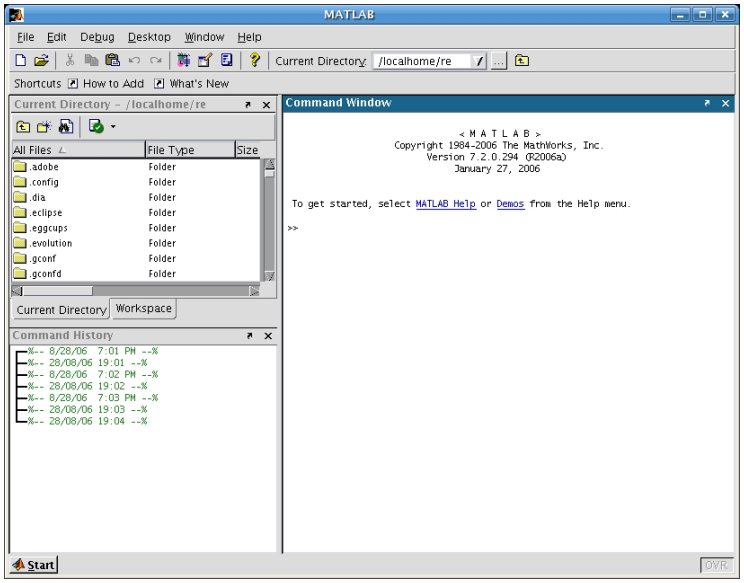
# TP1 – Initiation à Matlab

## 1 Généralités

MATLAB est l'abréviation en anglais de MATrix LABoratory. Donc l'objet de base manipulé sous MATLAB est la matrice, les calculs sont (devraient être) matriciels et/ou vectoriels. Ce progiciel est dédié aux mathématiques numériques et aux graphiques. Au lancement, on se retrouve avec un prompt ">>" qui indique le début d'une session MATLAB.




L'écran est divisé en 3 fenêtres (cf figure ci dessus) :

- *Command Window :* fenêtre de commande où l'utilisateur peut entrer multiples commandes ou équations mathématiques après le signe ">>" qui apparaît sur le côté gauche de la fenêtre. Pour exécuter une opération, il faut toujours appuyer sur la touche "enter" du clavier. De plus, il faut terminer l'opération par un point-virgule " ;" sinon, toutes les étapes du calcul seront affichées sur l'écran.
- *History Window :* fenêtre où s'affiche tout l'historique de vos commandes passées.
- *Current directory :* chemin où l'utilisateur exécute ses commandes.
- *Workspace :* espace de travail qui liste toutes les variables stockées pouvant être réutilisées dans les calculs qui suivent. Les commandes *who, whos* permettent de lister l'ensemble des variables utilisées. Par défaut, tout calcul est affectée dans la variable *ans*. La commande *clear* efface le contenu de toutes les variables utilisées.

**Attention** : Pour sortir de Matlab, il suffit d'utiliser les commandes *quit* ou *exit*.
Lorsque l'on veut interrompre un programme, la commande $CTRL - c$ permet de récupérer la main.

## Aide en ligne

Il existe plusieurs manières pour obtenir de l'aide en ligne sous matlab.

- *help* : "help" tout seul donne la liste des aides générales possibles ;

- *helpwin* : ouvre une fenêtre et donne accès à une aide détaillée, en gros à tout le manuel de Matlab ;
- *help nom de commande* : exemple, *help plot* indique la syntaxe des graphes en 2D ;
- *lookfor nom de commande* : exemple, $lookfor\ plot$ donne une liste de toutes les commandes qui ont un rapport avec *plot* ;
- *demo* : lance une démo générale de Matlab ;
- *help demos* : donne une liste des demos existantes.

# 2 Matrices et vecteurs

## 2.1 Constuction de vecteurs et de matrices

Pour définir un vecteur, les composantes sont séparées par des virgules pour les vecteurs lignes et par des points-virgules pour les vecteurs colonnes.

- vecteur ligne :
  `>> v=[2,3,7]`
- vecteur colonne :
  `>> w=[4;2;9]`
- vecteur à incrément constant : $x_{min} : d_x : x_{max}$ désigne l'ensemble des points de la forme $x_{min} + kd_x; k \in \mathbb{N}$ compris entre $x_{min}$ et $x_{max}$, l'extrémité étant $x_{min}$ étant toujours incluse et $d_x$ représente le pas d'incrémentation :
  `>> y=1:5:23`

Les matrices suivent la même syntaxe que les vecteurs. Les composantes lignes sont séparées par des virgules et chaque ligne est séparée de l'autre par une point virgule.

```
>> R=[1,5,8;7,10,23;85,4,6]
```

## 2.2 Quelques matrices utiles

- $zeros(n, m)$ construit une matrice nulle de taille $n \times m$,
- $ones(n, m)$ construit une matrice dont tous les éléments sont égaux à 1 et de taille $n \times m$,
- $eye(n)$ construit une matrice identité de taille $n \times n$ dont tous les éléments diagonaux sont égaux à 1,
- $rand(n, m)$ construit une matrice de taille $n \times m$ dont tous les éléments sont choisis aléatoirement avec la loi uniforme sur $[0\,;\,1]$,
- $randn(n, m)$ construit une matrice de taille $n \times m$ dont tous les éléments sont choisis aléatoirement avec la loi normale centrée réduite.

S'il n'y a qu'un seul paramètre d'entrée, $zeros(n)$ (resp. $ones(n), ...$) construit une matrice carrée de taille $n \times n$ de 0, (resp. de 1,...).

### Exercice

1. Construire les vecteurs et matrices suivants :

$$[1\ 4\ 2\ 9\ 14\ 3\ 16],\quad \begin{bmatrix} 6 \\ 2 \\ 15 \\ 8 \\ 24 \\ 7 \end{bmatrix},\quad \begin{bmatrix} 1 & 4 & 5 & 7 \\ 5 & 7 & 3 & 11 \\ 18 & 4 & 9 & 2 \end{bmatrix}$$

2. Construire un vecteur incrémental commençant par 3 jusqu'à 20 par pas de 0.5.
3. Construire à l'aide des fonctions précédentes, les matrices suivantes :

$$\begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{bmatrix},\quad \begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 115 \\ 4 & 8 & 12 & 16 \end{bmatrix},\quad \begin{bmatrix} 2 & 0 & 0 & 1 & 1 \\ 0 & 2 & 0 & 1 & 1 \\ 0 & 0 & 2 & 0 & 0 \end{bmatrix},$$

## 2.3 Quelques opérations sur les matrices

### 2.3.1 Opérations classiques

On considère la matrice A suivante :

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

- Transposition
  ```
  >> A'
  ```
- Addition, soustraction, multiplication, puissance (si $A$ est carrée)
  ```
  >> A+A'
  >> A-A'
  >> A*A'
  >> A^3
  ```
- Diagonale, trace, rang, déterminant
  ```
  >> diag(A)
  >> trace(A)
  >> rank(A)
  >> det(A)
  ```
- Inverse, pseudo-inverse
  ```
  >> inv(A)
  >> pinv(A)
  ```

### 2.3.2 Opérations élément à élément

Les opérations précédentes peuvent être effectuées élément par élément entre matrices.
On considère alors la matrice $D$ aléatoire de mêmes dimensions que la matrice $A$.

- Addition ou soustraction : pas de changement ;
- Multiplication, division, puissance (si $A$ et $D$ de mêmes dimensions ; $A$ pas nécessairement carrée)
  ```
  >> D.*A % chaque élément (ij) de D est multiplié par l'élément (ij) de A
  >> D./A % chaque élément (ij) de D est divisé par l'élément (ij) de A
  >> A.^3 % chaque élément (ij) de A est élevé à la puissance 3
  ```

### 2.3.3 Quelques fonctions utiles

- Les fonctions scalaires courantes c'est-à-dire *sin*, *exp*, *cos*, *tan*... peuvent aussi s'appliquer à des matrices, composante par composante :
  ```
  >> u=[0:2:20]
  >> v=sin(u)
  >> w=exp(u)
  ```
- la fonction $size(A)$ renvoie la taille de la matrice de $A$
  ```
  >> B=rand(4,5) % créé une matrice de taille 4 x 5
  >> [n,m]=size(B) % stocke en n le nombre de lignes de B et en m le nombre de colonnes de B
  ```
- la fonction $find(C(A))$ renvoie les indices dans le tableau A des composantes vérifiant la condition $C(A)$ :
  ```
  >> A=rand(1,5) % créé un vecteur ligne contenant 5 nombres repartis aléatoirement entre 0 et 1
  >> find(A>0.5) % renvoie les indices des composantes de A > 0.5
  >> find(A==0.2) % renvoie les indices des composantes de A égales à 0.2
  ```

**Exercice :**

1. Construire 2 matrices aléatoires, notées B et C, de dimensions $3 \times 5$.

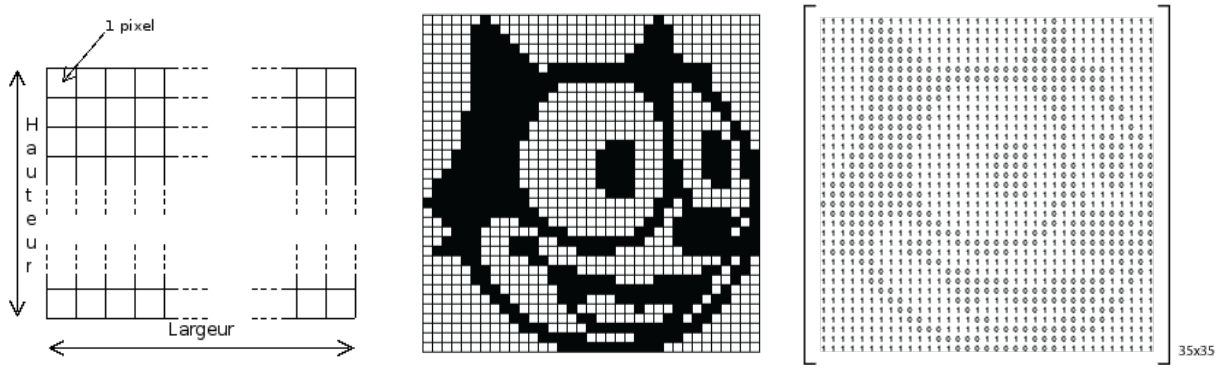
2. Calculer le produit élément à élément de B et C et le produit matriciel de B par la transposée de C. Ces matrices seront respectivement notées $F$ et $G$.
3. Lister tous les éléments de $F$ compris entre 0.2 et 0.6. Lister tous les élements de $G$ supérieur ou égal à 1.2.

#### 2.3.4 Extraction de vecteur/sous-matrice d'une matrice

Soit A une matrice aléatoire de taille $6 \times 8$.
- Extraction de la 4ème ligne de la matrice A :
  `>> A(4,:)`
- Extraction de la 2ème colonne de la matrice A :
  `>> A(:,2)`
- Extraction des 3 premières composantes de la 3ème ligne de la matrice A :
  `>> A(3,1:3)`
- Extraction de la sous-matrice formée des 2ième et 4ième lignes de la matrice A et de la 5ème à la 7ème colonne de A :
  `>> A([2,4],5:7)`

**Exercice : Application sur des images**

Une image matricielle est une image numérique dans un format de données qui se compose d'un tableau de pixels, généralement rectangulaire, qui peut se visualiser sur un moniteur d'ordinateur ou tout autre dispositif d'affichage RVB.




Exemple d'image matricielle

1. Charger et visualiser l'image avec les commandes suivantes :

```
>> I=imread('cameraman.tif'); % Charge l'image cameraman.tif
>> imshow(I) % Affiche la matrice I
```

2. Extraire de la matrice I la sous-matrice formée de la 30ème à la 130ème ligne et de la 50ème à la 180ème colonne.
3. Visualiser cette sous-matrice.
4. Charger l'image couleur *ngc6543a.jpg* et donner les dimensions de l'image, notée $J$. Extraire la sous-image couleur formée de la 150ème à la 400ème ligne et de la 150ème à la 450ème colonne de $J$.

# 3 Programmation

## 3.1 Tests et boucles

1. **Tests**
   Pour effectuer un test en Matlab, on dispose du classique $if..else..end$ agrémenté du $if...elseif...end$ :

```
>> x=5
>> if x>0, y=-x, else y=2, end
```

Il existe aussi *switch...case...end* permettant de gérer les tests portant sur un seul critère pouvant prendre plus de deux valeurs entières.

2. **Boucles**
   Il y a deux types de boucles en Matlab : les boucles *while* et les boucles $for$.
   La boucle $for$ parcourt un vecteur d'indices et effectue à chaque pas toutes les instructions délimitées par l'instruction *end*.

```
>> x=1;
>> for k=1:4, x=x*k, end
```

   La boucle *while* effectue une suite de commandes tant qu'une condition est satisfaite.

```
>> x=1;
>> while x<14, x=x+4, end
```

## 3.2 Fichiers M

Ces fichiers textes contiennent des lignes d'instructions MATLAB et ont une extension « .m ». Ils sont exécutés ligne par ligne par MATLAB. Ils peuvent être de 2 types différents, scripts ou fonctions.

1. **Scripts**
   Les scripts sous Matlab sont équivalent aux procédures, ils ne prennent pas d'argument. Les scripts partagent l'espace de travail de base (base workspace) avec la session interactive Matlab et les autres scripts.

   Si vous utilisez des variables temporaires, indices de boucles, etc, il est conseillé de les supprimer de l'espace de travail à la fin du script avec la commande *clear* :

```
...
clear i j
% fin du script
```

2. **Fonctions**
   Les fichiers $function$ sont équivalent aux sous-programmes. Une fonction peut posséder des arguments d'entrée et des arguments de sortie. De manière générale, la syntaxe de définition d'une fonction externe est :
   `function [`$y_1$`,...,`$y_m$`]=toto(`$x_1$`,..,`$x_n$`)`
   ⋮
   où *toto* est le nom de la fonction, $x_1, ..., x_n$ les $n$ arguments d'entrée et $y_1, ..., y_m$ les $m$ arguments de sortie. Les points verticaux symbolisent les instructions effectuées à l'appel de la fonction. Le passage des arguments d'entrée dans les fonctions se fait par valeur. Si une des variables de la procédure n'est pas définie à l'intérieur de celle-ci, elle doit obligatoirement être fournie en argument d'entrée. La récupération des valeurs calculées par la fonction *toto* se fait par les paramètres de sortie $y_1, \ldots, y_m$.

   Il est préférable d'utiliser le même nom de fichier que la fonction elle-même.
   A la différence des fichiers script classiques dans MATLAB, les variables à l'intérieur d'un fichier function ne sont pas disponibles à l'extérieur, elles ne sont visibles que dans leur propre espace de travail.

# 4 Commandes spécifiques en probabilités et statistiques

## 4.1 Outils classiques

Il existe plusieurs fonctions pour calculer les paramètres d'une distribution $P$ de points.
- mean : `mean(P)` moyenne de l'ensemble $P$. Si $P$ est une matrice, on peut spécifier si la moyenne est réalisée sur les vecteurs lignes ou colonnes (`mean(P,1)`, `mean(P,2)`).
- std : `std(P)` écart type de l'ensemble $P$. Si $P$ est une matrice, c'est le même fonctionnement que pour `mean`.

- var : `var(P)` variance de l'ensemble $P$. Si $P$ est une matrice, c'est le même fonctionnement que pour `mean`.
- covar : `cov(P)` covariance de l'ensemble $P$. Si $P$ est une matrice de taille $m \times n$ chaque colonne correspond à une variable et chaque ligne correspond à une observation. On a donc $m$ observations de chacune des $n$ variables. La commande `cov(P)` permet d'obtenir la matrice de covariance de $P$.

### 4.2 Lois de probabilités

Il existe plusieurs commandes pour générer des lois de probabilités. Dans ce qui suit, le paramètre `n` peut être un entier ou un vecteur de 2 entiers pour indiquer le vecteur ou la matrice à générer.

- *rpoiss* : `X=rpoiss(n,lambda)` permet de construire un vecteur ou une matrice X contenant des nombres aléatoires suivant une loi de Poisson de paramètre `lambda`.
- *rexpweib* : `X=rexpweib(n,lambda)` permet de construire un vecteur ou une matrice X contenant des nombres aléatoires suivant une loi exponentielle de paramètre `lambda`.
- *rgeom* : `X=rgeom(n,p)` permet de construire un vecteur ou une matrice X contenant des nombres aléatoires suivant une loi géométriques de paramètre `p`.
- *rand* : `X=rand(n)` génère un vecteur ou une matrice X contenant des nombres aléatoires suivant une loi uniforme sur [0,1].
- *randn* : `X=randn(n)` génère un vecteur ou une matrice X contenant des nombres aléatoires suivant une loi normale centrée réduite.

## 5 Représentation graphique

Dans toutes les représentations graphiques, les données à représenter doivent être des vecteurs colonnes ($x$ et $y$) ou des matrices ($z$).

- Pour tracer des courbes planes, l'instruction de dessin correspondante est $plot(x, y)$, éventuellement complétée par des arguments optionnels (couleur, type de trait, échelle sur les axes,...)
```
>> x=0:0.1:10
>> y=exp(-x)
>> plot(x,y) % relie les points (xi,yi) par un trait continu noir
>> plot(x,y,'r.') % matérialise les points (xi,yi) par un point rouge
```
- pour tracer des surfaces 3D, il existe des fonctions comme $mesh(z)$, éventuellement complétée par des arguments optionnels (couleur, type de trait, échelle sur les axes,...)
```
>> z=rand(5,5)
>> mesh(z)
```
- la fonction *hist* permet de tracer des histogrammes
```
>> z=rand(1000,1);
>> hist(z)
```

Les points peuvent être matérialisés par les symboles suivants : $o$ . $*$ $+$ $\times$.
Les couleurs sont repérées par leurs initiales en anglais : r(ed), b(lue), w(hite), y(ellow), g(reen), m(agenta), c(yan), (blac)k. On peut superposer plusieurs courbes/histogrammes sur un même graphe grâce à la fonction `hold on`.

**Exercice :**

1. Représenter la loi normale $\mathcal{N}(2, 1)$ sur l'intervalle $[-1, 5]$.
2. Représenter sur un même graphe la loi normale centrée réduite $\mathcal{N}(0, 1)$ sur l'intervalle $[-4, 4]$ et l'histogramme issu d'une distribution aléatoire de $n = 1000$ points d'une loi normale $\mathcal{N}(0, 1)$.

## 6 Conseil de programmation

Pour gagner du temps en Matlab, il faut éviter le plus possible les boucles dans les programmes. Illustrons ce fait sur un script :
`z=zeros(50);` % `Première méthode`

```
tic % initialise le temps
for i=1:50
  for j=1:50
    z(i,j)=1000;
  end;
end;
tElapsed=toc % affiche le temps écoulé


z=zeros(50); % Deuxième méthode
tic % initialise le temps
a=1000*ones(50);
z=a;
tElapsed=toc % affiche le temps écoulé
```

**Exercice :**

1. Ecrire le script et donner les temps écoulés pour les deux méthodes.
2. A partir de ce script, faire une fonction dont la variable d'entrée est le taille de la matrice `z` et les variables de sortie sont les temps écoulés pour les deux méthodes.
3. Tester avec d'autres tailles de matrice `z`. Représenter sur un même graphe les temps écoulés en fonction de la taille de la matrice `z` pour chacune des méthodes. Qu'observez-vous ?