TP1 : INTRODUCTION À MATLAB

Résumé. Matlab est un logiciel de calcul numérique, utilisé dans de nombreux domaines d'application. Il est basé sur le calcul matriciel. Matlab est d'ailleurs un raccourci pour "Matrix Laboratory". Le but de ce TP est d'introduire les commandes les plus courantes.

1. Accès et commandes générales

1.1. **Lancement.** Pour lancer Matlab, double-cliquer sur l'icône ou ouvrir une fenêtre de commande et taper `matlab &` (le & permettant de garder la main dans la fenêtre de commande). Un environnement s'affiche à l'écran sous vos yeux emerveillés. Il est composé en général des fenêtres suivantes :

- Current Folder, qui vous indique le répertoire courant,
- Workspace, qui vous donne les variables en mémoire, leur type et leur taille,
- Command History, qui regroupe dans une pile les commandes passées,
- Command Window, la fenêtre de commande. Le caractère >> signifie que Matlab attend une instruction.

Il est possible de modifier cet environnement en cliquant sur `Desktop`.

1.2. **Gestion des fichiers de l'espace de travail.** Vous pouvez utiliser la fenêtre Current Folder pour vous déplacer dans l'arborescence, créer des répertoires, en effacer... Il existe aussi les commandes suivantes : `pwd` (affiche le nom du répertoire courant pour Matlab), `cd` *rep* (change le répertoire courant pour Matlab qui devient *rep*), `dir` (fournit le catalogue d'un répertoire), `delete` (efface des fichiers ou des objects graphiques), `mkdir` (crée un répertoire).

Exemple 1. Créer un répertoire dans lequel vous travaillerez et placez-vous dedans. Tapez `diary` dans la fenêtre de commande. Cette commande va créer un fichier (journal de bord) intitulé *diary* qui garde la trace de toutes les commandes que vous avez tapées dans la fenêtre de commandes ainsi que les réponses de l'ordinateur (`diary off` permet d'arrêter l'écriture du journal de bord, tandis que `diary on` permet de la reprendre).

1.3. **Calculs élémentaires.** Dans la partie commandes de l'interface, taper
>> `5+8`
Résultat : >> `13`
Pour conserver le résultat, il faut l'assigner dans un objet :
>> `a=5+8`
>> `a`
Pour ne pas faire afficher le résultat, mettez `;` à la fin de la commande :
>> `a=5+8;`
Il existe des variables prédéfinies, comme `pi` (3.1415..), `Inf` (nombre infini), `NaN` (n'est pas un nombre, exprime parfois une indétermination), `i` (nombre complexe i).

1.4. **L'aide dans Matlab.** Mieux vaut apprendre à se repérer tout seul que de demander en permanence à son voisin comment faire. Ne serait-ce qu'au cas où il faudrait utiliser dans l'examen une fonction dont on ne se souvient que vaguement quelle est sa syntaxe... Pour accéder à l'aide de Matlab, cliquer sur l'icône représentant un point d'interrogation bleu ou taper `helpwin`. Sur la gauche, plusieurs onglets sont accessibles :
- Contents, où sont rangées les fonctions par thème et boîte à outils,
- Search Results, qui contient les résultats des recherches par mots-clés.

En tapant `help`, vous obtiendrez la liste de toutes les commandes par thèmes (général, elfun, matfun,...). Vous pouvez accéder directement au thème, par exemple, elfun (elementary functions) en tapant `help elfun`. Les commandes d'aides à retenir sont

`help`	donne la liste de toutes les commandes par thèmes
`help` *nom*	décrit la fonction *nom.m*
`lookfor` *nom*	recherche une instruction à partir du mot clé *nom*

Exercice 1. Trouvez la fonction qui donne les valeurs propres d'une matrice. Tapez `help` de cette fonction.

1.5. **Historique.** Matlab conserve l'historique des commandes. Il est donc possible de récupérer des instructions déjà saisies (et ensuite de les modifier dans le but de les réutiliser) :

$\uparrow$, $\downarrow$, $\rightarrow$, $\leftarrow$	permet de se déplacer dans les lignes de commandes tapées dans la fenêtre de commandes

1.6. **Variables d'environnement.** Matlab garde en mémoire les variables qui ont été créées. On les voit en haut, à gauche, dans la fenêtre Workspace. Sinon, on peut utiliser les lignes de commandes suivantes.
`who` donne la liste des variables présentes dans l'espace de travail
`whos` donne la liste des variables présentes dans l'espace de travail ainsi que leurs propriétés
`what` donne la liste des fichiers `.m` et `.mat` présents dans le répertoire courant

`clear` $var_1 \dots var_n$	efface les variables $var_1, \dots var_n$ de l'espace de travail
`clear`	efface toutes les variables crées dans l'espace de travail

Exemple 2.

(1) Tapez la commande `a=1:7`. Tapez les commandes `a`, `who` et `whos`.

(2) Utilisez $\uparrow$ pour modifier a : `a=1:2;`.

(3) Tapez la commande `b=a+2;`. Réexécutez les commandes `who` et `whos` en utilisant $\uparrow$. Tapez `clear` et `b`.

1.7. **Dialogue avec l'utilisateur.**

`disp(var)`	affiche le contenu de *var*
`rep=input('texte')`	affiche la chaîne de caractères `texte` et donne la main à l'utilisateur pour qu'il entre la valeur de la variable `rep`

Il existe aussi la commande `pause(n)` qui arrête l'exécution d'un programme pendant `n` secondes (par défaut (sans `n`) reprend l'exécution du programme lorsque l'utilisateur appuie sur une touche du clavier) et la commande `keyboard` qui arrête l'exécution d'un programme et donne la main à l'utilisateur jusqu'à ce qu'il presse la touche Entrée.

Exemple 3. Essayez les commandes suivantes :

```
n=input('entrez le nombre de simulations : ')
disp(['Vous avez tapé : ' num2str(n)])
```

2. Les types de données

Matlab traite essentiellement un type d'objet : *les matrices* ! Les scalaires sont des matrices 1×1, les vecteurs lignes des matrices $1 \times n$, les vecteurs colonnes des matrices $n \times 1$.

2.1. **Construction explicite.** On peut former des vecteurs et des matrices en entrant leurs coefficients.

- **scalaires**
  >> `s=30`

- **vecteurs numériques**
  >> `x=[1;2;3]` (les `;` séparent les éléments d'un vecteur colonne)
  >> `x=[1,2,3]` (les `,` ou les blancs séparent les élements d'un vecteur ligne)
  >> `x'`
  >> `y = [x,x,x]`
  >> `z = [x x x]`

- **matrices**
  >> `M=[11 12 13 14; 21 22 23 24; 31 32 33 34; 41 42 43 44]`
  (où les `;` séparent les lignes d'une matrice)
  Construction à partir de plusieurs vecteurs de même longueur :
  >> `y=[11;22;33];`
  >> `mat1=[x' y]`

- **vecteurs de chaîne de caractères**
  La chaîne de caractères est un vecteur ligne. Pour le créer, on entre les caractères en commençant et en terminant par `'` (*quote*).
  >> `ch='matlab'`

- **les nombres complexes**
  Dans Matlab, un nombre complexe est de la forme : $z = a + ib$.
  >> `c=2+i`

- **les polynômes**
  Matlab représente un polynôme sous forme d'un vecteur ligne contenant les coefficients classés dans l'ordre des puissances décroissances. Par exemple le polynôme P d'expression $P(x) = x^2 - 6x + 9$ est représenté par
  >> `P=[1 -6 9]`

Exercice 2. Entrez les différents vecteurs et matrices et donnez la longueur et la taille de chacun (Utilisez `help` pour trouver les fonctions qui donnent longueur et taille).

2.2. **Création rapide.** Certaines commandes permettent de créer plus rapidement des vecteurs précis :
>> `l1=1:10` (Un vecteur contenant les entiers de 1 à 10)
>> `l2=1:1:10`
>> `l3=10:-1:1`
>> `l4=1:0.3:pi`
>> `l1(2)=l3(3)`
>> `l4(3:5)=[1,2,3]`
>> `l4(3:5)=[]`
>> `l5=linspace(1,5,5)`
>> `help linspace`

```
>> whos
>> clear l1 l2 l3 l5
>> who
>> clc (efface le contenu de la fenêtre de commande)
>> clear
```

A noter, **une ligne de commande commençant par le caractère % n'est pas exécutée par Matlab.** Cela permet d'insérer des lignes de commentaires. Et il faut commenter ses programmes... surtout ceux de l'examen.

Exercice 3. Construire :

(1) une suite partant de -8 et allant à -5 par pas de 0.25.

(2) une suite décroissante d'entiers de 15 à 3.

(3) une suite de longueur 100 de $-\pi$ à π.

2.3. **Opérations vectorielles.** Les tableaux suivants résument certaines commandes couramment utilisées.

Vecteurs

`n:m`	nombres de `n` à `m` par pas de 1
`n:p:m`	nombres de `n` à `m` par pas de `p`
`linspace(n,m,p)`	`p` nombres de `n` à `m`
`length(x)`	longueur de `x`
`x(i)`	i-ème coordonnée de `x`
`x(i1:i2)`	coordonnées `i1` à `i2` de `x`
`x(i1:i2)=[]`	supprimer les coordonnées `i1` à `i2` de `x`
`[x,y]`	concaténer les vecteurs `x` et `y`

Matrices

`size(A)`	nombre de lignes et de colonnes de `A`
`A(i,j)`	coefficient d'ordre `i,j` de `A`
`A(i1:i2,:)`	lignes `i1` à `i2` de `A`
`A(:,j1:j2)`	colonnes `j1` à `j2` de `A`
`A(:)`	concaténer les vecteurs colonnes de `A`
`diag(A)`	coefficients diagonaux de `A`

Matrices particulières

`zeros(m,n)`	matrice nulle de taille `m,n`
`ones(m,n)`	matrice de taille `m,n` dont tous les coefficients valent 1
`eye(n)`	matrice identité de taille `n`
`diag(x)`	matrice diagonale dont la diagonale est le vecteur `x`
`rand(m,n)`	matrice de taille `m,n` à coefficients i.i.d. de loi uniforme sur $[0,1]$
`randn(m,n)`	matrice de taille `m,n` à coefficients i.i.d. de loi normale $\mathcal{N}(0,1)$

Remarquez que la commande `zeros(m,n,p)` marche aussi, car Matlab permet de manipuler des matrices à trois dimensions (ou plus).

Exemple 4. *Extraction de composantes*
Entrez la matrice
`>> A=[1 2 3 ; 2 3 1 ; 3 1 2 ]`
Quels sont les résultats des commandes suivantes ?

```
>> A([2 3],[1 3])
>> A([2 3],1:2)
>> A([2 3],:)
>> A([2 3],end)
>> A(:)
```

On peut créer des matrices par blocs. Par exemple
`>> C=[A, zeros(3,2); zeros(2,3), eye(2)]`

Exercice 4. Répliquez le vecteur colonne `[1; 3; 6]` pour en faire une matrice 3×19, de deux manières : en utilisant `ones` et en effectuant une multiplication matricielle, puis en trouvant la commande *ad hoc* de réplication.

Exercice 5. Ecrire la matrice carrée M d'ordre 12 contenant les entiers de 1 à 144 rangés par ligne. Extraire de cette matrice les matrices suivantes :
- la sous-matrice formée par les coefficients a_{ij} pour $i = 1, ..., 6$ et $j = 7, ..., 12$;
- celles des coefficients a_{ij} pour $(i, j) \in \{1, 2, 5, 6, 9, 10\}^2$;
- celle des coefficients a_{ij} pour $i + j$ pair.

3. Les opérations matricielles et les fonctions

3.1. **Les opérations matricielles.**

`A'`	transposée de `A` (conjuguée si `A` est complexe)
`rank(A)`	rang de `A`
`inv(A)`	inverse de `A`
`expm(A)`	exponentielle de `A`
`det(A)`	déterminant de `A`
`trace(A)`	trace de `A`
`poly(A)`	polynôme caractéristique de `A`
`eig(A)`	valeurs propres de `A`
`[U,D]=eig(A)`	vecteurs propres et valeurs propres de `A`
`+ -`	addition, soustraction
`* ^`	multiplication, puissance (matricielles)
`.* .^`	multiplication, puissance terme à terme
`A\b`	solution de $Ax = b$, équivalent à `inv(A)*b` si `A` est inversible
`b/A`	solution de $xA = b$
`./`	division terme à terme

Exercice 6. Essayez des fonctions sur la matrice `A`. Par exemple, quels sont ses valeurs et vecteurs propres ? Puis, construisez une matrice `C` de même taille que `A`. Essayez `A+C`, `A*C`, `A.*C`. Ensuite, définissez la matrice `B` comme étant la matrice `A` à laquelle on a ajouté à chaque colonne le vecteur `[1; 2; 3]`. Déterminez son noyau. Y a-t-il une fonction prédéfinie dans Matlab qui détermine le noyau d'une matrice ?

Exercice 7. Résolution d'un système sous-dimensionné :

$$\begin{cases} 2x_1 + x_2 - 3x_3 &= 1 \\ x_1 - 2x_2 + x_3 &= 2 \end{cases}$$

(1) Ecrivez le système sous la forme matricielle $Ax = b$ (où vous définissez `A` et `b`) et calculez le rang de la matrice A.

(2) Définissez le vecteur `c = [1; 1; 1]` et déterminez l'image du vecteur c par la matrice A.

(3) Résolvez l'équation $Ax = b$.

3.2. **Les fonctions.**

Fonctions élémentaires

`sqrt`	`exp`	`log`
`sin`	`cos`	`tan`
`asin`	`acos`	`atan`
`round`	`floor`	`ceil`
`abs`	`angle`	`conj`

Exemple 5. Construire un vecteur quelconque et essayer les fonctions ci-dessus.

Certaines fonctions de Matlab s'appliquent à l'ensemble d'un vecteur. Lorsqu'on les applique à des matrices, elles opèrent colonne par colonne. Le tableau suivant décrit le résultat de quelques unes de ces fonctions lorsqu'elles sont appliquées à un vecteur x :

Fonctions vectorielles

`max(x)`	maximum
`min(x)`	minimum
`sort(x)`	tri par ordre croissant
`[y, I] = sort(x)`	retourne en plus les indices des élements de `x`
`find(x)`	retourne les indices non nuls de `x`
`[I, J] = find(x)`	retourne des lignes (dans le vecteur `I`) et des colonnes (dans le vecteur `J`) des élements non nuls du `x`
`sum(x)`	somme des éléments de `x`
`cumsum(x)`	vecteur contenant la somme cumulée des éléments de `x`
`prod(x)`	produit des éléments de `x`
`cumprod(x)`	vecteur contenant le produit cumulé des éléments de `x`
`diff(x)`	vecteur des différences entre deux éléments consécutifs de `x`
`mean(x)`	moyenne des éléments de `x`
`var(x)`	variance
`std(x)`	écart type (voir aussi `std(x,1)`)

Exemple 6. Regardez l'effet des instructions suivantes.

```
>> x=rand(1,5)
>> mean(x)
>> std(x)
>> median(x)
>> sort(x)
>> A=rand(3)
>> [B, I]=sort(A)
>> sort(A')
>> max(A)
>> max(A')
>> max(max(A))
>> sum(A)
>> cumsum(A)
>> prod(A)
>> diff(A)
>> D=A([1,2],1:3)
```

```
>> sum(D,1)
>> sum(D,2)
```

Exercice 8. Soit X une matrice $2\times n$ contenant les coordonnées de n points du plan. Comment faire pour obtenir une matrice où les points sont ordonnés par ordre croissant des abscisses ?

Exercice 9.

(1) Soit le vecteur de dimension 8 de composantes : 3.2, 4.8, 3.3, 3.2, 3.1, 4.2, 3.2, 3.3. Entrez le vecteur $\mathbf{y} = (y_i)_{i=1,\dots,8}$ correspondant.

(2) Construisez à l'aide des fonctions précédentes la suite des moyennes

$$\bar{y}_n = \frac{1}{n}\sum_{i=1}^{n} y_i \ .$$

Extrayez $\bar{y}_8$. Donnez une fonction qui calcule directement $\bar{y}_8$ à partir de $\mathbf{y}$.

Exercice 10. Tirez 20 nombres aléatoirement dans l'intervalle $[0, 1]$. Quelle est la valeur minimale du vecteur et la position du coefficient qui la réalise ? Vérifiez.

Pour finir, voici quelques fonctions pour les polynômes. On rappelle que Matlab représente le polynôme $P = a_nX^n + \cdots + a_1X + a_0$ sous forme du vecteur ligne `[an...a0]`.

Fonctions sur les polynômes

`roots`	donne les racines du polynôme
`poly`	crée un polynôme à partir de ses racines
`polyval`	évalue la valeur d'un polynôme en un point
`conv`	multiplie deux polynômes
`polyder`	dérive un polynôme

4. Opérateurs relationnels et logiques

Ils permettent de relier logiquement deux *matrices*.

`Opérateurs relationnels`	<, <=, >=, == (égalité), ∼= (différent)
`Opérateurs logiques`	& (et), | (ou), ∼ ou `not` (non)

Attention de ne pas confondre `=` qui sert à affecter une valeur à une variable et `==` qui sert à tester l'égalité. Les opérateurs relationnels peuvent être utilisés avec des scalaires ou des matrices. Le résultat d'évaluation d'une expression relationnelle est `1` (vrai) ou `0` (faux). Appliqués à une matrices, ils rendent une matrice de même dimension, formée de 1 et de 0.

Exemple 7.

```
u=4
u==4
u<=12
```

Reprenez la matrice `A=[1 2 3; 2 3 1; 3 1 2]` en tapant `A=[` ↑.

```
Ar=(A<=2)
[B,I]= find(A==1)
```

(Retourne les coordonnées égales à 1, ainsi que leur position.)

Exercice 11. Reprenons le vecteur $\mathbf{y}$. Pour rappel, tapez `y`.

(1) Faites répondre Matlab à la question suivante : existe-t-il une coordonnée du vecteur **y** inférieure à 3.3 ?

(2) Construisez un vecteur logique **z** tel que la i-ème coordonnée de ce vecteur sera 1 si la $i$ème coordonnée du vecteur y est à l'extérieur de l'intervalle $[\bar{y}_8 - \sigma_8, \bar{y}_8 + \sigma_8]$ où σ_8 est la racine carrée de la variance d'échantillonnage.

Exercice 12.

(1) Tirez 100 nombres aléatoirement (et uniformément) dans l'intervalle $[0, 1]$ et groupez-les dans un vecteur $\mathbf{x} = (x_i)_{i=1,\ldots,100}$.

(2) Prenez $y_i = 2 * x_i$ pour tout $i = 1 \ldots, 100$.

(3) Prenez la partie entière de ces nombres (à l'aide de la fonction `floor`) : $z_i = [y_i]$. Ceci définit un vecteur **z**. (Au passage, notez qu'il existe plusieurs fonctions parties entières, avec des comportements différents, `ceil` par exemple...)

(4) Donnez la fréquence de 1 sur l'échantillon **z**. Pouvait-on s'attendre à ce résultat ?

Réinitialisez l'espace de travail en tapant `clear` (effacement des variables d'environnement) puis `clc`.

5. Graphiques

5.1. **Représentations de points dans le plan.** Il existe plusieurs possibilités pour représenter un ensemble de points $(x(i), y(i))$. Les plus utilisées sont énumérées ci-dessous.

`plot(x,y,'s')`	tracé d'une courbe ou d'un nuage de points
`bar(x,y,'s')`	tracé sous forme d'un histogramme
`stem(x,y,'s')`	diagramme en bâtons
`stairs(x,y)`	tracé en escalier des valeurs discrètes
`fplot`	représente des fonctions
`hist`	trace des histogrammes (voir aussi `histc`)

`'s'` est un paramètre facultatif constitué d'une chaîne de caractères qui spécifie le type de tracé (couleur, différents tracés en pointillés, symboles pour le tracé de points). Par défaut, le tracé est continu. Tapez `help plot` pour avoir la liste des valeurs possibles pour `'s'`.

5.2. **Gestion de la fenêtre graphique.**

`hold on`	les prochains tracés se superposeront aux tracés déjà effectués
`hold off`	le contenu de la fenêtre graphique active sera effacé lors du prochain tracé
`clf`	efface le contenu de la fenêtre graphique active
`figure(n)`	affiche ou rend active la fenêtre graphique numéro `n`
`close`	ferme la fenêtre graphique active
`close all`	ferme toutes les fenêtres graphiques
`subplot(n,m,p)`	partage la fenêtre graphique active en `m` $\times$ `n` espaces graphiques et sélectionne le `p`-ième.

`axis([xmin xmax ymin ymax])`	pour définir les échelles des axes
`grid`	quadrillage du graphique
`grid off`	
`title('titre')`	titre pour le graphique
`xlabel('titre')`	légende pour l'axe des abscisses
`ylabel('titre')`	légende pour l'axe des ordonnées
`legend('titre1','titre2',...)`	légende pour chaque courbe du graphique
`text(x,y,'texte')`	texte explicatif à la position (x, y)
`gtext('texte')`	texte positionné à l'aide de la souris

5.3. **Axes et légendes.**

Si `a` est une variable contenant le nombre $\sqrt{2}$, la commande `text(x,y,['le resultat est ' num2str(a,3)])` affichera le texte "`le resultat est 1.41`" à partir de la position `x,y` de la fenêtre graphique.

Exemple 8.

```
x=-pi:0.1:3*pi; y=x.*sin(x);
plot(x,y)
clf
plot(x,y);axis([-pi,3*pi,-6,9])
xlabel('x') ylabel('y')
title(['graphe de la fonction x sin(x) sur l''intervalle [' num2str(x(1)) ' ,
' num2str(x(end)) ']'])
plot(x,y,x,2*y)
plot(x,[y;2*y])
plot(x,y,'r-',x,2*y,'g+')

fplot('[x*sin(x),2*x*sin(x)]',[-pi,3*pi])
fplot(@(x) x*sin(x),[-pi,3*pi],'b-');; hold on
fplot('2*x*sin(x)',[-pi,3*pi],'yo'); hold off

t=0:0.1:2*pi;
plot(sin(t),sin(2*t))
plot(sin(t),sin(2*t),'c-')
```

Exercice 13. Soit f et g les fonctions définies sur l'intervalle $[0, 10]$ par :

$$f(x) = \exp\left(-\frac{x}{25}\right), \quad g(x) = \cos\left(\frac{x}{10}\right) .$$

Tracez ces deux fonctions (à l'aide de `fplot`) d'abord dans une même fenêtre graphique mais sur des graphes différents, puis dans une même fenêtre graphique et sur le même graphe.

5.4. **La sauvegarde d'une figure.** Une figure peut être sauvegardée sous plusieurs formats :
- sous un format propre à Matlab avec l'extension `.fig` (pour les versions récentes de Matlab). Pour cela, cliquer sur la commande `Save as` du menu `File` de la fenêtre graphique et entrer un nom de fichier avec l'extension `.fig` dans l'encadré qui apparaît. Un tel fichier peut être visualisé en utilisant la commande `Open` du menu `File`.
- sous un format PostScript en utilisant la commande `Export` du menu `File` d'une fenêtre graphique. Dans ce cas, un fichier *nomfichier.ps* est créé dans le répertoire courant.

Exercice 14.

(1) Tirer 100 couples de points (x, y) aléatoirement dans le carré $[0, 1] \times [0, 1]$.

(2) Représenter le nuage de points obtenus dans une fenêtre graphique.

(3) Calculer le centre de gravité G du nuage de points.

(4) Ajouter en rouge au nuage de points le centre de gravité.

(5) Sauvegarder la figure sous le nom `nuage.fig`. Fermer la fenêtre graphique. Ouvrir une nouvelle fenêtre graphique et faire réapparaitre le fichier `nuage.fig`.

(6) Faire afficher dans une même fenêtre graphique deux histogrammes, un pour les abscisses et l'autre pour les ordonnées des points tirés.

6. Utilisation de fichiers.

6.1. **Les fichiers de sauvegarde.** On a déjà vu en début de TP la commande `diary`. Voici d'autres façons de sauvegarder son travail.

`save` *nomfichier* var_1 ... var_n	sauve la valeur des variables var_1 ... var_n dans un fichier binaire *nomfichier.mat*
`save` *nomfichier*	sauve l'ensemble des variables existant dans l'espace de travail de Matlab dans un fichier binaire *nomfichier.mat*
`save` -ASCII *nomfichier* var_1 ... var_n	sauve la valeur des variables var_1 ... var_n dans un fichier texte *nomfichier*
`load` *nomfichier*	permet de récupérer toutes les données sauvegardées dans le fichier *nomfichier.mat*

Exemple 9. Sauvegardez `x`, `y` et le centre de gravité `G` dans un fichier binaire `sauv.mat`. Dans Matlab, avec la commande `clear`, effacez `x`, `y` et `G`. Puis, chargez le fichier `sauv.mat` afin de les récupérer. Vérifiez la récupération.

6.2. **Les programmes *script* (ou fichiers d'instructions).** Ce sont des fichiers texte avec une extension `.m`. Ils contiennent des suites d'instructions Matlab qui sont exécutées les unes après les autres. On peut créer un fichier en cliquant sur `File` puis `New`, `Script`.

Exemple 10. Sauvez dans le répertoire courant les lignes suivantes sous le nom `losange.m` :

```
x=[0 -1 0 1 ; -1 0 1 0 ]
y=[-1 0 1 0 ; 0 1 0 -1]
plot(x,y)
```

La commande `losange` affichera `x`, puis `y`, puis tracera un losange. Si ce fichier est placé dans un répertoire accessible, la commande `losange` devient une commande Matlab comme toutes les autres. On peut aussi lancer l'éxecution du script depuis l'éditeur en cliquant sur l'icône avec une flèche verte.

6.3. **Les fichiers de fonctions.** Comme les fichiers d'instruction, ce sont des fichiers textes avec une extension `.m`. Leur syntaxe est particulière. Ils contiennent la définition d'une fonction et portent le nom de cette fonction.

Exemple 11.

```
function d=densnorm(x)
% densnorm : densite de la loi N(0,1)
% densnorm(x) retourne (1/sqrt(2*pi))*exp(-x^2/2)
d=(1/sqrt(2*pi))*exp((-x.^2)/2);
```

Si ce fichier est placé dans un répertoire accessible, la fonction `densnorm` devient une fonction Matlab comme toutes les autres. Le texte placé en commentaire est le contenu de l'aide pour la nouvelle fonction.

Exercice 15. Utilisez la fonction ci-dessus pour tracer (avec `plot`) la densité de la loi normale centrée réduite entre -5 et 5 en faisant calculer 100 points. Aurait-on pu utiliser `fplot` ? Calculez $\mathbb{P}(-5 \leq X \leq 1.96)$ où X est une variable aléatoire de loi $\mathcal{N}(0,1)$.

Pour l'exercice suivant (et pour le reste des TPs) nous allons utiliser une boîte à outils, c'est-à-dire un ensemble de fonctions .m regroupées par thèmes. La boîte à outils *Stixbox* contient des fonctions utiles en statistiques, comme par exemple fonctions de répartition, densités, générateurs de nombres pour les lois standard. L'ensemble des fonctions de la boîte à outils est accessible via : `help stixbox`. Si cette commande ne marche pas, c'est qu'il vous faut charger *Stixbox*, pour cela tapez : `addpath /usr/local/MATLAB/toolbox/stixbox`.

Exercice 16. Ecrire une fonction `simule.m` qui prend `n` en argument et :

(1) construit un vecteur de n nombres aléatoires $(y_i)_{i=1,\dots,n}$ issus d'une loi normale de moyenne 2 et de variance 1,

(2) renvoie la moyenne et la variance de l'échantillon observé,

(3) sépare la fenêtre graphique en deux espaces graphiques et superpose respectivement sur chacun d'eux :
   - l'histogramme de $(y_i)_{i=1,\dots,n}$ (on pourra utiliser la fonction `histo`) et la densité de la loi normale de moyenne 2 et de variance 1, donne un titre à la figure et affiche la moyenne et la variance,
   - la fonction de répartition empirique (utiliser les fonctions `sort` et `stairs`) et la fonction de répartition théorique (utiliser la fonction `pnorm` de *Stixbox*), et donne un titre à la figure.

Faites tourner le programme pour d'autres valeurs de n, plus grandes ou plus petites.
Indication : la première ligne du fichier de fonction est `function [moy,vari] = simule(n)`.

7. Les commandes structurées

7.1. **L'instruction `for`.** La syntaxe en est la suivante :

```
for   variable = vecteur
  instructions
end
```

Les colonnes du *'vecteur'* sont affectées, l'une après l'autre à la variable '*variable*' et pour chacune de ces valeurs, les *'instructions'* sont exécutées. Remarquez qu'il faut privilégier les opérations vectorielles à l'utilisation de boucles.

Exemple 12. Ces quelques lignes calculent $n!$ pour $n = 1$ à 100.

```
fact = zeros(1,100);
fact(1)=1;
for k=2:100
fact(k)=fact(k-1)*k;
end
```

Mais on peut aussi taper simplement : `n=100; fact=cumprod(1:n);` (ou même `factorial(n)`)
L'exécution du premier programme prend significativement plus de temps que l'exécution du deuxième programme. Pour le voir, et comparer l'efficacité des algorithmes, utilisez `tic` (à placer en début de programme) et `toc` (à placer en fin de programme) qui permettent de compter le temps CPU écoulé.

7.2. **Les intructions conditionnelles.** La manière la plus brute de procéder est d'utiliser un bloc `if ... then ... else ... end`. La syntaxe en est la suivante :

```
if conditions
  instructions
end
```

Les '*instructions*' ne sont exécutées que si les '*conditions*' sont vérifiées, plus précisément si '*conditions*' a une valeur différente de 0. Une variante plus élaborée est :

```
if conditions
   instructions        (exécutées si les 'conditions' sont vérifiées)
else
   instructions        (exécutées si les 'conditions' ne sont pas vérifiées)
end
```

Ou encore (avec des emboîtements) :

```
if conditions1
   instructions        (exécutées si les 'conditions1' sont vérifiées)
elseif conditions2
   instructions        (exécutées si les 'conditions1' ne sont pas vérifiées
                        mais si les 'conditions2' le sont)
else
   instructions        (exécutées si ni les 'conditions1' ni les 'conditions2' ne sont vérifiées)
end
```

Exemple 13. Le programme suivant simule le lancer d'une pièce.

```
p=0.5;
u=rand;
if u<p
   disp('pile')
else
   disp('face')
end
```

Rappelons qu'une proposition logique (par exemple, `(x>0 & x<10)`) a pour valeur 1 si elle est vraie et 0 sinon. Dans les calculs, on peut donc éviter d'utiliser l'instruction `if` en introduisant des indicatrices d'ensembles.

Exercice 17. En tenant compte de cette remarque, simuler la réalisation d'une variable aléatoire de loi de Bernoulli de paramètre p. Puis, après cet échauffement, écrire un programme *script* `EstimeParam.m` qui :

(1) simule un 1000-échantillon de variables aléatoires de Bernoulli de paramètre 0.2,

(2) affiche le nombre de succès contenus dans ce 1000-échantillon,

(3) et trace la courbe des moyennes en fonction de la longueur de l'échantillon.

Exécuter le programme. Que remarque-t-on ? Quel est le théorème qui explique le phénomène ?

Exercice 18. Nous allons censurer des variables. Soit `u` une réalisation d'un 15-échantillon $X_1, \ldots, X_{15}$ de loi uniforme sur $[-5, 1]$. Mettez les observations plus petites que -2 à -10. Ceci forme `v`, réalisation d'un 15-échantillon $Y_1, \ldots, Y_{15}$. Quelle est la loi commune aux Y_i ?

7.3. **L'instruction `while`.** Ce format de boucle permet de s'arrêter conditionnellement (et non plus à rang fixé, comme dans une boucle `for`). La syntaxe en est la suivante :

```
while  conditions
   instructions
end
```

Les '*instructions*' sont exécutées tant que les '*conditions*' sont vérifiées.

Exercice 19. Ecrire une fonction `y = Puiss(n,M)` qui étant donné un entier n et une valeur maximale $M > n$, calcule les puissances entières de ce nombre, les agrège dans le vecteur `y` et s'arrête lorsque le résultat dépasse la valeur maximale.