Une présentation de SCILAB

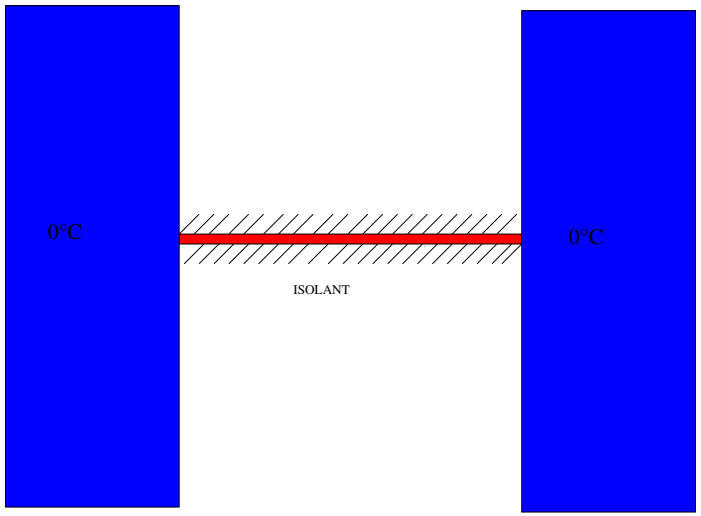
12 janvier 2013

Ouvrons une session Scilab et tapons quelques instructions.
On peut aussi utiliser un fichier de commandes (par exemple : *c1.sce*) - notez le suffixe .SCE -
ce qui permet d'éviter de taper des commandes à chaque fois et qui s'exécute (en mode "normal") seul.

Le fichier est un fichier texte, qu'on construit et affiche avec un *éditeur de texte*, par exemple **emacs** sous Linux, et on le charge dans Scilab.

Notons qu'un éditeur de texte *Scipad* est inclus dans Scilab et peut aussi être utilisé.

On veut représenter les courbes de diminution de la température d'une barre métallique, fixée par ses deux extrémités :



Les deux blocs aux extrémités sont très gros et, par conséquent, on peut considérer qu'ils absorbent la chaleur sans que leur température propre n'évolue. Initialement, la barre a été chauffée au centre, dans un 1er exemple, en deux endroits situés au 1/3 et 2/3 dans un 2eme exemple.

A quoi sert SCILAB ?

Scilab, acronyme de SCIentific LABoratory, permet de :

$->$ manipuler des données provenant de tableaux de nombres exprimés en virgule flottante et correspondant aux résultats d'une expérience

A quoi sert SCILAB ?

Scilab, acronyme de SCIentific LABoratory, permet de :

$->$ manipuler des données provenant de tableaux de nombres exprimés en virgule flottante et correspondant aux résultats d'une expérience ou, inversement,
$->$ produire, à partir d'un modèle théorique des résultats numériques pour pouvoir les confronter à l'expérience.

A quoi sert SCILAB ?

Scilab, acronyme de SCIentific LABoratory, permet de :

$->$ manipuler des données provenant de tableaux de nombres exprimés en virgule flottante et correspondant aux résultats d'une expérience ou, inversement,
$->$ produire, à partir d'un modèle théorique des résultats numériques pour pouvoir les confronter à l'expérience.

C'est un logiciel de calcul développé par l'INRIA et distribué sous licence open-source. C'est un concurrent du logiciel commercial Matlab (MATrix LABoratory) dont on peut d'ailleurs importer des fichiers.

Nous avons déjà fait un tour rapide du genre de manipulations que permet SCILAB.

Nous verrons des exemples, qui pourraient aussi se traiter sur des calculatrices un peu perfectionnées, mais avec
– une puissance de calcul beaucoup plus grande,
– un ensemble de fonctions plus important, qu'on peut facilement étendre,
– un rendu graphique bien meilleur (avec la possibilité de zoomer sur certaines parties, de faire tourner, etc...),
– des possibilités de programmation bien plus étendues...

L'interface de Scilab comprend 4 types de fenêtres :

L'interface de Scilab comprend 4 types de fenêtres :

- une fenêtre principale

L'interface de Scilab comprend 4 types de fenêtres :

- une fenêtre principale
- une fenêtre graphique

L'interface de Scilab comprend 4 types de fenêtres :

- une fenêtre principale
- une fenêtre graphique
- une aide (sous forme soit de navigateur, soit d'aide sur une commande)

L'interface de Scilab comprend 4 types de fenêtres :

- une fenêtre principale
- une fenêtre graphique
- une aide (sous forme soit de navigateur, soit d'aide sur une commande)
- un éditeur de texte pour la programmation (mais on peut le remplacer par n'importe quel autre éditeur de texte comme, par exemple, sous Linux : **emacs**.

L'interface de Scilab comprend 4 types de fenêtres :

- une fenêtre principale
- une fenêtre graphique
- une aide (sous forme soit de navigateur, soit d'aide sur une commande)
- un éditeur de texte pour la programmation (mais on peut le remplacer par n'importe quel autre éditeur de texte comme, par exemple, sous Linux : **emacs**.

Pour ouvrir une fenêtre principale

- sous Windows, on lance le programme *scilab*,
- sous Linux, soit on ouvre une fenêtre de terminal et on tape *scilab*, soit on ouvre un menu déroulant et on choisit *scilab*.

L'interpréteur de commande Nous allons successivement faire un certain nombre de calculs et vérifier que le logiciel se comporte plus ou moins comme une puissante calculatrice.

L'interpréteur de commande Nous allons successivement faire un certain nombre de calculs et vérifier que le logiciel se comporte plus ou moins comme une puissante calculatrice.

Ouvrons une session

```
--> 1+1
  ans =
     2.
--> a=sin(ans)
a =
  0.9092974
--> b =a+1
b =
   1.9092974
--> sin(2))
          !-- error 276
Missing operator, comma, or semicolon
```

Toute commande tapée dans le terminal correspond à une affectation.

A chaque fois que l'on presse la touche "Entrée", la machine effectue le calcul demandé et affiche le résultat. Si la commande est terminée par un point-virgule, le résultat est calculé, mais pas affiché.

// permet de faire des commentaires.
Au cas où la machine ne sait pas interpréter une commande, elle affiche un message d'erreur.

Une session Scilab est une suite d'opérations, quand la session est terminée, il n'en reste plus aucune trace !
Il vaut donc mieux écrire des programmes, à l'aide d'un éditeur de texte (**emacs** par exemple) qu'on fait exécuter par Scilab.

Les nombres utilisés par Scilab sont stockés, en virgule flottante, sur 64 bits. Ce qui permet de stocker et calculer avec des nombres de la forme

$$\pm a_0, a_1 a_2 \cdots a_{16} \times 10^e$$

où e est un entier compris entre -308 et $+308$.

En fait, ce nombre est constitué

- d'une mantisse $a_0, a_1 \cdots a_{16}$ et
- d'un exposant e.

Sur les 64 bits disponibles, 1 est réservé au signe, 52 à la mantisse (ce qui correspond à des nombres allant jusqu'à $2^{52} - 1$, et comme $2.22 \times 2^{52} \cong 10^{16}$, une précision de 10^{-16}) et les 11 restant codent e, avec 1 bit pour le signe et les 10 autres représentant des nombres jusqu'à $2^{1023} \cong 10^{308}$ (rappel : $2^{10} - 1 = 1023$).

Pour scilab tous les objets sont des **MATRICES**, ou des vecteurs, matrices à une seule ligne ou une seule colonne). Un nombre est également vu comme une matrice de taille 1×1.

Création de matrices

Voyons quelques briques élémentaires utiles pour construire des choses plus compliquées.

D'abord des constantes : par exemple : %eps qui donne la précision, %i pour les complexes, %pi pour le nombre π ou des constantes booléennes (voir + bas)... Puis des vecteurs et des matrices...

`1:4.5`	nombres entre 1 et 4.5 par incrément de 1
`1:1.5:3`	nombres entre 1 et 3 par incrément de 1,5
`linspace(a,b,n)`	vecteur constitué de n nombres régulièrement espacés entre a et b
`[]`	matrice vide
`[1,4,3]`	vecteur ligne $(1, 4, 3)$
`[1,2;3,4]`	matrice $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$
`ones(3,4)`	matrice de taille 3×4 remplie de 1
`zeros(3,4)`	matrice de taille 3×4 remplie de 0
`eye(5,4)`	matrice de taille 5×4 avec des 1 sur la diagonale et des 0 ailleurs
`diag(x)`	matrice carrée dont la diagonale est x
`diag(x,3)`	matrice carrée de i-ème diagonale x
`diag(M)`	vecteur constitué de la diagonale de M
`toeplitz(x)`	matrice de toeplitz bâtie sur le vecteur x
`rand(m,n)`	matrice aléatoire (loi uniforme sur $[0, 1]$) de taille $m \times n$

Opérations sur les matrices

On peut effectuer des opérations sur les matrices :

`A'`	transposée conjuguée de A (transposée dans le cas réel)
`A+B`	somme de deux matrices de mêmes tailles
`A*B`	produit de deux matrices de tailles compatibles
`5*A`	5 fois la matrice A
`A^n`	puissance n-ième d'une matrice carrée A
`A.*B`	produit coefficient par coefficient de deux matrices de mêmes tailles
`A.*n`	matrice constituée des puissances n-ièmes des coefficients de A

Concaténation de tableaux

Un exemple

`A=[1;2;3] , B=eye(3,3) , C=ones(1,4), X=[A,B;C]`
donne le résultat

$$A = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} , \; B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} , \; C = (1,1,1,1) ,$$

$$X = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 3 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Extraction d'une partie d'un tableau

Exemple

On rentre d'abord une matrice : `M=[1,2,3;4,5,6;7,8,9]`

On en extrait différentes parties :

`M(2,3)`	élément $M_{2,3}$
`M(:,1)`	première colonne
`M(1,:)`	première ligne
`M([2,3],:)`	matrice constituée des 2ième et 3ième lignes
`M([1,3],[1,2])`	matrice $\begin{pmatrix} 1 & 2 \\ 7 & 8 \end{pmatrix}$

Et en combinant extraction et affectations :

Exemple

`M(3,3)=0`	l'élément $M_{3,3}$ de M devient 0 le reste est inchangé
`M(1,:)=2*M(1,:)`	la première ligne est multipliée par 2
`M(:,[1,2])=M(:,[2,1])`	les colonnes 1 et 2 sont échangées
`M($,:)=[]`	remplace la dernière ligne par une ligne vide La taille de M devient 2×3

- Fonctions trigonométriques : `sin`, `cos`, `tan`

- Fonctions trigonométriques : `sin`, `cos`, `tan`
- Fonction trigonométriques inverses : `asin`, `acos`, `atan`

- Fonctions trigonométriques : `sin`, `cos`, `tan`
- Fonction trigonométriques inverses : `asin`, `acos`, `atan`
- Exponentielle, logarithme néperien ou décimal : `exp`, `log`, `log10`

- Fonctions trigonométriques : `sin`, `cos`, `tan`
- Fonction trigonométriques inverses : `asin`, `acos`, `atan`
- Exponentielle, logarithme néperien ou décimal : `exp`, `log`, `log10`
- Racine carrée : `sqrt`
  Valeur absolue, partie entière, signe : `abs`, `floor`, `sign`

- Fonctions trigonométriques : `sin`, `cos`, `tan`
- Fonction trigonométriques inverses : `asin`, `acos`, `atan`
- Exponentielle, logarithme néperien ou décimal : `exp`, `log`, `log10`
- Racine carrée : `sqrt`
  Valeur absolue, partie entière, signe : `abs`, `floor`, `sign`
- Somme et produit `sum` `prod`

- Fonctions trigonométriques : `sin`, `cos`, `tan`
- Fonction trigonométriques inverses : `asin`, `acos`, `atan`
- Exponentielle, logarithme néperien ou décimal : `exp`, `log`, `log10`
- Racine carrée : `sqrt`
  Valeur absolue, partie entière, signe : `abs`, `floor`, `sign`
- Somme et produit `sum` `prod`
- Ordre des coefficients de a par ordre décroissant : `sort(a)`.

- Fonctions trigonométriques : `sin`, `cos`, `tan`
- Fonction trigonométriques inverses : `asin`, `acos`, `atan`
- Exponentielle, logarithme néperien ou décimal : `exp`, `log`, `log10`
- Racine carrée : `sqrt`
  Valeur absolue, partie entière, signe : `abs`, `floor`, `sign`
- Somme et produit `sum` `prod`
- Ordre des coefficients de a par ordre décroissant : `sort(a)`.
  La commande `gsort(a)` fait la même chose, mais offre des arguments supplémentaires.

- Fonctions trigonométriques : `sin`, `cos`, `tan`
- Fonction trigonométriques inverses : `asin`, `acos`, `atan`
- Exponentielle, logarithme néperien ou décimal : `exp`, `log`, `log10`
- Racine carrée : `sqrt`
  Valeur absolue, partie entière, signe : `abs`, `floor`, `sign`
- Somme et produit `sum` `prod`
- Ordre des coefficients de a par ordre décroissant : `sort(a)`.
  La commande `gsort(a)` fait la même chose, mais offre des arguments supplémentaires.
- Norme d'un vecteur : `norm` ;
  Pour une matrice : déterminant : `det` ; inverse : `inv(A)`

- Fonctions trigonométriques : `sin`, `cos`, `tan`
- Fonction trigonométriques inverses : `asin`, `acos`, `atan`
- Exponentielle, logarithme néperien ou décimal : `exp`, `log`, `log10`
- Racine carrée : `sqrt`
  Valeur absolue, partie entière, signe : `abs`, `floor`, `sign`
- Somme et produit `sum` `prod`
- Ordre des coefficients de a par ordre décroissant : `sort(a)`.
  La commande `gsort(a)` fait la même chose, mais offre des arguments supplémentaires.
- Norme d'un vecteur : `norm` ;
  Pour une matrice : déterminant : `det` ; inverse : `inv(A)`
- Solution du système matriciel $Ax = b$: `A\b`

Constantes booléennes : `%t` et `%f` (vrai et faux).

Constantes booléennes : %t et %f (vrai et faux).

Fonctions booléennes usuelles : ==, <, >, >=, <=

Constantes booléennes : `%t` et `%f` (vrai et faux).

Fonctions booléennes usuelles : ==, <, >, >=, <=

Connecteurs logiques : ~ (non), | (ou), & (et).

Constantes booléennes : `%t` et `%f` (vrai et faux).

Fonctions booléennes usuelles : ==, <, >, >=, <=

Connecteurs logiques : ~ (non), | (ou), & (et).

Exemples

La commande `a=(1==2 | 1~=2)` affecte à la variable a la valeur `%t`.
Pour des matrices A et B, réelles, de même taille la commande `A<B` retourne la matrice des booléens `A(i,j)< B(i,j)`

Que fait la commande `find` ?

Une fonction mathématique est définie par deux expressions comme dans l'exemple $f : \mathbb{R}^2 \to \mathbb{R}$, $(x, y) \mapsto x^2 + y^3$.
En Scilab, on procède de même .
Ainsi pour définir mafonction

```
deff('z=mafonction(x,y)','z=x^2+y^3')
mafonction(3,4)
```

Exemple

```
deff('z=f(x)','z=x^2')
deff('z=eval2(g)','z=g(2)')\\
eval2(f)\\
x=1:2:10 ; feval(x,f)
```