

## SÉANCE D'INITIATION

### TABLE DES MATIÈRES

1. Lancement et instructions de base	3
2. Les notions de bases	3
2.1. Les variables	3
2.2. Les opérations arithmétiques	4
2.3. Les variables spéciales et constantes	4
2.4. Format des nombres et précision des calculs	4
2.5. Matrice vide ou tableau à zéro dimension	4
2.6. Vecteurs ou tableaux à une dimension	4
2.7. Vecteurs ou tableaux à deux dimensions	6
3. La programmation avec matlab et les fichiers script	7
3.1. les fonctions et les M-fichiers	7
3.2. La programmation	8
3.3. Quelques conseils	11
4. Résolution de systèmes linéaires	12
5. Valeurs et vecteurs propres	13
6. Les complexes	13
7. Les chaînes et autres types de données	14
7.1. Les chaînes	14
7.2. Les entiers	14
7.3. Les ensembles	15
8. Les polynômes	15
9. Les graphiques	16
9.1. Les graphes bi-dimensionnels	16
9.2. Les graphes tri-dimensionnels	19
9.3. Les interfaces graphiques	19
10. Les sons	19
11. Analyse de fonctions	19
12. L'utilisation du debugger	19
13. Calcul d'intégrales et résolution d'équations différentielles	20
14. Quelques exercices	21

Références	22
------------	----

Dans ces deux TP, on n'exigera pas que tout soit fait ; il s'agit de vous montrer avant tout la richesse de matlab dans différents domaines des mathématiques. Les sections 5, 7, 10, 11, 12 et 13 sont moins importantes que les autres et pourront être omises en première lecture.

## 1. Lancement et instructions de base

*demo* : démonstration de matlab.

*intro* : lance une introduction à matlab.

*help* : très précieuse, cette aide en ligne vous permet de tout savoir sur tout. Pour savoir comment l'utiliser, faites *help help*. Vous pouvez aussi consulter le fichier helpdesk.html, plus convivial, parfois mieux documenté ; de surcroît, il contient parfois des références pour comprendre le fonctionnement de tel algorithme.

*lookfor inverse* : affiche toutes les rubriques de l'aide contenant inverse.

*cd* : changer de répertoire.

*pwd* : afficher le répertoire courant.

*dir* ou *ls* : afficher la liste des fichiers du répertoire courant.

*delete* : supprimer un fichier.

... : pour terminer une expression à la ligne suivante.

↑ : passer à l'instruction précédente (permet, par exemple, de la modifier sans la retaper).

↓ : passer à l'instruction suivante.

À partir de la fenêtre de matlab, on peut, pour lancer des instructions matlab :

- soit lancer les instructions les unes après les autres, comme une «super calculatrice» (qui fait un nombre de choses incroyable!).

- soit lancer des fichiers script (cf. section 3).

Dans les deux cas, on tape une instruction par ligne (ou plusieurs séparées par des points-virgule). Si on tape juste l'instruction, le résultat apparaît juste après. Si on rajoute un point-virgule à la fin de la ligne, l'instruction est exécutée mais son résultat n'apparaît pas.

**attention** : matlab distingue les majuscules et les minuscules.

## 2. Les notions de bases

### 2.1. Les variables

Elles sont d'un type unique (matrice) et n'exigent aucune déclaration. On pourra, pour simplifier, distinguer les scalaires (réels ou complexes ; cf. section 9), les vecteurs et les matrices. Ces variables peuvent être redimensionnées.

*who* ou *whos* : liste des variables utilisées.

*clear X* : efface la variable X.

*clear* : efface toutes les variables.

*save temp X* : sauvegarde de X dans le fichier temp (qui a pour extension .mat).

*save temp* : sauvegarde de toutes les variables dans le fichier temp.

*load temp* : chargement des variables sauvegardées dans le fichier temp.

## 2.2. Les opérations arithmétiques

Elles sont comme sur les calculatrices :  $+, -, /, *, (,), ^$  (pour des puissances entières ou réelles) et avec les règles de priorité classiques.

## 2.3. Les variables spéciales et constantes

*pi* =  $\pi$

*eps* = distance entre 1.0 et le flottant le plus proche.

*Inf* =  $+\infty$  (au sens informatique du terme). Essayez de l'atteindre.

*NaN* = Not a Number : n'est pas un nombre et exprime parfois une indétermination : *Inf* – *Inf*, 0/0

*ans* : retourne la dernière réponse.

## 2.4. Format des nombres et précision des calculs

Matlab calcule toujours en double précision mais affiche les résultat sous plusieurs formats que l'on specifie en tapant *format xxx* :

*format short* ou *format*

*format long*

*format short e*

*format long e*

*format hex*

*format +*

*format bank*

*format rat*

## 2.5. Matrice vide ou tableau à zéro dimension

Si vous faites

`>> x = []` (i.e. le caractère [suivi du caractère ])

x contient la matrice vide. Cela peut servir si on définit une matrice que l'on agrandit successivement ; comme les récurrences que l'on initie à  $n = 0$ , on peut initialement avoir besoin d'une matrice vide.

## 2.6. Vecteurs ou tableaux à une dimension

Il y a plusieurs façons de saisir un tableau :

`>>x=[6 5 6 ]`

`>>x=[6, 5, 6 ]`

Ce vecteur est considéré comme une matrice à une ligne et trois colonnes :

`>> size(x)`

`>> [m,n]=size(x)`

`>> length(x)`

On peut concaténer deux vecteurs lignes :

`>> v=[x 1 2 3]`

```
>> w=[1 x 2 3]
```

On récupère les composantes d'un tableau en spécifiant son indice entre parenthèses :

```
>> v(2)
>> w(89)
```

Pour obtenir un vecteur dont les composantes sont espacées d'un pas constant avec la première et la dernière composantes sont connues :

```
>> x=0 :0.25 :1
>> y=-pi :0.3 :pi
>> z=linspace(-pi,pi,4)
```

Les opérations sur les vecteurs à n composantes sont naturellement celle de l'espace vectoriel  $(\mathbb{R}^n, +, *)$  :

```
>> x=[0 7 8 ] ; y=[8 5 3] ;
>> x+y
>> x-y
>> x-2
>> 5*y
>> y'
```

En faisant précéder d'un point les opérateurs \*, /, et ^, on effectue des opérations élément par élément :

```
>> x.*y
>> x.^2
>> x./y
>> x.^y
```

De même, en utilisant toutes les fonctions classiques (sin, exp, cos, sqrt,...) de matlab, on réalise ces opérations élément par élément.

```
>> y=[pi/3 pi/4 pi/6 ] ;
>> sin(y)
```

Il existe un grand nombre de fonctions matlab opérant directement sur les vecteurs. Citons :

*sum* : sommes des composantes d'un vecteur

*prod* : produit d'un vecteur

*mean* : moyenne des composantes d'un vecteur

**Exercice 2.1.** afficher les vecteurs  $[1^2, 2^2, \dots, 10^2]$ ,  $[\sin(1), \dots, \sin(10)]$ ,  $[\sin(1) - 1^3, \dots, \sin(10) - 10^3]$ , et calculer la somme  $\sum_{k=1}^{10} \sin(k) - k^3$ .

**Exercice 2.2.** Calculer par la méthode des rectangle  $\int_0^{\frac{\pi}{2}} \sin(x)dx$  en prenant 10, 100 puis 10000 points et comparez la à sa valeur exacte.

**Exercice 2.3.** Comparer avec la méthode des trapèze en utilisant l'instruction *trapz* :

```
>> n=10;
>> x=0 :pi/(2*n) :pi/2;
>> y=sin(x);
>> trapz(x,y)
>> (pi/(2*n))*trapz(y)
```

## 2.7. Vecteurs ou tableaux à deux dimensions

Saisie d'une matrice :

```
>> a=[1 2 3; 6 7 8]
>> b=[1 2 3
      6 7 8]
```

On peut aussi les saisir élément par élément.

Parfois, il peut aussi être très utile de rentrer une matrice à partir d'un fichier (par exemple un fichier de données déterminés par un autre logiciel ou pour sauvegarder des tableaux de données sur disque dur).

- faire un petit fichier de données *dudu.dat* (l'extension .dat est obligatoire) :

1	2
1	76
45	87

- le sauvegarder dans le répertoire de travail
- taper *load dudu.dat*. A quoi correspond *dudu*, pour matlab, et quelle est sa valeur ?
- si on tape *save temp dudu*, que se passe-t-il ?

Voici un certain nombre d'opérations sur les matrices ; à vous de les utiliser et d'en saisir le fonctionnement.

```
>> a=[1 2 3; 4 5 6; 7 8 0]
>>b=a'
>>c=a+b
>>c=a+b;
>>a*b
>>a^3
>>expm(a)
>>logm(a)
>>sqrt(m)
>>poly(a)
>>det(a)
>>magic(6)
>>a=magic(10)
```

```
>>a(1 : 5, 3)
>>a(1 : 5, 7 : 10)
>>a(:, 3)
>>a(1 : 5, :)
>>a(1,[1 3 5])
>>a(:, 10 : -1 : 1)
>>a(:)
>>a(:) = 1 : 100
```

Pour ces huit dernières opérations sur les matrices, vous pouvez consulter helpdesk à la rubrique colon.

Regardez l'aide pour utiliser :

*rand*  
*zeros*  
*ones*  
*eye*  
*linspace*  
*logspace*  
*max*  
*min*  
*sort*  
*sum*  
*prod*  
*cumsum*  
*cumprod*  
*rank*  
*pinv*  
*trace*

On peut aussi afficher, plus joliment, la matrice x, avec un quadrillage rouge et de caractères de taille 12 :

```
>> x=rand(5);
>> pltmat(x,'matrice aléatoire uniforme','r',12);
```

### 3. La programmation avec matlab et les fichiers script

#### 3.1. les fonctions et les M-fichiers

Dans les exemples précédents, vous avez utilisé un certain nombre de fonctions de matlab (par exemple, *det(a)* renvoie le déterminant de la matrice *a* : c'est une fonction à une variable, qui à toute matrice carrée renvoie son déterminant.

On peut définir soit-même ses propres fonctions. On veut définir la fonction de  $\mathbb{R}^3$  dans  $\mathbb{R}^2$  qui à toute matrice  $1 \times 3$ ,  $a=(x,y,z)$  associe la matrice  $1 \times 2$ ,  $b=(u,v)$  où

$$\begin{cases} u = \sqrt{x^4 + y^6} - z, \\ v = xyz, \end{cases}$$

On lance l'éditeur matlab (à partir de la fenêtre matlab, on fait file/new) ; avec cet éditeur, on tape :

```
function res = dudu(a)
res(1)=sqrt((a(1))^4+(a(2))^6)-a(3);
res(2)=a(1)*a(2)*a(3);
```

On sauve ce fichier avec le nom dudu (l'extension est .m par défaut) On lance

```
>> a=[1 4 5];
>> dudu(a)
```

On peut aussi définir la fonction avec un format légèrement différent :

```
function [x,y] = dudu(a)
x=sqrt((a(1))^4+(a(2))^6)-a(3);
y=a(1)*a(2)*a(3);
```

Quelle est la différence entre les deux définitions précédentes et la troisième possibilité :

```
function [x,y] = dudu(a1,a2,a3)
x=sqrt((a1)^4+(a2)^6)-a3;
y=a1*a2*a3;
```

Ainsi, on créera ses propres fonctions, chacune d'elle étant enregistrée dans un fichier xxx.m où xxx est le nom de cette fonction. Si on en définit plusieurs, on testera chacune d'elle indépendamment des autres. Au maximum, on essayera de faire des fichiers script de façon à utiliser matlab comme un véritable langage de programmation (cf. section 3.2).

Les variables utilisées localement dans une fonction ne sont reconnues que dans cette fonction. Si on veut utiliser des variables globales, on les définira par *global* (mais, on évitera le plus possible le recours à ces variables globales). Dans une fonction, les variables d'entrées ne sont pas modifiées au sein de la définition de la fonction.

**attention** : si on décide de stocker des fichiers script dans un répertoire différent de celui où l'on travaille, il faut l'indiquer à matlab, de façon à ce que, à chaque appel de fonction, il sache où elles se trouvent. Pour cela, on utilise *editpath* pour rajouter les différents répertoires où se trouvent les M-fichiers (de préférence à la fin).

### 3.2. La programmation

Aux M-fichiers de fonctions se rajoutent les M-fichiers de programmes (les fichiers script) ; plutôt que de taper les instructions les unes après les autres dans la fenêtre matlab, il peut être plus agréable de les mettre toute dans un M-fichier de commande, qui sera édité à partir de l'éditeur et lancé à partir de la fenêtre matlab, en tapant *nomfichier* ou bien *run nomfichier* où *nomfichier* est le nom du M-fichiers (sans l'extension .m).

Bien entendu, matlab est un langage récursif : une fonction peut s'appeler elle-même. Parfois, les informaticiens sont réticents à utiliser la récursivité (trop grand nombre d'appels récursifs de la fonction, non-controlabilité de la fin de la boucle) mais dans, un certain nombre de cas, on peut utiliser la récursivité. (cf. exercices 14.1, 14.2 et 14.3)

### 3.2.1. Les booléens.

En matlab, le booleen vrai est égal à 1 ; le booleen faux est égal à zéro.

Les différents opérateurs logiques sont :

$==$  : égalité.

$\sim$  : différent.

$<$  : strictement inférieur.

$>$  : strictement supérieur.

$\leq$  : inférieur ou égal.

$\geq$  : supérieur ou égal.

$\&$  : ET logique (AND).

$|$  : OU logique (OR).

$\sim$  : NON logique (NOT).

$xor$  : OU exclusif (XOR).

La fonction `exist('dudu')` permet de connaître la nature de 'dudu' : Elle renvoie  
 0 si dudu n'est pas défini (ni fonction, ni variable),  
 1 si dudu est une variable de l'espace de travail,  
 2 si dudu est un M-fichier trouvé dans l'un des répertoire indiqués dans les chemins de recherche de matlab,  
 3 si dudu est un fichier mex trouvé dans l'un des répertoire indiqués dans les chemins de recherche de matlab,  
 4 si dudu est une fonction compilée de simulink,  
 5 si dudu est une fonction du noyau de matlab (built-in function),  
 6 si dudu est un fichier P trouvé dans l'un des répertoire indiqués dans les chemins de recherche de matlab,  
 7 si dudu est un répertoire.

### 3.2.2. Les entrées/sorties.

L'instruction `disp(a)` affiche le contenu de la matrix a, qu'elle soit à coefficient scalaires ou de type chaîne (cf. section 7).

Pour rentrer une variable, on utilise `input` : Si on tape

```
>> x= input('entrez x : ');
```

x peut être : un scalaire, une matrice, une expression algébrique, une expression logique.

### 3.2.3. Les instructions conditionnelles.

Elles sont :

`if` booleen

```

instruction
end
```

```

if booléen
    instruction1
else
    instruction2
end
```

```

if booléen1
    instruction1
elseif booléen2
    instruction2
end
```

```

if booléen1
    instruction1
elseif booléen2
    instruction2
else
    instruction3
end
```

Si une variable (scalaire ou chaîne) peut prendre un ensemble fini de valeurs, on utilisera *switch* variable

```

case valeur1
    instruction1
case valeur2
    instruction2
...
otherwise
    instruction
end
```

```

while booléen
    instruction
end
```

### 3.2.4. Les instructions inconditionnelles : les boucles.

Elles sont du type :

```

for variable=expression
    instruction
```

*end*

Par exemple, si A est une matrice, l'instruction

```
for v=A
    instruction
end
```

est équivalente à (où n est le nombre de colonnes de A)

```
for j = 1:n, v=A( :,j);
    instruction
end
```

On peut aussi écrire

```
for j = 1:5
    ou
for j = -10:-3:-45
    ou
for x = -2.0:0.25:-0.75
```

Toutes ces boucles peuvent être imbriquées.

### 3.2.5. Instructions diverses.

*break* permet de sortir de la boucle.

*return* permet de revenir au M-fichier ayant appelé le programme.

*error('message')* affiche le message, émet un bip et interrompt l'exécution du programme.

Regarder aussi les instructions *pause*, *keyboard*.

## 3.3. Quelques conseils

Un très grand nombre de choses sont déjà programmées en matlab et il serait inutiles de les reprogrammer ; bien entendu, il serait excessif de vouloir connaître tout matlab, mais avant de vous lancer dans la programmation, consultez bien les documentations et/ou l'aide.

Utilisez les boucles le moins possible. Souvent on peut faire des calculs sur les composantes d'une matrice sans avoir à accéder à ses éléments. De nombreux calculs en boucles peuvent être remplacés par une interprétation matricielle (cf. exercices 14.4 et 14.5 ).

Faites du beau travail : décomposez le plus possible votre algorithme de façon à en programmer des petits blocs, indépendants les uns des autres. Il faut le plus possible utiliser des fonctions. Dans l'idéal, si vous avez un programme à faire, il devrait comporter un seul et court M-fichier principal, où l'on définit les valeurs des variables utilisées, on récupère en sortie, les différentes choses calculées ; ce programme fait appel à un certain nombre de fonctions, chacune d'elle en appelant éventuellement d'autres.

Documentez vos programmes : n'hésitez pas à mettre des commentaires. De plus, les lignes commentées qui suivent immédiatement la première ligne de la déclaration d'une fonction apparaîtront dans l'aide. Cela est très précieux et vous permet d'avoir des renseignements sur les fonctions que vous avez définies sans en visualiser les sources, en utilisant *help*. De plus *lookfor* cherchera dans ces

lignes. Les lignes de commentaires séparés du commentaire par une ligne blanche seront ignorées. Prenez exemple sur le fichier angle.m de matlab (ne pas le taper, mais trouvez-le en cherchant le fichier angle.m dans C :\MATLABR11) :

```
function p = angle(h)
% ANGLE Polar angle.
%   ANGLE(H) returns the phases angles, in radians, of a matrix
%   with complex elements. Use ABS for the magnitude.
p=atan2(imag(h),real(h));
```

#### 4. Résolution de systèmes linéaires

Pour matlab, il existe deux divisions : / et \ (division droite et gauche) ; en théorie :

$$a/b = ab^{-1} \text{ et } a\backslash b = a^{-1}b.$$

Ainsi, pour des scalaires  $x$  et  $y$ ,

$$x/y = \frac{x}{y} \text{ et } y\backslash x = \frac{x}{y}.$$

Pour une matrice carrée  $a$ ,  $\text{inv}(a)$  calcule l'inverse de  $a$ . Ainsi, en théorie,

$$a/b = a \text{ inv}(b) \text{ et } a\backslash b = \text{inv}(a) b.$$

Cependant, les deux calculs n'utilisent pas les mêmes algorithmes (en effet, pour résoudre le système linéaire  $ax = b$ , il est plus rapide d'en calculer sa solution directement que d'abord calculer l'inverse de  $a$  et de le multiplier ensuite par  $b$ ) ; voir les exemples qui suivent.

Si  $B$  est une matrice carrée,  $X = A\backslash B$  est la solution  $A^{-1}B$  de l'équation matricielle  $AX = B$  où  $X$  est de la même taille que  $B$ . En particulier, si  $B = b$  est un vecteur colonne, alors  $x = A\backslash b$  est la solution  $A^{-1}b$  de l'équation matricielle  $Ax = b$ .

Faire les quatre exemples suivant :

```
>> A=[1 2 ; 3 4];
>> B=[5 6 ; 7 8 ];
>> C=B /A
>> D=A \ B
>> C*A
>> A*D
>> B *inv(A)
>> inv(A)*B

>> a=[1 2 3; 4 5 6; 5 7 10];
>> inv(a)

>> a=[1 3 5 ; 1 2 4; 0 5 1];
>> b=[22; 17; 13]
>> a \b
```

```

>> a*(a \b)

>> tic; flops(0); for k=1 :1000; x=inv(a)*b; end; t=toc; n=flops;
>> n
>> t
>> tic; flops(0); for k=1 :1000; x=a\b; end; t=toc; n=flops;
>> n
>> t

```

Pour les deux derniers exemples, *tic* lance un chronomètre et *toc* l'arrête et en lit la valeur. *flops(0)* remet à zero le compteur des opérations élémentaires (addition, soustraction, multiplication ... de scalaires) et *flops* en détermine le nombre. Ces quatre instructions permettent, ici, de comparer le nombre d'opérations élémentaires et le temps mis par matlab pour calculer  $\text{inv}(a)^*b$  et  $a\b$ . Qu'en conclure ?

Matlab possède un grand nombre de méthodes plus précises pour résoudre des systèmes linéaires : Nous en donnons un catalogue très succinct avec les fonctions associées :

- décomposition lu : *lu*
- gradient conjugué : *bicg*
- gradient conjugué avec préconditionnement : *pcg*
- factorisation de Cholesky : *chol*
- factorisation QR : *qr*

## 5. Valeurs et vecteurs propres

L'instruction suivante permet de calculer les valeurs et vecteurs propres de la matrice carrée A :

```
>> [X,D]=eig(A)
```

Ces valeurs et vecteurs peuvent être complexes (cf section 6). Il existe un grand nombre d'outils dédiés à l'analyse spectrale de matrices ; nous renvoyons à la bibliographie.

## 6. Les complexes

Matlab accepte les nombres complexes sous la forme  $a+ib$  ou  $\rho\exp(i\theta)$  (on peut aussi noter  $j$  à la place de  $i$ ).

Toutes les opérations algébriques sur les complexes sont reconnues ( $*, +, /$ ). On dispose aussi des fonctions *real*, *imag*, *abs*, *angle*, *conj*.

On peut éléver un complexe à une puissance réelle grâce à la fonction :  $^{\wedge}$ . On dispose aussi de *sqrt*, *log*, *exp*.

Toutes les opérations définies pour les matrices à coefficients réels demeurent valables pour les matrices à coefficients complexes.

**attention** : ne pas dans utiliser un compteur noté  $i$  dans un programme où l'on utilise des complexes. (dans ce cas, une expression du type  $3+5i$  ne sera pas reconnue ou cela produira de graves confusions). La variable complexe  $i$  est réinitialisée après chaque *clear*.

**Exercice 6.1.** Écrire une fonction qui calcule les  $n$  racines d'un nombre complexe.

## 7. Les chaînes et autres types de données

### 7.1. Les chaînes

une chaîne est un vecteur ligne dont le nombre de composantes est égal à la longueur de la chaîne.

Exemple :

```
>> ch='une chaîne'
>> size(ch)
>> length(ch)
```

Pour concaténer deux chaînes, on écrit celle-ci comme un vecteur ligne :

```
>> ch=[ch, ' c''est pratique ','et agréable'];
>> ch
>> disp(ch)
>> disp('ch')
```

Pour rentrer une chaîne, on tape

```
>> ch= input('entrez la chaîne : ', 's');
```

Les fonctions *num2str* et *int2str* transforment respectivement un réel et un entier en une chaîne (cf. exercice 14.6).

Il existe un certain nombre de fonctions sur les chaînes dont voici quelques exemples :

```
>> abs
>> setstr
>> isstr
>>
>> str2num
>> sprintf
>> upper
>> lower
>> eval
>> blanks
>> deblank
```

### 7.2. Les entiers

Il existe un certain nombre de fonctions spécifiques aux entiers :

```
>> mod(a,b) : renvoie le quotient de la division entière de a par b (cf. rem).
>> factor(a) : renvoie les facteurs premiers de a.
>> primes(a) : renvoie un vecteur ligne avec les nombres premiers plus petit que a.
>> isprime(a) : retourne 1 si a est un nombre premier.
>> nextpow2(a) : renvoie n tel que  $2^{n-1} \leq a < 2^n$ .
>> perms(c) : renvoie toutes les permutations possibles du vecteur c.
>> nchoosek(v,k) : v est un vecteur de longueur supérieure ou égale à k ; cette fonction renvoie toute les permutations possibles de k éléments parmi les composantes de v.
```

### 7.3. Les ensembles

Les vecteurs lignes de matlab peuvent être aussi considérés comme des ensembles, dont on donne quelques fonctions :

*intersect*  
*ismember*  
*setdiff*  
*setxor*  
*unique*  
*union*

## 8. Les polynômes

Le polynôme

$$P(X) = a_n X^n + a_{n-1} X^{n-1} + \dots + a_1 X + a_0,$$

est représenté en matlab par le vecteur ligne  $[a_n \ a_{n-1} \dots \ a_1 \ a_0]$  (de longueur  $n + 1$ ).

Outre les opérations sur les matrices qui s'appliquent donc aux polynômes, il existent un certain nombre de fonctions spécifiques aux polynômes.

$>> conv(A,B)$  multiplie les deux polynômes A et B.  
 $>> [Q,R]=deconv(A,B)$  renvoie les deux polynômes Q et R tels que  $A = BQ + R$  avec degré de R strictement inférieur à celui de B.  
 $>> roots(P)$  renvoie les racines du polynôme P (dans  $\mathbb{C}$ ). Réciproquement, *poly* calcule le polynôme à partir de ces racines.  
 $>> P=[1 -6 11 -6];$   
 $>> S=roots(P)$   
 $>> poly(S)$   
 $>> poly(S')$

$>> polyval(f,X)$  renvoie les valeurs de la fonction f sur la matrice X.  
 $>> polyder(f)$  dérive le polynôme f.  
 $>> [q,d]=polyder(a,b)$  calcule la dérivée de a/b (où a et b sont deux polynômes) sous la forme q/d.  
 $>> polyval(f,A)$  renvoie  $f(A) = p_1 A^n + p_2 A^{n-1} + \dots + p_n A + p_{n+1} I$ , où A est une matrice et f un polynôme.

**Exercice 8.1.** Faire une fonction qui enlève les zéros superflus des polynômes (par exemple, elle renverra  $[0 \ 1 \ 2]$  sur  $[1 \ 2]$ ).

**Exercice 8.2.** Faire une fonction qui permette de faire la somme de deux polynômes de degrés différents (ce qui n'est pas possible avec matlab, car on ne peut additionner deux vecteurs de tailles différentes).

**Exercice 8.3.** Calculer le reste et le quotient de la division de A par B, où

$$A = 3X^6 - 8X^5 + X^4 + 16X^3 - 19X^2 + 9X + 6,$$

$$B = 3X^2 + 4X - 1.$$

Que remarquez-vous ? Comment pourrez on y remédier ? Faites en une fonction et testez la sur les calculs de la division euclidienne de  $A$  par  $B$  où

$$\begin{aligned} A &= X^5 - X^4 + 6X^3 + 2X^2 - 6X - 9, \\ B &= X^3 - X^2 + 7X + 1, \end{aligned}$$

et

$$\begin{aligned} A &= X^5 - X^4 + 6X^3 + 2X^2 - 7X - 1, \\ B &= X^3 - X^2 + 7X + 1. \end{aligned}$$

Matlab permet aussi de déterminer la décomposition en éléments simples dans  $\mathbb{C}$ . Si  $A$  et  $B$  sont deux polynômes, et si  $A/B$  n'admet que des pôles simples, on peut calculer, avec matlab, les résidus  $r_k$ , les pôles  $p_k$  et le polynôme  $K$  tels que

$$\frac{A}{B} = K + \frac{r_1}{X - p_1} + \frac{r_2}{X - p_2} + \dots + \frac{r_n}{X - p_n} + K.$$

Pour cela on tape,  $>> [r,p,k] = residue(A,B)$ .

Les résidus  $r_k$  sont stockés dans le vecteur  $r$ , les pôles  $p_k$  sont stockés dans le vecteur  $p$  et  $K$  dans le vecteur  $k$ .

Si la fraction  $A/B$  admet pour le pôle  $p_j$  une multiplicité  $m$ , matlab remplacera la fraction  $r_j/(X-p_j)$  par

$$\frac{r_j}{X - p_j} + \frac{r_{j+1}}{(X - p_j)^2} + \dots + \frac{r_{j+m-1}}{(X - p_j)^m}.$$

Si on cherche la décomposition en éléments simples dans  $\mathbb{R}$ , on utilisera la décomposition dans  $\mathbb{C}$  et on regroupera les termes conjugués deux à deux.

**Exercice 8.4.** Calculer la décomposition dans  $\mathbb{C}$  de

$$\frac{17X^2 + 36X + 9}{2X^3 + 4X^2 + 2X + 4}.$$

et en déduire sa décomposition dans  $\mathbb{R}$ .

**Exercice 8.5.** Calculer la décomposition de

$$\frac{2X^6 - 32X^5 + 188X^4 - 516X^3 + 709X^2 - 479X^2 - 479X + 134}{X^5 - 8X^4 + 24X^3 - 34X^2 + 23X - 6}.$$

## 9. Les graphiques

### 9.1. Les graphes bi-dimensionnels

#### 9.1.1. Généralités.

`plot(x)` représente les points de coordonnées  $(j, x_j)$ .

`plot(x,y)` représente les points de coordonnées  $(x_j, y_j)$ .

```
>> x=0 :0.1 :pi;
>> plot(x,sin(x));
```

*plot(Z)* représente les points de coordonnées ( $\text{re}(Z), \text{im}(Z)$ ).

*plot(A)* où (A est une matrice) représente A en fonction des indices.

```
>> plot([0 0 0 0 0; 1 2 3 4 5; 2 4 6 8 10]);
>> plot([0 0 0 0 0; 1 2 3 4 5; 2 4 6 8 10']);
```

*plot(x,A)* où (A est une matrice) représente A en fonction de x.

```
>> x=[0 3 6];
>> A=[x;2*x;3*x;x.^2];
>> plot(x,A);
>> plot(A,x);
```

*plot(...,str)* permet de préciser aux courbes faites précédemment : str est une chaîne contenant un, deux ou trois arguments qui sont le type de point, le type de tracé et la couleur (cf. tableau 1, page 18).

Exemple :

```
>> x=0 :0.1 :pi;
>> plot(x,sin(x),'r- -o');
```

*plot(x1,y1,str1,x2,y2,str2,...,xn,yn,strn)* permet de tracer n courbes  $(x_i, y_i)$  avec les paramètres optionnels str1, ... , strn.

Exemple :

```
>> a=0;
>> b=3;
>> n=50; lancer exemtrace1 (fourni).
```

Grâce à *comat*, matlab permet aussi d'étudier les courbes paramétriques du type

$$\begin{cases} x = x(t), \\ y = y(t), \\ t \in [a, b]. \end{cases}$$

Lancer *exemtrace2* (fourni).

*fplot('fct',lim,str)* est très pratique : elle permet de tracer le graphe de la fonction fct (M-file ou fonction interne) dans la fenêtre définie par  $\text{lim}=[x_{\min}, x_{\max}]$  ou  $\text{lim}=[x_{\min}, x_{\max}, y_{\min}, y_{\max}]$ ; str un paramètre optionnel, une chaîne définie dans le tableau 1.

### 9.1.2. Autres types de représentation.

Soit à tracer la courbe définie en polaire par

$$r(\theta) = e^{\cos(\theta)} - 2 \cos(4\theta) + \left( \sin \frac{\theta}{12} \right)^5.$$

	type de point
.	point
*	étoile
square	carré
diamond	losange
pentagram	pentagone
hexagram	héxagone
none	rien
o	lettre o
+	signe +
x	croix
<	triangle tourné vers la droite
>	triangle tourné vers la gauche
^	triangle tourné vers le haut
v	triangle tourné vers le bas

  

	type de tracé
-	continu
--	trait-trait
-.	trait-point
:	pointillé
none	rien

  

	couleur
g	vert
m	magenta
b	bleu
c	cyan
w	blanc
r	rouge
k	noir
y	jaune

TABLE 1. Les différentes options de plot.

On fera

```
>> theta=linspace(0,22*pi,1100);
>> r=exp(cos(theta)-2*cos.....
>> polar(t,r);
```

### 9.1.3. Contrôle des graphiques.

*clf* efface la fenêtre courante.

*hold on* active le contrôle sur le graphique courant.

*hold off* désactive le contrôle sur le graphique courant.  
*subplot* permet de faire des sous-graphes.

Exemples : faire marcher *exemtrace3* qui permet de tracer, sur un seul graphique, avec quatre sous graphes, la fonction  $g(x) = -x \sin(x)$ , sa dérivée, sa dérivée approchée et l'erreur.

*axis*[ $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ ] ou *axis*(str) où str est une chaîne prenant l'une des valeurs 'manual', 'auto', 'equal', 'square'... qui spécifie les options : trace les axes.

*grid* : grille

*title,xlabel* ... légendes et textes de la figure.

**Exercice 9.1.** Faire un petit fichier de commande permettant de tracer une fonction fct (interne ou M-fichier) avec plusieurs paramètres : nombre de points, abscisse minimales ,....

## 9.2. Les graphes tri-dimensionnels

Consulter l'aide en ligne pour avoir la syntaxe de *plot3 mesh* et *contour*.

## 9.3. Les interfaces graphiques

Pour l'instant, cette partie sort du cadre du cours, mais sachez que matlab permet de faire, sans trop de difficulté, un certaines d'interfaces graphiques très conviviales, grâce auxquelles, on peut lancer des instructions matlab en contrôlant les paramètres de façon visuelle.

## 10. Les sons

Si la carte son est correctement installée, vous pouvez jouer de la musique avec matlab. Vous pouvez aussi créer des sons de façon analytique et les écouter ensuite. Regardez la démonstration qui contient quelques sons en lançant *xpsound*.

## 11. Analyse de fonctions

Représenter la fonction  $g$  de  $\mathbb{R}$  dans  $\mathbb{R}$  telle que

$$g(x) = \frac{5x - 6,4}{(x - 1,3)^2 + 0,002} + \frac{9x}{x^3 + 0,03} - \frac{x - 0,4}{(x - 0,92)^2 + 0,005},$$

dont on trouvera une expression dans le fichier *exozero.m*. En trouver tous les zéros en utilisant la fonction *x=fzer('exozero',alpha)* où alpha désigne la valeur initiale de recherche. Pour la première racine, calculer *x=fzer('exozero',alpha,tol)* où tol est l'erreur admise. On prendra tol dans l'ensemble  $\{10^{-6}, 10^{-10}, 10^{-15}, \text{eps}, 10^{-20}\}$ . Que remarquez-vous ?

Trouver le minimum de  $g$  grâce à la fonction *fmin(g,a,b)*.

## 12. L'utilisation du debugger

Avec matlab, il est possible de debugger agréablement ses scripts, en suivant l'évolution de l'exécution d'un script, pas à pas, ou en s'arrêtant à des endroits précis, tout en contrôlant ou modifiant les différentes variables mises en jeu.

Pour cela, une fois que l'on a ouvert les différents scripts avec l'éditeur de matlab, il faut placer, en déplaçant le curseur, des "points d'arrêt" là où l'on désirer voir le programme s'arrêter provisoirement.

On lance ensuite le programme de la fenêtre matlab (ou on appelle la fonction) ; il apparaît une invite "K>>". On contrôle ensuite l'exécution du programme, à partir de l'éditeur de plusieurs façons :

- on fait debug/continu pour aller d'un point d'arrêt à l'autre.

- on fait F10 pour avancer pas à pas.

- on fait F11 pour avancer pas à pas en entrant dans chacune des sous-procédures appelées . **Attention**, dans ce cas, on pourra éventuellement entrer dans des sources de fonctions déjà programmées de matlab, qu'il ne faut pas modifier.

Dans tous les cas, apparaît une petite flèche dans l'éditeur qui indique à quel endroit du programme on se trouve. Parallèlement, dans la fenêtre matlab, on peut contrôler les variables et même les modifier. Ces variables sont spécifiques à un espace de travail, indiqués dans le menu **stack**

Quand le programme a tourné ou est en train de tourner avec le debugger, on peut accéder aux valeurs des variables en déplaçant la souris sur chacune des variables dans l'éditeur matlab : sa valeur apparaît alors.

**Exercice 12.1.** Ecrire un script faisant appel à plusieurs sous fonctions et tester le debugger.

### 13. Calcul d'intégrales et résolution d'équations différentielles

Outre les méthodes d'intégration numériques déjà vues (rectangles et trapèzes), on peut aussi citer la règle de Simpson ; matlab l'utilise avec l'instruction *quad(f,a,b)*.

Calculer l'intégrale double

$$\int_0^1 \int_0^1 e^{-x^2-y^2} dx dy,$$

en utilisant *dblquad(f,a,b,c,d)*. **attention**, dans la définition de f, il faudra prendre garde à taper  $f(x,y)=\exp(-x.^2-y.^2)$ .

Matlab permet aussi d'intégrer des équations différentielles ordinaires du type :

$$\begin{cases} X'(t) = F(t, X(t)), \\ X(t_0) = X_0. \end{cases}$$

en utilisant *ode45*.

Par exemple, on intégrera numériquement le problème sur [01]

$$\begin{cases} x'(t) = -x^2(t), \\ x(0) = 1, \end{cases}$$

en utilisant la fonction xprim (fichier xprim.m fourni) et en tapant

```
>> [T,X]=ode45('xprim',[0,1],1);
>> plot(T,X);
```

On pourra aussi regarder les fonctiond *ode23*, *ode113*, *ode23t...* qui correspondent à d'autres méthodes d'intégration.

## 14. Quelques exercices

**Exercice 14.1.** Déterminez la fonction factorielle en utilisant sa définition récursive :

$$\forall n \in \mathbb{N}, \quad n! = \begin{cases} n \times (n-1)! & \text{si } n \geq 1, \\ 1 & \text{si } n = 0. \end{cases}$$

Calculer, avec cette fonction,  $0!$ ,  $10!$ ,  $510!$  (que se passe-t-il dans ce cas?). Calculer  $(1,5)!$ ,  $(0,5)!$ . Comment peut on palier le problème mis en évidence ?

Utiliser les fonctions de matlab *prod*, *factorial*, et *gamma* pour calculer  $n!$ . Comparer, en terme de temps et de nombres d'opérations. Qu'en conclure ?

**Exercice 14.2.** Déterminez la fonction de Mc Carthy en utilisant sa définition récursive :

$$\forall n \in \mathbb{N}, \quad f(n) = \begin{cases} n - 10 & \text{si } n > 100, \\ f(f(n + 11)) & \text{sinon.} \end{cases}$$

Calculer  $f(n)$ , pour  $0 \leq n \leq 120$ . Que constate-t-on ? Calculer, pour  $0 \leq n \leq 120$ , le nombre d'itérations nécessaires pour calculer  $f(n)$ . La fonction  $f$  est elle définie si  $n$  n'est pas entier ? Faire un graphique mettant en évidence ces résultats.

**Exercice 14.3.** Faire la même étude pour la fonction de Collatz définie par :

$$\forall n \in \mathbb{N}, \quad \begin{cases} \text{si } n \leq 1, & f(n) = 1, \\ \text{si } n \geq 2, & f(n) = \begin{cases} f(n/2) & \text{si } n \text{ est pair,} \\ f(3n + 1) & \text{si } n \text{ est impair.} \end{cases} \end{cases}$$

La preuve que le calcul de cette fonction se finit toujours est encore un problème ouvert !

**Exercice 14.4.** Écrire une fonction pour calculer les sommes

$$T_1(n) = \sum_{i=1}^n \log(i) \text{ et } T_2(n) = \sum_{i=1}^n (\log(i))^2.$$

Pour chacune des ces deux sommes, on essayera deux façons différentes :

- on utilisera dans un premier temps l'instruction *for* ;
- on utilisera dans un second temps l'instruction *sum* ;

On comparera, pour différentes valeurs de  $n$ , les deux méthodes, notamment en terme de nombres d'opérations élémentaires et de temps de calcul. Conclure.

**Exercice 14.5.** On donne trois M-fichiers (somme1.m, somme2.m et somme3.m dont les sources figurent ci dessous). Les analyser et les comparer en terme de nombres d'opérations élémentaires et de temps de calcul. Conclure.

```
function x=somme1(n,m)
for i = 1:n
    xtemp = 0;
    for j = 1:m
        xtemp = xtemp+log(i)*exp(-j^2);
    end
    x(i) = xtemp;
end
```

```

function x=somme2(n,m)
x=sum(((log(1:n)).'*ones(1,m)).*(ones(n,1)*exp(-(1:m).^2)))';

```

```

function x=somme3(n,m)
x=sum(exp(-(1:m).^2)).*log(1:n);

```

**Exercice 14.6.** Écrire une fonction qui transforme un polynôme (stocké sous forme de tableau) en une chaîne de caractère ; par exemple, le polynôme [3 8 -7 0 1] aura pour image la chaîne '3X^4 +8X^3-7X^2+X' .

## Références

- [MM97] Mohand Mokhtari et Abdelhalim Mesbah. *Apprendre à maîtriser matlab*. Springer, 1997. ouvrage de la bibliothèque de l'UTBM sous la cote QA 188 MOK.
- [PES99] Eva Pärt-Enander et Anders Sjöberg. *The matlab 5 handbook*. Addison-Wesley, 1999. ouvrage de la bibliothèque de l'UTBM sous la cote QA 188 PAR.
- [Ref] *Matlab Reference guide*.