



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

Initiation Scilab II



Les types de données Scilab

Scilab est un logiciel de **calcul numérique** et ses objets de base sont les **matrices** ou **tableaux** d'un même type de données.

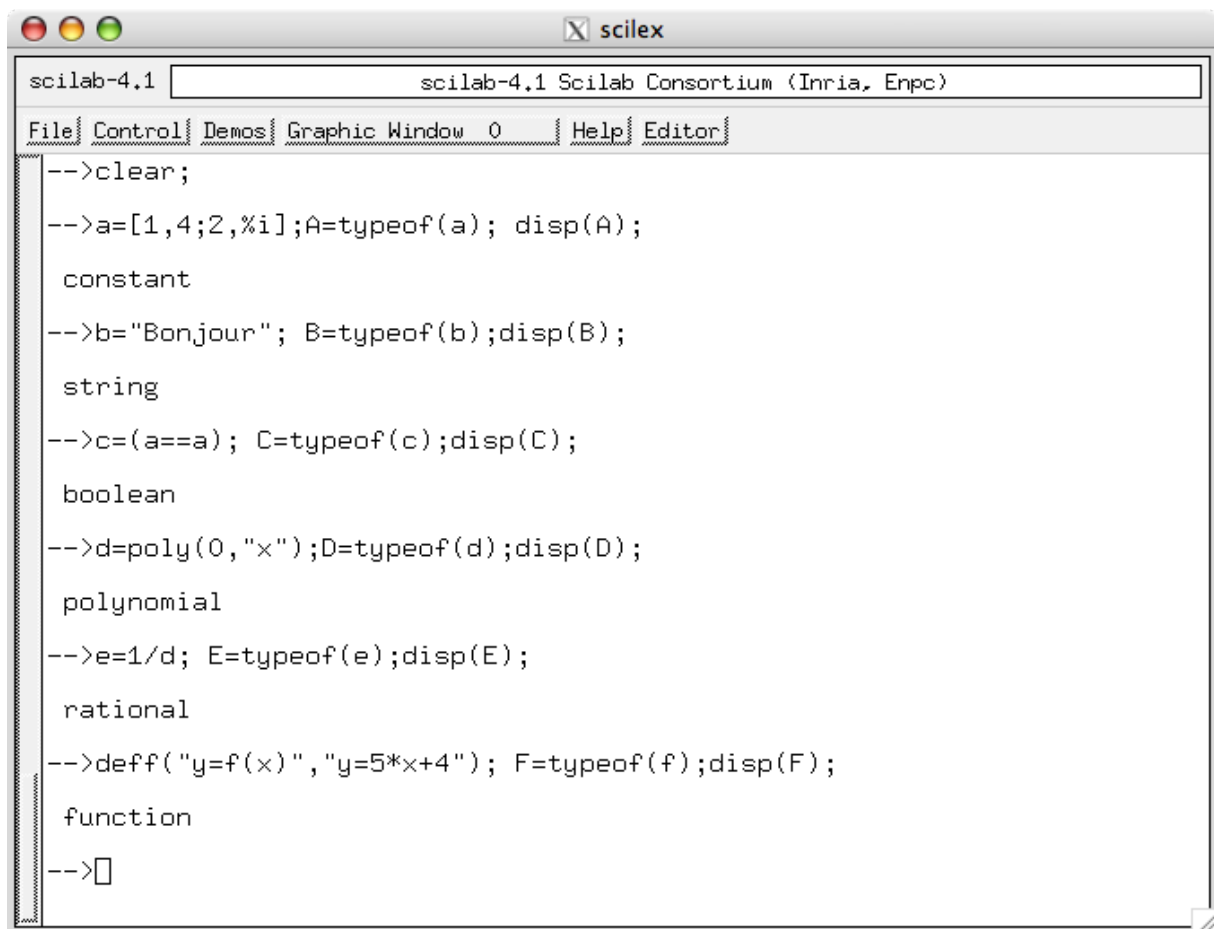
Il existe plusieurs « **types** » d'objets Scilab , avec, par exemple les types :

- **constant** : Les nombres réels ou complexes ou matrices de réels ou complexes
- **string** : Les chaînes de caractères ou matrice de chaînes.
- **boolean** : Les booléens ou matrice de booléens.
- **function** : les fonctions.
- **polynomial** : les polynômes ou matrice de polynômes.
- **rational** : les fractions rationnelles ou matrice de les fractions rationnelles.

Chaque type possède sa propre syntaxe.

Pour connaître le **type** d'une variable **xx**, on utilise la commande Scilab : `--> typeof(xx)`

➤ Exemple(s)



```
scilab-4,1 scilab-4,1 Scilab Consortium (Inria, Enpc)
File Control Demos Graphic Window 0 Help Editor
-->clear;
-->a=[1,4;2,%i];A=typeof(a); disp(A);
constant
-->b="Bonjour"; B=typeof(b);disp(B);
string
-->c=(a==a); C=typeof(c);disp(C);
boolean
-->d=poly(0,"x");D=typeof(d);disp(D);
polynomial
-->e=1/d; E=typeof(e);disp(E);
rational
-->deff("y=f(x)","y=5*x+4"); F=typeof(f);disp(F);
function
-->
```

Dans l'exemple précédent, on a défini, **par affectation** des variables qui sont mis temporairement en mémoire.

Pour connaître le nom des variables et données, utilisées et mises en mémoire on utilise la commande Scilab :

`--> who`

➤ Exemple(s)

Le groupe deux figures suivantes peut être assimilé au jeu de 7 erreurs.

On effectue deux appels à la commande `--> who`. La première avant la commande `--> clear`, et la deuxième après.

Que remarque-t-on ?

Figure 1

```
-->who
your variables are...

F      f      E      e      %s_r_p  D      d      C      c
B      b      A      typeof a      scicos_pal      %helps  home
SCIHOME PWD      TMPDIR  MSDOS  SCI      guilib  sparselib      xdesslib
percentlib      polylib  intlbr  elemllib  utillib  statslib  alglbr  siglib
optlib  autolib  rollib  soundlib  metalib  armalib  tkscilib  tdcslbr  s2flbr
mtllib  %F      %T      %z      %s      %nan  %inf  COMPILER  %gtk
%gui    %pvm  %tk    $      %t      %f      %eps  %io    %i
%e
using      7465 elements out of 5000000.
and      62 variables out of 9231

your global variables are...

LANGUAGE %helps  demolist %browsehelp      LCC      %toolboxes
%toolboxes_dir  INDEX
using      1629 elements out of 11000.
and      8 variables out of 767

-->■
```

L'instruction `--> clear` efface la mémoire temporaire.

Figure 2

```
-->clear

-->who
your variables are...

scicos_pal      %helps  home  SCIHOME  PWD      TMPDIR  MSDOS  SCI
guilib  sparselib  xdesslib  percentlib  polylib  intlbr  elemllib
utillib  statslib  alglbr  siglib  optlib  autolib  rollib  soundlib  metalib
armalib  tkscilib  tdcslbr  s2flbr  mtllib  %F      %T      %z      %s
%nan    %inf  COMPILER  %gtk  %gui  %pvm  %tk    $      %t
%f      %eps  %io    %i    %e
using      6450 elements out of 5000000.
and      48 variables out of 9231

your global variables are...

LANGUAGE %helps  demolist %browsehelp      LCC      %toolboxes
%toolboxes_dir  INDEX
using      1629 elements out of 11000.
and      8 variables out of 767

-->■
```

Les scalaires

Les scalaires désignent pour, `Scilab`, des matrices ou tableaux de nombres réels flottants en double précision ou de nombres complexes en double précision.

Les nombres réels ou complexes

Les nombres réels ou complexes sont des matrices une ligne une colonne.

Les opérations sur ces nombres sont donc des cas particuliers des opérations sur les matrices.

Opérations mathématiques	Instructions <code>Scilab</code>
addition +	+
soustraction −	-
multiplication ×	*
division ÷	/
puissance	^

Il existe quelques scalaires pré-définis `Scilab`.

Par exemple la constante `Scilab` pré-définie « `%eps` » donne la précision de la machine.

`%eps` est le plus grand nombre en double précision tel que : $1+(\%eps)/2==1$

```
-->%inf
-->%e
-->%pi
-->%i
```



Exercices

Evaluer le résultat des commandes suivantes.

```
-->%eps+%eps
-->%inf-1
-->%inf*0
-->1/%inf
-->a=%eps
-->Z=2+2*%i
-->abs(Z)
--> real(Z)
--> imag(Z)
-->Z*Z
-->sqrt(Z)
-->t=Z'
-->u=Z*Z'
-->v=cos(%pi)
```

Les scalaires aléatoires

Les scalaires peuvent être déterministes ou **aléatoires**. Scilab possède plusieurs générateurs de nombres aléatoires (plus précisément pseudo-aléatoires). La commande la plus simple est mise en œuvre lorsque l'on utilise la fonction `--> rand()`.

- La fonction prédéfinie `rand()` renvoie, par défaut, un nombre choisi « au hasard » entre 0 et 1.
- Des appels successifs à cette fonction donnent une suite de nombres aléatoires « indépendants » compris entre 0 et 1.
- Un chapitre de cette initiation, consacré à l'utilisation de ces nombres, apportera plus de précision sur l'utilisation des nombres aléatoires, en simulation ou pour construire des objets dits « quelconques ».

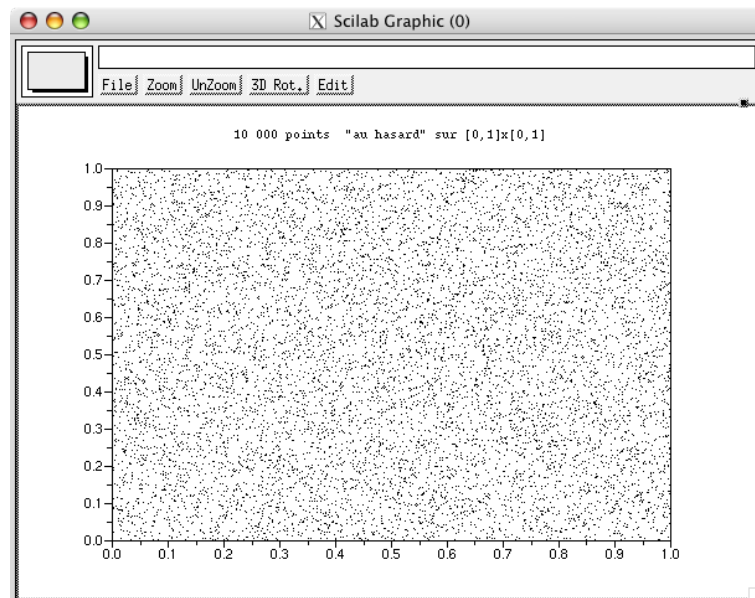
⚡ Exemple(s)

Effectuer successivement les commandes suivantes :

```
--> x=rand();
--> y=rand();
--> z=x*y;
--> u=-log(x);
--> v=-log(y);
--> w=u+v;
--> r=round(2*rand());
--> r=floor(6*rand());
```

⚡ Exemple(s)

```
--> x=rand(1,10000);y=rand(1,10000); clf; plot2d(x,y,0);
```



`a=rand(1,n)` donne un vecteur ligne de n points de $]0, 1[$, pris "au hasard".

Les tableaux de scalaires

Les tableaux monodimensionnels : Les vecteurs

On appelle **vecteur**, par analogie aux vecteurs de \mathbb{R}^n , une suite finie de données, représentée sous la forme d'un tableau sur une ligne ou sur une colonne. Il peut s'agir de vecteurs scalaires ou de vecteurs de chaînes de caractères ou vecteurs d'autres types, booléens, chaînes de caractères, polynômes etc.

Création de vecteurs de scalaires

Il existe de multiples modalités de construction de vecteurs. On pourra tester les différentes propositions suivantes.

- Un vecteur **ligne** peut être défini par la liste de ses éléments, séparés par un espace ou une **virgule** et insérée entre deux crochets [].

```
--> v= [ 1,3,5,9]
```

- Un vecteur **colonne** peut être défini par la liste de ses éléments, séparés par une **point-virgule** et insérée entre deux crochets [].

```
v1= [ 1 ; 3 ; 5 ; 9]
```

- Si V est un vecteur **ligne** son transposé V' est alors un vecteur **colonne** et réciproquement.

```
v1= [ 1,3,5,9]'
```

```
v2=v1'
```

Construction d'un vecteur dont les éléments forment une progression arithmétique

On utilise l'instruction rencontrée dans les boucles (sans espace) :

a=	début :	pas :	fin
----	---------	-------	-----

« a » désigne le vecteur scalaire **ligne** dont **début** est le premier terme de la progression. « pas » est le pas d'incrément, positif ou négatif (qui vaut 1 par défaut) et « fin » est la valeur à ne pas dépasser.

```
a= 3 :8 // progression partant de 3, de raison 1, et dont le dernier terme est  $\leq 8$ .
```

```
b=12 : - 3 : - 4 // progression partant 12, de raison -3, et de dernier terme  $\geq -4$ 
```

```
b=12 : 3 : 4 // donne le vecteur vide : []
```

```
c=1 :2 :2 // donne le vecteur c=!1!
```

- Autre commande possible : « **linspace** ».

```
c=linspace(1,4,10)// progression allant du début=1, à la fin = 4, en 10 étapes.
```

On remarquera qu'ici **début** < **fin**.

Construction d'un vecteur dont les éléments constituent une progression géométrique

```
q=0.5 ; g=q^(1 :10) ;
```

```
q=0.5 ; Q=cumprod( q*(1 :10)) ;
```

Vecteurs pouvant servir de boîte ou de container.

```
u=zeros(1 , 4) ; // on indique à scilab que u est un vecteur ligne de 4 éléments
u(2)=5 // u est alors un vecteur ligne dont l'élément d'indice 2 vaut 5
v=ones(1 , 4) ;
w1(5)=4, w2=w1'
```

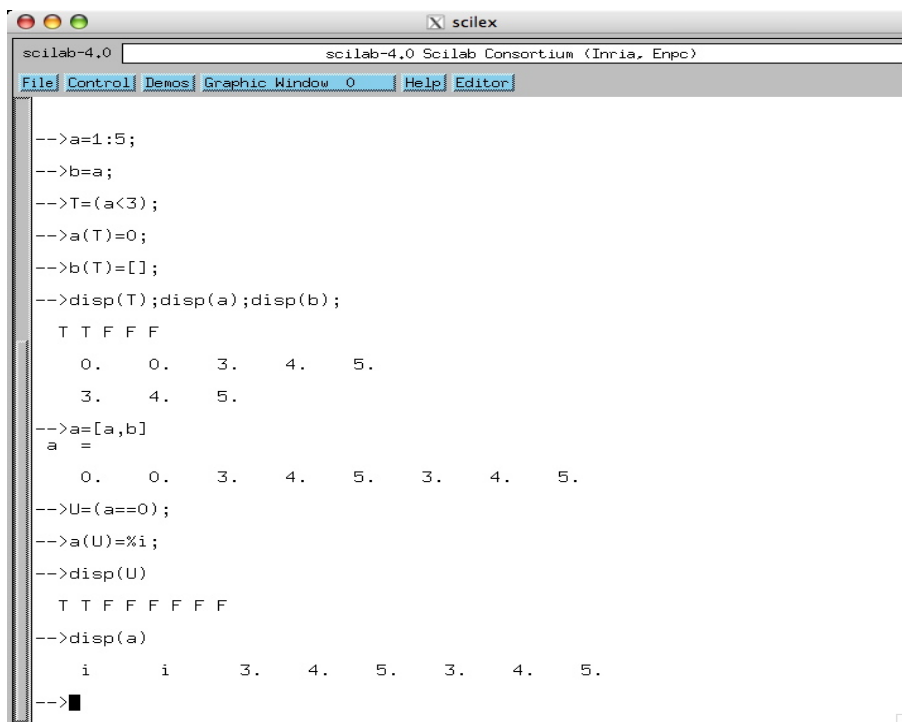
Le vecteur vide.

```
u=[] ; // utile pour initialiser une boucle
for i=1 :4 ; u=[u,sin(i*% pi)] ; end ; disp(u)
v=3 : 1 ; u=[v,u,v] ;
```

Accès aux éléments d'un vecteur, extraction, reconstruction

Expérimenter les commandes suivantes :

```
u=1 :3 :12 ; disp(u(3)) ; disp(u($)) // $ représente le dernier élément du vecteur
v=1 :3 :12 ; u=v(1 :3)
v=1 :3 :12, v(1 :3)=7
v=1 :3 :12, w=[v , v]
v=1 :3 :12,
w=[v , v , v] ; w($-3)=2 // $ représente le dernier élément du vecteur
v=1 :5 :12, w=[v , v , v] ; w($ :-1 :1)
v=1 :5 :12, w=[v , v , v] ; w($ :-1 :4) =[]
```



```
scilab-4.0
scilab-4.0 Scilab Consortium (Inria, Enpc)
File Control Demos Graphic Window 0 Help Editor

-->a=1:5;
-->b=a;
-->T=(a<3);
-->a(T)=0;
-->b(T)=[];
-->disp(T);disp(a);disp(b);
  T T F F F
  0.  0.  3.  4.  5.
  3.  4.  5.
-->a=[a,b]
a =
  0.  0.  3.  4.  5.  3.  4.  5.
-->U=(a==0);
-->a(U)=%i;
-->disp(U)
  T T F F F F F F
-->disp(a)
  i    i    3.  4.  5.  3.  4.  5.
-->■
```

Procédures et instructions prédéfinies sur les vecteurs

```
u=1 :3 :22 ; l=length(u) // nombre d'éléments du vecteur (longueur)
u=1 :3 :22 ; s=size(u) //taille du vecteur (considéré comme un tableau)
u=11 :3 :11 ;
isempty(u)
u=1 :20 ;
S_1=sum(u) // somme des éléments de u :  $\{u_i = i, \text{ pour } i = 1, 2, \dots, 20\}$ ,  $S_1 = \sum_{i=1}^{20} u_i$ 
P_1=prod(u) // produit des éléments de u :  $P_1 = \prod_{i=1}^{20} u_i$ 
u=1 :3 :22 ;
M=max(u)
m=min(u)
moy=mean(u)
EcartType=st_deviation(u)
```

Sommes et produits cumulés

`cumsum`

`cumprod`

La somme cumulative et le produit cumulatif sont des outils `Scilab` très utiles pour étudier la convergence d'une série ou d'un produit. ou pour construire certain type de vecteurs.

↪ **Exemple(s)**

```
u=1 :3 :22 ; S_2=cumsum(u) //  $S_2(k) = \sum_{i=1}^k u_i$ , pour  $k = 1, 1 \dots 20$ .
```

Il s'agit donc d'un vecteur de même longueur que le vecteur de départ.

```
u=1 :3 :22 ; P_2=cumprod(u) //  $P_2(k) = \prod_{i=1}^k u_i$ , pour  $k = 1, 1 \dots 20$ .
```

Il s'agit donc d'un vecteur de même longueur que le vecteur de départ.

↪ **Exemple(s)**

On cherche à évaluer l'évolution de la suite S_n en fonction $1 \leq n \leq 500$: $S_n = \sum_{k=1}^n \frac{1}{k^2}$,

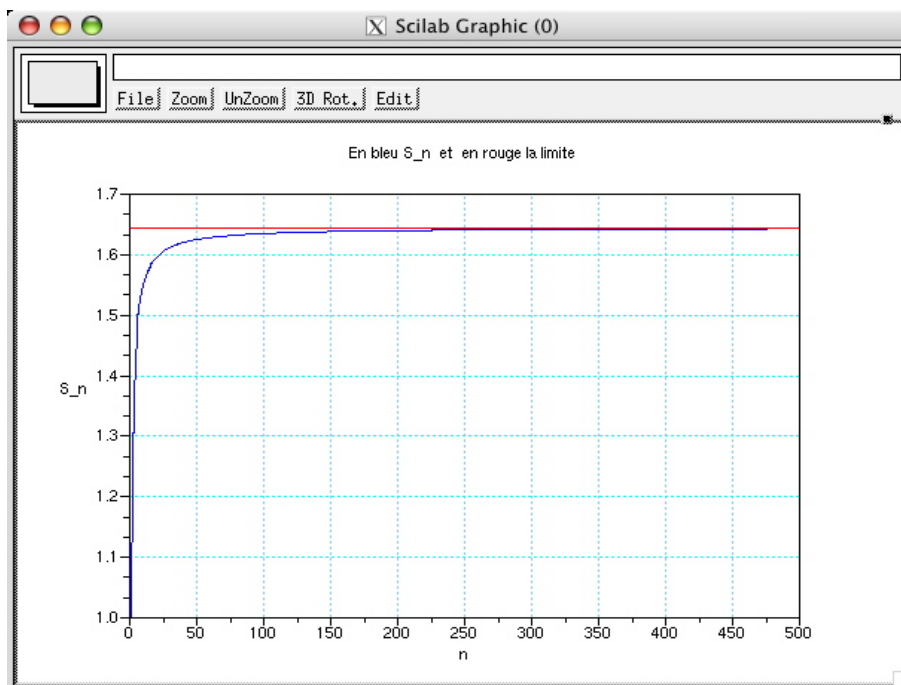
et on veut la comparer à : $\frac{\pi^2}{6}$ et donner une représentation graphique `Scilab` à cette comparaison.


```

N=500 ;
v=1 :N ;
w=v.^2 ;
u=ones(w)./w ; //ones(w) est un vecteur de même dim que w composé de 1
S=cumsum(u) ;
clf ;
plot2d(v,S,2) ;
plot2d(v,((%pi^2)/6)*ones(v),5) ;
disp(S($)-((%pi^2)/6)) ;

```

On obtient la figure suivante (modifiée pour être plus claire en utilisant l'éditeur graphique).



Opérations sur les vecteurs

```

u=1 :5 ; v=-u // opposition terme à terme
u=1 :5 ; a=3 v=a*u // multiplication par un scalaire
u=1 :5 ; v=3 :7 ; h=u*(v') // produit scalaire de deux vecteurs
u=1 :5 ; v=5 :-1 :1 ; u+v // addition terme à terme
u=1 :5 ; v=5 :-1 :1 ; u.*v // produit terme à terme, si les dimensions sont compatibles
u=1 :5 ; v=5 :-1 :1 ; u./v // division élément par élément (non nul)
u=cumprod(%i*ones(1,4)) ; u=u' // transposée de la conjuguée
u=cumprod(%i*ones(1,4)) ; u=u.' // transposée

```

```
u=%pi/6 cumsum(0.5*ones(1,4)) ; v=sin(u)
u=floor(6*rand(1,5))
w=sort(u) // consulter l'aide.
```



Exercices

Construire les vecteurs dont les éléments ont les propriétés suivantes :

a) Suite de nombres de 1 à 3, avec un pas de 0, 1.

b) Suite de nombres de 3 à 1, avec un pas de $-0, 1$.

c) Suite des carrés dix premiers nombres entiers. Calculer la somme : $S_2 = \sum_{k=1}^{10} k^2$.

d) Suite des inverses des carrés dix premiers nombres entiers.

f) **Calculer la somme** : $T = \sum_{k=1}^{10} \frac{1}{k^2}$. Comparer T à $\frac{\pi^2}{6}$

g) **Calculer la somme** : $S = \sum_{k=1}^{10} \frac{1}{k!}$. Que vaut $\delta = S - e$?

Construire les vecteurs dont les éléments ont les propriétés suivantes :

h) Suite de nombres de comportant 10 « 0 », suivis de 10 « 1 ».

i) Suite de nombres de la forme : $\{(-1)^k\}_{k=1}^{10}$.

j) Suite de nombres de la forme : $\{\frac{(-1)^k}{k!}\}_{k=1}^{10}$.

k) **Calculer la somme** : $Z = \sum_{k=1}^{10} \frac{(-1)^k}{k!}$. Comparer à une fonction de e .

Tableaux bidimensionnels : les matrices

Le logiciel **Scilab** a été spécifiquement conçu pour être particulièrement adapté au calcul matriciel. les procédés de constructions de matrices et les opérateurs sur les matrices sont très nombreux et très puissants. En voici quelques exemples.

Création de matrices et de tableaux scalaires

Par extension de ce qui a été vu pour les vecteurs, qui sont des cas particuliers de matrices ou de tableaux de données, la création de matrice ou de tableaux bidimensionnel suit les mêmes procédures.

Voici quelques exemples de construction à expérimenter.

Construction élément par élément d'une matrice

```
z=[2+%i,3;5,7;1,8]
```

Il s'agit d'une matrices de scalaires complexes, trois lignes deux colonnes. Les deux termes de la ligne 1 sont séparés par une « , » **virgule**.

On passe à la deuxième ligne par un « ; » **point-virgule**. Les deux termes de la ligne 2 sont séparés par une virgule.

On passe à la troisième ligne par un point-virgule, etc.

Il est nécessaire que les lignes soient de même taille : le tableau est rectangle.

```
zt=z' // transposée de la conjuguée de z; matrice 2 lignes 3 colonnes
```

```
zt=z.' // transposée de z; matrice 2 lignes 3 colonnes
```

Construction d'une matrice par un procédé spécifique

```
A(3,5)=2
```

On définit ex-nihilo une matrice A , trois lignes et cinq colonnes, composé de termes égaux à 0, sauf le termes $A_{(3,5)}$ qui vaut 2.

```
B=[1 :3;2 :4]
```

la première ligne de B est (1, 2, 3) suivie de **point-virgule**, la deuxième ligne vaut : (2, 3, 4)

```
C=ones(2,3)
```

C est une matrice, 2 lignes 3 colonnes dont tous les élément sont égaux à 1

```
D=eye(2,3)
```

D est une matrice, 2 lignes 3 colonnes. Les termes diagonaux valent 1, les autres sont nuls.

```
E=rand(2,3)
```

Matrice 2 lignes 3 colonnes. Les termes sont des nombres aléatoires indépendants $\in]0,1[$.

```
F=zeros(2,3)
```

Matrice 2 lignes 3 colonnes. Tous ses termes sont nuls.

■ G=diag(1 :4)

Matrice diagonale G. Les termes diagonaux valent (1,2,3,4).

■ H=triu(rand(4,3))

Matrice triangulaire supérieure (u pour "up") extraite de la matrice rand(4,3)

■ J=tril(rand(4,3))

Matrice triangulaire inférieure (l pour "low") extraite de la matrice rand(4,3).

Matrices prédéfinies

a=zeros(n,m)	a, matrice $n \times m$, $a_{i,j} = 0 \forall i,j$
b=ones(n,m)	b, matrice $n \times m$, $b_{i,j} = 1, \forall i,j$
c=zeros(M)	c, matrice de même dim. que M, $c_{i,j} = 0, \forall i,j$
d=ones(M)	c, matrice de même dim. que M, $d_{i,j} = 1, \forall i,j$
e=eye(M)	e, matrice de même dim. que M, $e_{i,i} = 1, (= 0 \text{ si } i \neq j)$
f=eye(n,m)	f, matrice de dim $n \times m$, $f_{i,i} = 1, (= 0 \text{ si } i \neq j)$
hasardU=rand(n,m,'u')	hasardU, matrice $n \times m$, $a_{i,j}$ aléatoire de loi uniforme
hasardN=rand(n,m,'n')	hasardN, matrice $n \times m$, $a_{i,j}$ aléatoire de loi normale
T=testmatrix('magi',n)	T, matrice-test : carré magique $n \times n$

Restructuration d'une matrice à partir d'un vecteur ou d'une matrice

L'instruction : `matrix`

- Soit $K = 1 : 12$; . *K est une matrice 1 ligne 12 colonnes*
- Posons : $L=matrix(K, 3,4)$; *Restructuration de K en une matrice 3 lignes,4 colonnes.*
- Posons : $M=matrix(L, 2,6)$; *Restructuration L en une matrice 2 lignes, 6 colonnes .*
- Et finalement, $M=matrix(L, 12,1)$; *Restructuration de M en une matrice 12 lignes, 1 colonne.*



Remarques.

- $G=matrix(1 :2, 3,4)$; conduit à un message d'erreur par manque de cohérence
- $GG=matrix(1 :12, 2,5)$; conduit à un message d'erreur par manque de cohérence.
- $A=[1 \ 2 \ 3 \ 4]$, $B=[1,2,3,4]$;// *A et B sont identiques. La virgule peut être remplacée par un espace, un blanc, ceci n'est pas recommandé car source d'erreur*
- $S=1$, $T=[1]$; $S==T$;// *S et T sont identiques.*
- On remarquera que la numérotation des éléments d'une matrice est effectuée de la façon suivante :

```

scilab-4.1
scilab-4.1 Scilab Consortium (Inria, Enpc)
File| Control| Demos| Graphic Window 0 | Help| Editor|

-->A=1:12;
-->B=matrix(A,3,4)
B =
    1.    4.    7.   10.
    2.    5.    8.   11.
    3.    6.    9.   12.

-->C=matrix(A,4,3)
C =
    1.    5.    9.
    2.    6.   10.
    3.    7.   11.
    4.    8.   12.

-->D=matrix(A,2,6)
D =
    1.    3.    5.    7.    9.   11.
    2.    4.    6.    8.   10.   12.

-->AA=matrix(D,1,12)
AA =
    1.    2.    3.    4.    5.    6.    7.    8.    9.   10.   11.   12.

-->u=A(3)
u =
    3.

-->v=B(3)
v =
    3.

-->■

```

Opérations sur les matrices scalaires

Chaque élément du tableau de scalaires est affecté, **terme à terme**. Les tableaux sont constitués d'éléments de même type.

⚡ Exemple(s)

```

a=rand(3,3); b=-log(a)
a=rand(3,3); b=sin(2*%pi*a)
a=rand(3,3); b=exp(a)
a=rand(3,3); p=a.*a // multiplication terme à terme : « .* »
a=rand(3,3); d=a./a // division terme à terme : « ./ »
a=rand(3,3); b=a.^2 // « .^ »

```



✍ Exercices

Evaluer les instructions suivantes :

```

a=ones(1,16); b=cumsum(a); c=matrix(b,4,4), s=sum(c)
a=ones(4,4); b=cumsum(a); c=matrix(b,1,12), s=sum(c) // que remarque-t-on ?
u=rand(3,3); r=sort(u), v=-sort(-u)
u=rand(3,3); M=max(u), m=min(u)
u=1+floor(6* rand(1,1)); v=1+floor(6* rand(1,1))
a=ones(u,v); t=size(a), l=length(a)

a=rand(3,3); p=a*a, pp=a^2

```

```

a=rand(3,3); b=ones(3,3); aplusb=a+b, afoisb=a*b
a=[0.2,0.8;0.6,0.4]; e=exp(a), ee=expm(a)
a=rand(3,3); s=sin(a).*sin(a)+cos(a).*cos(a), t=sin(a)*sin(a)+cos(a)*cos(a)
a=1 :10; b= a.*a , s=sum(a)
a=1 :10; b= a*a // message d'erreur, dimensions incompatibles
a=1 :10; b= a'*a
a=1 :10; b= a.*a'
```

Accès aux éléments d'une matrice, extractions, reconstructions



Exercices

```

M=Matrix(1 :9,3,3);
d=diag(M), mm=diag(d)
MM=pertrans(M)

M=Matrix(1 :15,3,5);
M22=M(2,2) // extraction localisée par un indice
a=M(2,$)
b=M($,2)
c=M($-1,2)
d=M($-1,2/2)
A=M(2, :)
B=M( :, $)

M=Matrix(1 :20,2,10);
N=M([1,2],[1,3,5]) // extraction déterminée par des vecteurs
M(2,2)=111; M // affectation

M=Matrix(1 :20,2,10);
M([1,2],[1,3,5])=0 // affectation localisée

M=Matrix(1 :20,2,10);
M([1,2],[1,3,5])=[] // suppression localisée
```

Opérations sur les matrices de dimensions compatibles, terme à terme

On considère des matrices A et B de mêmes dimensions.

$C=A+B$	$c_{i,j} = a_{i,j} + b_{i,j}$
$P=A.*B$	$p_{i,j} = a_{i,j} \times b_{i,j}$
$E=A.^B$	$e_{i,j} = a_{i,j}^{b_{i,j}}$
$D=A./B$	$d_{i,j} = \frac{a_{i,j}}{b_{i,j}}$
Mligne=[A,B]	matrice constituée de deux blocs en lignes (A, B)
Mcol=[A;B]	matrice constituée de deux blocs en colonne

Opérations sur les matrices de dimensions compatibles

On considère des matrices A et B de dimensions compatibles.

P=A*B	Produit matriciel classique
Mligne=[A,B]	matrice constituée de deux blocs en lignes (A, B)
Mcol=[A ;B]	matrice constituée de deux blocs en colonne
r=rank(A)	rang de A
k=kernel(B)	noyau de B

Opérations et fonctions sur les matrices carrées

Soit C une matrice carrée.

d=det(C)	d : déterminant de C
i=inv(C)	inverse de C (si $\det(C) \neq 0$)
e=expm(C)	exponentielle matricielle
s=spec(C)	valeurs propres de C
t=trace(C)	trace de C
t=poly(C, "x")	polynôme caractéristique de C

Fonctions pré-définies sur les matrices

On considère une matrice x, n lignes et m colonnes :

$$\mathbf{x} = (x_{(i,j)}, i \in [1, n], j \in [1, m])$$

Les fonctions suivantes s'appliquent termes à termes :

$$y_{(i,j)} = f(x_{(i,j)}), \forall i \in [1, n], j \in [1, m]$$

sin(x)	$\sin(x)$
cos(x)	$\cos(x)$
tan(x)	$\tan(x)$
cotg(x)	$\cotan(x)$
atan(x)	$\arctan(x)$
asin(x)	$\arcsin(x)$
acos(x)	$\arccos(x)$
sqrt(x)	\sqrt{x}
exp(x)	$\exp(x)$
log(x)	$\ln(x)$
log10(x)	$\log(x)$
abs(x)	$ x $
conj(x)	conjugué de x
x^y	x^y (*)

<code>floor(x)</code>	approximation de x dans \mathbb{Z} , vers $-\infty$
<code>int(x)</code>	approximation de x dans \mathbb{Z} , vers 0
<code>ceil(x)</code>	approximation de x dans \mathbb{Z} , vers $+\infty$
<code>round(x)</code>	approximation de x dans \mathbb{Z} , le plus proche
<code>sign(x)</code>	signe de x
<code>modulo(x,n)</code>	reste de la division de x (entier) par n (*)

Fonctions matricielles prédéfinies

<code>size(M)</code>	taille (m,n) de la matrice M
<code>length(M)</code>	nombre d'éléments de M ($m \times n$)
<code>M(i, :)</code>	extraction de la i-ème ligne de M
<code>M(:, j)</code>	extraction de la j-ème colonne de M
<code>M=diag(v,0)</code>	matrice $M_{i,i} = v_i$ ($=0$ si $i \neq j$)
<code>M=diag(v,k)</code>	matrice $M_{i,i+k} = v_i$ ($=0$ sinon)
<code>v=diag(M)</code>	extraction de la diagonale de M
<code>M(i, :)=[]</code>	élimine la i-ème ligne
<code>M(:, j)=[]</code>	élimine la j-ème colonne
<code>AA=A.'</code>	transposée de A
<code>AAA=A'</code>	transposée de la conjuguée de A
<code>MaxA=max(A)</code>	maximum des éléments de A
<code>minA=min(A)</code>	minimum des éléments de A
<code>s=sum(A)</code>	somme des éléments de A
<code>sl=sum(A,1)</code>	matrice ligne, somme écrasée des lignes
<code>sc=sum(A,2)</code>	matrice colonne, somme écrasée des colonnes
<code>p=prod(A)</code>	produit des éléments de A
<code>pl=prod(A,1)</code>	matrice ligne, produit "écrasé" des lignes
<code>pc=prod(A,2)</code>	matrice colonne, produit "écrasé" des colonnes

↳ Exemple(s)

```
-->A=1:10;
-->P=A'*A; disp(P);
    1.    2.    3.    4.    5.    6.    7.    8.    9.   10.
    2.    4.    6.    8.   10.   12.   14.   16.   18.   20.
    3.    6.    9.   12.   15.   18.   21.   24.   27.   30.
    4.    8.   12.   16.   20.   24.   28.   32.   36.   40.
    5.   10.   15.   20.   25.   30.   35.   40.   45.   50.
    6.   12.   18.   24.   30.   36.   42.   48.   54.   60.
    7.   14.   21.   28.   35.   42.   49.   56.   63.   70.
    8.   16.   24.   32.   40.   48.   56.   64.   72.   80.
    9.   18.   27.   36.   45.   54.   63.   72.   81.   90.
   10.   20.   30.   40.   50.   60.   70.   80.   90.  100.
```

-->■

↳ Exemple(s)

```
-->A=[0,1,1;1,0,1;0,0,1];
-->d=det(A);
-->p=poly(A,"x");
-->s=spec(A);
-->k=kernel(A);; K=kernel(A-eye(3,3));i=inv(A),A3=A^3;B3=inv(A3);
i =
    0.    1.   - 1.
    1.    0.   - 1.
    0.    0.    1.

-->disp(d);disp(s);disp(p);disp(k);disp(K);disp(A3);disp(B3);

- 1.

    1.
- 1.
    1.

    1 - x - x2 + x3

[]

0.7071068
0.7071068
- 2.664E-16

    0.    1.    3.
    1.    0.    3.
    0.    0.    1.

    0.    1.   - 3.
    1.    0.   - 3.
    0.    0.    1.

-->□
```

Les chaînes de caractères

Les chaînes de caractères sont des suites de caractères `ascii`, délimitées par les caractères apostrophes « ' » ou les guillemets anglo-saxons « " » (qui sont équivalents). Pour insérer dans une phrase les caractères 'apostrophe' ou des "guillemets", il faut, pour les différencier des délimiteurs, les faire précéder d'un indicateur (à nouveau ' ou ").

```
a="l'"examen final"; disp(a)
```

La **concaténation** ou la fusion de chaînes de caractères est effectuée par l'opérateur « + » .

```
b=" est fini ."; c=a+b; disp(c);
```

La fonction `length` qui renvoie le nombre de caractères de la chaîne.

Création de matrices de chaînes de caractères

```
a="bonjour";  
b="au revoir";  
c=[a,b];  
d=[a;b];  
e=[c;c];
```

Exemple(s)

```
a="Bonjour";  
disp(a); // disp (display) on visualise « a » sur l'écran  
b="L'"ensemble E" + ", est fini....";  
c= a+b;  
disp(c);  
  
a=1 :120;  
b=ascii(a); disp(b);  
c=ascii(b); disp(c);  
  
t=1/3;  
b=string(t); disp(b);  
c=ascii(b); disp(c);  
u=ascii(c); disp(u);  
l=length(c); disp(l);  
ll=length(u);disp(ll);
```

Opérations sur les chaînes de caractères

- La fonction `« length() »` appliquée à une chaîne de caractères renvoie le nombre de caractères de celle-ci.

La fonction concaténation est représentée par le symbole `« + »`.

- La fonction `« string() »` convertit une valeur numérique en une chaîne de caractères.

```
-->a=string(3+2);  
-->b=string(1/3);  
-->c=a+b; disp(c)
```

- La fonction `« evstr() »` convertit une chaîne de caractères en son évaluation Scilab .

```
--> a="sqrt(3)/2"; disp(a);  
--> b=evstr(a); disp(b)
```

- La fonction `« part() »` permet d'extraire une sous-chaîne grâce à un indice ou un tableau d'indices.

```
--> a=part("abcdefgh",2); disp(a);  
--> a=part("abcdefgh",[1,2,6]); disp(a);
```

- La fonction `« strindex() »` permet de rechercher une chaîne à l'intérieur d'une autre chaîne. La fonction fournit les indices de début des occurrences de la chaîne recherchée.

```
--> a=strindex("Mississippi","ss"); disp(a);  
--> a=strindex("Mississippi","i"); disp(a);
```

Définition d'un caractère Scilab

Scilab possède sa propre convention de représentation des caractères. Il est possible de connaître cette représentation au moyen de la fonction `« str2code() »`.

```
--> a=str2code("abcd"); disp(a);
```

Réciproquement, il est possible de synthétiser une chaîne de caractères à partir d'un tableau contenant les codes des caractères que l'on désire obtenir.

```
--> a=code2str([10,11,12]); disp(a);
```

D'une manière plus générale, on pourra utiliser cette représentation d'une chaîne de caractères chaque fois que l'on désirera manipuler ou transformer cette chaîne. Ainsi on remarquera que les lettres minuscules sont représentées par les entiers consécutifs de 10 à 35 et que les majuscules sont représentées par l'opposé de ces mêmes codes.

```
--> a=str2code("aA bB cC"); disp(a);
```

Définition d'un caractère ASCII

La représentation ASCII (American Standard Code for Information Interchange) comporte 128 caractères numérotés de 0 à 127. On utilise en Europe une extension de ce code nommé l'**ISO-latin** qui comporte 256 caractères. Les codes de 192 à 255 contiennent la plupart des caractères spécifiques des langues indo-européennes (accents, etc)

Scilab permet la conversion vers l'ASCII et réciproquement au moyen de la fonction `ascii`.

- La fonction `readc_()` permet la lecture d'une chaîne de caractère donnée au clavier.

↳ Exemple(s)

```
--> c="Oui" ;
--> while (c=="Oui") do
--> disp(" Voulez vous continuer? (Oui / Non )");
--> c=readc_();
--> end;
```

Une application à la cryptographie

On se propose de construire un code secret aléatoire permettant de communiquer un message crypté à un correspondant qui pourra le décrypter à l'aide d'une clef Scilab .

Le cryptage se fait sur les caractères Scilab dont le code va de -35 à $+60$. En voici un exemple.

Le codage consiste à permuter aléatoirement ces codes grâce à la fonction Scilab `rand` qui donne, à volonté, une suite de nombres aléatoires partant d'un germe fixé à l'avance (ici `rand("seed",0)`).

On donne donc au correspondant, par une voie secrète, uniquement trois valeurs de base : -35 , $+60$ et le germe 0. (Dans le message on ne mentionne surtout pas Scilab , qui est implicite).

La permutation aléatoire construite par Scilab est alors la même pour l'émetteur et le récepteur.

Le message crypter par l'émetteur en utilisant les fonctions `clef` et `cod`, définie ci-dessous, est envoyé en clair (tout le monde peut le lire).

Il est décrypté par le récepteur en utilisant les fonctions inverse `invclef` et `decod`. (On utilise ici le concept de fonction qui sera vu dans la suite).

```
clear ;
B=-35 :60 ; //alphabet de départ
rand("seed",0) ;
l=length(B) ;
A=rand(1 :l) ;
[s,S]=sort(A) ;
C=B(S) ;// alphabet d'arrivée

function y=clef(x)
b=str2code(x) ;
f=find(B==b) ;
g=C(f) ;
```

```

y=code2str(g)
endfunction

function y=invclef(x)
b=str2code(x);
f=find(C==b);
g=B(f);
y=code2str(g)
endfunction

function y=cod(t)
R=[]; ll=length(t);
for i=1 :ll
u=part(t,i);
R=R+clef(u);
end;
y=R
endfunction

function y=decod(x)
IR=[]; ll=length(x);
for i=1 :ll
u=part(x,i);
IR=IR+invclef(u);
end;
y=IR;
endfunction;

//Texte secret à crypter par l'émetteur. On peut le changer à volonté
text1=" Bonjour, aujourd''hui je me sens tres bien, ";
text2="car demain c''est les vacances! Je pars ...!!.";
text=text1+text2;

u=cod(text); // Texte crypter envoyé et reçu en clair
v=decod(u); // Texte décrypté par le récepteur

// Visualisation graphique
clf;
xselect()
plot2d(0,0,-6)
x=2*pi*(0 :0.01 :1);
plot2d(2*sin(x),2*cos(x),2)
a=gca();
a.isoview="on";
xtitle(" MESSAGE ULTRA-SECRET");
xstring(-2,0.7, u);
xstring(-0.4,1.5, " Voici le message!");
xstring(-0.7,0.4, " Agrandir au maximum la fenetre graphique et?");
xstring(-0.8,0.2, " Cliquer sur le triangle central pour obtenir la solution.");
xclick();
xstring(-1.3,-0.5, v);

```

Les Booléens

Les constantes et variables booléennes des tableaux sont des tableaux composés de :

données booléennes	Vraie	Faux
Instructions Scilab	%t	%f

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

Evaluation et opérations de comparaisons

Relations mathématiques	Instructions Scilab
égalité : =	==
différent : \neq	<> ou ~=
inférieur strict : <	<
supérieur strict : >	>
inférieur ou égal : \leq	<=
supérieur ou égale : \geq	>=
Opérateur logiques	Instructions Scilab
conjonction : et	&
disjonction : ou	
négation : non	~

↳ Exemple(s)

```
--> x=1 :10 ;
--> y=(x<5)
--> z=1*y // conversion automatique en scalaire
--> w=bool2s(y) // conversion explicite
--> u=x(y)

--> a=[0,1,0,1,0,1,0,1] ;
--> b=[0,1,0,1,0,1,0,1] ;
--> a==b

--> a=' bonjour ' ;
--> b=' au revoir ' ;
--> a+b
--> [a,b]
--> c=[a;b] ;
--> v=(c==a) // le premier signe "=" est une affectation. Le deuxième "==" est un test.
--> c(v)
```

Création de matrices de booléens

De la même façon que précédemment on peut construire des matrices de booléens et effectuer des opérations matricielles booléennes.

```
Tab1=[%T,%F;%T,%T];
Tab2=[%T,%F;%T,%F];
Tab3=(Tab2==Tab1);

V=[1 :5;5 :-1 :1];
T=(V<2);
U=T';
disp(U)
```

On dispose, de plus, de plusieurs fonctions primitives de sélection Scilab associées aux booléens.

La fonction : `find()`

Cette fonction permet de sélectionner les indices des éléments %T d'une matrice donnée possédant une propriété donnée u.

```
a=find(u==2) // sélectionne tous les éléments de u égaux à 2.
b=find(u==2, n)// sélectionne au plus n éléments égaux à 2.
c=a(b) // renvoie un vecteur colonne composé des éléments de u égaux à 2.
```

Cette fonction est alors utilisée pour extraire ou insérer des éléments particuliers d'une matrice donnée.

↪ Exemple(s)

```
A=rand(1,20);
w=find(A<0.4);
A(w)
w=find(A>100)
B=rand(1,20);
w=find(B<0.4,2) // au plus deux valeurs sont retournées
w=find(B<0.4)
B(w)=0; // remplace dans B tous les éléments < 0.4 par 0
```

La fonction dsearch

Il s'agit d'une fonction proche de `find` permettant de sélectionner des éléments d'une matrice. On pourra consulter l'aide.

La fonction tabul

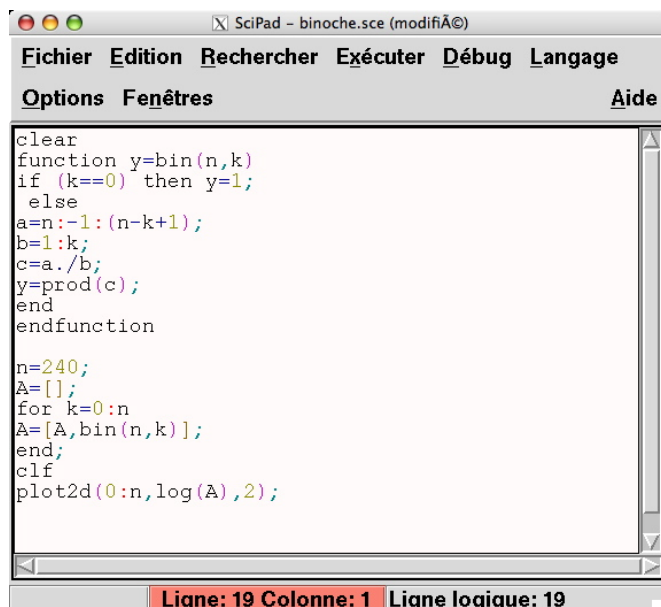
Il s'agit d'une fonction donnant le nombre d'occurrences des éléments d'une matrice. On pourra consulter l'aide.

Eléments de programmation

Script d'exécution

- Un script d'exécution est un **fichier texte** (Ascii) ou **fichier source**, est un programme contenant une suite d'instructions **Scilab** destiné à être compilé par le logiciel. Le choix de l'éditeur de texte est libre. Il existe, dans **Scilab**, un éditeur intégré au logiciel, **Scipad**, accessible par le menu **Editor** de la fenêtre principale comme il a été indiqué précédemment.
- Le fichier source doit se terminer par un retour de chariot, sinon la dernière ligne ne sera pas interprétée.
- On doit enregistrer (sauver) les nouvelles modifications de celui-ci avant chaque exécution.
- Le nom du fichier source doit avoir l'extension **.sce** (l'extension **.sci** sera réservée aux scripts de fonctions).
- Une fois le fichier enregistré (**save**) dans un dossier ou un répertoire approprié, il pourra être exécuté en utilisant le menu "File" → "File Operations" → "Exec", etc.
- On peut rappeler un fichier déjà exécuter selon le mode ci-dessus, en utilisant les flèches directionnelles de rappel dans le fenêtre principale de **Scilab**.

⚡ Exemple(s)

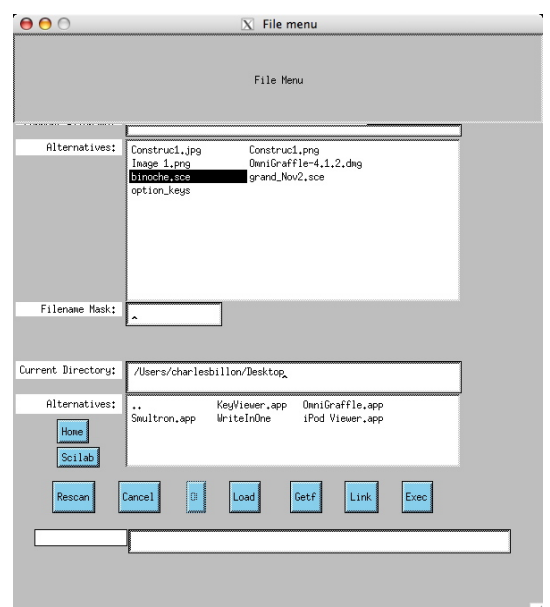


The screenshot shows the SciPad editor window titled "SciPad - binoche.sce (modifié)". The menu bar includes "Fichier", "Edition", "Rechercher", "Exécuter", "Débug", and "Langage". Below the menu bar are "Options Fenêtres" and "Aide". The main text area contains the following Scilab code:

```
clear
function y=bin(n,k)
if (k==0) then y=1;
else
a=n:-1:(n-k+1);
b=1:k;
c=a./b;
y=prod(c);
end
endfunction

n=240;
A=[];
for k=0:n
A=[A,bin(n,k)];
end;
clf
plot2d(0:n,log(A),2);
```

At the bottom of the window, a status bar indicates "Ligne: 19 Colonne: 1 Ligne logique: 19".



Outils de programmation



- Un **script** consiste en l'édition d'une succession de lignes d'instructions séparées par des « retour-chariot ».
- Chaque ligne peut comporter plusieurs instructions séparées par les signes de séparations et de ponctuations habituels (virgule, point-virgule, espace, crochets, etc) avec les valeurs spécifiques.
- La rédaction et la mise en page du **script** doivent être les plus claires possibles et comporter les commentaires nécessaires à la compréhension des objets proposés. Les noms des variables seront le plus explicite possible.

⚡ Exemple(s)

```
a=2; b=3; // cotés du triangle rectangle
c2=a^2+b^2;
c=sqrt(c2); // hypothénuse
disp(c);
```

- On évitera, dans l'écriture d'un script, l'emploi toléré dans certaines conditions d'un espace à la place d'une virgule. Ceci évitera des erreurs de compilation difficiles à corriger.
- Un script **Scilab** devra débiter par l'instruction « **clear ;** » qui efface la mémoire temporaire pour éviter certains bugs difficilement repérables.
- On pourra inhiber globalement l'affichage intempestif des calculs intermédiaires en déclarant ensuite l'instruction « **mode(-1)** ».
- On pourra tester les différents modes en tête d'un **script** : **mode(0)**, **mode(1)** et **mode(-1)**

Instructions conditionnelles

L'idée de l'instruction conditionnelle est de soumettre l'exécution d'une séquence d'instructions à la réussite (ou l'échec) d'une certaine condition.

L'instruction **if**

l'instruction **if** permet l'exécution conditionnelle de séquences d'instruction. Sa syntaxe la plus simple est :

```
if expression then instructions end
```

- Lorsque "*expression*" booléenne est vraie (resp. fausse), la séquence "*instructions*" est (resp. n'est pas) exécutée. On notera que les instructions terminées par une virgule seront affichées (imprimée), celles terminées par un point-virgule ne seront pas affichées.
- L'*expression* possède une valeur booléenne. Elle peut être la combinaison de plusieurs expressions dans l'algèbre des booléens.

↪ Exemple(s)

```
a=2; if (a==2) then b=2+4, end ;
disp(b) ;
c=rand() ; if (c<=2) then c=2, end ;
disp(c) ;
```

Il est possible de donner une séquence alternative d'instructions à exécuter lorsque la condition test est fausse. Cette instruction est introduite par le mot clef : **else**

```
if expression then instructions1 else instructions1 end
```

Lorsque "*expression*" est vraie "*instructions1*" est exécutée, dans le cas contraire "*instructions2*" est exécutée.

Remarque : Il est possible de remplacer des imbrications conditionnelles telles que :

```
if condition1 then actions1
else
    if condition2 then actions2
else
    if condition3 then actions3
else
    actions4
end
end
end
```

Par la forme équivalente plus rassemblée suivante :

```
if condition1 then actions1
elseif condition2 then actions2
elseif condition3 then actions3
else actions4
end
```

Cette dernière syntaxe utilise le mot-clef **elseif** et permet d'effectuer une série de tests successifs, auxquels sont associés des actions spécifiques.

↪ Exemple(s)

```
// Échange conditionnel des valeurs de deux variables
if a>b then c=a ; a=b ; b= c ; end
if a>b then a=a+b ; b=a-b ; a= a-b ; end
```

Instructions de sélection

L’instruction de sélection est très voisine de la dernière forme de l’instruction conditionnelle. Sa syntaxe est la suivante :

```
select expression0
  case expression1 then actions1
  case expression2 then actions2
  case expression3 then actions3
  else actions4
end
```

cette instruction fait appel aux mots-clefs :

`select, case, then, else, end.`

La valeur de "*expression0*" est calculée, puis comparée successivement aux valeurs de "*expression1*", "*expression2*", etc.

Dès que l’une de ces comparaisons possède la valeur *vraie*, l’action correspondante se réalise, et les autres ne sont plus testées.

Si aucune des comparaisons n’est *vraie* l’instruction suivant le mot-clef `else` est exécutée.

Instructions répétitives

Une instruction répétitive (boucle) a pour but d’exécuter plusieurs fois, de façon itérative, une instruction ou un groupe d’instructions.

Il existe en Scilab deux formes d’instructions répétitives : les boucles `for` et les boucles `while` .

L’instruction `for`

L’instruction `for` permet de répéter un groupe d’instructions, alors qu’une variable particulière (*la variable de boucle*) décrit un ensemble de valeurs.

`for variable = valeurs do instructions end`

Les mots-clefs relatifs à cette syntaxe sont : `for` , `do` , `end` .

- "*variable*" est le nom de la variable de contrôle de la boucle.

Elle est strictement locale et n’a pas d’existence en dehors de la boucle.

- "*instructions*" consiste en une ou plusieurs instructions (se terminant par une virgule ou un point-virgule avec le sens habituel)
- "*valeurs*" est la liste des valeurs que va décrire la variable de contrôle de la boucle. De manière générale cet ensemble de valeurs est un vecteur ou une matrice.

-
- Le mot clef `do` peut être présent (comme en pascal) ou omis.

⚡ Exemple(s)

```
a=1 ;
for n=1 :10 do
    a=a*n ;
end ;
disp(a)
k=1 :10 ;
a=1 ; for n=k do
    a=a*n ;
end ;
disp(a)
```

- **Attention** • Évaluer : `a=1 ; for n=1 :4 :10 do a=a*n ; end ;`

L'instruction `while`

L'instruction `while` « tant que » permet de répéter un groupe d'instructions, tant qu'une condition est réalisée.

`while expression do instructions end`

Les mots-clefs relatif à cette syntaxe sont donc `while` , `do` , `end` .

- "*expression*" doit fournir un résultat de type booléen.
- "*instructions*" consiste en une ou plusieurs instructions (se terminant par une virgule ou un point-virgule avec le sens habituel)

⚡ Exemple(s)

```
a=1 ; n=10 ;
while n>0 do
    a=a*n ;
    n=n-1 ;
end
disp(a) ;
```

L'instruction `while` admet une autre forme faisant intervenir le mot-clef `else` . Elle permet de préciser une ou plusieurs instructions qui doivent être exécutées lorsque, en fin de boucle la condition testée devient fausse.

↪ Exemple(s)

```
i=10 ;
while i>0 do
    i=i-3 ;
else i
end
```

L'instruction `break`

Utilisée à l'intérieur d'une boucle `for` ou `while`, l'instruction `break` permet d'interrompre la boucle et de sauter immédiatement à l'instruction qui suit la boucle dans le texte du programme en cours d'exécution.

↪ Exemple(s)

Exemple 1 :

```
i=10 ;
while i>0 do
i=i-3 ;
if i==7 then break ; end
else i
end
```

Exemple 2 :

```
k=0 ;
while 1==1, k=k+1 ;
if k > 100 then break,
end ;
end ;
disp(k) ;
```

Exemple 3 : On veut calculer approximativement : $\exp(1) = \sum_{k=0}^{\infty} \frac{1}{k!}$, la précision étant l' $\epsilon_{machine}$.

```
Nmax=10000 ; // Longueur initiale de la boucle for
e=1 ; ee=[1] ; // Initialisation de la boucle
for i=1 :Nmax
if (e>%eps) then e=1/prod(1 :i) ; ee=[ee,e] ;
else break ;
end ;
end ;
format(16) :
S=sum(ee) ;
l=length(ee) ;
disp(" La valeur trouvée de < e > est : " +string( S ))
disp("Taille effective de la somme : "+ string( l))
clf ;
plot2d(0 :(l-1),cumsum(ee),2) ;
xtitle(" Approximation de e : " +string(S)+ ". e machine : "+string(%e))
```



ATTENTION

On a remarquer que **Scilab** est optimisé pour le calcul numérique matriciel. De nombreux algorithmes faisant appel à des tests ou à des boucles, très gourmands en temps de calcul, peuvent trouver leur équivalents en utilisant les propriétés des opérations matricielles scalaires ou booléennes. Voici quelques exemples très simples.

⚡ Exemple(s)

- Calcul de factoriel n .

☞ Avec boucle :

```
n=20 ;  
a=1 ;  
for i=1 :n do  
a=a*i ;  
end ;
```

♡ ♡ ♡ Sans boucle : `n=20 ; a=prod(1 :n)`

- Construire une table de multiplication : $T_{i,j} = i \times j$, $1 \leq i, j \leq 20$.

☞ Avec boucle :

```
n=20 ;  
T=[] ;  
for i=1 :n do  
for j=1 :n do  
T(i,j)=i*j ;  
end ;  
end ;
```

♡ ♡ ♡ Sans boucle : `n=20 ; t=1 :n ; T=t'*t`

- Trouver le nombre de piles obtenu après le lancer de 20 pièces de monnaie bien équilibrées.

☞ Avec boucle :

```
n=20 ;  
a=0 ;  
for i=1 :n do  
if rand() $<$ 0.5 then a=a+1 : end ;  
end ;  
disp(a)
```

♡ ♡ ♡ Sans boucle : `n=20 ; A=sum(rand(1,n) $<$ 0.5) ; disp(A)`

Fonctions utilisateur

- L'un des principaux atouts de **Scilab** est de permettre à l'utilisateur de créer de nouvelles fonctions. Ces fonctions ne sont pas des fonctions au sens mathématique mais plutôt des programmes autonomes qui, une fois définis et mis en mémoire, peuvent être utilisés à volonté. Ces fonctions, écrites dans la syntaxe **Scilab**, peuvent avoir des paramètres d'entrées, des sorties, faire référence à des données externes et à d'autres fonctions ou procédures. Les fonctions utilisateur viennent compléter la vaste bibliothèque de fonctions prédéfinies **Scilab** ou fonction primitives **Scilab**. La liste en est donnée dans le fichier « aide ».

- Une fonction peut être insérée à l'intérieur du programme principal (**fichier.sce**) et utilisé au fil du programme.

- Le fichier peut également être sauvé et mise en mémoire dans un fichier **MesFonctions.sci** (on s'interdira les noms pris par les fonctions primitives).

- La structure utilisée est :

```
y=function( paramètres );  
    instructions;  
    instructions;  
endfunction
```

⚡ Exemple(s)

- **fact.sci**
function a=fact(n)
//factoriel de n
a=1;
for x=1 :n do
 a=a*x;
end
endfunction // « endfunction » en un seul mot

- **sinuscard.sci** (par exemple)
function x=sinuscard(t)
// f(x)=sin(x)/x , x ≠ 0, f(0)=1
// t peut être un vecteur de réels
// x est alors un vecteur
a=(t==0)
t(a)=1
x=sin(t)./t
x(a)=1
endfunction // « endfunction » en un seul mot

Chargement d'une fonction

Une fonction est chargée par l'opération `getf(' ../chemin/répertoire/fact.sci)`. On peut également la charger en utilisant le menu `file` → `getf`, puis en sélectionnant inter-activement le fichier.

Création de fonction en ligne

La procédure décrite ci-dessus est celle qui doit être systématiquement utilisée pour la définition de fonctions dans le cadre du développement d'un programme `Scilab`. Il existe une autre procédure de définition de fonction, directement utilisable *en mode terminal* et qui met en œuvre la fonction `deff`.

La syntaxe est la suivante :

`deff(en-tête, corps)`

La fonction attend deux paramètres. Le premier est une chaîne de caractères qui définit l'en-tête de la nouvelle fonction. Cet en-tête obéit à la même syntaxe que les *sci.files* hormis le mot-clef `function` qui ne doit pas être utilisé. Le second paramètre représente le corps de la fonction. Il est fourni sous la forme d'un tableau de chaînes de caractères, dans lequel chaque chaîne de caractères représente une des lignes de la fonction. Le tableau peut être fourni sous la forme d'une variable, ou encore sous la forme d'une suite, placée entre crochets, de chaînes de caractères séparées par des virgules.

↪ Exemple(s)

- ```
c=['a=4', 'b=3', 'c=5', 'r=a+b*c'];
e='[r]=toto()';
deff(e,c)
u=toto()
u=19
```

- ```
deff('z=demi(x)', 'z=x/2')  
d=demi(12.5)  
d=demi(12.5+2*i)
```

- ```
deff("y=carre(x)", "y=x.^2");
a=carre(16);
disp(a);
b=carre(1 :5);
disp(b);
```

♡ ♡ ♡ Il est préférable d'utiliser la forme canonique largement plus explicite :

```
function y=carre(x)
y= x.^2
endfunction
```



---

## Autres caractéristiques des fonctions

Certaines instructions sont spécifiques aux fonctions. C'est le cas de l'instruction `return` qui permet d'interrompre le déroulement d'une fonction et de revenir au programme ayant fait appel à cette fonction.

### ⚡ Exemple(s)

```
function z=inverse(x)
if x==0 then
z=%inf
return
end
z=1/x
endfunction
```

#### • Variables locales

On notera que toutes les variables utilisées dans le corps d'une fonction sont **locales** (symboles muets). Il n'est pas nécessaire de les déclarer dans le corps du programme principal. En contrepartie, il n'est pas possible de les modifier ni de les appeler à partir du programme principal où se situent les variables **globales**.

#### • Impression implicite

Le système n'effectue pas, pour les instructions ne se terminant pas par un point-virgule, l'impression implicite du résultat d'une expression interne à une fonction. Les diverses validations de fin d'instructions (virgule, point-virgule, passage à la ligne) peuvent être considérées comme équivalentes. (`mode(-1)`).

## Récurtivité

Les fonctions Scilab peuvent faire appel à d'autres fonctions, y compris à elles-mêmes.

### ⚡ Exemple(s)

```
function r=facto(n)
if n==0 then
r=1
else
r=n*facto(n-1)
end
endfunction
```

---

## Quelques autres caractéristiques des fonctions

### Exemple 1

```
function [u,v]=sincos(a)
u=sin(a)
v=cos(a)
endfunction
```

Voici deux exemples d'appels :

```
--> [x,y]=sincos(%pi/6)
v=0.8660254
u=0.5

--> k=sincos(%pi/6)
k= 0.5
```

**La règle est la suivante** : si une fonction rend  $n$  résultats et que, lors de l'appel, ces résultats ne affectés qu'à  $p$  variables (avec  $p < n$ ) seuls les  $p$  premiers résultats de l'appels sont retenus.

```
--> def(' [a,b,c,d]=essai(x)', 'a=x,b;2*x;c=x^ 2,s=sqrt(x)')
--> essais(5)
ans=5.

--> [x,y]=essai(5)
y=10.
x=5.

--> [x,y,u,v]=essais(5)
v=2.2336068
u=25.
y=10
x=5.
```

### Exemple 2

```
• function [u,v]=sincos(a)
k=argn()
u=sin(a)
disp('sin('+string(a)+') = '+string(u))
if k==2 then
disp(' Je calcul les deux resultats ')
v=cos(a)
disp('cos('+string(a)+') = '+string(v))
end
endfunction
```

---

### Exemple 3

```
u=-20 :0.01 :20 ;
function y=sinuscard(x)
t=find(abs(x)>1E-6) ; // On sélectionne les indices "i" tels que $|x_i| > 0.000001$
y=ones(x) ; // On définit le vecteur y tel que $y_j = 1$ pour tout j
y(t)=sin(x(t))./x(t) ; // $y_i = \frac{\sin(x_i)}{x_i}$, seulement pour les i sélectionnés
endfunction
clf() ;
plot2d(u',sinuscard(u'),2) ;
```

A rapprocher de :

```
u=-20 :0.01 :20 ;
function y=sinuscardi(x)
t=(abs(x)>1E-6) ; // On sélectionne les indices "i" tels que $|x_i| > 0.000001$
y=ones(x) ; // On définit le vecteur y tel que $y_j = 1$ pour tout j
y(t)=sin(x(t))./x(t) ; // $y_i = \frac{\sin(x_i)}{x_i}$, seulement pour les i sélectionnés
endfunction
clf() ;
plot2d(u',sinuscardi(u'),2)
```

### Exemple 4

**Un jeu.** On désire donner la valeur « - », au hasard, (pile ou face) à chacune des lettres d'un mot.

```
y=input('Entrer un mot, une chaine de caracteres.',string) ;
//y='Anticonstitutionnellement' ;
disp(' Le mot initial est en memoire ')
//disp(y)
disp(' Il s''agit d''un mot de '+ string(length(y))+ ' lettres')
Y=ascii(y) ;
x=round(rand(1,length(Y))) ;
t=find(x==0) ;
nont=find(x<>0) ;
Z=Y(nont) ;
disp(' Mot raccourci : '+ascii(Z))
a=ascii('-') ;
W=Y ;
W(t)=a ;
w=ascii(W) ;
disp(' Retrouver le mot initial...a partir de : '+w)
```

---

## La fonction `sort()`

La syntaxe de cette fonction est indiquée dans l'aide en ligne `Scilab`. Cette fonction ordonne par valeurs décroissantes les éléments d'un vecteur scalaire. On donnera des exemples sous la forme de construction de jeux mathématiques qui utilisent la construction de permutations aléatoires.

### Exemple(s)

#### Exemple 1

**Un jeu.** On désire effectuer une permutation aléatoire des lettres d'un mot donné et tenter de le retrouver le mot initial.

```
// Entrer un mot i.e. un chaine de caracteres. par exemple :
y='Anticonstitutionnellement';
disp(' Le mot initial est en memoire... ')
//disp(y)
Y=ascii(y);
long=length(Y);
[u,v]=sort(rand(1,long));
Z=Y(v);
disp(' Mot mystere : '+ascii(Z))
```

#### Exemple 2

##### Monsieur Jourdain s'entraîne à scilab

Le Bourgeois Gentilhomme lors de sa formation s'est initié au logiciel `scilab` et a découvert les trésors d'intelligence contenus dans l'aide en ligne. Il rédige le poème suivant, préparé pour sa bien-aimée :

« *Marquise Vos beaux yeux Me font Mourir D'Amour .* »

Mais il veut le mieux tourner.

Son maître de philosophie lui conseille la technique de la permutation aléatoire et lui confie le script suivant :

```
a=[];
a(1)=' Marquise';
a(2)=' Vos beaux yeux';
a(3)=' Me font';
a(4)=' Mourir';
a(5)=' D'Amour';
//b=strcat(a)
disp('.....')
for i=1 :10
 c=rand(1,5,'u'); // tableau de cinq nombres au hasard
 [d,e]=sort(c); // On ordonne
 disp(strcat(a(e))+'.')
end
disp(' Il y a cent vingt poèmes possibles. (5!)')
```

---

## La fonction `max()`

Cette fonction permet de donner le maximum des valeurs d'un tableau et de préciser l'indice de l'élément maximum du tableau.

### Exemple 1

```
A=rand(2,10);
[u,v]=max(A);
disp(u);
disp(v);
m=A(v);
disp(m);
```

### Exemple 2

```
A=rand(10000,10);
B=sum(A,'c');
scf(1);
clf(1);
subplot(1,3,1)
histplot(20,B,2);
[M,I]=max(A,'c');
subplot(1,3,2);
histplot(20,M,5)
[m,J]=min(A,'c');
subplot(1,3,3);
histplot(20,m,7)
```

---

## Polynômes et fractions rationnelles et Scilab

### Polynômes définis par leurs coefficients

En `scilab` un polynôme peut être défini par la liste ordonnée de ses coefficients, exprimé sous la forme d'un vecteur. Les coefficients sont alors donnés dans l'ordre des degrés croissants partant de l'indice 0. L'inconnue ou l'indéterminée est par exemple  $x$  est appelée la "graine" ("seed") du polynôme.

#### ⚡ Exemple(s)

- Evaluer les commandes :

```
p1=poly([1,2,3],"x","coeff")
q1=poly([3;5;1],"x","c") // (c pour coeff)
```

Il existe une autre façon de procéder pour définir ces polynômes à partir de ses coefficients.

- On pourra évaluer les commandes :

```
x=poly(0,"s","c"); // indéterminée "s"
disp(x)
type(x)
typeof(x)
p2=poly([1,2,3],"x","c");
disp(p2);
type(p2); // (les polynômes sont de type 2 ou polynomial)
typeof(p2)
```

- Inversement, la fonction `coeff(p)` donne les coefficients du polynôme  $p$ .

```
t=coeff(p2);
```

### Polynômes définis par leurs racines

Un polynôme peut être à partir de ses racines. Celles-ci sont définies par un vecteur  $V$ ,

#### ⚡ Exemple(s)

```
v=[1,2,3];
p=poly(v,"s","roots");
q=poly(ones(1,3),"s","r"); // (r pour roots)
r=poly(ones(1,3),"s","c");
disp(p);
disp(q);
disp(r);
disp(p+q+r);
t=coeff(p+q+r);
disp(t);
```

L'utilisateur a le choix de nommer la graine symboliquement  $s$ ,  $x$ ,  $y$  ou  $z$ . En revanche la composition de polynômes (tableaux, sommes, produits,...) nécessite l'emploi de polynômes possédant la même graine.

---

## Polynômes définis par leur expression algébrique

- On définit dans un premier temps l'inconnue (ou indéterminée) appelée "graine" puis on déclare le polynôme directement sous sa forme algébrique.

```
x=poly(0,"t");
p=1+2*x^2+4*x^3-5*x^4;
varn(p)
```

- `scilab` possède une variable pré-définie `%s` qui représente le monôme de degré 1 le germe étant alors nécessairement `s`.

```
p=1+2*s^2+4*s^3-5*s^4;
disp(p);
varn(p)
```

## Polynômes définis comme polynôme caractéristique d'une matrice

Soit  $M$  une matrice carrée. Le polynôme caractéristique de  $M$  est défini comme de déterminant :

$$\det(xI - M) = p(x)$$

```
M=[1,2;3,4];
p=poly(M,"x");
disp(p);
type(p);
typeof(p)
```

## Opérations sur les polynômes

On peut ajouter, retrancher, multiplier les polynômes, ..., pourvu qu'ils aient la même graine.

`scilab` possède quelques fonctions dédiées au calcul sur les polynômes.

|                          |                                                              |
|--------------------------|--------------------------------------------------------------|
| <code>degree(p)</code>   | renvoie le degré du polynôme $p$                             |
| <code>derivat(p)</code>  | procède à la dérivation du polynôme $p$                      |
| <code>coeff(p)</code>    | donne les coefficients du polynôme $p$ , par ordre croissant |
| <code>roots(p)</code>    | calcule les racines du polynôme $p$                          |
| <code>pdiv(p,q)</code>   | division euclidienne                                         |
| <code>ldiv(p,q)</code>   | division suivant les puissances croissantes                  |
| <code>gcm(p,q)</code>    | plus grand commun multiple des polynômes $p$ et $q$          |
| <code>bezout(p,q)</code> | plus petit commun multiple des polynômes $p$ et $q$          |
| <code>lcm(p,q)</code>    | plus petit commun multiple des polynômes $p$ et $q$          |
| <code>factors(p)</code>  | factorisation                                                |

*On pourra utiliser l'aide de Scilab pour préciser la syntaxe de ces fonctions.*

---

**⚡ Exemple(s)**

```
M=[1,-2,5,-4];
p=poly(M,"s","c");
disp(p);
x=%s;
q=1+2*x;
disp(q);
[r,d]=pdiv(p,q)
```

## Fractions rationnelles

Il s'agit du rapport formel de deux polynômes de même graine :

**⚡ Exemple(s)**

```
M=[1,-2,5,-4];
p=poly(M,"x","c");
disp(p);
x=poly(0,"x");
q=1-x;
disp(q);
u=p/q;
disp(u);
v=q/p; disp(v);
w=u*v; disp(w);
```

### Fonctions dédiées au calcul sur les fractions rationnelles.

- `derivat(u)` effectue la dérivée formelle de  $u$ .

```
M=[1,-2,5,-4];
p=poly(M,"x","c");
x=poly(0,"x");
q=1-x;v=q/p;
disp(v);
du=derivat(u)
```

- `simp(u)` simplifie la fraction rationnelle.

```
M=[1,-2,5,-4];
p=poly(M,"x","c");
x=poly(0,"x");
q=1-x;v=q/p;
sv=simp(v);
disp(sv)
```



---

## Valeurs prises par un polynôme ou une fraction rationnelle.

Les polynômes et les fractions rationnelles abordés ci-dessus sont définis symboliquement. On peut expliciter les valeurs prises par ceux-ci par la fonction Scilab : `horner()`.

### ⚡ Exemple(s)

```
s=% s;
r=(5+2*s+4*s^3+5*s^4)/(3+s^4);
t=-20 :0.1 :20;
rt=horner(r,t); // évaluation de r sur les éléments de t.
clf; // graphe de r
plot2d(t,rt,2);
den=denom(r); //dénominateur de r.
disp(den);
0=roots(den) // pôles de r.
num=numer(r); // numérateur de r.
disp(num);
invrt=horner(1/r,t); // on inverse r.
plot2d(t,invrt,5); //graphe de 1/r.
dr=derivat(r);
disp(r);
```



### 🔗 Exercices

#### Exercice 1

Résoudre l'équation suivante :

$$x^3 + x^2 - x + 1 = 0$$

#### Exercice 2

Résoudre l'équation suivante :

$$x^3 - x^2 - x + 1 = 0$$

#### Exercice 3

Résoudre l'équation suivante :

$$\frac{x^3 + x^2 - x + 1}{1 + x^2} = \frac{1 + x}{1 + x + x^2}$$

#### Exercice 4

Etudier et tracer le graphe de la fraction rationnelle suivante :

$$r(x) = \frac{x^3 + x^2 - x + 1}{(x-1)(x-2)(x-3)}$$

#### Exercice 5

Résoudre l'équation suivante :

$$1 + x + x^3 = (1 + x)e^x$$

---

## Scilab et probabilités élémentaires

### La fonction scilab `rand()`

Lorsqu'on lance au hasard une pièce bien équilibrée à pile ou face, on sous-entend que la probabilité de tomber sur pile est égale à celle de tomber sur face.

Si on la lance  $n$  fois, on disposera d'une suite de  $n$  résultats chacun égal à P ou F. Et, si l'on n'a pas triché, les résultats sont "indépendants" les uns des autres.

Pour simuler une telle suite on dispose de la fonction scilab `rand`.

#### ↪ Exemple(s)

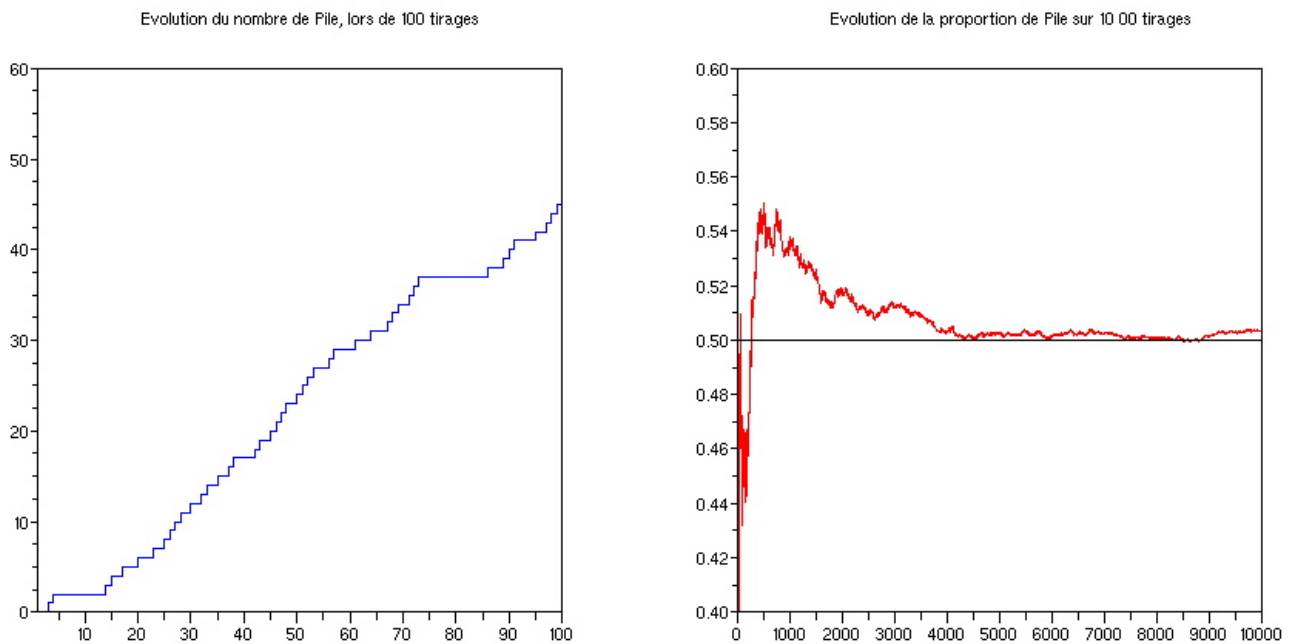
```
n=10; // n quelconque
rand('uniform');
PF=round(rand(1,n)) // Entier le plus proche.
SumP=sum(PF) // nombre de Pile obtenu (le nb de Face obtenu est donc n-SumP).
ProportionP=SumP/n // Proportion de Pile obtenu.
```

On tire une suite de  $n$  nombres aléatoires réels  $x_i$  entre 0 et 1 (aléatoire signifie en fait pour pseudo-aléatoire) et on pose  $PF_i = 1$  si  $x_i$  est plus proche de 1 que de 0 et  $PF_i = 0$  dans le cas contraire. La probabilité d'avoir 1 est égale à celle d'obtenir 0 et donc égale à 0.5

- Supposons que  $n$  soit très grand; que se passe-t-il ?

```
n=10000; // n fixé.
t=1 :n;
rand("uniform");
PF=round(rand(1,n));
SumP=cumsum(PF); // nombre de Pile obtenu, pour k=1,2, .., n.
ProportionP=SumP./t; // Proportion de Pile obtenu, pour k=1,2, .., n.
xset("window",1); // h=scf(1);clf(1);xselect()
subplot(1,2,1);
plot2d2(t(1 :100), SumP(1 :100), 2, rect=[1,0,100,60])
xtitle(' Evolution du nombre de Pile, lors de 100 tirages')
subplot(1,2,2);
plot2d(t, ProportionP, 5,rect=[0,0.4,n,0.6])
plot2d(t, 0.5*ones(t), 1)
xtitle('Evolution de la proportion de Pile sur 10 00 tirages')
```

La figure suivante montre l'évolution du nombre de pile obtenu et la proportion des pile en fonction du nombre de tirage.



On constate (à gauche) que le nombre de Pile augmente assez régulièrement et que la proportion de Pile à tendance à se stabiliser autour de 0.5, qui est la probabilité d'obtenir Pile lors un lancer.

Il s'agit ici d'une approche empirique du théorème fondamental de la statistique ou loi des grands nombres.

La courbe bleue (à droite) représente l'évolution de ce qui est appelé Processus de Bernoulli.

### Quelques propriétés de la fonction `rand()`

- On initialise le générateur de nombres aléatoires par la commande `rand("uniform")`, pour loi uniforme sur  $]0, 1[$ . (*Il n'y a pas plus de "chance" de tirer tel nombre plutôt que tel autre, dans  $]0, 1[$* ).
- `rand()` renvoie un nombre aléatoire de loi uniforme sur  $]0, 1[$ .
- `10*rand()` renvoie un nombre aléatoire de loi uniforme sur  $]0, 10[$ .
- L'instruction `A=rand(n,k)` renvoie une matrice  $A$ ,  $n$  lignes et  $k$  colonnes, composées de nombres aléatoires indépendants de loi uniforme sur  $]0, 1[$ .
- L'instruction `B=rand(X)` renvoie une matrice  $B$ , de même dimension que  $X$ , composées de nombres aléatoires indépendants de loi uniforme sur  $]0, 1[$ .
- On peut alors construire des programmes simulant des objets aléatoires très divers.

---

**↳ Exemple(s)**

### Simulation du lancer d'un dé non pipé.

On suppose que l'on dispose d'un dé parfait, bien symétrique et bien équilibré. Cela signifie intuitivement qu'il a autant de chance de tomber sur l'une de ces faces plutôt que sur une autre (hypothèse de symétrie). La probabilité de tomber sur l'une quelconque de ses faces est donc :  $\frac{1}{6}$ .

Ceci peut être simulé par la fonction suivante :

```
rand("u");
x=zeros(3,4);
function y= DeDe(x)
 //x est une matrice de scalaires
 rand(x);
 y=ceil(6*rand(x))
endfunction
A=DeDe(x)
```

### Simulation du lancer d'un dé pipé. Utilisation de la fonction dsearch

On suppose que le dé est pipé et avantage légèrement l'occurrence du 6. La probabilité de tomber sur  $i$  est notée  $p_i$ , pour  $i = 1, 2, \dots, 6$ .

On construit alors une fonction Scilab, nommée DePip, renvoyant le résultat simulé correspondant au vecteur théorique :  $p = (p_1, p_2, \dots, p_6)$ .

Ce résultat sera donné sous la forme d'une suite, de longueur arbitraire, de tirages (simulés) du dé pipé.

On recherchera, de plus, l'histogramme empirique des résultats du tirage, c'est-à-dire la proportion  $q_i$  des tirages donnant  $i$ , pour  $i = 1, 2, \dots, 6$ .

Finalement, une représentation graphique des résultats sera construite. Cela donne le script suivant :

```
clear;
// On change le dé pour favoriser le 6
p=ones(1,6)/6;
p($)=p($)+0.01; // $ est le dernier élément de p.
p=p/sum(p);
```

```
function y= DePip1(x, p)
 // x est un vecteur ligne de scalaires
 // p est la vecteur (p1,p2,...pn) avec pi>0 et sum(p)=1;
 pp=[0,cumsum(p)];
 X=rand(x);
 [y, occ] = dsearch(X,pp);
endfunction
```

---

```

function [y, occ]= DePip2(x, p)
 // x est un vecteur ligne scalaire
 // p est la vecteur (p1,p2,...pn), avec pi>0 et sum(p)=1;
 pp=[0,cumsum(p)] ;
 X=rand(x) ;
 [y, occ] =dsearch(X,pp) ;
endfunction

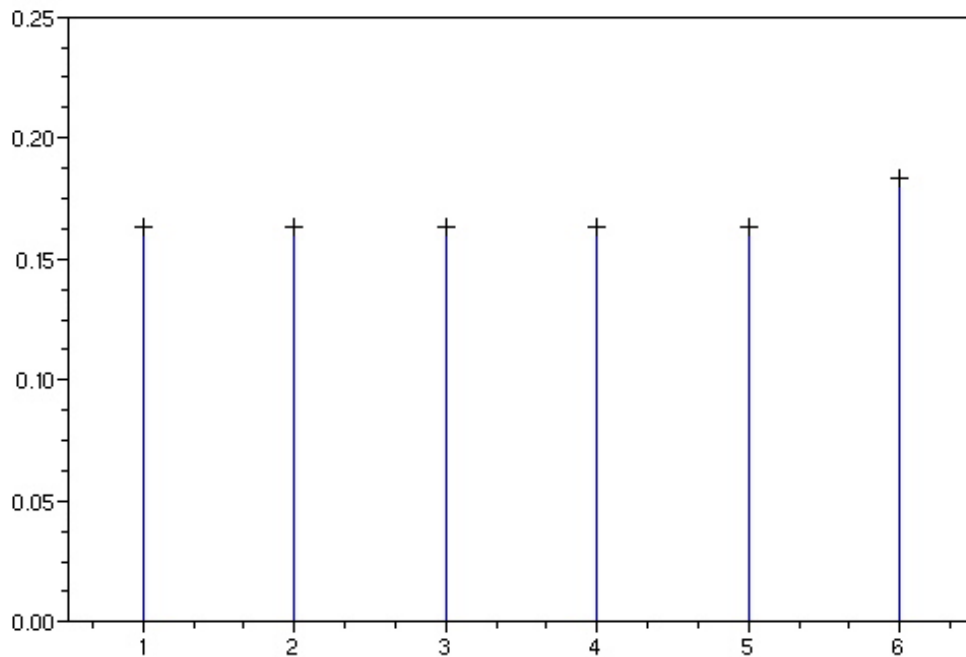
```

```

N=1000000 ;
u=1 :N ;
[a,b]=DePip2(u,p) ;
histo=b/N ;
clf()
plot2d3(1 :6,histo,2,rect=[0.5,0,6.5,0.25])
plot2d(1 :6,p,-1)
xlabel(" De pipe. En couleur, l''histogramme empirique."+ ...
"Les croix donnent l''histogramme theorique"+" Pour N = "+string(N))
xselect()

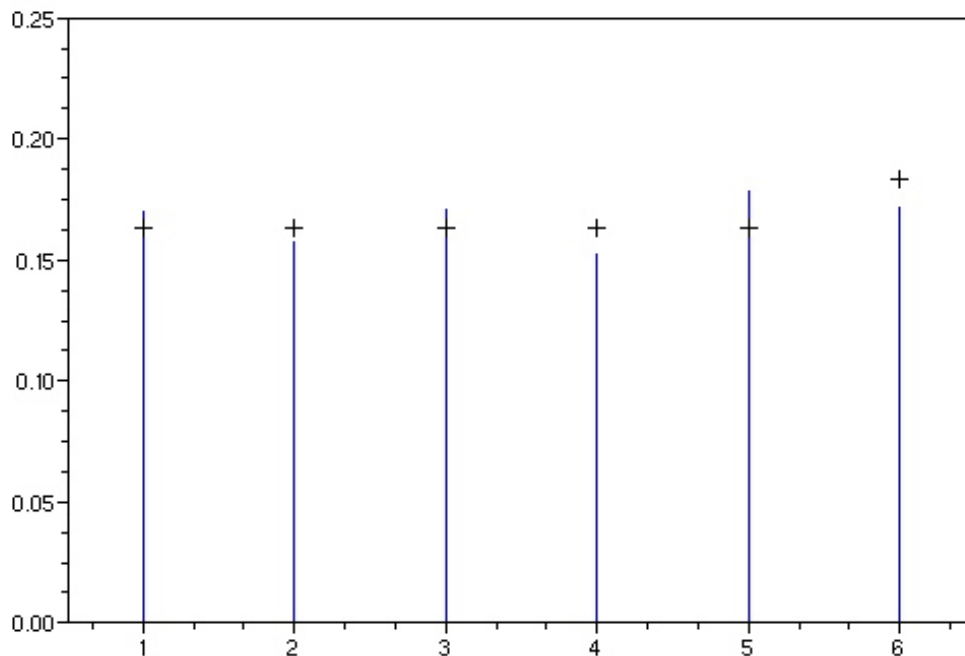
```

De pipe. En couleur, l'histogramme empirique. Les croix donnent l'histogramme theorique. Pour N = 1000000



---

De pipe. En couleur, l'histogramme empirique. Les croix donnent l'histogramme theorique. Pour N = 1000



## Générateurs de nombres aléatoire de la loi uniforme sur (0, 1)

Les réels "machine", compris entre 0 et 1, sont de la forme  $x = n2^{-p}$ ,  $n \in \{0, \dots, 2^p\}$ . Il n'y a donc pas plus  $2^p$  nombres différents sur  $[0, 1[$  et, comme ces nombres sont également espacés, le problème de donner des nombres aléatoires "réels", entre 0 et 1, se réduit à tirer des nombres entiers uniformément sur  $\{0, 1, \dots, 2^p - 1\}$ .

Les algorithmes de génération de suites de nombres  $(Y_n)_{n \geq 0}$  ont, tous, plus ou moins la forme de :

$$X_{n+1} = f(X_n), X_n \in \{0, 1, \dots, m\}, Y_n = X_n/m.$$

On en déduit plusieurs conséquences importantes :

- Il s'agit de suites parfaitement déterministes.
- Les  $Y_n$  ne sont pas indépendants.pseudo-aléatoires.
- La suite  $Y_n$  est périodique de période  $m$  au plus, parfois moins.
- Il s'agit donc de nombres **pseudo-aléatoires** et **pseudo-indépendants** simulant des nombres aléatoires indépendants.

---

Le générateur **Scilab** de nombres pseudo-aléatoires le plus simple est mis en œuvre lors de l'appel de la fonction `rand()`. Il utilise le modèle congruenciel linéaire suivant :

$$X_{n+1} = (aX_n + c) \text{ modulo } m,$$

avec  $m = 2^{31}$ ,  $a = 843314861$  et  $c = 453816693$

Ce générateur peut être considéré comme acceptable pour effectuer de simulations élémentaires. Il en existe de meilleurs (On pourra consulter par exemple l'aide pour connaître les propriétés de la fonction **Scilab** `grand`).

Pour plus d'informations concernant les enjeux liés aux générateurs de nombres aléatoires, on pourra consulter sur la toile les ouvrages de Pierre Lecuyer.

Le terme initial  $X_0$  est appelé le **germe** de la suite, fixé par défaut à 0. Ainsi, lorsque l'on appelle pour la première fois la fonction `rand()`, le premier terme récupéré est toujours :

$$u_1 = 45816693/2^{31} \approx 0.2113249$$



#### Exercices

a) Dans l'environnement **Scilab**, étudier le résultat de l'appel :

```
rand("uniform"); rand("n");
```

b) Que renvoient les instructions suivantes ?

```
1) germe=rand('seed')
```

```
2) germinit= rand('seed', 0); u=rand(1,5); v=rand(1,5);
[u,v]
```

```
3) germinit= rand('seed', 0);
[u,v]=rand(2,5);
[u,v]
```

```
4) germinit= rand('seed', 0); u=rand(1,5);
germinit= rand('seed', 0); v=rand(1,5);
disp([u,v]);
disp([u;v]);
```

```
5) d=getdate();
rand('seed',sum(d)),
[u,v]=rand(2,5)
```

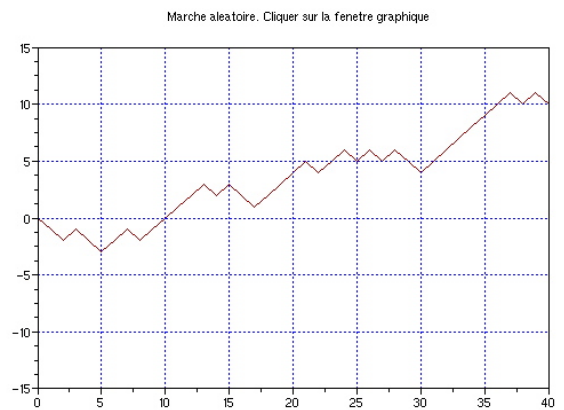
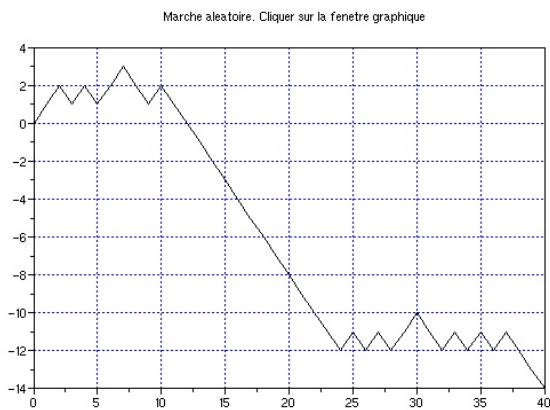
```
6) d=getdate(); rand('seed',sum(d)), u=rand(1,5);
d=getdate(); rand('seed',sum(d));
v=rand(1,5);
[u,v]
```



## Exercices

Que donne le script Scilab suivant ?

```
clear ;
h=scf(1) ;
clf(1) ;
A=gca(1) ;
A.grid=[2,2] ;
xtitle('Marche aleatoire. Cliquer sur la fenetre graphique') ;
b=[] ;
j=0 ;
l=1 :2 :33 ;
 for i=1
 n=40 ;
 j=j+1 ;
 x=rand(1,n) ;
 y=x<0.5 ;
 z=2*y-1 ;
 a=cumsum(z) ;
 a=[0,a] ;
 b(j)=a($) ;
 plot2d(0 :n,a,i) ;
 xclick() ;
 plot2d(0 :n,a,8) ;
 end ;
delete(h)
disp(' impacts finaux pour ' + string(length(l))+ ' essais : ')
disp(b) ;
```







## Exercices

Étudier le script scilab suivant :

```
clear ; xset('window',0) ; clf ;
t=0 :0.001 :1 ;
l=length(t) ;
rand("u") ;
x=rand(t) ;
y=rand(t) ;
subplot(1,2,1) ;
plot2d(x,y,-10) ; square(0,0,2,2) ;
xtitle(' Loi uniforme sur [0,1]x[0,1]')

rand("n") :
x=rand(t) ;
y=rand(t) ;
subplot(1,2,2) ;
plot2d(x,y,-10) ; xtitle(' Loi ? sur ??') ;
xselect() ;

xset('window',1) ; clf() ;
rand('n') ;
t=-1 :0.05 :1 ;
l=length(t) ;
u=rand(1,1) ;
plot3d(t',t',u,alpha=15, theta=45,flag=[5,2,3]) ;
xselect() ;

xset('window',2) ; clf() ;
rand('u') ;
u=rand(1,1) ;
square(0,0,1,1) ;
Matplot(10*u) ;
```

---

## Simulations aléatoires et histogrammes

### Tirages aléatoires.

Rappelons que la commande : `rand("uniform")` // ou `rand("u")` bascule le générateur de nombres aléatoires de Scilab en mode uniforme, ainsi les appels successifs à la fonction `rand()` fourniront des valeurs aléatoires indépendantes de même loi uniforme sur l'intervalle ouvert  $]0, 1[$ .

L'autre mode est `rand("normal")` // ou `rand("n ")` bascule le générateurs en mode "normal" pour gaussienne, centrée et réduite.

#### ⚡ Exemple(s)

Considérons la suite de commande suivantes :

```
rand("u");
aleas = rand(3,6); // ou rand(3,6,"u")
aleaBool = (aleas < 0.5);
alea01= double(aleaBool);
pm1 = 2*aleaBool - 1;
```

- `aleas` est une matrice  $3 \times 6$  à termes aléatoires,
- `aleaBool` est une matrice de booléens.
- La fonction `double` convertit les booléens en réels 1.0 pour T et 0.0 pour F.
- Finalement, la matrice `pm1` est une matrice aléatoire de +1 ou -1 avec même probabilité 0.5.

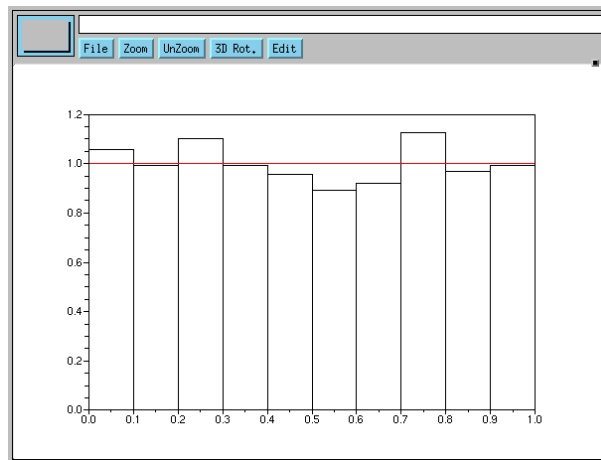
### Construction d'histogrammes

• On définit le vecteur subdivision `S` de l'intervalle  $]0, 1[$  par incréments de  $1/10$  :  
`S = 0 :0.1 :1`

• On définit un vecteur de 1600 tirages aléatoires de loi uniforme sur l'intervalle  $]0, 1[$ .  
`Donnees = rand(1, 1600); // rand("u") étant actif.`

• On trace alors l'**histogramme** de `Donnees`, relatif à la subdivision `s` en noir et de la densité uniforme en rouge

```
clf,
histplot(S, Donnees);
plot2d([0,1], [1,1], 5);
```



La fonction `histplot(S, Donnees)` prend deux arguments :

`S`, la subdivision, doit être un vecteur (ligne ou colonne) strictement croissant de,  $K = \text{length}(S)$  "piquets", encadrant  $K - 1$  intervalles :

$$S(1) < S(2) < \dots < S(K)$$

`Donnees` est un vecteur de données (en général, obtenues par tirages aléatoires).

La surface de chacun des  $K - 1$  rectangles de l'histogramme est égale à la proportion de données qui tombent dans sa largeur  $]S(k), S(k + 1)[$ .

Si toutes les données sont comprises entre les bornes extrêmes  $S(1)$  et  $S(K)$ , la surface totale de l'histogramme est égale à 1.

Ainsi, la fonction `histplot` fait deux choses :

- elle calcule les proportions
- Elle dessine les rectangles de surfaces correspondantes à l'écran (en noir par défaut).

On montre facilement, grâce à la Loi des Grands Nombres, que le graphe de l'histogramme est une bonne approximation celui de la densité de probabilité, lorsque le nombre de données `length(Donnes)` est grand.

On superpose ces deux graphes, dans une même fenêtre graphique avec des couleurs différentes.

#### ↪ Exemple(s)

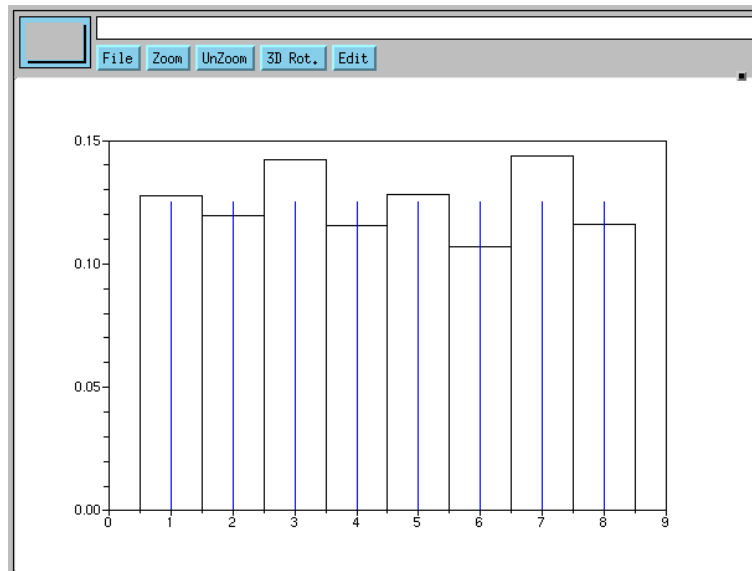
```
rand("uniform");
aleas = rand(3,6);
albool = aleas < 0.5;
double(albool);
pm1 = 2*albool - 1;
S = 0 :0.1 :1;
data = rand(1, 1600);
clf,
histplot(S, data),
plot2d([0,1], [1,1], 5)
xslect();
```

---

## Histogrammes de variables discrètes

Pour les variables aléatoires dont l'ensemble des les valeurs est un intervalle d'entiers  $E = [a, b]$ , fini, il faut adapter la subdivision  $S$ , pour que ses piquets encadrent convenablement  $E$ . On propose dans l'exemple suivant une façon élégante de représentation de l'histogramme.

```
E = 1 :8 ;
S = (E(1)-0.5) : (E($)+0.5) ;
Donnees = ceil(8*rand(1, 1600)) ; // approximation entière par excès
clf ;
histplot(s, Donnees) ;
plot2d3(E, ones(E)/8, 2) ;
//Tracés de l'histogramme en noir et de la loi uniforme sur E en bleu
```



## Simulations et histogramme de loi exponentielle

Les variables aléatoires  $X$  à valeurs réelles positives, de loi exponentielle vérifie, pour tout  $a, b \in \mathbb{R}^+$ ,

$$\mathbb{P}(X > a + b | X > a) = \mathbb{P}(X > b)$$

Cette condition est nécessaire et suffisante pour que cette loi possède une densité de la forme :

$$p_X(x) = \lambda e^{-\lambda x}, \quad x \in \mathbb{R}^+,$$

où  $\lambda = -\ln \mathbb{P}(X > 1)$  est le paramètre.

On remarque que, si une variable  $Y$  possède une loi uniforme (instruction `rand()`),  $X = -\frac{1}{\lambda} \ln(Y)$  possède une loi exponentielle de paramètre  $\lambda$ .

---

Le script suivant permet de représenter sur un même graphe la densité et l'histogramme d'une loi exponentielle de paramètre  $\lambda = 1$ .

```
// expo.sce
// Initialisation des constantes.
```

```
clear;
rand("uniform");
lambda = 1;
N = 1600;
a = 0;
b = 5;
```

```
// La densité.
```

```
function y = densExpo(para, x)
if (x < 0)
 y = 0;
else
 y = para*exp(-para*x);
end
endfunction
```

```
// La fonction donnant des nombres aléatoires de loi exponentielle, paramètre : para
```

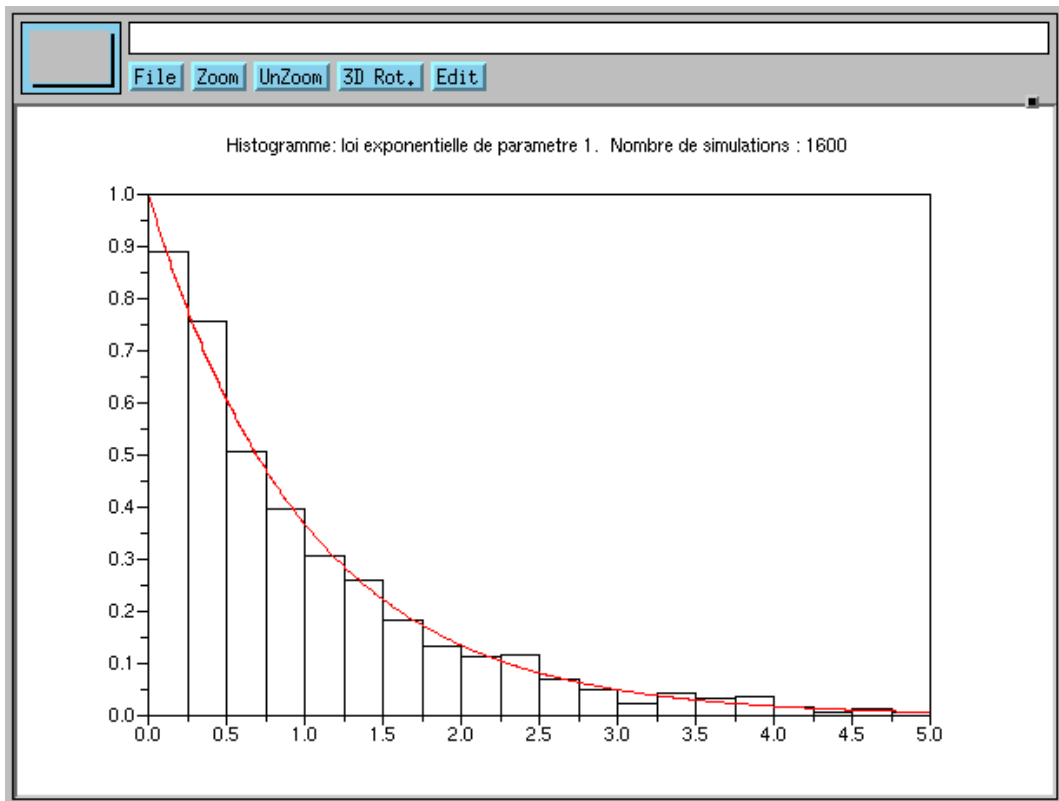
```
function X = simuExpo(para)
U = rand();
X = -log(U)/para;
endfunction
```

```
// Simulation proprement dite. Collection des nombres aléatoires
```

```
s = linspace(a, b, 21);
data = zeros(1,N);
for n = 1 :N
 data(n) = simuExpo(lambda);
end
```

```
// Représentation graphique.
```

```
x = linspace(a, b, 501);
px = zeros(x);
for i = 1 :length(x)
 px(i) = densExpo(lambda, x(i));
end
clf;
xlabel("Histogramme : loi exponentielle de parametre " + ...
string(lambda) + ". Nombre de simulations : " + string(N));
histplot(s, data);
plot2d(x, px, 5);
xselect();
```



Ce programme réalise  $N = 1600$  tirages successifs d'une variable aléatoire (v.a.) de loi exponentielle de paramètre  $\lambda = 1$ , il en trace l'histogramme en noir qui est comparé à la densité en rouge.

### Remarque à propos du script précédent

On a utilisé quelques grands principes de programmation

- Ne mettre qu'une instruction par ligne.

Les messages d'erreurs de **Scilab** étant parfois incompréhensibles, autant se faciliter le débogage en localisant les erreurs avec plus de précision et indenter harmonieusement les blocs pour rendre le code plus lisible.

- Les commentaires sont précédés de `//` sur la même ligne.

`// expo.sce` est le nom du programme.

Un programme doit être compréhensible, même pour son concepteur, dans les quelques heures ou quelque mois qui suivent.

- Ensuite viennent deux précautions, lorsqu'il s'agit d'une simulation probabiliste.

---

On doit effacer les variables et fonctions utilisateurs précédentes de la mémoire temporaire, ce qui évite des "bugs" difficilement détectables, d'autant que les messages d'erreurs de **Scilab** sont particulièrement abscons !

```
clear ;
rand("uniform") ;
```

On bascule le générateur de nombres aléatoires de **Scilab** en mode uniforme (qui est d'ailleurs le mode par défaut).

- Puis on initialise les paramètres globaux du programme, c'est ici, et seulement ici, qu'il faut intervenir pour les modifier.
- Les fonctions sont ensuite définies. `function ...endfunction`.
- Tous les blocs, conditionnels ou de boucle `if (else) while, for`, se terminent par `end`.
- On remarque, dans le script proposé, l'absence de `then`. En effet, `then` peut poser des problèmes lorsqu'il est mal placé. Donc ne pas le mettre, mais passer à la ligne après la condition qui, ici, comme en Java, est mise entre parenthèses.
- Le corps du programme vient ensuite. On procède aux N tirages à l'aide d'une boucle qui remplit le vecteur `data`.

La subdivision `s` sera utilisée pour l'histogramme.

- Il est plus efficace, en **Scilab**, d'initialiser un vecteur à sa taille finale avant remplissage, ce qui est fait pour `data` par la fonction `zeros`.

Les chaînes de caractères se concatènent par l'opérateur `+` et les scalaires sont convertis en chaînes par la fonction `string`. On place `...` (trois points horizontaux) avant le passage à ligne lorsque l'on coupe une instruction.

- La commande `xselect()` met en premier plan la fenêtre graphique, cela évite d'avoir à la rechercher dans le fouillis de l'écran...

---

## Scilab et algèbre linéaire

### Image et noyau d'une application linéaire

Soit  $u : \mathbb{R}^4 \rightarrow \mathbb{R}^5$  une application linéaire. On lui associe la matrice  $U$  définie par :

```
U=matrix(1 :20,4,5)
```

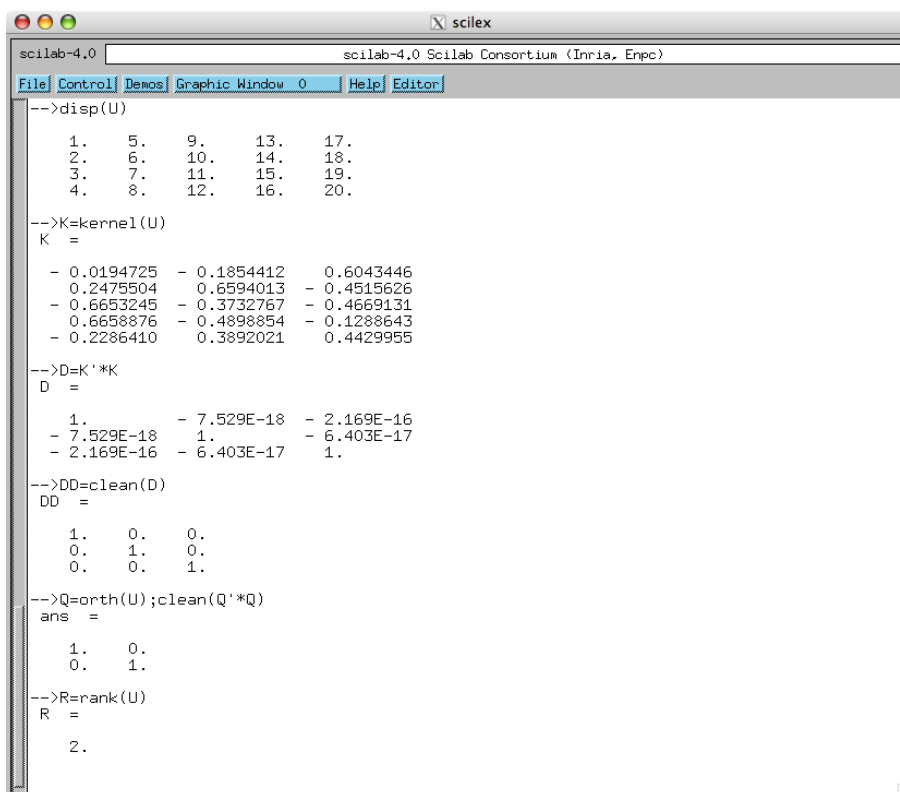
```
U =
 1. 5. 9. 13. 17.
 2. 6. 10. 14. 18.
 3. 7. 11. 15. 19.
 4. 8. 12. 16. 20.
```

Quelle sont les dimensions de son image et de son noyau ?

La fonction `orth(U)` renvoie une matrice dont les vecteurs colonnes forment une base orthogonale de l'image de  $U$ .

La fonction `kernel(U)` donne une matrice dont les vecteurs colonnes forment une base du noyau de  $U$ .

La fonction `rank(U)` renvoie le rang (numérique) de  $U$ .



```
scilab-4.0
scilab-4.0 Scilab Consortium (Inria, Epc)
File Control Demos Graphic Window 0 Help Editor
-->disp(U)
 1. 5. 9. 13. 17.
 2. 6. 10. 14. 18.
 3. 7. 11. 15. 19.
 4. 8. 12. 16. 20.
-->K=kernel(U)
K =
 - 0.0194725 - 0.1854412 0.6043446
 - 0.2475504 0.6594013 - 0.4515626
 - 0.6653245 - 0.3732767 - 0.4669131
 0.6658876 - 0.4898854 - 0.1288643
 - 0.2286410 0.3892021 0.4429955
-->D=K'*K
D =
 1. - 7.529E-18 - 2.169E-16
 - 7.529E-18 1. - 6.403E-17
 - 2.169E-16 - 6.403E-17 1.
-->DD=clean(D)
DD =
 1. 0. 0.
 0. 1. 0.
 0. 0. 1.
-->Q=orth(U);clean(Q'*Q)
ans =
 1. 0.
 0. 1.
-->R=rank(U)
R =
 2.
```

On remarque que Scilab donne des résultats numériques à `%eps` près. La dimension de l'image est donc 2, la dimension du noyau est 3.



---

## Projection orthogonale

On se place dans  $E = \mathbb{R}^n$  muni du produit scalaire  $\langle x, y \rangle = \sum_{i=1}^n x_i y_i$  et soit  $A$  un sous espace linéaire de  $E$  engendré par  $(e_1, e_2, \dots, e_d)$  ( $d < n$ ), vecteurs de  $E$ .

Soit  $f \in E$ . On cherche alors la projection orthogonale de  $f$  sur  $E$ , notée  $\hat{f}$ .

C'est l'unique élément de  $A$  vérifiant la relation :

$$\text{Pour tout } e \in A, \quad \langle f, e \rangle = \langle \hat{f}, e \rangle$$

On note  $U$  la matrice dont les colonnes sont les vecteurs  $(e_1, e_2, \dots, e_d)$  engendrant  $A$ .

Pour simplifier les calculs on construit la base orthonormale  $\{o_1, o_2, \dots, o_{d'}\}$ , de  $A$  en utilisant la fonction :

$$O = \text{orth}(U)$$

Pour tout  $i \neq j$ , on a  $\langle o_i, o_j \rangle = 0$  et  $\langle o_i, o_i \rangle = 1$ . Ou encore  $O' * O = I$

On a alors :

$$\hat{f} = \sum_{i=1}^{d'} a_i o_i$$

et pour tout  $i$ ,

$$\langle \hat{f}, o_i \rangle = a_i = \langle f, o_i \rangle$$

Ceci donne en écriture matricielle le vecteur colonne  $(a_i) = O' * f$  et finalement :

$$\hat{f} = O * O' * f$$

### **↪ Exemple(s)**

Soient dans  $E = \mathbb{R}^5$  le vecteur  $f = (1, 1, 1, 1, 1)^t$  et soit  $A$  un sous-espace vectoriel de  $E$  quelconque (aléatoire) de dimension 2.

Pour définir un sous-espace de dimension 2 quelconque  $A \subset \mathbb{R}^5$ , on utilise la fonction `rand(5,2)`, qui renvoie une matrice de deux colonnes dont les coordonnées sont des nombres (pseudo) aléatoires, indépendants.

On peut se demander si  $A$  est bien de dimension 2. La réponse théorique est  $A$  est « presque-sûrement » de dimension 2. La réponse numérique est que la probabilité que  $A$  soit de dimension 1 est infiniment inférieur à celle de gagner le gros lot du Loto.

```

scilab-4.1
scilab-4.1 Scilab Consortium (Inria, Enpc)
File Control Demos Graphic Window 0 Help Editor
-->f=ones(5,1);
-->A=rand(5,2);
-->O=orth(A);
-->V=clean(O'*O)
V =
 1. 0.
 0. 1.
-->f_chapeau=O'*O'*f
f_chapeau =
 0.8444005
 0.8772549
 0.6497161
 1.3500426
 0.9940442
-->f_chapeau'*A
ans =
 2.9910738 2.0155768
-->f'*A
ans =
 2.9910738 2.0155768
-->(f-f_chapeau)*A
ans =
 1.0E-15 *
 0.2957383 - 0.0329291
-->

```

On remarquera que les résultats sont numériques, donc à peu de choses près exacts.

### Valeurs propres et vecteurs propres



#### **Exercices**

On construit une matrice A quelconque ( $3 \times 3$ ), réelle et symétrique en posant :

```

rand('u'); a=int(5*rand(3,3));
A=a*a';

```

- 1) Vérifier que  $A=A'$ .
- 2) Cette matrice est-elle inversible ?
- 3) On pose :

```

[u,v]=spec(A); // voir dans l'aide « spec »... puis « bdiag »
u1=u(:,1);

```

---

```

u2=u(:,2) ;
u3=u(:,3) ;
lambda1=v(1,1) ;
lambda2=v(2,2) ;
lambda3=v(3,3) ;

```

4) Que donne les instructions suivantes ?

```

x1=clean(lambda1*u1-A*u1) ;
x2=clean(lambda2*u2-A*u2) ;
x3=clean(lambda1*u1-A*u1) ;

```

5) Que peut-on dire des instructions suivantes ?

```

disp("x = "+ string([x1,x2,x3]) ;
y=u*v-A*u ;
disp('y = ')
disp(clean(y)) ;
z=inv(u)*A*u ;
disp('z = ')
disp(clean(z)) ;
w=u*z*inv(u) ;
disp('w = ')
disp(clean(w)) ;
disp(['trace(A)='+string(trace(A)) ; 'trace(v)='+string(trace(v))])

```

6) On pose  $B=A./\max(\text{abs}(v))$

La limite de  $B^n$  existe-t-elle ?

Dans l'affirmative exprimer cette limite en fonction de  $u$ .

### Résolution de systèmes linéaires . Introduction : la fonction Scilab linsolve.



#### Exercices

On cherche à résoudre les problèmes suivant :

$\alpha$ )  $X$  est vecteur inconnu de dimension  $n$ . On ne connaît que l'image  $B$  de celui-ci par une application linéaire associée à une matrice  $(A_{i,j})_{1 \leq i,j \leq n}$ . Ainsi on cherche à résoudre l'équation linéaire :

$$AX = B$$

Dans un premier temps  $A$  et  $B$  sont supposés parfaitement connus. Etudier le script suivant :

```

clear
A=[10.,7.,8.,7.;7.,5.,6.,5.; 8.,6.,10.,9.;7.,5.,9.,10.] ; disp(A) ;
d=det(A) ; //matrice inversible?
B=[32, 23, 33,31]' ;
disp("B'' = '+ string(B')) ;

```

---

```

//1) Si A est inversible $X = A^{-1}B$
X=inv(A)*B;
disp("1) X'" = "+ string(X'));
BX=A*X; // vérification

// 2) Plus rapide : division matricielle à gauche ; $X=A\backslash b$ est la solution de $A*X=B$.
Y=A\B;
disp("2) Y'" = "+ string(Y'));
BY=A*Y;// verification

// 3) $[X_0, \ker A]=\text{linsolve}(A,B)$: donne les solutions de $AX+B=0$, de la forme $X_0+\ker A$
// Plus general. Ici $\ker A=[]$

[W,kerA]=linsolve(A,-B);
BW=A*W;
disp(" 3) W'" = "+ string(W'));

```

$\beta$ ) Les valeurs de  $B$  sont perturbées par des erreurs de mesures aléatoires,  $\Delta B$ . Que devient  $X$  ?

```

rand('normal')
rand('seed',sum(getdate()))
eps=rand(4,1); // par exemple
perturbB=max(abs(eps./B));
disp('perturbation max sur B, en pourcentage');
disp(string(perturbB))
disp("BB'" = ");
BB=B+ eps;
disp(string(BB'));
XX=inv(A)*BB;
disp("XX'" = " + string(XX'));
disp("Comparer avec X'" !!!" + string(X'));
BBX=A*XX;
YY=A\BB;
disp("YY'" = " + string(YY'))
BBY=A*YY;
[WW,ker(A)]=linsolve(A,-BB);
disp("WW'" = " + string(WW'))
BBW=A*WW;

```

$\gamma$ ) Les valeurs de  $A$  sont légèrement perturbées par un "bruit", ou erreurs de mesure.  $B$  est ici inchangé.

```

rand('normal')
deltaA=0.1*rand(4,4);
AAA=A+deltaA; disp(AAA);
disp(A)// pour comparer
[XXX,kerA]=linsolve(AAA,-B);
disp("XXX'" = " + string(XXX'));
disp(' Comparer avec X'" !!! ');
disp(string(X'));

```

---

## Retour sur les matrices réelles

### Définitions

- Une **matrice**  $X_{n \times p}$  est un tableau de nombres  $x_{(i,j)} \in \mathbb{R}$ , contenant  $n$  lignes et  $p$  colonnes. (syntaxe li-co).

**↪ Exemple(s)**

$$X_{2 \times 3} = \begin{pmatrix} x_{(1,1)} & x_{(1,2)} & x_{(1,3)} \\ x_{(2,1)} & x_{(2,2)} & x_{(2,3)} \end{pmatrix}, \quad Y_{3 \times 2} = \begin{pmatrix} y_{(1,1)} & y_{(1,2)} \\ y_{(2,1)} & y_{(2,2)} \\ y_{(3,1)} & y_{(3,2)} \end{pmatrix}$$

- Par convention, un **vecteur colonne**  $V_{n,1}$  est une matrice ( $n$  lignes, 1 colonnes). Lorsqu'il n'y a pas d'ambiguïté de dimension on note plus simplement ce vecteur :  $\vec{V}$ .

Exemple.

$V = \text{rand}(5,1)$  ;

◇  $\vec{V}$  peut être interprété comme un **vecteur de**  $\mathbb{R}^n$ .

◇ Un **vecteur ligne**  $W_{1,n}$  est une matrice (1, ligne  $n$  colonne). Exemple :  $V = \text{rand}(1,5)$

◇ Un vecteur ligne est le transposé d'un vecteur colonne.

- Une **matrice carrée** de dimension  $n$  est une matrice  $n \times n$ .
- La **diagonale** d'une matrice carrée  $X_{n \times n}$  est le vecteur de  $\mathbb{R}^n$ , noté  $\text{diag}(X) = D_X = (x_{(i,i)})_{i=1}^n$ .
- Une **matrice diagonale** est une matrice carrée dont les termes sont nuls, sauf éventuellement, les termes diagonaux.
- La **matrice identité** de dimension  $n$ ,  $\mathbb{I}_n$ , est une matrice  $n \times n$  dont les termes sont nuls, sauf les termes diagonaux, qui valent 1.

**↪ Exemple(s)**

$$\mathbb{I}_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbb{I}_1 = ( 1 )$$

- Une matrice carrée  $X_{n \times n}$  est **symétrique** si,  $x_{(i,j)} = x_{(j,i)}$ , pour tout  $i$  et  $j$ .

### Exemples

$$X = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 1 & 4 \\ 0 & 4 & 1 \end{pmatrix}$$

---

## Opérations sur les matrices

- **La somme** de deux matrices de **même dimensions**  $X_{n \times p}$  et  $Y_{n \times p}$ ,

$$S_{n \times p} = X_{n \times p} + Y_{n \times p}$$

est définie par :

$$s_{(i,j)} = x_{(i,j)} + y_{(i,j)}, \text{ pour tout } i \text{ et } j.$$

- **Le produit** d'une matrice  $X$  par un scalaire  $\lambda \in \mathbb{R}$ .

$$(\lambda \times X)_{(i,j)} = \lambda x_{(i,j)}, \text{ pour tout } i \text{ et } j.$$

- **Le produit** de deux matrices de **dimensions compatibles**  $X_{n \times p}$  et  $Y_{p \times r}$ , est une matrice  $n \times r$ , notée  $P = X * Y$  et définie par :

$$p_{(i,j)} = \sum_{k=1}^p x_{(i,k)} \times y_{(k,j)}, \text{ pour tout } 1 \leq i \leq n \text{ et } 1 \leq j \leq r.$$

◇ La compatibilité signifie que le **nombre de colonnes** de la matrice de gauche doit être égale au **nombre de lignes** de la matrice de droite.

◇ Si  $V$  est un vecteur de  $\mathbb{R}^n$  et  $X$  est une matrice  $n \times p$ ,  $X * V$  est un vecteur de  $\mathbb{R}^p$ .

◇ Si  $X$  et  $Y$  sont deux matrices carrées, on a en général,  $X * Y \neq Y * X$  (le produit matriciel n'est pas commutatif).

### ↪ Exemple(s)

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \text{ et } B = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \quad A * B = \begin{pmatrix} 1 & 4 \\ 3 & 8 \end{pmatrix} \neq B * A = \begin{pmatrix} 1 & 2 \\ 6 & 8 \end{pmatrix}$$

- **La puissance**  $p$ -ième d'une matrice carrée  $X_{n \times n}$  est définie par récurrence :

$$X^0 = \mathbb{I}_n \quad \text{et} \quad X^{p+1} = X^p * X = X * X^{p+1}, \quad p \geq 0$$

- **La transposée** d'une matrice  $X = X_{n \times p}$  est la matrice  $p \times n$ ,  $Y = X'$  avec :

$$y_{(i,j)} = x_{(j,i)}, \text{ pour tout } 1 \leq i \leq n \text{ et } 1 \leq j \leq p.$$

### Exemples

$$X = X_{2 \times 3} = \begin{pmatrix} x_{(1,1)} & x_{(1,2)} & x_{(1,3)} \\ x_{(2,1)} & x_{(2,2)} & x_{(2,3)} \end{pmatrix}, \quad Y = X' = \begin{pmatrix} x_{(1,1)} & x_{(2,1)} \\ x_{(1,2)} & x_{(2,2)} \\ x_{(1,3)} & x_{(2,3)} \end{pmatrix}$$

La première colonne devient la première ligne, e.t.c.

- 
- **La transposée d'un produit** de matrices  $X$  et  $Y$  vaut :

$$(X * Y)' = Y' * X'$$

- ◇ Le produit de deux matrices symétriques n'est pas en général symétrique.
- ◇ Si  $\vec{U}$  et  $\vec{V}$  sont deux vecteurs de  $\mathbb{R}^n$ ,  $\vec{U}' * \vec{V}$  est un nombre réel. On note ce nombre :

$$\langle \vec{U}, \vec{V} \rangle = \vec{U}' * \vec{V} = \vec{V}' * \vec{U}$$

nommé produit scalaire canonique dans  $\mathbb{R}^n$  de  $\vec{U}$  et  $\vec{V}$ . (il en existe d'autres)

- ◇ Si  $\vec{U}$  est un vecteur de  $\mathbb{R}^n$ ,  $\vec{U}' * \vec{U} = \langle \vec{U}, \vec{U} \rangle = \|\vec{U}\|_2^2$ , (le carré de sa norme euclidienne).
- ◇ Deux vecteurs  $\vec{U}, \vec{V}$  de  $\mathbb{R}^n$ , sont **orthogonaux** si  $\langle \vec{U}, \vec{V} \rangle = 0$ .

- **L'inverse** d'une matrice carrée  $n \times n$ ,  $C$  est définie par

$$C^{-1} * C = C * C^{-1} = \mathbb{I}_n$$

◇ L'inverse d'une matrice  $C$  n'existe que si son déterminant est non nul ou encore si la matrice est non-singulière. Une matrice est singulière si les lignes ou les colonnes composant la matrice sont linéairement dépendantes.

- **L'inverse d'un produit de matrices** carrées est obtenu par le produit des inverses, en inversant l'ordre des produits.

$$(A * B)^{-1} = B^{-1} * A^{-1}$$

$$(A * B * C)^{-1} = C^{-1} * B^{-1} * A^{-1}$$

- ◇ Une matrice  $C$  dont l'inverse est égale à sa transposée est une matrice **orthogonale**,

$$C * C' = C' * C = \mathbb{I}$$

- ◇ L'inverse et la transposée d'une matrice symétrique (inversible) sont des matrices symétriques.

- **Le rang d'une matrice**  $n \times p$  est le maximum de lignes ou de colonnes linéairement indépendantes.

- ◇ Une matrice carrée  $n \times n$ , possède une inverse si et seulement si son rang vaut  $n$ .

- **La trace d'une matrice carrée**  $C$ , notée  $tr(C)$ , est la somme de ces éléments diagonaux. On a la propriété suivante :

$$tr(A * B) = tr(B * A).$$

---

• **Théorème de Cayley-Hamilton.**

Soit  $A$  une matrice carrée  $n \times n$ , on définit son **polynôme caractéristique** par :

$$P(x) = \det(x\mathbb{I}_n - A) = x^n + p_{n-1}x^{n-1} + \dots + p_1x + p_0.$$

On a alors :

$$P(A) = A^n + p_{n-1}A^{n-1} + \dots + p_1A + p_0\mathbb{I}_n = 0_n.$$

◇ Ceci signifie que  $A^n$  est combinaison linéaire de  $\mathbb{I}_n, A, A^2, \dots, A^{n-1}$ .

**↪ Exemple(s)**

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad p(x) = x^2 - 5x - 2 \quad \text{et} \quad A^2 = 5A + 2\mathbb{I}_2$$

• **Matrices symétriques définies positives.**

Une matrice  $M_{n \times n}$  réelle symétrique est définie positive si, elle vérifie l'une des trois propriétés équivalentes suivantes :

1) pour tout vecteur  $\vec{V} \in \mathbb{R}^n$ , non nul,

$$\vec{V}' * M * \vec{V} > 0$$

2) Les valeurs propres de  $M$  sont strictement positives.

3) La forme bilinéaire symétrique définie sur  $\mathbb{R}^n$  par la relation suivante est un produit scalaire sur  $\mathbb{R}^n$ .

$$\langle x, y \rangle_M = x' M y$$

• **Propriétés des matrices symétriques définies positives.**

◇ La matrice  $M$  est seulement symétrique positive si, pour tout vecteur  $\vec{V} \in \mathbb{R}^n$ ,  $\vec{V}' * M * \vec{V} \geq 0$ .

◇ Toute matrice définie positive est inversible (à déterminant réel strictement positif), et son inverse est elle aussi définie positive.

◇ Si  $M$  est définie positive et  $r$  est un nombre réel strictement positif, alors  $rM$  est définie positive.

◇ Si  $M$  et  $N$  sont définies positives, alors  $M + N$  est définie positive.

◇ Si  $M$  et  $N$  sont définies positives, et si  $MN = NM$  (on dit qu'elles commutent), alors  $MN$  est définie positive.

◇ Une matrice  $M$  est définie positive si et seulement s'il existe une matrice définie positive  $A$  telle que  $A^2 = M$ ; dans ce cas, la matrice définie positive  $A$  est unique, et on peut la noter  $A = M^{1/2}$ .



---

• **Valeurs propres et vecteurs propres des matrices symétriques définies positives.**

Soit  $M_{n \times n}$  une matrice symétrique définie positive. Un vecteur propre  $\vec{V}_1 \in \mathbb{R}^n$  de  $M$ , associée à la valeur propre  $\lambda_1 \in \mathbb{R}$  est tel que :

$$\lambda_1 \vec{V}_1 = M * \vec{V}_1$$

◇ Les vecteurs propres associés à un matrice définie symétrique positive sont orthogonaux :

$$\langle \vec{V}_i, \vec{V}_j \rangle = 0, \text{ pour tout } i \neq j$$

◇ Les valeurs propres associées à un matrice symétrique définie positives sont strictement positifs.

◇ Si  $M_{n \times n}$  une matrice symétrique définie positive, il existe une matrice orthonormale  $O$  et une matrice diagonale  $\Delta$  telle que :

$$M = O' * \Delta * O$$

◇ Si  $M_{n \times n}$  une matrice symétrique définie positive,  $M$  est inversible et

$$M^{-1} = O' * \Delta^{-1} * O$$

◇ Le déterminant d'une matrice symétrique définie positive est égal au produit de ses valeurs propres (si une valeur propre est d'ordre  $p$ , elle intervient  $p$  fois).

---

## Retour sur les boucles

Scilab est un logiciel de calcul numérique optimisé pour traiter le calcul matriciel. Il est donc recommandé autant que possible d'écrire des programmes évitant les boucles.

### ↪ Exemple(s)

#### • La fonction factorielle.

Il existe plusieurs possibilités de construction de la fonction factorielle, la plus simple et la plus rapide étant celle n'utilisant pas de boucle

$$0! = 1, \quad n! = \prod_{k=1}^n k, \quad x \in \mathbb{R}, \quad n \in \mathbb{N}$$

```
//1) méthode directe. (n>0).
function y=fact(n)
y=prod([1 :n]);
endfunction
```

```
//2) Calcul récursif
function y= factorRecurs(n)
if (n>0) then y=n*factorRecurs(n-1), else y=1; end;
endfunction;
```

```
//3) Boucle for
function y=factorFor(n)
a=1;
for i=1 :n ,
a=i*a;
end;
y=a;
endfunction
```

```
//4) Boucle while
function y=factorWhile(n)
a=1;
i=0;
while (i<n)
i=i+1;
a=a*i;
end;
y=a;
endfunction
```

- 
- Calcul de  $S(x, n) = \sum_{k=0}^n \frac{x^k}{k!}$

On définit une fonction calculant sans boucle S(x,n) pour x réel et n entier.

```
function y=S(x,n)
a=x^(0 :n); // vecteur numérateur
b=cumprod([1,1 :n]); // vecteur dénominateur
y=sum(a./b);
endfunction
```

Initialisation de la boucle pour une représentation graphique

```
n=20;
x=-5 :0.1 :5;
y=[];
```

Début de la boucle for

```
for i=x
y=[y,S(i,n)];
end;
```

Fin de la boucle

```
h=scf(1); // propriétés de l'affichage graphique nouveau style.
clf
plot2d(x,y,2);
a=gca(); // propriétés des axes x, y, ...g(et) c(urrent) a(xe).
a.x_location="middle"; // position des axes.
a.y_location="middle";
a.grid=[5,5]; // grille en couleur [x=5,y=5]
h.background=4; // couleur de fond
h.figure_name= "S(x,k). Exemple (%d)"; // nom de la figure
```

- Même chose mais en "vectorisant" la fonction; l'entrée x est alors un vecteur. (sans boucle)

```
clear
function y=approx(x,n)
// x est un vecteur ligne ou un vecteur colonne
// n est un entier
// y sera un vecteur ligne
t=size(x); if (t(1)>t(2)) then x=x'; end; // transformation en vecteur ligne
a=[ones(x'),x'*ones(1 :n)];
b=cumprod(a,2);
c=diag(ones(0 :n)./cumprod([1,1 :n]));
d=b*c;
e=sum(d,2);
y=e'
endfunction
```

---

```
n=20 ;
x=(-5 :0.1 :5) ;
y=approx(x,n) ;
```

Puis représentation graphique ...

- Même chose en utilisant une procédure récursive ; entrée  $x$  vectoriel, réponse vectorielle.

```
function y=approcheExp(x,n)
if (n>0) then
y=(1/prod(1 :n)).*x.^n+approcheExp(x,n-1)
else
y=ones(x) ;
end
endfunction
x=-5 :0.1 :5 ;
n=20 ;
y=approcheExp(x,n) ;
```

Puis représentation graphique...



### ⚠ Exercices

- Comparaison des vitesses de calculs.

Soient deux vecteurs quelconques ,  $U = (u_i)_{i=1}^n \in \mathbb{R}^n$  et  $V = (v_j)_{j=1}^p \in \mathbb{R}^p$ .  
On veut construire la matrice  $A = (a_{i,j})$  telle que :

$$\forall i \in [1, n] \text{ et } \forall j \in [1, p], \quad a_{i,j} = u_i \times v_j$$

Plusieurs fonctions sont proposées. Quelle est la plus rapide ?

```
function A=matriceExtra(u,v)
for i=1 :length(u)
 for j=1 :length(v)
 A(i,j)=u(i)*v(j) ;
 end ;
end ;
endfunction

function B=matriceSuper(u,v)
b=u^v(1) ;
for j=2 :length(v)
 b=[b,u^v(j)] ;
end ;
B=b ;
endfunction
```

---

```

function B=matriceSuperBis(u,v) b=u(1)^v ;
for i=2 :length(u)
 b=[b;u(i)^v] ;
end ;
B=b ;
endfunction

function y=matriceUltra(u,v)
y=(u*ones(v')).^(ones(u)*v') ;
endfunction

```

- Evaluations des vitesses de calcul

1) Donnée des vecteurs (colonnes)

```

u=(1 :0.05 :20)' ;
v=(1 :0.05 :20)' ;

```

2) Début des évaluations des temps de calcul

```

tic() ;
A=matriceExtra(u,v) ;
t=toc() ;
disp("t Extra = "+string(t))

tic() ;
A=matriceSuper(u,v) ;
t=toc() ;
disp("t Super = "+string(t))

tic() ;
A=matriceSuperBis(u,v) ;
t=toc() ;
disp("t SuperBis = "+string(t))

tic() ; A=matriceUltra(u,v) ;
t=toc() ;
disp("t Ultra = "+string(t))

```

- De la nécessité de la boucle for.

On utilise les boucles pour répéter une opération impossible à vectoriser (car dépassant la capacité de stockage de la mémoire de l'ordinateur).

On simule  $N$  lancers d'un dés non pipés et on veut savoir si la moyenne des points obtenus est bien 3.5, pour  $N = 5.10^7$ .

```

K=5 ; // très petite boucle
n=100 ; // petite boucle
N=100000 ; // calcul vectoriel
clf ;

```

---

```

B=[] ;
for k=1 :K
 A=[] ;
 for i=1 :n
 t=sum(1+floor(6*rand(1,N)))/N ;
 A=[A,t] ; // accumulation des résultats (moyennes intermédiaires)
 end ;
Moyenne=sum(A)/n ; //moyenne des moyennes intermediares
B=[B,Moyenne] ;
disp("La moyenne des moyennes vaut , pour la petite boucle "+string(k)+" : "+ string(Moyenne));
plot2d(1 :n, cumsum(A)./(1 :n),k);
end ;
MoyenneFinale=mean(B) ;
disp('MoyenneFinale : '+ string(MoyenneFinale))
EcartType=st_deviation(B) ;
disp(' Ecart type '+ string(EcartType))

```

### • Ruine d'un joueur. De la nécessité de la boucle while ...

On considère un joueur disposant d'une fortune initiale  $x_0 \in [a, b]$  et jouant chaque seconde à un jeu de pile ou face ; il gagne +1 (saut à droite) ou  $-1$  (saut à gauche) avec même probabilité  $p = 0.5$ . On cherche savoir par exemple :

- le temps mis pour atteindre  $a$  =ruine (il perd tout) ou  $b$  =gain max ( il remporte le gain maximum).
- quelle est la probabilité de ruine du joueur à ce jeu ?

On simule alors un grand nombre de jeux et on calcule la fréquence de ruine et la moyenne des temps de jeux. (Loi des grands nombres)

```

rand('uniform') // initialisation du générateur de nombres aléatoire
J=[] ; // initialisation des fins de jeux ruine ou gain max
T=[] ;
x0=10;// fortune initiale
A=0 ; // seuil de ruine
B=40 ; // plafond de gain
N=1000 ; // nombre d'essais
for i=1 :N
 m=[] ;// initialisation du parcours d'une suite de jeux
 u=x0 ;
 while ((u<B) & (u>A)) ;
 j=2*round(rand())-1 ;
 u=u+j ;
 m=[m,u] ;
 end ;
 l=length(m) ;
 J=[J,m($)] ;
 T=[T,l] ;
end ;
NRuine=sum(J==A) ; fRuine=NRuine/N

```

---

```
NGain=sum(J==B); fGain=NGain/N
scf(1); clf(1);
plot2d(1 :N,T,2) // représentation des temps de jeux
scf(2); clf(2);
histplot(20,T,5) // histogramme des temps de jeux
lambda=1/mean(T);
MaxT=max(T); // recherche d'une courbe théorique s'ajustant à l'histogramme
Courbe=(lambda)*exp(-lambda*(0 :MaxT)); // $y = \lambda e^{-\lambda x}$?
plot2d(0 :MaxT,Courbe,2);
```

---

## Représentation IEEE des nombres réels

### Les nombres flottants

Les nombres réels sont représentés dans la plupart des machines par des nombres "flottants" et utilisent depuis 1985 la norme IEEE 754. (*IEEE, pour 'Institut of Electronics and Electrical Engineers*).

Il existe deux formats :

- Simple = 32 bits.

|           |              |              |
|-----------|--------------|--------------|
| s → signe | e → exposant | m → mantisse |
|-----------|--------------|--------------|

$$(1)^s \times 2^{e-127} \times 1.m$$

avec  $s = 1$  1bit ;  $e = 8$  bits ;  $m = 23$  bits.

$$Nb_{max} = 3,403 \dots 10^{38}$$

$$Nb_{min} = 1,175 \dots 10^{-38}$$

- Double = 64 bits (utilisé dans `Scilab`).

$$(1)^s \times 2^{e-1032} \times 1.m$$

avec  $s = 1$  1bit ;  $e = 11$  bits ;  $m = 52$  bits.

$$Nb_{max} = 1,797 \dots 10^{308}$$

$$Nb_{min} = 2,225 \dots 10^{-308}$$

Les nombres ont alors une précision maximale de 16 chiffres décimaux.

### Les arrondis

Le résultat exact d'une opération n'est généralement pas directement représentable par un nombre machine. Il doit être arrondi. Ceci pose un problème.

Comment choisir l'arrondi ?

- au plus près ?
- vers  $+\infty$  ?
- vers  $-\infty$  ?
- vers 0 ?

**⚡ Exemple(s)**

```
a=sin(%pi) ;
```

```
b=clean(a) ; //arrondi forcée à 0, pour a ∈ [-1010, +1010]
```



---

## Affichage

Par défaut, Scilab affiche 10 caractères, comprenant le point décimal et le signe. Si l'on veut plus de chiffres on utilise la fonction `format`. Pour avoir par exemple 20 caractères, ce qui donne 18 chiffres, on écrit :

```
-->f=format() // format actif
-->a=1/3; disp(a)
-->format(20)
-->b=1/3; disp(b);
-->format(30)
-->b=1/3; disp(b);
```

Que remarque-t-on ? (Au delà de `format(19)` l'affichage résiduel n'a plus aucun sens).

### ⚡ Exemple(s)

```
//Le plus grand nombre.
a=1.797e308; disp(a);
b=1.798e308, disp(b);
```

Au delà de `1.797e308`, Scilab répond `inf`, qui signifie  $\infty$ . On a alors un `overflow`

```
a=1.7976e308; disp(a);
b=1.1*a; disp(b);
c=0.9*a; disp(c)
```

```
//Le plus petit nombre.
a=2.251e-208; disp(a);
b=a*%eps; disp(b);
c=a*%eps/2; disp(c);
```

## Précision machine

La précision machine est défini par l' $\epsilon$ -machine. C'est la distance entre 1 et le plus proche des nombres flottants qui lui est supérieur. Il vaut : `2.220E-16`.

```
a=(1+%eps/2== 1);disp(a);
b=(1+0.6*%eps==1); disp(b);
c=(1+0.6*%eps==1+%eps); disp(c);
disp(a);
disp(b);
```

---

## Les problèmes liés à la précision et aux arrondis



### Exercices

#### Exemple 1

Étudier le résultat des commandes suivantes :

```
a=[sin(%pi);sin(%pi*10^10);sin(%pi*10^15)];
disp(a);
```

#### Exemple 2

Éxecuter le programme suivant :

```
B=4095.1; A=B+1; x=1;
x=(A*x)-B
x=(A*x)-B
x=(A*x)-B
x=(A*x)-B
x=(A*x)-B
x=(A*x)-B
```

ou encore :

```
B=4095.1; A=B+1; x=1;
for i=1 :5
x=(A*x)-B;
C(i)=x;
end; disp(C)
```

#### Exemple 3

On cherche à représenter le graphe de la fonction :  $x \mapsto f(x) = \frac{1 - \cos(x)}{x^2}$ , sur l'intervalle  $]10^{-9}, 10^{-6}]$ , à l'aide du script suivant :

```
x=1e-9 :1.e-11 :1.e-6;
y=(1-cos(x))./(x.**2);
clf;
plot2d(x,y,2);
```

On remarque que le graphe correspondant se situe de part et d'autre de  $\frac{1}{2}$ .

On sait par ailleurs que lorsque  $x \in \mathbb{R}^+$ ,  $1 - \cos(x) - \frac{1}{2}x^2 \leq 0$ .

Que se passe-t-il ?

#### Exemple 4

On évalue le script suivant :

```
a=sqrt(%eps/2);
b=(1-cos(a))/a^2;
disp(b);
```

La réponse Scilab est 1, résultat manifestement faux. Que se passe-t-il ?

---

## Erreur liée à une fonction prédéfinie Scilab.

Scilab est un logiciel de calcul numérique. Tout calcul est susceptible de comporter des erreurs. On étudiera, par exemple, les erreurs liées à la fonction Scilab : `horner` .

Un polynôme en  $X$  est une expression de la forme :

$$P(X) = a_0 + a_1X + a_2X^2 + \dots + a_nX^n$$

$a_p$  est le coefficient de degré  $p$ , et  $n$  est le degré du polynôme.

Un élément  $x_0$  tel que  $P(x_0) = 0$  est une racine du polynôme.

La syntaxe Scilab donne :

```
a=[1,2,3]; P=poly(a,'X','c');
// polynôme en X de coefficients a ('c' pour 'coeff')
b=[1,2,3]; Q=poly(b,'X','r');
// polynôme en X de racines 1,2,3 ('r' pour 'roots')
rr=roots(P), aa=coeff(Q);
```

On peut évaluer la valeur d'un polynôme en un point  $x_0$  ou sur un vecteur.

```
v=0 :0.1 :2;
y=horner(P,1),
yy=horner(Q,v).
```

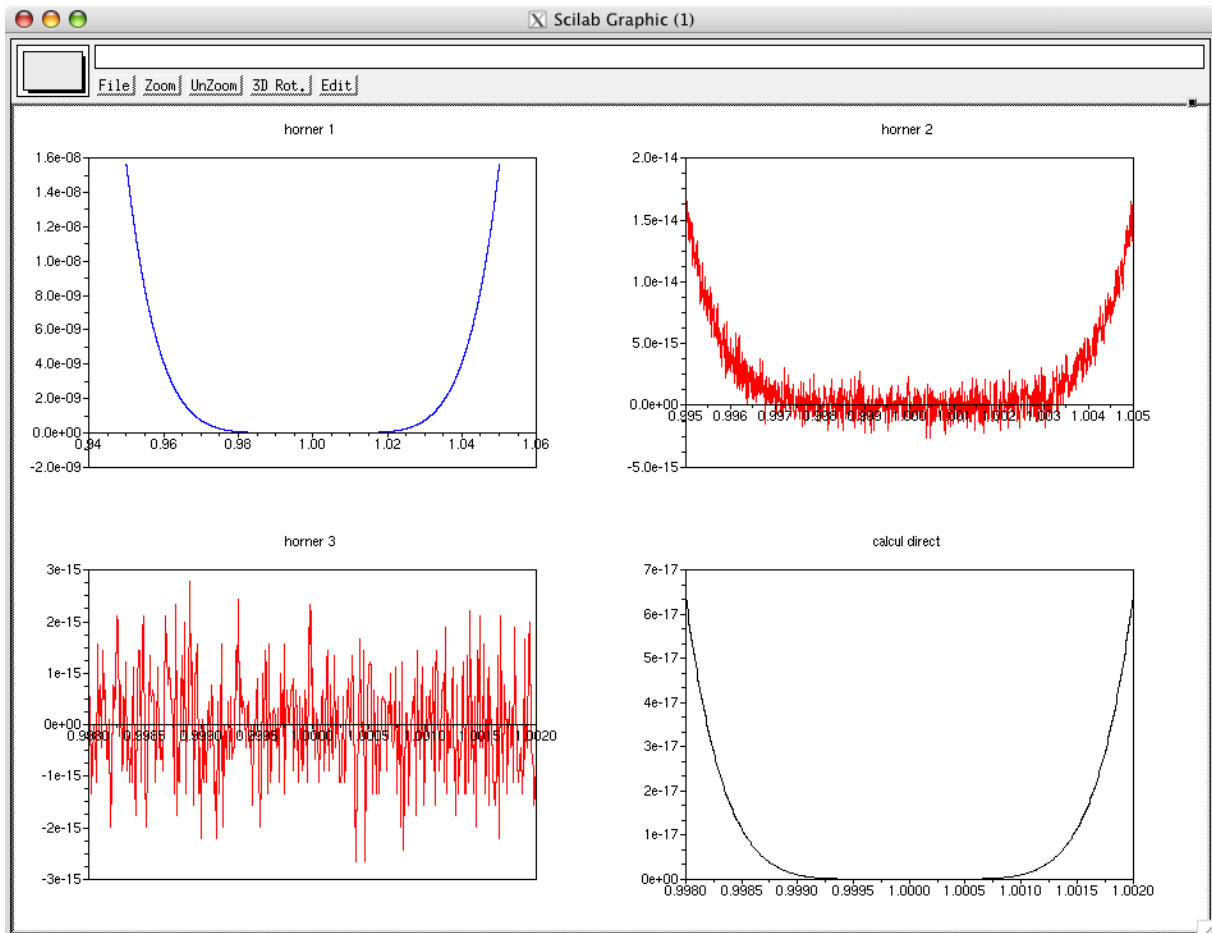
Pour étudier avec Scilab , au voisinage de 1, le graphe de la fonction polynomiale :

$$f(x) = (x - 1)^6,$$

on compose le script :

```
P=poly(ones(1,6),'X');
// par défaut, cela donne : P=poly(ones(1,6),'X','roots').
t=0.95 :1.e-5 :1.05;
xset('window',1);
clf;
subplot(2,2,1)
plot2d(t,horner(P,t),2);
xtitle('horner');
t=0.995 :1.e-5 :1.005;
subplot(2,2,2)
plot2d(t,horner(P,t),5);
xtitle('horner');
t=0.998 :1.e-5 :1.002;
subplot(2,2,3)
plot2d(t,horner(P,t),5);
xtitle('horner');
subplot(2,2,4);
plot2d(t,(t-1).^6,7);
xtitle('calcul direct');
```

On obtient la figure suivante :



On remarque que la précision dans les cadres horner 2 et horner 3 est très mauvaise. Ceci est lié à l'algorithme de définition de la fonction Scilab prédéfinie horner.

## Les tolérances IEEE

Les modes d'exceptions IEEE pour le calcul en réels flottants sont au nombre de trois.

- `ieee(0)` :

les exceptions produisent une erreur, les calculs s'arrêtent.

- `ieee(1)` :

les exceptions produisent l'affichage d'un message d'attention, les calculs continue selon les règles `ieee` avec l'introduction de `Nan` ou `inf`.

---

• `ieee(2)` :

les exceptions produisent l'introduction de `Nan` ou `+/- inf`. Les calculs continuent :

| Type               | Exemple                               | Résultat           |
|--------------------|---------------------------------------|--------------------|
| Opération invalide | $0/0, 0 \times \infty, \infty/\infty$ | Nan (not a number) |
| overflow           | $1/0$                                 | $\pm\infty$        |

Une opération non valide produit une réponse `Nan`.

La constante `Nan` est introduite par l'expression : `%nan`.

Une fois engendrée, `Nan` (not a number) se propage dans toutes les opérations suivantes.

**⚡ Exemple(s)**

```
a=ieee() // le mode actif.
ieee(2) // On passe au mode 2.
b=1/0 ;
c= b-1*10^10 ;
d=1/c ;
disp(b) ;
ll=log(0), 1/ll
```

---

## Recherche des solutions de systèmes d'équations non linéaires

### Utilisation de la fonction Scilab : fsolve

La fonction Scilab `fsolve` cherche à calculer numériquement les solutions approchées (ou encore les "zeros") de système d'équations non linéaires. La syntaxe d'appel la plus simple est :

```
x=fsolve(X0, fonction)
```

- X0 est un ( vecteur) réel, valeur initiale de l'argument de la fonction externe notée ici `fonction` à partir de laquelle le logiciel va chercher la solution.

#### ⚡ Exemple(s)

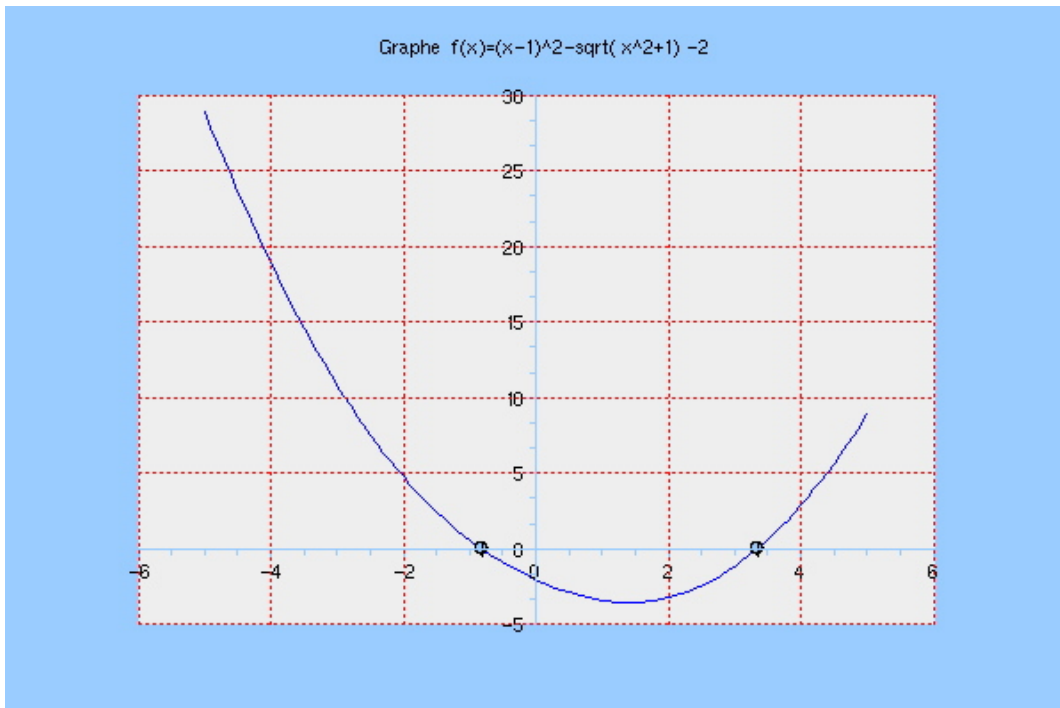
Quelles sont les solutions de l'équation suivante ?

$$(x - 1)^2 - \sqrt{x^2 + 1} = 2$$

La fonction à étudier est donc :

$$f(x) = (x - 1)^2 - \sqrt{x^2 + 1} - 2$$

```
clear ;
x=-5 :0.1 :5 ;
function y=fct(x) ;
 y=(x-1).^2-sqrt(x.^2+1)-2 ;
endfunction
f1=scf(1) ;
clf(1) ;
f1.background= 12 ;
f1.figure_name=" Graphe 2" ;
y=fct(x) ;
plot2d(x,y,2) ;
a=gca() ;
a.x_location="middle" ;
a.y_location="middle" ;
a.grid=[5,5] ;
a.background=31 ;
a.foreground=12 ;
xtitle(" Graphe f(x)=(x-1)^2-sqrt(x^2+1) -2 ") ;
u=fsolve(-1,fct) ;
disp(fct(u) ;
plot2d(u,fct(u),-3) ;
v=fsolve(6,fct) ;
disp(fct(v)) ;
plot2d(v,fct(v),-3) ;
```



Dans tous les cas, il est important de vérifier que la solution trouvée par scilab est correcte.

⚡ Exemple(s)

Quelles sont les racines du polynôme p ?

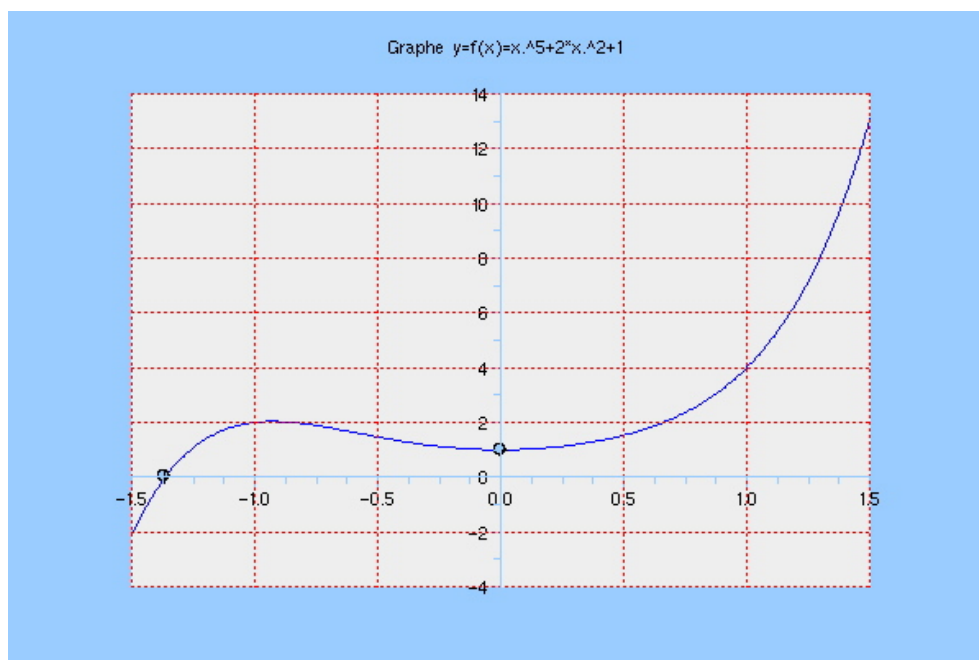
$$p(X) = 1 + 2X^2 + X^5$$

```
x=-1.5 :0.01 :1.5;
function y=fct(x);
y=x.^5+2*x.^2+1
endfunction
// copié-collé
f1=scf(1);
clf(1)
f1.background= 12;
f1.figure_name=" Grphe 2";
y=fct(x);
plot2d(x,y,2);
// copié-collé
a=gca();
a.x_location="middle";
a.y_location="middle";
a.grid=[5,5];
```

```

a.background=31 ;
a.foreground=12 ;
xtitle(" Graphe f(x)=y=x.^5+2*x.^2+1") ;
u=fsolve(-1.5,fct) ;
plot2d(u,fct(u),-3) ;
v=fsolve(6,fct) ;
plot2d(v,fct(v),-3) ;
disp('Racine 1 = ' +string(u)) ;
disp('Racine 2 = ' +string(v)) ;

```



La solution r2 n'est manifestement pas une bonne solution. L'algorithme semble butter sur un minimum relatif.

### ⚡ Exemple(s)

Quelles sont les racines du polynôme p (deuxième méthode) ?

$$p(X) = 1 + 2X^2 + X^5$$

```

p=poly([1,0,2,0,0,1],"x","coeff") ;
disp(p) ;
r=roots(p) ;
t=(r==conj(r)) ;
disp("racines complexes de p = "+string(r)) ;
disp("racines reelle de p = "+string(r(t))) ;
// u=factors(p) ;

```





### ⚡ Exercices

On cherche à résoudre dans  $\mathbb{R}^2$  le système :

$$\begin{cases} x^2 + y^2 = 3 \\ y = \exp(-(x^2 - x)/2) \end{cases}$$

- 1) Montrer graphiquement, avec `scilab`, que le système suivant possède deux solutions réelles.
- 2) Utiliser l'instruction `scilab fsolve` pour évaluer ses deux solutions :  $(x_1, y_1)$  et  $(x_2, y_2)$ .

- 1) mode manuel.

```
clear
t=0 :0.05 :2.1*%pi ;
f1=scf(1) ;
clf(1) ;
f1.background= 12 ;
f1.figure_name=" Graphe 3" ;
a=gca(1) ; // (Get Current Axes)
a.x_location="middle" ;
a.y_location="middle" ;
a.isoview="on" ; // axes orthonormés : oui
a.grid=[5,5] ;
a.background=31 ;
a.foreground=12 ;
xtitle(" Graphe x^2+y^2=3, en bleu et y=exp(-(x*(x-1)/2), en rouge ") ;
plot2d(sqrt(3)*cos(t),sqrt(3)*sin(t),2) ;
tt=-2 :0.05 :2 ;
plot2d(tt,exp(-(tt.^2-tt)/2),5) ; xselect
xstring(0.5,+2,'Cliquez une fois sur chaque solution ',0,1)
s1=xclick() ;
xstring(0.5,+1.8,'Cliquez encore une fois! ',0,1)
s2=xclick() ;
plot2d(s1(2),s1(3),-1)
disp('solution graphique 1 : x1 = '+ string(s1(2))+ ' ; y1 = '+ string(s1(3)))
disp('solution graphique 2 : x2 = '+ string(s2(2))+ ' ; y2 = '+ string(s2(3)))
plot2d(s2(2),s2(3),-1)
xclick() ;
```

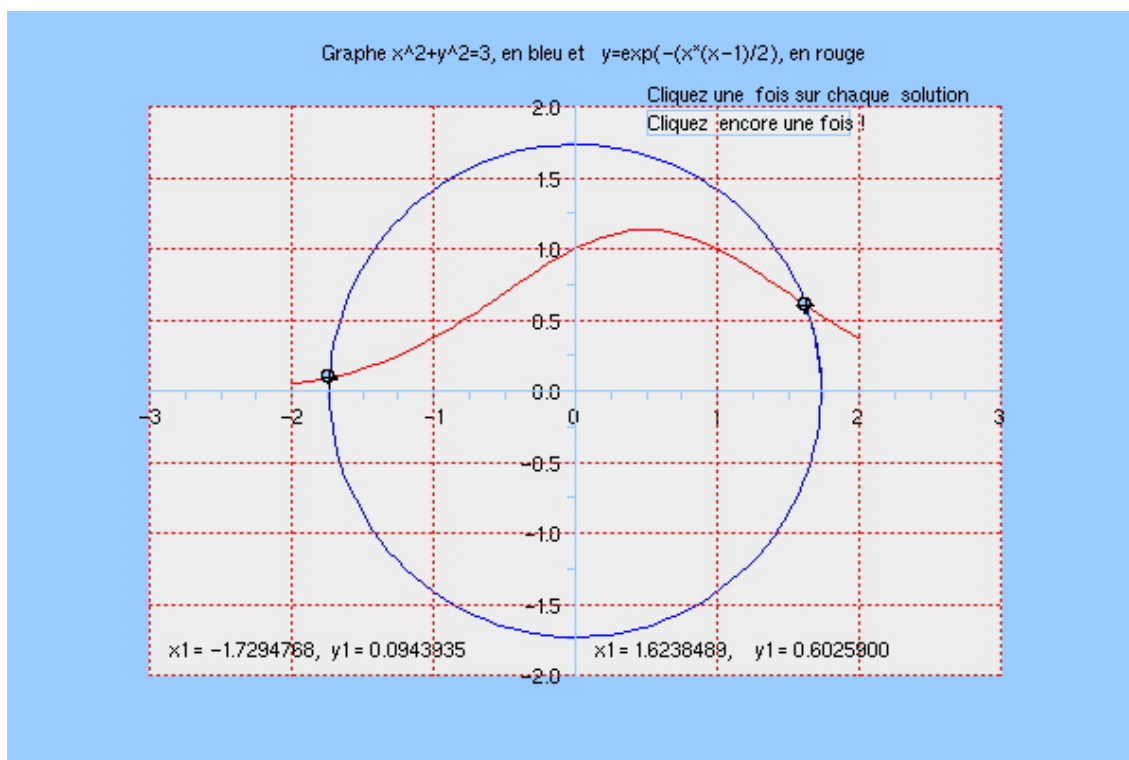
- 2) Utilisation de la fonction `fsolve`.

```
function Z=f1(x)
 Z(1)= x(1)^2+x(2)^2-3 ;
 Z(2)=exp(-0.5*(x(1)^2-x(1)))-x(2) ;
endfunction
u1=fsolve([-1,1],f1) ;
u2=fsolve([+1,1],f1) ;
plot2d(u1(1),u1(2),-3)
```

```

plot2d(u2(1),u2(2),-3)
xstring(-2.9,-1.9, " x1 = "+ string(u1(1))+ ", y1 = "+ string(u1(2)));
xstring(0.1,-1.9, " x1 = "+ string(u2(1))+ ", y1 = "+ string(u2(2)));

```



**Exercices**

**Exercice 1** Vérifier dans chaque cas que les équations suivantes ont une unique solution.

a)  $2 \cos(x) - x = 0, \quad x > 0.$

b)  $\exp(x) = y$  et  $x^2 + y^2 = 2$ , avec  $x > 0$  et  $y > 0$

**Exercice 2**

Résoudre le système d'équations

$$x^2 - y^2 = 1$$

$$x + 2y = 1$$

---

**Exercice 3**

Résoudre le système d'équations

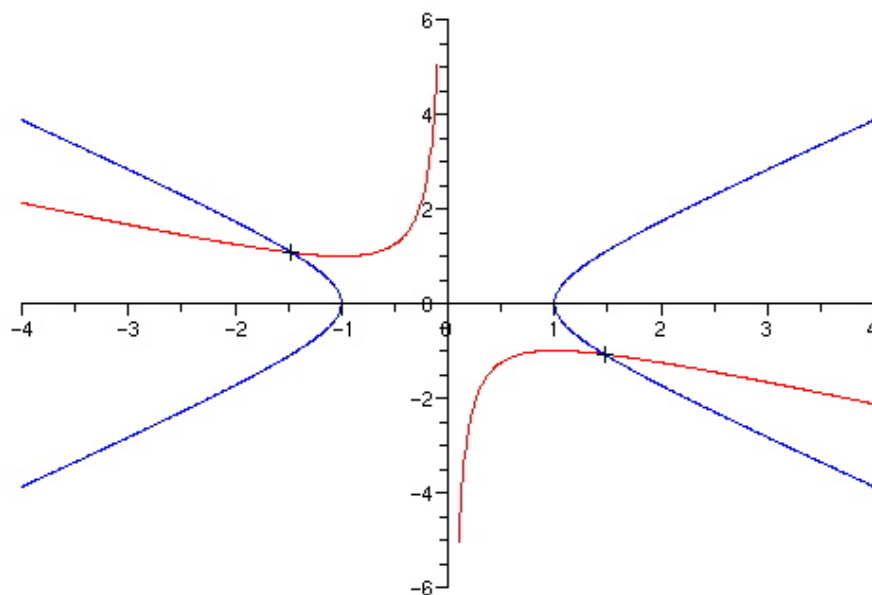
$$\begin{aligned}x^2 - y^2 + x &= 1 \\x + 2y &= 1\end{aligned}$$

**Exercice 4**

Trouver la solution éventuelle du système non linéaire suivant :

$$\begin{aligned}x^2 + 2xy &= -1 \\y^2 + 2xy &= -2\end{aligned}$$

Solution graphique de l'exercice7



---

## Résolution d'équations différentielles

### la fonction ode()

#### Systemes dynamiques et equations differentielles

Un systeme dynamique est un systeme qui evolue avec le temps, par exemple :

- Evolution d'un compte remunere.
- Evolution du cours d'un panier d'actions.
- Dynamique de populations.
- Cinetique de reactions chimiques.
- Mouvement d'un systeme de particules, soumis a differents champs de forces.

Ces systemes dynamiques peuvent parfois etre modelises par un systeme d'equations differentielles qu'il s'agit de resoudre soit analytiquement soit numeriquement.

On cherche alors :

- L'existence de solutions : quand et ou y-a-t-il une ou plusieurs solutions ?
- Comment se comportent les solutions a partir d'un point de depart, autour de ce point, a l'infini ?

Peu d'equations differentielles peuvent etre resolues (integrees) analytiquement. Et meme si elles le sont le calcul final reste numerique.

- On ne sait pas resoudre analytiquement, par exemple, des equations aussi simples que :

$$y' = y^2 - t$$

$$y' = \sin(ty)$$

$$y' = \exp(ty)$$

#### Classification des systemes d'equations differentielles

Le systeme le plus general est le systeme de  $n$  equations differentielles d'ordre  $p$  :

$$f(t, y, y', \dots, y^{(p)}) = 0 \quad f : U \subset \mathbb{R} \times \mathbb{R}^n \times \dots \times \mathbb{R}^n \rightarrow \mathbb{R}^n$$

ou  $U$  est un ouvert et  $y(t)$  est une fonction inconnue d'un intervalle  $I \subset \mathbb{R}$  dans  $\mathbb{R}$ .

En introduisant les fonctions supplementaires :

$$Y_1 = y', \quad Y_2 = y'', \quad \dots, \quad Y_{p-1}$$

on peut se ramener au systeme de  $np$  equations differentielles du premier ordre suivant :

$$f(t, y, Y_1, Y_2, \dots, Y_{p-1}, Y'_{p-1}) = 0, \quad y' = Y_1, \quad Y'_1 = Y_2, \quad \dots, \quad Y'_{p-2} = Y_{p-1}.$$

---

Finalement il s'agit donc de résoudre des systèmes d'équations différentielles du premier ordre :

$$y' = f(t, y, y') = 0$$

Ce système est dit explicite lorsqu'il peut se mettre sous la forme :

$$y' = f(t, y)$$

### Méthode d'Euler.

On veut calculer une approximation de la solution  $y(t)$  sur  $[a, b]$ .

- On subdivise l'intervalle  $[a, b]$  par un pas d'intégration  $h$ .

$$h = \frac{b - a}{N}$$

$$t_n = a + nh$$

- On note  $y_n$  la solution approchée de  $y(x_n)$ , obtenue par la relation de récurrence :

$$y_0 = y(0) , \text{ donnée et}$$

$$y_{n+1} = y_n + hf(t_n, y_n) , \text{ pour } n \geq 0.$$

- Il s'agit donc d'une approximation de la courbe par sa tangente.

#### ⚡ Exemple(s)

On veut résoudre l'équation  $y'(x) = y(x)$  sur l'intervalle  $[0, b]$ , avec pour condition initiale  $y(0) = 1$ .

On définit une suite :

$$y_0 = 1 \text{ et } y_{n+1} = y_n + \Delta t \cdot y_n, \text{ pour tout } n \geq 0. \Delta t \text{ est le pas de } t.$$

```
clear ;
function [x,y]=Euler(b,dx)
// x : subdivision
// y : solution sur x
n=floor(b/dx) ;
y=zeros(1,n+1) ;
x=0 :dx :n*dx ;
y(1)=1 ;
for i=2 :n+1
y(i)=y(i-1)+dx*y(i-1) ;
end ;
endfunction
[u,v]=Euler(2,0.001) ;
clf()
plot2d(u,v,2)
plot2d(u,exp(u),5)
```



**Exercices**

- 1) Calculer par la méthode d'Euler la solution, sur  $[0, 0.5]$ , de :  $y' = \frac{1}{1+x^2}$ ,  $y(0) = 0$ .
- 2) Calculer par la méthode d'Euler la solution, sur  $[0, 0.5]$ , de :  $y'(x) = y^2(x)$ ,  $y(0) = 0$ .
- 2) Calculer par la méthode d'Euler la solution, sur  $[0, 6]$ , de :  $\frac{y'}{y} = -1$ ,  $y(0) = 1$ .

### La fonction scilab ode

Le logiciel Scilab permet de résoudre de manière approchée toute équation (ou système d'équations) différentielle du type :

$$y' = f(t, y) \quad ; \quad y(t_0) = y_0,$$

sous réserve d'existence et d'unicité de la solution.

Le "solveur" de systèmes différentielles explicites de scilab est la fonction `ode`. Son nom vient de l'acronyme "*Ordinary Differential Equation solver*".

C'est en fait une interface pour le package ODEPACK écrit en fortran qui contient lui-même différents solveurs (`isoda`, `isode` ...) qui utilisent deux méthodes principales :

pour les problèmes stables et aux solutions explicites : méthode d'Adams ;

pour les problèmes aux solutions instables (raides) : méthode BDF (Backward Differentiation Formula).

Par défaut `ode` choisit lui-même la méthode selon la nature du problème. `ode` possède de plus une liste d'arguments qui ne seront pas évoqués ici.

• **La fonction `ode` nécessite l'entrée dans l'ordre de quatre arguments :**

- 1) la donnée initiale :  $y_0$ .
- 2) le temps initial  $t_0$ .
- 3) les instants de calcul de la solution.
- 4) la fonction extérieure  $f$  (qui a toujours à deux arguments  $t$  et  $y$ ).



**Exercices**

Déterminer la solution sur  $\mathbb{R}^+$  de l'équation différentielle suivante :

$$y' + 2y = e^{-t} + 1 \quad ; \quad \text{avec } y(0) = 1 \quad (*)$$

---


Il est facile de montrer que la solution de (\*) est :

$$f(t) = \frac{1}{2}(1 - e^{-2t}) + e^{-t}$$

Cela donne le script :

```
function yprim=fct(t,y)
yprim=-2*y+exp(-t)+1;
endfunction
t0=0;
y0=1;
T=t0 :0.1 :10;
sol=ode(y0,t0,T,fct);
clf()
plot2d(T,sol,2) // La solution calculée par ode()
plot2d(T,0.5*(1-exp(-2*T))+exp(-T),5); // La solution formelle
```



 Exercices

Etudier le script suivant :

```
function dx=fct(t,x)
dx(1)=x(2);
// y'=Z
dx(2)=-0.4*x(2)-0.5*x(1);
// Z'=-0.4Z-0.5y; d'où y''=-0.4y'-0.5y
endfunction
t0=0;
// temps initial
y0=[0;1];
// état initial
T=t0 :0.01 :30;
sol=ode(y0,t0,T,fct);
// solution par ode
clf()
plot2d(T,sol(1,:),2)
p=poly([0.5,0.4,1],"x","c")
// on résout l'E.D. d'ordre 2 : y''+0.4y'+0.5y=0; y(0)=0 et y'(0)=1
r=roots(p) // racines de la résolvante
im=imag(r(1))
reel=real(r(1))
z=(1/im)*exp(reel*T).*sin(im*T);
// solution analytique
plot2d(T,z,5);
```



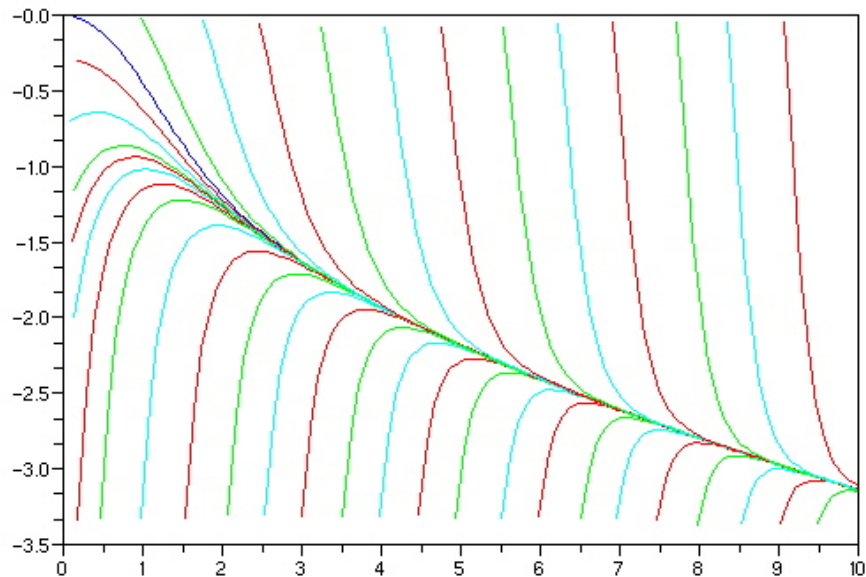
### Exercices

Résolution de la célèbre équation :  $\frac{dy}{dt} = y^2 - t$  avec  $y(0) = y_0$

Etudier le script suivant :

```
clear
function yprim=f(t,y)
yprim=y.^2-t;
endfunction;
t0=0; y0=0; tfin=10; dt=0.1; T=t0 :dt :tfin;
sol=ode(y0,t0,T,f);
h=scf(1);
clf(1);
plot2d(T,sol,2)
xselect()
xtitle("Pour continuer cliquer gauche, pour terminer cliquer droit dans le cadre")
// tracé interactif
while(%t)
[c,t0,y0]=xclick();
if (c==2) then break end;
T=t0 :dt :tfin;
sol=ode(y0,t0,T,f);
plot2d(T(1 :length(sol)),sol,2);
end;
```

Pour continuer cliquer gauche, pour stopper cliquer droit dans le cadre







### Exercices

Etudier le script suivant :

```
function dy=fct(t,y)
 dy(1)=cos(t)*y(2);
 dy(2) =y(1)-y(2);
endfunction
t0=0;
y0=[10;2];
t=0 :0.1 :10
z=ode(y0, t0, t, fct);
plot2d(t,z(1, :))
```

Ce script permet de calculer les solutions du système d'équations différentielles (d'inconnues  $y_1$  et  $y_2$ ), pour des conditions initiales données :

$$\begin{aligned}y_1'(t) &= \cos(t)y_2(t) \\ y_2'(t) &= y_1(t) - y_2(t)\end{aligned}$$

On a donc  $y_2(t) = \frac{y_1'(t)}{\cos(t)}$  et  $y_2'(t) = y_1(t) - y_2(t)$ . Finalement on résout l'équation

$$y'' + (1 + \tan(t))y' - y \cos(t) = 0$$

## Système d'équations différentielles linéaires linéaires

On considère le système d'équations différentielles à coefficients constants :

$$(*) \begin{cases} y'(t) = Ay(t) \\ y(t_0) = y_0 \in \mathbb{R}^n \end{cases}$$

où  $A$  est une matrice carrée constante, à valeurs dans  $\mathbb{R}^n$ , c'est-à-dire un élément de  $\mathcal{M}_n(\mathbb{R})$ .

On montre (*théorème de Cauchy-Lipschitz*) que, pour toute condition initiale, il existe une solution unique.

On définit pour, toute matrice  $A \in \mathcal{M}_n(\mathbb{R})$ , la série :

$$\exp(A) = \sum_{n=0}^{+\infty} \frac{A^n}{n!}$$

On peut montrer que cette série est convergente dans  $\mathcal{M}_n(\mathbb{R})$  et sa limite est nommée exponentielle de  $A$ , qui est notée  $\exp(A)$ .

On rappelle les propriétés importantes de l'exponentielle d'une matrice :

$$\begin{cases} \exp(A_1 + A_2) = \exp(A_1) \exp(A_2), & \text{si } A_1 \text{ et } A_2 \text{ commutent} \\ \exp(PAP^{-1}) = P \exp(A)P^{-1} \\ \frac{dt}{t}(\exp(tA)) = A \exp(tA) = \exp(tA)A \end{cases}$$

La solution du système différentiel (\*) est alors :

$$y(t) = \exp(tA)y_0$$



### ⚠ Exercices

Trouver la solution du système d'équations différentielle suivant et donner une représentation graphique de  $y_1(t)$ .

$$(*) \begin{cases} y_1' &= -0.9y_1 & -0.1y_2 & -0.1y_3 \\ y_2' &= -0.1y_1 & +0.1y_2 & +y_3 \\ y_3' &= y_1 & -y_2 & +0.2y_3 \end{cases}$$

$$y_1(0) = 7, \text{ et } y_2(0) = y_3(0) = 0$$

#### Solution

```
clear
A=[-0.9,-0.1,-0.1;-0.1,0.1,1;1,-1,-0.2];
y0=[7;0;0];
B=[];
T=0 :0.01 :40;
for t=T
 B=[B,expm(t*A)*y0];
end;
clf();
plot2d(T,B(1, :),2);

// solution avec ode
function ydot=fct(t,y)
 ydot(1)=A(1,1)*y(1)+A(1,2)*y(2)+A(1,3)*y(3);
 ydot(2)=A(2,1)*y(1)+A(2,2)*y(2)+A(2,3)*y(3);
 ydot(3)=A(3,1)*y(1)+A(3,2)*y(2)+A(3,3)*y(3);
endfunction

t0=0;
sol=ode(y0,t0,T,fct);
plot2d(T,sol(1, :),5)
plot2d(T,zeros(T))
```



## Exercices

### Exercice 1

Résoudre sur  $[0,10]$  avec Scilab les deux équations différentielles suivantes :

a)  $y' = -y + ty^2$ , avec  $y(0) = 1$  (Bernoulli d'ordre 2).

b)  $y'' + \sin(y) = 0$  avec  $y(0) = 0$  et  $y'(0) = 1$  (pendule pesant).

Tracer les solutions obtenues en fonction de  $t$ .

#### Solution

```
clear
function dy=fct1(t,y)
dy=-y+t*y^2;
endfunction
t0=0;y0=1;t=:0.1:10;
z1=ode(y0,t0,t,fct1);
xset('window',1);
clf()
subplot(1,2,1)
plot2d(t,z1,2);
//
function dy=fct2(t,y)
dy(1)=y(2);
dy(2)=-sin(y(1));
endfunction
t0=0;y0=[0;1];t=:0.1:10;
z2=ode(y0,t0,t,fct2);
subplot(1,2,2)
plot2d(t,z2(1,:),5);
```

### Exercice 2

Résoudre le système d'équations différentielles suivant (dit du Brusselator) sur l'intervalle  $[0, 20]$  et tracer les solutions obtenues,  $u_1$  et  $u_2$  :

$$u_1'(t) = 2 - 5u_1 + u_2u_1^2$$

$$u_2'(t) = 4u_1 - u_2u_1^2$$

avec :  $u_1(0) = 1$  et  $u_2(0) = 1$

#### Solution

```
clear
function y=fct3(t,x)
y(1)=2-5*x(1)+x(2)*x(1)*x(1);
```

---

```

y(2)=4*x(1)-x(2)*x(1)*x(1) ;
endfunction
t0=0 ;x0=[1 ;1] ;
t=0 :0.01 :20 ;
y=ode(x0,t0,t,fct3) ;
xset('window',1)
clf
subplot(1,3,1)
plot2d(t,y(1, :),3) ;
subplot(1,3,2)
plot2d(t,y(2, :),2) ;
subplot(1,3,3)

```

### Exercice 3

Résoudre sur  $[0, 10]$  l'équation différentielle suivante :

$$y'(t) = y(2 - \sqrt{y}) - y^2/(1 + y^2),$$

avec la condition initiale  $y(0) = 1$  et tracer la solution obtenue.

### Exercice 4

Résoudre sur  $[0, 10]$  l'équation différentielle suivante :

$$y'(t) = -y^4/(1 + y^4) + y(1 - y^2),$$

avec la condition initiale  $y(0) = 2$  et tracer la solution obtenue.

### Exercice 5

Résoudre sur  $[0, 20]$  l'équation différentielle suivante (dite du bourgeon de l'épinette) :

$$y'(t) = y(2 - y) - y^2/(1 + y^2).$$

### Exercice 6

Résoudre sur  $[0, 20]$  l'équation différentielle suivante :

$$y'(t) = y^2 - y \sin(t) + \cos(t), \text{ avec } y(0) = 0.$$

### Exercice 7

On considère l'équation différentielle :  $y''(t) - y'(t) + y(t) + \sin(2t)$

Calculer par `scilab` la solution de cette équation différentielle vérifiant les conditions :  $y(0) = 1$  et  $y'(0) = 0$ .

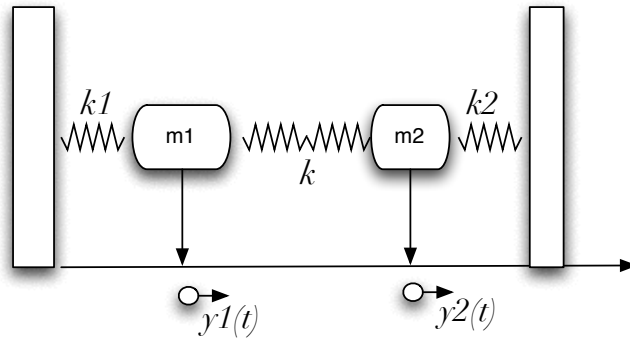
Représenter la solution sur l'intervalle  $[0, 5]$ .

---

## Quelques exemples classiques

### Exemple 1

On considère un système de deux masses de masse  $m_1$  et  $m_2$  soumises aux efforts de trois ressorts de raideur  $k_1$ ,  $k$  et  $k_2$ , selon le schéma suivant :



On suppose  $m_1 = 50$ ,  $m_2 = 30$ ,  $k = 100$ ,  $k_1 = 10$  et  $k_2 = 7$ . (par exemple)

On note  $y_1(t)$  et  $y_2(t)$  les positions par rapport au repos des masses  $m_1$  et  $m_2$  au temps  $t$ .

Les équations régissant  $y_1$  et  $y_2$  s'écrivent alors, s'il n'y a pas de frottement (ou viscosité) :

$$\begin{cases} m_1 y_1'' = k(y_1 - y_2) - k_1 y_1 \\ m_2 y_2'' = -k(y_1 - y_2) - k_2 y_2 \end{cases}$$

Les équations régissant  $y_1$  et  $y_2$  s'écrivent alors, s'il y a viscosité  $v_1 = 0.05$  et  $v_2 = 0.1$  :

$$\begin{cases} m_1 y_1'' = k(y_1 - y_2) - k_1 y_1 - v_1 y_1' \\ m_2 y_2'' = -k(y_1 - y_2) - k_2 y_2 - v_2 y_2' \end{cases}$$

Résoudre successivement ces deux systèmes d'équations différentielles en utilisant la fonction scilab `ode`, pour  $t_0 = 0$  et  $y_1(t_0) = 0$ ,  $y_2(t_0) = 1$ .  $T$  allant de 0 à 100

On pourra tester la fonction suivante :

```
function yprim=fct(t, y,k,k1,k2,m1,m2,v1)
 yprim(1)=y(3) ;
 yprim(2)=y(4) ;
 yprim(3)=(1/m1)*(k*(y(2)-y(1))-k1*y(1))-v1*y(3) ;
 yprim(4)=(1/m2)*(k*(y(1)-y(2))-k2*y(2))-v2*y(4) ;
endfunction
```

Représenter l'évolution dans le temps de  $y(2) - y(1)$ .

---

**Exemple 2. Problème de Lotka-Voltera. Système dynamique : proies / prédateurs.**

Durant la première guerre mondiale, à Trieste, la pêche avait diminuée et les quelque prélèvements des poissons dans les filets montraient une considérable augmentation de la proportion des requins (peu intéressant) par rapport aux sardines. Voltera instruit de la situation par le bureau des pêches proposa le modèle suivant.

On est en présence d'un système où cohabitent une population de sardines et une population de requins.  $x(t)$  est l'effectif de la population des sardines.

$y(t)$  est l'effectif de la population des requins

- La population des sardines croît proportionnellement à l'effectif  $x(t)$  de celle-ci, mais décroît proportionnellement à l'effectif des requins, s'il y a rencontre entre les deux populations  $bx(t)y(t)$ .

$$x'(t) = ax(t) - bx(t)y(t)$$

- Plus la population de sardines est importante plus les requins prolifèrent s'il y a rencontre  $cx(t)y(t)$ . Les requins mangent les sardines. Mais les requins sont en compétition entre-eux. Plus le nombre de requins est grand moins ils trouvent à manger, moins ils prolifèrent.

$$y'(t) = cx(t)y(t) - dy(t)$$

- On a donc le modèle dynamique suivant :

$$\begin{cases} x'(t) = ax(t) - bx(t)y(t) \\ y'(t) = cx(t)y(t) - dy(t) \end{cases}$$

Voltera en déduisit sans pouvoir faire de calculs numériques que plus l'on pêche de poisson plus la proportion de sardines (donc de poissons intéressants) était importante.



**Exercices**

Retrouver ce résultat avec pour, application numérique :

a)  $a = 3.1, b = 1, c = 1, d = 2.1$

a')  $a = 3, b = 1, c = 1, d = 2$

b)  $a = 1, b = 1, c = 1, d = 4$

---

Voici un script possible.

```
function dY=Voltera(t,Y,a,b,c,d)
 dY(1)=a*Y(1)-b*Y(1)*Y(2);
 dY(2)=c*Y(1)*Y(2)-d*Y(2);
endfunction

t0=0;
T=t0 :0.05 :5;
Y0=[2;3];

//cas a
a=3.1;b=1;c=1;d=2.1;
sol=ode(Y0,t0,T,Voltera);

scf(0); clf(0);
plot2d(T,sol(1, :)-2,2); // sardines
plot2d(T,sol(2, :)-3,5); // requins
xtitle("a) fluctuation de la population de sardines et des requins x0=2,y0=3. Clicker!")
xselect();
xclick();

scf(1); clf(1); xselect()
plot2d(2,3,-2)
plot2d(sol(1, :),sol(2, :),2)
xtitle(" a) Phases. clicker gauche pour rechercher l''equilibre")

while(%T)
 [u,v,w]=xclick();
 if (u==2) then break end; // click droit
 sol=ode([v;w],t0,T,Voltera);
 plot2d(sol(1, :),sol(2, :),2);
 plot2d(v,w,-3)
end;

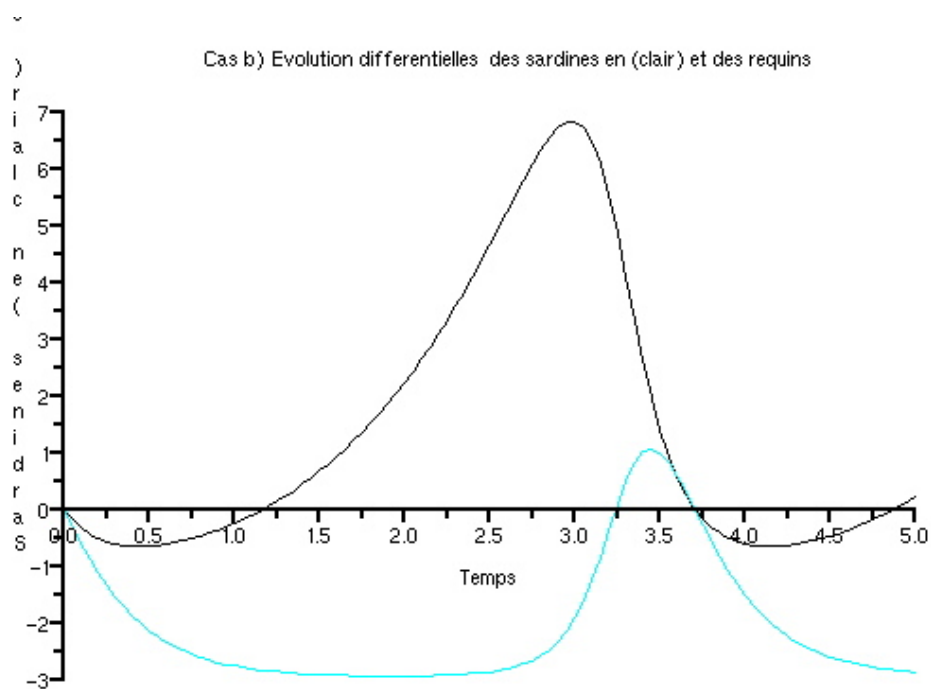
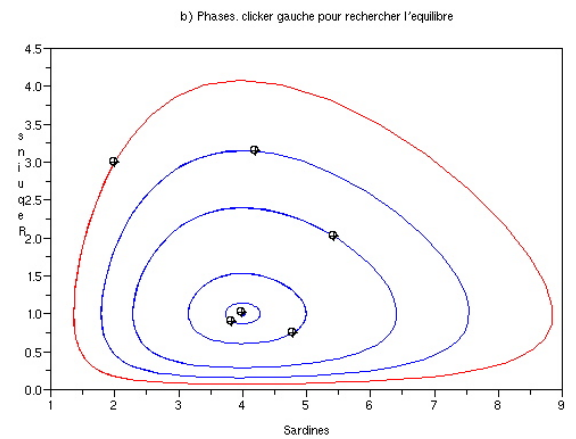
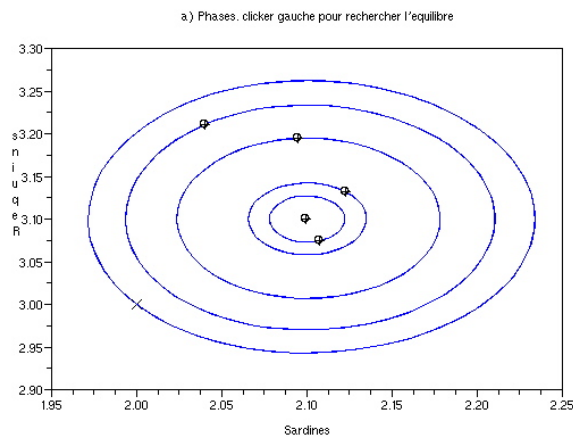
scf(2); clf(2); plot2d(T,sol(1, :)-v,2) plot2d(T,sol(2, :)-w,5)
xtitle("a) fluctuation de la population de sardines et des requins x0="+string(v)+"y0="+string(w))
xselect()

//cas b
scf(3); clf(3); a=1;b=1;c=1;d=4;
sol2=ode(Y0,t0,T,Voltera);
plot2d(T,sol2(1, :)-2,2)
plot2d(T,sol2(2, :)-3,5)
xtitle(" b) fluctuation de la population de sardines et des requins x0=2,y0=3. Clicker!")
xclick();
scf(4); clf(4);
plot2d(sol2(1, :),sol2(2, :),5);
plot2d(2,3,-3);
xtitle(" b) Phases. clicker gauche pour rechercher l''equilibre");
xselect()
while(%T)
 [u,v,w]=xclick();
 if (u==2) then break end;
```

```

sol=ode([v;w],t0,T,Voltera);
plot2d(sol(1,:),sol(2,:),2);
plot2d(v,w,-3)
end;
scf(5); clf(5);
plot2d(T,sol(1,:)-v,2);
plot2d(T,sol(2,:)-w,5);

```





---

## Représentation graphique des séries de Fourier avec scilab

### Rappels

On analyse une fonction périodique continue par morceaux,  $f$ , à valeurs réelles ou complexes, de période  $T$ , de fréquence  $F$ , appelée fréquence fondamentale ou fréquence du fondamental. Les coefficients de Fourier complexes de  $f$  (pour  $n \in \mathbb{Z}$ ) sont donnés par :

$$c_n = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) e^{-i \frac{2\pi n}{T} t} dt$$

Si  $n > 0$ , on appelle harmonique de rang  $n$  de la fonction  $f$  la fonction sinusoïdale de fréquence  $nF$  obtenue en tenant compte des coefficients de Fourier d'indice  $n$  et  $-n$ . C'est la fonction :

$$c_n e^{+i \frac{2\pi n}{T} x} + c_{-n} e^{-i \frac{2\pi n}{T} x}$$

Pour  $n = 0$ , le coefficient  $c_0$  n'est autre que la valeur moyenne de  $f$  sur une période. La série de Fourier de  $f$  est la série de fonctions obtenue en sommant les harmoniques successifs.

### Conventions usuelles pour les fonctions à valeurs réelles

Pour une fonction  $f$  réelle et périodique on posera plutôt :

$$a_0 = c_0 = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) dt$$

et, pour  $n > 0$  :

$$a_n = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cos\left(nt \frac{2\pi}{T}\right) dt \quad \text{et} \quad b_n = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \sin\left(nt \frac{2\pi}{T}\right) dt$$

L'harmonique de rang  $n$  se réécrit alors comme la fonction :

$$a_n \cos\left(nx \frac{2\pi}{T}\right) + b_n \sin\left(nx \frac{2\pi}{T}\right) = \chi_n \cos\left(nx \frac{2\pi}{T} + \phi_n\right)$$

### Propriétés des coefficients de Fourier de $f$

Certaines propriétés de  $f$  se traduisent sur les coefficients de Fourier.

- Si  $f$  est une fonction **paire**, on a  $c_{-n} = c_n$  pour tout  $n$ . Si en outre  $f$  est **réelle**, on obtient  $b_n = 0$ , pour tout  $n$ .
- Si  $f$  est une fonction **impaire**, on a  $c_{-n} = -c_n$ , pour tout  $n$ . Dans le cas **réel** cela donne  $a_n = 0$  pour tout  $n$ .

Ces propriétés sont en fait des équivalences : on peut lire la parité de la fonction sur son spectre en fréquences. Plus généralement, si deux fonctions continues ont la même analyse en fréquences (mêmes coefficients de Fourier), elles sont identiques. Dans le cas continu par morceaux, elles coïncident en tous les points sauf un nombre fini.

Par exemple, si une fonction continue a tous ses coefficients de Fourier nuls sauf un nombre fini, c'est en fait un polynôme trigonométrique.

## Comportement des coefficients de Fourier de $f$ à l'infini

Les coefficients de Fourier de  $f$  tendent vers 0 lorsque  $n$  tend vers l'infini : **théorème de Riemann-Lebesgue**.

• Plus la fonction est régulière, plus le rythme de décroissance des coefficients de Fourier est élevé. Par exemple, pour une fonction continue et  $\mathcal{C}^1$  par morceaux, on établit, par intégration par parties,

$$c_n(f') = \frac{2i\pi n}{T} c_n(f)$$

Cela prouve que la suite  $c_n(f)$  tend vers 0 plus vite que la suite  $1/n$ .

Plus généralement, pour une fonction de classe  $\mathcal{C}^k$  et  $\mathcal{C}^{k+1}$  par morceaux, on établit

$$c_n(f^{(k+1)}) = \left(\frac{2i\pi n}{T}\right)^{k+1} c_n(f)$$

Ainsi, la suite  $c_n(f)$  tend vers 0 plus vite que la suite  $1/n^k$ .

## Théorème de convergence ponctuelle (de Dirichlet)

Sous des hypothèses de régularité convenables, la fonction peut effectivement se décomposer comme superposition de fonctions sinusoïdales.

Précisément une fonction périodique  $f$  de période  $T$ , **continue et dérivable par morceaux** est, en chaque point, somme de sa série de Fourier :

$$f(x) = \sum_{n=-\infty}^{+\infty} c_n(f) e^{inx \frac{2\pi}{T}}$$

Pour les fonctions à valeurs réelles, on peut écrire cela avec les conventions adaptées :

$$f(x) = a_0(f) + \sum_{n=1}^{+\infty} \left( a_n(f) \cos\left(nx \frac{2\pi}{T}\right) + b_n(f) \sin\left(nx \frac{2\pi}{T}\right) \right)$$

Si la fonction  $f$  n'est pas continue, le théorème tient encore, à ceci près que la série de Fourier converge vers la régularisée de  $f$ , c'est-à-dire la fonction obtenue en faisant la demi-somme des limites à droite et à gauche en chaque point.

La démonstration du théorème se base sur le fait que la série de Fourier se calcule par produit de convolution avec un polynôme trigonométrique aux propriétés remarquables : le noyau de Dirichlet.

En 1829, Dirichlet établit une première version de cette démonstration, qui fut complétée par Jordan en 1881 pour aboutir au théorème actuel (avec en fait une hypothèse plus faible encore : "localement à variations bornées" au lieu de "dérivable par morceaux").

---

### Exemple. Scilab. 1.

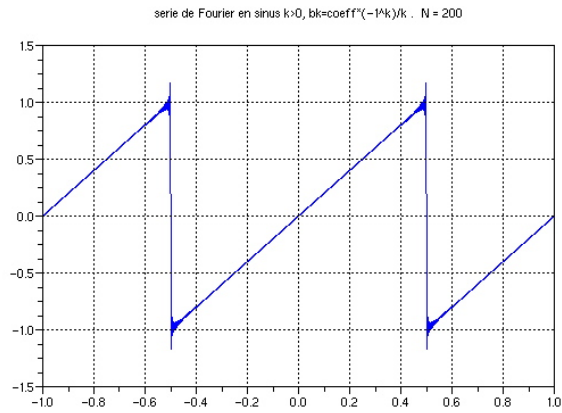
$$f(x) = \sum_{n=1}^{\infty} \frac{1}{n} \cos(nx)$$

```
//Terme général : cos(nt)/n
clear
N=100; // longueur de la série
t=0.1 :0.005 :0.9;
k=1 :N;
TT=2*%pi*(k'*t);
S=cos(TT);
coef=ones(k)./k;
SS=diag(coef)*S;
seriP=-(1/%pi)*sum(SS,1);
scf(1)
clf(1)
plot2d(t,seriP,2)
xgrid()
xtitle('serie de Fourier en cos, k>0, ak=1/k . N = ' + string(N)) tt=2*%pi*t;
plot2d(t,(1/%pi)*log(2*sin(tt./2)),5)
```

### Exemple. Scilab. 2.

$$f(x) = -\frac{2}{\pi} \sum_{n=1}^{\infty} (-1)^n \frac{1}{n} \sin(nx)$$

```
clear
N=200;
t=-1 :0.0005 :1;
k=1 :N;
TT=2*%pi*(k'*t);
S=sin(TT);
coef=(-1)^k./k;
SS=diag(coef)*S;
seriP=(-2/%pi)*sum(SS,1);
scf(2); clf(2);
plot2d(t,seriP,2);
xgrid()
m=max(seriP);
xtitle('Serie de Fourier en sinus k>0, bk= coeff*(-1^k)/k . N = ' + string(N))
```



### Exemple. Scilab. 3

```

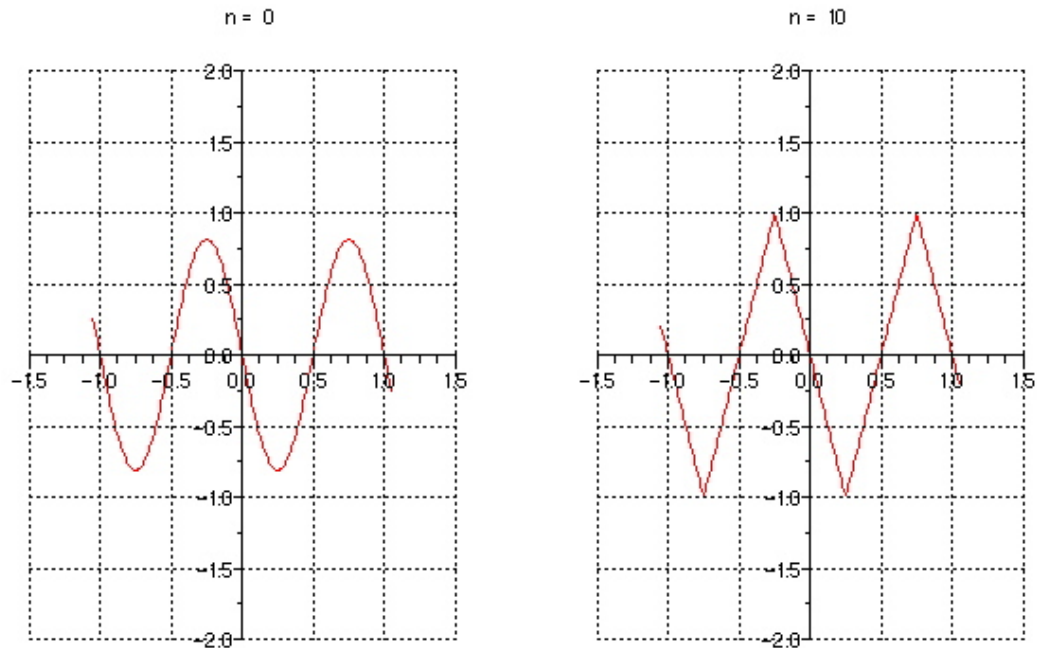
clear
t=-1.05 :0.01 :1.05 ;//Intervalle d'étude
n=0 ;
u=1 :2 :2*n+1 ;
TT=2*%pi*(u'*t) ;//matrice
S=sin(TT) ;
v=u.^2 ;
w=(-1).^(1 :n+1) ;
SS=diag(w./v)*S ;
FF=sum(SS,1).*(8./(%pi^2)) ;
scf(1) ; clf(1) ;
subplot(1,2,1) ;
plot2d(t,FF,5) ;
a=gca() ;
a.x_location="middle" ;
a.y_location="middle" ;
a.isoview="on" ;
xtitle(" n = "+ string(n))
xgrid()
n=10 ;
u=1 :2 :2*n+1 ;
TT=2*%pi*(u'*t) ;
S=sin(TT) ;
v=u.^2 ;
w=(-1).^(1 :n+1) ;
SS=diag(w./v)*S ;
FF=sum(SS,1).*(8./(%pi^2)) ;
subplot(1,2,2) ;
plot2d(t,FF,5) ;
a=gca() ;

```

```

a.x_location="middle";
a.y_location="middle";
a.isoview="on";
xtitle(" n = "+ string(n));
xgrid();

```



### Exemple. Scilab 4 : Phénomène de Gibbs

Soit la fonction  $f$  de période  $T = 1$  et égale à  $-1$  sur  $[-1/2, 0[$  et égale à  $1$  sur  $[0, 1/2[$  il est clair que  $f$  est continue par morceau et vérifie les conditions de Dirichlet. La série de Fourier associée est alors :

$$S^f(x) = \frac{4}{\pi} \sum_{k=0}^{\infty} \frac{1}{2k+1} \sin((2k+1)2\pi x), \quad S_n^f(x) = \frac{4}{\pi} \sum_{k=0}^n \frac{1}{2k+1} \sin((2k+1)2\pi x)$$

Le script suivant représentant l'évolution de la série de Fourier, lorsqu'on augmente la somme des harmoniques, permet de constater que  $S_n^f(x)$  converge vers  $f$  en tout point de continuité de  $f$ .

Pourtant, au voisinage de  $0$ , constate une anomalie appelée "phénomène de Gibbs". le max de  $S_n^f(x)$  tend vers une constante  $\gamma$  nommée constante de Gibbs. On peut montrer que :

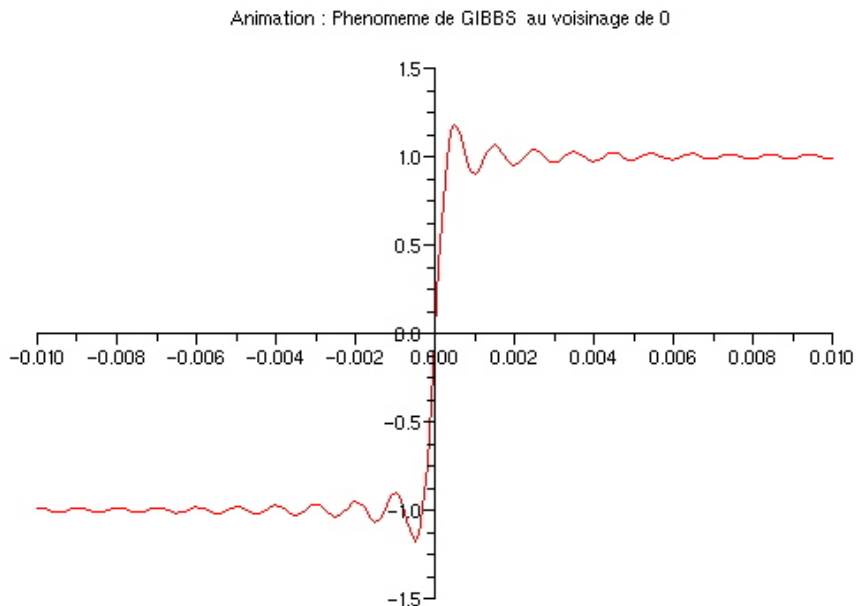
$$\gamma = \frac{2}{\pi} \int_0^{\pi} \frac{\sin(x)}{x} dx$$

---

```

clear()
hf=scf(1);
xselect();
clf(1) :
hf.figure_name='Convergence d''une serie de Fourier';
xtitle(" Animation : Phenomeme de GIBBS au voisinage de 0 ")
hf.pixmap='on' ;//début de l'animation
t=-0.01 :0.0001 :0.01 ;//Intervalle d'étude
GG=zeros(t);
a=gca();
a.x_location="middle";
a.y_location="middle";
for n=1 :25 :501
plot2d(t,GG,8)
u=1 :2 :2*n+1;
TT=2*%pi*(u'*t) ;//matrice
S=sin(TT);
ZZ=diag(ones(u)./u)*S;
GG=sum(ZZ,1).*(4./%pi);
plot2d(t,GG,5)
show_pixmap()
end
m=max(GG);
hf.figure_name= "Le graphe passe 0." + " Le max atteint : gamma = "+string(m);

```



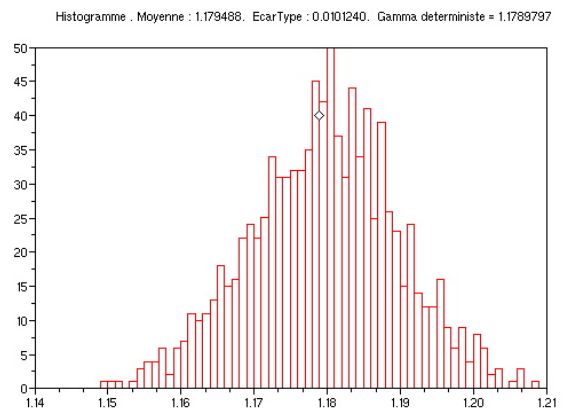
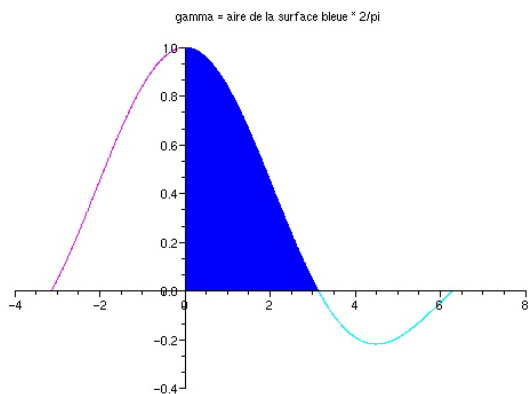
Le programme précédent permet d'évaluer graphiquement la constante  $\gamma$ . On peut calculer la valeur approchée de  $\gamma$  par différentes méthodes. `scilab` possède la fonction `integrate` (voir `help`).

On peut également utiliser la méthode de Monte-Carlo comme le montre le script suivant.

```

clear
// début approx déterministe
Gintegrate=integrate('sinc(x)', 'x',0,%pi)*2./%pi;
// fin approx déterministe
rand('u')
n=10000; //nombre de points pour chaque évaluation par Monté-Carlo
k=1000; //nombre d'évaluations
Gibbs=zeros(1,k); //initialisation
for j=1 :k
x=%pi*rand(1,n);
y=rand(1,n);
z=x.*y;
T=(z<sin(x));
u=2*sum(1*T)./n;
Gibbs(j)=u; end
scf(1); clf(1);
plot2d(1 :k,Gibbs,2)
plot2d(1 :k,Gintegrate*ones(1 :k),1)
xtitle('Affichage des resultats de chaque essais. Approx deterministe : '+string(Gintegrate))
xselect()
scf(2);
clf(2);
s=(Gintegrate-0.03) :0.001 :(Gintegrate+0.03);
histplot(s,Gibbs,5)
plot2d(Gintegrate,40,-5);
xselect()
GGIBBS=mean(Gibbs);
Ecar_Type=st_deviation(Gibbs);
xtitle('Histogramme . Moyenne : '+string(GGIBBS)+ '. EcarType : '+ string(Ecar_Type)+...
". Gamma deterministe = "+ string(Gintegrate))

```



---

## Utilisation de la fonction Scilab ode pour évaluer $\gamma$

$$\gamma = \frac{2}{\pi} \int_0^{\pi} \frac{\sin(t)}{t} dt$$

On détermine tout simplement la primitive de  $\frac{\sin(t)}{t}$  sur  $[0, \pi]$  ou ici sur  $[0, 100]$  s'annulant en 0, au moyen d'une équation différentielle :  $y' = \frac{\sin(t)}{t}$

```
//clear
function yprim=fct(t,y)
yprim=sinc(t);
endfunction
y0=0;
t0=0;
T=t0 :0.01 :100;
sol=ode(y0,t0,T,fct);
clf();
plot2d(T,(2/%pi)*sol,5);
plot2d(T,ones(T),4);
g=(2/%pi)*max(sol);
xtitle(" la valeur de gamma est ici : "+string(g));
```

Cela donne le graphe suivant pour  $F(t) = \frac{2}{\pi} \int_0^t \frac{\sin(x)}{x} dx$  :

la valeur de gamma est ici : 1.1789796

