

TRAVAUX PRATIQUES DE l'UE MFAppro

Informatique 3A

**MATHÉMATIQUES FONDAMENTALES POUR L'INFORMATIQUE
(APPRENTIS)**

2026-2027, Automne

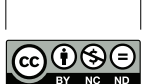
Jérôme Bastien

Document compilé le 5 septembre 2026

Le lien original de ce document est le suivant :

<http://utbmjb.chez-alice.fr/Polytech/MFIappro/TPMFIappro.pdf>

Ce document est mis à disposition selon les termes de la licence Creative Commons : Paternité - Pas d'Utilisation Commerciale - Pas de Modification ; 3.0



<http://creativecommons.org/licenses/by-nc-nd/3.0/>

ou en français

<http://creativecommons.org/licenses/by-nc-nd/3.0/deed.fr>

Liste des Travaux Pratiques

Avant-propos	iii
Travaux Pratiques 1. Connaissances de base de matlab	1
1.1. Lancement et instructions de base	1
1.2. Les notions de bases	2
1.3. Le répertoire courant et le chemin d'accès	3
1.4. La programmation avec matlab	4
1.5. Résolution de systèmes linéaires	7
1.6. Valeurs et vecteurs propres	8
1.7. Les complexes	8
1.8. Les polynômes	8
1.9. Les graphiques	9
1.10. L'utilisation du debugger	9
1.11. Matlab symbolique	9
1.12. Quelques exemples en matlab	11
1.13. Quelques exercices	16
Travaux Pratiques 2. TP spécifiques	19
2.1. Logique	19
2.2. Approximations numériques de π	19
2.3. Problème $3n + 1$	19
2.4. Tri de Crêpes	19
2.5. Changement de variables dans une intégrale	20
2.6. Énumération de parties	20
2.7. Problème des flux d'argent	20
2.8. Tours de Hanoï	20
Annexe A. Matlab/Octave à distance	23
A.1. Matlab à distance	23
A.2. Octave sur votre machine	23
Bibliographie	25

Avant-propos

Ce polycopié constitue les TP de Mathématiques Fondamentales pour l'Informatique (Apprentis) du département Informatique 3A (2026-2027, Automne).

Ce polycopié de TP et les fichiers matlab utilisés sont normalement disponibles à la fois

- en ligne sur <http://utbmjb.chez-alice.fr/Polytech/index.html> à la rubrique habituelle ;
- en cas de problème internet, sur le réseau de l'université Lyon I : il faut aller sur :
 - 'Poste de travail',
 - puis sur le répertoire 'P:' (appelé aussi '\\teraetu\Enseignants'),
 - puis 'jerome.bastien',
 - puis 'Polytech',
 - puis 'Informatique 3A'.
 - enfin sur 'MFIappro'.

Pour l'utilisation de Matlab/Octave à distance, on pourra consulter l'annexe A page 23.

Connaissances de base de matlab

Les sections 1.1, à 1.11 sont extraites¹ de [BM03]. Elles constituent le cœur de ce TP. Quelques sources matlab sont évoquées dans ces sections. Toutes ces sources sont disponibles sur <http://utbmjb.chez-alice.fr/Polytech/index.html>

La section 1.12 est un ensemble d'exemples², qui sera traité de façon facultative.

L'objet de ce TP est de rappeler de façon succincte au lecteur les notions élémentaires de matlab à partir de quelques exemples, utilisés dans certains TP.

Le lecteur ne connaissant pas matlab pourra consulter des guides spécialisés (par exemple [HLR01], [92], [PS99], [MM97] et [Mok00])

Nous indiquons à chaque fois les séquences à taper sous matlab.

1.1. Lancement et instructions de base

- **help** : très précieuse, cette aide en ligne vous permet de «tout savoir sur tout». Pour savoir comment l'utiliser, faites **help help**. Cette aide est dite en ligne, puisqu'on l'utilise directement dans la fenêtre de commande matlab. Vous pouvez aussi lancer l'aide de matlab, en cliquant **help matlab** dans le menu help de la fenêtre de commande matlab ; cette aide est plus conviviale, parfois mieux documentée. De surcroît, elle contient parfois des références pour comprendre le fonctionnement des algorithmes.
- **lookfor xxx** : affiche³ le nom des toutes les fonctions qui contiennent le mot xxx (en anglais !) dans la première ligne de commentaire. C'est *grosso modo* la fonction réciproque de **help**.

Nous soulignons l'importance des fonctions **help** (ou de la fenêtre d'aide) et **lookfor** ; l'utilisateur de matlab n'hésitera pas à les utiliser avant de programmer une fonction.

- **demo** : démonstration de matlab, très complète, dans laquelle on trouvera des exemples variés, couvrant l'ensemble des domaines d'utilisation.
- **intro** : lance une introduction à matlab.
- **cd** : permet de changer de répertoire.
- **pwd** : affiche le répertoire courant.
- **dir** ou **ls** : affiche la liste des fichiers du répertoire courant.
- **delete** : supprime un fichier.
- **...** : pour terminer une expression à la ligne suivante.
- **↑** : fait passer à la commande précédente et permet, par exemple, de la modifier sans la retaper.
- **↓** : fait passer à la commande suivante.

À partir de la fenêtre de matlab, on peut exécuter des commandes matlab :

- les unes après les autres, comme une «super calculatrice», qui ferait un nombre de choses incroyable,
- ou groupées sous forme de fichiers script ou de fonctions (cf. section 1.4.3).

1. Elles en constituent l'annexe B.

2. Tous extraits des sujets de TD.

3. Attention, cette fonction est très lente, puisqu'elle consulte l'ensemble des fichiers .m de matlab

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

1.2. Les notions de bases

1.2.1. 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 1.7), les vecteurs et les matrices.

1.2.2. Les opérations arithmétiques

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

1.2.3. Vecteurs ou tableaux à une dimension

Il y a plusieurs façons de saisir un vecteur, par exemple :

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

Ce vecteur est considéré comme une matrice à une ligne et trois colonnes. On pourra le vérifier en tapant :

```
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 vecteur en spécifiant son indice entre parenthèses :

```
v(2)
```

Pour obtenir un vecteur dont les composantes varient à pas constant entre deux valeurs connues, on utilisera

```
x=0:0.25:1
y=-pi:0.3:pi
z=0:10
```

Si on impose le nombre de valeurs, on utilisera

```
t=linspace(-pi, pi, 4)
```

En général, on privilégiera l'emploi de `:` plutôt que de `linspace`, plus lent d'emploi.

Les opérations sur les vecteurs à n composantes sont naturellement celles de l'espace vectoriel $(\mathcal{M}_{n,p}(\mathbb{R}), +, \cdot)$: on additionne les matrices composante par composante et on les multiplie par un scalaire en multipliant chacune des leurs composantes par ce scalaire.

En faisant précéder d'un point les opérateurs $*$, $/$, et $^$, on peut effectuer d'autres 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 calcule l'image de chaque composante.

Il existe un grand nombre de fonctions matlab opérant directement sur les vecteurs. Citons par exemple **sum** et **prod**.

Chaque fonction matlab que vous programmerez sera de préférence matricielle. Comme les fonctions matlab prédéfinies, elles doivent, si possible, opérer sur un tableau avec la règle suivante : l'image d'un tableau par une fonction est le tableau des images.

1.2.4. Vecteurs ou tableaux à deux dimensions

Saisie d'une matrice :

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

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
a*b
a^3
exp(a)
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 l'aide à la rubrique colon.

1.3. Le répertoire courant et le chemin d'accès

Le répertoire courant (current directory) est le répertoire où l'on travaille (cela est important si on utilise des fichiers stockés sur disque dur). Pour le modifier, cliquer sur les trois petits points de la fenêtre de commande (à côté de **current directory**) et choisir le répertoire souhaité.

Par ailleurs, il est possible de rajouter au chemin d'accès (path) un certain nombre de répertoires ou de sous-répertoires où matlab viendra chercher les fichiers *.m recherchés (pour les éditer, les faire fonctionner, consulter leur commentaire). Pour cela, cliquer **set path** dans le menu **file**, puis **add folder** ou **add folder with subfolder**. On aura intérêt à mettre en fin de path (avec l'instruction **move bottom**) les répertoires rajoutés de

façon que matlab consulte tout d'abord ses propres répertoires avant les vôtres. Dans le path, figurent en effet par défaut les différents sous-répertoires de matlab. Grâce au path, si vous utilisez **edit**, **help**, **lookfor** ou si vous lancez une exécution, matlab ira consulter l'ensemble des fichiers des répertoires présents dans le path, même si vous travaillez dans un autre répertoire courant.

1.4. La programmation avec matlab

1.4.1. 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 pour toute matrice carrée renvoie son déterminant.

On peut définir soi-même ses propres fonctions. On veut définir la fonction de $\mathbb{R}^3$ dans $\mathbb{R}^2$ qui à toute matrice 1×3 , $a=(x, y, z)$ associe la matrice 1×2 , $b=(u, v)$ 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 choisit **file/new**) ou on tape **edit** dans la fenêtre de commande ; 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 enregistre ce fichier avec le nom dudu (l'extension est .m par défaut). On tape dans la fenêtre de commande matlab :

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

Il est indispensable que le nom d'une fonction matlab soit identique au nom du fichier dans laquelle elle est stockée.

EXERCICE 1.1. Soit la fonction de $\mathbb{R}$ dans $\mathbb{R}$, définie par morceaux par :

$$f(x) = \begin{cases} 0 & \text{si } x < 0, \\ 1 & \text{si } 0 \leq x \leq 2, \\ 2 & \text{si } 2 < x < 18, \\ 3 & \text{si } x \geq 18. \end{cases}$$

Écrire cette fonction sous matlab de façon matricielle, c'est-à-dire, de telle sorte que l'image d'un tableau soit le tableau des images. On pourra consulter la fonction fournie **escalier**.

Pour comprendre le fonctionnement vectoriel de cette fonction, dont la ligne principale est

```
res=((0<=x)&(x<=2))+((2<x)&(x<18))*2+(x>=18)*3;
```

il faudra consulter la correction de l'exercice de TD 8.2 pour comprendre que la ligne ci-dessus ne fait que traduire la propriété (8.1) vue dans cet exercice qui donne ici :

$$\forall x \in \mathbb{R}, \quad f(x) = 0 \times \chi_{]-\infty, 0[} + 1 \times \chi_{[0, 2]}(x) + 2\chi_{]2, 18[}(x) + 3\chi_{[18, +\infty[}(x)$$

Une autre possibilité permet de définir une fonction sans passer par un fichier (on réservera cette solution à une fonction simple et utilisée un nombre restreint de fois) en utilisant la fonction **inline** très pratique, dont voici un exemple d'utilisation :

```
g=inline('x.^2+sin(x)');
```

ou de façon équivalente (cette dernière solution est d'ailleurs l'unique retenue dans les dernières versions de matlab)

```
g=@(x) x.^2+sin(x);
```

Pour calculer $g(2)$, il suffira de taper, comme pour les fonctions usuelles :

```
g(2)
```

Cette fonction est en mémoire vive (le vérifier en tapant **whos**) et le demeure tant qu'on ne fait pas **clear** et qu'on reste dans la même session de matlab. On pourra consulter l'aide à propos de cette fonction.

On créera ses propres fonctions, chacune d'elle étant enregistrée dans un fichier xxx.m où xxx est le nom de cette fonction ou déclarée en inline. Si on en définit plusieurs, on testera chacune d'elle indépendamment des autres.

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'entrée ne sont pas modifiées au cours de l'exécution de la fonction.

1.4.2. La programmation

Aux M-fichiers de fonctions se rajoutent les M-fichiers de programmes, dits aussi script; plutôt que de taper les instructions les unes après les autres dans la fenêtre matlab, il peut être plus agréable de les mettre toutes 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-fichier (sans l'extension .m).

Bien entendu, matlab autorise le 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-contrôlabilité de la fin de la boucle).

1.4.2.1. Les entrées/sorties

L'instruction **disp(a)** affiche le contenu de la matrice a, qu'elle soit à coefficients scalaires ou de type chaîne.

Pour saisir une variable, on utilise **input** : si on tape

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

la variable x peut être : un scalaire, une matrice, une expression algébrique, une expression logique ou symbolique (voir section 1.11).

1.4.2.2. Les instructions conditionnelles

Citons les instructions **if**, **end**, **else**, **switch**, **case**, **while**.

1.4.2.3. Les instructions inconditionnelles : les boucles

Citons les instructions **for**, **end**.

1.4.3. Différence entre fonction et script

Rappelons les quelques points suivants :

- *Fonction* Une fonction possède des arguments d'entrée, et pas nécessairement de sortie. Elle commence systématiquement par la ligne de commande :

```
function [x,y,z,...]=nom_fonction(a,b,c,...)
```

où $x, y, z, \dots$ sont d'éventuelles variables de sortie et $a, b, c, \dots$ sont les variables d'entrée.

Toutes les fonctions de calculs matlab fonctionnent ainsi ; voir par exemple **det**, **eig**, **diff** ou **int** (en symbolique, voir section 1.11). On pourra aussi consulter les exemples fournis **somme1** ou **escalier**.

Le formalisme adopté dans les TP, pour présenter les fonctions sera le suivant :

$[x, y, z, \dots] = \text{nom_fonction}(a, b, c, \dots)$

- on décrira les variables d'entrée $a, b, c, \dots$: leur nom, leur nombre, leur type
- on décrira les éventuelles variables de sortie $x, y, z, \dots$: leur nom, leur nombre, leur type

On utilisera la fonction décrite ci-dessus en écrivant dans la fenêtre de commande matlab :

$[x, y, z] = \text{nom_fonction}(a, b, c)$

- *Script* Un script est un fichier de commandes sans variables d'entrée ni de sortie. En revanche, dans un script, l'utilisateur peut saisir des variables en utilisant par exemple la fonction **input**.

Le formalisme adopté dans les TP, pour présenter succinctement les scripts sera le suivant :

nom_script

- descriptif des variables saisies par l'utilisateur ;
- descriptif des actions exécutées par ce script.

On pourra aussi consulter la commande **demo** de matlab, qui est un script, ou des exemples fournis : **demo_exemple_un** ou **demo_exemple_formel1**.

On utilisera le script décrit ci-dessus en écrivant dans la fenêtre de commande matlab :

nom_script

- *Stockage des scripts et des fonctions* Les fonctions et les scripts constituent tous les deux des fichiers *.m. La fonction **nom_fonction** doit être stockée dans un fichier nommé **nom_fonction.m**.

Pour les TP, la plupart des fichiers à écrire sont des fonctions. Quelques scripts seront demandés : nous les nommerons alors **demo_***.

1.4.4. Quelques conseils

Un très grand nombre de choses sont déjà programmées sous matlab et il serait inutile 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, plus efficace sous matlab.

Faites du beau travail : décomposez le plus possible votre algorithme de façon à en programmer des petits blocs, indépendants les uns des autres. Dans l'idéal, si vous avez un programme à écrire, il devrait comporter un seul et court M-fichier principal, où l'on définit les valeurs des variables utilisées ; on récupère en sortie, les différents objets calculés ; 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 écrire 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 (uniquement dans la première, qui doit donc contenir en quelques mots un descriptif sommaire mais précis de la fonction). Les lignes de commentaires séparées du commentaire par une ligne blanche seront ignorées par les fonctions **help** et **lookfor** et permettront à l'utilisateur de commenter son programme.

1.5. Résolution de systèmes linéaires

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

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

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 , **inv**(a) calcule l'inverse de a . Ainsi :

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

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 . On étudiera les exemples proposés ci-dessous.

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 $Ax = b$.

Traiter les exemples suivants :

```
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 2 3 ; 2 4 6 ; 1 0 2];
inv(a)

for j=[0,10,100,1000]
    disp(inv([1 2 3 ; 2 4 6+eps*j ; 1 0 2]));
end

a=[1 3 5 ; 1 2 4 ; 0 5 1];
b=[22; 17; 13]
a \b
a*(a \b)

tic ; for k=1:100000; x=inv(a)*b ; end ; t=toc ;
disp(t);

tic ; for k=1:100000 ; x=a \b ; end ; t=toc ;
disp(t);
```

Pour les deux derniers exemples, **tic** lance un chronomètre et **toc** l'arrête et en lit la valeur. Ces deux commandes permettent, ici, de comparer le temps mis par matlab pour calculer $\text{inv}(a)*b$ et $a \setminus b$. Qu'en conclure ?

1.6. Valeurs et vecteurs propres

La commande 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 1.7). Il existe un grand nombre d'outils dédiés à l'analyse spectrale de matrices ; nous renvoyons à la bibliographie.

1.7. 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 élever un complexe à une puissance réelle grâce à la fonction : `^`. On dispose aussi de **sqrt**, **log**, **exp**.

Taper par exemple

`z=(2+3*i)*(5-6*i);`

`theta=angle(z)`

`r=abs(z)`

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

Attention, ne pas utiliser un compteur noté `i` ou `j` dans un programme où l'on utilise des complexes.

1.8. 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 existe 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.
- **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.
- si A est une matrice et f un polynôme, **polyval**(f,A) renvoie un tableau constitué de l'image de chacun des élément de A.

Traiter l'exemple suivant :

```
P=[1 -6 11 -6];
S=roots(P)
poly(S)
```

1.9. Les graphiques

1.9.1. Les graphiques bi-dimensionnels

Rappelons que (et de façon non exhaustive!) :

- **plot**(x) représente les points de coordonnées (j, x_j) , pour $1 \leq j \leq p$, où p est la longueur du vecteur x .
- **plot**(x,y) représente les points de coordonnées (x_j, y_j) , pour $1 \leq j \leq p$.
- **plot**(Z) représente les points de coordonnées $(\text{re}(Z), \text{im}(Z))$, si Z est un vecteur complexe.
- **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.

Par exemple, taper les commandes suivantes :

```
x=linspace(-pi/2, pi/2, 1000);
plot(x, cos(x), x, sin(x));
```

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.

On pourra aussi utiliser **ezplot**.

1.9.2. Les graphiques tri-dimensionnels

On étudiera par exemple les fonctions **plot3**, **mesh** ou **surf**.

1.9.3. D'immenses possibilités

Les possibilités de matlab sont immenses et méritent d'être approfondies ...

1.10. L'utilisation du debugger

Avec matlab, il est possible de debugger agréablement ses sources, en suivant l'évolution de l'exécution d'une fonction ou 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 édité (avec la fonction **edit**) les différentes fonctions et sources, il faut placer, en déplaçant le curseur, des "points d'arrêt" là où l'on désire voir le programme s'arrêter provisoirement.

On lance ensuite le programme à contrôler depuis la fenêtre de command matlab; il apparaît une invite "K". On contrôle ensuite l'exécution du programme dans la fenêtre d'édition de plusieurs façons, en appuyant sur les touches F5, F10 ou F11 (voir dans le menu **debug** leur signification).

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.

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.

1.11. Matlab symbolique

Nous parlons aussi de matlab formel.

Nous ne donnons ici que quelques exemples très simples. Il convient de consulter l'aide ou les démos pour avoir un regard beaucoup plus complet sur la partie symbolique de matlab, très riche.

On pourra par exemple consulter l'aide pour étudier les fonctions **int**, **diff** et **limit**. Traiter les exemples suivants :

```
syms x a b;
y=x^2+a*x+b;
disp(y);
disp(diff(y,x));
disp(diff(y,a));

ou

syms x h;
limit((sin(x+h)-sin(x))/h,h,0)

ou

int(sin(x)/x,0,inf)
```

On pourra aussi étudier les fonctions **sym**, **syms**, **solve**, **expand**, **simplify**, **simple**, **subs**, **subexp**, et **pretty**.

Traiter encore les deux exemples suivants qui déterminent de façon symbolique les maximums des fonctions $x \mapsto (a-x)(x-b)$ et $x \mapsto (h-x)x(x+h)$ définies respectivement sur $[a,b]$ et $[-h,h]$, avec tracé des fonctions. Voir les deux scripts **demo_exemple_formel1** et **demo_exemple_formel2** :

% exemple de calcul formel avec trace.

```
syms a b x;
f=(a-x)*(x-b);
fp=diff(f,x);
d=solve(fp);
pretty(simplify(subs(f,x,d)));
eval([ 'f1=@(x,a,b)' , vectorize(char(f)) , ';' ]);

disp( 'pour_voir_le_graphe , appuyez_sur_une_touche ' );
pause;
X=-1:2/500:1;
Y=f1(X,-1,1);
plot(X,Y);
```

% exemple de calcul formel avec trace.

```
syms h x;
f=(h-x)*x*(x+h);
fp=diff(f,x);
d=solve(expand(fp));
q=simplify(subs(f,'x',d));
pretty(q);
eval([ 'f1=@(x,h)' , vectorize(char(f)) , ';' ]);

disp(f);
```



```

disp(f1);

disp('pour_voir_le_graphe, appuyez_sur_une_touche');
pause;
X=-1:2/500:1;
Y=f1(X,1);
plot(X,Y);

```

On pourra aussi utiliser la fonction `matlabFunction` disponible dans la version 2019 de matlab.

1.12. Quelques exemples en matlab

Dans cette section, nous donnons quelques exemples en matlab : les lignes présentées dans ce TP pourront être tapées et exécutées les unes à la suite des autres, ou mieux, entrées dans script, qui est exécuté ensuite.

Quelques exemples seront issus de TD de cette UV.

1.12.1. Dérivées

On renvoie à la section 1.11.

1.12.2. Développements limités

Voir l'aide la fonction `laylor`.

Taper par exemple

```

syms x;
taylor(exp(-x))

taylor(log(1+x),6,0)

taylor(sin(x)/x,6,0)

```

On pourra traiter des exemples issus des TD.

Voir aussi la fonction fournie `developpement_limite.m`.

1.12.3. Intégrales

Grâce au calcul symbolique, on peut calculer des intégrales simples comme :

$$I = \int_0^{\pi/2} \cos^2(x) dx, \quad J = \int_0^1 \sqrt{\frac{ax+1}{x+3}} dx, \quad (a \text{ appartient à } \mathbb{R}_+^*)$$

en tapant

```

syms x a;
I=int((cos(x)^2),0,pi/2);
disp(I);
J=int(sqrt((a*x+1)/(x+3)),x,0,1);
disp(J);

```

Pour mieux visualiser la valeur de J , on peut aussi remplacer la dernière ligne par

```
pretty(J);
```

On peut aussi calculer des intégrales impropres comme

$$K = \int_0^{\infty} e^{-x^2} dx,$$

en tapant

```
syms x
K=int(exp(-x^2),0,inf);
```

Comme les mathématiciens, matlab ne sera pas toujours déterminer les primitives des fonctions, comme le montrent

```
syms x
disp(int(exp(-x^2),1,inf));
disp(int(exp(-x^3.5),1,2));
```

REMARQUE 1.2. On notera la différence des résultats produits par les deux lignes précédentes ; la première d'entre elles renvoie un résultat exprimé à partir de la fonction matlab **erf**. Cette fonction est une fonction qui est connue par les mathématiciens, qu'on peut utiliser sous matlab, en symbolique et en numérique, mais dont on ne dispose pas d'expression explicite exprimées avec des fonctions usuelles⁴. Si on tape dans la fenêtre de commande matlab :

help erf

on apprend la définition de cette fonction :

$$\forall x \in \mathbb{R}, \quad \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

La seconde ligne de l'exemple donné ci-dessus indique en revanche que l'intégrale demandée

$$\int_1^2 e^{-x^{3.5}} dx,$$

n'est pas connue, même à l'aide de fonctions auxiliaire comme erf.

1.12.4. Matrices

On veut résoudre les systèmes suivants

$$\text{a) } \begin{cases} 2x + y = 3, \\ 2x + 4y = 6. \end{cases} \quad \text{b) } \begin{cases} x + y + z = 6, \\ 2x - 3y + 4z = 8, \\ x + y - z = 0. \end{cases}$$

Sous matlab, on écrira

```
A=[2 1;2 4];
B=[3;6];
X1=A\B;
C=[1 1 1 ;2 -3 4;1 1 -1];
D=([6 8 0])';
X2=C\D;
```

On peut aussi calculer des inverses de matrices :

```
inv([1 1 1 ;2 -3 4;1 1 -1]),
```

On peut aussi le faire en symbolique :

4. exactement comme la fonction logarithme par exemple.

```
inv(sym([1 1 1 ;2 -3 4;1 1 -1])),
inv(sym([1 1 1 ;2 -3 4;1 1 -pi])),
```

On peut calculer des déterminants de matrices : on veut déterminer l'inversibilité des matrices :

$$\text{a) } A = \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}, \quad \text{b) } A = \begin{pmatrix} 1 & 2 & 3 \\ 1 & -1 & 1 \\ 2 & 2 & 3 \end{pmatrix}.$$

Il suffit de taper

```
det([1 2;3 1]),
det([1 2 3;
     1 -1 1;
     2 2 3])
```

On pourra aussi utiliser la fonction `resoud_system` et reprendre les exemples 4.12 page 41, 4.13 page 41 et 4.14 page 41 du cours. Taper

```
A1=...
[1 3 4 ;
5 -2 1;
3 4 2];
b1=[19 4 17].';
A2=...
[1 2 3;
1 1 2;
2 3 5];
b2=[1 2 3].';
A3=A2;
b3=[1 2 4].';
[x1,d1,V01,V1,verifm1]=resoud_system(A1,b1,[],1);
[x2,d2,V02,V2,verifm2]=resoud_system(A2,b2,[],1);
[x3,d3,V03,V3,verifm3]=resoud_system(A3,b3,[],1);
```

On doit obtenir la solution dans le premier cas, une matrice `V2` non vide dans le second cas et une matrice `x3` vide dans le troisième cas.

1.12.5. Équations différentielles

Pour résoudre l'équation différentielle

$$\begin{aligned} y'(t) + 2y(t) &= \cos(2t), \\ y(0) &= 2, \end{aligned}$$

on écrit

```
dsolve('Dy+2*y=cos(2*t)', 'y(0)=2')
```

Soit à résoudre les deux équations différentielles :

$$\begin{aligned} 2y'' + 5y' - 3y &= 0, \\ 2y'' + 5y' - 3y &= t^3 + t^2 - 1. \end{aligned}$$

On écrit :

```
dsolve('2*D2y+5*Dy-3*y=0'),
dsolve('2*D2y+5*Dy-3*y=t^3+t^2-1'),
```

On peut aussi rajouter des conditions aux limites : par exemple, pour traiter la dernière équation différentielle avec

$$y(0) = 1, \quad y'(0) = -8,$$

on écrira

```
dsolve('2*D2y+5*Dy-3*y=t^3+t^2-1','y(0)=1','Dy(0)=-8'),
```

On peut aussi écrire pour que l'affichage soit plus beau :

```
pretty(dsolve('2*D2y+5*Dy-3*y=t^3+t^2-1','y(0)=1','Dy(0)=-8')) ,
```

Traitons un autre exemple. Dans un premier temps, on résoud l'équation différentielle :

$$v''(x) + \omega_0^2 v(x) = K \sin(\omega x),$$

en tapant sous matlab

```
pretty(dsolve('D2y+omega0^2*y=K*sin(omega*x)','x')) ,
```

On prend ensuite en compte les conditions initiales

$$v(0) = v(L) = 0,$$

en tapant sous matlab :

```
pretty(dsolve('D2y+omega0^2*y=K*sin(omega*x)','y(0)=0','y(L)=0','x')) ,
```

On pourra ensuite résoudre l'équation différentielle avec $\omega_0 = \omega$:

$$v''(x) + \omega^2 v(x) = K \sin(\omega x), \quad (1.1)$$

avec les conditions aux limites

$$v(0) = v(L) = 0. \quad (1.2)$$

On pourra aussi s'intéresser à la solution de l'équation différentielle (6.77)-(6.78) avec $L = \pi/\omega$ où K est quelconque puis quand $L = \pi/\omega$ et $K = 0$.

1.12.6. Diagonalisation de matrices

Dans le cours, on a diagonalisé la matrice

$$A = \begin{pmatrix} 2 & 0 & 1 \\ 1 & 1 & 1 \\ -2 & 0 & -1 \end{pmatrix}.$$

Il suffit de taper

```
A=[2 0 1;1 1 1;-2 0 1];
disp(A);
pause;
[Q,D]=eigs(A);
disp(diag(D));
pause;
disp(Q);
```

On peut aussi déterminer les valeurs propres et le polynôme caractéristique de façon symbolique. On pourra consulter l'exemple suivant :

```
A=[2 0 1;1 1 1;-2 0 1];
sp=eig(A);

sps=eig(sym(A));
disp('valeurs_propres_(numeriques)');
disp(sp);
disp('valeurs_propres_(symboliques)');
pretty(sps);
pause;

disp('plus_grand_module_des_parties_imaginaires_(numerique)');
disp(max(abs(imag(sp))));
disp('plus_grand_module_des_parties_imaginaires_(symbolique)');
disp(max(abs(eval(imag(sps)))));
pause;

P=poly(A);
Ps=poly(sym(A));
disp('polynome_caracteristique');
disp(P);
disp('polynome_caracteristique_(symbolique)');
disp(Ps);
```

Il existe une autre façon de calculer le polynôme caractéristique de A (elle est moins jolie et moins concise sous matlab, mais elle montre l'utilisation des fonction `eye` et `collect`) :

```
A=[2 0 1;1 1 1;-2 0 1];
syms x;
Pss=det(sym(A)-x*eye(3));
disp(Pss);
disp(collect(Pss));
```

On peut montrer que la matrice suivante n'est pas diagonalisable :

$$A = \begin{pmatrix} 8 & 12 & 10 \\ -9 & -22 & -22 \\ 9 & 18 & 17 \end{pmatrix}.$$

On s'en convaincra sous matlab, en tapant, une fois que A est rentrée :

```
[Q,R]=eig(A);
[Qs,Rs]=eig(sym(A));
disp(Q);
disp(Qs);
```

1.12.7. Ensemble et parties

En cours de rédaction.

Voir les fonctions **intersect**, **union**, **setxor**, **setdiff**, **ismember** et **unique**.

1.13. Quelques exercices

EXERCICE 1.3.

On pose

$$f(t) = \cos(2t)t^4,$$

$$F(x) = \int_0^x f(t)dt,$$

$$G(x) = \frac{\sin(x)}{x}.$$

- (1) Tracer sur l'intervalle $[-\pi, \pi]$ les fonctions f , F , G et G' sur quatre graphiques différents.
- (2) Rajouter sur le graphe de g sa tangente et son développement limité à l'ordre 2, les deux étant calculés en 1/2.

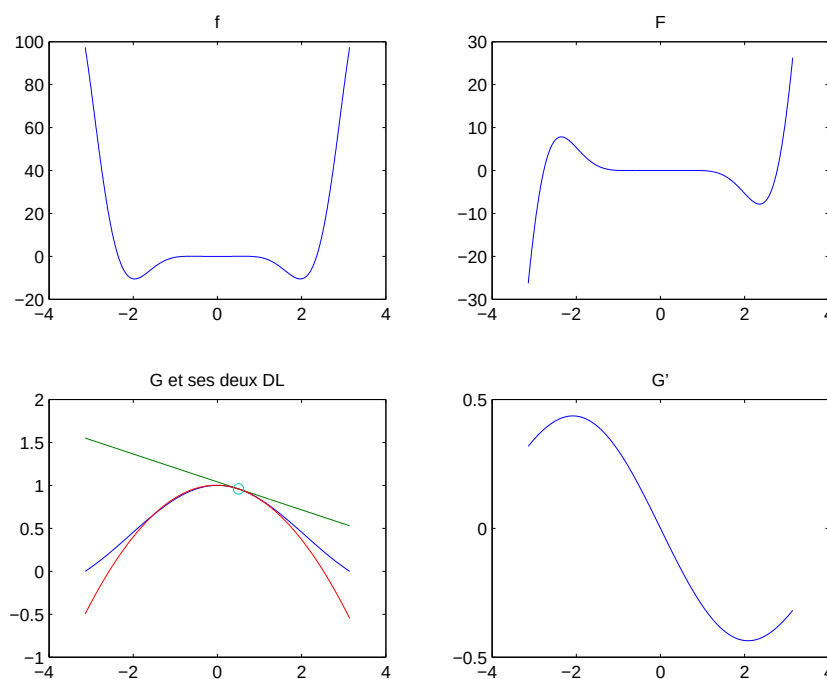


FIGURE 1.1. Les fonctions f , F , G et G' et les deux développements limités de G .

On consultera le script du corrigé : `corrige_exercice_application01.m`. Le graphe obtenu est donné en figure 1.1.

EXERCICE 1.4.

- (1) Reprendre, grâce à matlab symbolique, l'exercice 1.14 de TD.

On essaiera, si possible, de programmer les diverses méthodes proposées dans la correction page 4 des corrigés de TD.

On consultera le script du corrigé : `corrige_exercice_application02.m`.

(2) Reprendre, de même, l'exercice 1.15 de TD.

On consultera le script du corrigé : `corrige_exercice_application02b.m`.

TRAVAUX PRATIQUES 2

TP spécifiques

Dans ces TP, d'autres langages que Matlab pourront être utilisés.

2.1. Logique

2.1.1. Table de vérité

- (1) Écrire une fonction qui crée une table de vérité pour deux propositions P et Q .
- (2) Retrouver les tables de vérité (dans les notes de cours) 10.1 page 85, 10.2, 10.4 et 10.5.

2.1.2. Lois booléennes exhaustives

Programmer les diverses lois des exercices de TD 8.6 et 8.7. En particulier, tenter de retrouver les tableaux 8.6, 8.7, 8.9 et 8.12 des corrigés de TD.

On pourra consulter les fonctions `lois_booléennes.m`, `lois_booléennes_exhaustives.m` et `lois_exhaustives_booléennes_deux.m`, disponibles à l'adresse habituelle.

2.2. Approximations numériques de π

Voir l'annexe N page 231 du cours et essayer de programmer les différentes suites proposées approchant π .

2.3. Problème $3n + 1$

Voir la section 16.1 page 139 du cours.

- (1) Écrire une fonction qui permette de calculer les valeurs et le temps de vol de la suite de Syracuse de $N \in \mathbb{N}^*$. Obtenir par exemple la figure 16.1 page 140 du cours.
- (2) Faire varier ensuite N entre 1 et une valeur maximale. Obtenir par exemple la figure 16.2 page 141 du cours.

REMARQUE 2.1. Ceux qui sont à l'aise en Python pourront essayer de comprendre comment a été obtenue la figure 16.3 page 142 du cours.

On pourra consulter la fonction `troisnplusun.m` disponible à l'adresse habituelle.

2.4. Tri de Crêpes

Voir la section 16.2 page 143 du cours.

Écrire une fonction fondée sur l'algorithme 16.2 page 144 du cours et retrouver par exemples les résultats de la section du cours 16.2 du cours.

On pourra consulter la fonction `triclepes.m` disponible à l'adresse habituelle.

2.5. Changement de variables dans une intégrale

Voir les remarques 3.3 page 27 et 3.5 page 30 et l'annexe F du cours.

Écrire une fonction qui permette de calculer α , β et F en fonction de a , b et f .

Retrouver par exemple les résultats de l'annexe F du cours.

On pourra consulter les fonction `changement_variable_integrale.m` et `demo_changement_variable_integrale.m` disponibles à l'adresse habituelle.

2.6. Énumération de parties

Voir la section 9.4.2 page 84 du cours.

Écrire une fonction qui permette d'énumérer les parties de l'ensemble $\{1, \dots, n\}$ avec les différentes méthodes vues.

Retrouver par exemple les résultats obtenus dans les exercices de TD 8.1 et 8.3.

On pourra consulter la fonction `parties.m` et `demoparties.m`, disponibles sur le site habituel.

2.7. Problème des flux d'argent

Reprendre, en le programmant, l'exercice de TD 16.1 et en utilisant les fichiers de données évoqués dans la remarque 16.1 du TD.

- (1) On pourra utiliser le fichier de données http://utbmjb.chez-alice.fr/Polytech/MFIappro/donnees/gite_2019.txt (qui contient un choix arbitraire de créanciers indiqués dans la dernière colonne) et la fonction `dettes_gite.m` et taper la commande suivante pour obtenir les dates et les nom des créanciers, déterminés avec la méthode de la question 1 de la correction de l'exercice 16 de TD :

```
[dettes, creancier, nomcreancier, I, J] = dettes_gite('gite_2019.txt');
```

- (2) On pourra utiliser le fichier de données http://utbmjb.chez-alice.fr/Polytech/MFIappro/donnees/gite_2019_brut.txt et la fonction `dettes_gite_bis.m` et taper la commande suivante pour obtenir les dates et les nom des créanciers, déterminés avec la méthode de la question 2 de la correction de l'exercice 16 de TD :

```
[E, S, nomcreancierb, nomdebiteurb] = dettes_gite_bis('gite_2019_brut.txt');
```

2.8. Tours de Hanoï

On consultera l'exercice de TD 11.17.

La correction est en partie extraite de https://fr.wikipedia.org/wiki/Tours_de_Hano%C3%AF

On pourra aussi consulter <https://fr.mathworks.com/matlabcentral/answers/53247-tower-of-hanoi-problem>

- (1) Montrer l'algorithme récursif 2.1.
- (2) En déduire l'écriture de la fonction matlab `tour_hanoi`, disponible sur le site habituel.
- (3) Vérifier que si l'on tape

```
tour_hanoi(4,1,2,3)
```

on obtient

```
Déplacer le disque supérieur de la tour 1 vers la tour 3.
Déplacer le disque supérieur de la tour 1 vers la tour 2.
Déplacer le disque supérieur de la tour 3 vers la tour 2.
Déplacer le disque supérieur de la tour 1 vers la tour 3.
```

Algorithme 2.1 Algorithme de déplacement des tours de Hanoï *tour-hanoi*(n, D, A, I) : affiche les déplacements à effectuer

Entrée :

n : nombre de disques utilisés.
 D : emplacement de départ.
 A : emplacement d'arrivée.
 I : emplacement intermédiaire.

Sortie :

$\mathcal{D}$: les déplacements à faire.

si $n \neq 0$ **alors**

tour-hanoi(n, D, A, I)
affiche : "Déplacer le disque de D vers A "
tour-hanoi(n, I, A, D)

fin si

Déplacer le disque supérieur de la tour 2 vers la tour 1.
Déplacer le disque supérieur de la tour 2 vers la tour 3.
Déplacer le disque supérieur de la tour 1 vers la tour 3.
Déplacer le disque supérieur de la tour 1 vers la tour 2.
Déplacer le disque supérieur de la tour 3 vers la tour 2.
Déplacer le disque supérieur de la tour 3 vers la tour 1.
Déplacer le disque supérieur de la tour 2 vers la tour 1.
Déplacer le disque supérieur de la tour 3 vers la tour 2.
Déplacer le disque supérieur de la tour 1 vers la tour 3.
Déplacer le disque supérieur de la tour 1 vers la tour 2.
Déplacer le disque supérieur de la tour 3 vers la tour 2.

Matlab/Octave à distance

Vous avez deux les possibilités suivantes pour utiliser Matlab (section A.1) et son clone, libre et gratuit, Octave (section A.2).

A.1. Matlab à distance

Utilisez une machine virtuelle en consultant :

<https://etu.univ-lyon1.fr/outils/acces-distant-aux-fichiers-et-aux-applications-pedagogiques>

Il faut donc faire (pour windows, pour les autres systèmes d'exploitation, voir l'url donnée ci-dessous)

- Ouvrez le menu Démarrer -> Tous les programmes -> Accessoires -> Connexion bureau à distance (ou parfois Accessoires -> Communication -> ...);
- La boîte de dialogue "Connexion bureau à distance" apparaît;
- Tapez `tseetu.univ-lyon1.fr` dans le champ "Ordinateur", puis cliquez sur le bouton "Connexion".

Attention, cette solution a des inconvénients :

- Le réseau de la fac est trop aléatoire! On peut avoir un bon débit puis dans l'heure, il devient catastrophique. De plus, pour qu'un TP ait officiellement lieu avec cette solution, une réservation de salle virtuelle doit être faite. Donc, sauf dans le cas où cette réservation est faite et annoncée, cette solution est dédiée aux utilisations individuelles.
- Vous aurez, accès *via* une machine virtuelle à votre disque réseau (commençant par U:) et il faudra gérer vos fichiers et répertoires sur ce disque et pointer sur ce disque depuis Matlab.

D'autres logiciels utilisés à Lyon I sont disponible sur cette machine virtuelle (comme Maple).

A.2. Octave sur votre machine

(1) Installer Octave. Voir <https://www.gnu.org/software/octave/download>

(2) Installer le symbolique d'Octave

(a) Voir par exemple

<https://sites.google.com/site/lm3tpoptimisation/guide-octaveinstall-config>, qui présente une installation sans Python (d'autres installations utilisant des bibliothèque de Python sont possibles).

(b) Regarder l'exemple pour le "Symbolic package" et suivre pas-pas l'installation.

(c) N'oubliez pas, à chaque utilisation de la partie sybmbolique d'Octave, de taper

```
pkg load symbolic
```

Attention, la première ou les première fois il affiche `Symbolic pkg v2.7.1:` et puis, il faut attendre un peu ...

(d) Faites le test final suivant : tapez (et interprétez!)

```
syms x
int((cos(x))^2)
```

Quelques liens (certains sont contextuels et peuvent changer selon la version d'Octave).

<https://octave.org/doc/v5.2.0/>

<https://octave.org/octave.pdf>

https://octave.sourceforge.io/list_functions.php?sort=alphabetic

Bibliographie

- [92] *Matlab, Reference guide*. The Math Work, Inc., 1992.
- [BM03] J. BASTIEN et J.-N. MARTIN. *Introduction à l'analyse numérique. Applications sous Matlab*. Ouvrage disponible à la bibliothèque Sciences de Lyon 1 (cote : 519.4 BAS, 4^e étage). Voir <https://www.dunod.com/sciences-techniques/introduction-analyse-numerique-applications-sous-matlab>. Paris : Dunod, 2003. 392 pages.
- [HLR01] B. R. HUNT, R. L. LIPSMAN et J. M. ROSENBERG. *A guide to matlab, for beginners and experienced users*. Cambridge University Press, 2001.
- [MM97] M. MOKHTARI et A. MESBAH. *Apprendre à maîtriser matlab*. Springer-Verlag, 1997.
- [Mok00] M. MOKHTARI. *Matlab 5.2 & 5.3 et Simulink 2 & 3 pour étudiants et ingénieurs*. Springer-Verlag, 2000.
- [PS99] E. PÄRT-ENANDER et A. SJÖBERG. *The matlab 5 handbook*. Addison-Wesley, 1999.