Commandes pour survivre en *Scilab*

ECS 1ère et 2ème année

1. Demander une valeur à l'utilisateur et l'affecter

```
x=input("Entrer un nombre ou un vecteur : ");
x=input("Entrer une phrase : ", "s");
```

Cette seconde méthode permet d'enregistrer une phrase dans la variable `x`. Le `"s"` signifie `string`, pour chaîne (de caractères).

2. Afficher une valeur (nombres, vecteurs, chaines,...)

```
disp(x, "La valeur de x est : ");
```

Attention : `disp` affiche ses arguments dans l'ordre inverse.

3. Opérations et fonctions de base (x est un nombre, vecteur,...)

Math	*Scilab*
$\sqrt{x}$	`sqrt(x)`
$\ln(x)$	`log(x)`
e^x	`exp(x)`
$\lvert x\rvert$	`abs(x)`
$\lfloor x\rfloor$	`floor(x)`
$\mathcal{U}[0,1]$	`rand()`

Math	*Scilab*
Somme	`sum`
Produit	`prod`
puissance	`^`
π	`%pi`
e	`%e`
i	`%i`

Logique	*Scilab*
Vrai	`%T`
Faux	`%F`
égalité	`==`
différent	`<>`
et	`&`
ou	`|`

4. Vecteurs et opérations

(a) `a:h:b`

retourne : le vecteur ligne `[a,a+h,a+2h,...,b']` où `b'` est le plus petit nombre de la suite tel que `b'+h>b`.

```
-->1:2:10
   1.    3.    5.    7.    9.
```

(b) `linspace(a,b,n)`

retourne : le vecteur `[a,a+(b-a)/(n-1),a+2(b-a)/(n-1)...,b]`

Partage l'intervalle `[a,b]` en `n` points. Instruction équivalente à `a:(b-a)/(n-1):b`.

```
-->linspace(3,4,5)
 ans =
   3.    3.25    3.5    3.75    4.
```

(c) `ones(1,n)`
retourne : le vecteur ligne `[1, 1,..., 1]` composé de `n` 1. (`ones(n,1)` pour un vecteur colonne)

```
-->ones(1,5)
 ans =
   1.   1.   1.   1.   1.
```

(d) `zeros(1,n)`
retourne : le vecteur ligne `[0, 0,..., 0]` composé de `n` 0. (`zeros(n,1)` pour un vecteur colonne)

```
-->zeros(3,1)
 ans =
   0.
   0.
   0.
```

(e) **Opérations coordonnée par coordonnée**
`u.*v` : multiplie coordonnée par coordonnée.
`u./v` : divise coordonnée par coordonnée.
`u.^v` : coordonnée de `u` à la puissance coordonnée de `v`.

5. Alternatives

Syntaxe :

```
if  test  then
     instructions
else
     instructions
end
```

Exemple :

```
if x<>0 then   //si x différent de 0
   disp(1/x, "inverse de x : ")
else
   disp("x n''a pas d''inverse")
end
```

Remarque : le `else...` n'est pas obligatoire.

6. Boucles

(a) Boucle for

Syntaxe :

```
for i=a:h:b
      instructions
end
```

Exemple :

```
s=0
// calcul de la somme des multiples de 3, de 3 à 99
for i=3:3:100
   s=s+i
end
```

(b) Boucle while

Syntaxe :

```
while test
      instructions
end
```

Exemple :

```
x=input("Entrer un nombre positif : ")
while x<0
   x=input("Entrer un nombre positif : ")
end
```

7. Fonctions

(a) Fonctions *Scilab*

Syntaxe :

```
function [y_1,...,y_n]=NomFonction(x_1,...,x_p)
      suite d'instructions
      calculant les y_1,...,y_n
      en fonction de x_1,...,x_p
endfunction
```

Exemple :

```
function m=Moyenne(vect)
   //length : taille du vecteur
   m=sum(vect)/length(vect)
endfunction
```

(b) **Fonctions mathématiques**

Syntaxe :

```
deff("[y1,y2,...]=NomFonction(x1,x2,....)","description")
```

Exemples :

```
deff("[y]=sincard(x)", "y = sin(x)./x")

deff("[y1,y2]=MaFonction(x1,x2)", "y1=x1+x2; y2=x1*x2;"
```

Ici, `sincard(1:3)` retourne le vecteur `[sin(1)/1, sin(2)/2, sin(3)/3]`
Et `MaFonction(3,-8)` retourne le vecteur `[-5,-24]`

(c) `feval` **: images par une fonction**
Si la fonction `f` est déjà définie (en mode `function` ou `deff`) et `x` est un vecteur alors `y=feval(x,f)` crée le vecteur `y` composé des images de `x` par `f`.

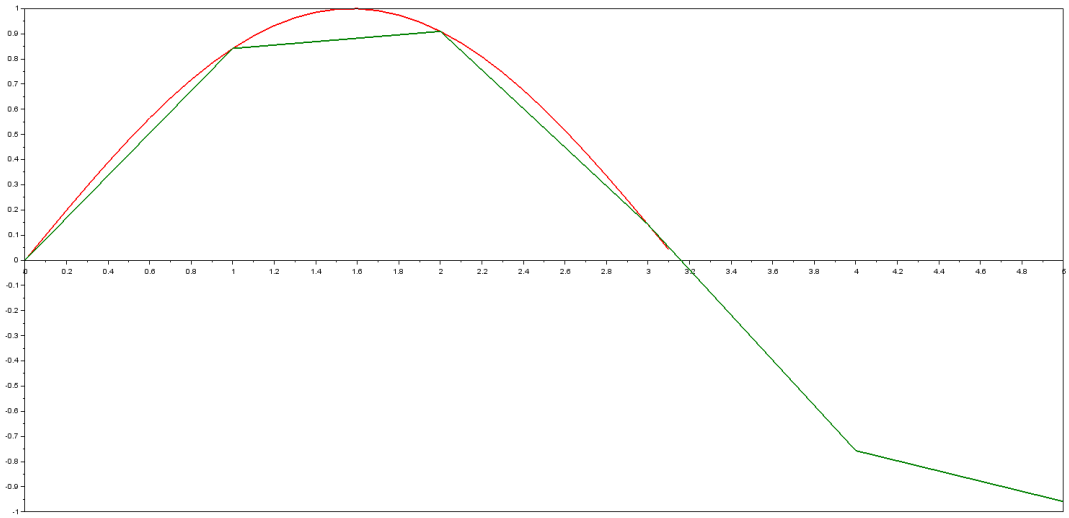
8. Graphiques

(a) `plot(x,y)` : affiche `y` en fonction de `x` (doivent avoir la même dimension).

(b) `plot(x,y,z,t)` : affiche `y` en fonction de `x` et `t` en fonction de `z` (doivent avoir la même dimension).

Exemple :

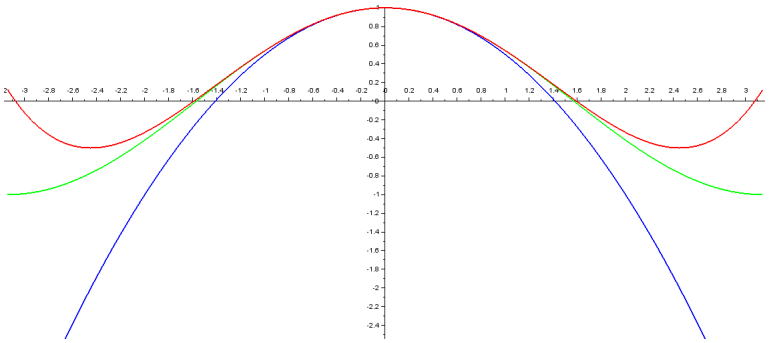
```
-->x=0:0.1:%pi;}
-->y=sin(x);}
-->z=0:1:5;}
-->t=sin(z);}
-->plot(x,y,z,t)}
```



(c) `plot2d(x, [y1,y2,...])` : Offre plus de possibilités que `plot`. Affiche `y1,y2,...` en fonction de `x` (mêmes dimensions). Attention, `x, y1, y2,...` doivent être des vecteurs colonnes !

Exemple :

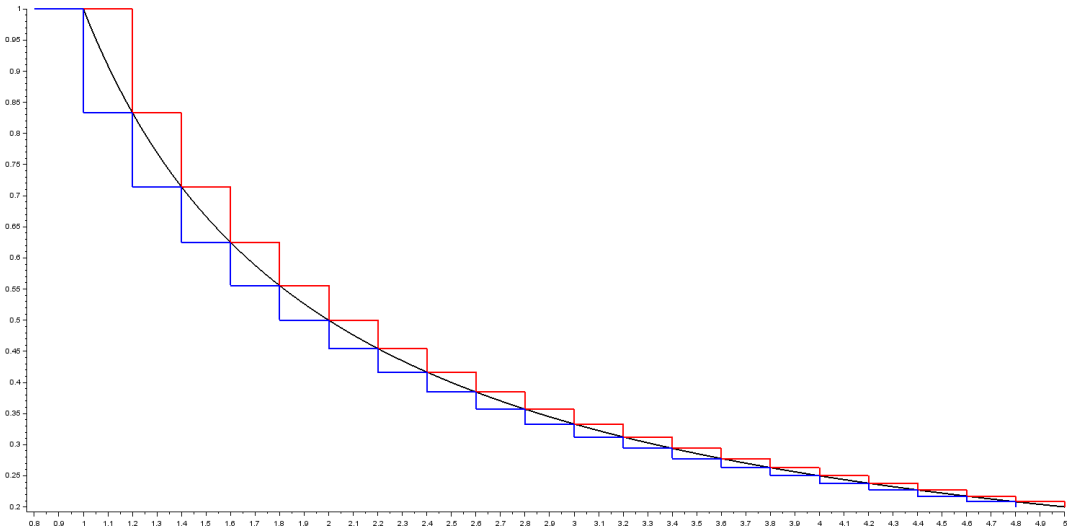
```
-->x=-%pi/2:0.01:%pi/2;
//vecteur colonne :
-->x=x'
-->plot2d(x,[cos(x), 1-x.^2/2, 1-x.^2/2+x.^4/24])
```



(d) `plot2d2(x,y)` : Presque identique à `plot2d` à la différence près que `plot2d2` est plus adaptée aux fonctions constante par morceaux. Elle est très utile, par exemple, pour illustrer la méthode des rectangles, tracer des créneaux.

Exemple :

```
-->x=0:0.01:5;
-->y=1 ./x;
-->z=0:0.2:5;
-->t=1 ./z;
-->plot2d(x,y)
-->plot2d2(z,t)
-->plot2d(z-0.2,t)
```

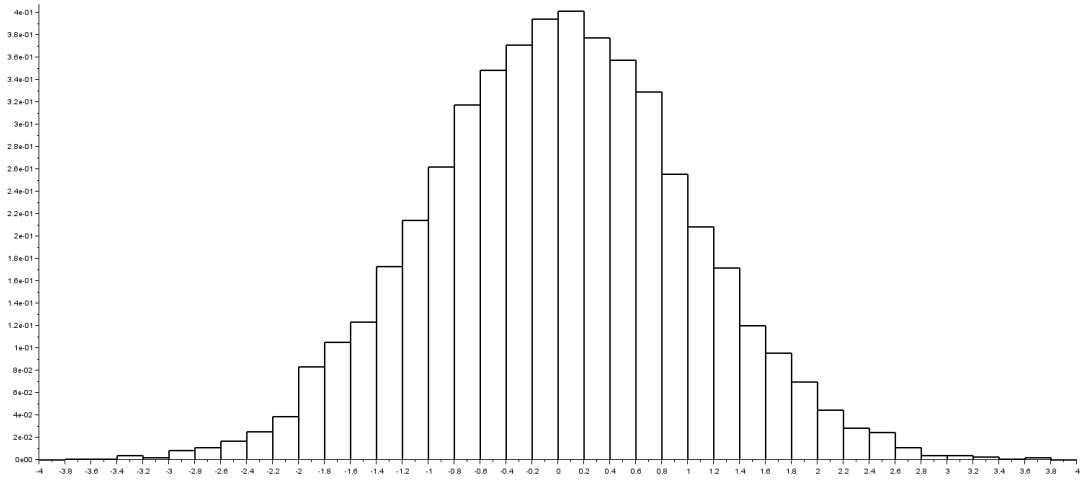


(e) `fplot2d(x,f)` : Si `x` est un vecteur et si `f` est une fonction (définie par `function` ou par `deff`) alors `fplot2d(x,f)` trace le graphe `plot2d(x,y)` où `y=f(x)`.

(f) `histplot(C,X)` : Trace un histogramme. Rectangles de base $]I_i = c_i, c_{i+1}[$ (coordonnées de `C`) et de surface proportionnelle au nombre de coordonnées de `X` appartenant à I_i.

Exemple :

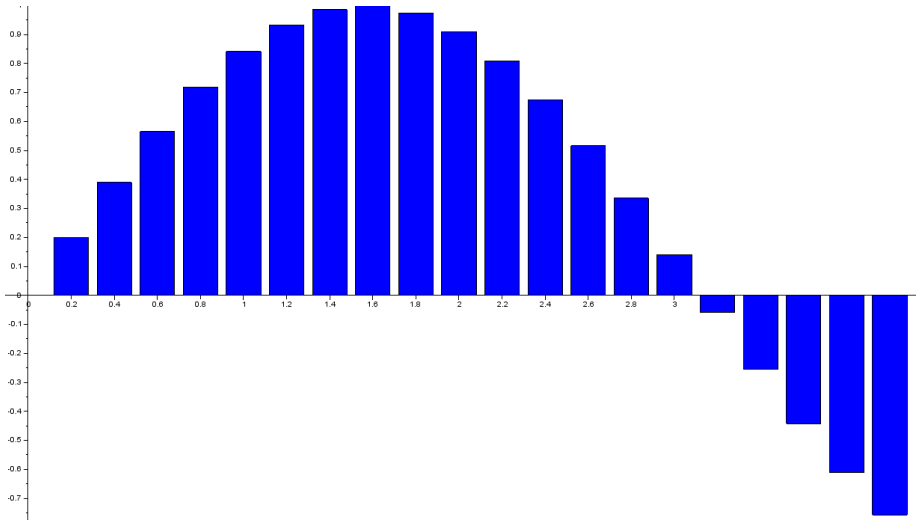
```
-->x=rand(1,10^4,"normal");
-->C=-4:0.2:4;
-->histplot(C,x)
```



(g) `bar(x,y,l)` : Trace le diagramme en bâtons de `y` en fonction de `x` avec des bâtons de largeur `l`.

Exemple :

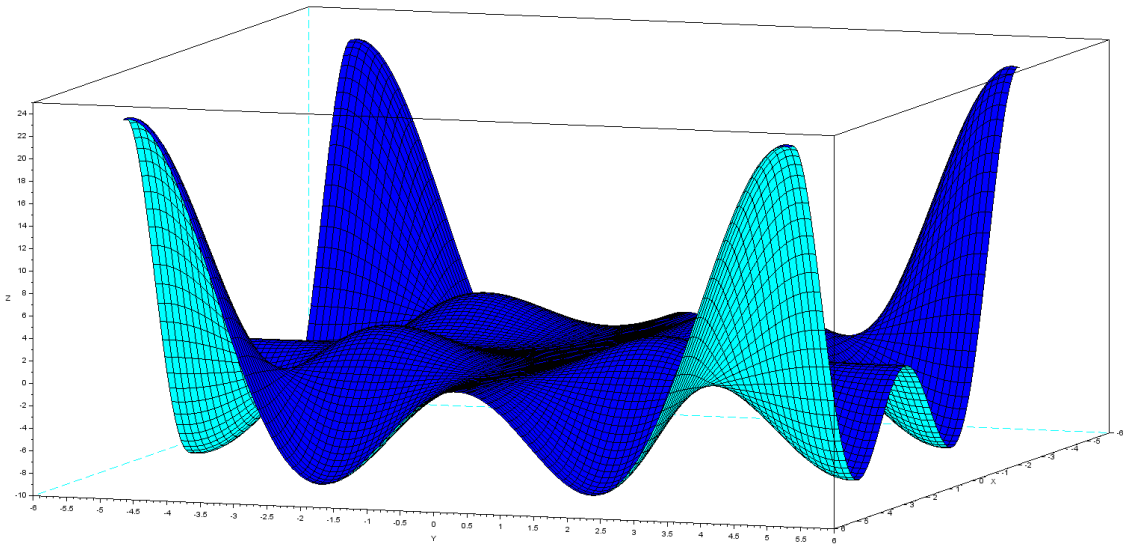
```
-->x=0:0.2:4
-->y=sin(x);
-->bar(x,y)
```



(h) `plot3d(x,y,z)` : Permet de créer une surface avec `x` un vecteur de taille n, `y` un vecteur de taille p et `z` une matrice de taille $n \times p$ on obtient le graphe de l'application f définie par $f(x_i, y_j) = z_{i,j}$.

Exemple :

```
-->x=linspace(-5,5,100);
-->y=x;
-->z=(x'.*sin(x'))*(y.*sin(y));
//Graphe de la fonction f(x,y)=x*sin(x)*y*sin(y)
-->plot3d(x,y,z)
```



(i) `fplot3d` : Mêmes fonctionnalités que `fplot2d` mais en `3d`...

9. Matrices

(a) Écriture : on écrit les matrices ligne par ligne, le séparateur étant le point-virgule ;

```
-->M=[1, 2, 3; 4, 5, 6]
M =
   1.   2.   3.
   4.   5.   6.
```

(b) Transposition : `M'` transpose la matrice `M`.

```
-->N=M'
N =
   1.   4.
   2.   5.
   3.   6.
```

(c) Concaténation : `[M,N]` concatène horizontalement les matrices (vecteurs) `M` et `N` tandis que `M;N` les concatène verticalement.

Exemples avec la matrices `M` précédemment définie :

```
-->[M,M]
 ans =
   1.   2.   3.   1.   2.   3.
   4.   5.   6.   4.   5.   6.
-->[M;M]
 ans =
   1.   2.   3.
   4.   5.   6.
   1.   2.   3.
   4.   5.   6.
```

(d) `ones(n,p)` : créer la matrice de taille $n \times p$ dont tous les coefficients valent 1.

(e) `zeros(n,p)` : créer la matrice nulle de taille $n \times p$.

(f) `eye(n,p)` : créer la matrice de taille $n \times p$ dont les coefficients diagonaux valent 1 et les autres 0.

Exemple :

```
-->eye(3,5)
 ans =
   1.   0.   0.   0.   0.
   0.   1.   0.   0.   0.
   0.   0.   1.   0.   0.
```

(g) `diag` : si `A`$= (a_{i,j}) \in \mathcal{M}_{n,p}(\mathbb{C})$ alors la commande `diag(A)` retourne le vecteur contenant les coefficients diagonaux : $[a_{11}, a_{22}, ...]$.
`diag([x1,...xn])` retourne la matrice diagonale correspondante.

(h) `rand(n,p)` : créer une matrice de taille $n \times p$ dont les coefficients indépendants sont aléatoires et suivent la loi uniforme sur [0,1].

(i) `A^n` : calcule la puissance $n^{\text{ème}}$ de la matrice `A`.
Attention ! `A^n`$\neq$`A.^n`, ce dernier calcule la matrice dont les coefficients sont les puissances $n^{\text{ème}}$ des coefficients de `A`.

(j) Inverse : `inv(A)` ou alors `A^(-1)` calcule l'inverse de la matrice `A` si elle existe.

(k) `rank(A)` : calcule le rang le la matrice `A`.

(l) Extraire une ligne/colonne d'une matrice :
`A(i,:)` extrait la $i^{\text{ème}}$ ligne de la matrice `A`.
`A(:,j)` extrait la $j^{\text{ème}}$ colonne de la matrice `A`.

10. Simulation de variables aléatoires

(a) `rand` :
`rand(n,p, "uniform")` : retourne une matrice aléatoire de taille $n \times p$ dont les coefficients indépendants suivent la loi uniforme sur $[0, 1]$.

`rand(n,p, "normal")` : retourne une matrice aléatoire de taille $n \times p$ dont les coefficients indépendants suivent la loi normale centrée et réduite : $\mathcal{N}(0, 1)$.

(b) `grand(n,p,"loi")` : s'emploie de la même manière que `rand()` mais permet de simuler beaucoup plus de lois :

i. **Loi uniforme** : `grand(a, b, "unf", m, n)` matrice de taille $a \times b$ dont les coefficients indépendants suivent une loi uniforme sur $[m, n]$.

ii. **Loi uniforme sur des entiers** : `grand(a, b, "uin", m, n)` matrice de taille $a \times b$ dont les coefficients indépendants suivent une loi uniforme sur $[\![m, n]\!]$ (très pratique pour simuler des lancers de dés par exemple).

iii. **Loi normale** : `grand(a,b,"nor",m,v)` matrice de taille $a \times b$ dont les coefficients indépendants suivent une loi normale d'espérance `m` et de variance v^2.

iv. **Loi binômiale** : `grand(a,b,"bin",n,p)` matrice de taille $a \times b$ dont les coefficients indépendants suivent une loi binômiale de paramètres `(n,p)`.

v. **Loi exponentielle** : `grand(a,b,"exp",m)` matrice de taille $a \times b$ dont les coefficients indépendants suivent une loi géométrique d'espérance `m`.

vi. **Loi géométrique** : `grand(a,b,"geom",p)` matrice de taille $a \times b$ dont les coefficients indépendants suivent une loi géométrique de paramètre `p`.

vii. **Loi de Poisson** : `grand(a,b,"poi",m)` matrice de taille $a \times b$ dont les coefficients indépendants suivent une loi de Poisson d'espérance `m`.

Il existe beaucoup d'autres paramètres : `"prm"` (retourne une permutation), `"gamma"`, `"markov"`, `"beta"`,...

(c) `cdfnor` : On note $F_{m,\sigma}$ la fonction de répartition d'une loi normale de paramètres (m,σ), soit :

$$F_{m,\sigma}(x) = \int_{-\infty}^{x} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{t-m}{\sigma}\right)^2} dt$$

En posant $F_{m,\sigma}(x) = p$, `cdfnor` permet de déterminer un des nombres $(p, 1-p, x, m, \sigma)$ quand on lui fournit les autres (`-->help cdfnor` pour plus de précisions).

11. D'autres commandes

(a) `mean(X)` : calcule la moyenne des coefficients du vecteur ou de la matrice `X`.

(b) `min(X), max(X)` : retournent respectivement le plus petit et le plus grand coefficient du vecteur ou de la matrice `X`.

(c) `find(test(A))` : retourne la position des éléments de `A` vérifiant le `test`.

Exemple :

```
-->A=[1 -2 3; 0 -5 -1]
 A=
   1. -2. 3.
   0. -5. -1.
-->find(A<0)
 ans =
   3.   4.   6.
```

(d) `length(A)` : retourne la taille du vecteur ou de la chaîne de caractères `A`.

(e) `size(A)` : retourne, sous forme de vecteur, les dimensions du vecteur ou de la matrice `A`.

(f) `trace(A)` : retourne la trace de la matrice `A`.

(g) `triu(A)` : retourne, la matrice triangulaire supérieure correspondant à `A`.

(h) `V=spec(A)` : retourne dans le vecteur `V` les valeurs propres de `A` (matrice carrée réelle ou complexe). `[P,V]=spec(A)` permet d'obtenir en plus les vecteurs propres sous forme d'une matrice.