre les bien connus $>, <, =<, >=, ==$, on a vu dans les premières manipulations $\sim=$ pour la négation de $==$. Evidemment, tous s'appliquent aux matrices et renvoient des matrices de 1 et 0 selon que la relation testée est vraies ou fausse. A noter, le ET logique &, le OU logique |, et la négation : ~
Ce dernier est un opérateur unaire (un seul argument) et non binaire comme les précédents.

La syntaxe de l'instruction *while* :

```
while <expression>
<ins_1>;
<ins_2>;
<ins_n>;
end;
```

Tant que $< expr >$ est non nulle les instructions sont exécutées. L'exemple suivant illustre le fait qu'on peut ne pas mettre un booléen après *while* :

```
function [a]=fact(n)
a=1;
while n
a=a*n;
n=n-1;
end;
```

Il faut signaler que l'instruction *break* permet de sortir de la boucle en cours et l'instruction *return* permet de sortir de la fonction. L'instruction $error('blabla')$ interrompt l'exécution du programme et affiche l'erreur 'blabla'.

Exercice 5 *Programmer dans des fichiers fonctions trois façons différentes le calcul de $n!$ et écrire un script pour observer les performances de calculs de chaque fonction. (indication : Matlab peut être utilisé pour la récursion)*
Comparer à cette occasion la différence entre le while et le for, et le for à une écriture vectorialisée.

On voit donc tout l'intérêt de privilégier les formulations vectorielles plutôt que l'utilisation de boucles : les performances en temps de calcul sont souvent améliorées ainsi que la lisibilité du code.

Remarques générales sur la rédaction code :

Vous aurez nécessairement à débuguer vos codes. Pour simplifier le débugage et vous assurer que votre code réalise bien sur des exemples ce que vous en attendez, on peut conseiller de rédiger un commentaire dans l'éditeur de texte pour les fonctions, boucles et autres points sensibles de votre code. Pour insérer un commentaire dans un script, faites commencer votre commentaire par %. Les caractères après ce symbole ne seront pas interprétés par Matlab.
Morceler son code en briques élémentaires permet souvent de le clarifier et aussi de le tester de manière bien plus robuste et fiable. Attention toutefois à ne pas trop morceler le code, sous peine de diminuer la lisibilité du code.
Ne pas oublier que le jury d'agrégation peut regarder vos codes : il faut de manière générale garder à l'esprit lors de la rédaction du code que celui-ci doit pouvoir être facilement compris par d'autres programmeurs. (En particulier, je vous conseille de ne pas chercher à compliquer le code !)

Outputs, inputs, et représentation des résultats

Enregistrer ses résultats est important pour présenter plus rapidement au jury vos conclusions, surtout si le temp d'exécution est important. Parfois, il se peut qu'un résultat intermédiaire nécessite beaucoup de temps de calcul, et si ce résultat peut être réutilisé, on a intérêt à le stocker. Exemple :

```
>> ma_variable_1=[5.03]
>> ma_variable_2=['grr']
>> save mes_variables.txt ma_variable_1 ma_variable_2 -ascii
>> save mes_variables ma_variable_1 ma_variable_2
>> clear
>> what
>> load mes_variables.mat
>> who
>> autre_variable=[0:3:30]
>> save('mes_variables','autre_variable')
>> clear
>> load mes_variables.mat
>> who
```

Exercice 6 *En s'inspirant de l'exemple précédent, stocker* 10^7 *tirages indépendants de gaussiennnes centrées réduites.*

On peut aussi enregistrer les structures qui rappellent de loin la programmation orientée objet, ce qui peut être agréable pour leur manipulation.

```
>> Nicolas.taille=1.68
>> Nicolas.sexe='masculin'
>> Nicolas.age='55ans'
>> save fiche_technique.mat -struct NicolasS;
```

```
>> clear
>> load fiche_technique.mat
>> who
```

Matlab est un outil qui permet de générer facilement de beaux graphiques. Il est fortement conseillé (notamment le jour de l'oral) d'accompagner les résutats numériques de sorties graphiques, si celles-ci peuvent faciliter la compréhension du message. La fonction *plot* est le mot qu'on ne doit pas oublier. D'autres fonctions sont utiles pour l'option A comme *hist* ou *bar*, qui permet de tracer des histogrammes. L'exemple ci-dessous donne la distribution du vecteur y selon le vecteur x et souligne un faux ami : *histc*.

```
>> x = -5:0.1:5;
>> y = randn(50000,1);
>> hist(y,x)
>> rep=cumsum(histc(y,x));
>> figure(2),bar(x,rep)
```

On peut aussi effectuer des tracés $3D$ avec la fonction $plot3$, ou $surf$ ...

```
[X,Y]=meshgrid(-9:0.2:9,-2:0.1:2);
Z = X .* exp(-0.5*X.^2 + Y.^2)+3*Y;
plot3(X,Y,Z)
figure(2),surf(X,Y,Z)
hold on
Z = X .* exp(-0.5*X.^2 + Y.^2)-3*Y;
surf(X,Y,Z)
hold off
pause(3)
surf(X,Y,Z)
figure(3),contour(X,Y,Z)
```

Exercice 7 *Utiliser les tirages de gaussiennes précédemment enregistrés et représenter sous forme d'histogramme la réparition de $(V_i - 5)_+$ où V_i représente un tirage de gaussienne et x_+ est la partie positive de x. En calculer l'espérance et la variance.*

Pour agrémenter les graphes et les rendre toujours plus lisibles : donner un titre au graphique, nommer les axes, mettre des couleurs... (Le jour de l'oral, c'est plus facile de lire directement sur le graphe plutôt que de rechercher à quoi correspond le graphe.) Dans l'exemple ci-dessous, remarquer la sauvegarde (sous l'extension par défaut $.fig$) d'un graphe et son ouverture.

```
X=linspace(0,1,1000);
Y=exp(-X);
plot(X,Y)
title('trace de la fonction exp(-x)','fontsize',20)
xlabel('axe des abscisses')
ylabel('axe des ordonnees')
hgsave('mafigure')
pause(3)
```

```
close all
pause(2)
openfig('mafigure')
```

Enfin, pour aller plus loin dans l'illustration des résultats, les commandes moviein, getframe et movie peuvent être utilisées pour donner une animation du résultat. Ceci peut être intéressant (amusant ?) si le problème fait intervenir des quantités dépendant du temps.

Exercice 8 *Soit une chaîne de Markov à nombre d'état finie et irréductible. Représenter dynamiquement la convergence d'une mesure de proba initiale quelconque vers l'unique probabilité invariante.*

Utilisation des connaissances sur des exemples :

Exercice 9 *Un peu d'analyse pour finir et autre...*

1. Question :

Proposer et implémenter un schéma numérique pour la résolution d'équations différentielles ordinaires.

2. Question :

Implémenter une recherche dichotomique.

Exercice 10 *Illustration de la loi de Wigner*

1. Question :

Générer des matrices hermitiennes de taille n dont les coefficients diagonaux suivent une loi normale $N(0,1)$, et des coefficients extradiagonaux $X_d^{j,j}$ de parties réelles $X_r^{j,k}$ et imaginaires $X_{im}^{j,k}$ de loi $N(0,\frac{1}{2})$. Sur la partie triangulaire supérieure, on suppose que ces variables aléatoires sont indépendantes entre elles. ($X_d^{j,j}$, $X_r^{j,k}$ et $X_{im}^{j,k}$ pour $0 < j < k \leq n$). Rédiger une fonction correspondante qui prend comme argument une fonction qui génère une variable aléatoire.

2. Question :

Soit M une matrice générée à la question 1. On note $\tilde{M} = \frac{1}{\sqrt{n}}M$. Ecrire un script qui prend comme argument n la taille de la matrice et N un entier qui représente le nombre de tirages de matrice et qui donne la répartition des valeurs propres de tirages indépendants de matrices $\tilde{M}$.

3. Question :

Tracer la densité $f(x) = \frac{1}{2\pi}\sqrt{4-x^2}$ sur $[-2,2]$ et nulle en dehors de cet intervalle. Que constate-t-on ?
Que se passe-t-il si on change la variable gaussienne en une variable centrée réduite ?

Pour aller plus loin et pour les prochains TP...

Se renseigner sur :
- La boîte à outil Stibox.
- Conversion des types de données.
- Les erreurs d'arrondis.
- La gestion par Matlab du path et des répertoires.

Bon courage !

Une liste non exhaustive de fonctions utiles :

Références

[1] Jean-Thierry Lapresté : *Introduction à Matlab*, Ellipses

[2] Gilles Stoltz : *TP1 : Introduction à Matlab*, ENS Ulm (disponible sur le web)

[3] Alain Lichnewsky : *Modélisation et informatique*, cours disponible sur le web