Débuter en MATLAB

IFIPS Cycle préparatoire

Université Paris-Sud

Bernard Héron - Frédéric Pascal - Pierre Pansu

22 mars 2008

Table des matières

1 Introduction

MATLAB (matrix laboratory) est un système interactif de calcul numérique utilisable comme une calculette et qui dispose d'un grand nombre de fonctions, d'un langage de programmation et d'outils de visualisation graphique.
La structure de base sur laquelle travaille MATLAB est la matrice . Cela se fait sans déclarations de type et les allocations de mémoire ou les redimensionnements sont effectués

sans intervention de l'utilisateur, d'où une grande souplesse d'emploi.

2 Fonctionnement

2.1 Lancement

Pour lancer une session MATLAB, se reporter aux feuilles de TP.

2.2 L'environnement de travail

MATLAB met à disposition de l'utilisateurs plusieurs fenêtres (la fenêtre de commandes, la fenêtre de l'éditeur, la fenêtre de l'aide, la fenêtre de lancement, la fenêtre de l'historique, la fenêtre de l'occupation mémoire, la fenêtre du répertoire).
Pour l'usage que l'on en fera, on utilisera EXCLUSIVEMENT
– la fenêtre de commandes dans laquelles seront tapées les commandes,
– la fenêtre éditeur qui permet d'éditer des fichiers de commandes,
– la fenêtre de l'aide.

2.3 Les modes de travail

MATLAB fonctionne suivant deux modes, le mode interactif et le mode programmation. Dans les deux cas, l'utilisateur peut définir ses propres fonctions et les utiliser. Le premier est décrit au paragraphe 3.3.2, le second au paragraphe 6.

2.4 Accès à l'aide en ligne

Cliquer sur **?**.

2.5 Les noms de variables et de fonctions

Les noms de variables et de fonctions sont constitués d'une lettre suivie d'au plus 30 autres caractères, chacun pouvant être un chiffre, une lettre ou le caractère souligné (*underscore* en anglais). MATLAB fait la distinction entre minuscules et majuscules.

Conseil : Donner des noms significatifs aux variables et aux fonctions.

Conseil : Commencer toute nouvelle page d'instructions MATLAB par `clear all` . Cette instruction détruit toutes les variables personnelles créées auparavant.

Conseil : La commande `whos` liste les variables actives (en indiquant leur taille, ce qui est très utile pour débugger).

3 Machine à calculer

3.1 Le mode interactif

C'est le mode calculette. Les instructions sont tapées successivement dans l'ordre d'exécution dans la fenêtre de commande. Chaque ligne d'instructions est terminée par un retour-chariot (touche *Entrée*) qui valide la ligne. La ligne d'instructions est alors interprétée

et les instructions qu'elle contient sont immédiatement exécutées. Le résultat (respectivement un message d'erreur) est affiché en cas de bon fonctionnement (respectivement en cas d'erreur).
L'affichage du résultat d'une instruction est parfois indésirable : pour le supprimer, il suffit de terminer l'instruction par un point-virgule.

Conseil : Utiliser la touche $\boxed{\uparrow}$ pour rappeler une commande antérieure, puis, après éventuelle modification, la relancer par un retour-chariot.

3.2 Calcul sur des nombres

Même fonctionnement et même ordre de priorité que sur toutes les machines à calculer standard : d'abord l'élévation à une puissance ($\hat{}$), puis la multiplication et la division ($*$ et $/$) et enfin l'addition et la soustraction ($+$ et $-$). Pour modifier l'ordre des opérations ou encore pour le mettre en évidence, on utilise des parenthèses.

Fonctions : `sin, cos, tan, cot, asin, acos, atan, acot, cosh, sinh, tanh, coth, acosh, asinh, atanh, acoth, exp, log, log10, sqrt, abs, sign, mod, rem, round, floor, ceil, fix`.

Exemples :

```
x=3/2+1.9
y=1+3^4/2*5
z=(1+3)^4/2*5
a=abs(sin(x))+log(y)+exp(-z)
```

Les instructions ci-dessus donnent à x, y, z les valeurs 3.4, 203.5, 640.

3.3 Tracé de courbes

3.3.1 Bien comprendre la syntaxe

Pour tracer la courbe représentative de la fonction `sin` sur l'intervalle $[0, 2\pi]$, il suffit de taper

```
x=0:0.01:2*pi;
plot(x,sin(x))
```

On a l'impression que la première commande définit un intervalle sur lequel est définie la fonction `sin`, et que la seconde produit une courbe continue. Ce n'est pas du tout comme cela que MATLAB interprète ces commandes. Pour le voir, lancer les instructions ci-dessus, en remplaçant 0.01 par 1, et en retirant le point-virgule à la fin de la première instruction (qui empêche l'affichage du résultat). MATLAB ignore les intervalles et les fonctions d'une variable réelle, il ne connait que des listes de valeurs numériques. La commande

```
x=0:1:2*pi
```

crée une liste (aussi appelée vecteur-ligne) de nombres, 0 1 2 3 4 5 6. `sin(x)` désigne la liste sin(0) sin(1) sin(2) sin(3) sin(4) sin(5) sin(6). Autrement dit, la fonction `sin` est appliquée à chaque terme de la liste `x`. `plot(x,sin(x))` trace une ligne brisée reliant successivement les points $(0, \sin(0))$, $(1, \sin(1))$,..., $(6, \sin(6))$. Noter que MATLAB ne fera aucune différence entre une courbe représentative de fonction et une courbe paramétrée.

3.3.2 Piège

Pourquoi l'instruction

```
x=0:1:2*pi;
y=x*sin(x)
```

provoque-t-elle une erreur ? Parce que, pour MATLAB, les vecteurs-lignes sont des cas particuliers de matrices, que les matrices ont une façon bien particulière de se multiplier entre elles, qui diffère de la multiplication terme à terme (voir au paragraphe 4.3). Au lieu de `y=x*sin(x)`, il faut écrire `y=x.*sin(x)`. . De même, au lieu de `y=sin(x)/x`, il faut écrire `y=sin(x)./x`, et au lieu de `y=2^x`, il faut écrire `y=2.^x`, lorsque `x` est un vecteur-ligne. **Cette erreur est très fréquente**.

3.3.3 Principe de la commande `plot`

– **Tracé d'une ligne brisée**
Il s'agit plus exactement de tracés de lignes brisées. Si x, y sont deux vecteurs-lignes (ou deux vecteurs-colonnes) de même taille n, la commande `plot(x,y)` réalise le tracé de la ligne brisée qui relie les points de coordonnées $(x(j), y(j))$ pour $j = 1, \ldots, n$ dans une fenêtre graphique séparée. On peut choisir la couleur et le type du trait (trait plein, tirets, pointillés ou points isolés) et placer des marqueurs (cercles, croix, triangles, etc.) aux sommets de la ligne brisée (pour lister les possibilités, faire `help plot`).

Exemple : Pour tracer en rouge le graphe d'une fonction `f` définie dans un fichier `f.m` (voir au paragraphe 6), il suffit en général (pour des fonctions "lisses") de 50 à 100 points d'abscisses équiréparties pour un rendu satisfaisant.

```
x=-1+2*[0:100]/100;
plot(x,f(x),'r')
```

Les `' '` sont là pour que la lettre `r` soit interprétée comme une chaîne de caractères (en l'occurrence, la lettre r toute seule), et non comme une instruction.

– **Tracé de plusieurs lignes brisées**
Chaque nouvelle commande `plot` ouvre a priori un nouvel espace graphique dans la fenêtre graphique et efface donc le précédent tracé. Pour tracer plusieurs lignes brisées dans un même espace, on dispose de 2 possibilités.
  – 1) les lignes brisées sont tracées simultanément :
  par exemple `plot(x,f(x),'ro', x,5*sin(x),'k--', x,0*x,'b')` trace la ligne brisée définie par x et $0 * x$ en trait plein bleu, celle définie par x et $5 * sin(x)$ en pointillés noirs et trace seulement un cercle rouge à l'emplacement des points définis par x et $f(x)$.
  – 2) les lignes brisées sont tracées successivement en maintenant le tracé précédent, pour cela on dispose de la commande `hold on` qui permet de conserver le tracé tant que `hold off` ou `clf` n'est pas appelée. Reprenons l'exemple précédent

```
plot(x,f(x),'ro')
hold on
plot(x,5*sin(x),'k--')
plot(x,0*x,'b')
hold off
```

Les fenêtres graphiques sont appelées Figure No 1 (fenêtre par défaut), Figure No 2, etc. La commande `figure(n)` permet d'ouvrir (ou de préciser si elle est déjà ouverte)

la fenêtre graphique Figure No n dans laquelle on souhaite travailler.

La commande `subplot` permet d'ouvrir plusieurs espaces graphiques dans une même fenêtre.

– **Options**
Les commandes `title` , `gtext` , `legend` , `xlabel` permettent de documenter les tracés (pour afficher les règles d'emploi de ces options, faire `help title`, `help gtext`, etc).

La commande `axis` s'emploie après la commande `plot` pour recadrer le tracé. En particulier, la commande `axis equal` retrace la figure en axes orthonormés et la commande `axis([xmin xmax ymin ymax])` reconstruit la partie du graphique limitée au rectangle `[xmin, xmax]`$\times$`[ymin, ymax]`.

Signalons que Matlab dispose d'autres commandes de tracé, par exemple `loglog`, `semilogx`, `semilogy`, `polar`, `plot3d`,...

3.4 Affichage des nombres

MATLAB offre au moins quatre types d'affichage (court ou long, scientifique ou non) que l'on sélectionne avec les commandes
– `format short` (qui est le format par défaut)
– `format short e`
– `format long`
– `format long e`
Pour le nombre $x = 1/700$, ces formats donnent respectivement les affichages
0.0014
1.4286e-03
0.00142857142857
1.428571428571429e-03
mais (heureusement !) le changement de format n'affecte pas la valeur de x.
Le format sélectionné reste actif jusqu'au lancement d'une nouvelle commande `format`.

Remarque : Pour présenter des tableaux de valeurs numériques, on préfère généralement définir des formats *ad hoc* et on utilise alors la commande `fprintf` avec une description du format d'affichage inspirée du langage C.

3.5 Nombres complexes

Les variables `i` et `j` désignent le nombre complexe $\sqrt{-1}$ sauf si d'autres valeurs numériques leur ont été affectées par l'utilisateur (on évitera donc dans ce cas d'utiliser `i` ou `j` comme indice de boucle).

Exemples :

```
z1=1+2i
z2=(1+sqrt(3)*j)/2
```

Fonctions complexes : `real, imag, conj, abs, angle`.

4 Les matrices

Pour MATLAB, tout est matrice. Une *matrice* `A`, c'est un tableau rectangulaire de valeurs numériques, appelés les *éléments* de la matrice, et repérés par leur position : `A(i,j)` est l'élément situé sur la i-ème ligne et la j-ème colonne. On parle de vecteurs-ligne pour désigner les matrices à une seule ligne, et de vecteurs-colonne pour désigner les matrices à une seule colonne.

4.1 Construction

– **Vecteurs-lignes à coefficients régulièrement espacés** :
On a déjà rencontré la syntaxe commode
`u=debut :pas :fin` ou `u=[debut :pas :fin]` et plus simplement
`v=debut :fin` ou `v=[debut :fin]` lorsque `pas` vaut 1.

Exemples : `u=-10 :4 :2` équivaut à `u=[10 -6 -2 2]`
`v=2 :5` équivaut à `v=2 :1 :5` c'est-à-dire à `v=[2 3 4 5]`
`w=5 :2` équivaut à `w=5 :1 :2` c'est-à-dire à `w=[]`

– **In extenso à la main** :
On peut aussi rentrer les coefficients un par un.
- Vecteur-ligne (matrice à 1 seule ligne) : `u=[10 -7 3 8]` ou bien `u=[10,-7,3,8]`
- Vecteur-colonne (matrice à 1 seule colonne) : `v=[2 ;0 ;-5]` ou bien `v=[2 0 5]'`
- Tableau à 2 indices : `A=[10 7 ; 2 4 ; -5 0]` ou encore, terme à terme

```
A(1,1)=10 ; A(1,2)=7 ; A(2,1)=2 ; A(2,2)=4 ; A(3,1)=-5 ;
A(1,1)=10 ; A(1,2)=7 ; A(2,1)=2 ; A(2,2)=4 ; A(3,1)=-5 ;
```

Remarque : Si `A` est une matrice (à coefficients complexes), alors `A'` désigne sa trans-conjuguée et `A.'` sa transposée. Les deux coïncident si `A` est à coefficients réels.

Remarque : Si on omet de donner une valeur à certains coefficients d'une matrice que l'on crée (ou que l'on complète), alors MATLAB leur affecte la valeur 0.

– **Formes prédéfinies** :
Matlab dispose de fonctions définissant des matrices comme `eye, ones, zeros, rand, magic, hilb`...

Par exemple `eye(3)` est la matrice identité d'ordre 3, `zeros(2,4)` est la matrice nulle à deux lignes et 4 colonnes, `ones(1,5)` est le vecteur ligne `[1 1 1 1 1]`.
Attendu que si `A` est un tableau à deux indices, l'instruction `[nl,nc]=size(A)` affecte à la variable `nl` (resp. `nc`) le nombre de lignes (resp. de colonnes) de `A`, l'instruction `B=rand(size(A))` crée une matrice `B` de même dimensions que `A` et à coefficients *aléatoires* dans $[0, 1]$.

– **Par concaténation de matrices de tailles compatibles** :
Les instructions `u=[1 7 9]` puis `u=[u 3]` équivalent à `u=[1 7 9 3]`.
L'instruction `M=[ 5*ones(2,3) -eye(2) ; 1 :3 zeros(1,2) ]` crée la matrice

$$M = \begin{pmatrix} 5 & 5 & 5 & -1 & 0 \\ 5 & 5 & 5 & 0 & -1 \\ 1 & 2 & 3 & 0 & 0 \end{pmatrix}$$

4.2 Accès aux éléments d'une matrice

– **Accès à un élément à l'aide des parenthèses et des numéros d'indices** :
`u(2), A(1,3), v(end)`

– **Accès à une section d'une matrice à l'aide de vecteurs d'indices** :
si `L` et `C` sont des vecteurs d'entiers (lignes ou colonnes peu importe), `A(L,C)` est la matrice composée des éléments $A(i,j)$ avec $i \in$ `L` et $j \in$ `C`. Par exemple pour la matrice `M` définie précédemment `M([1 3],[2 :4])` est la matrice

$$\begin{pmatrix} 5 & 5 & -1 \\ 2 & 3 & 0 \end{pmatrix}$$

Si `x` est un vecteur, le sous-vecteur formé des composantes d'indice impair de `x` est `x(1 :2 :end)` tandis que `x(end :-1 :1)` est le vecteur déduit de `x` en renversant l'ordre des composantes. On utilise le deux-points comme abréviation de `[1 :end]`. Ainsi, `A( :,k)` (resp. `A(k, :)`) désigne la colonne (resp. la ligne) d'indice `k` de la matrice `A`. De même, si `C` est un vecteur d'entiers, alors `A( :,C)` est la matrice `A(1 :end,C)`.

4.3 Opérations

– **Opérations matricielles :**
`A+B, A*B, A-B` désignent les opérations matricielles au sens mathématique lorsque ces opérations ont un sens, c'est-à-dire quand les dimensions des matrices ou vecteurs `A` et `B` sont compatibles.

– **Puissance de matrice carrée :**
Si A est une matrice carrée alors
`A^n` avec $n \in \mathbb{N}$ est le produit répété n fois de la matrice A,
`A^(-n)` avec $n \in \mathbb{N}$ est égale à `inv(A)^n`,
`A^s` avec $s \notin \mathbb{Z}$ est égale à `V*D^s*inv(V)` où `[V,D]=eig(A)`.

– **Division matricielle :**
- `A\B` est la solution `X` (éventuellement au sens des moindres carrés) de `AX=B` si les dimensions sont compatibles.
- `B/A` est la solution `X` (éventuellement au sens des moindres carrés) de `XA=B` si les dimensions sont compatibles.

Remarque : Si `A` est une matrice carrée de dimension $[n,n]$, inversible et si `B` est un vecteur colonne de dimension $[n,1]$ alors `A\B` et `inv(A)*B` sont identiques mais le calcul n'est pas effectué de la même façon.

Remarque : Si `A` est une matrice carrée de dimension $[n,n]$, inversible et si `B` est un vecteur ligne de dimension $[1,n]$ alors `B/A` et `B*inv(A)` sont identiques mais le calcul n'est pas effectué de la même façon.

– **Opérations terme à terme :**
On les a déjà rencontrées (paragraphe 3.3.2) dans le cas particulier des vecteurs-lignes. Soit `A` et `B` deux matrices de mêmes dimensions
- si `op` est l'un des opérateurs `^, *, /` alors le résultat de `A.op B` est la matrice de même dimension que `A` ou `B` et dont les éléments sont les $\texttt{A}(i,j)$ `op` $\texttt{B}(i,j)$ (règle analogue si `A` et `B` sont des vecteurs de mêmes dimensions).
- le résultat de `A.\ B` est la matrice de même dimension que `A` dont les éléments sont les $\texttt{B}(i,j)$ `/`$\texttt{A}(i,j)$ (règle analogue si `A` et `B` sont des vecteurs de mêmes dimensions).

– **Opérations matrice (ou vecteur) - scalaire :**
Si `A` est une matrice, si `b` est un scalaire
- et si `op` est l'un des opérateurs `+, -, *` alors `A op b` ou `b op A` est la matrice de même dimension que `A` dont les éléments sont les $\texttt{A}(i,j)$ `op b` (règle analogue si `A` est un vecteur).
- alors `A/b` ≡ `A./(b*ones(size(A)))`
- alors `A.\b` ≡ `b./A` ≡ `(b*ones(size(A))./A`
- alors `A.^b` ≡ `A.^(b*ones(size(A)))`
- alors `b.^A` ≡ `(b*ones(size(A)).^A`

Remarque : Dans le cas de `*` et `/`, l'opération correspond à l'opération mathématique.

Remarque : Si `A` est carrée, `b^A` est la matrice `V*b.^D*inv(V)` où `[V,D]=eig(A)`.

– **Fonctions de matrices**
On peut appliquer les fonctions numériques internes (`sin, sqrt, exp, abs, ...`) énumérées en 2.1 et 2.3 à des vecteurs et des matrices. Elles opèrent alors élément par élément. Les fonctions numériques définies par l'utilisateur (cf. section 5) opèrent de même.

– **Fonctions de manipulations de matrices**
Matlab dispose d'un grand nombre de fonctions intrinsèques sur les matrices : `sum`, `prod`, `max`, `min`, `det`, `cond`, `inv`, `eig`, `svd`, `lu`, `spy`, `diag`, `triu`, `tril`, `mean`, `std`...

Exemple 1 : Si `a < b` sont deux réels, les composantes du vecteur `a+[0 :10]/10*(b-a)` constituent une subdivision régulière de l'intervalle `[a,b]` de pas `(b-a)`/10 et est équivalent à `[a :(b-a)/10 :b]`.

Exemple 2 :
Si on pose `u=[1 4 2 5]` et `v=[3 -1 0 2]` alors
`u+v` vaut `[4 3 2 7]`
`u*v'` vaut `[9]`
`u'*v` vaut la matrice carrée 4×4 `[v ; 4*v ; 2*v ;5*v]`
`u*v`, `u^v`, `u^2`, `2^u` provoquent une erreur
`u.*v` vaut `[3 -4 0 10]`
`u.^v` vaut `[1 0.25 1 25]`
`u./v` vaut `[0.3333 -4.0000 Inf 2.5000]` avec un warning
`u.\v` vaut `[3.0000 -0.2500 0 0.4000]`
`u+2` vaut `[3 6 4 7]`
`u/2` vaut `[0.5 2 1 2.5]`

`u.\2` et `2./u` valent `[2 0.5 1 0.4]`
`u.^2` vaut `[1 16 4 25]`
`2.^u` vaut `[2 16 4 32]`
`sqrt(u)` vaut `[1.0000 2.0000 1.4142 2.2361]`
`u\2` vaut `[0 ; 0 ; 0 ; 0.4]` (solution au sens des moindres carrés de `u*x=2`)
`u\v` vaut $\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.6 & -0.2 & 0 & 0.4 \end{pmatrix}$ (solution au sens des moindres carrés de `u*x=v`)

Exemple 3 : Si `A=[0 2 ; -1 0]` alors l'opération matricielle `A*A` (ou `A^2`) et l'opération terme à terme `A.^2` donnent respectivement les matrices

$$\begin{pmatrix} -2 & 0 \\ 0 & -2 \end{pmatrix}, \qquad \begin{pmatrix} 0 & 4 \\ 1 & 0 \end{pmatrix}.$$

5 Instructions de contrôle

Ce sont les boucles et les branchements. Les boucles permettent de répéter commodément une suite d'instructions ; le nombre de fois est soit connu d'avance (boucles inconditionnelles), soit déterminé au cours de l'exécution (boucles conditionnelles). Les branchements conditionnels (ou tests) permettent de choisir un traitement parmi plusieurs possibles en fonction de critères évalués lors de l'exécution.

5.1 Boucles inconditionnelles

Leur forme générale est

```
for k = val
  liste d'instructions
end
```

où k est une variable et *val* est un **vecteur-ligne**.
La *liste d'instructions* est exécutée m fois en donnant successivement à la variable k les valeurs $val(1), \ldots, val(m)$ où m(`=length(val)`) désigne la taille du vecteur *val*.

Exemple : si (x_n) est la suite récurrente définie par $x_0 = 1$ et $x_{n+1} = \sin(x_n)$ pour $n \geq 0$, on calcule x_{10} en écrivant

```
x=1;
for n=1:10
   x=sin(x);
end
x
```

Boucles imbriquées : ces boucles peuvent être imbriquées comme dans l'exemple ci-dessous (calcul des 6 premières lignes du triangle de Pascal)

```
A=zeros(6);
for i=1:6
   A(i,1)=1;
   for j=2:i
```

```
        A(i,j)=A(i-1,j-1)+A(i-1,j);
    end
end
```

5.2 Boucles conditionnelles

Elles s'écrivent

`while` *condition*
  liste d'instructions
`end`

où la *condition* s'exprime à l'aide des opérateurs arithmétiques

`<`	(strictement inférieur à)	`<=`	(inférieur ou égal à)
`>`	(strictement supérieur à)	`>=`	(supérieur ou égal à)
`~=`	(différent de)	`==`	(égale)

et des opérateurs logiques

`&`	et
`|`	ou
`~`	non

A l'exécution, la *liste d'instructions* est répétée aussi longtemps que la *condition* est satisfaite. Par exemple,

```
n=0;
x=0;
while ( x <= 0.99 ) & ( x >= -0.99 )
   n=n+1;
   x=sin(n);
end;
```

calcule le plus petit entier positif n tel que $|\sin(n)| > 0.99$ et la valeur de $\sin(n)$. On peut afficher ces deux nombres avec l'instruction `[n,x]` ou `disp([n,x])` .

Remarque : Pour écarter le risque de boucle infinie, mieux vaut écrire

`iter=0 ;`
`while` *condition* `& ( iter < 1000 )`
  liste d'instructions
  `iter=iter+1 ;`
`end`

Noter que si besoin, on peut interrompre un calcul de MATLAB en maintenant la touche *Ctrl* enfoncée et en tapant la touche C.

Remarque : Les opérandes des opérateurs arithmétiques doivent être un couple de scalaires ou de matrices de mêmes dimensions. Si *op* est l'un des 6 opérateurs arithmétiques et si `A` et `B` sont deux matrices de mêmes dimensions, alors `A` *op* `B` est une matrice `C` de mêmes dimensions que `A` et `B` dont les coefficients sont définis un à un par $\mathtt{C}(i,j) = 1$ ou 0 selon que la relation $\mathtt{A}(i,j)$ *op* $\mathtt{B}(i,j)$ est vraie ou non.
Pour MATLAB la condition `A` *op* `B` est vraie si et seulement si aucun coefficient de `C` n'est nul, c'est-à-dire ssi toutes les relations $\mathtt{A}(i,j)$ *op* $\mathtt{B}(i,j)$ sont vraies.
Les opérateurs logiques permettent de combiner plusieurs conditions.

5.3 Branchements conditionnels

Leur forme générale est

`if` *condition_1*
  liste_1 d'instructions
`elseif` *condition_2*
  liste_2 d'instructions
    ⋮
`else`
  liste d'instructions
`end`

Le nombre de blocs `elseif` *condition*, *liste d'instructions* est quelconque (éventuellement nul). De même, la présence d'un bloc `else` *liste d'instructions* à la fin est optionnelle. Seule la *liste d'instructions* qui suit la première *condition* évaluée vraie est exécutée (il se peut très bien qu'il n'y en ait aucune). Ensuite, l'exécution se poursuit par l'instruction qui vient immédiatement après le `end`.

Exemple : si a, b, c et d sont des variables ayant reçu des valeurs réelles, le rayon spectral ρ de la matrice $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ peut se calculer comme suit

```
delta= (a-d)^2 + 4*b*c;
if delta <= 0
   rho=sqrt(a*d-b*c)
else
   rho=( abs(a+d) + sqrt(delta) )/2
end
```

6 Programmation

6.1 Le mode programmation

A la différence du mode interactif, les instructions sont tapées successivement dans l'ordre d'exécution dans un fichier dont l'extension (fin du nom du fichier) est `.m`, par exemple `essai.m`, et cela se fait en utilisant un éditeur de texte. Pour faire exécuter les instructions, il suffit alors de taper le nom du fichier sans l'extension (`essai` dans le cas de l'exemple) dans la fenêtre de commandes. L'avantage de cette façon de travailler est qu'en cas d'erreur dans l'une des instructions, il n'est pas nécessaire de tout retaper : il suffit de corriger l'erreur dans le fichier, de sauvegarder le fichier et de relancer la commande. On désigne un tel fichier du nom de **script**.

6.2 Comment Matlab retrouve-t'il les scripts ?

Quand on lance la commande `essai`, MATLAB cherche le fichier `essai.m` dans le répertoire de travail. Il faut lui avoir préalablement indiqué le chemin qui mène à ce répertoire. La procédure est indiquée dans la feuille de TP.

Si MATLAB ne parvient pas à exécuter un script, il faut d'abord s'assurer que le fichier `.m` correspondant existe bien dans le répertoire de travail.

6.3 Fonctions définies par l'utilisateur

Si un même traitement doit être appliqué à divers jeux de paramètres, on a la possibilité de créer une fonction, c'est-à-dire un programme qui explicite ce traitement sur des arguments virtuels (= des noms de variables) et que l'on peut faire exécuter sur tout ensemble convenable d'arguments effectifs (= des nombres, des matrices ...).

Attention : Les lignes d'instructions d'une fonction de nom `truc` doivent impérativement être enregistrées dans un fichier de nom `truc.m`

Nous nous contenterons de montrer le principe des fonctions sur un exemple simple. Supposons qu'on ait créé un fichier `moyenne.m` contenant les lignes

```
function y=moyenne(x)
% x (argument virtuel) est suppose etre un vecteur
% y (argument de sortie) doit etre affecte d'une valeur
y=sum(x)/length(x);
```

Alors, l'instruction `mean=moyenne([1 3 -5 2 0])` affecte à `mean` la valeur 0.2.
Une fonction peut avoir plusieurs arguments de sortie. Pour illustrer ce point, modifions comme suit le fichier `moyenne.m`

```
function [moy,etyp]=moyenne(x)
% x (argument viruel) est suppose etre un vecteur
% moy et etyp (arguments de sortie) doivent etre affectes
m=length(x);
% m est une variable locale donc inconnue a l'exterieur de cette fonction
moy=sum(x)/m;
etyp=sqrt( sum(x.^2)/m - moy^2 );
```

L'instruction `[m e]=moyenne([1 3 -5 2 0])` affecte alors simultanément à `m` la valeur moyenne 0.2 et à `e` l'écart-type 2.7856... des nombres $1, 3, -5, 2, 0$. Il est à noter que l'instruction `mean=moyenne([1 3 -5 2 0])` affecte comme précédemment la valeur 0.2 (*i.e.* le premier argument de sortie) à `mean`.

6.4 Recommandations

1. Ecrire une seule instruction par ligne.
2. Eviter de donner à un script le nom d'une fonction MATLAB. Sinon, ce fichier d'instructions remplacera la fonction MATLAB pour la suite de la session de travail.
3. Mettre des commentaires et sauter des lignes blanches pour faire bien ressortir la structure du programme. Les lignes de commentaires commencent par le caractère `%`.
4. Une instruction trop longue pour tenir sur une seule ligne peut se continuer sur une nouvelle ligne (ou plusieurs) : il suffit de taper $\cdots$ (trois points consécutifs sans rien derrière) à la fin d'une ligne pour indiquer que l'instruction se poursuit sur la ligne suivante.

7 Résolution approchée d'une équation différentielle du premier ordre

7.1 Principe

Une équation différentielle du premier ordre s'écrit sous la forme $y' = F(t, y)$ où t est le temps, $t \mapsto y(t)$ la fonction inconnue, F une fonction de deux variables.

Exemples. Dans l'équation $y' = ty$, $F(t, y) = ty$. Dans l'équation $y' = -2y + 1$, $F(t, y) = -2y + 1$ ne dépend pas de t. Dans l'équation $y' = \sin t$, $F(t, y) = \sin t$ ne dépend pas de y.

7.2 Mode d'emploi des solveurs de Matlab

Il s'agit de résoudre l'équation différentielle $y' = F(t, y)$. On commence par programmer la fonction F dans un fichier `F.m`.

Si, dans la fenêtre de commande, on tape `[t  y]=ode45(@F,[t0 tfinal],y0)`, Matlab retourne un vecteur *colonne* t, subdivision de l'intervalle $[$`t0`$,$ `tfinal`$]$, et un vecteur *colonne* y contenant les valeurs aux points de t d'une solution approchée de $(\mathcal{E})$ sur l'intervalle $[$`t0`$,$ `tfin`$]$ de condition initiale $y($`t0`$) =$ `y0`.

Exemple. Résoudre $y' = ty$ sur l'intervalle $[0, 3]$ avec comme condition initiale $y(0) = 1$.

On programme la fonction $F(t, y) = ty$ dans un fichier `F.m`,

```
function z=F(t,y)
z=t*y;
```

Puis on programme la résolution de l'équation différentielle, on trace la courbe représentative de la fonction obtenue, ou on récupère la valeur approchée de la solution en $t = 1.645$.

```
[t y]=ode45(@F,[0 3],1);
plot(t,y)
```

```
sol=ode45(@F,[0 3],1);
  s=deval(sol,1.645);
```

Exemple. Résoudre le système différentiel $y' = ty$ sur l'intervalle $[0, 3]$ avec comme condition initiale $y_1' = -ty_2$, $y_2' = y_1$ sur l'intervalle $[0, 3]$ avec comme conditions initiales $y_1(0) = 0$, $y_2(0) = 1$.

On programme la fonction vectorielle $F(t, y_1, y_2) = (-ty_2, y_1)$,

```
function v=F(t,y)
z(1)=-t*y(2);
z(2)=y(1);
v=z';
```

Remarquer le $v = z'$ qui fait de $F(t, y)$ un vecteur colonne. Puis on résoud le système, et on représente les deux fonctions $t \mapsto y_1(t)$ et $t \mapsto y_2(t)$, ou bien la courbe paramétrée $t \mapsto y(t) = (y_1(t), y_2(t))$.

```
 [t y]=ode45(@F,[0 3],[0 1])
plot(y)
```

```
[t y]=ode45(@F,[0 3],[0 1])
plot(y(:,1),y(:,2))
```

Index