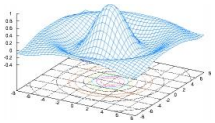


Logiciel de calcul scientifique (Octave)

Julien Ah-Pine

julien.ah-pine@eric.univ-lyon2.fr

Licence IDEA 2e année - Université Lumière Lyon 2

Motivations de ce cours

- Maîtrise d'un logiciel de calcul numérique :
  - Pour mettre en œuvre sur des cas pratiques les méthodes et modèles quantitatifs théoriques appris dans les autres cours
  - Pour visualiser les données par l'intermédiaire de graphes
  - Etre opérationnel dans la suite de votre parcours universitaire (statistiques, mathématiques, économétrie, …) et professionnel (implémentation d'algorithmes et de prototypes)

Octave ?

- Logiciel de calcul numérique compatible en grande partie avec **Matlab**
- Langage de haut niveau interprété
  - Eléments redimensionnés dynamiquement
  - On parle de **scripts** plutôt que de programmes (par opposition à des langages classiques comme C/C++)

Pourquoi Octave ?

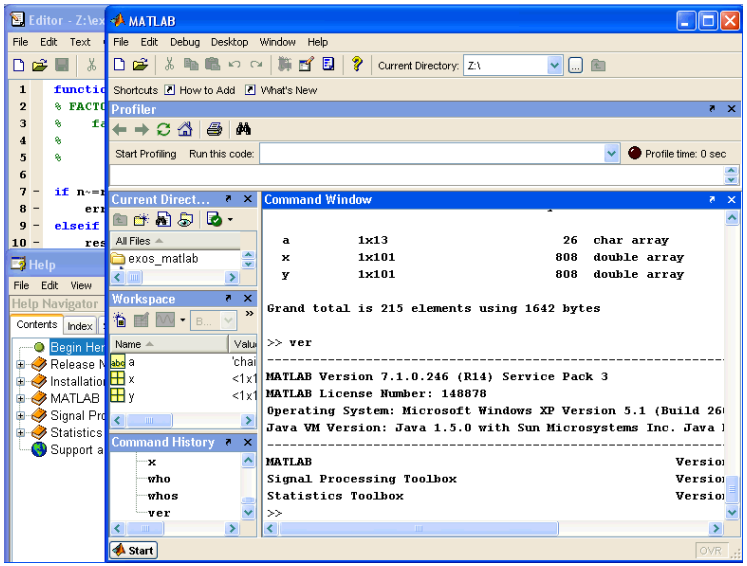
- Logiciel libre (Licence GPL -*GNU General Public Licence*-)
- Disponible pour de multiples plate-formes

Comparaison Matlab / Octave

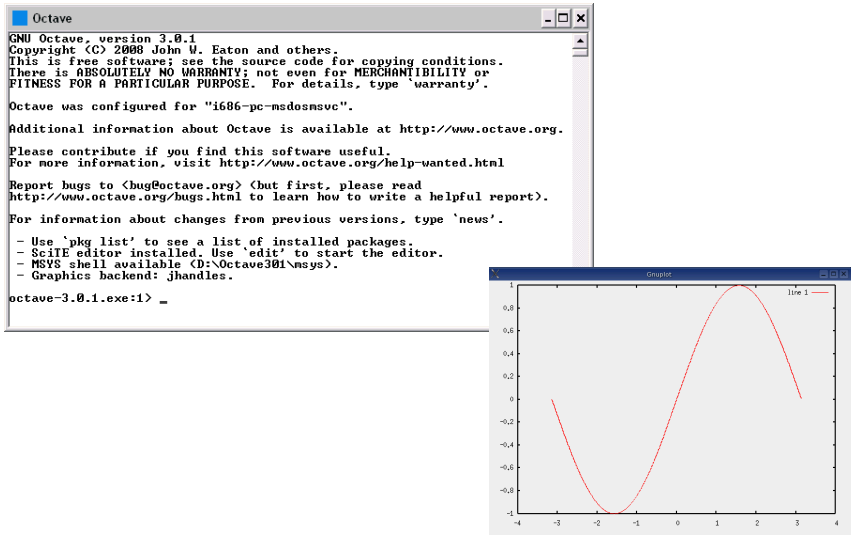
- **Matlab :** logiciel commercial, nombreuses *toolboxes*, IDE intégré (animations), GUI

- **Octave :** licence GPL, distribution standard, système de packages

Matlab



Octave



Plan et organisation du cours

- 4 CM :
  - Premiers pas sous Octave (CM1)
  - Syntaxe du langage Octave (CM2)
  - Graphes 2D sous Octave (CM3)
  - Programmation sous Octave (CM4)

- 8 TD :
  - 2 groupes de TD

Emploi du temps du cours

Date	14h	16h
23/09/2010	**CM1**	
30/09/2010	**CM2**	TD1/GR1
07/10/2010	*TD1/GR2*	**CM3**
14/10/2010	*TD2/GR2*	TD2/GR1
21/10/2010	*TD3/GR2*	TD3/GR1
04/11/2010	**CM4**	
18/11/2010	*TD4/GR2*	TD4/GR1
25/11/2010	*TD5/GR2*	TD5/GR1
02/12/2010	*TD6/GR2*	TD6/GR1
09/12/2010	*TD7/GR2*	TD7/GR1
16/12/2010	*TD8/GR2*	TD8/GR1

Modalités de l'évaluation (à confirmer)

- Contrôle continu (30%)
- Examen final sur machine (70%)

CM1 : Premiers pas

- Lancer Octave
- Aide en ligne
- Quelques illustrations de commandes
- Types de données
- Variables
- Fonctions

Débuter sous Octave

- Téléchargement sur http://www.octave.org/download.html
- Systèmes Windows, Linux, MacOS
- Interpréteur Octave :

```
$ octave
GNU Octave, version 2.1.53 (i586-mandrake-linux-gnu).
Copyright (C) 2004 John W. Eaton.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTIBILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type `warranty'.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/help-wanted.html

Report bugs to <bug-octave@bevo.che.wisc.edu> (but first, please read
http://www.octave.org/bugs.html to learn how to write a helpful report).

octave:1>
```

Aide en ligne de commande

- help *commande*

```
octave-3.0.1:22> help plot
 -- Function File:  plot (Y)
 -- Function File:  plot (X, Y)
 -- Function File:  plot (X, Y, PROPERTY, VALUE, ...)
 -- Function File:  plot (X, Y, FMT)
 -- Function File:  plot (H, ...)
    Produces two-dimensional plots.  Many different combinations of
    arguments are possible.  The simplest form is

        plot (Y)

    where the argument is taken as the set of Y coordinates and the X
    coordinates are taken to be the indices of the elements, starting
    with 1.
[etc.]
```


Premiers pas

- Additions, soustractions, etc.

```
octave-3.0.1:1> 2+2
ans =  4
octave-3.0.1:2> 12.3-7.4
ans =  4.9000
octave-3.0.1:3>
```

- Introduction de variables

```
octave-3.0.1:3> x = 12.3
x =  12.300
octave-3.0.1:4> 3 - x
ans = -9.3000
octave-3.0.1:5> y = x - 3.2
y =  9.1000
octave-3.0.1:6>
```


Premiers pas

- Vecteurs et matrices

```
octave-3.0.1:6> a = [ 1 , 4 , 4 ]
a =

  1  4  4

octave-3.0.1:7> b = [ 1 , 1 , 1 ]
b =

  1  1  1

octave-3.0.1:8> a + b
ans =

  2  5  5
```


Premiers pas

```
octave-3.0.1:9> a * b
error: operator *: nonconformant arguments (op1 is 1x3, op2 is 1x3)
error: evaluating binary operator `*' near line 9, column 3
octave-3.0.1:9> a * 2
ans =
  2  8  8

octave-3.0.1:10> c = [ 2 ; 3 ; 1 ]
c =
  2
  3
  1

octave-3.0.1:11> a * c
ans =  18
octave-3.0.1:12> a * b'
ans =  9
```

Résoudre des équations

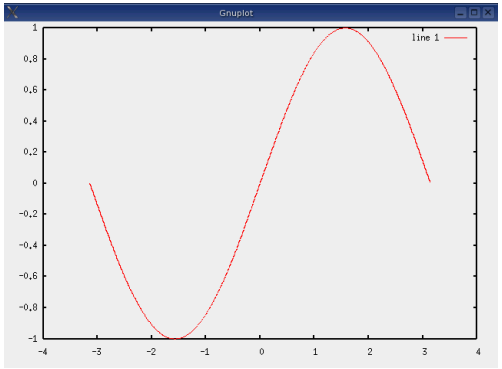
- Equation linéaire $\begin{cases} x + 3y - 2z = -3 \\ 3x - 4y + 3z = 28 \\ 5x - 5y + 4z = 7 \end{cases}$

```
octave-3.0.1:14> A = [ 1, 3, -2; 3, -4, 3; 5, 5, -4 ]
A =
   1   3  -2
   3  -4   3
   5   5  -4
octave-3.0.1:15> b = [ -3; 28; 7 ]
b =
  -3
  28
   7
octave-3.0.1:16> A \ b
ans =
   5.0000
   2.0000
   7.0000
```


Imprimer des courbes

- Utilisation de Gnuplot pour l'affichage

```
octave-3.0.1:18> x = [ -pi : 0.01 : pi] ;
octave-3.0.1:20> plot( x , sin(x) )
```



Types de données

- Types numériques: entiers, réels, complexes…
- Chaînes de caractères
- Tableaux : vecteurs, matrices…
- Types composés : structures, tableaux cellulaires…

Les variables

- En mémoire durant la session dans l'espace de travail (*workspace*)
- Pas besoin de déclarer le type des variables *a priori*
- Un nom de variable est valide s'il débute par une lettre et est suivi par un nombre quelconque de lettres, chiffres et _
- Par exemple : x_min, A351, très_longue_variable

Les variables

- Les noms sont sensibles à la casse (*case-sensitive*) : MAT_A est différent de mat_a
- Une expression est une construction valide faisant usage de nombres, de variables, d'opérateurs et de fonctions
- Par ex. : pi*r^2, sqrt((b^2)-(4*a*c))
- On peut stocker la valeur d'une expression dans une variable : variable = expression

Environnement d'Octave

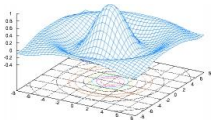
- clear *A* : efface la variable A
- clear : efface tout
- whos: liste détaillée des variables
- save *nom_fichier* : sauvegarde la session
- load *nom_fichier* : charge une session

Les fonctions

- Fonctions prédéfinies en Octave (ex : sqrt)
- Ces fonctions sont implémentées soit au niveau du noyau (*built-in*), soit au niveau des M-files et packages
- L'utilisateur peut créer ses propres fonctions
- Dans ce cas, ce sont des suites de commandes Octave regroupées sous un nom de fonction permettant de commander leur exécution

Sources

- Cours de J. Velcin
- http://www.gnu.org/software/octave/
- http://www.octave.org/download.html
- http://enacit1.epfl.ch/cours_matlab/
- http://www.gnu.org/software/octave/doc/interpreter/index.html



Syntaxe du langage Octave

Julien Ah-Pine
julien.ah-pine@eric.univ-lyon2.fr

Licence IDEA 2e année - Université Lumière Lyon 2

CM2 : Syntaxe du langage Octave

- Généralités
- Opérateurs (arithmétiques, relationnels, logique)
- Fonctions (mathématiques, logiques)
- Séries, vecteurs, matrices
- Opérateurs et fonctions matriciels
- Chaînes de caractères
- Dates et heures
- Structures
- Tableaux cellulaires
- Entrées/Sorties

Caractères spéciaux

Caractère	Description
;	exécute la commande sans affichage
,	séparateur de commandes, délimiteur
:	opérateur de définition de séries et de plages d'indices
#	commentaires
'	délimite les chaînes de caractères, transposition

Formatage des nombres

Commande	Type d'affichage	Exemple
format {short {e} }	notation fixe à 5 chiffres e = notation flottante avec exposant	72.346 7.2346e+001
format long {e}	notation plus précise à 15 chiffres e = avec exposant	72.2137210937092 7.22137210937092e+001
format bank	format monétaire	72.35
format hex	en base hexadécimale	3325ef25a001
format rat	approximation avec des rationnels	10/3

Les packages d'Octave

- On parle de « toolboxes » pour Matlab
- Disponible à l'adresse suivante : http://octave.sourceforge.net/packages.html
- Penser à charger les packages à chaque session d'utilisation (commande : pkg load|unload *package*)
- La liste des fonctions avec les packages : http://octave.sourceforge.net/doc/index.html

Environnement d'Octave

- Evaluation des commandes, fonctions, expressions réalisée dans un ordre bien précis
- **Attention** à ne pas utiliser les noms prédéfinis
- Le *path* est défini dans deux variables :

DEFAULT_LOADPATH	fonctions d'Octave de base
LOAD_PATH	fonctions définies par l'utilisateur

- Pour modifier le LOAD_PATH, il faut utiliser les commandes path et addpath et les ajouter au fichier de démarrage .octaverc

Commandes en relation avec l'OS

Commande	Effet
dir {chemin\} {fichier(s)} ls {chemin\} {fichier(s)}	Affiche la liste des fichiers du répertoire courant ou du répertoire spécifié
type {chemin\} fichier	Affiche le contenu du M-file spécifié
delete fichier(s) unlink('fichier')	Détruit le ou les fichier(s) spécifié(s)
mkdir('répertoire')	Crée le sous-répertoire spécifié
rmdir('répertoire')	Détruit le sous-répertoire spécifié
rename('nom1','nom2')	Renomme le fichier nom1 avec nom2
system('commande')	La commande spécifiée est exécutée par l'OS
computer	Retourne le type de machine sur laquelle on exécute Octave
version	Retourne le numéro de version d'Otave

Constantes et variables d'Octave

Constante	Description
pi	3.14159265358979
i ou j	racine de -1 (nombre imaginaire)
e ou exp(1)	2.71828182845905
Inf ou inf	infini
NaN ou nan	indéterminé
realmin	environ 2.2e-308
realmax	environ 1.7e+308
eps	environ 2.2e-16
Variable	
ans	variable réponse par défaut (*answer*)
nargin	nombre d'arguments passés à une fonction
nargout	nombre d'arguments retournés par une fonction

Opérateurs arithmétiques

Opérateur	Description
+	Addition
{var2=} ++var1	Equivalent à var1=var1+1 ; var2=var1 ;
{var2=} var1++	Equivalent à var2=var1 ; var1=var1+1 ;
-	Soustraction
{var2=} --var1	Equivalent à var1=var1-1 ; var2=var1 ;
{var2=} var1--	Equivalent à var2=var1 ; var1=var1-1 ;
*	Multiplication
/	Division à droite
\	Division à gauche
^	Puissance
( )	Parenthèses

Opérateurs relationnels

Opérateur	Description
==	Test d'égalité
~=	Test de différence
<	Test d'infériorité
>	Test de supériorité
<=	Test d'infériorité ou égalité
>=	Test de supériorité ou égalité

Opérateurs logiques

Opérateur	Description
~ exp	Négation logique
exp1 & exp2	ET logique
exp1 | exp2	OU logique
xor (exp1 , exp2)	OU EXCLUSIF logique

Fonctions mathématiques

Fonction	Description
sqrt(var)	Racine carrée de var
exp(var)	Exponentielle de var
log(var) et log10(var)	Logarithme naturel, resp. base 10, de var
cos(var) et acos(var)	Cosinue, resp. arc cosinus, de var. Angle exprimé en radian
sin(var) et asin(var)	Sinus, resp. arc sinus, de var. Angle exprimé en radian
factorial(n)	Factorielle de n (c'est-à-dire : n*(n-1)*(n-2)*...*1)
rand, rand(n), rand(n,m)	Génère un nombre / une matrice de nombres aléatoires compris entre 0.0 et 1.0
abs(var)	Valeur absolue (positive) de var
mod(var1,var2) rem(var1,var2)	Fonction var1 "modulo" var2 Reste ("remainder") de la division de var1 par var2
fix(var), round(var), floor(var), ceil(var)	Fonctions d'arrondi

Fonctions logiques

Fonction	Description
isinf(var)	Vrai si la variable var est infinie positive ou négative (Inf ou -Inf)
isnan(var)	Vrai si la variable var est indéterminée (NaN)
isfinite(var)	Vrai si la variable var n'est ni infinie ni indéterminée
isepmty(var)	Vrai si la variable var est vide (de dimension 1x0), faux sinon
isstr(var) ou ischar(var)	Vrai si var est une chaîne de caractères, faux sinon
exist ('objet' {,'var|builtin|file|dir'})	Vérifie si l'objet spécifié existe. Retourne "1" si c'est une variable, "2" si c'est un M-file, "3" si c'est un MEX-file, "4" si c'est un MDL-file, "5" si c'est une fonction builtin, "6" si c'est un P-file, "7" si c'est un directoire. Retourne "0" si aucun objet de l'un de ces types n'existe

Séries

- Construction de séries linéaires
- Syntaxe : début:fin
- Ajout d'un pas : début:pas:fin
- Avec un nombre d'éléments déterminé :
  - échelle linéaire : linspace(début,fin {,nbval})
  - échelle logarithmique : logspace(début,fin {,nbval})

Vecteurs

- Premier élément numéroté 1 (et non 0)
- Affectation avec les [] et adressage avec ()
- Exemple de vecteur ligne :

```
octave-3.0.1:1> Vec = [ 1 5 10 ]
Vec =
   1    5   10
```

- Exemple de vecteur colonne :

```
octave-3.0.1:2> Vec2 = [ 1 ; 2 ; 3 ]
Vec2 =
  1
  2
  3
```

Vecteurs

Syntaxe	Description
vec = [val1 val2 val3 ...] = [val var expr ...]	Création d'un vecteur ligne vec contenant les valeurs val, variables var, ou expressions expr spécifiées.
vec = [val ; var ; expr ...] = [val1 val2 ...] = [var val var val ...]'	Création d'un vecteur colonne vec contenant les valeurs val (ou variables var, ou expressions expr) spécifiées.
vec'	Transposée du vecteur vec
vec = début{:pas}:fin	Initialisation d'un vecteur ligne vec à une série linéaire
vec = linspace(début,fin{,n}) var = logspace(début,fin{,n})	Initialisation d'un vecteur ligne vec à une série linéaire, respectivement logarithmique

Vecteurs

Syntaxe	Description
vec(i)	i-ème élément du vecteur ligne ou colonne vec
vec(i{:p}:j) vec(i{:p}:end)	Adressage des éléments d'indice i à j du vecteur ligne ou colonne vec avec un pas de "1" ou de "p" si spécifié
vec([i j k:l])	La notation d'indices entre crochets [] permet de désigner un ensemble continu ou discontinu d'éléments.
vec(i { {:p} :j}) = *val*	Etend la taille du vecteur à i ou j éléments en affectant aux i-ème à j-ème éléments la valeur val spécifiée
for k=i{:p}:j vec(k)=*expression* end	Initialise les éléments du vecteur ligne vec par l'expression spécifiée
vec(i:j)=[] vec([k l m])=[]	Destruction des éléments du vecteur vec
length(vec)	Retourne la taille (nombre d'éléments) du vecteur vec

Matrices

- Tableau rectangulaire à 2 dimensions NxM

1	0	0	4
1	1	-4	0
2	2	0	0
0	0	0	1
3	0	0	1

- Généralisation des vecteurs
- Numéro de ligne et numéro de colonne séparés par une virgule

Matrices

Syntaxe	Description
mat=[v11 v12 ... v1m ; v21 v22 ... v2m ; ... ... ; vn1 vn2 ... vnm]	Définit une matrice mat de n lignes x m colonnes dont les éléments sont initialisés aux valeurs v_{ij}.
mat=[vco1 vco2 ...] mat=[vli1 ; vli2 ; ...]	Construit la matrice mat par concaténation de vecteurs colonne vco_i ou de vecteurs ligne vli_i spécifiés.
[mat1 mat2 {mat3...}] [mat4; mat5 {; mat6...}]	Concaténation de matrices (ou vecteurs).
ones(n{,m})	Renvoie une matrice de n lignes x m colonnes dont tous les éléments sont égaux à "1".
zeros(n{,m})	Renvoie une matrice de n lignes x m colonnes dont tous les éléments sont égaux à "0".
eye(n{,m})	Renvoie une matrice identité de n lignes x m colonnes dont les éléments de la diagonale principale sont égaux à "1" et les autres éléments sont égaux à "0".

Matrices

Syntaxe	Description
diag(vec) diag(mat)	Appliquée à un vecteur vec ligne ou colonne, cette fonction retourne une matrice carrée dont la diagonale principale porte les éléments du vecteur vec et les autres éléments sont égaux à "0". Appliquée à une matrice mat (qui peut ne pas être carrée), cette fonction retourne un vecteur-colonne formé à partir des éléments de la diagonale de cette matrice
mat2 = repmat(mat1,M,N)	Renvoie une matrice mat2 formée à partir de la matrice mat1 dupliquée en "tuile" M fois verticalement et N fois horizontalement
mat=[]	Crée une matrice vide mat de dimension 0x0
[n m] = size(var)	Renvoie, sur un vecteur ligne, la taille (nombre n de lignes et nombre m de colonnes) de la matrice ou du vecteur var.

Matrices

Syntaxe	Description
length(mat)	Retourne le plus grand nombre parmi le nombre de lignes et de colonnes d'une matrice.
mat(i,j)	Désigne l'élément (i,j) de mat.
mat(i:j,k:m)	Désigne la partie de la matrice mat dont les éléments se trouvent dans les lignes i à j et dans les colonnes k à m.
mat([lignes],[cols])	La notation d'indices entre crochets [] permet de désigner un ensemble continu ou discontinu de lignes et/ou de colonnes.
mat(i) mat(i:j)	Recherche effectuée en numérotant les éléments de la matrice colonne après colonne.
mat(i:j,:)=[] et mat(:,k:m)=[]	Destruction de lignes ou de colonnes d'une matrice.

Opérateurs matriciels

Opérateur	Description
+ -	Addition Soustraction
*	Produit matriciel
.*	Produit élément par élément
\	Division matricielle à gauche **Attention :** A\B est la solution "X" du système linaire "A*X=B".
/	Division matricielle à droite **Attention :** B/A est la solution "X" du système "X*A=B.
./	Division à droite élément par élément
.\	Division à gauche éléments par éléments
^	Elévation à la puissance matricielle
.^	Elévation à la puissance éléments par éléments
[,] et [;] ou cat	Concaténation horizontale, respectivement verticale

Fonctions matricielles

Fonction	Description
‘	Transposition normale de matrices réelles et transposition conjuguée de matrices complexes.
reshape(var,M,N)	Cette fonction de redimensionnement retourne une matrice de M lignes x N colonnes contenant les éléments de var.
vec = mat(:)	Déverse la matrice mat colonne après colonne sur le vecteur-colonne vec.
sort(var)	Fonction de tri par éléments.
sortrows(mat {,no_col})	Trie les lignes de la matrice mat dans l'ordre croissant des valeurs de la première colonne, ou dans l'ordre croissant des valeurs de la colonne no_col
fliplr(mat) flipud(mat)	Retournement de la matrice mat par symétrie horizontale (left/right), respectivement verticale (up/down)
rot90(mat {,K})	Effectue une rotation de la matrice mat de K fois 90 degrés dans le sens inverse des aiguilles d'une montre.

Fonctions mathématiques sur vecteurs et matrices

- Les fonctions mathématiques présentées précédemment peuvent aussi être utilisées sur des vecteurs et matrices.
- Elles s'appliquent alors à tous les éléments et retournent donc également des vecteurs ou des matrices.
- **Par ex. :** y=sin(0:0.1:2*pi)

Fonctions de calcul matriciel et statistiques

Fonction	Description
norm(vec)	Calcule la norme (longueur) du vecteur vec.
dot(vec1,vec2)	Calcule la produit scalaire des 2 vecteurs vec1 et vec2.
cross(vec1,vec2)	Calcule la produit vectoriel (en 3D) des 2 vecteurs vec1 et vec2 (ligne ou colonne, mais qui doivent avoir 3 éléments !).
inv(mat)	Inversion de la matrice carrée mat. Une erreur est produite si la matrice est singulière.
det(mat)	Retourne le déterminant de la matrice carrée mat.
trace(mat)	Retourne la trace de la matrice mat, c'est-à-dire la somme des éléments de sa diagonale principale.
rank(mat)	Retourne le rang de la matrice mat, c'est-à-dire le nombre de lignes ou de colonnes linéairement indépendants.
min(var{,d}) max(var{,d})	Appliquées à un vecteur ligne ou colonne, ces fonctions retournent le plus petit, resp. le plus grand élément du vecteur.

Fonctions de calcul matriciel et statistiques

Fonction	Description
sum(var{,d}) prod(var{,d})	Retourne la somme ou le produit des éléments du vecteur / de la matrice.
mean(var{,d})	Retourne la moyenne arithmétique des éléments du vecteur / de la mtrice.
std(var{,f{,d}})	Retourne l'écart-type des éléments du vecteur / de la matrice.
median(var{,d})	Calcule la médiane.
cov	Retourne vecteur ou matrice de covariance.
eig, svd, cond	Fonctions en relation avec vecteurs propres et valeurs propres.
lu, chol, qr, qzhess, schur, svd, housh, krylov...	Fonctions en relation avec les méthodes de décomposition/factorisation de type : - LU, Cholesky, QR, Hessenberg, - Schur, valeurs singulières, householder, Krylov...

Chaînes de caractères

- Stockées sous la forme de vecteurs-ligne
- Affectation :

```
octave-3.0.1:1> ch = 'ma chaine de caracteres'
ch = ma chaine de caracteres
octave-3.0.1:2> ch2 = "ma chaine avec\ndes\t\"caracteres
speciaux\""
ch2 = ma chaine avec
des     "caracteres speciaux"
```

- string(i:j) retourne une partie de la chaîne

```
octave-3.0.1:3> ch(4:12)
ans = chaine de
```

Chaînes de caractères

Fonction	Description
[s1 s2 s3...] strcat(s1,s2,s3...)	Concatène les chaînes s1, s2, s3...
ms = str2mat(s1,s2,s3...) ms = char(s1,s2,s3...) ms = [s1 ; s2 ; s3 ...]	Produit une **matrice de chaînes de caractères** ms contenant la chaîne s1 en 1ère ligne, s2 en seconde ligne, s3 en 3ème ligne, etc...
ms(i,:) ms(i,j:k)	Retourne la i-ème ligne de la matrice de chaînes ms, respectivement la sous-chaîne de cette ligne allant du j-ème au k-ème caractère.
length(string)	Retourne le nombre de caractères de la chaîne string.
deblank(string) blanks(n)	Supprime les caractères <espace> terminant la chaîne. Retourne une chaîne de n caractères <espace>.
findstr(string,s1)	Retourne la position dans string de toutes les chaînes s1 qui ont été trouvées.

Chaînes de caractères

Fonction	Description
strmatch(ms,s1 {,'exact'})	Retourne les numéros des lignes de la matrice de chaîne ms qui 'commencent' par la chaîne s1.
regexp(ms, pattern)	Effectue une recherche dans ms à l'aide du motif défini par l'expression régulière pattern.
strrep(string, s1, s2)	Retourne une copie de la chaîne string dans laquelle toutes les occurences de s1 sont remplacées par s2.
split(string, sep)	Découpe la chaîne string selon la chaîne-séparateur sep.
sortrows(ms)	Trie par ordre alphabétique croissant les lignes de la matrice de chaînes ms.
strcmp(string1, string2)	Compare les 2 chaînes string1 et string2 : retourne 1 si elles sont identiques, 0 sinon.
isstr(var) ou ischar(var) isletter(string) isspace(string)	Retourne 1 si var est une chaîne de caractères, 0 sinon. Retourne un vecteur qui teste chacun des caractères de string.

Chaînes de caractères

Fonction	Description
lower(string) upper(string)	Convertit la chaîne string en minuscules, respectivement en majuscules.
abs(string) double(string)	Convertit les caractères de la chaîne string en leurs codes décimaux selon la table ASCII ISO-Latin-1.
char(var)	Convertit les nombres de la variable var en caractères.
num2str(val)	Permet de convertir val en une chaîne de caractères.
mat2str(mat {,n})	Convertit la matrice mat en une chaîne de caractère incluant les crochets [ ].
sscanf(string,format)	Permet de récupérer le(s) nombre(s) se trouvant dans la chaîne string.
str2num(string)	Convertit en nombres le(s) nombre(s) se trouvant dans la chaîne string.
eval(expression)	Évalue (exécute) l'expression Octave spécifiée.

Polynômes

- Représenté par ses coefficients (ordre déc.)
- Par ex., $p(x) = 3x^3-5x+1.5$

```
octave-3.0.1:31> p = [ 3  0  -5  1.5 ]
p =
   3.0000  0.0000   -5.0000   1.5000
```

- Instanciation :

```
octave-3.0.1:34> polyval(p,3)
ans =  67.500
```

Polynômes

Fonctions	Description
polyval(c,x) polyval(c,vec)	Evalue le polynôme c avec la valeur x. Evalue le polynôme c avec les valeurs contenues dans vec.
polyderiv(c)	Retourne les coefficients du polynôme dérivé de c.
[p,yf] = polyfit(x,y,n)	Retourne le polynôme p de degré n qui approxime au mieux la courbe des points décrits par les vecteurs x et y.
polyint(c)	Retourne les coefficients de l'intégrale du polynôme c.
polyreduce(c)	Réduit un polynôme en retirant les coefficients inutiles.
roots(c)	Retourne la ou les racines du polynômes.
polyout(c {,x})	Affiche le polynôme c. L'option x permet de préciser un nom de variables sous la forme d'une chaîne de caractères.

Date et temps

- Gérés sous forme de nombres
- Origine fixée au 1er janvier de l'an 0
  datestr(1.00001)
- Incrémenté de 1 chaque jour
- Les heures, minutes et jours sont des fractions

Date et heure courantes

Fonctions	Description
now	Retourne le nombre exprimant la date et heure locale courante. **rem(now,1)** pour la partie décimale **floor(now)** pour la partie entière **datestr(rem(now,1),'HH:MM:SS')** retourne l'heure courante sous forme de chaîne.
date	Retourne la date courante sous forme de chaîne de caractère au format 'dd-mmm-yyyy'.
clock	Retourne la date et heure courante sous forme d'un vecteur-ligne de 6 valeurs numériques. **fix(clock)** pour avoir des valeurs entières.

Conversion de dates

Fonction	Description
datenum(annee, mois, jour {, heure, minute, seconde }) datenum(date_string)	Retourne le nombre exprimant la date spécifiée par la chaîne date_string. Par exemple, datenum(2005,4,8,20,45,0) et datenum('08-Apr-2005 20:45:00') retournent le nombre 732410.8645833334.
datestr(date_num, format)	Convertit en chaîne de caractères la date/heure date_num spécifiée de façon numérique.
vec = datevec(date)	Retourne un vecteur ligne de 6 valeurs numériques définissant l'année, mois, jour, heure, minutes, secondes.

Diverses fonctions

Fonction	Description
calendar calendar(annee,mois) calendar(date)	Retourne une matrice 6x7 contenant le calendrier du mois courant, ou d'une date spécifiée.
vec = weekday(date)	Retourne le numero_jour (nombre) et nom_jour (chaîne) correspondant à la date spécifiée.
eomday(annee,mois)	Retournent le nombre de jours du mois/annee (spécifié par des nombres).
cputime	Retourne le nombre de secondes de processeur consommées par Octave depuis le début de la session.
tic val = toc	La fonction tic démarre un compteur de temps, et la fonction toc l'arrête en retournant le temps écoulé en secondes.
etime(t2,t1)	Retourne le temps, en secondes, séparant l'instant t1 de t2.
pause(secondes)	Se met en attente durant le nombre de secondes spécifié.

Structures

- Une structure est un objet se composant de plusieurs champs qui peuvent être de types différents (chaîne, matrice, vecteur, etc.).
- Accès aux champs d'une structure avec la syntaxe suivante : structure.champs
- Possibilité de créer des tableaux de structures
- Possibilité de sauvegarder les structures sous forme de texte : save -text …

Structures

- Exemple d'un individu :
  - nom
  - prénom
  - âge
  - date de naissance
  - adresse
  - marié
  - enfants

  etc.

Structures

```
octave-3.0.1:1> individu.nom = 'Macrez'
individu =
{
  nom = Macrez
}
octave-3.0.1:2> whos individu
*** local user variables:

  Prot Name          Size                Bytes  Class
  ==== ====          ====                =====  =====
   rwd individu      1x1                     6  struct
Total is 1 element using 6 bytes
octave-3.0.1:3> individu.prenom = 'Ferdinand'
individu =
{
  nom = Macrez
  prenom = Ferdinand
}
```

Structures

- Création d'une structure :

```
personne.nom='Dupond';
personne.prenom='Jules';
personne.age=25 ; personne.code_postal=1010 ;
personne.localite='Lausanne';
```

- Notation plus compacte :

```
personne(2)=struct('nom','Muller','prenom','Robert','age',28,'code_
postal',2000,'localite','Neuchatel')
```

- Ajout de nouveaux champs et sous-champs :

```
personne(2).enfants={'Arnaud','Camille'} ;
personne(1).tel.prive='021 123 45 67' ;
personne(1).tel.prof='021 987 65 43' ;
```

Structures

- Accès aux structures et aux champs :
  - structure(i)
  - structure([i j:k])
  - structure (i) . champs
  - structure (i) . champs . sous_champs
- On peut obtenir la liste des valeurs d'un champs
- Suppression d'un élément ou d'un champs avec []

Structures

Fonction	Description
rmfield(struct, champs)	Supprime un champ de la structure.
fieldnames(struct)	Retourne la liste des champs de la structure (ou du tableau de structures) struct.
getfield(struct,'champ')	Est identique à struct.champ, donc retourne le contenu du champ champ de la structure struct.
isstruct(var) isfield(struct,'champ')	Test si var est un objet de type structure (ou tableau de structures) : retourne 1 si c'est le cas, 0 sinon. Test si champ est un champ de la structure (ou du tableau de structures) struct : retourne 1 si c'est le cas, 0 sinon.
[n m]=size(tab_struct) length(tab_struct)	Retourne le nombre n de lignes et m de colonnes du tableau de structures tab_struct, respectivement le nombre total de structures.
for k=1:length(ts) % accès à ts(k).champ end	On boucle ainsi sur tous les éléments du tableau de structures tab_struct pour accéder aux valeurs correspondant au champ spécifié.

Tableaux cellulaires

- Se distingue du tableau standard en ce sens qu'il peut se composer d'objets de types différents (scalaire, vecteur, chaîne, matrice, structure...)
- Les tableaux cellulaires peuvent être multidimensionnels
- Pour définir un tableau cellulaire et accéder à ses éléments, on recourt aux accolades { }

Construction du tableau cellulaire

- Exemple à 2 dimensions :
  - tab(i , j) = { valeur }

  ou
  - tab{ i , j } = valeur
- On peut définir le tableau en une seule opération :
  - tab = { val1 , val2 ; val3 , val4 }

Interrogation du tableau cellulaire

- Toujours à 2 dimensions :
  - tab(i , j) : retourne le container de la cellule i,j
  - tab(i , :) : retourne un nouveau tableau cellulaire composé de la ligne i
  - tab(: , j) : idem avec la colonne j
  - tab{ i , j } : retourne le contenu de la cellule i,j
- Les tableaux cellulaires peuvent être utilisés comme paramètres d'entrée et de sortie à toutes les fonctions Octave

Tableaux cellulaires

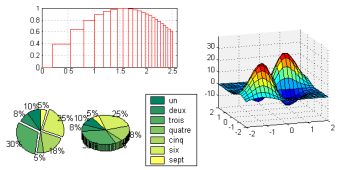
Fonction	Description
cell(n) cell(n,m) cell(n,m,o,p...)	Crée un objet de type tableau cellulaire carré de dimension nxn, respectivement de n lignes x m colonnes. Avec plus que 2 paramètres, crée un tableau cellulaire multidimensionnel.
iscell(var) iscellstr(var)	Teste si var est un objet de type tableau cellulaire. Teste si var est un tableau cellulaire de chaînes.
[n m]=size(tab_cel)	Retourne la taille du tableau cellulaire tab_cel.
tc_string = cellstr(ms)	Conversion de la "matrice de chaînes en un tableau cellulaire de chaînes.
ms = char(tc_string)	Conversion du tableau cellulaire de chaînes en une matrice de chaînes.
celldisp(tab_cel)	Affiche récursivement le contenu du tableau cellulaire.
num2cell	Conversion d'un tableau numérique en tableau cellulaire.
struct2cell, cell2struct	Conversion d'un tableau de structures en tableau cellulaire, et vice-versa.

Entrées – sorties (I/O)

- Lecture et écriture dans des flux (*streams*)
- Les trois flux standards :
  - stdin : entrée standard
  - stdout : sortie standard
  - stderr : erreur
- On peut créer un flux sur un fichier pour pouvoir le lire ou y écrire.

Fonctions d'I/O

Fonction	Description
[fid,msg] = fopen(name,mode,arch)	Ouvre le fichier *name* et lui associe le flux *fid*. En cas de problème, un message msg est retourné. *mode* prend ses valeurs parmis : 'r', 'w', 'a', 'r+', 'w+', 'a+'. *arch* prend ses valeurs parmis : 'native', 'ieee-le', 'ieee-be', etc.
fclose(fid)	Ferme le flux (et donc le fichier) *fid*.
fputs(fid,s)	Ecrit la chaîne *s* dans le flux *fid*.
fgetl(fid {,len}) fgets(fid, {,len})	Lit le flux *fid* jusqu'à la fin de la ligne, du fichier ou *len* caractères.
feof(fid)	Retourne 1 si la fin du flux (du fichier) a été atteint.
fdisp(fid,x)	Ecrit la valeur de la variable x dans le flux *fid*.
freport()	Affiche la liste des fichiers ouverts.
fseek(fid, offset {, origin})	Place le pointeur à un endroit précis dans le flux. On part de l'*origin* (SEEK_SET, SEEK_CUR, SEEK_END) et on parcourt *offset* caractères.



Graphiques 2D sous Octave

Julien Ah-Pine
julien.ah-pine@eric.univ-lyon2.fr

Licence IDEA 2e année - Université Lumière Lyon 2

Back-ends sous Octave

- Interface possible avec qoctave
- Utilisation de logiciels externes :
  - Gnuplot
  - JHandles
  - OctPlot
  - Octaviz
  - Yapso

  etc.

GNU Octave Forge 3.0

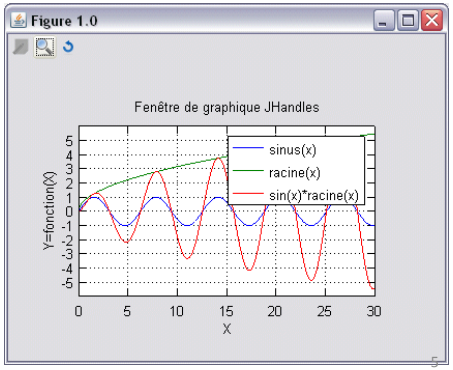
- **Gnuplot :** indépendant d'Octave, essentiellement orienté courbes 2D et surfaces 3D; capable maintenant de remplir des surfaces colorées.
- **JHandles :** conçu spécifiquement pour Octave et écrit en Java, ce nouveau back-end vise à implémenter de façon étendue les types de graphiques 2D/3D MATLAB.

Premier exemple

```
x=0:0.1:10*pi;
y1=sin(x); y2=sqrt(x); y3=sin(x).*sqrt(x);
plot(x,y1,x,y2,x,y3);
plot(x,y1,'b;sinus(x);',x,y2,'g;racine(x);',x,y3,'r;sin(x)*racine(x);');
grid('on');
axis([0 30 -6 6]);
set(gca,'Xtick',0:5:30);  set(gca,'Ytick',-5:1:5);
title('Fenêtre de graphique MATLAB / JHandles / Gnuplot');
xlabel('X'); ylabel('Y=fonction(X)');
legend('sinus(x)','racine(x)','sin(x)*racine(x)');
```

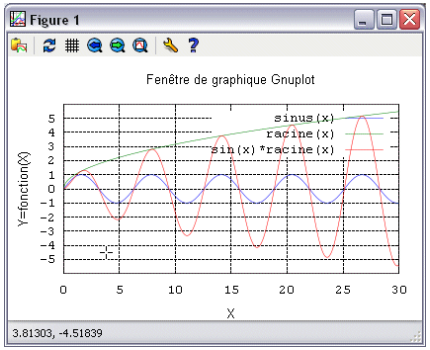
Jhandles 0.3.3

- Technologie très jeune et encore instable
- Barre d'outils :
  - loupe
  - rotation
- Sauvegarde par l'intermédiaire de la commande print



Gnuplot 4.2

- Affichage des coordonnées
- Barre d'outils :
  - copier/coller
  - rafraîchir
  - grille on/off
  - autoscaling

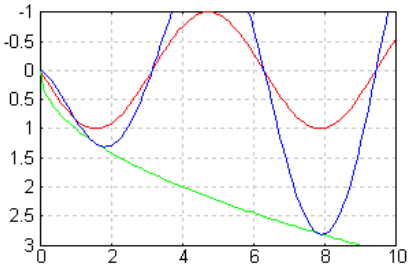


Détermination des axes

Commande	Description
axis([Xmin Xmax Ymin Ymax { Zmin Zmax }])	Rrecadre le graphique en utilisant les valeurs spécifiées des limites inférieures/supérieures des axes X, Y {et Z}
axis('auto')	Se remet en mode « autoscaling ».
axis('manual')	Verrouille les limites d'axes courantes.
lim_xyz = axis	Retourne le vecteur-ligne lim_xyz contenant les limites [Xmin Xmax Ymin Ymax { Zmin Zmax }].
xlim, ylim, zlim	Idem, mais ne fonctionne que sur un axe à la fois.
axis('equal'), axis('image'), axis('tight')	Définit le même rapport d'échelle pour les axes X et Y.
axis('square')	Définit les rapports d'échelle de façon la zone soit carrée
axis('normal')	Annule l'effet des "aspect ratio" equal ou square.
axis('off | on')	Désactive/rétablit l'affichage du cadre/axes/graduation et quadrillage du graphique.
axis('ij') et axis('xy')	Inverse l'ordre de l'axe Y.

Exemple

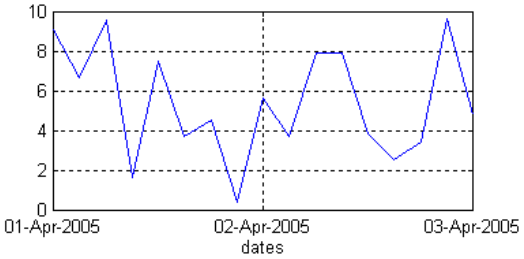
```
x=0:0.1:10*pi;
y1=sin(x); y2=sqrt(x); y3=sin(x).*sqrt(x);
plot(x,y1,x,y2,x,y3);
legend('off');
grid('on');
axis([0 10 -1 3]);
axis('ij');
```

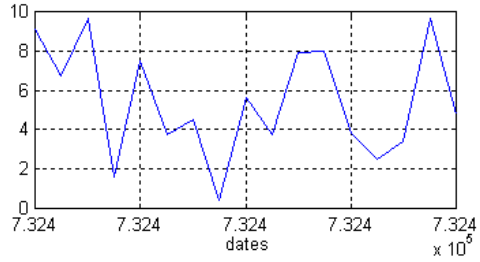


Graduations et grille

Commandes	Description
set(gca,'Xtick | Ytick | Ztick', [valeurs])	Spécifie les valeurs (suite de valeurs en ordre coissant), sur l'axe indiqué (gca = get current axes).
set(gca,'XTickLabel | YTickLabel | ZTickLabel', labels)	Spécifie le texte à afficher (label) en regard de chaque tick.
grid('on | off') grid	Activation/désactivation de l'affichage du quadrillage (*grid*).
box('on|off')	Activation/désactivation de l'affichage, autour du graphique, d'un cadre (graphiques 2D) ou d'une "boîte" (graphiques 3D).
xlabel('label_x'), ylabel('label_y'), zlabel('label_z')	Définit et affiche le texte de légende des axes X, Y et Z (étiquettes, labels).

Exercice en TD



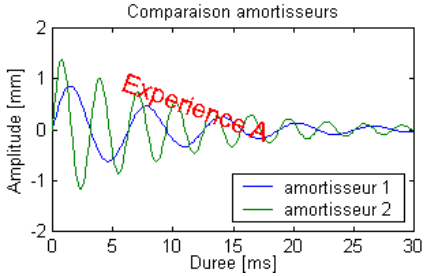


Légende, titre et textes

Commande	Description
legend('legende_t1','legende_t2'... {,pos})	Définit et place une légende sur le graphique en utilisant les textes spécifiés pour les tracés t1, t2... La position de la légende est définie par le paramètre pos.
legend('off')	Désactive l'affichage de la légende.
title('titre')	Définit un titre de graphique qui est placé au-dessus de la zone.
text(x, y, { z,} 'chaîne' {,'propriété','valeur'...})	Définit l'annotation chaîne qui est placés sur le graphique aux coordonnées x, y {,z} spécifiées. Des attributs (police, taille, couleur, orientation...) peuvent être spécifiés par des couples propriété/valeur.
gtext('chaîne')	L'emplacement du texte dans le graphique est défini interactivement à l'aide de la souris.

Exemple

```
x=linspace(0,30,200);
y1=sin(x)./exp(x/10); y2=1.5*sin(2*x)./exp(x/10);
plot(x,y1,x,y2);
xlabel('Duree [ms]');
ylabel('Amplitude [mm]');
title('Comparaison amortisseurs');
legend('amortisseur 1','amortisseur 2',4);
text(6,1,'Experience A', 'FontSize',14,'Rotation',-20,'Color','red');
```



Graphiques multiples

- Même fenêtre active par défaut
- Trois possibilités :
  - superposer les tracés
  - tracer les graphiques côte-à-côte
  - utiliser des fenêtres distinctes

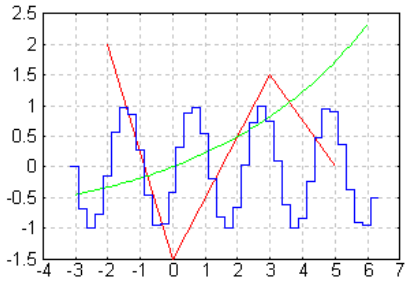
Superposition

- Le même système d'axes est utilisé et les courbes sont tracées avec des couleurs différentes.

Commande	Description
hold('on')	Accumuler (superposer) les ordres de dessin qui suivent dans la même figure.
hold('off')	Mode par défaut : on écrase l'ancien tracé.
ishold	Retourne l'état courant du mode hold pour la figure active.
line(x, y {,z })	Permet de dessiner par accumulation dans un graphique sans que hold soit à on !

Exemple

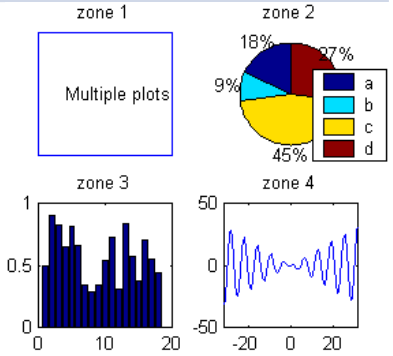
```
x1=[-2 0 3 5]; y1=[2 -1.5 1.5 0];
plot(x1,y1);
hold('on');
fplot('exp(x/5)-1',[-3 6]);
x3=-pi:0.25:2*pi;
y3=sin(3*x3);
stairs(x3,y3);
grid('on');
```



Graphiques côte-à-côte

Commande	Description
subplot(L,C,i)	Découpe la fenêtre graphique courante en L lignes et C colonnes, c'est-à-dire en L x C espaces qui disposeront chacun leur propre système d'axes (mini graphiques).

```
subplot(2,2,1);
plot([0 1 1 0 0],[0 0 1 1 0]);
text(0.2,0.5,'Multiple plots');
axis('off'); legend('off'); title('zone 1');
subplot(2,2,2);
pie([2 1 5 3]); legend('a','b','c','d');
title('zone 2');
subplot(2,2,3);
bar(rand(18,1)); title('zone 3');
subplot(2,2,4);
fplot('x*cos(x)',[-10*pi 10*pi]);
title('zone 4');
```



Graphiques dans des fenêtres distinctes, figures

Commande	Description
figure	Ouvre une nouvelle fenêtre de graphique (figure), et en fait la fenêtre de tracé active.
figure(n)	La figure numéro n devient la fenêtre de tracé active.
gcf	Retourne le numero de la fenêtre de graphique active.
shg	Fait passer la fenêtre de figure courante au premier plan.
clf	Efface le(s) graphique(s) dans la fenêtre de figure courante.
close	Referme la fenêtre graphique active (figure courante).
close (n)	Referme la fenêtre graphique de numero spécifié.
close all	Referme toutes les fenêtre graphique.

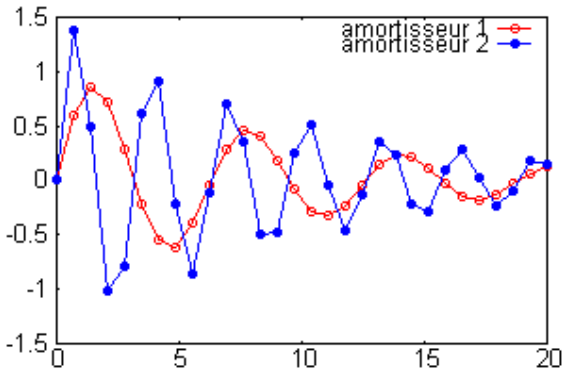
Traits, symboles, couleurs

- Utilisation du paramètre linespec :

Code	Effet	Code	Effet
y	couleur jaune	o	symbole o
r	couleur rouge	*	symbole *
g	couleur verte	s	carré vide
b	couleur bleue	x	croix oblique
w	couleur blanche	.	symbole point
k	couleur noire	d	losange vide
m	couleur magenta	p	étoile à 5 branches
--	trait continu	h	étoile à 6 branches
:	trait pointillé		

Exemple

```
x=linspace(0,20,30);
y1=sin(x)./exp(x/10); y2=1.5*sin(2*x)./exp(x/10);
plot(x,y1,'r-o',x,y2,'b:.');
legend('amortisseur 1','amortisseur 2');
```

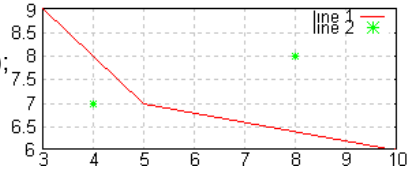


Fonctions graphiques

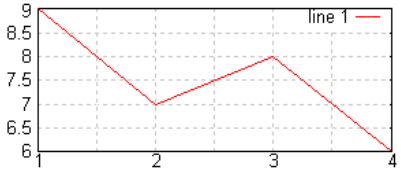
Fonction	Description
plot(x1, y1 {,linespec} {, x2, y2 {,linespec} ...})	xi et yi sont des vecteurs (ligne ou colonne), et le graphique comportera autant de courbes indépendantes que de paires xi/yi.
plot(vect)	Idem mais avec un vecteur X uniforme [1 2… length(*vect*)].
plot(mat)	Idem mais avec autant de courbes que de colonnes dans *mat*.
plot(var1,var2 ...)	Cela dépend de la nature de var1 et var2 (vecteur, matrice).
fplot('fonction', [xmin xmax] {, nb_points })	Graphique 2D d'une fonction y=f(x) : trace la fonction f(x) spécifiée entre les limites xmin et xmax.
semilogx(...)	Axe X logarithmique, axe Y linéaire.
semilogy(...)	Axe X linéaire, axe Y logarithmique.
loglog(...)	Axes X et Y logarithmiques.

Exemples

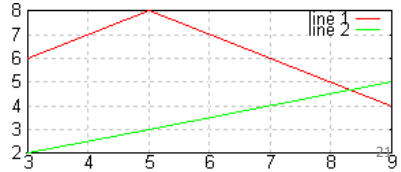
```
plot([3 5 6 10], [9 7 NaN 6], [4 8], [7 8], 'g*');
```




```
plot([9 ; 7 ; 8 ; 6]);
```

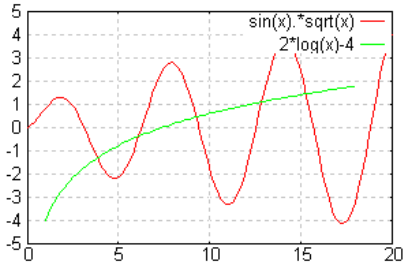



```
plot([3 5 9], [6 8 4 ; 2 3 5]);
```



Exemples

```
fplot('sin(x).*sqrt(x)',[0 20]);
hold('on');
fplot('2*log(x)-4',[1 18]);
grid('on');
```

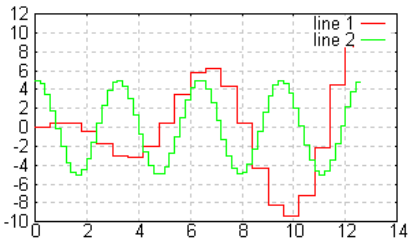


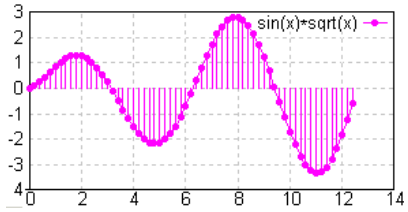

```
fplot('sin',[0 10]);
hold('on');
fplot('cos',[0 10]);
grid('on');
```

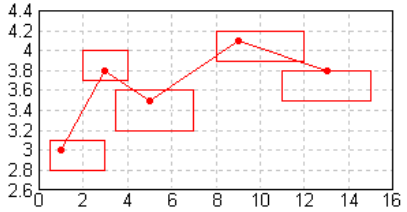
Fonctions graphiques

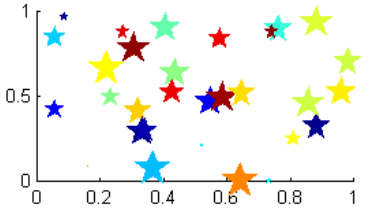
Fonctions	Description
plotyy(x1, y1, x2, y2 {,'type1' {,'type2'}})	Trace la courbe définie par les vecteurs x1 et y1 relativement à l'axe Y de gauche, et la courbe définie par les vecteurs x2 et y2 relativement à l'axe Y de droite.
stairs({x,} y)	Dessine une ligne **en escaliers** pour la courbe définie par les vecteurs (ligne ou colonne) x et y.
stem({x,} y {, linespec })	Graphe la courbe définie par les vecteurs (ligne ou colonne) x et y en affichant une ligne de **rappel verticale** (bâtonnet, pointe) sur tous les points de la courbe.
errorbar(x, y, error {,format})	Graphe la courbe et ajoute, à cheval sur cette courbe et en chaque point de celle-ci, des **barres d'erreur**.
errorbar(x, y, lower, upper {,format})	Idem mais avec des barres d'erreur asymétriques.
scatter(x, y {,size {,color } })	Dessin du **semis de points** défini par les vecteurs de coordonnées x et y.

Fonctions graphiques





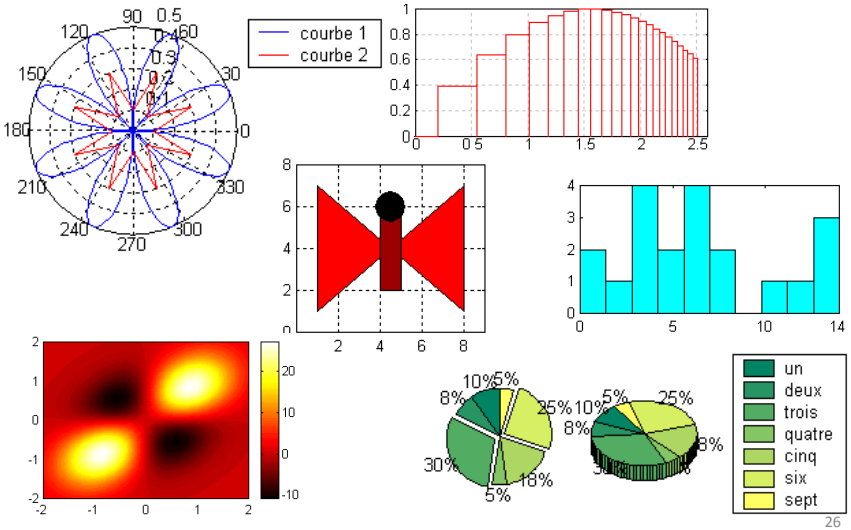




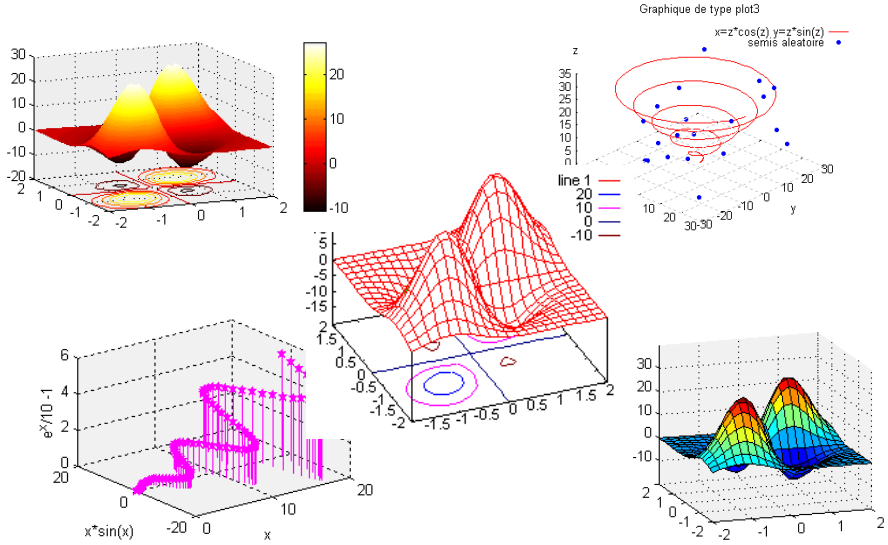
Fonctions graphiques (encore)

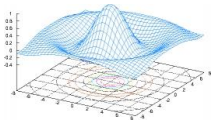
Fonction	Description
fill(x, y, couleur)	Dessin 2D de surfaces remplies.
pie(val {,['label1';'label2';...;'label n']})	Graphique en camembert.
bar({x,} y)	Graphique 2D en barres.
hist(y {,n}) hist(y, x)	Histogramme 2D de distribution de valeurs.
polar(angle, rayon {,linespec})	Graphique 2D de lignes et/ou semis de points en coordonnées polaires.
pcolor({X, Y,} Z)	Affichage en pseudo-couleurs de la matrice *Z*.

Fonctions graphiques (encore)



Graphiques 3D





Programmation sous Octave

Julien Ah-Pine
julien.ah-pine@eric.univ-lyon2.fr

Licence IDEA 2e année - Université Lumière Lyon 2

Généralités

- Les "M-files" sont des fichiers au format texte qui contiennent des instructions Octave et qui portent l'extension .m
- Utilisation d'un éditeur de texte (xemacs, etc.)
- On distingue deux types de M-files :
  - les scripts
  - les fonctions
- Octave est un langage interprété

Interactions utilisateur

Fonction	Description
disp(var)	Affiche le contenu de la variable *var*.
printf('format', variable(s))	Affiche les variables de façon formatée. **Ex. :** printf('variable v= %6.1f et variable t= %s \n',v,t)
error({'id',} 'message' {,variable(s)...})	Affiche le message spécifié sous la forme "error: message", puis l'exécution du script ou de la fonction est interrompue.
{string=}lasterr	Affiche (ou récupère) le dernier message d'erreur.
beep	Effectue un beep sonore.
var = input('prompt' {, 's'})	Affiche le *prompt* spécifié, puis attend que l'utilisateur entre quelque-chose au clavier terminé par la touche <Enter>.
ch=menu('Titre','bouton1'…)	Affiche un menu de choix entre plusieurs options.
pause pause(secondes)	Effectue une pause, c'est-à-dire attend que l'utilisateur frappe n'importe quelle touche au clavier pour continuer son exécution.

Structures de contrôle

- Boucle « Pour » :

```
Pour i=1 à N Faire
    instructions
FinPour
```

⇔

```
for indice=matrice ou vecteur
  commandes...
end
```

- Boucle « Tant que » :

```
Tantque (test)Faire
    instructions
FinTantque
```

⇔

```
while expression_logique
  commandes...
end
```


Structures de contrôle

- Test « Si… Alors » :

```
Si (test) Alors
    instructions
Sinon
    instructions
FinSi
```

⇔

```
if exp_logique_1
  commandes_1
{ elseif exp_logique_2
    commandes_2 }
{ else
    autres_commandes }
end
```

- Exemples d'expression logique :

  (x<y) (y==2) ((x+1<z) | (x>=y)) etc.

Structures de contrôle

- Test « Cas parmi » :

```
Cas i parmi
    cas 1:
        instructions
    cas 2:
        instructions
    défaut:
        instructions
FinCasparmi
```

```
switch variable
    case { val1, val2… },
        commandes_1
    case { val3, val3… },
        commandes_2
    otherwise
        autres_commandes
end
```


Commandes utiles

Commande	Description
return	Termine l'exécution de la fonction ou du script.
error({'id',} 'message' {,variable(s)...})	Affiche le message spécifié sous la forme "error: message", puis l'exécution du script ou de la fonction est interrompue.
beep	Effectue un beep sonore.
{var=} nargin	A l'intérieur d'une fonction, retourne le nombre d'arguments d'entrée passés lors de l'appel à cette fonction.
{var=} nargout	A l'intérieur d'une fonction, retourne le nombre de variables de sortie auxquelles la fonction est affectée lors de l'appel.
{string=} inputname(k)	A l'intérieur d'une fonction, retourne le nom de variable du k-ème argument passé à la fonction.
{string=} mfilename	A l'intérieur d'une fonction ou d'un script, retourne le nom du M-file de cette fonction ou script, sans son extension .m

Commandes utiles

Commande	Description
global var(s)	Définit la(les) variable(s) spécifiée(s) comme globale(s).
persistent var(s)	Utilisable dans les fonctions seulement, cette déclaration définit la(les) variable(s) spécifiée(s) comme statique(s).
eval('expression1', {'expression2'})	Évalue et exécute l'expression1 Octave spécifiée. En cas d'échec, évalue l'expression2.
class(objet)	Retourne la "classe" de objet (double, struct, cell, char).
source('M-file.m')	Exécute le M-file spécifié.
echo on	Activer le mode debugging (M-files).
echo off	Désactiver le mode debugging (M-files).

Scripts

- Un **script** (ou programme) n'est rien d'autre qu'une suite de commandes Octave valides.
- Invoqués sans arguments car ils opèrent sur les variables de l'environnement (*workspace*).
- Pour documenter un script : %
- Pour exécuter un script :
  - directement : nom du m-file, puis <entrée>
  - via la commande source(''nomDuFichier.m'')

Exemple de script

```
%SOMPROD Script réalisant la somme et le produit de 2 nombres, vecteurs ou matrices
% Ce script est interactif, c'est-à-dire qu'il demande interactivement les 2 nombres, vecteurs
ou matrices dont il faut faire la somme et le produit (élément par élément)

V1=input('Entrer 1er nombre (ou expression, vecteur ou matrice) : ') ;
V2=input('Entrer 2e  nombre (ou expression, vecteur ou matrice) : ') ;

if ~ isequal(size(V1),size(V2))
      error("les 2 arguments n'ont pas la meme dimension")
end

disp('Somme =') , disp(V1+V2)
disp('Produit = ') , disp(V1.*V2)

return  % Sortie du script (facultatif)
```


Fonctions

- Se distinguent des scripts :
  - appel : var_sortie = nom_fonction(arg_entree,...)
  - utilise les arguments d'entrées et des variables locales (et non celles de l'environnement)
- Passage des paramètres par **valeurs**
- Déclaration de la fonction fct(…) dans un fichier portant le même nom : *fct.m*

```
function [arg_sortie, ...] = nom_fonction(arg_entree, ...)
```


Exemples

- La fonction *fct1.m* ci-dessous mémorise le nombre de fois qu'elle a été appelée :

```
function []=fct1()
      global COMPTEUR
      COMPTEUR=COMPTEUR+1;
      printf('fonction appelee %04u fois
      \n',COMPTEUR)
return
```

- Pour l'appeler :

```
global COMPTEUR   % cela déclare le compteur également global dans le
workspace
COMPTEUR = 0 ;    % initialisation du compteur
fct1          % => cela affiche "fonction appelee   1 fois"
fct1          % => cela affiche "fonction appelee   2 fois"
```


Exemples

- La fonction *fct2.m* ci-dessous mémorise le nombre de fois qu'elle a été appelée :

```
function []=fct2()
      persistent compteur
      if isempty(compteur)
            compteur=0 ;
      end
      compteur=compteur+1 ;
      printf('fonction appelee %04u fois \n',compteur)
return
```

- Pour l'appeler :

```
fct2          % => cela affiche "fonction appelee   1 fois"
fct2          % => cela affiche "fonction appelee   2 fois"
```

Exemples

```
function [somme,produit] = fsomprod(a,b)
%FSOMPROD somme et produit de 2 nombres, vecteurs ou matrices
%   Usage: [S,P]=FSOMPROD(V1,V2)
%      Retourne matrice S contenant la somme de V1 et V2,
%      et matrice P contenant le produit de V1 et V2
%      élément par élément

if nargin~=2
      error("cette fonction attend 2 arguments")
end
if ~ isequal(size(a),size(b))
      error("les 2 arg. n'ont pas la même dimension")
end

somme=a+b;
produit=a.*b;  % produit élément par élément !
return         % sortie de la fonction
```