Initiation à Scilab

Yassine Ariba

Sommaire

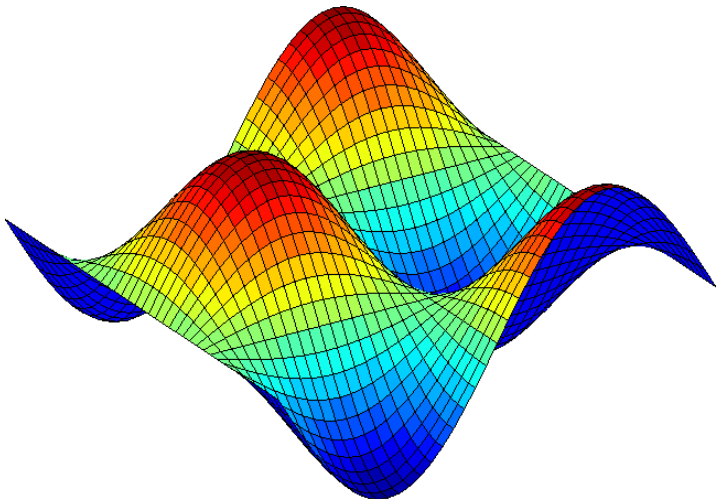
Sommaire

Qu'est ce que Scilab ?

Scilab est la contraction de *Scientific Laboratory*. Scilab est :

- un logiciel de calcul numérique,
- un langage de programmation interprété,
- utilisé pour toutes les applications scientifiques et l'ingénierie,
- multi-plateforme : Windows, MacOS et Linux,

Conçu initialement par l'Inria à partir des années 80, le logiciel est maintenant développé par la société française Scilab Entreprises



Plus d'infos : `www.scilab.org`

Principales fonctionnalités de Scilab :

- Mathématiques et simulation
- Visualisation 2D et 3D
- Optimisation
- Statistiques
- Automatique
- Traitement du signal
- Développement d'applications

Plus d'infos : `www.scilab.org`

Licence

- Scilab est un logiciel *open source*.
- Il est régi par la licence CeCILL [1] (compatible GPL).
- Il est une alternative gratuite à MATLAB® [2].
- Le logiciel, ainsi que les sources, sont téléchargeables à l'adresse :

`http://www.scilab.org/download/`

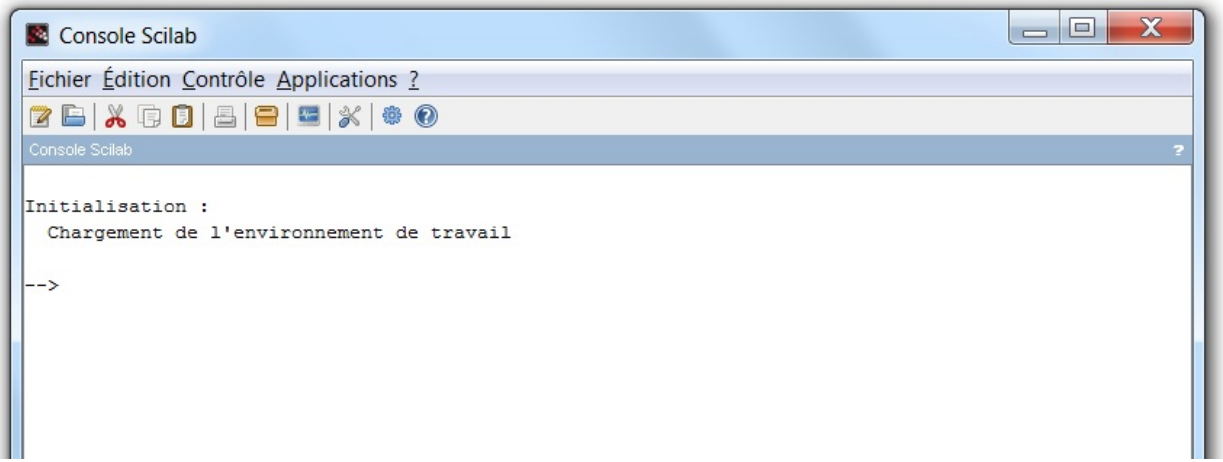
La version de Scilab utilisée dans cette initiation est la

`version 5.4.0`

1. Plus d'infos : `http://www.cecill.info`
2. MATLAB est une marque déposée par la société The MathWorks, Inc.

Getting started

Scilab s'utilise interactivement en tapant des commandes dans la *console*.



- Les instructions doivent être entrées sur l'invite `-->`
- puis lancées en tapant la touche entrée.
- Scilab exécute les calculs correspondants,
- et renvoie sa réponse dans la console ou une nouvelle fenêtre.

Premier exemple introductif :

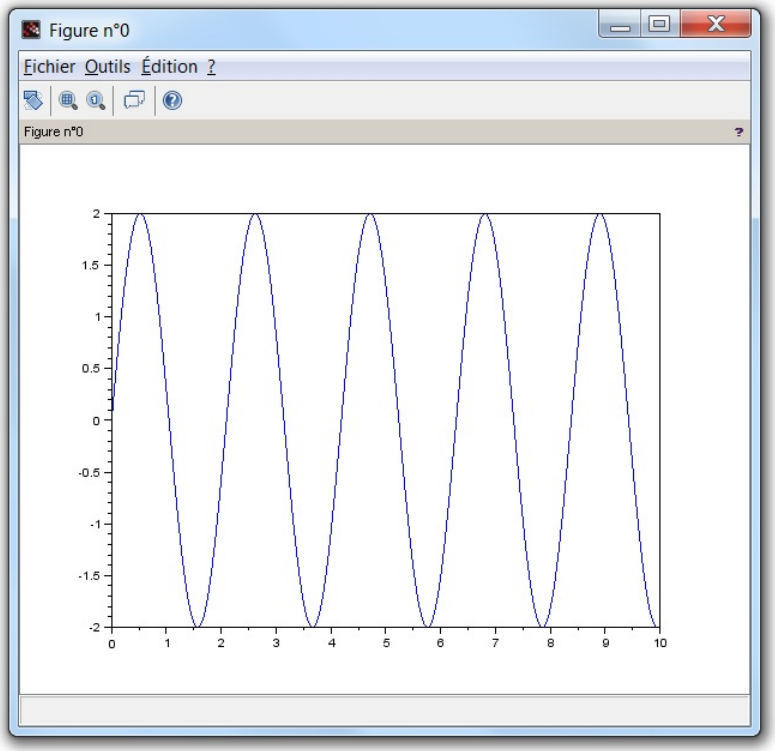
```
--> A = 2;
--> t = [0:0.01:10];
--> y = A*sin(3*t);
--> plot(t,y);
```

- Ligne 1 : affectation de la valeur 2 à la variable A.
- Ligne 2 : définition d'un vecteur t dont les composantes vont de 0 à 10 par pas de 0.01.
- Ligne 3 : calcul d'un vecteur y à partir d'opérations mathématiques.
- Ligne 4 : tracé de y par rapport à t sur un graphique 2D.

Notons que le " ; " spécifie à Scilab de ne pas afficher sa réponse.

Premier exemple introductif :

```
--> A = 2;
--> t = [0:0.01:10];
--> y = A*sin(3*t);
--> plot(t,y);
```



Second exemple introductif :

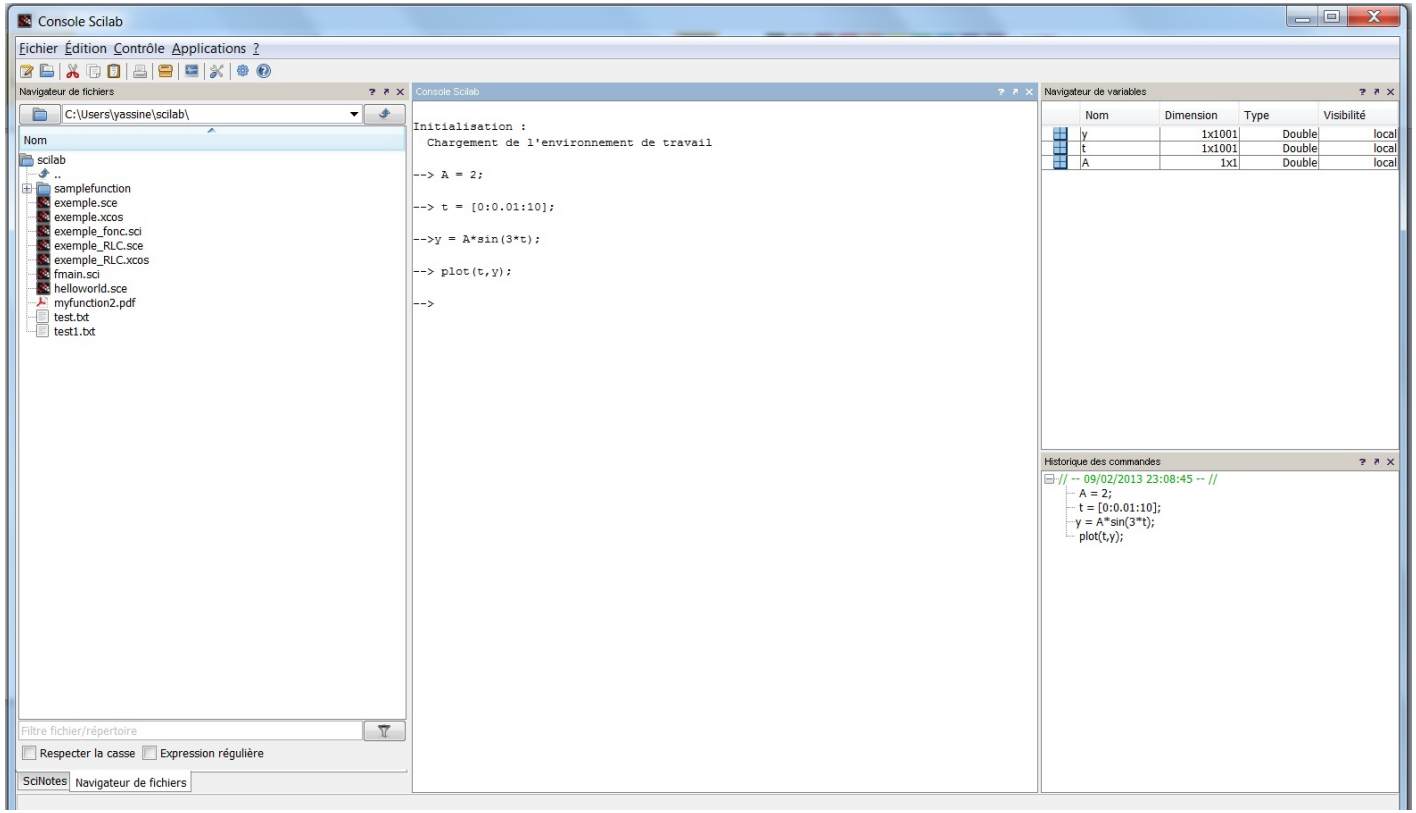
Soit le système d'équations linéaires suivant

$$\begin{cases} 2x_1 + x_2 = -5 \\ 4x_1 - 3x_2 + 2x_3 = 0 \\ x_1 + 2x_2 - x_3 = 1 \end{cases} \quad \text{ou encore} \quad \begin{bmatrix} 2 & 1 & 0 \\ 4 & -3 & 2 \\ 1 & 2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -5 \\ 0 \\ 1 \end{bmatrix}$$

Résolution du système à l'aide de Scilab

```
--> A = [2 1 0 ; 4 -3 2 ; 1 2 -1];
--> b = [-5;0;1];
--> x = inv(A)*b
 x   =
    1.75
  - 8.5
  - 16.25
```

Scilab propose en fait un environnement intégrant divers fenêtres pour une interface conviviale.



- La console (interface de commande avec Scilab)
- Historique des commandes (mémorise les commandes passées)
- Navigateur de fichiers (explorateur pour ouvrir des fichiers)
- Navigateur de variables (variables actuellement définies dans Scilab)
- ...

Sommaire

Opérations et fonctions élémentaires

Dans son utilisation la plus simple, Scilab est une “super-calculatrice” :

```
--> (1+3)*0.1
 ans  =
    0.4

--> 4^2/2
 ans  =
    8.

--> 2*(1+2*%i)
 ans  =
    2. + 4.i

--> %i^2
 ans  =
  - 1.

--> cos(3)^2 + sin(3)^2
 ans  =
    1.

--> exp(5)
 ans  =
    148.41316

--> abs(1+%i)
 ans  =
    1.4142136
```

Opérations élémentaires

+	addition
-	soustraction
*	multiplication
/	division à droite
\	division à gauche
^	puissance

Quelques fonctions élémentaires

sin	cos	tan	cotg
asin	acos	atan	sec
sinh	cosh	tanh	csc
abs	real	imag	conj
exp	log	log10	log2
sign	modulo	sqrt	lcm
round	floor	ceil	gcd

```
--> conj(3+2*%i)
 ans  =
    3. - 2.i

--> log10(10^4)
 ans  =
    4.
```

Opérations booléennes

- La valeur booléenne *vraie* s'écrit : `%T`.
- La valeur booléenne *fausse* s'écrit : `%F`.

`&`	*et* logique
`|`	*ou* logique
$\sim$	*non* logique
`==`	égal
$\sim=$ ou `<>`	différent
`<` (`<=`)	inférieur (ou égal)
`>` (`>=`)	supérieur (ou égal)

```
--> %T & %F
 ans  =
  F

--> 2 == 2
 ans  =
  T

--> 2 < 3
 ans  =
  T
```

Variables

Une variable est définie par un opérateur d'affectation : " = "

```
--> a = 2.5;
--> b = 3;
--> c = a*b
 c  =
    7.5

--> c+d
     !--error 4
Variable non définie : d
```

- Un nom de variable peut être composé de lettres `a` → `z`, `A` → `Z`, de chiffres `0` → `9` et des caractères `%`, `_`, `!`, `#`, `?`, `$`.
- Scilab est sensible à la casse.
- Ne pas confondre l'affectation " = " avec l'égalité mathématique.
- La déclaration de variable est implicite (quelque soit le type).

Variables mathématiques pré-définies

`%i`	le nombre imaginaire $i = \sqrt{-1}$
`%e`	la constante d'Euler e
`%pi`	le nombre π
`%inf`	l'infini ∞
`%t ou %T`	valeurs booléennes vraies
`%f ou %F`	valeurs booléennes fausses

```
--> cos(2*%pi)
 ans  =
    1.

--> %i^2
 ans  =
  - 1.
```

Sommaire

Définition et manipulation de vecteurs

Un vecteur est défini par une succession de nombre entre crochets

```
--> u = [0 1 2 3]
 u  =
    0.    1.    2.    3.
```

Génération automatique

```
--> v = [0:0.2:1]
 v  =
    0.    0.2    0.4    0.6    0.8    1.
```

Syntaxe : **`debut:pas:fin`**

Les fonctions mathématiques sont applicables et sont opérées sur chaque élément

```
--> cos(v)
 ans  =
    1.    0.980    0.921    0.825    0.696    0.540
```

On peut aussi définir des vecteurs colonnes

```
--> u = [1;2;3]
 u  =
    1.
    2.
    3.
```

Quelques fonctions utiles :

`length`	renvoie la taille du vecteur
`max`	renvoie la valeur maximale
`min`	renvoie la valeur minimale
`mean`	renvoie la valeur moyenne
`sum`	calcul la somme des éléments
`prod`	calcul la produit des éléments

```
--> length(v)
 ans  =
    6.

--> mean(v)
 ans  =
    0.5
```

Définition et manipulation de matrices

Les matrices sont définies ligne par ligne avec le séparateur “;”

```
--> A = [1 2 3 ; 4 5 6 ; 7 8 9]
 A  =
    1.    2.    3.
    4.    5.    6.
    7.    8.    9.
```

Matrices particulières :

`zeros(n,m)`	matrices de taille $n \times m$ de zéros
`ones(n,m)`	matrices de uns
`eye(n,n)`	matrice identité
`rand(n,m)`	matrice aléatoire (valeurs $\in [0, 1]$)

Accès aux éléments de la matrice : `A(`*sélection ligne(s)*`,`*sélection colonne(s)*`)`

```
--> A(2,3)
 ans  =
    6.

--> A(2,:)
 ans  =
    4.    5.    6.

--> A(:,[1 3])
 ans  =
    1.    3.
    4.    6.
    7.    9.
```

cas particulier pour les vecteurs : 1 seul argument `v(3)` (ce qui donne `0.4`)

Les éléments peuvent être directement modifiés

```
--> A(2,3) = 0;
--> A
 A  =
    1.    2.    3.
    4.    5.    0.
    7.    8.    9.
```

Quelques fonctions utiles :

`size`	renvoie les dimensions d'une matrice
`det`	renvoie le déterminant d'une matrice
`inv`	calcul la matrice inverse
`rank`	renvoie le rang d'une matrice
`diag`	extrait la diagonale d'une matrice
`triu`	extrait la matrice-triangle supérieure
`tril`	extrait la matrice-triangle inférieure
`spec`	renvoie les valeurs propres d'une matrice

```
--> B = [1 0 ; 2 2];
--> det(B)
 ans  =
    2.

--> inv(B)
 ans  =
    1.    0.
  - 1.    0.5

--> triu(A)
 ans  =
    1.    2.    3.
    0.    5.    6.
    0.    0.    9.
```

Opérations matricielles

Les opérations de base `+, -, *, /, ^` sont directement applicables

- Attention à la compatibilité des dimensions des opérandes.
- Opérateur transposé : “.’” , opérateur transposé et conjugué : “’”

```
--> C = [ 1 0 ; 3 1 ; 0 2];
--> D = [1 1 ; 4 0];
--> B + D
 ans  =
    2.    1.
    6.    2.

--> B * inv(B)
 ans  =
    1.    0.
    0.    1.

--> A * C
 ans  =

    7.    8.
    19.   17.
    31.   26.

--> A + B
        !--error 8
Addition incohérente.
```

Les fonctions élémentaires sont appliquées à chaque élément

```
--> M = [0 %pi/2 ; -%pi/2 %pi ];
--> sin(M)
 ans  =
    0.     1.
  - 1.     1.225D-16

--> t = [0:0.2:1];
--> exp(t)
 ans  =
    1.    1.2214    1.4918    1.8221    2.2255    2.7182
```

Il existe des versions spécifiques de certaines fonctions pour le calcul matriciel

expm	logm	sqrtm
sinm	cosm	^

Opérations éléments par éléments

.*	./	.^

```
--> A = [0 4 ; 1 2];
--> B = [1 2 ; 5 -3];
--> A * B
 ans  =
    20.   - 12.
    11.   - 4.

--> A .* B
 ans  =
    0.     8.
    5.   - 6.

--> A.^2
 ans  =
    0.     16.
    1.     4.

--> exp(t)./(t+1)
 ans  =
    1.    1.0178    1.0655    1.1388    1.2364    1.3591
```

Sommaire

Les graphes 2D

Le tracé d'une courbe dans un plan x-y se base sur : **plot**

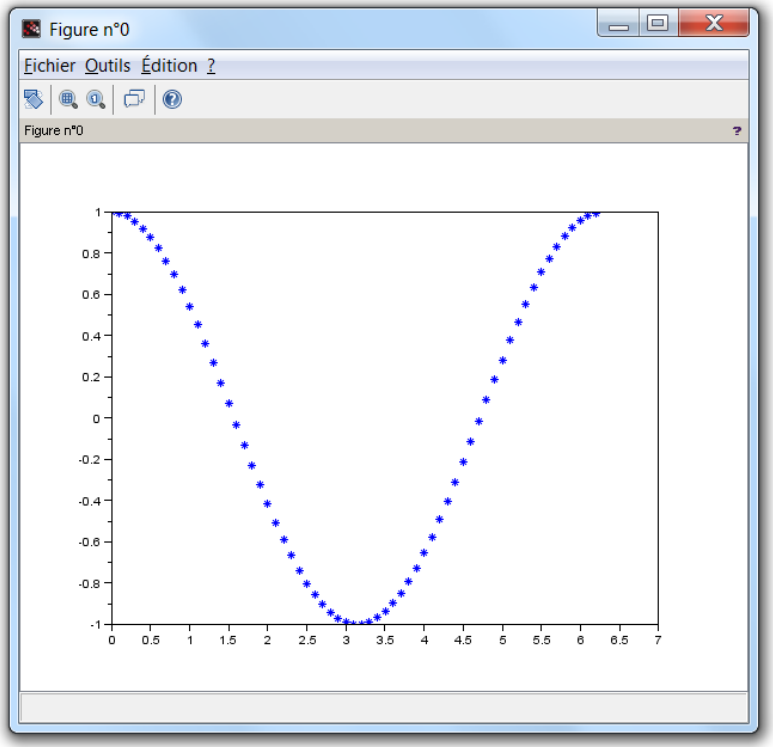
```
--> x = [0:0.1:2*%pi];
--> y = cos(x);
--> plot(x,y,'*')
```

Les graphes 2D

Le tracé d'une courbe dans un plan x-y se base sur : `plot`

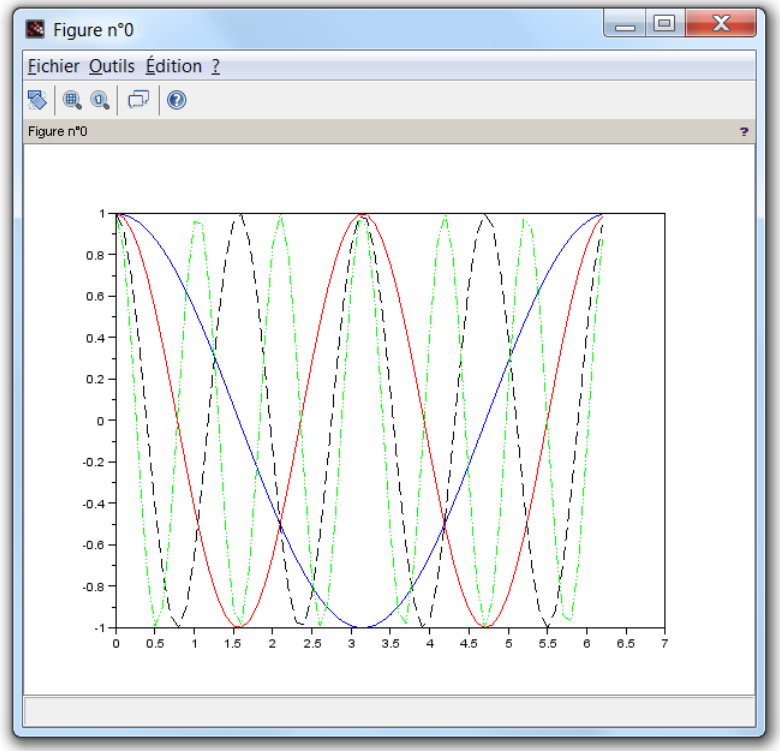
```
--> x = [0:0.1:2*%pi];
--> y = cos(x);
--> plot(x,y,'*')
```

- `plot` trace un point pour chaque couple `x(i)-y(i)`.
- `x` et `y` doivent être de même taille.
- Par défaut, une ligne est tracée entre chaque point.
- Le 3$^{\text{ième}}$ argument défini le style de la courbe.



```
--> x = [0:0.1:2*%pi];
--> y2 = cos(2*x);
--> y3 = cos(4*x);
--> y4 = cos(6*x);
--> plot(x,y1);
--> plot(x,y2,'r');
--> plot(x,y3,'k:');
--> plot(x,y4,'g--');
```

```
--> x = [0:0.1:2*%pi];
--> y2 = cos(2*x);
--> y3 = cos(4*x);
--> y4 = cos(6*x);
--> plot(x,y1);
--> plot(x,y2,'r');
--> plot(x,y3,'k:');
--> plot(x,y4,'g--');
```



- Plusieurs courbes peuvent être superposées.
- La commande `clf` permet d'effacer les tracés.
- Voir l'aide de `LineSpec` pour plus de détails sur les types de tracés.

Les graphes 3D

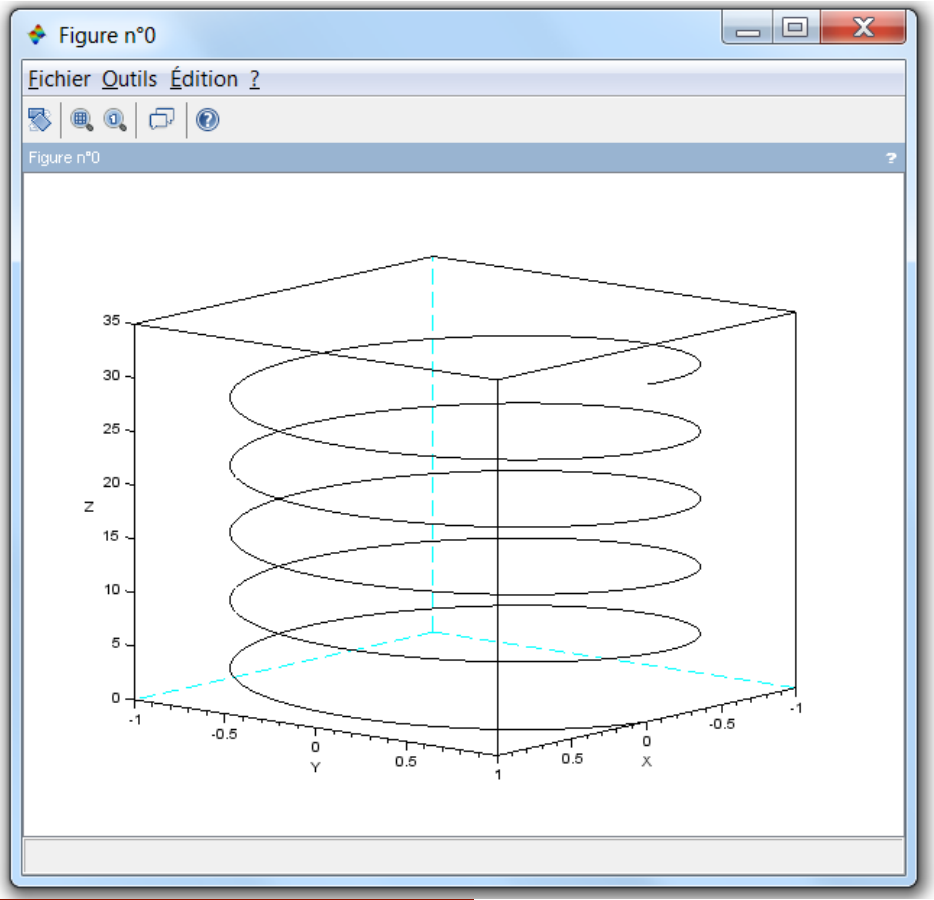
Le tracé d'une courbe paramétrique dans l'espace se base sur : **param3d**

```
--> t = 0:0.01:10*%pi;
--> x = sin(t);
--> y = cos(t);
--> z = t;
--> param3d(x,y,z);
```

Les graphes 3D

Le tracé d'une courbe paramétrique dans l'espace se base sur : `param3d`

```
--> t = 0:0.01:10*%pi;
--> x = sin(t);
--> y = cos(t);
--> z = t;
--> param3d(x,y,z);
```

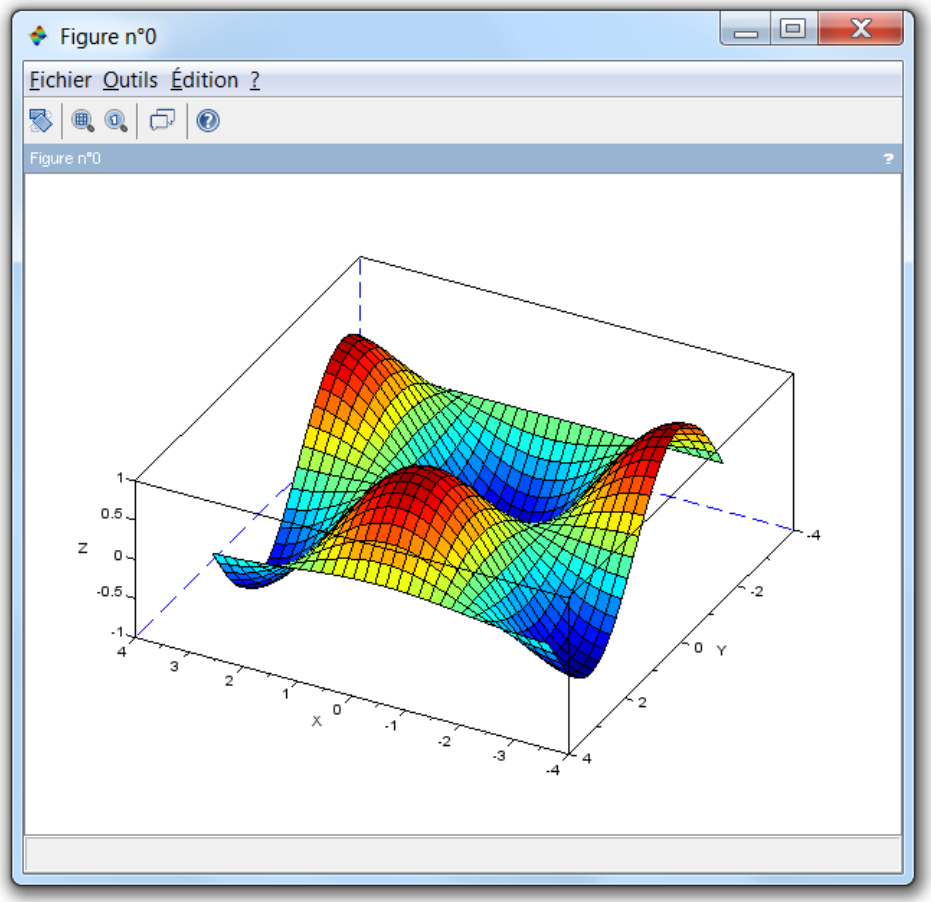


Le tracé d'une surface dans l'espace se base sur : **surf**

```
--> x = [-%pi:0.2:%pi];
--> y = [-%pi:0.2:%pi];
--> [X,Y] = meshgrid(x,y);
-->  Z = cos(X).*sin(Y);
--> surf(X,Y,Z)
--> f=gcf();
--> f.color_map = jetcolormap(32);
```

Le tracé d'une surface dans l'espace se base sur : **surf**

```
--> x = [-%pi:0.2:%pi];
--> y = [-%pi:0.2:%pi];
--> [X,Y] = meshgrid(x,y);
-->  Z = cos(X).*sin(Y);
--> surf(X,Y,Z)
--> f=gcf();
--> f.color_map = jetcolormap(32);
```



Généralités

Scilab possède de nombreuses fonctions graphiques :

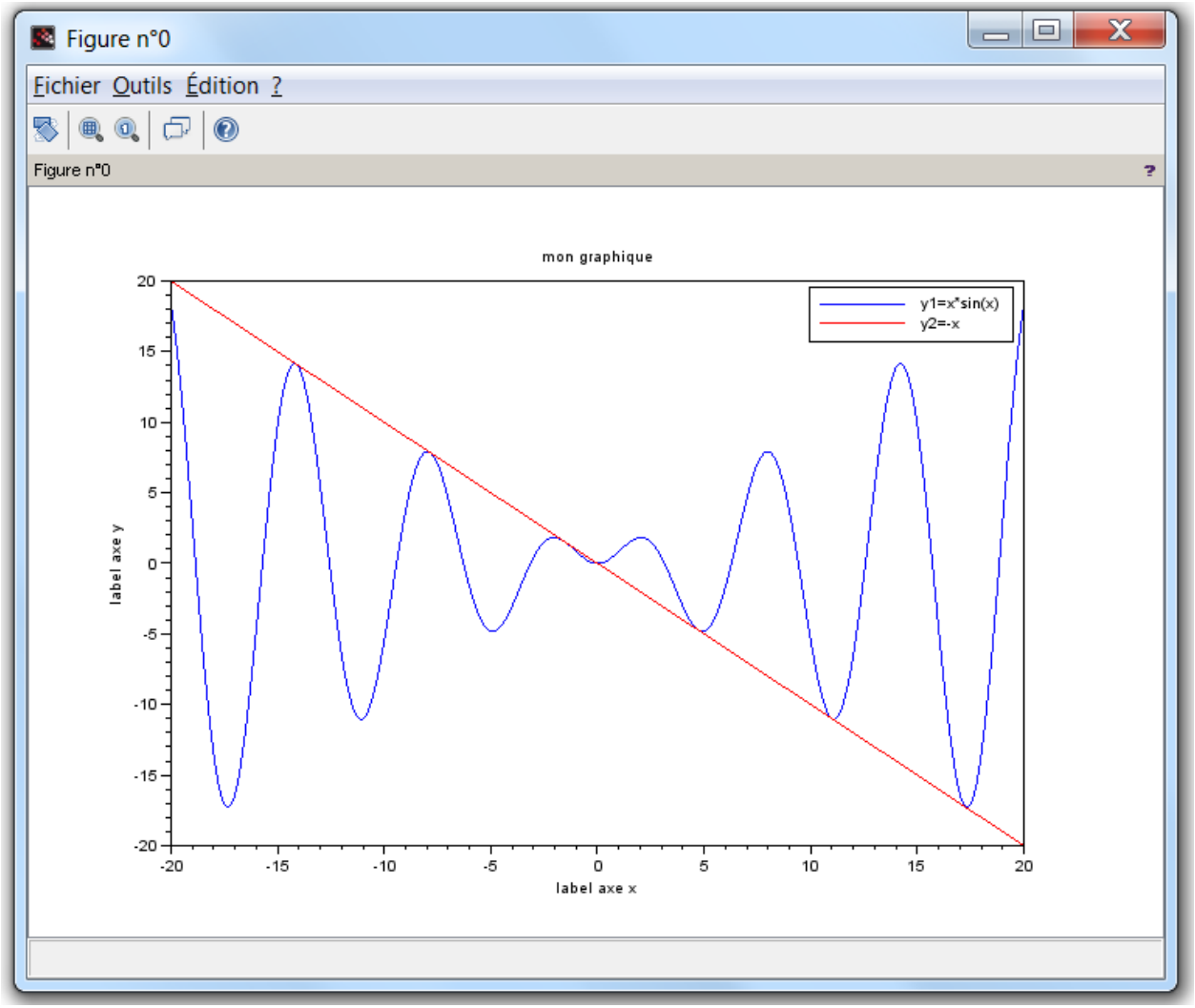
`plot`	graphe 2D
`contour`	courbes de niveau dans un plan
`surf`	surface 3D
`pie`	graphe en “camembert”
`histplot`	histogramme
`hist3d`	histogramme 3D
`bar`	graphe en “baton”
`polarplot`	graphe en coordonnées polaires

Des instructions sont aussi disponibles pour l’habillage d’une figure :

`title`	ajout d’un titre pour la figure
`xtitle`	ajout d’un titre et de labels pour les axes
`legend`	ajout d’une légende

```
--> x = linspace(-20,20,1000);
--> y1 = x.*sin(x);
--> y2 = -x;
--> plot(x,y1,'b',x,y2,'r')
--> xtitle('mon graphique','label axe x','label axe y');
--> legend('y1=x*sin(x)','y2=-x');
```

```
--> x = linspace(-20,20,1000);
--> y1 = x.*sin(x);
--> y2 = -x;
--> plot(x,y1,'b',x,y2,'r')
--> xtitle('mon graphique','label axe x','label axe y');
--> legend('y1=x*sin(x)','y2=-x');
```



Sommaire

Les scripts

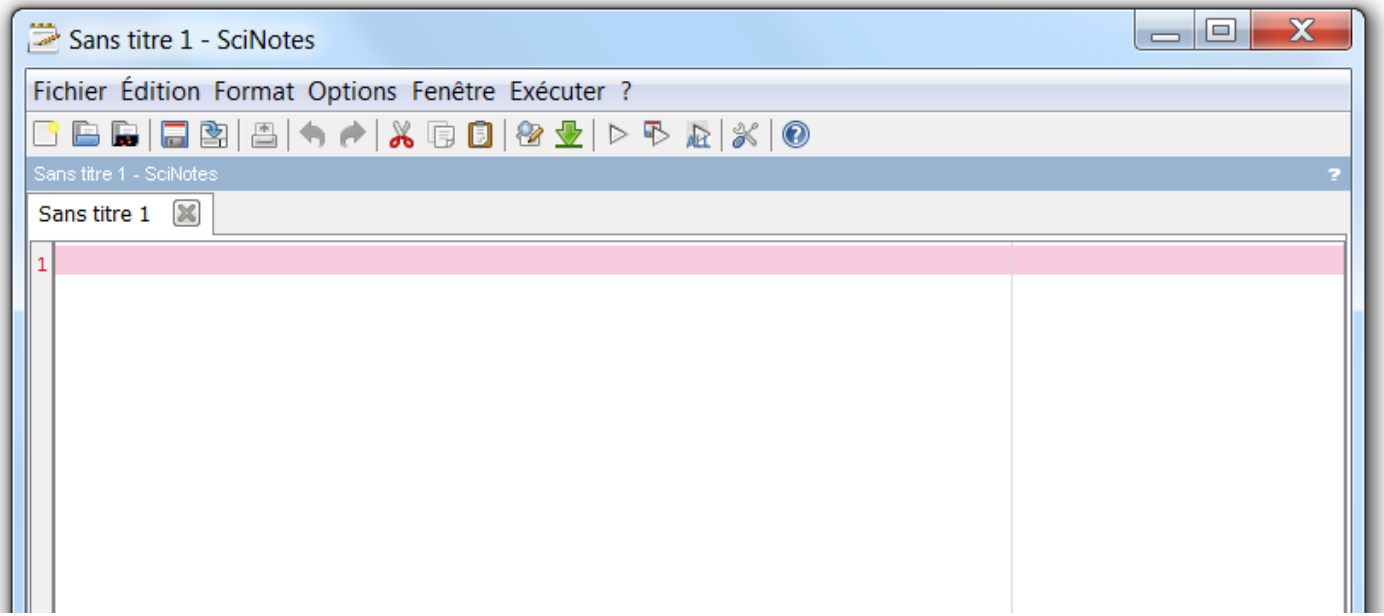
Un script est un ensemble d'instructions rassemblées dans un fichier.

- Scilab propose un véritable langage de programmation (interprété).
- Scilab a son propre éditeur, mais n'importe quel éditeur de texte suffit.
- Les fichiers ont pour extension "`.sce`".
- L'éditeur se lance depuis "*Applications* > *SciNotes*" ou en tapant `editor` dans la console.

Les scripts

Un script est un ensemble d'instructions rassemblées dans un fichier.

- Scilab propose un véritable langage de programmation (interprété).
- Scilab a son propre éditeur, mais n'importe quel éditeur de texte suffit.
- Les fichiers ont pour extension "`.sce`".
- L'éditeur se lance depuis "*Applications* > *SciNotes*" ou en tapant `editor` dans la console.



Exemple de script (dans l'éditeur) : **monscript.sce**

```
// rayon de la sphère
 r = 2;

 // calcul de l'aire
 A = 4*%pi*r^2;

 // calcul du volume
 V = 4*%pi*r^3/3;

 disp(A,'Aire:');

 disp(V,'Volume:');
```

Dans la console :

```
-->exec('monscript.sce', -1)

 Aire:

    50.265482

 Volume:

    33.510322
```

Le fichier doit se situer dans le *répertoire courant*

- Commentaires : les mots qui suivent `//` ne sont pas interprétées.
- Le répertoire courant peut être modifié dans le menu *Fichier* de la console.
- Le chemin complet peut aussi être spécifié dans la commande

```
exec('C:\Users\yassine\scilab\monscript.sce', -1)
```

- Un raccourci dans la barre d'outils de Scinotes permet de lancer l'exécution.
- Les variables définies avant (directement dans la console) sont visibles et modifiables dans le script.

Autre exemple (dans l'éditeur) : **monscript2.sce**

```
x1 = -1; x2 = 1;
x = linspace(x1,x2,n);

y = exp(-2*x).*sin(3*x);

plot(x,y);
disp('voir tracé sur la figure');
```

Dans la console :

```
--> n = 50;
-->exec('monscript2.sce', -1)

 voir tracé sur la figure
```

Ici la variable **n** doit être définie au préalable.

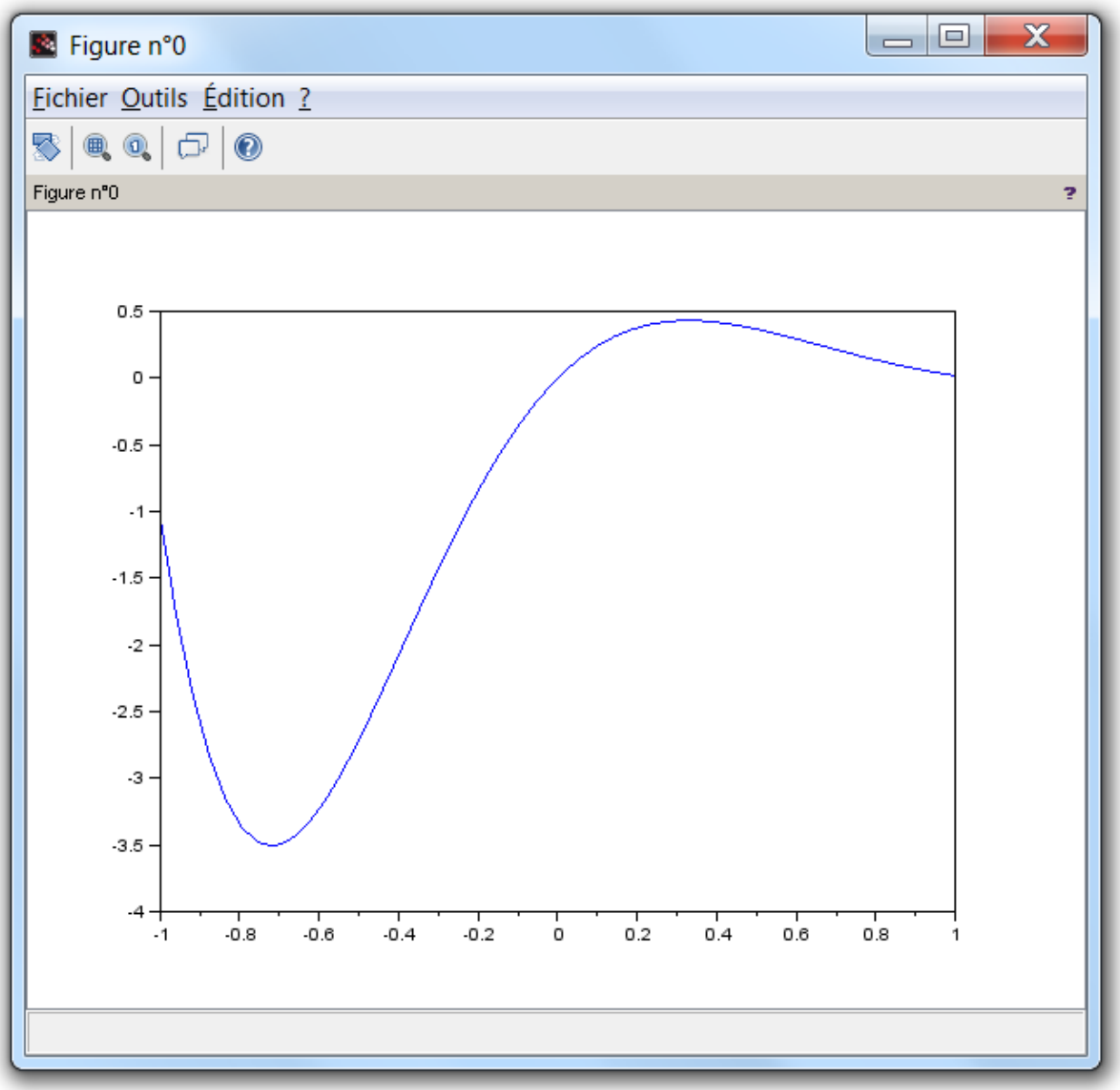


Figure n°0
Fichier Outils Édition ?
Figure n°0
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
-4
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

Les fonctions

L'utilisateur peut définir ses propres fonctions

- Comme pour les scripts, une fonction est définie dans un éditeur de texte tel que "*SciNotes*"
- Les fichiers ont pour extension "`.sci`".
- Les fonctions doivent être chargées dans Scilab (à l'aide de l'instruction `exec`) avant de pouvoir être utilisées.

Définition générale d'une fonction :

function [*out1*,*out2*,...] = mafonction(*in1*,*in2*,...)

corps de la fonction

endfunction

Exemple : racines d'une équation du 2^{nd} degré.

Dans un fichier "*new_functions.sci*" :

```
function [x1,x2] = racines_equ2d(a,b,c)
    // racines de ax^2 + bx + c = 0
    delta = b^2 - 4*a*c
    x1 = (-b - sqrt(delta))/(2*a)
    x2 = (-b + sqrt(delta))/(2*a)
endfunction
```

Dans la console :

```
--> exec('new_functions.sci', -1)
--> [r1,r2] = racines_equ2d(1,3,2)
 r2  =
  - 1.

 r1  =
  - 2.
```

Quelques remarques :

- Les variables définies dans la console sont visibles dans la fonction mais ne sont pas modifiables.
- Les variables définies dans la fonction ne sont pas visibles dans la console.

Boucles et branchements

Le langage de Scilab comprend les structures de contrôle classiques en algorithmie

La condition `if`

if *expression booléenne* then
instructions 1
else
instruction 2
end

```
if (x>=0) then
    disp("x est positif");
else
    disp("x est négatif");
end
```

Combinaison de plusieurs branchements suivant la valeur d'une variable

Le branchement `select`

select *variable*
case *valeur 1*
    instructions 1
case *valeur 2*
    instructions 2
else
    instruction 3
end

```
select i
case 1
    disp("One");
case 2
    disp("Two");
case 3
    disp("Three");
else
    disp("Autre");
end
```

Répétition d'une série d'instructions selon un compteur.

La boucle `for`

for $variable = début : pas : fin$

$instructions$

end

```
n = 10;
for k = 1:n
    y(k) = exp(k);
end
```

Répétition d'une série d'instructions tant qu'une expression booléenne est vraie.

La boucle `while`

while (*expression booléenne*)

instructions

end

```
x = 16;
while ( x > 1 )
    x = x/2;
end
```

Et aussi :

- L'instruction **break** interrompt une boucle et en sort.
- L'instruction **continue** interrompt une boucle et poursuit à la suivante.

Autant que possible, il est préférable de coder un calcul en utilisant les vecteurs / matrices. En effet dans Scilab, la *vectorisation* est 10 à 100 fois plus rapide qu'une boucle *for* ou *while*.

```
tic
S = 0;
for k = 1:1000
    S = S + k;
end
t = toc(); disp(t);

tic
N = [1:1000];
S = sum(N);
t = toc(); disp(t);
```

```
-->exec('monscript.sce', -1)

    0.029

    0.002
```

Sommaire

Exercice 1

Les équations paramétriques d'une ellipse centrée à l'origine sont :

$$\begin{cases} x(t) = A\cos t \\ y(t) = B\sin t \end{cases} \qquad \text{avec} \qquad 0 \leq t \leq 2\pi$$

1. Tracer ces équations dans le plan x-y pour $A = 2$ et $B = 1$.
2. Ajouter un titre et des labels sur les abscisses/ordonnées.

Solution

```
--> A = 2; B = 1;
--> t = [0:0.01:2*%pi];
--> x = A*cos(t);
--> y = B*sin(t);
--> plot(x,y)
--> xtitle('Une ellipse','x','y')
```

Exercice 2

Recherche de la racine d'une fonction par dichotomie. Soit une fonction continue strictement croissante sur $[a, b]$ telle que

$$f(a) < 0 \qquad \text{et} \qquad f(b) > 0$$

L'objectif est de trouver x_0 tel que $f(x_0) = 0$. Algorithme :

1. évaluer la fonction en $c = \frac{a+b}{2}$,
2. si $f(c) < 0$, l'intervalle de recherche devient $[c, b]$,
3. si $f(c) > 0$, l'intervalle de recherche devient $[a, c]$,
4. ce processus est ensuite réitéré...

A.N. : Déterminer la racine des fonctions

- $f(x) = 2x^4 + 2.3x^3 - 16x^2 - 8x - 17.5$ sur l'intervalle $[0, 100]$,
- $g(x) = \tan(x^2) - x$ sur l'intervalle $[0.5, \pi/3]$.

Solution

```
a = 0; b = 100;
precision = 0.0001;
ecart = 1;

while(ecart > precision)

    c = (a+b)/2;
    f = 2*c^4+2.3*c^3-16*c^2-8*c-17.5;

    if f<0
        a=c;
    elseif f>0
        b=c;
    else
        a=b;
    end

    ecart = b-a;
end

disp(c);
```

Résultats attendus :

- $f(x) = 0 \quad \rightarrow \quad x_0 = 2.7358$
- $g(x) = 0 \quad \rightarrow \quad x_0 = 0.83365$

Exercice 3

Soit un signal créneau $f(t)$ d'amplitude A, de période T et de valeur moyenne nulle. Sa décomposition en série de Fourier est donnée par :

$$f(t) = \sum_{n=1}^{+\infty} a_n \cos(n\omega t) \qquad \text{avec} \qquad \begin{cases} a_n = \dfrac{2A}{n\pi} \sin(n\dfrac{\pi}{2}) \\ \omega = \dfrac{2\pi}{T} \end{cases}$$

Représenter le signal $f(t)$ à partir de sa décomposition en série de Fourier pour différentes valeurs de n.

A.N. : On prendra $A = 2$ et $T = 0.5$. Echelle temporelle : $0 \leq t \leq 2$ avec un pas de 0.001.

Solution

```
A = 2 ; T = 0.5; n = 7; w = 2*pi/T;
t = [0:0.001:2];
f = 0;

for k=1:n
    a(k) = 2*A/(k*pi)*sin(k*pi/2);
    f = f + a(k)*cos(k*w*t);
end

plot(t,f);
```