

SCILAB/MATLAB : premiers pas

14 septembre 2009

1 Matlab/Scilab

1.1 Présentation générale

MATLAB et SCILAB sont à la fois des langages de programmation et des environnements de développement pour le calcul numérique. MATLAB est développé et commercialisé par la société américaine The MathWorks. MATLAB est utilisé dans les domaines de l'éducation, de la recherche et de l'industrie pour le calcul numérique, mais aussi dans les phases de développement de projets. SCILAB a été développé à l'INRIA et il est accessible gratuitement sous forme exécutable et source sur le serveur [http ://www.scilab.org/](http://www.scilab.org/)¹. La version actuelle tourne sur les machines Unix, PC-Linux et Windows ainsi que sur les Mac.

Ces deux langages peuvent être utilisés dans un mode interactif, où les instructions sont exécutées immédiatement après qu'elles aient été tapées. Cependant, la plupart du temps, on s'en sert en écrivant des programmes ou des fonctions (avec un éditeur de texte quelconque), que l'on exécute ensuite dans une fenêtre MATLAB ou SCILAB (rappelons que pour exécuter un programme donné, il faut s'être rendu, à l'aide des instructions Linux, dans le répertoire où il se trouve). Enfin, il existe des interfaces utilisateurs sophistiquées pour des parties spécifiques.

1.2 Différences MATLAB/SCILAB

Les deux langages sont très similaires : leur fonctionnement interactif est très semblable et les fonctions ont en général la même syntaxe dans les deux langages. Cependant, quelques différences de syntaxe font que la compatibilité n'est pas totale (par exemple, la fonction `eig` de MATLAB, donnant le spectre d'une matrice, s'appelle `spec` en SCILAB). Par ailleurs, MATLAB étant beaucoup plus utilisé dans l'industrie, un certain nombre de Toolboxes (des "boîtes à outils", ensembles de fonctions associées à des champs d'applications précis, que l'on peut acheter ou télécharger pour augmenter les possibilités du langage) ont été développés en MATLAB sans avoir toujours d'équivalents en SCILAB.

¹Il est recommandé de faire attention aux versions dites "alpha", qui ne sont pas encore stabilisées. Pour Mac, on pourra par exemple préférer la version disponible ici :

[http ://www.hds.utc.fr/mottelet/Darwin/](http://www.hds.utc.fr/mottelet/Darwin/)

1.3 Sur ce tutoriel

De nombreux tutoriels pour chacun de ces langages se trouvent sur le web. On trouvera par exemple un certain nombre de tutoriels Scilab sur la page

http://www.cmapx.polytechnique.fr/~benaych/aleatoire_index.html

Ce tutoriel est destiné à l'utilisation de Scilab, même si, comme on l'a dit plus haut, les différences avec Matlab sont minimales, la question des Toolboxes mise à part.

2 Mode interactif

Pour lancer Scilab taper **scilab**. Une fenêtre s'ouvre dans laquelle s'affiche le "prompt" `->`. Chaque commande passée à Scilab est interprétée, le résultat est affiché, et l'utilisateur reçoit de nouveau le prompt `->` pour la commande suivante. Un **help** en ligne est disponible : utiliser **help item**.

Voici quelques exemples de commandes :

.....

```
-->a=1;
```

On donne la valeur 1 à la variable **a**. Le point virgule (`;`) en fin de ligne signifie que le résultat de la commande ne doit pas être affiché à l'écran (seulement évalué).

.....

```
-->c=[1 2];b=1.5
b      =
```

1.5

Deux commandes sur la même ligne, le résultat de la seconde étant affiché à l'écran (pas de point virgule).

.....

```
-->a=1;b=1.5;
```

```
-->2*a+b^2
ans      =
```

4.25

Les variables assignées sont gardées en mémoire. Le résultat est mis dans une variable par défaut appelée **ans**.

.....

```
-->sqrt([4,-4])
ans      =
```

```
!    2.    2.i !
```

Appel à une fonction avec un argument vectoriel. Le résultat est un vecteur ligne complexe.

.....

```
-->X=rand(2,2);[Q,R]=qr(X)
```

R =

```
! - 0.7850226 - 0.3181927 !
```

```
! 0.          0.0887097 !
```

Q =

```
! - 0.2691959 - 0.9630854 !
```

```
! - 0.9630854 0.2691959 !
```

Appel à une fonction admettant une variable d'entrée (matrice aléatoire X) et deux variables de sortie (matrices Q et R).

.....

3 Variables

Scilab reconnaît et manipule les variables matricielles suivantes (pour plus de détails, utiliser le **help**) : matrices constantes à coefficients réels ou complexes (**help matrices**), matrices polynomiales (**help poly**), matrices de chaînes de caractères (**help strings**), matrices booléennes (**help boolean**), matrices creuses (**help sparse**), matrices de fractions rationnelles (**help rational**). La structure de liste permet de manipuler des objets plus complexes (**help list**). Pour définir une matrice on utilise les crochets [et]. On sépare les colonnes par une virgule , (ou un blanc) et les lignes par un point virgule ;. Les scalaires, les vecteurs ligne, les vecteurs colonne sont considérés comme des matrices 1x1, 1xn et nx1 respectivement.

On peut définir une matrice sans en donner explicitement tous les éléments : par exemple **A(10,12)=3** donne une matrice à 10 lignes et 12 colonnes dont tous les éléments sont mis à zéro (sauf **A(10,12)** qui vaut 3). Scilab reconnaît les hypermatrices (matrices à N dimensions) et on peut ainsi définir une hypermatrice de taille (3,4,5) par la commande **A(3,4,5)=0**.

Le symbole deux-points : est utilisé pour créer des vecteurs particuliers (implicites), en particulier des vecteurs d'indices (**help colon**).

```
-->i=1:2:11
```

i =

```
! 1. 3. 5. 7. 9. 11. !
```

est le vecteur ligne de un à onze par pas de deux. Les manipulations principales matricielles sont :

■ Concatenation : la commande

```
-->A=[A11,A12;A21,A22]
```

crée une matrice A à partir des matrices A11, A12, A21, A22, comme on le fait avec des scalaires. Bien sûr, les matrices concaténées sont en nombre arbitraire, mais doivent être de dimensions compatibles. La taille d'une matrice s'obtient par la commande **[n1,nc]=size(A)**.

► Extraction : la commande

-->A(i,j)

où i et j sont deux vecteurs d'indices, renvoie la sous matrice extraite de A en sélectionnant les lignes d'indices $i(1)$, $i(2)$, ... et les colonnes d'indices $j(1)$, $j(2)$,

► Insertion : la commande

-->A(i,j)=B

insère la matrice B dans la sous matrice $A(i,j)$.

La notation $A(:,j)$ est utilisée par exemple pour construire la sous matrice de A obtenue en prenant toutes les lignes de A et les colonnes sélectionnées par le vecteur j . Enfin $A(:)$ est le vecteur colonne obtenu en concaténant les colonnes de la matrice A . Les opérations arithmétiques sur les matrices constantes, polynomiales ou de fractions rationnelles sont les suivantes :

► Additions $+$, soustractions $-$, multiplications $*$, puissances $^$ s'utilisent avec la syntaxe matricielle habituelle

► La transposée de la matrice A se note A'

► Les opérations "élément à élément" se notent .op , ainsi $X=A.*B$ désigne la matrice X telle que $X(i,j)=A(i,j)*B(i,j)$.

► Les fonctions élémentaires telle que abs , sin , cos , log , exp s'appliquent à des vecteurs ou matrices de taille quelconque, '**élément à élément**'.

► Les inversion à droite et à gauche se notent $/$ et $\backslash$. Pour une matrice carrée inversible $X=B/A$ s'interprète comme $X=\text{inv}(A)*B$ c'est-à-dire que X est la solution de $A*X=B$. Pour des matrice non inversibles, la résolution est faite au sens des moindres carrés.

Les chaînes de caractères sont indiquées entre deux quotes, par ex. "**chaîne**" et les polynômes sont créés par la commande `poly`, e.g. `p=poly(1,2,3,'x')`. Les matrices de chaînes de caractères et les matrices polynomiales (ainsi que les matrices creuses ou les matrices de fractions rationnelles) sont manipulées avec la même syntaxe que les matrices ordinaires réelles ou complexes.

4 Fonctions

Scilab contient plusieurs centaines de fonctions qui s'utilisent de manière interactive. Par exemple la commande `[Q,R]=qr(X)` réalise la factorisation QR de la matrice X et retourne les deux matrices de la factorisation dans les variables Q et R . Pour définir une nouvelle fonction dans Scilab il faut procéder en deux étapes. D'abord on définit la nouvelle fonction dans un fichier, en langage Scilab, puis on charge ce fichier à l'aide de la commande Scilab `getf`. La première ligne du fichier doit obligatoirement respecter une syntaxe telle que :

```
function [X,Y]=myfunction(A,B,C)
```

Cette syntaxe signifie que l'on définit une nouvelle fonction appelée `myfunction` qui a ici les 3 paramètres d'entrée A,B,C (vecteurs, matrices,...) et retourne deux paramètres de sortie (vecteurs, matrices,...) X,Y . Un tel fichier peut contenir plusieurs fonctions. Attention, ce fichier doit se terminer par un retour-chariot. Ce fichier, nommé par exemple `myfunction.sci` doit ensuite être chargé dans Scilab par la commande :

```
-->getf('myfunction.sci')
```

On peut alors utiliser dans Scilab la nouvelle fonction `myfunction`. La programmation en langage Scilab comprend, outre les appels aux fonctions, des boucles `for` ou `while`, des instructions de contrôle telles que `if-then-else` ou `select-case`. Pour plus de détails utiliser la commande `help`.

5 Quelques commandes utiles

On donne ici quelques points d'entrée avec les mots clés correspondants à des commandes Scilab. Utiliser le `help` et le "SEE ALSO" qui termine chaque fichier `help` pour d'autres commandes et plus de détails.

- ➡ Sauvegarde : `save`, `load`, `who`
- ➡ Ecriture, lecture : `read`, `write`
- ➡ Comparaisons, booléens : `==`, `>=`, `>`, `=`, `&` (ou `and`), `|` ou `or`
- ➡ Fenêtres de dialogue, boutons : `x_choose`, `x_dialog`, `m_xdialog`, `addmenu`
- ➡ Fonctions élémentaires : `sum`, `prod`, `cumsum`, `cumprod`, `sqrt`, `diag`, `cos`, `max`, `round`, `sign`, `fft`, `mean`, `st_deviation`, `corr`
- ➡ Exécution d'un fichier de commandes : `exec`
- ➡ Programmation : `function`, `deff`, `argn`, `for`, `if`, `end`, `while`, `select`, `warning`, `error`, `break`, `return`
- ➡ Debug : `pause`, `return`, `abort`
- ➡ Matrices particulières : `zeros`, `eye`, `ones`, `matrix`, `empty`
- ➡ Algèbre linéaire : `det`, `inv`, `qr`, `svd`, `bdiag`, `spec`, `schur`
- ➡ Polynômes : `poly`, `roots`, `coeff`, `horner`, `clean`, `freq`
- ➡ Systèmes linéaires et matrices de transfert : `syslin`
- ➡ Génération de nombres aléatoires : `rand`, `grand`
- ➡ Classement, tri : `sort`, `find`
- ➡ Splines, interpolation : `splin`, `interp`, `interpfn`
- ➡ Chaînes de caractères : `string`, `part`, `evstr`, `execstr`
- ➡ Graphiques : `plot`, `xset`, `driver`, `plot2d`, `xgrid`, `locate`, `plot3d`, `Graphics`
- ➡ Intégration d'équations différentielles : `ode`, `dassl`, `dassrt`, `odedc`
- ➡ Optimisation : `optim`, `quapro`, `linpro`, `external`
- ➡ Systèmes dynamiques interconnectés : `scicos`
- ➡ Ajout d'un programme C ou Fortran : `link`, `call`, `addinter`
- ➡ Evaluation du temps CPU d'une séquence d'instructions : `timer`, `tic`, `toc`

6 S'il ne fallait retenir qu'une chose !

Eviter les boucles en programmant "matriciellement".

Pour mieux vous convaincre : utiliser les fonctions, décrites ci-après, `rand`, `variance` et `mean`, pour écrire deux fonctions `scilab`, l'une vectorielle l'autre scalaire qui calculent la moyenne et la variance d'un vecteur `x` de taille 10^4 , généré aléatoirement suivant une loi normale. Comparer les temps de calcul de vos fonctions, en utilisant la fonction `timer()` ou `tic`, `toc`.

De manière générale penser que tout le code Scilab est visible : des centaines de fonctions sont disponibles, et il faut absolument éviter de réécrire ce qui (peut-être) existe déjà en consultant le help et le code source des fonctions de Scilab. La commande **apropos keyword** peut aider à trouver une fonction particulière qui a un lien avec **keyword**.

7 Quelques fonctions utiles

*Pour le TP Pricing d'option et portefeuille de couverture, vous serez amenés à utiliser les fonctions **scilab** décrites ci-après. Ces fonctions sont optimisées en mode vectoriel, ce qui les rend particulièrement efficaces. A titre d'exemple, comparez les temps de calcul obtenus par une programmation séquentielle, qui fait appel à des boucles imbriquées et un appel aux fonctions **scilab**.*

7.1 rand

La fonction **rand** est un générateur de nombres aléatoires.

Quelques façons d'appeler rand

- ➡ **rand(m1,m2,... [,key])** où les m_i sont des entiers, dimension de la matrice à générer, **key** indique le choix de la loi choisie, **key** peut prendre les valeurs **'uniform'** ou **'normal'**. La notation **[,key]** indique que le paramètre **key** est optionnel.
- ➡ **rand(x)** génère une matrice de nombres aléatoires, dont les dimensions sont identiques à celle de x
- ➡ **rand(key)** permet de choisir la loi pour le générateur, uniforme ou normale, par défaut c'est une loi uniforme.
- ➡ **rand("info")** indique la loi du générateur.

ATTENTION **rand(m)**, si m est un scalaire, retourne un scalaire généré aléatoirement, pour obtenir un vecteur de nombres aléatoires de longueur m il faut utiliser : **rand(m,1)**

Exemples

1. **x=rand(2,3,'uniform')** renvoie dans x une matrice de taille 2×3 de nombres aléatoires uniformément distribués sur $[0,1]$, dans **scilab** cela donne :

```
--> x=rand(2,3,'uniform')
x  =

!   0.8415518    0.4094825    0.1138360 !
!   0.4062025    0.8784126    0.1998338 !
```

2. **y=rand(3,2,2,'normal')** renvoie dans y une matrice de taille $3 \times 2 \times 2$ de nombres aléatoires suivant une loi normale dont le générateur est une Gaussienne de moyenne 0 (centrée) et de variance 1, dans **scilab** cela donne :

```
--> y=rand(3,2,2,'normal')
y  =
```

```
(:,:,1,1)

!   0.4694334    0.3268713 !
!   1.4027611    0.6886984 !
!   2.0320066    0.0613533 !
(:,:,2,1)

! - 2.1894218    0.4249849 !
! - 0.1890526   - 0.1481575 !
!   0.7088748   - 0.7460990 !
```

7.2 sum

La fonction **sum** permet de sommer les valeurs d'un vecteur ou les valeurs d'une matrice, éventuellement en ligne ou en colonne.

Quelques façons d'appeler sum

- ⇒ **y=sum(x)** retourne la somme de tous les éléments de la matrice **x**
- ⇒ **y=sum(x,'r')** ou **y=sum(x,1)** retourne la somme des lignes de la matrice **x**
- ⇒ **y=sum(x,'c')** ou **y=sum(x,2)** retourne la somme des colonnes de la matrice **x**

Exemples

Soit la matrice *x*, telle que :

```
x =
```

```
!   1.    4. !
!   2.    5. !
!   3.    6. !
```

1. **sum(x)**, somme des éléments de *x* dans **scilab** cela donne :

```
-->sum(x)
ans =
```

```
21.
```

2. **sum(x,1)**, somme des éléments des lignes de *x* dans **scilab** cela donne :

```
-->sum(x,1)
ans =
```

```
!   6.    15. !
```

3. **sum(x,2)**, somme des éléments des colonnes de *x* dans **scilab** cela donne :

```
-->sum(x,2)
ans =
```

```
!   5. !
!   7. !
!   9. !
```

7.3 variance

La fonction **variance** permet de calculer la variance des valeurs d'un vecteur ou d'une matrice.

Définition

La variance *var* est définie, dans **scilab**, par :

$$var = \frac{\sum_{i=1}^M (x_i^2 - \bar{x}^2)}{M - 1}$$

où $\bar{x} = \frac{\sum_{i=1}^M x_i}{M}$ est la moyenne des x_i .

Quelques façons d'appeler variance

- ➡ **s=variance(x)** retourne la variance de tous les éléments de la matrice x
- ➡ **s=variance(x,'r')** ou **s=variance(x,1)** retourne la variance des valeurs lignes de la matrice x
- ➡ **s=variance(x,'c')** ou **s=variance(x,2)** retourne la variance des valeurs colonnes de la matrice x

Exemples

Soit la matrice *x*, telle que :

```
x =  
  
!   1.   2.   3. !  
!   4.   5.   6. !
```

1. **s=variance(x)**, variance des éléments de *x* dans **scilab** cela donne :

```
-->s=variance(x)  
s =  
  
3.5
```

2. **s=variance(x,1)**, variance des éléments des lignes de *x* dans **scilab** cela donne :

```
-->s=variance(x,1)  
s =  
  
!   4.5   4.5   4.5 !
```

3. **s=variance(x,2)**, variance des éléments des colonnes de *x* dans **scilab** cela donne :

```
-->s=variance(x,2)  
s =  
  
!   1. !  
!   1. !
```

7.4 st_deviation

La fonction `st_deviation` permet de calculer l'écart type (*standard deviation*) des valeurs d'un vecteur ou d'une matrice.

Définition

L'écart type *ec* est définie, dans `scilab`, par :

$$ec = \sqrt{\frac{\sum_{i=1}^M (x_i - \bar{x})^2}{M - 1}}$$

où $\bar{x} = \frac{\sum_{i=1}^M x_i}{M}$ est la moyenne des x_i .

Quelques façons d'appeler `st_deviation`

- ➡ `s=st_deviation(x)` retourne l'écart type de tous les éléments de la matrice `x`
- ➡ `s=st_deviation(x,'r')` ou `s=st_deviation(x,1)` retourne l'écart type des valeurs lignes de la matrice `x`
- ➡ `s=st_deviation(x,'c')` ou `s=st_deviation(x,2)` retourne l'écart type des valeurs colonnes de la matrice `x`

Exemples

Soit la matrice x , telle que :

```
x =  
  
!   1.   2.   3.   4. !  
! - 4.  - 3.  - 2.  - 1. !
```

1. `st_deviation(x)`, écart type des éléments de x dans `scilab` cela donne :

```
-->st_deviation(x)  
ans =  
  
2.9277002
```

2. `st_deviation(x,1)`, écart type des éléments des lignes de x dans `scilab` cela donne :

```
-->st_deviation(x,1)  
ans =  
  
!   3.5355339   3.5355339   3.5355339   3.5355339 !
```

3. `st_deviation(x,2)`, écart type des éléments des colonnes de x dans `scilab` cela donne :

```
-->st_deviation(x,2)  
ans =  
  
!   1.2909944 !  
!   1.2909944 !
```

7.5 zeros et ones

La fonction **zeros** génère une matrice de 0, **ones** génère des matrices de 1. Leur utilisation est identique :

Comment les appeler ?

- ➡ **y=zeros()**, respectivement **y=ones()**, initialise le scalaire y à 0, respectivement à 1
- ➡ **y=zeros(x)**, respectivement **y=ones(x)**, initialise la matrice y à 0, respectivement à 1, où y a les mêmes dimensions que x .
- ➡ **y=zeros(m1,m2,...)**, respectivement **y=ones(m1,m2,...)**, initialise à 0, respectivement à 1, une matrice y de taille $m_1 \times m_2 \times \dots$

ATTENTION si m est un scalaire, **zeros(m)**, respectivement **ones(m)**, retourne un scalaire égal à 0, respectivement à 1 ; pour obtenir un vecteur de longueur m initialisé à 0 il faut utiliser : **zeros(m,1)**, respectivement **ones(m,1)**

8 Variables prédéfinies

symbole	définition	sens
%s	<code>poly(0,'s')</code>	polynôme s
%i	<code>sqrt(-1)</code>	$i^2 = -1$
%t ou %T	<code>1==1</code>	booléen "vrai"
%f ou %F	<code>1~=1</code>	booléen "faux"
%inf	<code>1/0</code>	∞
%nan	<code>0/0</code>	"Not a Number"
%e	<code>exp(1) = 2.718...</code>	$\text{Log}(e) = 1$
%eps	<code>max{x, 1+x=1}</code>	epsilon machine
%pi	<code>4*atan(1)</code>	π

9 Programmation

Voici des exemples simples de programmation. Les ... sont des instructions Scilab quelconques. Pour plus de précision utiliser la commande **help**.

- ➡ Boucle **for** : `for i=1:n; ...; end`
- ➡ Boucle **while** : `while k<n; ...; end`
- ➡ Instruction If-then-else :
 - `if x==0 then ...; end`
 - `if x==0 then ...; else ...;end`
 - `if x==0 then ...; elseif x==1 ...; else ...; end`
- ➡ Selection :
 - `select x`
 - `case 0`
 - `...`
 - `case 1`
 - `...`
 - `else`
 - `...`

```

end
    ► Sortie de boucle :
    for k=1:100; ...;
        if x>1000 then break;end
    end
    ► Sortie de fonction :
    function y=f(x)
        if x>1000 then ...; return ;end

```

10 Graphique

x et y étant deux vecteurs de même taille n , la commande `plot(x,y)` ouvre une fenêtre graphique et produit un graphe des points $y(i)$ en fonction des points $x(i)$. (Si x est omis le vecteur $x=1:n$ est pris par défaut). Pour obtenir une copie postscript (imprimante) du dessin cliquer sur le bouton **print** du menu **File** de la fenêtre graphique. Pour insérer le graphique dans un texte **Latex** cliquer sur le bouton **Export** du même menu. Pour dessiner plusieurs courbes sur un même graphique, utiliser la commande `plot2d(X,Y)`. Cette commande donne (en superposition) un graphe de chaque *colonne* de la matrice Y en fonction de chaque *colonne* de la matrice X . La plupart des commandes graphiques travaillent par défaut en mode superposition. On utilisera la commande `xbasc()` pour effacer les fenêtres graphiques (ou utiliser la commande `clear` dans le menu **File** de la fenêtre graphique courante).

La commande `xset()` ouvre une fenêtre de dialogue permettant de fixer un environnement graphique particulier. La commande `xset('default')` remplace le contexte graphique par défaut.

Pour les graphiques complexes (3D, animation) copier sur les démos !

11 Exporter des graphiques

- On procède comme suit. Affichez un graphique en utilisant les instructions `x=0:0.05:10;y=sin(x);plot(x,y)` par exemple. En utilisant le sous-menu **export** du menu **file** de la fenêtre graphique choisissez l'option **Postscript-Latex**, spécifiez un nom de fichier (**graphique** par exemple) et validez vos choix en cliquant (une seule fois !) sur le bouton **OK**. Attention, choix **portrait** => dessin horizontal (tel qu'il apparaît à l'écran) et choix **landscape** => dessin vertical (il faudra tourner la page d'un quart de tour).

Deux fichiers sont créés : **graphique.tex** et **graphique.eps**. Vous disposez par ailleurs de votre propre fichier **Latex** dans lequel vous désirez introduire le graphique **scilab**.

Insérez alors les instructions suivantes dans votre fichier **Latex**.

```

\input{graphique.tex}
\dessin{Ceci est la 1\'}{egende}{Numerodefigure}

```

Compiler votre fichier **Latex** (command `latex`), générer le fichier postscript (commande `dvips`).

La figure ci-dessous a été obtenue par le fichier **Latex** suivant, les fichiers **graphique.tex** et **graphique.eps** étant obtenus par **Scilab** en exportant le graphique donné par la commande `x=0:0.05:10;y=sin(x);plot(x,y)` :

```

\documentclass{article}
\usepackage{epsf,epsfig}
\begin{document}
\section{My first plot}
\input{graphique.tex}
\dessin{Ceci est la l\'egende}{Numerodefigure}
\end{document}

```

12 My first plot

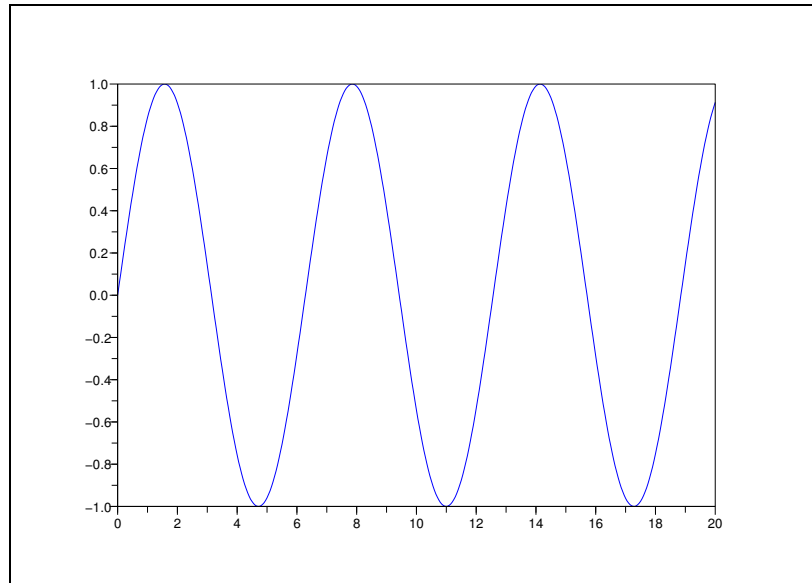


FIG. 1 – Ceci est la légende

NB : Sous TeXShop (l'éditeur de Latex pour Mac) il faut choisir, dans le mode de composition, "TeX et DVI" pour que ce protocole fonctionne. Avec "Pdftex" comme mode de composition, il faut, auparavant, convertir le fichier image **graphique.eps** en un fichier pdf, et ensuite, remplacer, dans le fichier **graphique.tex** (qui s'ouvre avec le même éditeur de texte que les autres fichiers Latex), **graphique.eps** par **nom_du_fichier.pdf**.