

FSAB1104 - Commandes de base en MATLAB

Introduction

Voici quelques commandes élémentaires en MATLAB, pour avoir plus d'informations sur une commande et ses possibilités, tapez dans la console :

`help laFonctionMatlab`

Pour avoir une aide plus détaillée, utilisez l'aide de MATLAB :-).

Avant de commencer, voici une commande pratique quand vous avez demandé à MATLAB de faire une boucle infinie ou un autre truc amusant et que vous voulez arrêter les calculs sans tout couper :

Ctrl+C (Windows) Pomme+C (Mac).

1 Affichage, graphiques, ...

- `plot(vx, vy, ['aspect'], ['options'], ['val-options'])` - affiche en 2D l'ensemble des points $(v_x(i), v_y(i))$ reliés en ordre où v_x, v_y sont des vecteurs de même taille. Voici quelques options¹ utiles :
 - `['-or']` : lie les couples de points par une ligne (-), indique les points par un rond (o) et utilise la couleur rouge (r pour red). Autres exemples : `'--g'`, `'-*b'`, `'k'`, ... dans l'ordre : ligne pointillée de couleur verte (g), ligne bleue (b) avec points en étoile, ligne simple noire, ... (`help plot` pour plus de choix).
 - `['linewidth', 2]` : change l'épaisseur de la ligne (1 étant la valeur par défaut).
 - `['linestyle', 'none']` : permet de n'afficher que les points et pas les segments qui les relient.
 - `['markersize', 5]` : change la taille des points $(v_x(i), v_y(i))$.
 - `['markerfacecolor', 'red']` : change le remplissage des points.
 - ... voir aussi dans l'aide MATLAB "LineStyle" pour une liste plus complète.

Ex :

```
x = [0 1 2 3]; y = x;
```

```
plot(x, y, '-ok', 'linestyle', 'none', 'markerfacecolor', 'red', 'markersize', 5)
```

- `axis equal` - permet d'utiliser la même échelle pour l'axe des abscisses que pour celui des ordonnées (par exemple pour faire un beau cercle).
- `grid on` - permet d'afficher une grille de référence dans le graphique.
- `legend('1-plot1', '1-plot2', ...)` - affiche une légende du graphique dans l'ordre des plots effectués avec les messages précisés.

¹L'ordre dans lequel les options sont précisées n'a pas d'importance.

- `xlabel('...')`; `ylabel('...')`; - nomme l'axe des abscisse et des ordonnées.
- `hold on` - permet d'effectuer plusieurs plots dans un même graphique.
- `figure` - crée un nouveau cadre pour de nouveaux plots.
- `subplot(m,n,i)`; `plot(...)` - permet dans une même figure d'avoir plusieurs ($m \times n$) graphiques différents. L'indice 'i' allant de 1 à $m \times n$ permet de choisir dans quel subplot on se place.
Ex :

```
subplot(1,2,1); % deux graphiques côte à côte
plot(...); % effectue un plot dans le cadre de gauche (i=1)
subplot(1,2,2); plot(...); % plot dans le cadre de droite (i=2)
```
- `[x y] = meshgrid(vecx, vecy)` - renvoie une matrice correspondant à une grille quadrillée et caractérisée par les deux vecteurs. C'est particulièrement utile pour réaliser des plots en 3D. Notons que si `vecy` est le même que `vecx` (pour un carré), il n'est pas nécessaire de le préciser. (cf. ci-dessous).
- `surf(vx, vy, vz, ['options'], ['val-options'])` - surface 3D liant les points ($v_x(i), v_y(i), v_z(i)$). La commande `mesh` lie aussi les points en 3D mais par un filet (mesh). La commande `plot3` ne fait qu'afficher les points sans les lier.
Ex :

```
[x y] = meshgrid(-2.5:.1:2.5); % maillage 1/10 pour un carré centré à l'origine
z = exp(x+y); % opération effectuée en chaque noeud du maillage
surf(x,y,z);
```
- `format long` - permet d'afficher dans la console les nombres avec 15 décimales. (Ex : `pi` renverra 3.1416 par défaut. Après cette commande, on aura plutôt 3.141592653589793). `format short` permet de revenir à 4 décimales.
- `tic`; `['code matlab']`; `toc`; - affiche dans la console le temps mis pour exécuter le code matlab situé entre les commandes `tic`, `toc`.
- `fprintf('message [%format]', [vars])` - affiche un message dans la console pouvant contenir la valeur de certaines variables.
Ex :

```
x = pi^3;
fprintf('le cube de pi est : %.5f', x);
```

Le format permet de définir notamment le nombre de décimales affichées (ici 5), le "f" allant pour float. Se référer à l'aide MATLAB pour plus de détails.
- `disp(var)` - affiche le contenu de la variable dans la console.
- `home` ou `clc` - vide la console.
- `clear` - supprime les variables définies auparavant.
- `close [all]` - ferme la (les si `all`) figure(s) ouverte(s).

2 Vecteurs, Vectorisation & Matrices

- `linspace(x1, xend, nbre)` - crée un vecteur allant de `x1` à `xend` et contenant exactement `nbre` points.
- `x1:step:xend` - crée un vecteur allant de `x1` à `xend` avec un certain pas (step).

- `.*`, `./`, `.^` - opérations de vecteurs sur vecteurs. La première multiplie les composantes d'un vecteur par celles d'un autre vecteur de même taille, la seconde divise de la même manière et la troisième est l'opération puissance (les composantes du premier vecteur exposant les composantes du second).

Ex :

```
k=[0 1 3 2];
v=factorial(k);% v et k sont des vecteurs de même taille
u=v.*k;% u est encore un vecteur de même taille
```

- `sum(vec)` ou `prod(vec)` renvoie la somme (resp. le produit) des éléments du vecteur `vec`.
- `A\b` - tente de résoudre le système $A\vec{x} = b$ par élimination Gaussienne. Cette commande est, pour ce qui nous concerne, **toujours** préférable à l'inversion de la matrice `A`.
- `matTranspose = mat'` - transposition matricielle.
- `diag(vec)` - si le vecteur `vec` est de taille n , cette commande crée une matrice remplie de zéros sauf pour sa diagonale principale où l'on retrouve les éléments du vecteur.
- `conv(vec1, vec2)` - effectue la convolution de deux vecteurs. Pour ce qui nous intéresse actuellement, cela revient à considérer les deux vecteurs comme les coefficients de polynômes et à trouver les coefficients du polynôme résultant du produit de ces deux polynômes.

Ex :

```
p1 = [1 1 2]; % x^2+x+2
p2 = [1 2]; % x+2
p = conv(p1,p2); % (x^2+x+2)(x+2) = (x^3+3x^2+4x+4) -> [1 3 4 4]
```

Pour éviter d'utiliser inutilement des boucles `for`, il est souvent utile de considérer une écriture "vectorielle" de la boucle² que l'on souhaite effectuer. Cela rend le code plus concis, plus performant et ça améliore la disposition du gentil correcteur par rapport à votre copie.

Ex :

```
x=linspace(0,2,100);
y=zeros(1,length(X)-1);
for i=2:length(x)
    y(i-1)=x(i-1)+x(i);
end
```

Le code ci-dessus n'est pas vectorisé, on préférera écrire la ligne suivante :

```
y = x(1:end-1)+x(2:end);
```

3 Interpolation, Approximation

- `p = spline(X,Y)` - renvoie un polynôme par partie correspondant à l'interpolation par spline cubique des points (X,Y) .
- `ord = ppval(p, abs)` - renvoie les ordonnées correspondantes aux abscisses contenues dans le vecteur `abs` après application d'un polynôme par partie.
- `ord = spline(X,Y,abs)` - effectue les deux premières commandes en une seule.

²ce n'est cependant pas toujours possible, notamment lorsque les éléments du vecteur sont définis de manière récursive.

- `pfit = polyfit(X,Y,deg)` - approxime les points (X,Y) par un polynôme de degré précisé. `pfit` est un vecteur contenant les coefficients du polynôme.
- `ord = polyval(pfit, abs)` - renvoie les ordonnées correspondantes aux abscisses contenues dans le vecteur `abs` après application d'un polynôme (donné sous forme de vecteur de coefficients).
- `quad`, `quadl`, `dblquad`, ... - effectue des intégrales numériques. Taper `help quad` pour connaître les paramètres à entrer. Le premier sera toujours un handle (`@uneFonction`).

Ex :

```
myfun = inline('x.^2 + x + 1'); % déf. rapide d'une fonction (poss. dans la console)
quad(@(x)myfun(x), 0, 2); % intègre la fonction par rapport à 'x' entre 0 et 2
```