SY06 - Travaux Pratiques

TP0 : introduction à Matlab

Les travaux pratiques de SY06 s'appuient sur les problèmes traités durant les séances de travaux dirigés. Il s'agit essentiellement de réaliser des simulations numériques mettant en évidence les rôles et influences des paramètres majeurs intervenant dans ces problèmes.

Le programme des travaux pratiques est le suivant :
- TP0 : introduction à Matlab,
- TP1 : modulation d'amplitude (problème n ° 1 du chapitre 2 du polycopié de cours),
- TP2 : surveillance vibratoire d'un roulement à billes (problème n ° 2 du chapitre 2 du polycopié de cours),
- TP3 : analyse temps-fréquence par spectrogramme (problème n ° 3 du chapitre 2 du polycopié de cours)
- TP4 : multiplexage en phase (problème du chapitre 3 du polycopié de cours),
- TP5 : directivité d'une antenne formée de plusieurs capteurs (problème du chapitre 7 du polycopié de cours).

Les travaux pratiques sont réalisés en monôme ou en binôme. Un compte-rendu par groupe est à rendre dans un délai de 2 semaines pour les TP 1 à 4. Le compte-rendu du TP 5 sera à rendre dans un délai de 1 semaine. Le compte-rendu est à rendre par voie électronique (http ://moodle.utt.fr) au format pdf (les compte-rendus rédigés *à la main* peuvent être scannés). Le compte-rendu type (de 10 pages maximum) inclut :
- un **nom de fichier explicite** et **le ou les noms des membres du groupe**,
- la description du problème et la modélisation qui en a été faite,
- des figures et les commentaires associés.

L'ensemble des scripts et fonctions Matlab exploités sera systématiquement joint au compte-rendu.

La notation tiendra compte de la réalisation (les codes, un minimum commentés), des résultats mais aussi des commentaires et explications les complétant. N'hésitez donc pas à justifier de façon claire et concise les résultats que vous obtenez.

Objectif de ce TP0

L'objectif de ce TP0 est de se familiariser avec l'environnement Matlab. A l'issu de celui-ci vous devez pouvoir :

- créer et manipuler scalaires, vecteurs et matrices,
- différencier et mettre en oeuvre des scripts et des fonctions,
- être en mesure de réaliser des "graphiques",
- avoir quelques éléments de programmation.

Introduction

Matlab, logiciel commercial développé par la société *The MathWorks*, est destiné au **calcul numérique** (Matlab permet, dans une certaine mesure, de réaliser des calculs formels mais ce n'est pas à cet

fin que nous l'utiliserons ici). A la fois langage de programmation **interprété** et environnement de développement, Matlab est particulièrement adapté aux manipulations de matrices et l'affichage des courbes (...) il est utilisé pour le traitement du signal, l'analyse statistique, le traitement d'images, la simulation de systèmes, l'optimisation et la modélisation et simulation de systèmes dynamiques ... Matlab est exploité dans un contexte aussi bien industriel que de recherche et dans la très grande majorité des secteurs d'activité.

La communauté d'utilisateurs Matlab est importante et active : de très nombreuses ressources existent sur le net (... mais aussi au SCD) :

— http ://www.mathworks.fr/products/matlab/
— http ://www.mathworks.fr/fr/help/matlab/getting-started-with-matlab.html

Environnement et aide en ligne

L'environnement de Matlab est composé d'une console (*command window*) et d'un éditeur (cf. figure 1) ainsi que d'une multitude de fenêtres périphériques non essentielles (*command history*, *workspace*, *current folder*, ...). Quelle que soit la manière utilisée pour saisir les instructions Matlab (au *prompt* de la console, *via* un script ou une fonction) c'est dans la console que s'affichent les résultats et les erreurs éventuelles (avec une explication de celles-ci).

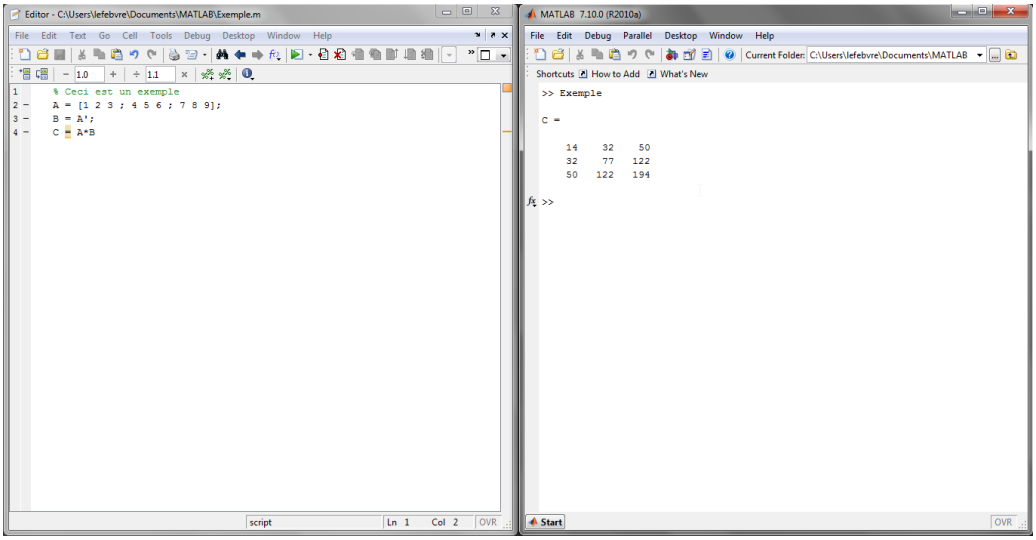


FIGURE 1 – Matlab, environnement (console et éditeur)

Le tableau suivant liste quelques fonctions "système", permettant notamment d'afficher l'éditeur de scripts et de fonctions ainsi que de localiser ou modifier le répertoire de travail courant de Matlab.

Fonctions "système"	
`exit`	quitte Matlab
`edit`	éditeur Matlab
`pwd`	répertoire courant
`dir`	fichiers du répertoire courant
`cd`	changement de répertoire
`clc`	efface la console
`help`	aide en ligne

La commande `help` ***`fonction`*** permet d'obtenir de l'aide sur la commande ***`fonction`***. Sous Matlab cette aide est donnée directement dans la console. L'aide peut être également affichée dans un envi-

ronnement de navigation dédié avec la commande `doc` *`fonction`*. Dans tous les cas, si *`fonction`* est omis, c'est un accès à la documentation complète qui est proposé.

Matrice, vecteur, scalaire, *constantes*

Un des grands avantages de Matlab est de ne pas nécessiter de déclarations préalables des variables. Par défaut, tous les *objets* manipulés sont des matrices ou vecteurs :

- une matrice 1×1 est un scalaire,
- une matrice $1 \times n$ est un vecteur ligne de dimension n,
- une matrice $n \times 1$ est vecteur colonne de dimension n.

La commande `A = [1 2 3 4 ; 5 6 7 8 ; 9 10 11 12 ; 13 14 15 16]` créé une matrice 4×4 et l'affecte à la variable `A`. Le "`;`", ou le retour chariot, sépare les différentes lignes de la matrice. En pratique, on peut utiliser n'importe quel nom pour les variables tant que ce nom ne commence pas par un chiffre et ne contient pas de caractères spéciaux.

```
>> A =
 1   2   3   4
 5   6   7   8
 9   10  11  12
 13  14  15  16
```

La commande `B=[1,2,3,4]` crée un vecteur ligne et l'affecte à la variable `B`. La "`,`" ou l'espace séparent deux éléments sur une ligne.

```
>> B =
 1  2  3  4
```

La commande `C=[1;2;3;4]` crée un vecteur colonne et l'affecte à la variable `C`.

```
>> C =
 1
 2
 3
 4
```

La commande `D=pi/6` affecte le résultat de l'opération $\pi/6$ à la variable `D`.

```
>> D =
 0.5235988
```

Le résultat des commandes suivies de "`;`" n'est pas affiché sur la console. Le contenu d'une variable est affichée par la saisie de son nom sur la console (en omettant le "`;`"). Toutes les variables définies par l'utilisateur dans la console sont stockées dans l'espace de travail (le *workspace*).

Exercice 1 :

1. Saisir la commande `A = [1:4 ; 5:8 ; 9:12 ; 13:16];`
   Quel est le rôle de l'instruction "`:`" ?
2. Saisir les commandes `v = 1:0.5:9;` et `u = [0:pi/12:2*pi];`
   Affiner le rôle de l'instruction "`:`".
3. On a `B=[1 2 3];` et `C=[4;5;6];`
   Quel est le résultat de la commande `D = [C [B ; B ; B]]` ?
   Quel est le résultat de la commande `E = [[4 5 6 7] [B ; B ; B]]` ? Pourquoi ?

La composante $A(i, j)$ de la matrice A est accessible par la commande `A(i,j)` où `i` et `j` sont des entiers. Les composantes de la matrice A sont ainsi modifiables.

Exercice 2 : On a `A = [1:4 ; 5:8 ; 9:12 ; 13:16];`

1. Saisir les commandes `A(:,2)`, `A([2 4],:)` et `A(:, 3)`.
   Quel est le rôle de l'instruction " : " ?
2. Saisir les commandes `A(6)`, `A(2)`, `A(end)`, `A(end-1)`
   Quel est le rôle de l'instruction `end` ?

Le tableau suivant énumère quelques fonctions permettant la génération de matrices ou de vecteurs.

Génération de matrices	
`eye(n)`	matrice *identité* de dimensions $(n \times n)$
`ones(n,m)`	matrice de 1 de dimensions $(n \times m)$
`zeros(n,m)`	matrice de 0 de dimensions $(n \times m)$
`rand(n,m)`	matrice dont les composantes suivent une loi uniforme $[0, 1]$
`linspace(x1,x2,n)`	vecteur de n valeurs équidistantes de $x1$ à $x2$
`logspace(x1,x2,n)`	vecteur de n valeurs logarithmiquement équidistantes de $x1$ à $x2$

Comme on a pu le voir précédemment un certain nombre de constantes sont prédéfinies `pi`, `eps`, `1i`... ainsi que différentes valeurs spécifiques (`inf`, `NaN`, ...). On prendra garde à ne pas redéfinir ces constantes par mégarde.

```
>> pi = 3
 3
```

On prendra également soin de réserver, si nécessaire, les caractères `i` et/ou `j` pour l'écriture des nombres complexes. Même si, en pratique, on pourra prendre les notations `1i` et/ou `1j`.

```
>> sqrt(-1)
 0 + 1.0000i
```

Exercice 3 :

1. Anticiper l'effet des commandes suivantes : `a = 1, b = 2, c = a+j*b, abs(c), angle(c), j = 1, c = a+j*b, abs(c), angle(c)`

Le tableau suivant cite quelques fonctions élémentaires sur la gestion de l'espace de travail associé à la console.

Génération de matrices	
`clear`	suppression des variables [1] du *workspace*
`clear A`	suppression de la variable A du *workspace*
`whos`	liste les variables du *workspace*
`size(A)`	dimensions de l'objet "matrice" A

Opérations matricielles

Le tableau suivant énumère quelques-unes des opérations élémentaires réalisables sur les objets matriciels.

Opérations matricielles élémentaires		
`+`	`.+`	addition (terme à terme)
`-`	`.-`	soustraction (terme à terme)
`*`	`.*`	produit (terme à terme)
`/`	`./`	division (terme à terme)
`^`	`.^`	puissance (terme à terme)
`'`		transpose

Remarque : attention aux opérations "`/`" (division) et "\" qui constituent de faux amis : on pourra évaluer les expressions `5/2` et `2/5`).

Fonctions de base

Les tableaux suivants listent quelques-unes des fonctions Matlab *natives* cette liste n'est absolument pas exhaustive ! On peut différencier les fonctions qui opèrent généralement sur des scalaires mais qui peuvent opérer terme à terme sur des vecteurs ou des matrices :

Fonctions *scalaire*	
`cos`, `acos`	cosinus, arc cosinus
`sin`, `asin`	sinus, arc sin
`tan`, `atan`	tangente, arc tangente
`log`, `log10`	logarithme naturel, logarithme décimal
`exp`	exponentiel
`abs`	valeur absolue, module
`angle`	argument (phase)
`sqrt`	racine carré
`round`	arrondi à l'entier le plus proche

Les fonctions qui opèrent généralement sur des vecteurs mais qui opèrent ligne-à-ligne ou colonne-à-colonne sur des matrices :

Fonctions *vecteur*	
`min`, `max`	minimum, maximum
`sum`	somme
`prod`	produit
`mean`	moyenne arithmétique
`std`	écart-type

Les fonctions qui opèrent sur des matrices.

Fonctions *matrice*	
`diag`	diagonale de la matrice
`trace`	trace de la matrice
`norm`	norme de la matrice
`det`	déterminant de la matrice
`cond`	conditionnement de la matrice
`inv`	inverse de la matrice
`rank`	rang de la matrice

Sauvegarde et chargement de données

Les données d'entrée et de sortie des sessions Matlab peuvent être sauvegardées et rechargées dans différents type de fichiers. Les plus simples d'utilisation sont les fichiers binaires. La commande `save` permet ainsi de sauvegarder tout ou partie des variables présentes en mémoire (l'extension du fichier de destination est classiquement ".mat" sous Matlab). La commande `load` permet de recharger les variables précédemment sauvegardées par la commande `save`.
L'autre format couramment utilisé est le format *ascii* pour sa généricité. Les écritures et lectures peuvent être réalisées avec les commandes `save` et `load` avec l'option `'-ascii'`.

Exercice 4 :

1. Expérimenter les instructions suivantes :

```
A = rand(4,3);
B = rand(3,4);
save('Stockage_A.txt','A','-ascii');
whos
clear A;
whos
C = load('Stockage_A.txt','-ascii');
save mesdonnees; clear;
whos
load mesdonnees;
whos
D = load('mesdonnees');
whos
```

Script et fonctions

Il est possible, pour ne pas avoir à saisir au prompt de la console l'ensemble des instructions constituant un programme, d'écrire ces instruction dans un fichier qui sera ensuite directement appelé depuis la console. Ce fichier, d'extension `.m`, est appelé *script* Matlab et est exécuté directement en saisissant son nom sur la console (pourvu qu'il soit dans le répertoire de travail de Matlab où qu'il soit dans un répertoire référencé (cf. commande `path`).

Remarque : Les scripts permettent de capitaliser toutes les saisies et modifications réalisées sous la console : soyez fainéants !

Une *fonction* Matlab est un fichier regroupant une séquence d'instructions pouvant être exécutée depuis Matlab, au même titre qu'un *script*, mais disposant d'**argument(s)** d'entrée et/ou de sortie. L'extension du fichier est également en `.m`.

Remarque : scripts et fonctions peuvent être commentées en faisant précéder chaque ligne de commentaire du caractère "`%`" ... C'est tellement plus pratique lorsque l'on reprend son travail la veille de rendre son compte-rendu ...

Scripts et fonctions peuvent mutuellement s'entre-appeler (le schéma classique étant celui d'un script faisant appel à des fonctions qui font elles-mêmes éventuellement appel à d'autres fonctions). Une fonction peut contenir des sous-fonctions mais cette démarche est, sauf cas particulier, source de confusion et est à proscrire. Par ailleurs, les fonctions ont leur propre "espace mémoire" : les variables définies à l'intérieur d'une fonction ne sont pas connues depuis le *workspace* ou l'espace mémoire de la fonction appelante : les variables d'une fonction sont dites **locales**. Inversement, sous Matlab, une fonction ne peut, sauf déclarations particulières, pas accéder aux variables du *workspace* ou de la fonction appelante si elles n'ont pas été passée en tant qu'arguments de cette fonction.

Scripts et fonctions sont des fichiers au format *texte* modifiables depuis n'importe quel éditeur de texte.

Graphiques

La fonction `plot` permet de tracer des graphiques de fonctions en deux dimensions. Concrètement la fonction `plot(x,y)` représente les points `(x(i),y(i))` en les reliant par des segments de droites. Selon

les arguments passés à la fonction `plot` les points et les segments sont ou non affichés. La technique de représentation peut se décomposer de la façon suivante :

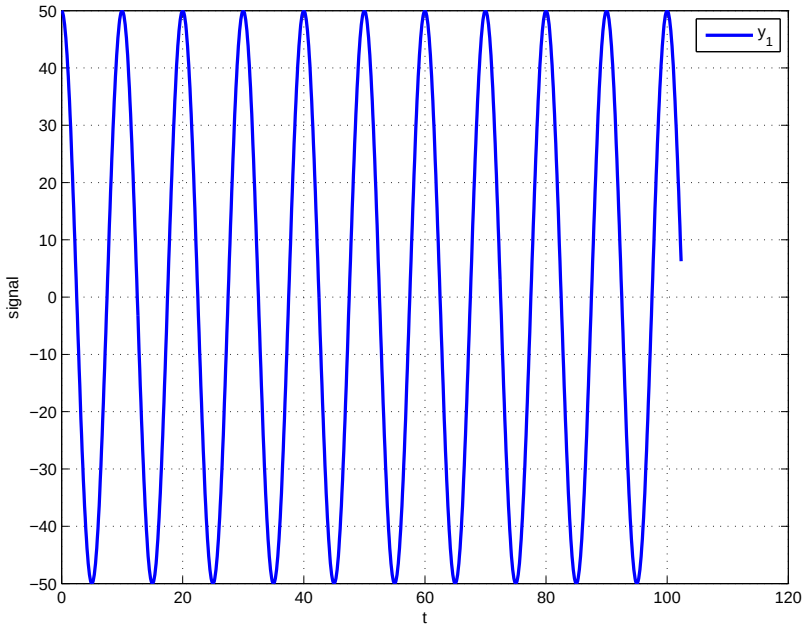
- discrétisation du domaine de représentation,
- évaluation de la fonction en chaque point de ce domaine discrétisé,
- exécution de l'instruction graphique avec l'ensemble des données obtenues.

Un exemple est donnée ci-après en 2D.

exemplefigure.m

```
Fe = 10;
N = 1024;
t = (0:1/Fe:(N-1)/Fe)';
f1 = 0.1;
k1 = 50;
y1 = k1.*sin(2*pi*f1*t);
figure;
plot(t,y1,'b');
xlabel('t');
ylabel('signal');
grid on;
legend('y_1');
```

```
>> exemplefigure;
```



Le tableau suivant cite quelques fonctions graphiques élémentaires.

Fonctions "graphiques"	
`plot`	graphique en 2D
`plot3d`	graphique en 3D
`contour`	lignes de niveaux d'un graphique en 3D
`mesh`	maillage d'un graphique en 3D
`xlabel`	étiquette de l'axe x
`ylabel`	étiquette de l'axe y
`zlabel`	étiquette de l'axe t
`title`	titre du graphique

`grid on/grid off`
Sous Scilab, `set(gca(),"grid",[1 1])`
`set(gca(),"grid",[-1 -1])`
	quadrillage

`subplot`	composition en sous-graphiques

`hold on/hold off`
Sous Scilab, `set(gca(),"auto_clear","on")`
`set(gca(),"auto_clear","off")`
	surcharge d'un graphique

`close`	fermeture du graphique courant

Exercice 5 : En vous appuyant sur le script `exemplefigure.m` illustrant la fonction :

$$y_1(t) = k_1 \sin(2\pi f_1 t)$$

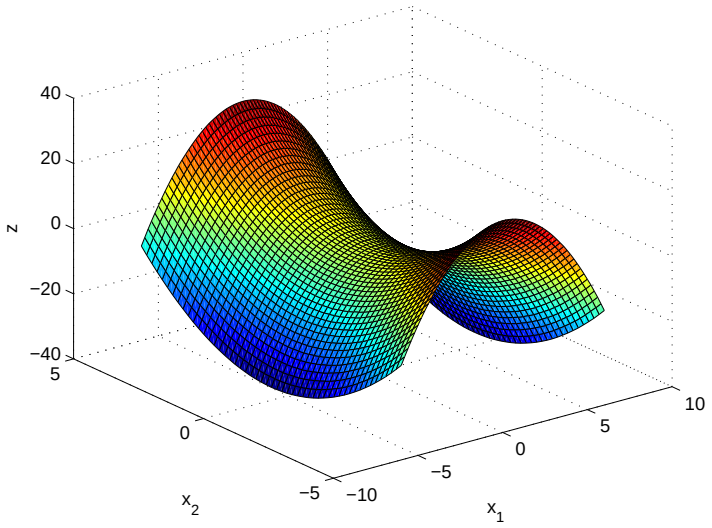
1. tracer sur le même graphique que $y_1(t)$ la fonction $y_2(t) = t \cdot cos\left(2\pi \frac{f_1}{2} t\right)$

Le script suivant fournit un exemple de graphique en 3D.

exemplefigure3d.m

```
[mX1,mX2] = meshgrid(-6:0.2:6,-5:0.2:5);
vX1 = mX1(:);
vX2 = mX2(:);
mXp = [vX1.^2 vX2.^2 vX1 vX2];;
mXp = [mXp ones(size(vX1,1),size(vX1,2))];
F = [-1 1 0 0 0]';
vZ = mXp*F;
mZ = reshape(vZ,size(mX1,1),size(mX1,2));
figure;
surf(mX1,mX2,mZ);
xlabel('x_1');
ylabel('x_2');
zlabel('z');
grid on;
```

```
>> exemplefigure3d;
```



Exercice 6 :

1. Créer un signal x_1 d'une durée de 7 secondes à une fréquence d'échantillonnage de 8 kHz contenant :
   — une sinus de fréquence 440 Hz sur une durée de 2 secondes,
   — 0.5 seconde de signal nul,
   — une sinus de fréquence 1760 Hz sur une durée de 2 secondes,
   — 0.5 seconde de signal nul,
   — une sinus de fréquence 880 Hz sur une durée de 2 secondes.
2. si l'ordinateur est équipé d'une carte son, le signal x_1 peut être écouté à l'aide de la commande `sound`. soit `x1` le signal généré, saisissez les commandes suivantes :
   `[vB,vA] = butter(4,1320/(8000/2));`
   `x2 = filter(vB,vA,vX);`
3. tracer les signaux x_1 et x_2 sur le même graphique (le signal x_2 peut également être écouté).
4. proposer une interprétation au bloc de commande permettant d'obtenir x_2.

Éléments de programmation

Comme tous les langages de programmation, Matlab dispose d'instructions de branchements conditionnels et des structures de boucle. Parmi les instructions de branchement conditionnel, l'instruction `if ... else ... end` évalue une expression logique et exécute un groupe d'instructions si l'expression logique est vraie.

```
if length(x) ~= 1
   disp('x n''est pas un scalaire.');
else
   if x>=0
      disp('x est un scalaire positif.');
   else
      disp('x est pas un scalaire négatif.');
   end
end
```

Parmi les structures de boucle on trouvera la boucle `for ... end` qui évaluera un groupe d'instructions un certain nombre de fois et la boucle `while ... end` qui évalue un groupe d'instruction tant qu'une expression logique est vraie.

```
A = zeros(5,4);
for indLigne = 1:size(A,1)
   for indColonne = 1:1:4
      A(indLigne,indColonne) = indLigne+indColonne/10;
   end
end
```

Le tableau suivant inventorie quelques commandes complémentaire pour la gestion des branchements conditionnels et des structures de boucle.

Fonctions de "branchement" et de boucles	
`elseif`	"sinon si" dans une embranchement `if`
`switch` (sous Scilab `select`) `... case ... case ... otherwise ... end`	embranchement multiple
`break`	interruption d'une boucle
`return`	retour à la fonction appelante

Vectorisation

Afin d'optimiser le temps d'exécution des programmes Matlab il convient de chercher à exploiter au mieux les fonctions prédéfinies qui sont nativement *vectorisées* c'est-à-dire qu'elles sont optimisées pour s'appliquer directement à une matrice, un ensemble de vecteurs ou un vecteur.
Afin de mettre en évidence cet intérêt on propose dans l'exercice suivant de réaliser deux programmes fonctionnellement identiques mais une fois *via* une programme classique tel qu'on le ferait en *C* par exemple et une autre fois en exploitant les fonctions "natives" de Matlab.

Exercice 7 : Soit `A = rand(n,n);` et `B = rand(n,n);` avec $n = 500$.

1. En exploitant les commandes `tic` et `toc` et en vous inspirant de l'exemple précédent illustrant l'instruction `for`, déterminer le temps nécessaire pour l'exécution d'un script affectant à chaque élément de C : $C(i,j)$ le produit $A(i,j)B(i,j)$.
2. Proposer et mettez en oeuvre une structure de programme avec deux boucles imbriquées réalisant le même travail.
3. Comparer, éventuellement à l'aide d'un graphique, les temps d'exécutions de ces deux programmes pour différents n (5, 50, 500, 5000).
4. Proposer une seconde structure de programme ne mettant en jeu qu'une unique bouche `for ... end`.

Matlab pour la rédaction de mon compte-rendu ?

Les commandes suivantes permettent de publier au format pdf ou au format html le script Matlab intitulé `nomdufichier` :

— `publish('nomdufichier','pdf');`
— `publish('nomdufichier','html');`

Cette fonctionnalité peut s'avérer pratique pour directement réaliser les compte-rendus ... à vous de voir si vous adoptez ou non cette solution.

Exercice 8 :

— Publier aux formats pdf ou html le script `VerifChap1Exo1.m`.
— Comment pourriez vous améliorer le travail déjà réalisé.