MATLAB : prise en main

F. Delebecque et J.C. Pesquet

1 Premiers pas

Pour lancer Matlab taper `matlab`. Une fenêtre s'ouvre dans laquelle on reçoit alors le "prompt" `>>`. Chaque commande passée à Matlab est interprétée, le résultat est affiché, et l'utilisateur recoit de nouveau le prompt `>>` pour la commande suivante : c'est le **mode interactif**. Un `help` en ligne est disponible : utiliser `help item`.

Voici quelques exemples de commandes :

```
-->a=1;
```

On donne la valeur 1 à la variable `a`. Le point virgule (`;`) en fin de ligne signifie que le résultat de la commande ne doit pas être imprimé sur l'écran (seulement évalué).

```
-->c=[1 2];b=1.5
 b         =

    1.5
```

Deux commandes sur la même ligne, le résultat de la seconde étant imprimé à l'écran (pas de point virgule).

```
-->a=1;b=1.5;

-->2*a+b^2
 ans       =

    4.25
```

Les variables assignées sont gardées en mémoire. Le résultat est mis dans une variable par défaut appelée `ans`.

```
-->sqrt([4,-4])
 ans       =

   2.    2.i
```

Appel à une fonction avec un argument vectoriel. Le résulat est un vecteur ligne complexe.

```
-->X=rand(2,2);[Q,R]=qr(X)
 R  =

 - 0.7850226  - 0.3181927
```

```
   0.           0.0887097
 Q  =

 - 0.2691959  - 0.9630854
 - 0.9630854    0.2691959
```

Appel à une fonction admettant une variable d'entrée (matrice aléatoire `X`) et deux variables de sortie (matrices `Q` et `R`).

2 Vecteurs et matrices

Comme son nom l'indique Matlab est spécialement conçu pour manipuler des matrices. Matlab reconnait et manipule les variables matricielles suivantes (pour plus de détails, utiliser le `help`) : matrices constantes à coefficients réels ou complexes (`help matrices`), matrices de chaines de caractères (`help strings`), matrices creuses (`help sparse`). Les fonctions `cell` et `struct` permettent de définir et manipuler des objets plus complexes (`help cell`).

La manière la plus simple d'entrer une matrice est d'utiliser une ligne explicite d'éléments. Dans la liste, les éléments sont séparés par des blancs ou des virgules, et des point virgules (;) sont utilisés pour indiquer la fin de ligne. La liste est encadrée par des crochets `[ ]`. Par exemple, l'instruction

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

fournit comme résultat

```
A =
   1  2  3
   4  5  6
   7  8  9
```

La variable `A` est donc une matrice de dimension 33. Les éléments d'une matrice peuvent être formĆs de n'importe quelle expression Matlab. Par exemple, l'instruction

```
>> x = [-1.3 sqrt(3) (1+2+3)*4/5]
```

fournit

```
x =
   -1.3000 1.7321 4.8000
```

Une matrice avec une seule ligne ou une seule colonne est un **vecteur** , et une matrice 11 est un **scalaire**. Les éléments d'une matrice peuvent être référencés par leurs indices placés entre parenthèses. La commande `size(A)` fournit le nombre de lignes et le nombre de colonnes de `A`. `size(A)` est elle-même une matrice de taille 12, mémorisée si nécessaire par `[m n] = size(A)`.

Les "deux points"

On peut utiliser le deux points de différentes manières dans Matlab (voir `help colon`). Il sert fondamentalement à construire un vecteur dont les valeurs des éléments sont incrémentées séquentiellement. Tapez par exemple

```
>> x = 3:9
```

pour obtenir

```
x =
   3 4 5 6 7 8 9
```

L'incrément est de 1 par défaut. Pour avoir un autre incrément, tapez des commandes telles que

```
>> x = 1:0.5:4
>> x = 6:-1:0
```

La plupart des fonctions Matlab acceptent des entrées vectorielles et produisent des sorties vectorielles. La commande

```
>> y = sqrt(1:10)
```

construit un vecteur d'entiers de 1 à 10 et prend la racine carrée de chacun d'entre eux. Essayez-la.

Autre subtilité : quel est l'effet de chacune de ces deux commandes et pourquoi ?

```
>> 1+1:5
>> 1+(1:5)
```

Ce dernier exemple montre qu'il faut s'efforcer de taper les commandes de manière non ambigüe.

2.1 Manipulations

Concaténations de matrices ou de vecteurs.

Les dimensions d'une matrice ou d'un vecteur peuvent être augmentées en introduisant de nouveaux éléments empruntés à une autre matrice ou un autre vecteur. Soit par exemple le vecteur `x = [1 3 5]`. La commande

```
>> x = [x 6 8 10]
```

fournit le résultat suivant

```
x =
   1  3  5  6  8  10
```

De même la commande

```
>> y = [x;1:6]
```

donne

```
y =
   1  3  5  6  8  10
   1  2  3  4  5   6
```

Extraction d'une sous-matrice.

On peut aussi utiliser les deux points pour extraire une sous-matrice d'une matrice `A`.

`A(:,j)`	extrait la $j$ème colonne de `A`. On considère successivement toutes les lignes de `A` et on choisit le $j$ème élément de chaque ligne.
`A(i,:)`	extrait la $i$ème ligne de `A`.
`A(:)`	reforme le matrice `A` en un seul vecteur colonne en concaténant toutes les colonnes de `A`.
`A(j:k)`	extrait les éléments j à k de `A` et les stocke dans un vecteur ligne
`A(:,j:k)`	extrait la sous-matrice de `A` formée des colonnes j à k.
`A(j:k,:)`	extrait la sous-matrice de `A` formée des lignes j à k.
`A(j:k,q:r)`	extrait la sous-matrice de `A` formée des éléments situés dans les lignes j à k et dans les colonnes q à r.

Ces définitions peuvent s'étendre à des pas d'incrémentation des lignes et des colonnes différents de 1.

Transposition

Le caractère spécial (') sert à désigner la transposée d'un matrice.

La commande `A = [1 2 3;4 5 6;7 8 9]'` produit la matrice

```
A =
   1 4 7
   2 5 8
   3 6 9
```

Les lignes de `A'` sont les colonnes de `A`, et vice versa. Si `A` est une matrice complexe, `A'` est sa transposée conjuguée, ou transposée hermitienne. Pour obtenir une transposée non conjuguée, il faut employer les deux caractères point-prime (`.'`).

2.2 Opérateurs

Opérations arithmétiques

Additions `+`, soustractions `-`, multiplications `*` , puissances `^` s'utilisent avec la syntaxe matricielle habituelle.

Attention aux dimensions : on ne peut ajouter ou soustraire que des matrices de même taille. Il existe cependant une manière simple de soustraire le même scalaire à tous les éléments d'une matrice. Par exemple, `x = [1 2 3 4], x = x-1` produit le résultat

```
x =
   1 2 3 4

x =
   0 1 2 3
```

La multiplication de deux matrices n'a de sens que si leurs dimensions "internes" sont égales.

$$(A * B)_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \tag{1}$$

où n est le nombre de colonnes de `A` et aussi le nombre de lignes de `B`.

Essayez de taper `A = [1 2 3;4 5 6]; B = [7;8;9]; A*B`. Vous devriez obtenir le résultat suivant

```
ans =
     50
    112
```

Fonctions élementaires

Les fonctions telle que `abs, sin, cos, log, exp` s'appliquent à des vecteurs ou matrices de taille quelconque.

Les opérateurs `max` et `min` rendent un vecteur ligne contenant le max (min) de chaque colonne.

Les opérateurs logiques rendent des matrices de coefficients `0` ou `1`. Essayez `a=rand(3,6)`, puis `(a>0.2) & (a<0.8)`.

Chaînes de caractères

Les chaînes de caractères sont indiquées entre deux quotes, par ex. `''chaine''`. Les matrices de chaines de caractères et les matrices creuses sont manipulées avec la même syntaxe que les matrices ordinaires réelles ou complexes.

Produits internes et externes :

Le produit interne, ou **scalaire**, de deux vecteurs colonnes `x` et `y` est le scalaire défini comme le produit `x'*y` ou, de manière équivalente `y'*x`. Par exemple, `x = [1;2],y = [3;4], x'*y` conduit au résultat

```
ans =
    11
```

La **norme hermitienne** d'un vecteur est définie comme la racine carrée du produit scalaire du vecteur avec lui même. On peut aussi la calculer Ĕ l'aide de la fonction `norm`. Essayez par exemple de calculer la norme du vecteur `[1 2 3 4]`. Vous devriez obtenir `5.4772`.

Le **produit externe**, ou antiscalaire, de deux vecteurs colonnes est la matrice antiscalaire `x*y'`. De même, le produit externe de deux vecteurs lignes est la matrice `x'*y`.
Tout scalaire peut être multiplié par une matrice. La multiplication se fait alors élément par élément. Ainsi, `A = [1 2 3;4 5 6;7 8 9];A*2` donne le résultat suivant

```
ans =
    2  4  6
    8 10 12
   14 16 18
```

Vérifier que `2*A` donne le même résultat.

Inversions

L'inverse d'une matrice est calculée à l'aide de la fonction `inv(A)` qui n'est valide que si `A` est carrée. Si la matrice est singulière, ou non inversible, un message apparaî t. Si vous tapez `inv(A)` avec la matrice précédente, vous devriez obtenir

```
    Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND=2.937385e-18

    ans =
        1.0e+16*
        0.3152   -0.6304    0.3152
       -0.6304    1.2609   -0.6304
        0.3152   -0.6304    0.3152
```

Si ce message aparaŇt, la matrice concernée n'est pas nécessairement singulière, mais son nombre de conditionnement `rcond(A)` est si petit que la précision du calcul de l'inverse est sujette à caution. Pour vous en convaincre, calculez les valeurs propres de `A`, en tapant `eig(A)`. L'inverse d'une matrice est utilisée pour résoudre un système d'équations linéaires. Ainsi, pour résoudre le système

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & -2 & 4 \\ 0 & -2 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 2 \\ 7 \\ 3 \end{pmatrix} \tag{2}$$

vous pouvez taper `A = [1 2 3;1 -2 4;0 -2 1]; b = [2;7;3]; inv(A)*b` pour obtenir

```
ans =
    1
   -1
    1
```

vérifiez que la réponse est correcte en tapant `A*ans`. Qu'observe t'on ?

Matlab offre un autre moyen de résoudre un système linéaire, basé sur l'élimination de Gauss, qui est plus rapide que le calcul d'une l'inverse. La syntaxe est `A\b`. Elle est valable tant que `A` possède le même nombre de lignes que `b`. Essayez.

Valeurs propres

La commande `eig(A)` affiche les valeurs propres de `A` ...si `A` est carrée pas trop grande !

L'opérateur "point"

En calcul matriciel, on a souvent besoin d'effectuer une opération élément par élément. Dans matlab, ces opérations sont appelées opérations sur des réseaux ou des tableaux (*array operations*). Bien entendu, l'addition et la soustraction sont des opérations qui se font élément par élément. L'opération `A.*B` désigne la multiplication, élément par élément, des matrices `A` et `B`. Construisez arbitrairement deux matrices 33, `A = rand(3)` et `B = rand(3)`, et tapez successivement

```
>> A*B
>> A.*B
```

Qu'observe t'on ? Supposons que nous voulions calculer le carré de chaque élément de `A`. La bonne manière de spécifier ce calcul est

```
>> A_square = A.^2
```

où le point indique qu'une opération doit être effectuée sur chaque élément de `A`. Sans ce point, `A` est multipliée par `A` suivant les règles usuelles de la multiplication des matrices, ce qui conduit à un résultat totalement différent

```
>> A^2
```

Essayez également

```
>> 2.^A            % pour l'exponentielle de matrice
```

3 Variables complexes

Le nombre complexe $\sqrt{-1}$ est prédéfini dans Matlab et stocké dans les deux variables `i` et `j`. `i` et `j` sont des variables et leur contenu peut être changé. Si l'on tape `j = 5`, alors 5 est la nouvelle valeur de `j` qui ne contient plus $\sqrt{(-1)}$. Il faut taper `j = sqrt(-1)` pour restaurer la valeur originale. Notez la manière dont est affichée une variable complexe. Si l'on tape `i`, on doit voir sur l'écran

```
i =
   0+1.0000i
```

La même valeur sera affichée pour `j`. On peut entrer des variables complexes en utilisant `j`. Entrez par exemple `z1 =1+2*j` et `z2 = 2+1.5*j`. Comme `j` est considéré comme une variable, il faut utiliser le signe `*` de la multiplication, sinon apparaî tra un message d'erreur. Matlab ne fait pas de différence entre les variables réelles et une variable complexe (sinon à la mise en mémoire). Les variables peuvent être ajoutées, soustraites, multipliées et même divisées. Tapez par exemple `x = 2, z = 4 + 5*j`, et `z/x`. Les parties réelle et imaginaire de `z` sont toutes deux divisées par `x`. Matlab traite simplement `x` comme une variable dont la partie imaginaire est nulle. Une variable complexe dont la partie imaginaire est nulle est traitée comme une variable réelle. Soustrayez `2*j` de `z1` et affichez le résultat.

Matlab contient plusieurs fonctions prédéfinies pour manipuler les nombres complexes. `conj(z)`, retourne le complexe conjugué du nombre complexe `z`.

`real(z)` extrait la partie réelle du nombre complexe `z`. Tapez

```
>> z = 2+1.5*j; real(z)
```

pour obtenir le résultat

```
ans =
     2
```

De même, `imag(z)` extrait la partie imaginaire du nombre complexe `z`. Les fonctions `abs(z)` et `angle(z)` calculent le module et la phase du nombre complexe `z`. Tapez par exemple

```
>> z = 2+2*j;
>> r = abs(z)
>> theta = angle(z)
>> z = r*exp(j*theta)
```

La dernière commande montre comment retrouver le nombre complexe original à partir de son module et de sa phase.

Si `z = x+j*y` où `x` et `y` sont réels, alors `conj(z)` est égal à `x-j*y`. Vérifiez ceci pour différents nombres complexes en utilisant la fonction `conj(z)`.

Les matrices et les vecteurs peuvent avoir des éléments complexes. Par exemple, la commande

```
>> A = [1 2;3 4]+j*[5 6;7 8]
```

et la commande

```
>> A = [1+j*5 2+j*6;3+j*7 4+j*8]
```

sont équivalentes, elles produisent toutes deux la matrice

```
>> A =
      1.0000+5.0000i 2.0000+6.0000i
      3.0000+7.0000i 4.0000+8.0000i
```

4 Fonctions

Matlab contient plusieurs centaines de fonctions qui s'utilisent de manière interactive. Par exemple la commande `[Q,R]=qr(X)` réalise la factorisation QR de la matrice `X` et retourne les deux matrices de la factorisation dans les variables `Q` et `R`. Pour définir une nouvelle fonction dans Matlab il faut procéder en deux étapes. D'abord on définit la nouvelle fonction dans un fichier, en langage Matlab, et on donne au fichier le suffixe `.m`.

La première ligne du fichier doit obligatoirement respecter une syntaxe telle que :

```
function [X,Y]=myfonction(A,B,C)
```

Cette syntaxe signifie que l'on définit une nouvelle fonction appelée `myfunction` qui a ici les 3 paramètres d'entrée `A,B,C` (vecteurs, matrices,...) et retourne deux paramètres de sortie (vecteurs, matrices,...) `X,Y`. Attention, ce fichier doit s'appeler `myfonction.m` pour être reconnu par Matlab. Si la fonction `myfunction` est modifiée il faut utiliser la coomande

```
>>clear myfunction
```

pour utiliser la nouvelle fonction `myfunction`. La programmation en langage Matlab comprend, outre les appels aux fonctions, des boucles `for` ou `while`, des instructions de controle telles que `if-then-else` ou `switch-case`. Pour plus de détails utiliser la commande `help`.

5 Quelques commandes utiles

On donne ici quelques points d'entrée avec les mots mots clés correspondants à des commandes Matlab. Utiliser le `help` et le "SEE ALSO" qui termine chaque fichier `help` pour d'autres commandes et plus de détails.

- Sauvegarde : `save, load, who`
- Ecriture, lecture : `fopen, fread, fwrite`
- Comparaisons : `==`, `>=`, `>`, `=`, `&` ,`|`
- Fonctions élémentaires : `sum, prod, cumsum, cumprod, sqrt, diag, cos, max, round, sign, fft`
- Exécution d'un fichier de commandes : taper le nom du fichier
- Programmation : `function, nargnin, nargout, for, if, end, while, switch, warning, error, break, return`
- Debug : `keyboard, return`
- Matrices particulières : `zeros, eye, ones, reshape, empty`
- Algèbre linéaire : `det, inv, qr, svd, eig, qz`
- Polynomes : `roots`
- Génération de nombres aléatoires : `rand, randn`
- Classement, tri : `sort, find`
- Chaines de caractères : `eval`
- Graphiques : `plot, hold on, xaxis`
- Intégration d'équa. diffs : `ode23, ode45`
- Optimisation : `fmins`
- Systèmes dynamiques interconnectés : `simulink`

6 Quelques conseils utiles

Eviter les boucles en programmant "matriciellement" : c'est beaucoup plus rapide (essayez, utilisant les ordres `tic` et `toc` pour chronométrer). Pour les graphiques complexes (3D, animation) copier sur les démos ! De manière générale penser que tout le code Matlab est visible : des centaines de fonctions sont disponibles, et il faut donc éviter de reécrire ce qui (peut-être) existe déjà en consultant le help et le code source des fonctions de Matlab. La commande `lookfor keyword` peut aider à trouver une fonction particulière qui a un lien avec `keyword`.

7 Exemple

Voici un petit exemple de fonction Matlab `[mean,var,median]=stats(x)` qui calcule la moyenne, la variance et la médiane d'un vecteur `x`.

```
function [mean,var,median]=stats(x)
//Moyenne, variance et mediane du vecteur x
n=length(x);mean=sum(x)/n;var=sum((x-mean).^2)/(n-1);
x=sort(x);
if modulo(n,2)==0 then
      median=x((n+1)/2);   //impair
else
      median=(x(n/2)+x(n/2+1))/2;   //pair
end
```

On l'utilise alors comme ceci :

```
-->x=0:100;[mean,var,median]=stats(x)
 median  =
    50.5
```

```
var  =
   858.5
mean  =
   50.
```

8 Programmation

Voici des exemples simples de programmation. Les `...` sont des instructions Matlab quelconques. Pour plus de précision utiliser la commande `help`.

- Boucle `for` : `for i=1:n; ...; end`
- Boucle `while` : `while k<n; ...; end`
- Instruction If-then-else :

```
if x==0 then ...; end
if x==0 then ...; else ...;end
if x==0 then ...; elseif x==1 ...; else ...; end
```

- Selection :

```
switch x
  case 0
  ...
  case 1
  ...
  else
  ...
end
```

- Sortie de boucle :

```
for k=1:100; ...;
    if x>1000 then break;end
end
```

- Sortie de fonction :

```
function y=f(x)
    if x>1000 then ...; return ;end
```

9 Graphique

`x` et `y` étant deux vecteurs de même taille `n`, la commande `plot(x,y)` ouvre une fenêtre graphique et produit un graphe des points `y(i)` en fonction des points `x(i)`. (Si `x` est omis le vecteur `x=1:n` est pris par défaut).

De même, `surf(x,y,z)` affiche la surface définie par `z` aux points (x_i, y_j) (attention aux dimensions).

Pour obtenir une copie Postscript du dessin utiliser la commande

```
>>print -deps filename
```

On peut modifier un graphique (légendes, titre, ...) a posteriori (ordre `title` par exemple). Essayez aussi l'interface utilisateur `plotedit`.