# Initiation à l'utilisation de Matlab dans le cadre du cours de Mécanique rationnelle

## 1. Matlab

Matlab est en même temps un langage de programmation et un environnement permettant d'utiliser ce langage. Ce langage fut développé dans les années 70 pour des applications impliquant des matrices, l'algèbre linéaire et l'analyse numérique. Le mot "Matlab" est en fait la forme courte de "Matrix Laboratory".
Comme ce language existe et est utilisé depuis longtemps, de nombreuses bibliothèques d'outils et de fonctions en rapport avec des applications bien spécifiques ont été développées et testées. Ainsi, Simulink est la boîte d'outil pour le traitement du signal.

## 2. Pourquoi Matlab ?

Les raisons de l'utilisation de Matlab dans l'enseignement de la mécanique rationnelle sont nombreuses. Tout d'abord parce que l'informatique occupe une place prédominante dans l'industrie et dans la recherche, ensuite Matlab est fortement utilisé dans l'industrie pour réaliser toute sorte de simulations, enfin Matlab est un logiciel très didactique utilisant un code simple et court sans compilation aux fonctionnalités multiples (toolbox, intégration de code C, programmation de haut niveau).

## 3. Introduction

### Interface Matlab

L'environnement de Matlab s'ouvre sur une fenêtre de dialogue. Les instructions peuvent être introduites directement dans cette fenêtre, Matlab exécutera la ligne de commande une fois `Return` enfoncé. Matlab affichera directement dans la fenêtre de dialogue le résultat de l'exécution de la commande sauf si la ligne est terminée par un "`;`". Attention, Matlab différencie les majuscules des minuscules.
A tout moment, une aide peut être obtenue en tapant la commande `help` toute seule ou suivie de la fonction qui pose problème.

*Help*

Le format des données peut être choisi par l'utilisateur. Par défaut, le format utilisé est le format `short`. La notation utilisée peut elle aussi être choisie par l'utilisateur.

*Format*

**Ex. 1 :** Tapez la ligne de commande suivante :

```
>>help
>>help format
>>pi
>>format long
>>pi
>>format bank
>>pi
>>format short e
>>pi
>>format
>>format compact
>>pi
>>pi;
```

### Variables

*+ / * -*

La déclaration des variables est très facile sous Matlab, le type de la variable étant défini lors de la première assignation. Les opérateurs classiques sont les suivants : `+ - * /` . L'utilisation de parenthèses étant elle aussi classique. Le résultat d'opérations sur les variables est stocké dans une autre variable. Par défaut, cette variable est `ans`. D'autres opérateurs sont définis comme l'exposant : `2^3` signifiant $2^3$. La racine carrée est donnée par `sqrt(var)`. Le calcul des fonctions trigonométriques se fait via les fonctions `cos(theta) sin(theta) tan(theta) cot(theta)`. Pour plus de renseignement voir `help elfun`.
Attention, l'angle est toujours donné en radian.

*Sqrt*

*Sin Cos Tan Cot*

**Ex. 2**

```
>>A = 3000
>>a = 200
>>aA = 500
>>x=2e3
```

Calculez $y=ax^2+Ax+aA$

*Who Whos Clear*

A tout moment, il est possible d'avoir la liste des variables dans l'environnement (`who`) ainsi que leur taille et leur type (`whos`). Les variables peuvent être effacées par la commande `clear var`, la commande `clear` utilisée seule effaçant toute les variables.

**Ex.3**

```
>>whos
>>clear A
>>A
>>who
>>clear
>>a
>>y
```

## Matrices

Matlab traite essentiellement des matrices. Ces matrices peuvent être de différente taille : matrice 1x1, matrices lignes, colonnes, matrices nxm. La définition d'une matrice est aisée : on introduit entre crochets les éléments ligne à ligne. Les éléments sont séparés par un espace ou une virgule et les lignes par un "`;`". Un élément d'une matrice est désigné par ses indices. Ainsi, si on veut l'élément $A_{ij}$ de la matrice A, on entre la commande suivante `A(i,j)`, où i et j sont respectivement les numéros de ligne et de colonne de l'élément considéré. De la même manière, on peut accéder à un ligne ou une colonne particulière d'une matrice en remplaçant l'indice colonne ou l'indice ligne par un "`:`". Une manière très facile de remplir une matrice ligne dont tous les éléments sont espacés d'un certain pas est le suivant : `a=[n0:p:nf]` où n0, p et nf sont respectivement le premier nombre, le pas et le dernier nombre.

**Ex 4 :**

- Pour entrer la matrice suivante : $\begin{pmatrix} 1 & 2 & 3 \\ 6 & 5 & 4 \\ 7 & 8 & 9 \end{pmatrix}$, tapez les commandes :

```
>>A = [1 2 3 ; 6 5 4 ; 7 8 9]
>>A(2,3)
>>A(3,:)
```

- Entrez la matrice b : $\begin{pmatrix} 1 & 5 & 9 & 10 \\ 2 & 7 & 6 & 1 \\ 8 & 4 & 3 & 8 \end{pmatrix}$ et calculez `z = cos²(b13)`
- Entrez une matrice ligne dont le premier élément est -5, le dernier est 5 et le pas est de 0.5.

Différentes opérations sont possibles sur les matrices. L'addition, la soustraction, la multiplication en respectant les dimensions respectives des différentes matrices.

**Ex 5 :**

- Trouvez `C = kA` où `k = 5`
- Trouvez `D = A+C`
- Trouvez `E = A*b`

D'autres fonctions (création automatique de matrices diagonales, zéros, uns, random,…; opérations sur les indices, taillle des matrices, inversion de matrices…) existent et sont

détaillées dans `help elmat`. Lors des exercices, le temps, la position d'un point seront souvent stockés dans des matrices lignes et différentes opérations seront effectuées sur les éléments d'une matrice. Pour indiquer que les opérations portent sur les éléments de la matrice et non sur la matrice elle-même, il suffit de faire précéder l'opérateur par un point.

**Ex6 :**

Créez une matrice `X(3x4)` dont les éléments sont déterminés de manière aléatoire. Calculez la matrice `Y` dont les éléments `Yij = cos²(10*Xij).`

## Racines d'un polynome

*Roots*

Parmi les nombreuses possibilités de Matlab, on trouve la résolution d'une équation polynomiale du degré n. Pour cela on utilise la commande roots(p) où p est un vecteur contenant les coefficients du polynôme rangé par ordre de degré décroissant.

**Ex7:**

Dans l'exercice sur le vol parabolique de la première séance, trouvez le temps pour lequel z s'annule.

## Visualisation

Une fonctionnalité très intéressante de Matlab consiste en la visualisation des résultats. Des graphiques à deux et trois dimensions peuvent être obtenus.

*Plot*

La fonction `PLOT` permet de tracer des graphes. `PLOT(x,y,s)` où `x` est un vecteur ligne contenant les abscisses, `y` un vecteur ligne contenant les oordonnées et `s` est une chaine de caractère précisant le style de la courbe tracée. La courbe se dessine dans une fenêtre séparée. Il est possible de créer plusieurs fenêtre dans lesquelles on viendra dessiner différents graphe. Pour cela il suffit de créer une fenêtre supplémentaire par la commande `figure(n)` où `n` est le numéro de la fenêtre. Par défaut, à chaque fois que l'on dessine un nouveau graphe dans une fenêtre, ce dernier écrase la graphe précédent. Si l'on veut conserver le graphe précédent, il suffit d'entrer la commande `hold on`. Cette commande garde son effet jusqu'à la prochaine commande `hold off`.

*Figure*

*Hold*

**Ex8 :**

- Créez une matrice `t` représentant le temps dont le premier élément est 0, le dernier est 10 et le pas est 1. Créez ensuite la matrice aléatoire `x` de même dimension que `t`. Tracer ensuite le graphe de `x` en fonction de `t` en rouge.
- Dans une autre fenêtre tracez un demi-cercle centré sur (0,0) et de rayon 10 en rouge.
- Tracez la trajectoire de l'avion de l'exercice 1 (vol parabolique) de la première séance.

  Pour rappel, $\begin{cases} x = v_0 \cos q_0 \cdot t \\ z = -gt^2/2 + v_0 \sin q_0 \cdot t \end{cases}$, $v_0$=800km/h, g=9.79 m/s², ?$_0$=10°

Une autre fonction de visualisation permet de dessiner un graphe en coordonnées polaires grace à la fonction `POLAR(theta,rho)`. Attention, theta et rho doivent être deux matrices lignes de même dimension.

*Polar*

**Ex 9 :**

Dans l'exercice 3 de la première séance, tracez les graphes des trajectoires suivie par le point pour les différentes valeurs de n (n=-1/2, n=1/2 et n=-2).

Pour rappel, la trajectoire est donnée par $r^n = a^n \cos(nq)$.

## 4. Programmation Matlab

Comme nous l'avons déjà dit, Matlab est également un langage de programmation. Déclaration de variables, opérations sur ces variables, sont les premières étapes de la programmation. Abordons maintenant les caractéristiques de programmation plus avancées de Matlab bien que classiques.

*If*

### If

L'utilisation de la condition if est simple :

```
>>if condition
      Action
  elseif condition
      Action
  else Action;
  end
```

*For*

### For

```
>>for i=a:p:b, Action; end
```

Où i est le compteur partant de a incrémenté à chaque pas de p et terminant à b. a n'est pas nécessairement plus petit que b et le pas peu être quelconque (négatif, positif, unitaire, 2,…).

*While*

### While

```
>>while condition,
      Action;
  end
```

### Fonctions - Scripts : M-Files

Des fichiers contenant du code Matlab sont appelés des M-Files (fichiers M). Il existe deux sortes de fichier M : les fonctions et les scripts. Ils sont appelés soit depuis l'environnement Matlab, soit depuis d'autres fonctions ou scripts. Souvent, lors de la programmation Matlab, le programme principal est lui aussi un objet .m appelé depuis la fenêtre de dialogue. L'appel du fichier M lance son exécution. Une fonction accepte des arguments en entrée et renvoie un argument. Un script n'accepte pas d'argument ni en entrée ni en sortie et ne fait qu'exécuter une série de commandes Matlab. Un script travaille sur les variables globales de l'environnement tandis qu'une fonction opère par défaut sur des variables locales à la fonction. Lorsque l'on fait appel à un fichier M, cela se passe comme si l'on faisait appel à une boîte noire dont on ignore le contenu (sauf évidemment si l'on a écrit la boîte). On sait ce qu'elle fait mais pas comment elle le fait.
Ces fonctions sont écrites dans des fichiers .m à partir d'un éditeur de texte. Matlab possède son propre éditeur de texte que l'on lance par la commande `edit`.

*Edit*

La syntaxe utilisée pour une fonction est la suivante :

```
function [a,b]=function_name(c,d)
%commentaires définissant la fonction et qui apparaîtront %lors de
l'appel à l'aide sur la fonction. Attention, le %nom de la
fonction doit impérativement commencer par une %lettre.
Commandes Matlab
```

La première ligne est la déclaration de la fonction : elle comprend le mot clé function suivi des arguments de sortie (ici `[a,b]`), du nom de la fonction et des arguments d'entrée. Le nombre d'arguments en entrée et en sortie dépend de la fonction. Si une fonction ne renvoie aucun argument en sortie, on utilisera la syntaxe suivante :
`function function_name(c,d)`. De même si une focntion n'accepte pas d'argument en entrée, la syntaxe sera `function [a,b] = function_name`. Il est conseillé de sauver le fichier M sous le nom de la fonction (*function_name.m*). Si les noms sont différents, c'est le nom du fichier qui sera pris en compte par Matlab. Attention, veillez à sauver vos fonctions et scripts sous un nom contenant vos initiales dans le chemin donné par défaut par l'éditeur de Matlab *(\\CFAO-SERVER\ Data\Mathlab\)*.

**Ex 10 :**

- Ecrivez une fonction qui calcule la moyenne de deux nombres.
- Ecrivez une fonction qui accepte en entrée deux points, qui calcule le point milieu entre ces points et qui dessine le tout. (2 dimensions). Utilisez la fonction créée précédemment.

**Ex 11 :**

Ecrivez une fonction qui dessine la trajectoire de l'exercice 3 de la séance 1 et qui adapte le dessin (et les bornes de calcul) en fonction de la valeur de n. Pour cela, créez une fonction qui accepte en entrée les bornes, le pas et n. Dans cet exercice, on considère que n ne peut prendre que les trois valeurs (-1/2 1/2 -2).

Pour rappel, la trajectoire est donnée par $r^n = a^n \cos(nq)$

La programmation sous Matlab possède encore de nombreuses caractéristiques (classes, objets, interaction avec utilisateurs, interfaces, appel de code C Fortran,…) qui ne seront pas abordées dans le cadre de cette initiation à l'utilisation de Matlab.

## *5. Ordinary Differential Equations (ODE)*

Un grand avantage de Matlab est la résolution des équations différentielles ordinaires. Matlab possède des fonctions prédéfinies appelées *ODE Solvers* qui permettent de résoudre différents types d'équations différentielles ordinaires (voir `help funfun`).
Pour la résolution de ces équations, on applique la démarche suivante :

- On transforme l'équation différentielle d'ordre n ( $y^{(n)} = f(t, y, y', ...., y^{(n-1)})$ en un système d'équations différentielles d'ordre 1 en effectuant la substitution $y_1=y$, $y_2=y'$,…,$y_n=y^{(n-1)}$.On obtient donc le système suivant :

$$\begin{cases} y'(1) = y(2) \\ y'(2) = y(3) \\ \bullet \\ \bullet \\ \bullet \\ y'(n) = f(t, y(1), y(2), ..., y(n-1)) \end{cases}$$

- On code ce système du premier ordre dans une fonction qui accepte deux arguments t et y et qui renvoie un vecteur colonne :

```
function dy = F(t,y)
dy = [y(2);y(3);…;f(y(2);y(3);…;y(n-1)]
```

- On applique un solver au problème par la syntaxe suivante :
  [T,Y] = solver('F',tspan,y0) où F est le nom de la fonction, tspan sont les bornes du problème (l'intervalle d'intégration), y0 les conditions initiales du problème. T contient les différentes valeurs de t et Y contient dans chaque colonne les y,y',…,$y^{(n-1)}$ correspondant. **Voir annexe1**.

**Ex 12 :**

On va résoudre l'équation différentielle suivante :y''' = 3y'' + y'y.
Le système d'équation différentielles ordinaires du premier ordre est le suivant :

$y'_1 = y_2$ ou encore : Y'=F(t,Y), Y(0) = Y0 où Y = [y1;y2;y3]
$y'_2 = y_3$
$y'_3 = 3y_3 + y_2y_1$

Les conditions initiales sont les suivantes : $y = y_1 = 0$ , $y' = y_2 = 1$ , $y'' = y_3 = -1$.
On décrit le système d'ordre 1 dans le fichier M suivant :

```
function dy=odex_initiales(t,y)
dy = [y(2);y(3);3*y(3)+y(2)*y(1)]
```

Ceci peut aussi s'écrire de la manière suivante en sachant que dy doit être un vecteur colonne :

```
function dy=odex_initiales(t,y)
dy(1,1) = [y(2)]
dy(2,1) = [y(3)]
dy(3,1) = [3*y(3)+y(2)*y(1)]
```

Ensuite, l'appel au solver s'effectue en utilisant la ligne suivante :

```
>>[T,Y] = ode45('odex_initiales',[0 1],[0;1;-1]);
```

Tracez ensuite le graphe de y en fonction de t. Puis tracez le graphe de y' en fonction de t et enfin, celui de y'' en fonction de t.

Il faut remarquer que le la sortie de la fonction ode45 est constituée d'un vecteur T contenant des valeurs numériques de la variable indépendante (prises dans l'intervalle d'intégration et choisies par le solver en fonction de la méthode de résolution et du sytème) et d'une matrice Y constituée de trois colonnes contenant les valeurs numériques de y, y' et y'' calculées en chaque t du vecteur T. Cette matrice est le résultat de l'intégration du vecteur dy du système d'équations différentielles d'ordre 1.
Nous avons vu ici un appel simple du solver. Il existe des appels plus complexes permettant de fournir d'autres paramètres au système ou en utilisant l'argument `option`. Cet argument
permet de changer les paramètres d'intégration par défaut. On définit l'argument `options` en utilisant la fonction `odeset` :

```
>>options = odeset('RelTol',1e-4)
```

Ici, nous avons par exemple défini une erreur relative maximale de 1e-4. D'autres champ comme une erreur absolue peuvent être définis.
Les solveurs cachent des méthodes de résolution numérique des équations différentielles ordinaires. Ces méthodes doivent être utilisées avec prudence car leur efficacité et la valeur des résultats donnés dépendent grandement du système auquel elles sont appliquées. Normalement, il est nécessaire de connaître ces méthodes, leurs avantages et leurs inconvénients, avant de les utiliser. Dans le cadre de ce cours, nous ne verrons pas ces méthodes. Pour plus de renseignements, veuillez vous référer au cours d'Analyse de 2e Candidature. Nous nous limiterons donc à leur utilisation en tant que boîtes noires.
Cependant nous illustrerons ce problème par un exemple.

**Ex 13 :**

Résolvez l'équation de Van der Pol $y''-\mu(1-y^2)y'+y=0$ dans trois cas :

- D'abord pour mu = 1 en utilisant le solveur ode45 avec les conditions initiales y1(0)=2 et y'1(0)=0 dans l'intervalle [0 20]. Tracez y en fonction du temps.
- Ensuite, pour mu = 1000 : idem. Que constatez-vous? Tracez y=f(t) en ne représentant que les points.
- Enfin, pour μ=1000 avec ode15s les mêmes conditions initiales mais cette fois dans l'intervalle [0 3000]. Tracez y=f(t) et dans une autre fenêtre, y'=f(t).

Dans le cas d'un système d'équations différentielles ordinaires de degré 2, on agit comme suit :

On part du système : $\begin{cases} \ddot{x} = f_1(t, x, \dot{x}, y, \dot{y}) \\ \ddot{y} = f_2(t, x, \dot{x}, y, \dot{y}) \end{cases}$ ou encore $\begin{cases} \ddot{y}_1 = f_1(t, y_1, \dot{y}_1, y_3, \dot{y}_3) \\ \ddot{y}_3 = f_2(t, y_1, \dot{y}_1, y_3, \dot{y}_3) \end{cases}$

En posant $y_1 = x, y_2 = \dot{x}, y_3 = y, y_4 = \dot{y}$, on obtient le système suivant :

$$\begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = f_1(t, y_1, y_2, y_3, y_4) \\ \dot{y}_3 = y_4 \\ \dot{y}_4 = f_2(t, y_1, y_2, y_3, y_4) \end{cases}$$

Que l'on résout de la même manière que dans le cas d'une équation différentielle ordinaire. De même dans le cas d'un système d'ordre supérieur.

**Ex 14 :**

Soit un point libre lancé dans l'air. Si on néglige, les frottements de l'air, ce point parcourra une trajectoire appelée parabole de tir. Que devient cette trajectoire, si on considère que les frottements induisent une force s'opposant au sens d'avancement et proportionnelle à la vitesse?

Les équations de Newton deviennent : $\begin{cases} \ddot{x} = -\frac{kx}{m}\dot{x} \\ \ddot{y} = -\frac{ky}{m}\dot{y} \\ \ddot{z} = -\frac{kz}{m}\dot{z} - g \end{cases}$ . Résolvez ce système avec les conditions initiales

$x_0 = 0, \dot{x}_0 = 10, y_0 = 0, \dot{y}_0 = 10, z_0 = 0, \dot{z}_0 = 50$ et m=kx=ky=kz=1, g=10 dans l'intervalle de temps[0,10]. Tracez la trajectoire du point en 3D. Tracez l'évolution de la vitesse selon x au cours du temps et la position en x au cours du temps.

## Schéma de la résolution numérique d'une équation différentielle ordinaire d'ordre n.