



Université Pierre et Marie Curie
L.206
2007-2008
Charles Billon

Initiation Scilab I



Indroduction

Scilab est un logiciel de **calcul numérique** développé par l'ENPC et l'INRIA et distribué gratuitement par le consortium Scilab ¹.

Ce logiciel peut être installé sur différentes plate-formes : Microsoft, Unix, Linux, mac OS X, etc. Il peut être téléchargé sur le site www.scilab.org. Les fonctionnalités de **Scilab** sont très proches de celles de **matlab**, qui est un logiciel commercial.

L'intérêt de ce logiciel est de pouvoir disposer d'un outil à la fois souple et puissant pour résoudre numériquement de nombreux problèmes de mathématiques appliquées et de trouver presque expérimentalement des solutions à ces problèmes.

On notera enfin que la connaissance de **Scilab** est requise pour tout les candidats à l'agrégation de mathématiques dans la cadre d'une épreuve orale de modélisation. De nombreux cours et exercices sont disponibles sur la toile. Il commence, de plus, à être enseigné au lycée.

Ce logiciel scientifique, en développement constant, possède plusieurs versions dont la plus récente est la version 4.1.1.

*

¹ Comprenant, entre autres, les sociétés ou organismes : Anagram Technologies, Appedge, Axs Ingenierie, CEA, CNES, Cril Technology, Dassault Aviation, Ecole polytechnique, EADS, EDF, Esterel Technologies, IFP, INRIA, Klippel, PSA, Renault, Styrel Technologies, Thales, TNI, Scaleo Chip, Mandriva, ...

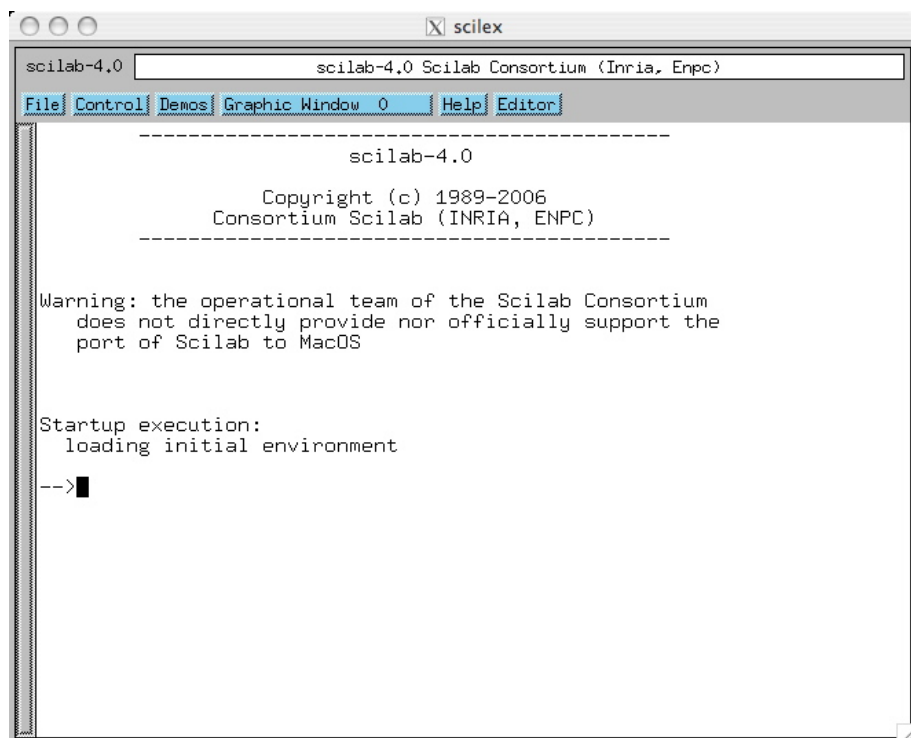
Ouverture d'une session Scilab

Une fois lancé, le logiciel **Scilab** affiche dans sa fenêtre principale la suite de caractères :

« --> █ » : c'est le "*l'invite*" de commande.

Cette invite signale que **Scilab** est en attente d'une commande de l'utilisateur.

Fenêtre initiale sur Mac OS X,



On remarque que cette fenêtre comporte une barre de menus qui peuvent être explorés à l'aide du pointeur lié à la souris. Les propriétés de ces différents menus seront déclinées dans la suite du cours.

- Le menu **File**.
- Le menu **Control**.
- Le menu **Demos**.
- Le menu **Graphic window**.
- Le menu **Help**.
- Le menu **Editor**

Utilisation de Scilab comme une calculette

Scilab peut être utilisé comme une calculette, en prenant en compte les règles habituelles de priorité relatives aux opérations mathématiques classiques :

$+$, $-$, $\times$ ($*$ pour Scilab), $/$, $^$ (pour puissance), associées aux parenthèses ()

⚡ Exemple



```
scilab-4.1
scilab-4.1 Scilab Consortium (Inria, Enpc)
File Control Demos Graphic Window 0 Help Editor
-->2+3
ans =
    5.
-->a=2+3
a =
    5.
-->a=3+0.5;b=2,c=3,d=a*b-c
b =
    2.
c =
    3.
d =
    4.
-->ans+1
ans =
    6.
-->ans*ans
ans =
    12.
-->
```

Mode d'emploi de la "calculette"

- La dernière ligne affichée dans la fenêtre principale de **Scilab** et qui débute par l'invite :

« --> ■ »,

est la **ligne courante** ou ligne de commandes ou encore ligne d'instructions.

- C'est sur cette ligne que l'on inscrit les instructions destinées au logiciel. Il est possible de se déplacer horizontalement sur cette ligne au moyen des flèches directionnelles du clavier, ◀ ou ▶, d'éliminer ou d'ajouter des caractères.

-
- La ligne courante peut comporter plusieurs commandes séparées par une **virgule** ou un **point-virgule**.
 - Le point-virgule « ; » est un inhibiteur d’affichage : l’instruction, une fois validée, est évaluée, son résultat est gardé en mémoire mais n’est pas affiché (sauf si la commande indique explicitement l’affichage).
 - La ligne d’instructions qui se termine par la commande « **Entrée** » est immédiatement exécutée.



- La commande « **Entrée** » est appelée suivant l’humeur « retour chariot », « Enter », « Exécute », etc.
- Il est possible, pour des suites d’instructions nécessitant plusieurs lignes d’écriture, d’indiquer que la commande en cours se poursuit sur la ligne suivante en utilisant **trois points** consécutifs en fin de ligne.

```
-->a=sin(0.1987654)* sin(0.1987654)+cos(0.1987654)...
-->*cos(0.1987654) ;
-->disp(a) // pour display
```

- Une ligne, ou un groupe de caractères précédé par « // » n’est pas lue par **Scilab**. Il peut s’agir alors d’un commentaire.
- Les résultats sont donnés à "epsilon machine" (%eps) près. Il s’agit d’un calcul numérique et non d’un calcul symbolique.
- On notera que **Scilab** est conçu pour travailler sur des nombres réels ou complexes.

```
--> sqrt(-4) // donnera pour réponse :
ans = 2.i
// puis,
-->sqrt(ans) // donnera la réponse (answer) :
ans = 1.+i // ans est une variable temporaire
```

- Avec cette "calculatrice", on dispose d’un grand nombre de fonctions prédéfinies.

On dispose par exemple des fonctions :

```
sqrt() pour racine carrée, définie sur  $\mathbb{C}$ 
exp() pour l’exponentielle
log( ) pour le logarithme népérien
sin( ) pour la fonction sinus etc.
```

- Dans **Scilab**, les calculs sont essentiellement numériques. Les nombres réels ou complexes sont en double précision, avec une précision maximale de 16 chiffres décimaux et leur valeurs absolues est comprise entre 10^{-308} et 10^{+308} .
- Par défaut, les résultats sont affichés avec 10 caractères, comprenant le point décimal et le signe.
- On notera que « log », comme beaucoup de logiciels de ce type, représente le logarithme Népérien « ln ». On utilise les fonctions « log2 » et « log10 » pour avoir respectivement les logarithmes en base 2 et en base 10.
- La fonction « **disp(a)** » affiche la chaîne de caractères représentant « a », sans l’affecter à la variable **ans** qui conserve alors son affectation courante.

Les types de données ou d'objets simples

Les commandes interprétées par **Scilab** peuvent être des expressions ou des instructions, construites à partir d'objets reconnus par le logiciel. Les objets ou données **Scilab** de base sont des matrices de différents types : scalaires, booléens, chaînes de caractères, polynômes, listes, etc.

- On dispose de divers « **types** » d'objets **Scilab** ; ceci sera développé dans la suite.

Objets	Réels ou complexes	Booléens	Chaînes de caractères	Polynômes	Fonctions	...
Type scilab	Constant	boolean	strings	polynomial	function	...

- Par ailleurs, on distingue deux formes de données : celles qui sont **prédéfinies** et celles qui sont définies par **affectation**.

- Les données numériques prédéfinies

Constantes mathématiques	π	e	$\epsilon_{machine}$	i
Instructions Scilab	%pi	%e	%eps	%i

- Les données booléennes prédéfinies

Constantes booléennes	Vrai	Faux
Instructions Scilab	%t	%f

Ces constantes booléennes permettent d'évaluer la valeur logique d'une relation.

- Les données ou variables scalaires définies par affectation

L'affectation est l'instruction qui consiste à donner une valeur à une variable que l'on crée ou que l'on modifie. Sa syntaxe est simple : on choisit un nom qui peut être une lettre ou un groupe de lettres et on lui **attribue** une valeur à l'aide du signe « = ».

➡ Exemple

```
Ma_donnee=sin(2*%pi) ; disp(a) ;
```

- Les variables "chaînes de caractères"

Une chaîne de caractères est définie par affectation sous la forme :

```
a=" Bonjour, le serveur est occupé, merci de rappeler plus tard."
```

```
b=" L"'an 1515 : c"'est Marignan. "
```

On affecte à « **a** », la valeur :

« Bonjour, le serveur est occupé, merci de rappeler plus tard. »,
en utilisant les deux séparateurs identiques, d'initialisation « " », puis de fermeture « " ».



Pour **Scilab**, le séparateur « " » a la même valeur que « ' ». En revanche, ces deux signes ont des valeurs différentes intégrées dans une chaîne de caractères.

Ainsi, pour distinguer les caractères « " » ou « ' », insérés dans une chaîne de caractères, d'un **séparateur** de fermeture, il faut les écrire : « "" » ou « '' » ou « ''' » ou « ''' ».

Ces chaînes de caractères sont de type **Scilab** strings.

On peut fusionner deux chaînes de caractères par l'opérateur de **concaténation** « + ».

Les opérations simples

• Opérations arithmétiques

Opérations mathématiques	Instructions Scilab
addition +	+
soustraction −	−
multiplication ×	* ou .*
division ÷	/ ou ./
puissance	^ ou .^

• Opérateurs relationnels

Relations mathématiques	Instructions Scilab
égalité =	==
différent ≠	< >
inférieur strict <	<
supérieur strict >	>
inférieur ou égal ≤	<=
supérieur ou égal ≥	>=

• Conversion chaîne ↔ scalaire

Conversion	Instructions Scilab
a (scalaire) → A (strings)	A=string(a)
A (strings) → a (scalaire)	A=0.11 ; a=evstr(A)

Les tableaux

Les tableaux ou matrices de données peuvent être définis par affectation sous la forme suivante :

```
-->a=[1,2,3,%pi;4,5,6,%e;7,8,9,10];
-->disp(a);
```

1.	2.	3.	3.1415927
4.	5.	6.	2.7182818
7.	8.	9.	10.

```
-->■
```

- Le symbole « [» initialise le tableau.

On définit alors des lignes de données de même taille :

- Les éléments successifs d'une ligne sont séparés par le symbole « virgule » : « , ».
- Le passage d'une ligne à une autre est indiqué par le symbole « point-virgule » : « ; ».
- Le symbole «] » clôt le tableau.
- « v=[] » est un tableau vide.

L'exemple précédent donne un tableau de scalaires. On construit de la même façon un tableau de chaîne de caractères.

```
-->A=["a","bb","ccc";"Bonjour","l'abeille","Au revoir"];
-->disp(A)
```

a	bb	ccc	!
!	!	!	!
Bonjour	l'abeille	Au revoir	!

```
-->■
```



- Un tableau doit comporter des données de même type. les lignes doivent comporter le même nombre d'éléments
- Lors de l'écriture d'une matrice, la virgule séparant deux éléments d'une même ligne peut être remplacé par un espace. Ceci est fortement déconseillé car source d'erreurs.
- Nous étudierons dans la suite de multiples autres façons de définir des tableaux, ainsi que les diverses opérations que **Scilab** propose.



Exercices

- Soient $a = 12$ et $b = 16$ les longueurs des cotés d'un angle droit d'un triangle rectangle. Calculer la longueur de son hypoténuse.

```
--> a=12 ; //On affecte la valeur 12 à la variable, a est mis en mémoire.  
--> b=16 ; // On affecte la valeur 16 à la variable b est mis en mémoire.  
--> sqrt( a^2+b^2) ; //La variable temporaire par défaut ans est affectée de la valeur  $\sqrt{a^2 + b^2}$ .  
--> disp(ans) // disp pour « display ». On affiche la valeur de la variable temporaire ans.  
--> c=sqrt( a^2+b^2) ; //La nouvelle variable c est affectée de la valeur  $\sqrt{a^2 + b^2}$ .
```

- Quelle est l'aire d'un disque de 12 cm de diamètre ?

```
--> r=12/2 ; A=%pi*r^2 ; disp(A) ;
```

- Soit $c=3+4\pi$; comment obtenir les parties réelle et imaginaire de c , son conjugué son module et son argument ?

- Calculer le logarithme népérien de 10.

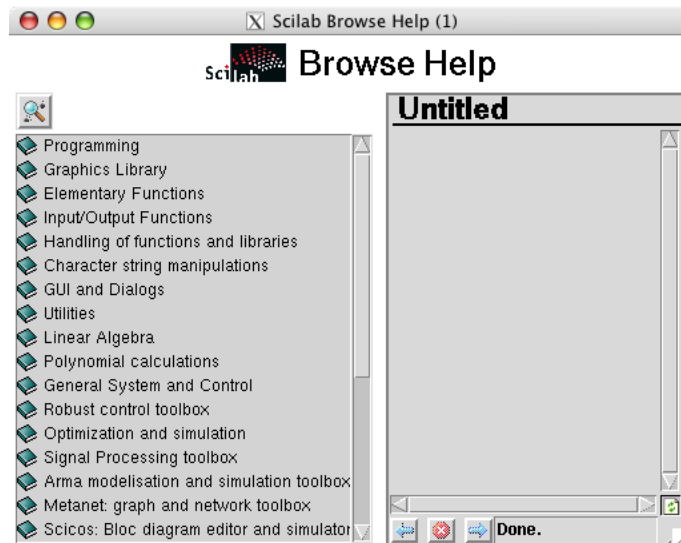
```
--> a=10 ; Reponse=log(10) ; disp(Reponse) ;
```

```
scilab-4.1 scilab-4.1 Scilab Consortium (Inria, Enpc)  
File Control Demos Graphic Window 0 Help Editor  
loading initial environment  
-->a=sqrt(-6); b=-%i;  
-->c=a*conj(a)  
c =  
6.  
-->d=a*a  
d =  
- 6.  
-->e=a^3  
e =  
- 14.696938i  
-->f=exp(b)  
f =  
0.5403023 - 0.8414710i  
-->g=cos(1)+%i*sin(1)  
g =  
0.5403023 + 0.8414710i  
-->h=real(f)+%i*imag(f)  
h =  
0.5403023 - 0.8414710i  
-->
```

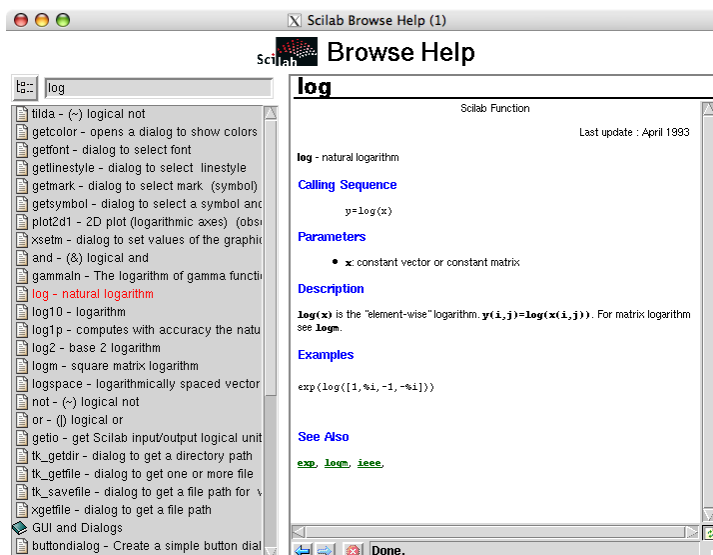
Le menu d'aide en ligne



La fenêtre principale **Scilab** propose divers menus. Le menu d'aide **Help** est particulièrement important. Ce menu donne de façon interactive (en français ou en anglais) les propriétés de chacune des instructions, symboles, fonctions, procédures **Scilab** prédéfinies.



On navigue dans les différents dossiers proposés, à la recherche d'une information adaptée. On peut également utiliser le moteur de recherche situé en haut à gauche de menu **Help** en entrant un mot clé supposé.



☺ Cette aide (en français ou en anglais) est évidemment essentielle et doit être utilisée sans modération.

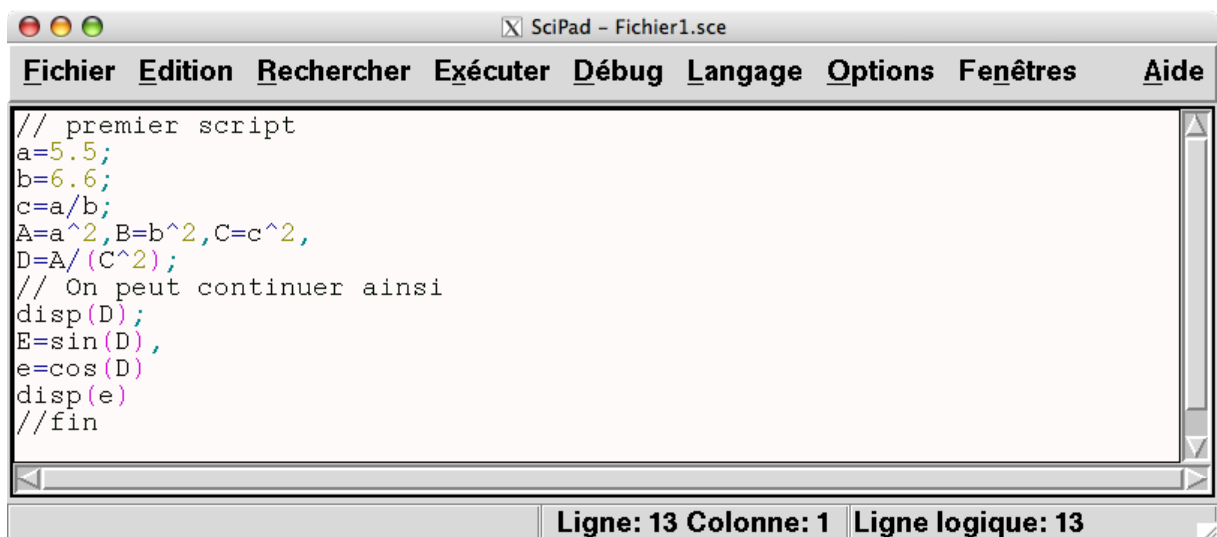
Editeur de texte

Le logiciel possède un éditeur de texte intégré, dénommé « **SciPad** » qui permet d'utiliser **Scilab** en :

mode différé

- Pour accéder à cet éditeur de texte on utilisera le menu « **Editeur** » de la fenêtre principale.
- On édite alors un **fichier source**, comportant des suites d'instructions, selon les règles du mode immédiat (attention aux valeurs des virgules, des point-virgules, des retours à la ligne, etc).
- Le fichier ainsi composé, **et terminé par un retour à la ligne**, constitue un **script** ou encore un programme.
- On valide (save) ce **script** en lui donnant un nom adapté avec l'extension **.sce**. On choisira un dossier de sauvegarde permettant de le retrouver facilement.

⚡ Exemple



The screenshot shows a window titled "SciPad - Fichier1.sce". The menu bar includes "Fichier", "Edition", "Rechercher", "Exécuter", "Débug", "Langage", "Options", "Fenêtres", and "Aide". The text area contains the following code:

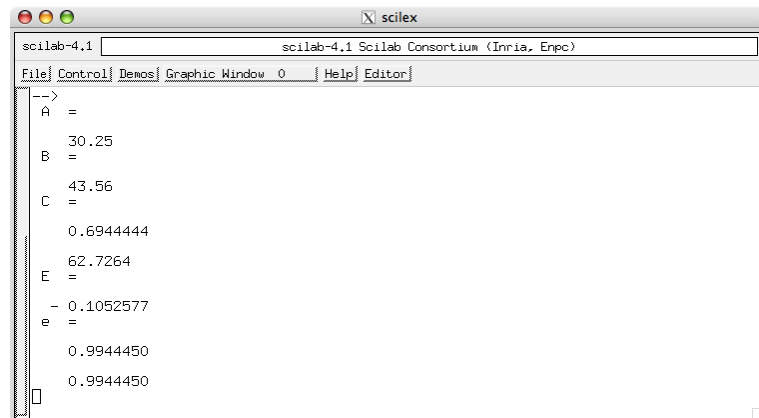
```
// premier script
a=5.5;
b=6.6;
c=a/b;
A=a^2,B=b^2,C=c^2,
D=A/(C^2);
// On peut continuer ainsi
disp(D);
E=sin(D),
e=cos(D)
disp(e)
//fin
```

The status bar at the bottom indicates "Ligne: 13 Colonne: 1" and "Ligne logique: 13".

Il est possible d'accéder d'anciens fichiers, « **AnciensFichier.sce** », de les modifier ou d'effectuer des tâches répétitives par « Copier/Coller », puis de sauvegarder le fichier, transformé, sous un autre nom, « **NouveauFichier.sce** », etc.

- On compile (load) alors le fichier sauvegardé dans **Scilab** en utilisant le menu « **Exécuter** » et le sous-menu « **Charger dans Scilab** ».

Cela donne alors dans la fenêtre **Scilab** le résultat suivant :

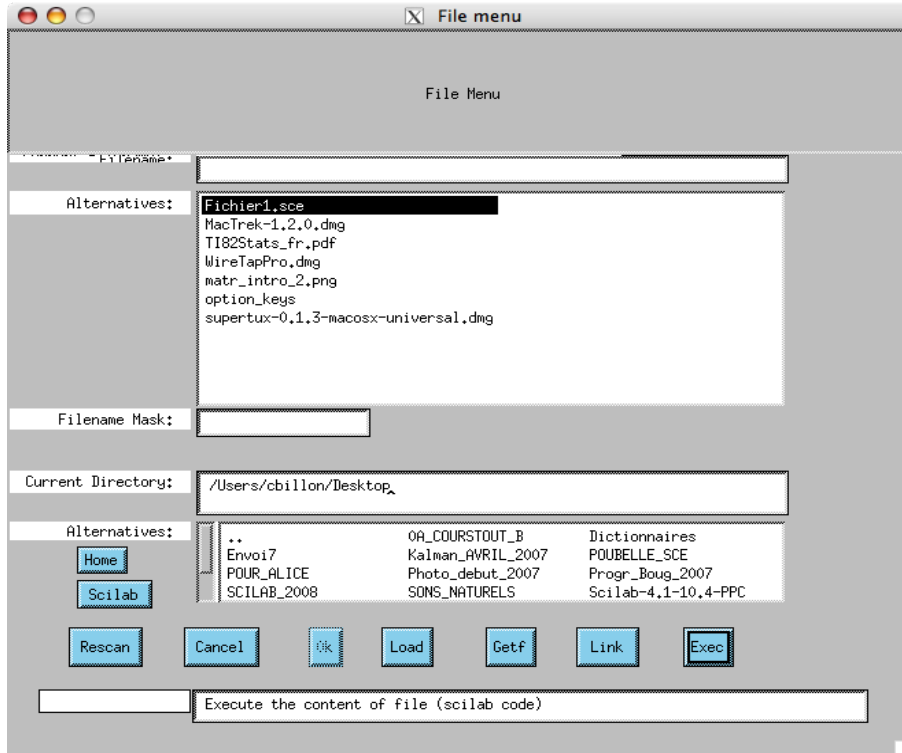


La compilation du script entraîne l'exécution immédiate des instructions et l'affichage des résultats.

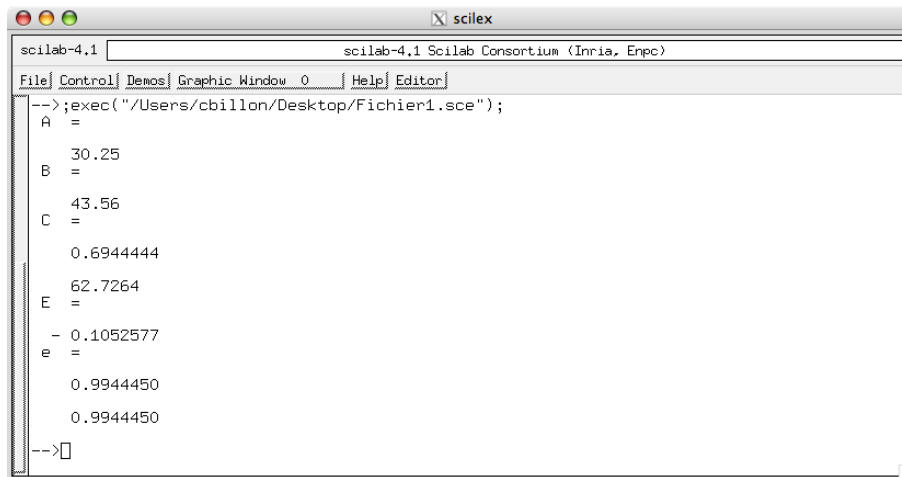


Remarque

Il est possible également charger le fichier `fichier1.sce` à partir du menu « **File** » de la fenêtre principale **Scilab**, puis en passant par le sous-menu « **File Operations** », choisir le chemin conduisant au fichier `fichier1.sce` ; Il suffit de valider le chargement de ce fichier par la commande « **Exec** ».



Cela donne alors le résultat suivant, dans la fenêtre principale :



```

scilab-4.1
scilab-4.1 Scilab Consortium (Inria, Enpc)
File Control Demos Graphic Window 0 Help Editor
--> exec("/Users/cbillon/Desktop/Fichier1.sce");
A =
    30.25
B =
    43.56
C =
    0.6944444
    62.7264
E =
    - 0.1052577
e =
    0.9944450
    0.9944450
-->

```

On remarquera que cette procédure affiche clairement, dans la fenêtre principale, le nom du fichier compilé et son chemin d'accès. On peut modifier ce fichier, le sauvegardé puis le rappelé, en utilisant les commandes de rappel ▲, ▼.

- Un script peut être composé sur n'importe quel autre éditeur de texte élémentaire, en fichier texte ascii **.sce**, indépendamment de l'éditeur de texte de **Scilab**.



On dispose donc de deux modes **Scilab** : le **mode immédiat**, ou en ligne, pour tester rapidement des instructions et le **mode différé** ou mode programmation qui permet d'élaborer des calculs plus longs ou plus élaborés.



Remarque à propos la rédaction d'un script. Pour **Scilab** le caractère « **espace** » possède une valeur qui lui est propre, par défaut, dépendant du contexte. On évitera d'utiliser ce caractère alors qu'il existe un caractère dédié équivalent. **Exemple** : définition de tableaux.

```

--> A=[1,2], B=[1 2], C=[12]
A =
    1.    2.
B =
    1.    2.
C =
    12.

```

Scilab, les boucles et les tests

La boucle itérative « for »

La boucle « for » est une instruction itérative de la forme :

<code>for</code>	<code>variable = valeurs</code>	<code>do</code>	<code>suite d'instructions</code>	<code>end</code>
------------------	---------------------------------	-----------------	-----------------------------------	------------------

- « *variable* » est la variable de boucle.
- « *suite d'instructions* » représente les commandes que l'on veut répéter.
- « *valeurs* » est l'ensemble des valeurs que doit décrire la variable de boucle.
- En introduction on définira l'ensemble des *valeurs* à l'aide de l'instruction suivante :

<code>début :</code>	<code>pas :</code>	<code>fin</code>
----------------------	--------------------	------------------

- ☞ « `début` » est la valeur initiale de la variable de boucle,
- ☞ « `fin` » est la valeur finale qui ne doit pas être dépasser.
- ☞ « `pas` » est le pas de progression de la variable de boucle.
- ☞ Lorsque le « `pas` » vaut +1, il peut être omis.

<code>début :</code>	<code>fin</code>
----------------------	------------------

➤ Exemple

```
// Calcul de factoriel 10. (10!)  
a=1;  
for i=1 :10 do  
  a=a*i;  
end;  
disp(a);  
disp(i);
```

- ☞ On remarquera que la variable « `a` » est **initialisée** avant le début de la boucle.
- ☞ La suite d'instructions se termine par « `;` » ; ceci évite l'affichage des valeurs intermédiaires de la variable `a`. avant le début de la boucle.
- ☞ Il est possible de remplacer « `do` » par « `then` », par « `,` », « `;` » ou par un passage à la ligne.

La boucle répétitive « while » (*tant que*)

La boucle « **while** » est une instruction répétitive de la forme :

while	<i>condition</i>	do	<i>suite d'instructions</i>	end
--------------	------------------	-----------	-----------------------------	------------

Ce que l'on peut traduire par :

« **tant que** la *condition* reste vraie, **faire** la *suite d'instructions donnée* **fin de tant que** »

- « *condition* » fournit un résultat booléen.
- « *suite d'instructions* » consiste en une ou plusieurs instructions (se terminant par une virgule ou un point-virgule avec le sens habituel).

Exemple

On veut calculer $10!$ en utilisant la récurrence : $0! = 1$ et, pour $n \geq 1$, $n! = n \times (n - 1)!$.

<pre>b=1; a=1; while (a<11) do b=b*a; a=a+1; end; disp(a);</pre>	<pre>// Calcul de factoriel 10 avec la boucle while // initialisation de la boucle // tant que a est <11, faire : // b est remplacé par la valeur en mémoire de b fois a, // a est remplacé par la valeur en mémoire de a plus 1 // fin de la boucle</pre>
---	---

Autre possibilité en utilisant le mot-clé **else** :

```
b=n; a=1;
while (n>0) do
    a=n*a;
    n=n-1;
else
    disp(a);
end;
```

■ On remarque que les boucles doivent être initialisées par des constantes ou variables judicieusement choisies.

Le test conditionnel « if ... then ... else »



Test bidirectionnel

if	<i>condition vraie</i>	then	<i>suite d'instructions 1</i>	
	else			<i>suite d'instructions 2</i>
end				

⚡ Exemple

```
scilab-4.1  scilab-4.1 Scilab Consortium (Inria, Enpc)
File Control Demos Graphic Window 0 Help Editor
-->A=rand();
-->if (A<0.5) then
-->B=0;
-->else
-->B=1;
-->end
-->disp(A);disp(B);
    0.2113249
    0.
-->
```

- A est un nombre aléatoire compris entre 0 et 1. (ici 0,2113249)
- Si $A < 0.5$, alors on affecte à B la valeur 0,
- sinon, on donne à B la valeur 1.
- fin du test.
- Ici B vaut donc 0.

Même chose en mode différé. Cela donne l'écriture du script suivant :

```
A=rand();
if (A<0.5) then
    B=0;
else
    B=1;
end;
disp(A);disp(B);
```

Remarque. On peut omettre **then** et le remplacer par un point-virgule. Cela donne :

```
A=rand(); if (A<0.5); B=0; else B=1; end; disp(B);
```



Test polydirectionnel

if	<i>condition 1 vraie</i>	then	<i>suite d'instructions 1</i>	
	elseif	<i>condition 2 vraie</i>	then	<i>suite d'instructions 2</i>
	elseif	<i>condition 3 vraie</i>	then	<i>suite d'instructions 3</i>
else			<i>suite d'instructions 4</i>	
end				

⚡ Exemple

```

scilab-4.1
scilab-4.1 Scilab Consortium (Inria, Enpc)
File Control Demos Graphic Window 0 Help Editor
-->A=rand();
-->if (A<0.1) then
-->B="bonjour";
-->elseif (A>0.9) then
-->B="au revoir";
-->elseif(A>0.4) then
-->B="bye";
-->else
-->B="good morning";
-->end;
-->disp(A);disp(B);
0.7560439
bye
-->

```

Cela peut être écrit sur un script de la façon suivante :

```

A=rand() ;
if (A<0.1) then
    B="bonjour" ;
elseif (A>0.9) then
    B="au revoir" ;
elseif(A>0.4) then
    B="bye" ;
else
    B="good morning" ;
end ;
disp(A) ;disp(B) ;

```



Test de branchement

select		<i>expression 0</i>		
	case	<i>expression 1</i>	then	<i>suite d'instructions 1</i>
	case	<i>expression 2</i>	then	<i>suite d'instructions 2</i>
	case	<i>expression 3</i>	then	<i>suite d'instructions 3</i>
	else			<i>instructions par défaut</i>
end				

- La valeur de « *expression 0* » est donnée puis comparée aux valeurs de « *expression 1* », « *expression 2* »...
- Dès que l'une de ces comparaisons rend la valeur booléenne **Vraie**, la suite d'instructions correspondante est effectuée ; les autres ne sont pas testées.
- Si toutes les comparaisons proposées donnent **Faux** la suite d'instructions par défaut, introduite par **else**, est effectuée.

➤ Exemple

Dans un fichier `Salutation.sce` on peut écrire :

```
A=floor(4*rand());
select A
    case 0 then
        disp( "BONJOUR" );
    case 1 then
        disp( "AU REVOIR" );
    case 2 then
        disp( "ni hao" );
    else
        disp( "zai jian!" );
end ;
```

Fonctions ou macros

Les fonctions ou macros sont des enchaînements de commandes qui peuvent être individualisées par un nom global singulier, à l'intérieur d'un script et utilisées, chacune une fois définie, comme une nouvelle instruction.

- La syntaxe utilisée est :

```
function y=MaCommande( paramêtres );  
    instructions;  
    instructions;  
    y=...  
endfunction
```

Exemple

- On veut construire une commande qui calcule le prix toutes taxes comprises (ttc) d'un ordinateur, connaissant son prix hors-taxa (ht).

```
function y=PrixTTC(x);  
    y=1.195*x;  
endfunction  
//application  
MonOrdiHT=1520; // Entrée  
a= PrixTTC(MonOrdiHT); // Sortie  
disp(a)
```

Exemple

- On veut construire une commande qui inverse l'ordre des éléments d'une chaîne de caractères.

```
function y=Renversement(x);  
    X=ascii(x)  
    Y=X($ :-1 :1)  
    y=ascii(Y)  
endfunction;  
// Utilisation de la fonction  
a="engagelejeuquejelegagne"; // Chaîne de caractères (entrée)  
b=Renversement(a); // sortie  
disp(b)
```

On développera, dans la suite de cette introduction, la notion de fonction.

Scilab et applications graphiques élémentaires



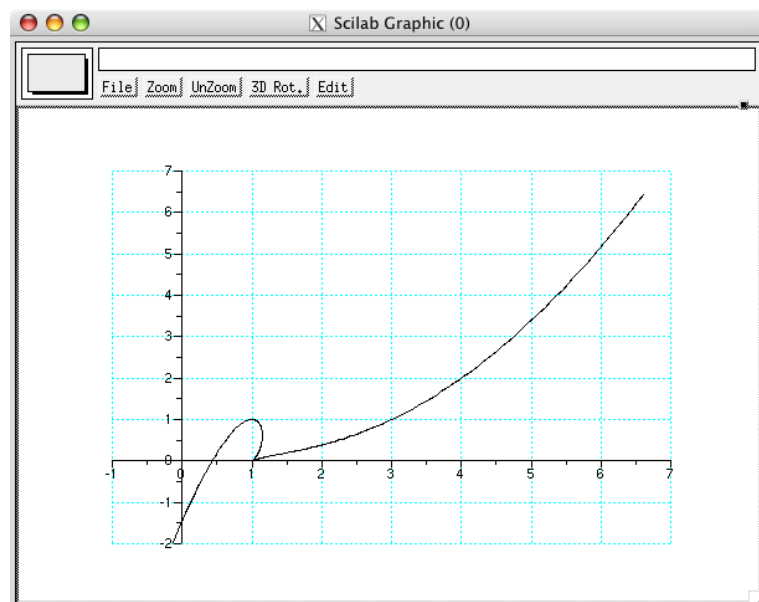
On veut représenter par exemple le graphe de la fonction, définie dans un repère rectangle, par la paramétrisation suivante :

$$x(t) = 1 + t^2 + t^3; \quad y(t) = t^2 + t^3 - t^5, \text{ pour } t \in [-1.5, +1.5]$$

Cela donne, par exemple, le script suivant :

```
t=-1.5 :0.01 :1.5;  
X=1+t.^2-t.^3;  
Y=t.^2+t.^3-t.^5;  
clf  
plot2d(X,Y);
```

On obtient la figure suivante :



- On a défini t comme un tableau de valeurs allant de -1.5 à $+1.5$ avec un pas de $0,01$.
- On calcule les valeurs $X(t)$ et $Y(t)$, pour chaque point du tableau t , ce qui donne les deux tableaux X pour les abscisses et Y pour les ordonnées.

■ La commande `plot2d(X,Y)` trace dans une figure l'ensemble des points $(X(t), Y(t))$ dans un repère rectangulaire adapté.

Le logiciel **Scilab** possède de nombreuses commandes permettant de tracer des courbes. On a ainsi la possibilité de visualiser immédiatement et simplement des résultats numériques. On donne, dans ce qui suit, un aperçu élémentaire de quelques commandes graphiques avec en particulier la commande `plot2d`.

➤ Exemple

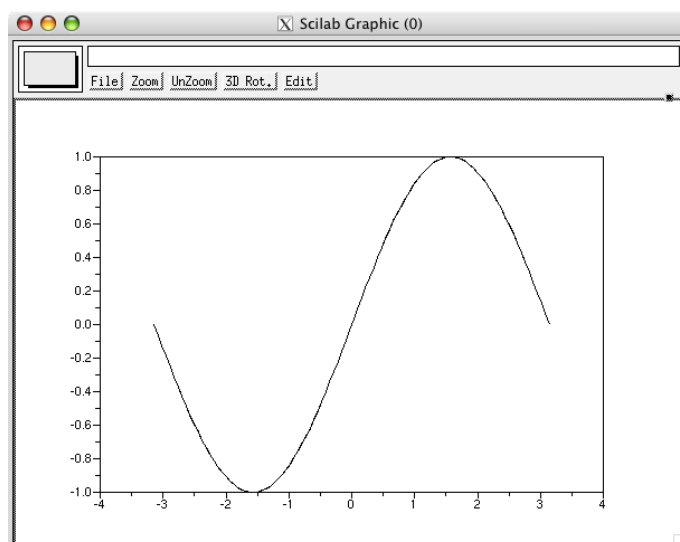
Soit à représenter la fonction $x \mapsto \sin(x)$ sur une période, dans un repère rectangulaire.

- On définit d'abord **en abscisses**, 100 valeurs $(x_i; 1 \leq i \leq 100)$ régulièrement espacées entre $-\pi$ et $+\pi$.
- On calcule ensuite les valeurs **en ordonnées** : pour $i = 1 \dots, 100$, $(y_i = f(x_i) = \sin(x_i))$.
- Le graphe de cette fonction sera tracé dans une fenêtre graphique par la commande : `plot2d(x,y)`

Ceci donne script suivant :

```
x= linspace(-%pi,%pi,100); // suite de 100 points équirépartis entre  $-\pi$  et  $+\pi$ .
y=sin(x);
plot2d(x,y);
```

On compile le script **Scilab** qui donne figure peu satisfaisante suivante.



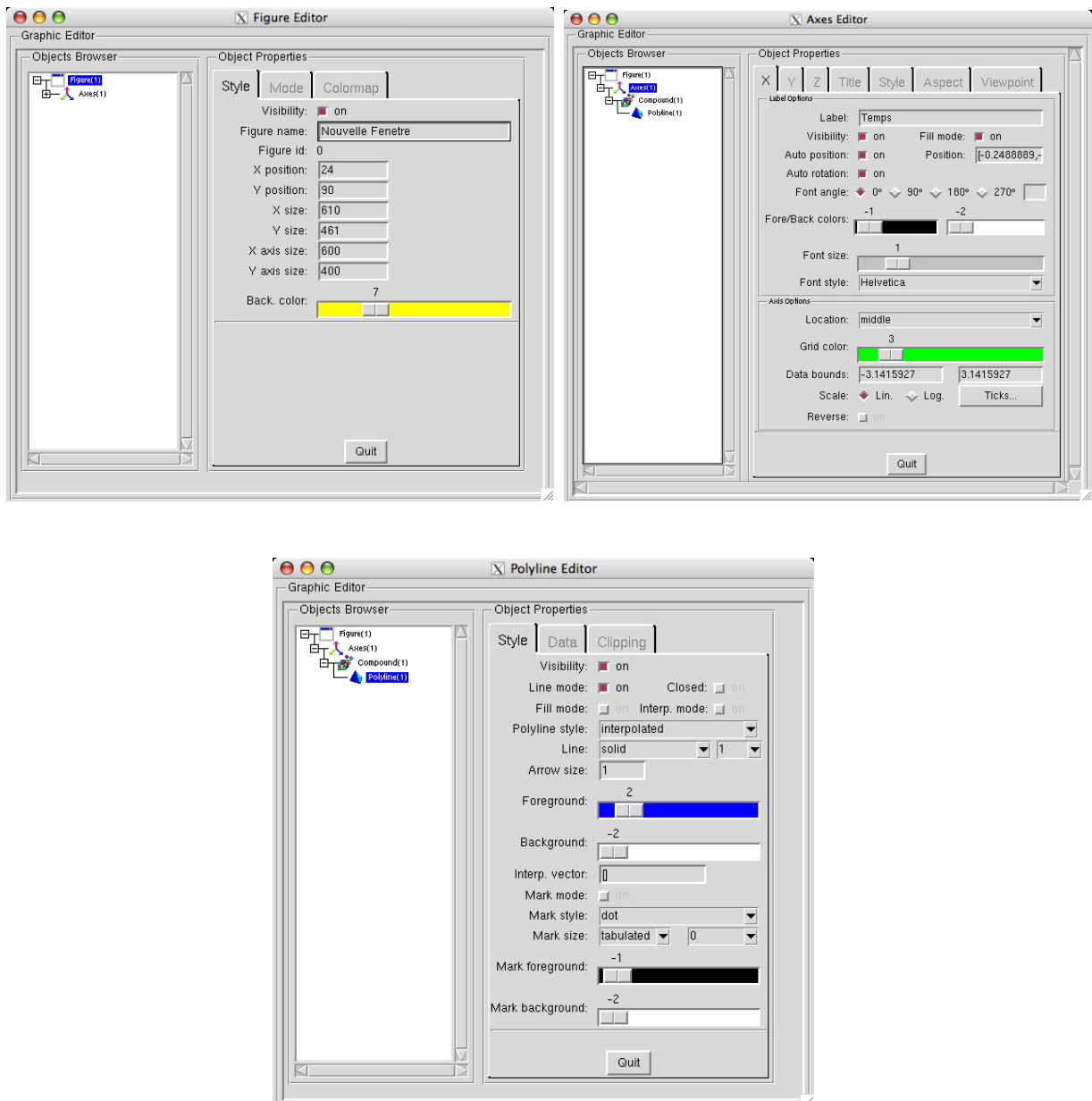
Ce graphe élémentaire peut être modifié dans un deuxième temps, en utilisant l'éditeur graphique.



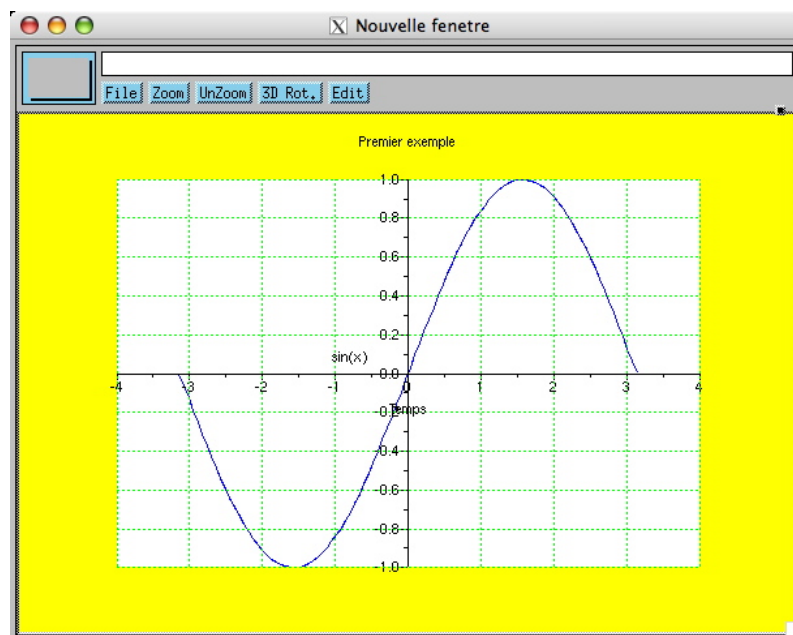
Modification interactive d'une figure graphique

Le présentation d'une figure peut être modifiée de façon interactive. On sélectionne, dans la fenêtre Scilab `Graphic(0)` qui apparaît, le menu « **Edit** » et le sous-menu « **Propriétés de la figure** ».

Exemple de modification



On obtient finalement la nouvelle représentation :



On peut tracer deux courbes sur la même figure par surcharge :

```
x= linspace(-%pi,%pi,100) ;
y=sin(x) ;
z=cos(x) ;
plot2d(x,y) ;
plot2d(x,z) ;
```

Il existe une autre possibilité de représenter deux courbes sur une même figure.

```
clf; // efface la figure courante
x=linspace(-%pi,%pi,100)'; // vecteur colonne
plot2d(x,[sin(x),cos(x)]);
```

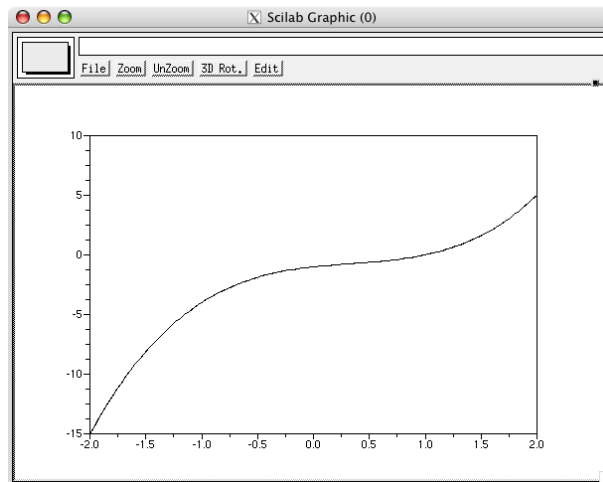
➡ Exemple

Représentation de la fonction $x \mapsto f(x) = x^3 - x^2 + x - 1$, sur un repère rectangulaire, $x \in [-2 + 2]$.

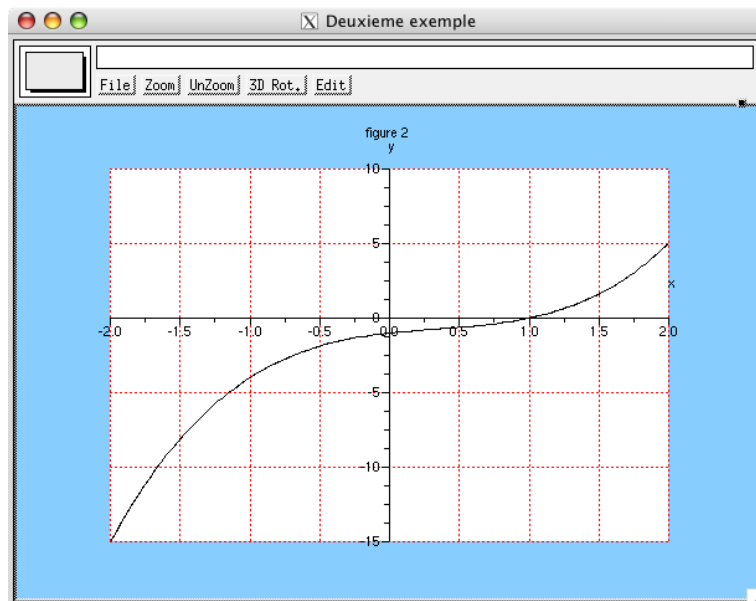
On écrit alors le script suivant :

```
clf(); //on efface la figure précédente
x= linspace(-2,+2,100) ;
y=x.^3-x.^2+x-1 ;
plot2d(x,y) ;
```

Cela donne la figure :



Après une modification à partir le l'éditeur graphique on obtient :



On remarque que la barre de menu comporte les menus complémentaires « Zoom » et « UnZoom ».

- Le menu « Zoom » sélectionné permet de déterminer sur la fenêtre graphique, à l'aide du pointeur, une zone rectangulaire à agrandir. Une nouvelle fenêtre apparaît. on peut itérer cette opération et ainsi observer le comportement d'une courbe dans une zone précise.

« UnZoom » permet un retour au graphe initial.

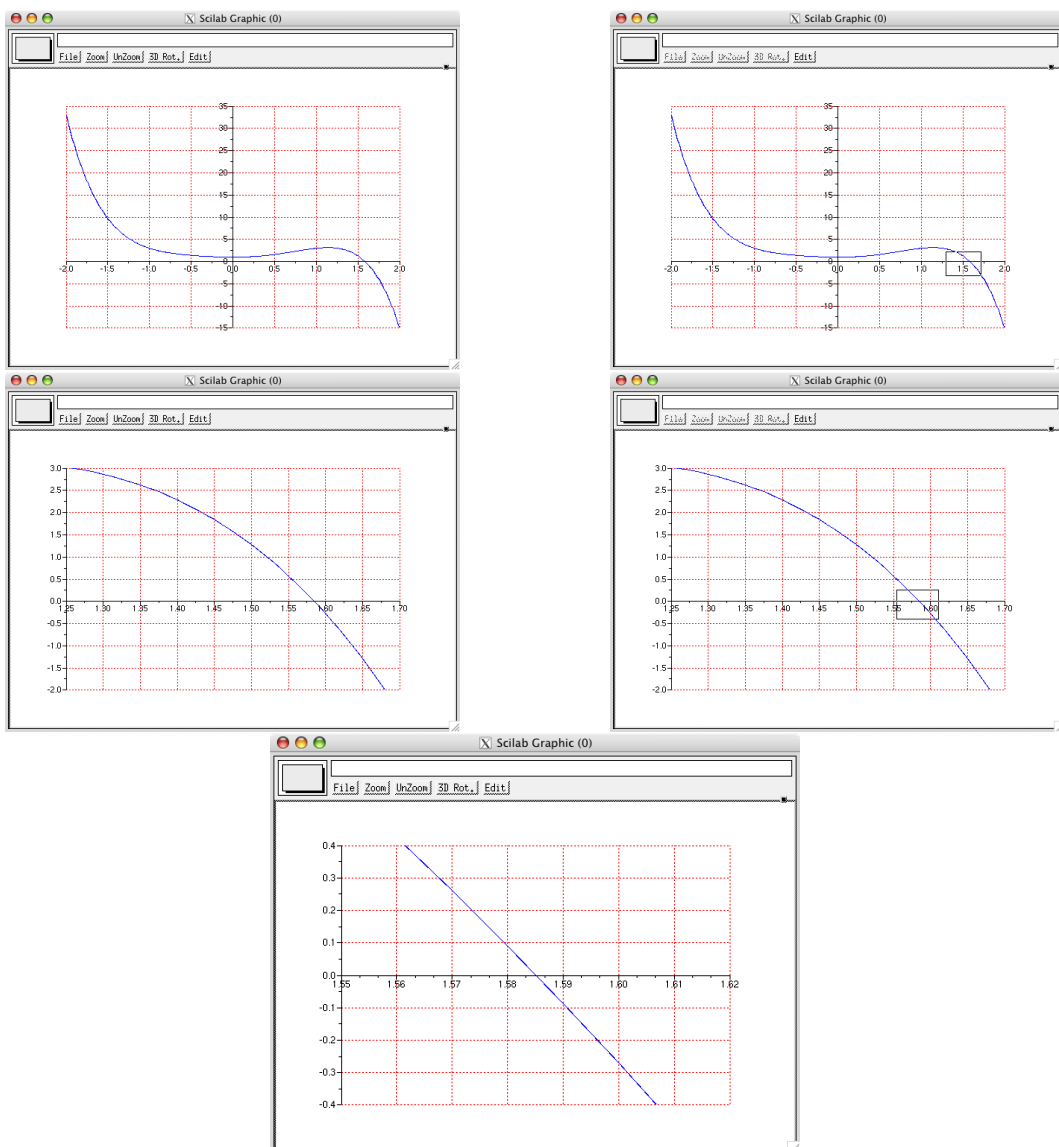
➤ Exemple

Quelles sont les racines de l'équation $1 + 2x^2 + x^3 - x^5 = 0$?

On écrit le script expérimental suivant :

```
x=-2 :0.01 :2 ;  
y=1+2*x.^2+x.^3-x.^5 ;  
  
clf()  
plot2d(x,y,2) ;
```

Après modification classique de la figure on obtient successivement, en utilisant la loupe ou le zoom :



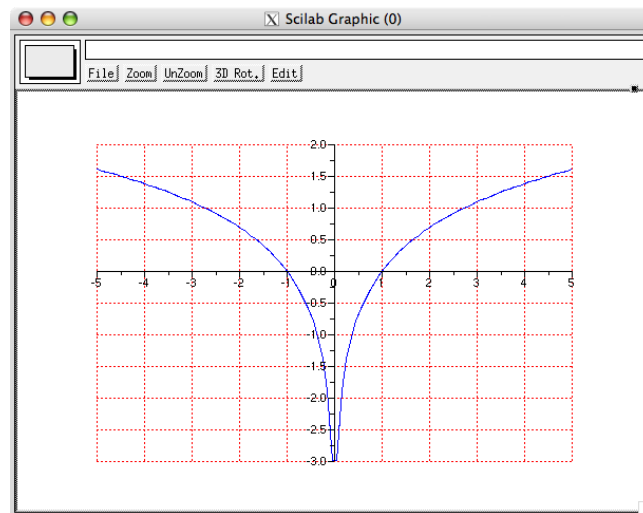
On peut conclure qu'il existe une racine peu différente de 1,185.



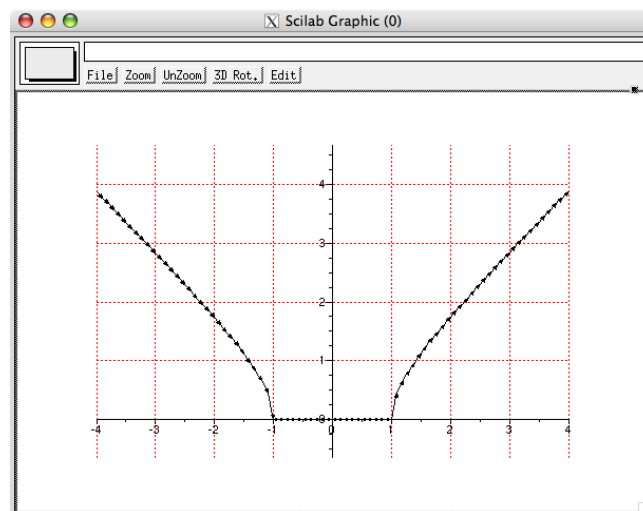
Les arguments de la commande `plot2d` sont toujours considérés comme réels. Ainsi si l'on donne des nombres complexes comme argument, `plot2d` utilise leurs parties réelles sans donner de message d'erreur. Par exemple, si l'on trace la fonction $y = \ln(x)$ sur $[-5, +5]$, en utilisant la suite de commandes :

```
x=linspace(-5,5,100) ;  
plot2d(x,log(x)) ;
```

La fonction `plot2d` tracera, entre -5 et 0 , la partie réelle du logarithme (qui est complexe pour ces valeurs).



■ Autre exemple : tracé du graphe de $y = \sqrt{x^2 - 1}$ sur $[-4, +4]$, à l'aide de `plot2d`.



Tracés de plusieurs cellules graphiques dans une même fenêtre

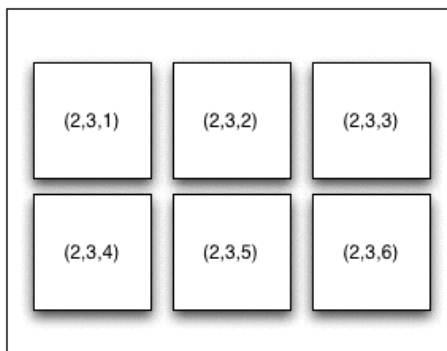
L'instruction `subplot` permet de partager la fenêtre graphique en un tableau de cellules graphiques.

⚡ Exemple

On veut représenter, par exemple, sur une même fenêtre graphique 6 courbes chacune dans une cellule propre dans un tableau de deux et trois colonnes. On utilise la procédure `subplot`.

`subplot( 2 (lignes), 3 (colonnes), i (numéro de la cellule))`

On notera, dans la figure suivante, le principe de numérotation particulier des cellules : de droite à gauche et de haut en bas.



⚡ Exemple

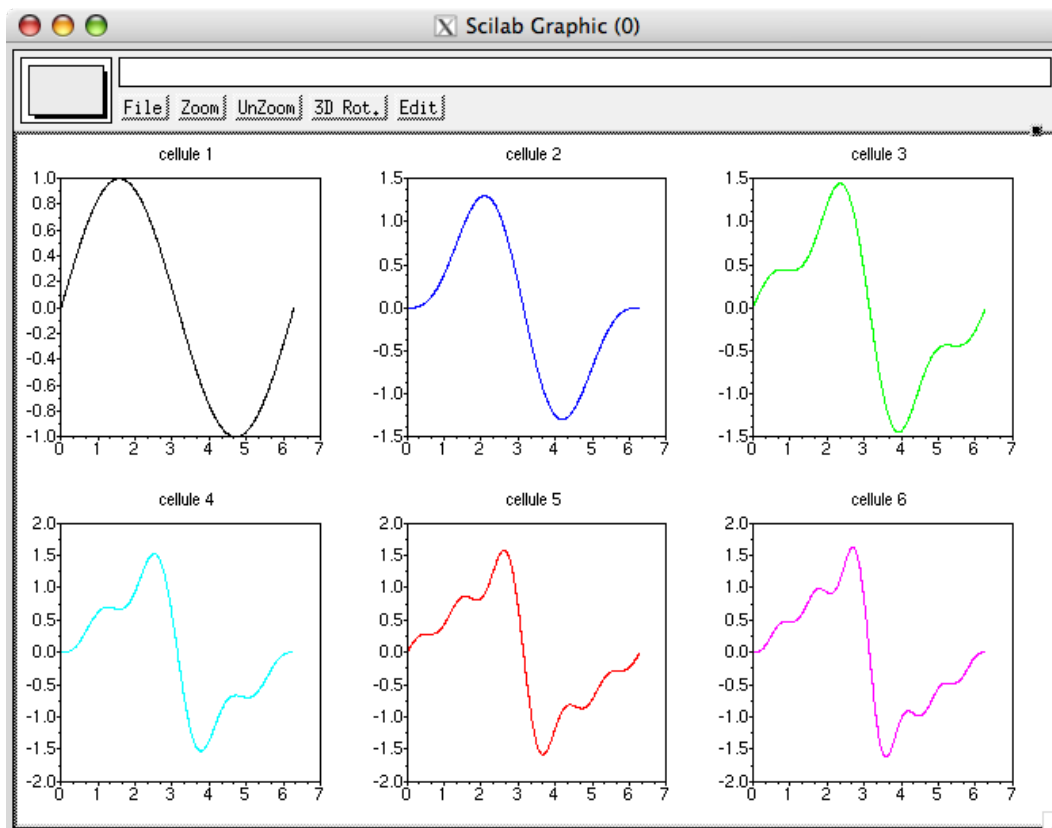
On considère le script suivant qui calcule :

$$S_n(x) = \sum_{k=1}^n (-1)^{k+1} \frac{1}{k} \sin(kx) \quad , \text{ pour } n = 1, 2, \dots, 6,$$

et représente les résultats dans des cellules distinctes.

```
x=0 :0.01 :2*%pi ;
y1=sin(x) ;
y2=y1-(1/2)*sin(2*x) ;
y3=y2+(1/3)*sin(3*x) ;
y4=y3-(1/4)*sin(4*x) ;
y5=y4+(1/5)*sin(5*x) ;
y6=y5-(1/6)*sin(6*x) ;
Y=[y1;y2;y3;y4;y5;y6] ;
clf()
for i=1 :6 do
    subplot(2,3,i) ;
    plot2d(x,Y(i, :),i) ;
    xtitle("cellule "+string(i))
end
```

On obtient la figure suivante :



On remarquera à nouveau la numérotation spécifique associée à la procédure `subplot`.

Variantes de la fonction `plot2d()`

La fonction `plot2d()` possédait, dans l'ancien mode graphique, plusieurs variantes : `plot2d2()`, `plot2d3()` et `plot2d4()`. On consultera l'aide pour plus de précisions. Avec l'éditeur graphique du nouveau mode graphique `plot2d()` fournit beaucoup plus d'alternatives de représentations graphiques. En voici pour-tant une exemple.

➤ Exemple

```
x=0 :0.5 :10 ; clf ;
subplot(2,2,1) ; plot2d(x,sin(x)) ; xtitle("plot2d()") ;
subplot(2,2,2) ; plot2d2(x,sin(x)) ; xtitle("plot2d2()") ;
subplot(2,2,3) ; plot2d3(x,sin(x)) ; xtitle("plot2d3()") ;
subplot(2,2,4) ; plot2d4(x,sin(x)) ; xtitle("plot2d4()") ;
```

Caractéristiques du nouveau mode graphique Scilab

Tracés par « `plot2d()` » et éditeur graphique

Les tracés de courbes sont réalisés dans des « **figures** », formés d'objets graphiques, comme les axes, les tracés (appelé « **Polylines** » pour les courbes), les labels, etc.

Ces objets ou parties de la figure sont appelées « **entités** » et possède chacune des propriétés que l'on peut modifier.

On a vu qu'il est possible d'utiliser l'éditeur « **Edit** » de la fenêtre graphique.

On peut utiliser le sous-menu « **Démarrer sélecteur d'entité** » (start entity picker) et cliquer avec la souris sur la partie de la figure que l'on veut modifier.

Il est également possible de modifier les paramètres d'une figure à l'aide d'instructions **Scilab** insérées dans le script.

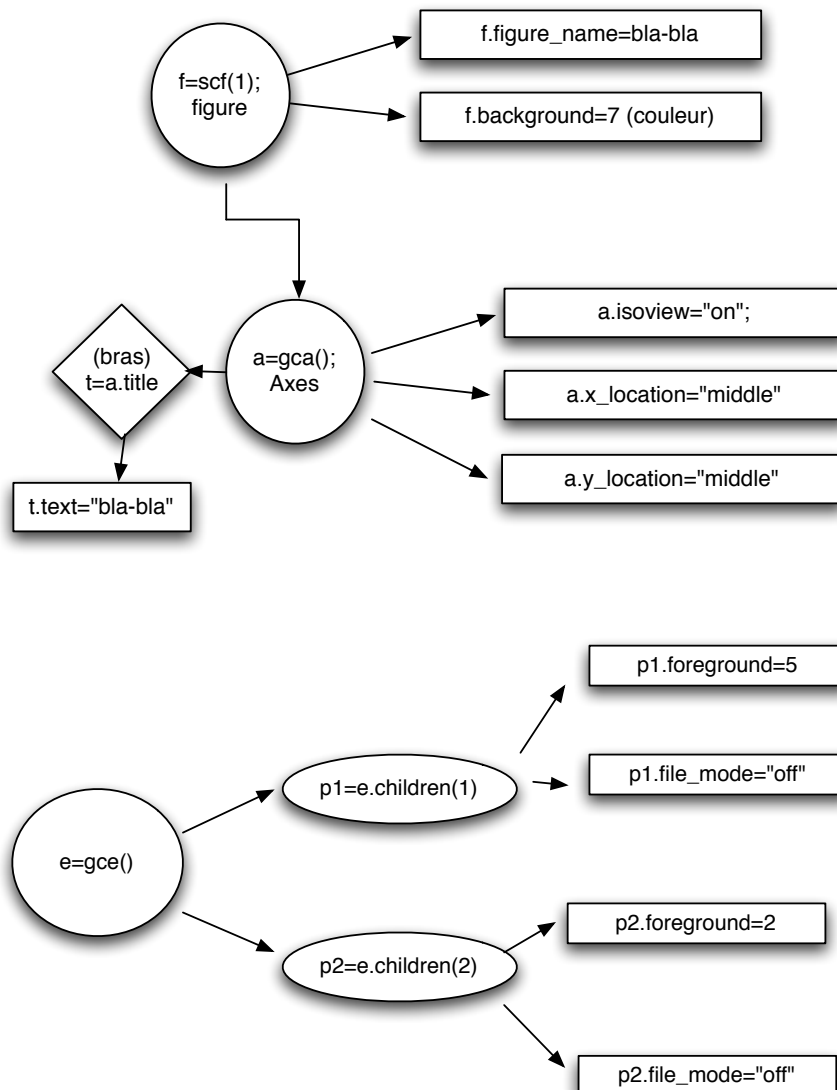
- `scf()`. Pour les propriétés générales de la figure.
- `gca()`. Pour les propriétés des axes, du titre, etc.
- `gce()`. Pour les propriétés des tracés (compound).

Exemple

```
f=scf(1);
clf(1);
x=(-7 :0.01 :7)';
clf;plot2d(x,[sin(x),cos(x)]);
a=gca() // get current axes
e=gce() // get current entity.
```

La figure suivante présente de façon simplifiée l'arbre correspondant aux tracés effectués. Chaque tracé est composé d'objets appelés « **entités** » qui ont des propriétés et des liens hiérarchiques entre eux.

On peut modifier de façon interactive chaque élément de la figure : axes, tracés, couleurs, titres, légendes, etc. Ces modifications peuvent être inscrites dans le corps d'un script en utilisant la hiérarchie spécifique de chacune des composantes.



Hiérarchie du tracé

- La commande `a=gca()` (« **Get Current Axes** ») permet d'accéder à l'entité « **axes** » courant, c'est-à-dire aux axes de la figure que l'on vient de tracer.
- La commande `e=gce()` (« **Get Current Entity** ») permet d'accéder à la dernière entité tracée, c'est-à-dire à l'entité « **Compound** », qui est l'ensemble des deux courbes tracées, appelées « **Polyline** ».
- On passe d'une entité à ses enfants en utilisant le champ « **children** ».

Ces différents champs sont accessibles par les commandes suivantes :

```

a=gca()
e=gce()
p1=e.children(1)
p2=e.children(2)

```

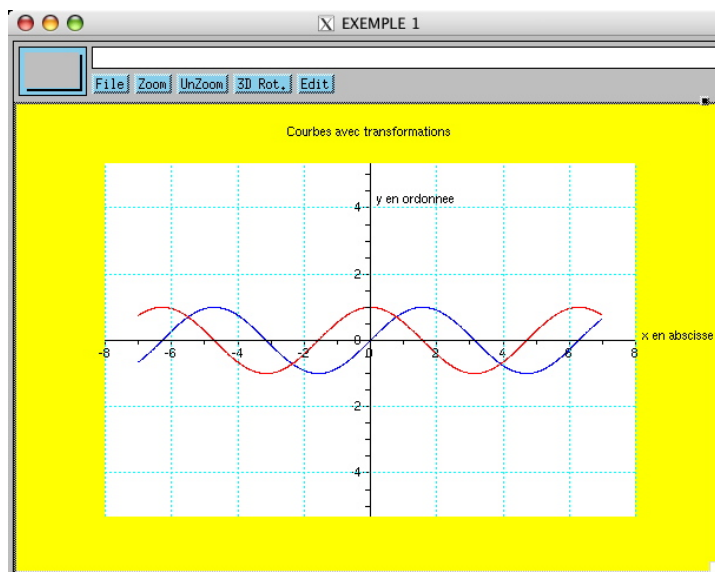
On peut alors modifier les propriétés de axes ou des courbes par des instructions insérées dans le script.

➡ Exemple

```

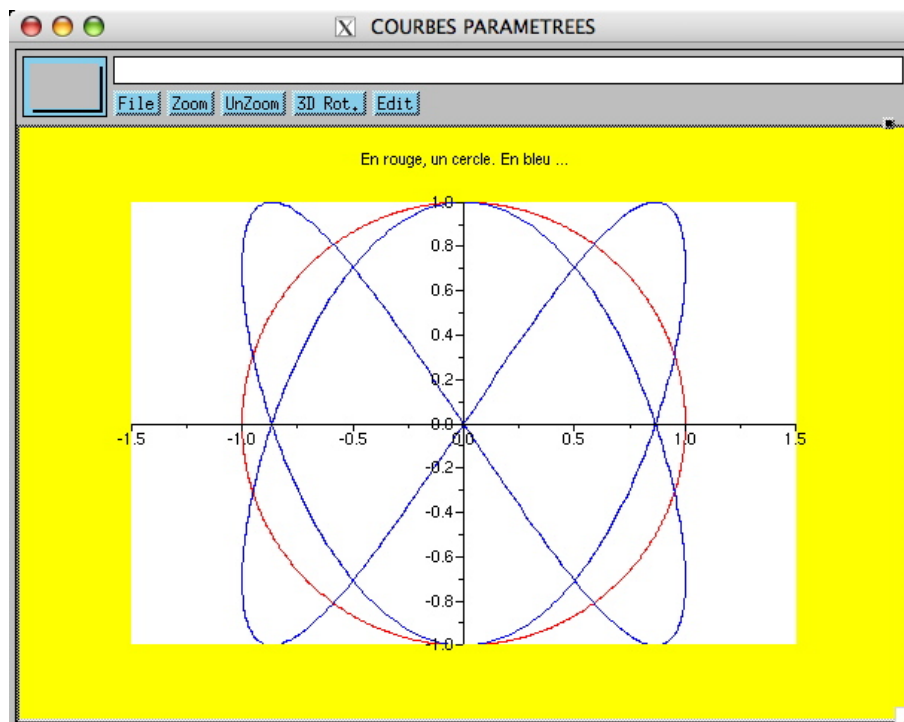
f=scf(1); // On ouvre la figure numéro 1
f.figure_name="EXEMPLE 1"; // Nom de la fenêtre graphique
f.background=7; // Couleur de fond (externe) jaune
x=(-7 :0.01 :7)'; clf;plot2d(x,[sin(x),cos(x)]);
a=gca();
e=gce();
p1=e.children(1);
p2=e.children(2);
p1.foreground=5; // la couleur du graphe (1) est rouge
p2.foreground=2; // la couleur du graphe (2) est bleue
a.x_location="middle"; // Propriétés des Axes
a.y_location="middle";
a.isoview="on"; // axes orthonormés
a.grid=[4,4]; //grilles de couleur 4 et 4 (vert)
t=a.title; // bras titre de Axes
t.text=" Figure et courbes apres transformations ";
LX=a.x_label; // Handle ou bras de Axes des X
LX.text= " x en abscisse";
LX.position=[8,0];
LY=a.y_label; // Handle ou bras de Axes des Y
LY.text= " y en ordonnee";
LY.position=[0,4];
LY.font_angle=0;

```



➤ Exemple d'une courbe paramétrée

```
t=0 :0.01 :2*%pi ;
x=sin(t) ;
y=cos(t) ;
f=scf(1);clf(1) ;
f.figure_name=" COURBES PARAMETREES" ;
f.background=7 ;
plot2d(x,y,5) ;
a=gca() ;
a.isoview="on" ;
a.x_location="middle" ; a.y_location="middle" ;
plot2d(sin(2*t),cos(3*t),2)
t=a.title; t.text=" En rouge, un cercle. En bleu ..."; f=scf(1) ;
f.figure_name=" COURBES PARAMETREES" ;
```



- On remarquera que les commandes déclinées dans un script ne sont valables que pour la figure courante.
- Si l'on veut modifier le comportement par défaut du graphisme, la fonction **gda** « get default axes » doit être utilisée, car elle retourne les axes par défaut.

Par exemple si l'on veut des axes orthonormés et que les axes se coupent en (0,0), par défaut, à l'ouverture d'une session, on écrit, en début de session **Scilab**, les commandes :

```
A=gda() ;  
A.x_location="middle" ;  
A.y_location="middle" ;  
A.isoview="on" ;
```

Pour que **Scilab** exécute toujours, par défaut, ces commandes à son lancement, il faut les mettre dans son fichier d'initialisation. Celui-ci s'appelle « `scilab.ini` ». S'il n'existe pas il faut le créer. Il doit se trouver dans un dossier particulier de la machine. Pour avoir le nom de ce dossier, on tape dans **Scilab** la commande `SCIHOME`.

Par exemple sous macOSX cela donne :

```
--> SCIHOME  
SCIHOME =  
/Users/Moi/.Scilab/scilab-4.0.
```

Il existe bien sûr d'autres commandes graphiques qui seront abordées dans la suite du cours, au fil des chapitres. L'aide **Scilab** comporte la syntaxe de toutes les fonctions graphiques.

➡ Exemple de graphe animé

```
hf=scf(1) ;  
xselect() ;  
clf() ;  
hf.figure_name='Animation' ;  
hf.pixmap='on' ;  
x=0 :0.005 :1 ;  
t=0 :0.005 :20 ;  
u1=zeros(x) ;  
for i=1 :1001  
plot2d(x,u1',8,rect=[0,-2,1,2]) ;  
u1=cos(4*t(i))*sin(5*pi*x).*sin(16*pi*x*t(i)).*sin(5*pi*x*t(i)) ;  
plot2d(x,u1',2+modulo(i,5),rect=[0,-2,1,2]) ;  
show_pixmap()  
end ;  
delete(hf) ;
```

➤ Exemple de représentation en coordonnées polaires

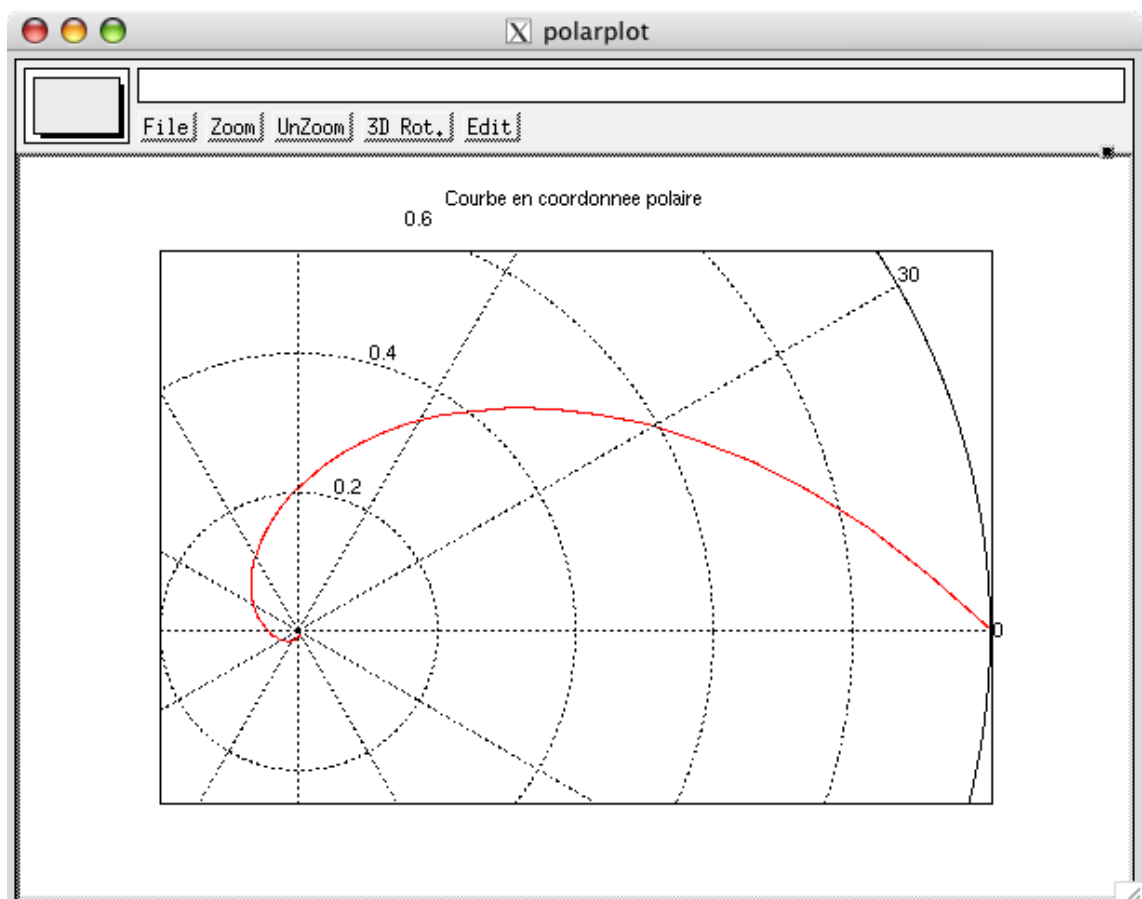
On veut représenter en coordonnées polaires sur un repère orthonormé la fonction :

$$\theta \mapsto \rho(\theta) = \exp(-\theta), \quad \theta \in [0, 2\pi]$$

Cela donne le script suivant :

```
theta=linspace(0,2*%pi);  
rho=exp(-theta);  
clf;  
polarplot(theta,rho);
```

Après arrangement de la représentation par l'éditeur graphique, cela donne :



Quelques commandes graphiques importantes

`clf()`

Efface la fenêtre courante ; `clf(x)` efface la fenêtre numéro `x`.

`xtitle(" bla-bla")`

Inscrit un titre, qui est une chaîne de caractères, dans le graphe courant. (consulter l'aide pour connaître ses différents paramètres).

`xclick()`

Le programme se met en **pause** dans l'attente d'un « appui » sur l'un des boutons de la souris à l'intérieur le cadre de la fenêtre graphique.

Cette commande est intéressante pour permettre de relever expérimentalement les coordonnées de points sur un graphe.

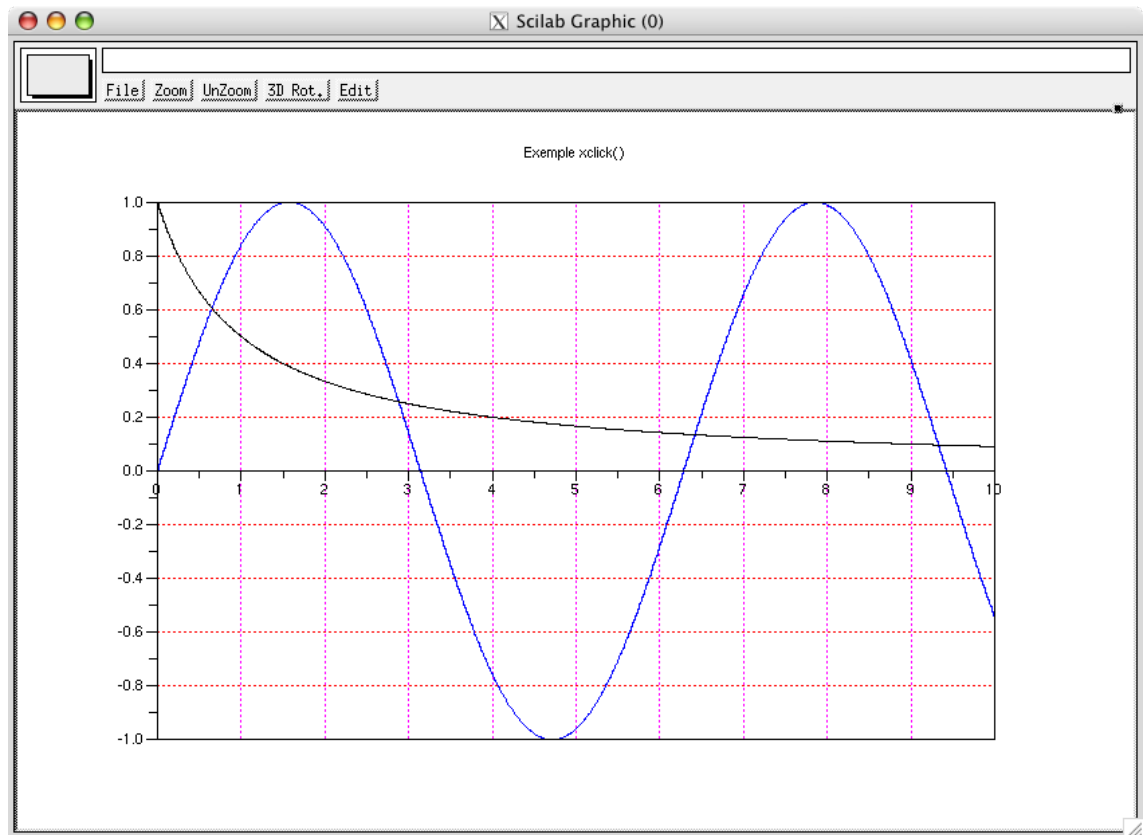
↪ Exemple

On cherche graphiquement les solutions de l'équation : $\{\sin(x) = \frac{1}{1+x}\}$ sur $[0, 10]$. On peut utiliser le script suivant :

```
x=0 :0.01 :10 ;
y=sin(x) ;
z=1.0./(1+x) ;
clf() ;
plot2d(x,y,2) ;
plot2d(x,z,1) ;
t=xclick()// On clique sur l'intersection des deux courbes
u=xclick()
v=xclick()
w=xclick()
```

`t,u,v,w` sont des vecteurs lignes associés aux quatre appels successifs à la fonction `xclick`.

Les deux figures suivantes donnent la représentation graphique associé à ce script et les résultats affichés dans la fenêtre principale.



On pratique successivement un click gauche sur chacun des points d'intersection des deux courbes (après un ou deux zoom). On obtient le résultat :

```
t =
0.    0.65    0.6057095
u =
0.    2.8780761    0.2561083
v =
0.    6.4178971    0.1349622
w =
0.    9.3286353    0.0963308
```

Les vecteurs obtenus possèdent trois composantes. Par exemple pour $\mathbf{t}$:

$\mathbf{t}(1)=0$ représente le numéro et le mode du click pratiqué.

$\mathbf{t}(2)=0.65$ représente l'abscisse du pointeur associé au click pratiqué.

$\mathbf{t}(3)=0.6057095$ représente l'ordonnée du pointeur associé au click pratiqué.

- Vérifier la pertinence de ces quatre résultats.

On consultera l'aide en ligne pour plus de précision sur cette fonction.

Les entrées et sorties sous Scilab

Dialogue utilisateur ◀ ▶ Scilab

Pour inclure des données lors de l'exécution d'un script, on dispose de plusieurs instructions permettant à l'utilisateur de commander l'évolution du programme. En voici quelques exemples simples à utiliser en mode différé.

halt()

Cette commande interrompt l'exécution du programme en attente d'une frappe sur le clavier.

Exemple

```
disp(" Je suis bloqué. Pour continuer, appuyer sur une touche ")
halt();
disp(" Merci")
```

input()

Cette commande interrompt l'exécution du programme et propose un message à l'adresse de l'utilisateur et attend une réponse clavier sous forme soit d'un scalaire soit d'une chaîne de caractères. La réponse utilisateur est alors affectée à une variable judicieusement choisie. le programme continue alors.

Exemple

1. Affectation scalaire.

```
x=input(" Donner la valeur initiale de température, entre 0 et 100");
if (x<15) then
disp("c'est trop froid");
else disp("c'est bon, merci!");
end;
```

2. Affectation chaîne de caractères.

```
x=input(" Donner un mot d'au moins 4 lettres.", "string");
L=length(x);
[u,v]=sort(rand(1,L));
y=str2code(x);
Y=y(v);
X=code2str(Y);
disp(" Avez vous donner " + X + "?");
```

`readc_()`

Cette commande interrompt l'exécution du programme en attente d'une **chaîne de caractères** sur le clavier, suivie d'un retour chariot.

- Si la chaîne est vide `readc_()` est équivalente à `halt()`.
- Si la chaîne correspond à l'écriture ou au calcul d'un nombre, on peut la transformer en une constante réelle par l'instruction `evstr()`.

`x_dialog()`

Cette commande interrompt l'exécution du programme et propose une boîte de dialogue, comportant, sous forme de (matrice de) **chaîne de caractères** :

```
Resultat= x_dialog("commentaire ..." , " valeurs suggérées ...")
```

- Un commentaire.
- Des valeurs suggérées.
- Cette commande affecte à **Resultat** :
 - La ou les valeurs suggérées si **O.K** est pressé.
 - Et `[]` (vide) si **" Cancel"** est pressé.
- Le programme continue alors.

Exemple

```
a=x_dialog("Donnez un entier entre 0 et 10 : par exemple ", "5");  
disp("a = "+a)  
disp(typeof(a))  
b=evstr(a);  
disp("b = "+ string(b))  
disp(typeof(b))
```

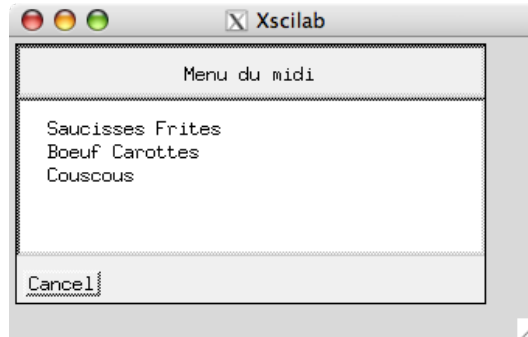
```
a=x_dialog("entrer une matrice 3x3",["[1 0 0";"0 1 0";"0 0 1]"]);  
M=evstr(a)
```

`x_choose()`

On peut facilement générer un menu, lorsque l'utilisateur a besoin de faire un choix parmi plusieurs options, avec la commande `x_choose()`, qui s'utilise de la façon suivante :

```
choix=x_choose(["Saucisses Frites","Boeuf Carottes", "Couscous"], "Menu du midi");
```

Le menu qui apparaît présente l'aspect suivant :



- Le premier argument de `x_choose()` est une matrice de chaînes de caractères, contenant les textes associés aux diverses options.
- Le deuxième argument est le titre du menu apparaissant au dessus des options.
- Après un clic de la souris sur l'une des options la fonction affecte à `choix` le numéro de l'option choisie. (ici 1, 2 ou 3; 0 correspondant à « Cancel »).

⚡ Exemple

- Initialisation.

```
choix=1;  
titre=["sin(x)", "cos(2x)", "sin(3x)","Quitter"];  
x=-2*%pi : 0.01 : 2*%pi;  
y=[sin(x) ; cos(2*x) ; sin(3*x)];
```

- Application.

```
while (choix>0)&(choix<4)  
    choix=x_choose(titre, " Choisissez une fonction");  
    if (choix==0)|(choix==4)  
        disp("Au revoir"); xdel(0);  
    else  
        clf()  
        plot2d(x,y(choix, :),2*choix);  
        xselect();  
        xgrid()  
        xtitle(titre(choix))  
    end;  
end;
```

- Tant que l'option « Quitter »(0) ou « cancel »(4) n'ont pas été choisies, on propose un choix suivi d'une action.

Dialogue fichier utilisateur ◀ ▶ Scilab

`save()`

Il est possible de sauvegarder des résultats ou des fonctions intéressantes, dans des fichiers binaires **Scilab** pour les récupérer dans une autre session, en utilisant l'instruction `save()`.

➤ Exemple

- On sauve avec `save()` puis on efface toutes les variables temporaires de la session avec `clear`

```
x=0 :0.1 :2*% pi
y=sin(x) ;
M=[x ;y] ;
save("Exemple1",M) ;
clear
```

- On récupère le fichier **Exemple1** avec `load()` pour l'utiliser, éventuellement dans un autre programme, ici un graphe de la fonction sinus.

```
load("Exemple1") ;
clf() ;
plot2d(M(1, :),M(2, :),2) ;
```

➤ Exemple

```
Jour=" Aujourd'hui, nous sommes le : ";
save("Exemple2",Jour);// On sauve Jour dans le fichier Exemple2
clear; On efface la mémoire temporaire
load("Exemple2");// On charge le fichier
disp(Jour + date())
```



Les données sauvegardées sont dans des fichiers binaires et ne peuvent être lues que par **Scilab**. Pour sauver les données dans un fichier **ascii**, donc lisibles, il est nécessaire d'utiliser des instructions issues de la préhistoire informatique. (`write` et `read`).