Tutoriel MATLAB

Introduction à l'algorithmique numérique

Année académique 2016-2017

1 MATLAB

MATLAB est un langage de programmation *interprété*, c'est-à-dire qu'il existe une ligne de commande et que l'ordinateur effectue toutes les commandes que l'on y entre une par une. À ce titre, MATLAB peut s'apparenter à une calculatrice géante. Le terme *MATLAB* vient de l'agrégation de *matrix* et *laboratory*, indiquant ainsi clairement sa philosophie : faciliter la manipulation de haut niveau de concepts mathématiques et rendre la **manipulation de matrices** un jeu d'enfant. Le principe de MATLAB est que l'on doit écrire des commandes comme on écrit des mathématiques (ou presque...). Sa philosophie de base doit par ailleurs toujours rester à l'esprit du programmeur : il faut essayer à tout prix d'écrire des commandes **matricielles et vectorielles** car leur manipulation est optimisée par l'interpréteur.

La première étape pour commencer à utiliser MATLAB est de définir des vecteurs et des matrices comme variables. Le sigle `=` indique l'affectation. Pour introduire un vecteur ou une matrice, on indique les éléments entre crochets `[ ]`. Un espace ou une virgule séparent les éléments d'une même ligne ; un point-virgule indique un passage à la ligne suivante de la matrice.

```
>> x=[3 9 4 2,1 -1 ,0]

x =

     3     9     4     2     1    -1     0

>> A=[1 -1 ,2;7 7 -2]

A =

     1    -1     2
     7     7    -2
```

Des commandes permettent de définir des matrices ou des vecteurs très fréquemment utilisés. Par exemple, `a:step:b` crée un vecteur avec tous les entiers de a à b avec un saut de *step*. La commande `zeros` crée une matrice de zéros, la commande `ones` crée une matrice de uns et la commande `eye` crée une matrice identité. La commande `linspace` divise un intervalle donné en intervalles égaux. La commande `rand` crée une matrice de nombres aléatoires. Voir l'aide `help elmat` pour plus de commandes.

```
>> a=3:7

a =

     3     4     5     6     7

>> a=3:2:7

a =

     3     5     7

>> B=ones(2,3)

B =

     1     1     1
     1     1     1

>> C=eye(3)

C =

     1     0     0
     0     1     0
     0     0     1

>> b=linspace(0,10,4)

b =

         0    3.3333    6.6667   10.0000
```

On peut accéder aux éléments d'une matrice en indiquant ses indices entre parenthèses.

```
>> A=[1 2 3;4 5 6]

A =

     1     2     3
     4     5     6

>> A(1,2)

ans =

     2
```

Comme expliqué précédemment, toutes les manipulations dans MATLAB peuvent s'effectuer **vectoriellement**. On peut donc accéder à plusieurs éléments d'une matrice et ainsi créer une sous-matrice ou un sous-vecteur. De telles sélections peuvent êtres réalisées en utilisant notamment le symbole : qui permet de sélectionner une série d'éléments.

```
>> A=[1 2 3 4;5 6 7 8;9 10 11 12]

A =

     1     2     3     4
     5     6     7     8
     9    10    11    12

>> x=A([1 2],[1 3])

x =

     1     3
     5     7

>> y=A(2,1:3)

y =

     5     6     7

>> z=A(1,2:end)

z =

     2     3     4

>> w=A(:,4)

w =

     4
     8
    12
```

De même, on peut modifier les éléments d'une matrice ou d'un vecteur.

```
>> a=1:5

a =

     1     2     3     4     5
```

```
>> a([1 3 5])=[-1 -2 -3]

a =

    -1     2    -2     4    -3
```

2 Opérations sur les matrices

Toutes les opérations usuelles sur les matrices sont définies. On peut ainsi aisément utiliser `+,-,*,^`. Le sigle `'` correspond à la transposition.

```
>> A=[1 2 ;3 4]

A =

     1     2
     3     4

>> B=[2 3;1 0]

B =

     2     3
     1     0

>> C=A+B

C =

     3     5
     4     4

>> D=A*B

D =

     4     3
    10     9

>> E=A^2

E =

     7    10
    15    22

>> F=E'
```

```
F =

     7    15
    10    22
```

Il est souvent utile d'effectuer des opérations élément par élément sur une matrice. Lorsque l'on met un point devant une opération, celle-ci s'effectue élément par élément. Ceci est surtout utile pour la multiplication, la division et l'exponentiation.

```
A =

     1     2
     3     4

>> B=[2 3;1 0]

B =

     2     3
     1     0

>> C=A.*B

C =

     2     6
     3     0

>> D=A.^2

D =

     1     4
     9    16
```

La fonction `size` renvoie les dimensions d'une matrice, tandis que la fonction `find` permet de trouver les indices correspondant aux valeurs non nulles d'un tableau.

```
>> A=[1 -1 3 4 -4]

A =

     1    -1     3     4    -4

>> size(A)

ans =
```

```
     1     5

>> i=find(A>=0)

i =

     1     3     4
```

Finalement, mentionnons la possibilité de trier des vecteurs (`sort`), de calculer le `max`, le `min`, la somme colonne par colonne (`sum`), le produit colonne par colonne (`prod`) qui sont des fonctions vectorielles très fréquemment utilisées. Cf. l'aide de MATLAB `help` ou `doc` qui renseignent la multitude de possibilités disponibles.

Exercices

1)

- Créer un vecteur ligne `x` de 5 nombres successifs entre 2 et 3 et séparés par des intervalles égaux ;
- ajouter 1 au deuxième élément ;
- créer un deuxième vecteur ligne `y` de la même dimension mais dont les éléments sont les nombres pairs successifs en commençant à 4 ;
- créer une matrice `A`, dont la première ligne est égale à `x`, la deuxième ligne est une ligne de 1 et la troisième ligne est égale à `y`.

2) Soient deux matrices

$$A = \begin{bmatrix} 4 & -5 \\ \sqrt{3} & \pi/4 \end{bmatrix},\ B = \begin{bmatrix} 2 & 3+i \\ -72/3 & 0.2 \end{bmatrix} \tag{1}$$

où i est le nombre imaginaire. Calculer les éléments suivants :

$$A+B,\ AB,\ A^2,\ A^T,\ B^{-1},\ B^T A^T,\ A^2+B^2-AB \tag{2}$$

3 Graphiques

MATLAB offre des fonctionnalités puissantes pour la création de graphiques. L'affichage le plus simple est généré par la fonction `plot`. La gestion des différentes figures se fait à l'aide de la fonction `figure`. Pour afficher plusieurs courbes sur une même figure, on active le maintien du graphique à l'aide de la commande `hold on`. Ce maintien est désactivé via la commande `hold off`. Pour sauver une figure, on utilise la fonction `print`. La fonction `close` ferme la figure active alors que la commande `close all` ferme toutes les figures.

D'autres fonctions permettent de définir la zone d'affichage (`axis`), une grille (`grid`), un titre (`title`) ou des labels pour les différents axes (`xlabel`, `ylabel`) :

```
t = 0:pi/20:2*pi;
y = sin(t);
plot(t,y)
```

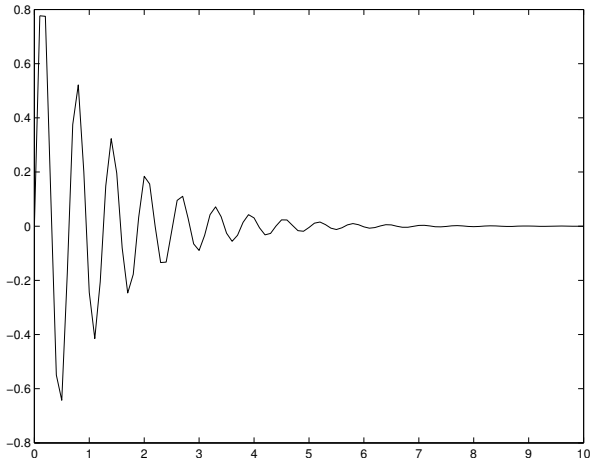
```
axis ([0 2*pi -1.5 1.5])
grid
title('sinus')
xlabel('temps t')
ylabel('sinus(t)')
```

Exercices

1) Tracer le graphique de la fonction

$$y(x) = e^{-0.8x}\sin\omega x \tag{3}$$

pour $\omega = 10$ rad/s et $x \in [0\ 10]$ s. Utiliser l'opérateur « : » afin de définir le vecteur `x` avec un incrément de 0.1 s.
Solution:



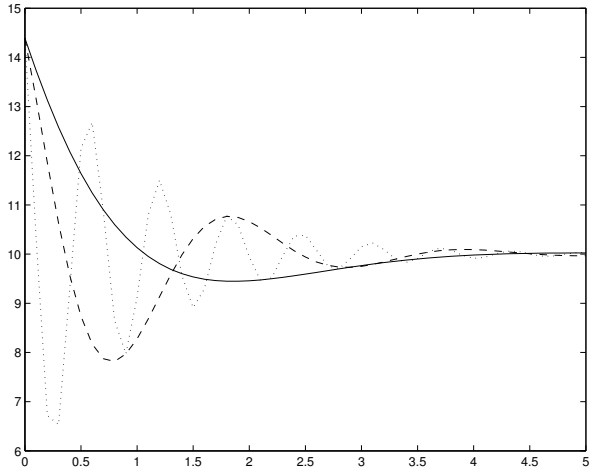
2) Soit la fonction

$$y(x) = 10 + 5e^{-x}\cos(\omega x + 0.5) \tag{4}$$

Écrire des commandes afin de tracer le graphique pour $\omega = 1, 3, 10$ rad/s et $x \in [0\ \ 5]$ s. Les trois courbes doivent apparaître en noir, avec une ligne continue pour $\omega = 1$ rad/s, une ligne en traits discontinus pour $\omega = 3$ rad/s et une ligne en pointillés pour $\omega = 10$ rad/s. Le type de trait à utiliser peut être spécifié via `plot` :

- `plot(x, y, '-')` : trait solide ;
- `plot(x, y, ':')` : trait discontinu ;
- `plot(x, y, '-.')` : trait discontinu 2 ;
- `plot(x, y, '--')` : trait discontinu 3.

Solution:



MATLAB offre de nombreuses autres options de traçage. Citons également la possibilité de tracer avec des couleurs, afin de nettement mieux différencier plusieurs courbes sur un même graphique. `plot(x, y, 'r-')` trace en trait continu rouge (« r » pour « red »), `plot(x, y, 'y:')` en trait discontinu jaune (« y » pour « yellow ») etc.

4 Structures de contrôle

MATLAB intègre des structures de contrôle parmi lesquelles on retrouve :

- `if` : contrôle conditionnel sur une expression logique ;
- `switch` : contrôle conditionnel sur une expression logique ;
- `for` : contrôle de boucle sur un compteur ou un itérateur ;
- `while` : contrôle de boucle sur une expression logique.

L'aide donne une description précise de ces structures de contrôle.

```
if (a<=b)
  do something
else
  do another thing
end
```

Les opérateurs de comparaison les plus usuels sont `<,<=,>,>=,==,~=`.

```
>> a=0;
>> for i=1:10
a=a+i;
end;
>> a

a =

    55
```

Exercices

1) La fonction exponentielle admet le développement en série

$$\exp(x) = \sum_{i=0}^{\infty} \frac{x^i}{i!} \tag{5}$$

Écrire une fonction MATLAB qui calcule cette série limitée aux $n+1$ premiers termes. On peut utiliser la fonction `factorial` et l'opérateur `^`.

2) Même exercice que précédemment mais sans utiliser la fonction `factorial` ni l'opérateur `^` (seuls les opérateurs `+`, `*` et `/` sont autorisés).

5 Fonctions

Une partie importante de l'utilisation de MATLAB concerne l'écriture de fonctions. Une fonction est souvent écrite dans un fichier séparé et peut être accédée de la même façon que les fonctions prédéfinies de MATLAB. La syntaxe de l'en-tête d'une fonction est le terme `function` suivi d'une ou plusieurs sorties entre crochets, le signe `=`, puis le nom de la fonction, et enfin de son ou ses entrées entre parenthèses.

```
function [a,b] = minmax(x,y)
```

Cette fonction doit être écrite dans le fichier `minmax.m` (c'est-à-dire que le nom du fichier doit être le même que le nom de la fonction, en rajoutant l'extension « .m »). Cette fonction admet deux entrées (représentées par `x` et `y`). Ces entrées peuvent être de n'importe quel type : scalaires, vecteurs, matrices, caractères ou autres (il existe encore d'autres types dans MATLAB que nous n'expliciterons pas ici). Cette fonction admet par ailleurs deux sorties `a` et `b` qui peuvent de nouveau être de n'importe quel type.

La fonction que nous souhaitons écrire admet deux vecteurs en entrée et donne en sortie un vecteur a tel que $a_i = \min\{x_i, y_i\}$ et un vecteur b tel que $b_i = \max\{x_i, y_i\}$.

```
function [a,b] = minmax(x,y)
nn=min(length(x),length(y));
for i=1:nn
  a(i)=min(x(i),y(i));
  b(i)=max(x(i),y(i));
end;
```

À noter que l'interpréteur de MATLAB adapte dynamiquement la taille des vecteurs. Cependant, il est conseillé d'allouer la taille des vecteurs initialement car l'allocation dynamique coûte beaucoup de temps de calcul. Dans cette fonction, il est donc de bonne pratique, avant la boucle, de commencer par

```
a=zeros(nn);
b=zeros(nn);
```

Remarquons que cette fonction n'a pas été écrite dans l'*esprit MATLAB* car il est, dans ce cas, bien plus efficace de profiter des fonctions `min` et `max` de MATLAB. En règle générale, il est toujours préférable d'éviter les boucles. Nous aurons donc

```
function [a,b] = minmax(x,y)
nn=min(length(x),length(y));
a=min([x(1:nn);y(1:nn)]);
b=max([x(1:nn);y(1:nn)]);
```

Pour une implémentation encore plus robuste, il faudrait idéalement vérifier que les deux vecteurs sont des vecteurs lignes et dans le cas contraire les transformer en vecteurs lignes.

Remarque Il est souvent utile de penser, lors de l'écriture d'une fonction, à ce que celle-ci soit capable de travailler vectoriellement ou mieux, matriciellement comme c'est le cas pour la plupart des fonctions prédéfinies de MATLAB (comme `max` et `min`).

Exercices

1. Écrire une fonction qui prend deux vecteurs colonnes x et y à n dimensions en entrée et qui renvoie en sortie une matrice $A = \frac{xy^T}{\|x\|^2}$ et un vecteur b tel que $b_i = \max\{0, x_i, y_i\}$ pour toute composante i. Si au moins un des deux vecteurs n'est pas colonne, la fonction renvoie un scalaire valant -1.

2. Écrire une fonction qui reçoit en entrée un vecteur p et un vecteur x. La sortie de la fonction renvoie un vecteur y qui calcule la valeur du polynôme représenté par $p(0) + p(1)x + p(2)x^2 + p(3)x^3 + \cdots$ en chacun des points donnés par le vecteur x. Utiliser votre fonction pour tracer le graphique de $p(x) = x^2 + 1$ sur l'intervalle $[-2, 2]$.
   (Vous venez d'implémenter la fonction `polyval` de MATLAB. Voir `help polyfun` pour plus d'informations sur le traitement des polynômes par MATLAB.)
3. Écrire la fonction `parite` qui reçoit en entrée un vecteur de nombres entiers et qui sort un vecteur avec des 1 à la place de tous les nombres pairs et des -1 à la place de tous les nombres impairs.
4. Écrire une fonction qui reçoit en entrée un vecteur correspondant à un nombre brut de secondes. La sortie de la fonction est constituée de quatre vecteurs correspondant au nombre de jours, d'heures, de minutes et de secondes de chaque composante de l'entrée.

6 Entrées/sorties

Pour le développement d'un programme interactif, la sortie écran et l'entrée clavier se font respectivement à l'aide des fonctions `disp` et `input`.

La sauvegarde et le chargement des variables se font respectivement à l'aide des fonctions `save` et `load`. Pour une gestion plus souple de la lecture et de l'écriture d'un fichier, on peut être amené à utiliser les fonctions `fopen`, `fclose`, `fprintf`, `fscanf`, `fgetl` et `fgets`.

La concaténation de chaînes de caractères est réalisée via de l'opérateur `[ ]`. Exemple :

```
>> ['Hello', 'world']

ans =

Helloworld
```

Il peut être utile de transformer des nombres en chaînes de caractères et vice versa à l'aide des fonctions `num2str` et `str2num`. Pour une gestion plus souple des chaînes de caractères, on peut employer les fonctions `sprintf` et `sscanf`.

7 Ça coince...

Il n'y a pas que l'aide qui puisse aider l'utilisateur en cas de problème. D'abord, en cas de mauvaise utilisation de MATLAB, un message d'erreur apparaît. Par exemple, le sinus du caractère `'a'` n'a pas de sens, et un message d'erreur explicite est renvoyé.

```
>> sin('a')
??? Undefined function or method 'sin' for input arguments of
type 'char'.
```

Ensuite, l'éditeur comprend un analyseur syntaxique mettant en évidence — en soulignant en rouge et dans la marge de droite — les avertissements et les erreurs d'un fichier qui y est édité. Une aide contextuelle à la résolution de ces problèmes est fournie.

Il est utile de mentionner l'existence des « nombres » `i` (le nombre imaginaire), `Inf` (l'infinité positive) et `NaN` (le nombre indéfini ou « Not-a-Number »). Il est également utile de mentionner l'existence des fonctions `is*` permettant de détecter l'état d'une entité (par exemple les trois nombres précédents peuvent être testés avec les fonctions `isreal`, `isinf` ou `isnan`).

8 Outils d'analyse des performances

Pour tester la performance d'un code et éventuellement l'améliorer, l'utilisateur peut tirer avantage du *profiler* de MATLAB. Celui-ci se contrôle via la commande `profile`, ou directement via son interface graphique (accessible par la commande `profile viewer`).

Le profiler donne des statistiques sur le nombre d'appels et le temps passé lors de l'exécution d'un code. Ces informations permettent de mettre en évidence les parties les plus gourmandes du code. Pour l'analyse d'un code très rapide, il peut se révérer utile de répéter celui-ci un certain nombre de fois, de manière à observer des temps d'exécution non négligeables.

Les fonctions `tic` et `toc` sont également très pratiques pour évaluer le temps d'exécution d'une section de programme.

Exercices

1) Comparer les deux programmes suivants :

```
tic;
for i=1:1000,
  for j=1:1000,
    x(i,j)=i+j;
  end
end
toc
```

et

```
tic;
x=zeros(1000);
for i=1:1000,
  for j=1:1000,
    x(i,j)=i+j;
  end
end
toc
```

Comment pouvez-vous expliquer cette différence de performance ?