Une courte introduction à MatLab

Ouardane JOUANNOT

26 juin 2006

Généralités

Présentation

MatLab est un langage puissant qui ressemble de près au basic. Il est particulièrement efficace pour tout traitement mathématique de données, en revanche, l'interface avec l'utilisateur est assez rigide, ainsi vous serez obligés d'utiliser la console de MatLab pour utiliser vos programmes : en plus clair, vous ne pourrez pas programmer un jeu de vaisseau spatial avec matlab, car la partie graphique est pauvre.
En revanche, MatLab est un formidable outil mathématique. Il fait facilement et rapidement des calculs complexes et maitrise beaucoup de fonctions mathématiques usuelles ; il manipule également les complexes. Mais MatLab brille le plus dans l'utilisation de matrices. Qui plus est, MatLab n'utilise que des matrices... Même quand vous pensez manipuler un nombre, c'est en fait un matrice 1,1 que vous utilisez.

0.1 Que pourrez vous faire avec MatLab ?

Vous pourrez facilement faire des calculs répétitifs avec MatLab : vous tapez un programme. Vous entrez vos paramètres et leurs variations, et MatLab vous sort tous les résultats. Vous pouvez également tracer des graphiques, 2D ou 3D. Vous avez ensuite tout un arsenal de structures logiques qui vous permettent de classer, ou traiter des données.

0.2 Plan

Ce document traitera donc en premier lieu de la structure des programmes de MatLab, puis des déclarations de variables et des attributions de valeurs, suivi des calculs et des fonctions mathématiques. Nous verrons ensuite les structures logiques et les boucles. Une fois ceci fait, nous nous interesserons aux graphiques. Puis pour finir, nous verrons quelques algorithmes qui peuvent être utiles (tri, équations différentielles...).

Chapitre 1

Organisation de MatLab

1.1 Présentation

Un programme MatLab s'organise de la manière suivante : un fichier script est le fichier principal de votre programme, il fera appelle à diverses fonctions. Ces fonctions sont soit des fonctions définies par vous, soit des fonctions internes de MatLab.
Il est important de comprendre que ce script et ces fonctions correspondent à des fichiers. Il faut donc préciser à MatLab quels fichiers utiliser, et pour cela, vous devez vous placer dans le bon environnement. C'est ce que nous allons voir immédiatement.

1.2 Environnement

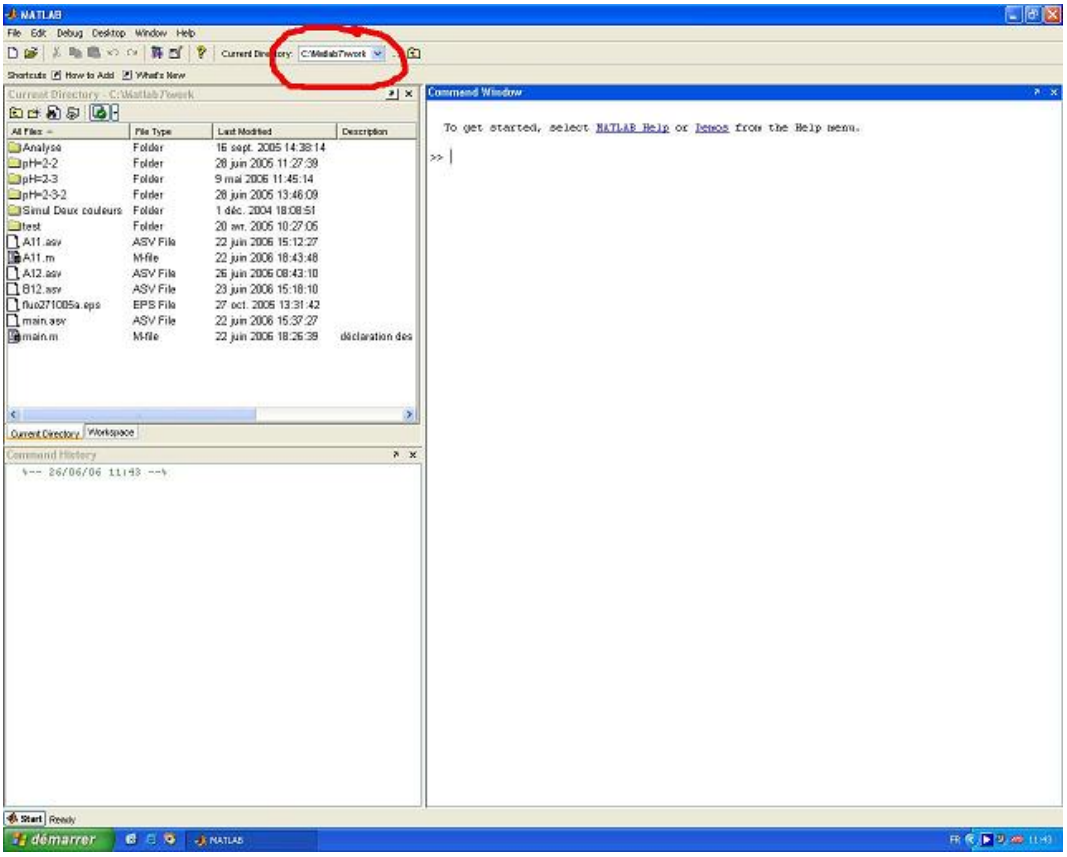
1.2.1 Qu'est-ce qu'un environnement ?

Un environnement est un simple dossier de votre disque dur. Il faut donc indiquer à MatLab dans quel dossier se placer. C'est dans ce dossier que MatLab irra chercher les fonctions que vous avez définies, ainsi que le script.

1.2.2 Comment se placer dans le bon environnement ?

Dans la fenêtre principale de MatLab, tapez simplement l'adresse de votre dossier. Par exemple $\boldsymbol{C:\backslash MatLab7\backslash Work\backslash CelluledeThomas}$.

Attention : De nombreux bugs, erreurs, et autres problèmes viennent du fait que vous ne travaillez pas dans le bon environnement. Vérifiez l'environnement lorsque vous constatez que vous avez toujours le même problème quelque soit les modifications que vous faites sur votre programme !


MATLAB
File Edit Debug Desktop Window Help
Current Directory: C:\Matlab7\work
Shortcuts How to Add What's New
Current Directory - C:\Matlab7\work
All Files
File Type
Last Modified
Description
Analyse Folder 16 sept. 2005 14:38:14
pH=2-2 Folder 28 juin 2006 11:27:39
pH=2-3 Folder 9 mai 2006 11:45:14
pH=2-3-2 Folder 28 juin 2006 13:46:09
Simul Deux couleurs Folder 1 déc. 2004 18:08:51
test Folder 20 avr. 2005 10:27:05
A11.asv ASV File 22 juin 2006 15:12:27
A11.m M-file 22 juin 2006 18:43:48
A12.asv ASV File 26 juin 2006 08:43:10
B12.asv ASV File 23 juin 2006 15:18:10
flue271005a.eps EPS File 27 oct. 2005 13:31:42
main.asv ASV File 22 juin 2006 15:37:27
main.m M-file 22 juin 2006 18:26:39 déclaration des
Current Directory
Workspace
Command History
%-- 26/06/06 11:43 --%
Command Window
To get started, select MATLAB Help or Demos from the Help menu.
>>
Start
Ready
démarrer
MATLAB

Chapitre 2

Scripts et fonctions

2.1 Scripts

Le script est votre programme principal. Parfois, c'est même le seul élément de votre programme.

2.1.1 Comment créer un script ?

Pour écrire votre script, cliquez sur l'icone nouveau de MatLab. Une nouvelle fenêtre s'ouvre, avec une page blanche. Vous pouvez commencer à taper votre programme à cet endroit. Vous pouvez également cliquer sur ***Fichier⇒Nouveau⇒M-file***

2.1.2 Syntaxe obligatoire d'un script :

Aucune commande n'est indispensable pour un script. *Conseils :* Commencez votre programme par un commentaire qui decrit votre script. Pour cela utilisez la commande *%* suivi d'un texte descriptif. Le texte devrait apparaitre en vert.

2.2 Fonctions

Les fonctions ne sont pas indispensables au bon fonctionnement d'un programme. Mais elles améliorent la lisibilité, et évitent de taper de nombreuses fois les mêmes instructions. D'autre part, MatLab n'a pas d'instructions du type GoTo (c'est une commande en basic qui permet d'aller d'un point à un autre du programme), les fonctions peuvent donc se substituer à ce type d'instruction.

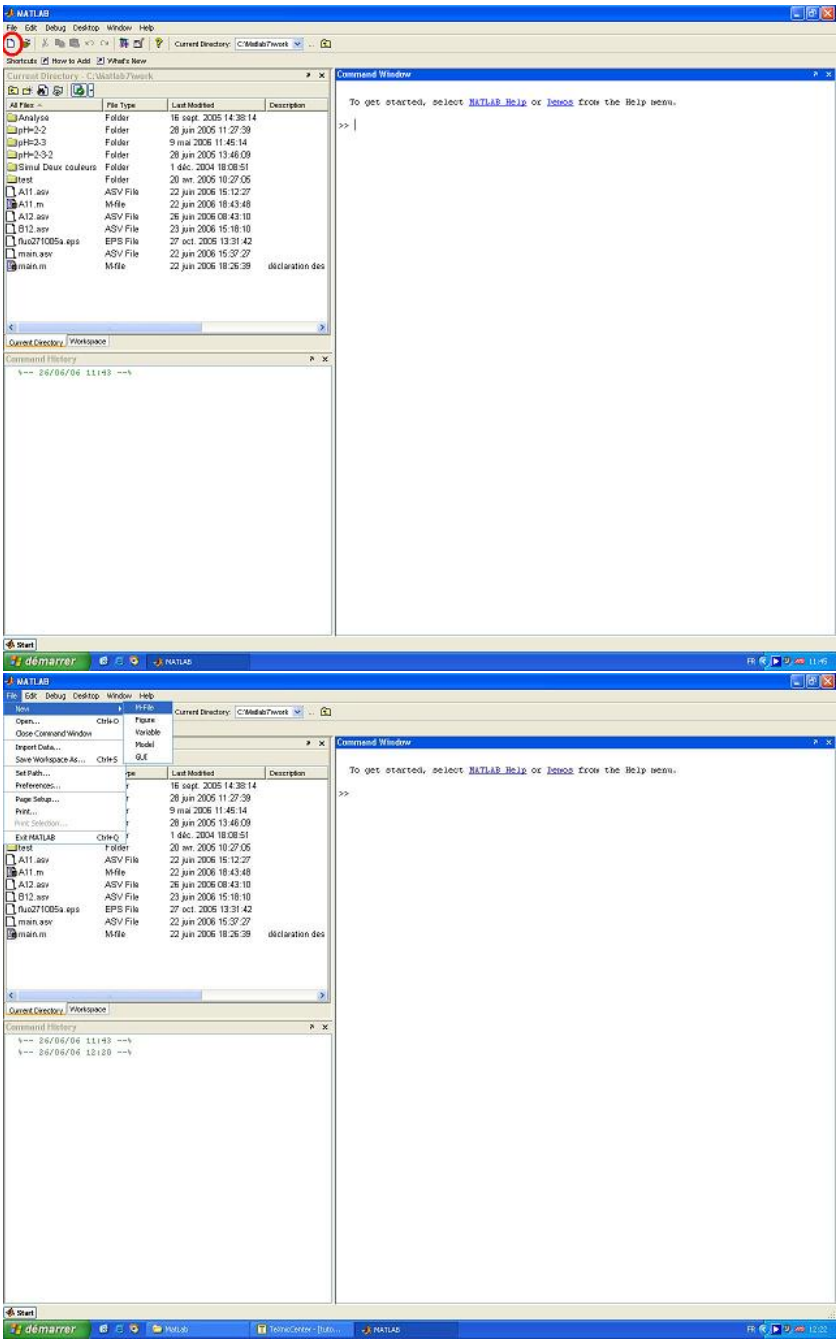
2.2.1 Comment créer une fonction ?

La démarche est exactement identique à celle de la création d'un script. En revanche, lorsque vous enregistrerez votre fonction, il faudra mettre une attention toute particulière dans le nom que vous choisirez. (voir paragraphe suivant)

2.2.2 Syntaxe obligatoire d'un script :

Votre script doit commencer par la syntaxe suivante :

function [paramètres de sortie] = nomdelafonction (paramètres d'entrée)



Les paramètres d'entrée et de sortie doivent être des noms de variables (voir variables). Ces variables sont définies juste pour la fonction, et ne peuvent pas être importées depuis votre script : c'est à dire que si vous avez une variable nommée ***exemple1*** dans votre script et que vous faites appel à une fonction qui utilise une variable ***exemple1***, ces deux variables sont distinctes, et n'ont pas forcément la même valeur. Ainsi, il est préférable de donner à chaque fois des noms différents au variables d'un script et d'une fonction.

Si vous êtes amenés à utiliser plusieurs paramètres d'entrée et de sortie, séparez ces différents noms de paramètres par des virgules.

Votre fonction doit finir par la commande : ***end***

Exemple

```
function [A,B]=produitetsomme(alpha,beta,gamma)
opérations sur alpha, beta, gamma
afféctation des résultats des opérations à A et B
end
```

Conseils : Donnez au fichier le nom de la fonction (produitetsomme dans notre exemple). Lorsque vous êtes amenez à executer une fonction un nombre très important de fois, évitez d'y ajouter des commentaires, cela ralentirait l'exécution de vos programmes.

Chapitre 3

Variables

Comme précisé en introduction, les variables de MatLab sont toutes des matrices complexes. Ainsi MatLab ne requiert pas de déclarations fastidieuses de types de variables. Il est important de le savoir, car vos fonctions fonctionneront tout aussi bien, quelque soit le type de votre paramètre d'entrée (réel, complexe, vecteur, matrice...)

3.1 Déclaration et affectation

3.1.1 Déclaration :

Comme nous venons de le voir, il n'est pas utile de déclarer les variables. C'est un important gain de temps et d'énergie, mais les erreurs sont plus faciles : il suffit que vous fassiez une faute de frappe sur le nom d'une variable et vous ne trouverez pas de résultats, ou pire, des résultats faux, sans pour autant que MatLab ne vous prévienne.

Toutefois, vous ne pourrez pas remplir une matrice au fur et à mesure, c'est à dire affecter des valeurs dans certaines cases, si MatLab ne sait pas à l'avance quel type de matrice vous remplissez. Pour faire cela, vous devez affecter la valeur 0 ou 1 dans toutes les cases de la matrice en une seule fois. Vous ne pouvez pas non plus faire des tests logiques sur une variable qui n'a pas encore de valeur (vous ne pouvez pas vérifier si la variable ***exemple1*** vaut 3 si vous n'avez pas affecté une valeur à ***exemple1***).

3.1.2 Affectation

Affectation d'un réel ou d'un complexe à une variable : Pour affecter une valeur à une variable, la syntaxe est la suivante :

nomdelavariable=valeurouvariable

Les noms de variables valides sont les chaines de caractères majuscule ou minuscule et de chiffres. Toutefois, comme il est facile de faire une faute de frappe, et que MatLab ne le détecte pas forcément, mieux vaut éviter de jouer sur les majuscules.

Affectation de 0 ou de 1 à une matrice / définition de la taille d'une matrice Pour créer une matrice remplie de 0 ou de 1 les syntaxes sont les suivantes :

– Pour affecter des zéros :

nomdelamatrice=*zeros*(nombre de cases suivant la dimension 1,nombre de cases suivant la dimension 2,nombre de cases suivant la dimension 3,nombre de cases suivant la dimension 4,....).

– Pour affecter des uns :

nomdelamatrice=*ones*(nombre de cases suivant la dimension 1,nombre de cases suivant la dimension 2,nombre de cases suivant la dimension 3,nombre de cases suivant la dimension 4,....).

Comme vous pouvez le constater, les matrices peuvent avoir autant de dimensions que vous le souhaitez.

Remplissage d'une matrice : Vous pouvez également taper entièrement votre matrice manuellement. Pour cela, la syntaxe est :

nomdelamatrice=[valeurs de la ligne 1 séparées par des espaces ; valeurs de la ligne 2 séparées par des espaces ; valeurs de la ligne 3 séparées par des espaces ; ...]

Remplissage d'une case de matrice : La syntaxe pour remplir une case est :

nomdelamatrice(coordonnée suivant la dimension 1, coordonnée suivant la dimension 2 , coordonnée suivant la dimension 3 , ...)=valeur

Affectation d'une chaine de caractères à une variable : Vous pouvez également affecter des chaines de caractères à vos variables. Pour cela, il suffit de taper la syntaxe habituelle : ***variable='chainedecaractères'***. Il faut noter que chaque lettre est en fait un vecteur de 3 nombres. Vous pouvez donc très bien faire des opérations du type ***'chemise'+1***. L'intéret peut ne pas être évident à première vue, mais il y a probablement de nombreuses applications à l'analyse de textes. Il faut pour cela s'intéresser de plus près au codage des lettres.

Affectation choisie par l'utilisateur : Si vous voulez que ce soit l'utilisateur qui choisisse la valeur d'une variable, vous pouvez utiliser la commande ***input***, qui s'utilise de la façon suivante :

variable=input('texte explicatif qui permettra à l'utilisateur de comprendre ce qu'on lui demande')

Exemples

– ***a=ones(5)*** donne ***a*** $\Rightarrow \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix}$

– ***b=zeros(3,2)*** donne ***b*** $\Rightarrow \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

– ***c=[1 3 5 7;2 4 6 8;10 11 13 14]*** donne ***c*** $\Rightarrow \begin{pmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \\ 10 & 11 & 13 & 14 \end{pmatrix}$

– ***k=26 l=k*** donne ***k*** $\Rightarrow$ ***26 et l*** $\Rightarrow$ ***26***

– ***p=ones(4,9)*** donne $\boldsymbol{p} \Rightarrow \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$

Puis ***p(3,2)=3i*** (*Note :* i est le i de $\mathbb{C}$ que MatLab comprend parfaitement, sauf si on a affecté à i une autre valeur.)

$\boldsymbol{p}$ donne alors $\boldsymbol{p} \Rightarrow \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 3i & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$

– ***K=input('Entrez la valeur de la constante thermodynamique')*** L'utilisateur verra alors apparaitre la phrase :"Entrez la valeur de la constante thermodynamique". Puis le programme attendra qu'une valeur (ou une matrice, ou un texte) soit rentré. La valeur saisie sera alors affectée à K. L'utilisateur peut taper ce qu'il veut : même si vous attendez qu'il tape un réel, il peut entrer une matrice ou un texte. Il faut alors utiliser une structure logique pour vérifier la saisie. (cf 5, page 12)

Chapitre 4

Calculs et fonctions

Une fois qu'on sait déclarer et affecter des variables, il est aisé de faire des opérations sur ces variables. Les commandes de MatLab sont assez intuitives dans ce domaine.

4.1 Opérations simples

Les opérations simples sont évidemment disponibles. Il faut une fois de plus rappeler que toutes les variables de MatLab sont donc des opérations matricielles :

- addition : *+*
- soustraction : **-**
- multiplication : ***
- division : */*
- a puissance b : ***a^ b***

Pour effectuer une multiplication ou une division terme à terme, il faut ajouter un point devant le signe de l'opération :

Exemple : $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$

$B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

A*B donne $\begin{pmatrix} 1a+3b & 2a+4b \\ 1c+3d & 2c+4d \end{pmatrix}$

Alors que A.*B donne $\begin{pmatrix} 1a & 2b \\ 3c & 4d \end{pmatrix}$

4.2 Fonctions

4.2.1 Fonctions usuelles

La plus part des fonctions mathématiques sont intégrées par matlab : les fonctions sinusoïdales, logarithme, exponetielle, etc... La syntaxe générale est ***variable=nomdelafonction(paramètre(s))***
Voici un résumé non exhaustif des fonctions disponibles :

Fonction mathématique	Syntaxe MatLab
cos	***cos***
sin	***sin***
tan	***tan***
arccos	***acos***
arcsin	***asin***
arctan	***atan***
ln	***log***
log	***log10***
e	***exp***

4.2.2 Appliquer une fonction à une matrice ?

Vous pouvez très bien appliquer une fonction à une matrice. Le résultat est une matrice de même dimension. Chaque élément de la matrice est devenu l'image par la fonction appliquée.

4.2.3 Fonction définie par l'utilisateur :

Vous pouvez également utiliser les fonctions que vous avez créé vous même (cf 2.2, page 4). La syntaxe est exactement la même.

Chapitre 5

Structure logique et boucles

L'utilisation de structures logiques est très puissante dans MatLab, car MatLab peut comparer des matrices entières, ce qui facilite beaucoup la comparaison de données. En revanche, l'utilisation de boucles est très rarement nécessaire. Si vous en utilisez une, c'est que vous passez probablement à côté d'un algorithme beaucoup plus puissant.

5.1 Structure conditionnelles

Les structures conditionnelles se basent sur la logique booléenne. Pour MatLab, $vrai = 1$ et $faux = 0$. Ainsi, pour vérifier des conditions sur des variables ou des nombres, il faudra utiliser des commandes qui renvoient 1 ou 0.

5.1.1 Conditions :

Les conditions sont des expressions qui renvoient 1 ou 0.

- Pour vérifier une égalité, on utilise la syntaxe : ***variable == variable ou valeur)***
- Pour vérifier une différence, on utilise la syntaxe : ***variable ∼= variable ou valeur)***
- Pour vérifier une inégalité, on utilise une des syntaxes suivante : ***variable > variable ou valeur,*** $<$ ***,*** $=<$***,*** $=>$

Ceci fonctionne si les variables sont des matrices 1,1, autrement dit des réels ou des complexes. Si vous voulez comparer deux matrices, il faut utiliser d'autres commandes. Si vous utilisez les commandes ***==, <, >, =<, =>)*** sur des matrices, les matrices sont comparées termes à termes. Le résultat est une matrice de 0 et de 1, qui donne 1 lorsque la relation est vérifiée entre les 2 termes comparés, et 0 sinon. Il faut ensuite utiliser une des deux instructions suivante :

- ***any(matrice)*** qui renvoi 1 si il y a au moins un 1 dans la matrice, et 0 sinon.
- ***all(matrice)*** qui renvoi 1 si tous les éléments de la matrice sont des 1, et 0 sinon.

Ainsi, pour vérifier l'égalité entre deux matrices, il faut écrire : ***all(matrice1==matrice2)***

5.1.2 Opérateurs booléens

Il existe 3 opérateurs booléens qui modifient le résultat de conditions :
- $\sim$: opérateur NON. Rend 1 pour une variable qui vaut 0. Rend 0 pour une variable qui vaut 1.
- & : opérateur ET. Rend 1 pour 1&1. Rend 0 pour 1&0 ou 0&1 ou 0&0.
- | : opérateur OU. Rend 1 pour 1|0 ou 0|1 ou 1|1. Rend 0 pour 0|0.

5.1.3 Syntaxes des structures conditionnelles :

Structure if : La syntaxe générale est :

if condition
instructions
elseif condition
instructions
(*Note : *la séquence elseif n'est pas nécéssaire. Elle peut être répétée un nombre indéfini de fois)
else condition
instructions
(*Note : *la séquence else n'est pas nécéssaire. Elle ne peut être utilisée qu'une seule fois.)
end

Si la condition qui suit ***if*** renvoit un 1, le programme executera les instructions suivantes. Sinon, il vérifiera la condition qui suit le ***elseif*** suivant. Si celle ci renvoit un 1, il exécutera les instructions suivantes. Enfin, le ***else*** final n'est pas forcément suivi d'instructions. Dans ce cas, le programme exécutera les instructions si aucune des conditions précédentes n'a été vérifiée.

5.1.4 Switch case

Il existe une autre forme de syntaxe conditionnelle : la structure switch case.
Elle s'utilise de la façon suivante :

switch(expression)
case(valeur1)
instructions
case(valeur2)
instructions
...
otherwise
(*Note : *facultatif)
end

Le programme exécutera les instructions qui correspondent à la valeur case correspondante : si la variable vaut 13, alors le programme exécutera les instructions qui suivent ***case(13)***. Sinon, le programme exécutera les instructions qui suivent ***otherwise*** si cette commande est présente, sinon, le programme ne fera rien.

5.1.5 Exemples :

Cet exemple permettra d'illustrer la structure logique, ainsi que de synthétiser les précédents thèmes.
On se propose d'écrire une fonction qui résout $0 = ay + b$.

```
function [y]=resol(a,b)
if a==0
        sitch(b)
        case(0)
                y='quelconque'
        otherwise
                y='nondefini'
        end
else
        y=a/b
end
```

Maintenant, comment écrire un script qui utilisera cette fonction ?

```
% ce programme permettra de résoudre l'équation cx+d=0
c=input('entrez la valeur de c (réel) pour l'équation cx+d=0')
d=input('entrez la valeur de d (réel) pour l'équation cx+d=0')
x=resol(c,d)
```

Vous remarquez qu'il n'est pas nécessaire, sinon déconseillé, d'utiliser les mêmes noms de variables entre le script et la fonction. Il faut bien comprendre ce qu'il se passe. Lorsque la fonction est apellée, la fonction prend la valeur de c et l'affecte dans a, et prend la valeur de d pour l'affecter dans b (seul l'ordre compte). La fonction fait des calculs, et rend la valeur de y, que le script affecte à x.

5.2 Boucles

Les boucles sont des structures qui permettent de répéter un important nombre de fois des instructions. C'est une structure plutôt lente et peu adaptée à MatLab, mais qui peut être indispensable parfois (nous le verrons pour les algorithmes de tri). Il existe plusieurs types de boucles.

5.2.1 Boucle WHILE :

En général, les langages sont pourvus de types de boucles conditionnelles : une boucle "while" et une boucle "as". MatLab ne possède qu'une boucle while. Cette boucle est une boucle conditionnelle : les instructions comprises dans la boucle se répèteront tant qu'une certaine condition est remplie. La syntaxe est :

```
while condition
instructions
end
```

5.2.2 Boucle FOR :

La boucle "for" permet de répéter une opération un nombre prédéfini de fois (il n'y pas de condition) des instructions. On définit une variable de boucle (notée j en général). On définit ensuite une borne initiale, une borne finale et un pas. A chaque itération, la variable est incrémentée du pas. Si la variable prend une valeur supérieure ou égale, la boucle s'arrête.
La syntaxe est donc :

```
for variable = borne inférieure : pas : borne supérieure (on peut sous entendre le pas, ce qui donne variable = borne inférieure : borne supérieure, le pas est alors de 1)
instructions
end
```

5.2.3 Exemples :

Prenons l'exemple d'un script qui donnera les images de points par un trinôme.

```
% Programme de calcul d'images par un trinome
a=input('Entrez le réel a du trinome ax^2 + bx +c')
b=input('Entrez le réel b du trinome ax^2 + bx +c')
c=input('Entrez le réel c du trinome ax^2 + bx +c')
binf=input('Entrez la borne inférieure')
bsup=input('Entrez la borne supérieure')
while bsup¡binf
        binf=input('Entrez la borne inférieure')
        bsup=input('Entrez la borne supérieure')
end
%Cette séquence permet de vérifier que la borne inférieure est bien inférieure à la borne supérieure. Tant que ce n'est pas le cas, le programme redemande les bornes.
pas=input('précisez l'écart que vous voulez entre les points')
for j=binf :i :bsup
        y = a*j^2 + b*j + c
end
```

5.2.4 Attention ! !

Certaines boucles peuvent être infinies ! ! (Si vous entrez un pas négatif par exemple, si vous utilisez une boucle ***while*** avec une condition toujours vraie... Il faut toujours bien réfléchir à la fin de sa boucle. Si le programme boucle infiniement, utilisez le racourcit ctrl + c. Cela met fin à l'exécution du programme. D'autre part, vous pouvez insérer la commande ***break*** dans votre script. Cela a pour effet de mettre fin à la boucle. Toutefois, les informaticiens n'aiment pas trop l'utilisation de la commande ***break*** dans une boucle "FOR", mais cela fonctionne parfaitement.

Chapitre 6

Graphiques

MatLab permet de tracer des courbes et des surfaces. Il existe donc plusieurs types de commandes, celles rattachées au graphiques 2D et celles au graphiques 3D.

6.1 2D

MatLab trace en 2D un vecteur en fonction d'un autre. Il faut donc utiliser 2 vecteurs de même dimension. Pour cela, il existe quelques commandes utiles.

6.1.1 Définir un vecteur x :

Pour définir un vecteur facilement, il faut utiliser la commande ***linspace***. Voici la syntaxe :
vecteurx = linspace(borneinférieure,bornesupérieure,nombredepoints)
ou bien
vecteurx = linspace(borneinférieure,bornesupérieure)
(Dans ce cas la, le programme prendra 100 points)

6.1.2 Tracer un vecteur en fonction d'un autre :

Une fois que vous avez défini votre vecteur, vous pouvez lui appliquer les transformations que vous souhaitez, et avoir un vecteur y. Il ne reste plus qu'à tracer x en fonction de y. Pour cela, on utilise la commande ***plot*** de la manière suivante :
plot(vecteurx,vecteury)
Vous pouvez choisir la couleur en ajoutant un argument à la fonction plot avec une lettre entre guillements, par exemple, pour obtenir une courbe rouge, il faudra taper : ***plot(x,y,'r')***.

6.1.3 Superposer des graphiques

Vous constaterez bien vite que lorsque vous demandez à MatLab de tracer une courbe, il efface les précédentes, et trace la nouvelle courbe sur une nouvelle page. Pour éviter cela et obtenir plusieurs courbes sur le même graphique, il faut utiliser la commande ***hold on***. Il suffit de la placer juste après votre commande ***plot*** dans le script, et le prochain graphique sera tracé sur la même courbe.

6.1.4 Exemple

On se propose ici de tracer $cos(x)$ et $sin(x)$ entre 0 et 2π sur le même graphique.

```
%Ce script trace cos(x) en rouge et sin(x) en bleu sur 0 - 2pi
x=linspace(0,2*pi,125)
cosinusdex=cos(x)
sinusdex=sin(x)
plot(x,cosinusdex,'r')
hold on
plot(x,sinusdex,'b')
```

6.2 3D

Les graphiques 3D sont plus difficiles à maîtriser. MatLab trace une matrice de dimension i,j en fonction d'un vecteur de taille i et d'un vecteur de taille j.

6.2.1 Définir une matrice adaptée au graphiques 3D :

Une fois définis les vecteurs x et y, vous devrez créer une matrice de dimension dim(x),dim(y). Pour cela, il faudra utiliser la fonction ***meshgrid***. Cette fonction rend deux valeurs. La première valeur est une matrice dont chaque ligne est identique au vecteur x, et qui comporte dim(y) ligne, c'est donc une matrice dim(y),dim(x). La deuxième valeur est une matrice dont chaque colonne est identique au vecteur y, et qui comporte dim(x) colonnes. c'est donc une matrice dim(y),dim(x), comme la précédente. Vous devrez donc utiliser la syntaxe :

```
vecteurx = vecteur (en utilisant linspace par exemple)
vecteury = vecteur (idem)
[matriceX,matriceY]=meshgrid(vecteurx, vecteury)
```

Vous pourrez ensuite faire les opérations que vous voulez sur matriceX et matriceY. Par exemple, si votre fonction à deux variables utilise l'exponentiel de la somme de x et y, vous devrez utiliser ***exp(matriceX+matriceY)***.

6.2.2 Tracer une surface

Il existe deux commandes pour tracer une surface : ***surf*** et ***mesh***. Nous allons voir les différences.

Avec surf : la commande ***surf*** trace une surface pleine. La couleur de la surface représente l'altitude. La syntaxe à utiliser est :

```
surf(vecteurx, vecteury, matricerésultatsxy)
```

Avec mesh : la commande ***mesh*** trace une grille de points. La syntaxe à utiliser est :

```
mesh(vecteurx, vecteury, matricerésultatsxy)
```

6.2.3 Exemple

On se propose de tracer $z = sin(x * y) * e^(x + y)$ pour x variant de 0 à 2π et y variant de -5 à 5.

```
x=linspace(0,2*pi)
y=linspace(-5,5)
[X  Y]=meshgrid(x,y)
Z=sin(X.*Y)*exp(X+Y)
surf(x,y,Z)
```

Chapitre 7

Conclusion

7.1 Quelques commandes utiles

Fonction	Utilisation	Description
;	ligne d'instructions ;	Cette commande permet d'éviter d'afficher les résultats des calculs faits à cette ligne. Utilisez les à chaque ligne pour éviter de polluer l'écran de résultats.
disp	disp('texte')	Affiche le texte entre guillement à l'écran. Utile pour préciser ou présenter des résultats
size	[nlign ncol]=size(matrice)	Cette commande renvois deux valeurs. le nombre de ligne et le nombre de colonnes.

7.2 Algorithme de tri

Cette partie aura pour but d'illustrer quelques fonctions qui ont déjà été vues, tout en proposant un algorithme de tri clefs en main.

7.2.1 tri par bulle

Le tri par bulle consiste à prendre les éléments d'un vecteur deux à deux, de les comparer, et de les permuter s'ils ne sont pas dans le bon ordre. Il suffit de répéter cette opération suffisament pour obtenir un vecteur trié.

```
function [vecttrie]=triparbulle(vecnontrie)
test=0
[un long]=size(vecnontrie)
vectest=vecnontrie
while test==0
        for i=1 :long
        if vecnontrie(i)¿vecnontrie(i+1)
                tmp=vecnontrie(i+1)
                vecnontrie(i+1)=vecnontrie(i)
                vecnontrie(i)=tmp
        end
        if vectest==vecnontrie
                test=1
        end
        vectest=vecnontrie
end
end
```

Ce tri n'est pas très efficace, mais l'algorithme est simple. Ce tri va plus vite avec un vecteur déjà presque rangé. Ici, le tri est faire par ordre croissant, pour avoir un ordre décroissant, il suffit d'inverser le sens de l'inégalité dans la structure ***if***.

7.3 Le mot de la fin

Cette introduction vous a donné les bases pour écrire des programmes très simples, mais MatLab peut faire bien plus ! Il fit certaines courbes, résoud des équations (on peut même résoudre des équations différentielles), mais pour cela, il faut se plonger dans les fonctions avancées de MatLab. D'une manière générale, la solution la plus évidente est rarement l'algorithme le plus efficace. Chaque fois que vous voulez composer un programme, réfléchissez bien aux données que vous allez utiliser, et au moyen le plus rapide d'arriver au résultat, en gardant à l'esprit que ce que MatLab fait le mieux, c'est de faire des opérations sur des matrices.

7.4 contact

Pour toute question : ouardane@crans.org