DÉPARTEMENT D'INFORMATIQUE

UNIVERSITÉ NICE SOPHIA ANTIPOLIS

Introduction à la programmation Matlab/Octave

NOTES DE COURS

Version: 3 novembre 2015

Arnaud Malapert

(firstname.lastname@unice)

Table des matières

Lecture 1
Concepts et outils logiciels

Notes du cours *Introduction à la programmation Matlab/Octave*
Présenté le 3 novembre 2015 par Arnaud Malapert
du Département d'informatique de l'Université Nice Sophia Antipolis

1 Calcul scientifique

1.1 Calcul formel

Calcul formel

Principe

- Procédé de transformation d'expressions mathématiques.
- Les objets de ce calcul ne sont plus les variables de l'expression mathématique mais les opérations elles-mêmes.
- populaire au début des années 1970 et a participé à la création de l'IA.

Exemples

- Résoudre un problème en fonction de un ou plusieurs inconnue(s),
- Dériver une fonction, déterminer sa primitive, simplifier son expression.

1.1

Types d'expression

- des polynômes avec de multiples variables ;
- fonctions standards, spéciales ou composées de diverses expressions ;
- dérivées, intégrales, sommes et produits d'expressions ;
- matrices d'expressions.

1.2

Manipulations symboliques

- simplification, qu'elle soit automatique ou effectuée à partir d'hypothèses ;
- substitution de symboles ou de valeurs numériques par des expressions ;
- changement de forme des expressions : expansion de produits et de puissances, réécriture de fractions partielles, réécriture de fonctions trigonométriques comme exponentielles, etc. ;
- différentiation relative à une ou plusieurs variables ;
- optimisation globale, qu'elle soit conditionnelle ou non ;
- factorisation partielle ou complète ;
- solution d'équations linéaires et de quelques équations non-linéaires ;
- solution de quelques équations différentielles ;
- calcul de limites de certaines fonctions ;
- quelques intégrales définies ;
- transformées (Laplace, Fourier, etc.) ;
- expansion et somme de séries ;
- opérations sur les matrices incluant le produit et l'inversion, etc. ;
- affichage d'expressions mathématiques, (par ex. avec TEX).

1.3

Identités remarquables de degré 2

$$(a+b)^2 = a^2 + 2ab + b^2$$
$$(a-b)^2 = a^2 - 2ab + b^2$$
$$(a-b) \times (a+b) = a^2 - b^2$$

Résolution de $x^2 - x - 1 = 0$

$$\begin{aligned} x^2 - x - 1 &= \left(x - \frac{1}{2}\right)^2 - \frac{5}{4} \\ &= \left(x - \frac{1+\sqrt{5}}{2}\right) \times \left(x - \frac{1-\sqrt{5}}{2}\right) \end{aligned}$$

1.4

Résolution d'une équation du second degré

une *équation du second degré* peut s'écrire sous la forme

$$ax^2 + bx + c = 0$$

où x est l'inconnue et les lettres a, b et c représentent les coefficients réels, avec a différent de 0. Le *discriminant* de l'équation est la valeur Δ définie par :

$$\Delta = b^2 - 4ac.$$

— Si le discriminant est strictement positif, l'équation admet deux racines distinctes :

$$x_1 = \frac{-b - \sqrt{\Delta}}{2a} \quad \text{et} \quad x_2 = \frac{-b + \sqrt{\Delta}}{2a}$$

— Si le discriminant est nul, l'équation admet une racine double :

$$x_1 = x_2 = -\frac{b}{2a}$$

— Si le discriminant est négatif, l'équation n'admet aucune racine réelle.

1.5

Interprétation graphique

1.6

À vous de jouer !

Exercice 1. Est-il possible de résoudre l'équation suivante grâce à la méthode du déterminant ?

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

Indices : identités remarquables de degré 3

$$(a-b)^3 = a^3 - 3a^2b + 3ab^2 - b^3$$
$$(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$$

Solution.
— Oui, le développement donne une équation du second degré : $x^2 + x + 1 = 0$.
— Par contre, elle n'admet pas de racine réelle ($\Delta = -3$).

1.7

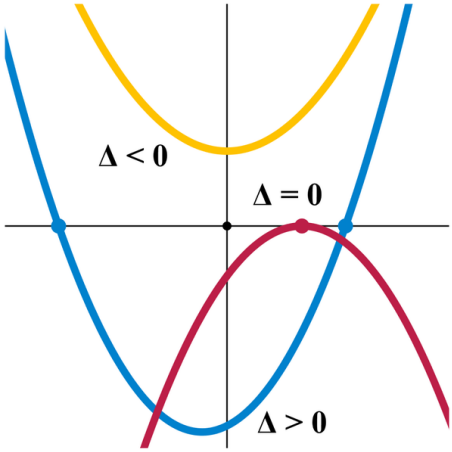


FIGURE 1 – Le signe du discriminant apporte une information sur le graphe de la fonction.

1.2 Calcul numérique

Calcul numérique

Principe

— Résoudre par des calculs purement numériques des problèmes d'analyse mathématique.
— Utilisation de résultats empiriques de calculs numériques (axiomes mathématiques, théorèmes, preuves).

Applications

— Conception de structures (systèmes aéronautiques ou automobiles).
— Systèmes complexes et chaotiques (prévision numérique du temps).
— Modélisation 3D (effets spéciaux).
— Statistiques appliquées (démographie, modèles économiques)
— Analyse financière ou boursière.

1.8

Étude des erreurs

Erreur d'arrondi

Il est impossible de représenter en pratique tous les nombres réels exactement sur une machine à états finis.

Erreur de troncature

La solution approchée obtenue diffère de la solution exacte.

Discrétisation

La discrétisation d'un problème induit une erreur car la solution du problème discret ne coïncide pas forcement avec la solution du problème continu.

Propagation des erreurs

Une fois que l'erreur est générée, elle se propagera généralement tout au long du calcul.

1.9

Stabilité numérique

Stabilité numérique

un algorithme est numériquement stable si une erreur, une fois générée, ne croît pas trop durant le calcul.

— Il ne faut pas que l'algorithme diverge.

Conditionnement

— Problème bien conditionné : la solution ne change que d'une faible quantité si les données du problème sont changées d'une faible quantité.

— Problème mal conditionné : la moindre erreur dans les données peut provoquer une erreur très importante dans la solution trouvée.

— Cependant, un algorithme qui résout un problème bien conditionné peut être ou ne pas être numériquement stable.

— L'analyse numérique consiste à trouver un algorithme stable pour résoudre un problème mathématique bien posé.

1.10

Méthode de dichotomie

— Supposons une *fonction continue* $f : [a, b] \to \mathbb{R}$ telle que $f(a)$ et $f(b)$ soient de signes opposés.

— D'après le théorème des valeurs intermédiaires, f doit avoir au moins un zéro dans l'intervalle $[a, b]$.

— La méthode de dichotomie consiste à diviser l'intervalle en deux en calculant $c = (a + b) \div 2$.

— Il y a maintenant deux possibilités : ou f(a) et f(c) sont de signes contraires, ou f(c) et f(b) sont de signes contraires.

— L'algorithme est alors appliqué au sous-intervalle dans lequel le changement de signe se produit (*récursive*).

— L'erreur absolue est au plus $\frac{b-a}{2^{n+1}}$ après n étapes.

— La méthode converge linéairement, ce qui est très lent par comparaison avec la *méthode de Newton* (fonction dérivables).

— Le nombre maximum d'itérations nécessaires pour satisfaire une tolérance relative ϵ est : $2^{n+1} = 1/\epsilon$

1.11

Résolution de $x^2 - x - 1 = 0$ par dichotomie

Réductions de $[a, b]$

1. $[0, 4]$
2. $[0, 2]$
3. $[1, 2]$
4. $[1.5, 2]$
5. $[1.5, 1.75]$

1.12

Recherche dichotomique dans un tableau

Recherche l'indice d'un élément dans un tableau

— L'élément peut ne pas être présent dans le tableau.

Recherche séquentielle $O(n)$**.**

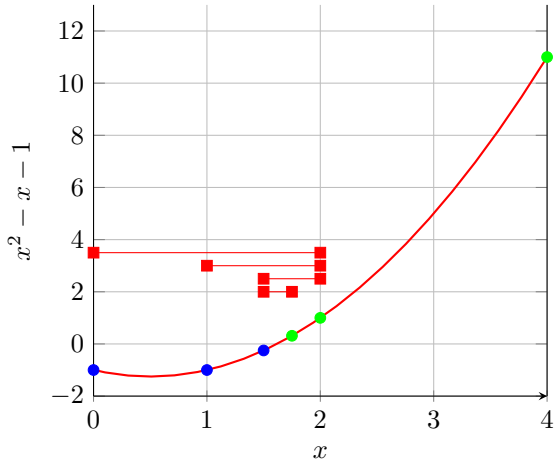
On parcourt les éléments du tableau jusqu'à ce que :

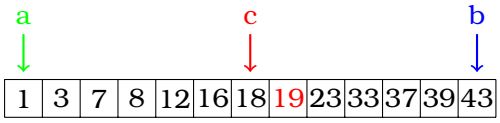
1. l'élément courant soit l'élément recherché ;
2. on ait atteint la fin du tableau

Recherche dichotomique $O(\log n)$

— On suppose maintenant que le tableau est trié (ordre partiel).

1.13





Exemple : calcul de puissances entières x^n

Méthode simple : n opérations

Multiplier x par lui-même n fois.

Propriétés algébriques sur les puissances

$$a^m \times a^n = a^{m+n} \qquad (a^m)^n = a^{m \times n}$$

Exponentiation rapide : $\log_2(n)$ opérations

$$\texttt{puissance}(x,n) = \begin{cases} \text{x} & \text{si n=1} \\ \texttt{puissance}(x^2, \frac{n}{2}) & \text{si n est paire} \\ x \times \texttt{puissance}(x^2, \frac{n-1}{2}) & \text{si n est impaire} \end{cases}$$

Calcul de 7^9

$7^9 = 7 \times (7^2)^4 = 7 \times (49^2)^2 = 7 \times (2401)^2 = 7 \times 5764801 = 40353607$

1.14

2 Outils logiciels

Fortran (FORmula TRANslator)

- Langage de programmation utilisé principalement en calcul scientifique.
- John Backus, pionnier de l'informatique, publie les spécifications en 1954.
- IBM proposa le premier compilateur FORTRAN en 1956 (25 000 lignes, pour l'IBM 704).
- Très nombreuses bibliothèques de fonctions utilisables en Fortran.
- Des compilateurs performants produisent des exécutables très rapides.
- Beaucoup de programmes scientifiques sont à présent écrits en C et C++

1.15

2.1 Calcul formel

Calcul formel

Bibliothèques

— Le dépôt Netlib (Fortran et C).
— Logiciels commerciaux : IMSL, NAG et HSL ;
— une alternative libre est la GSL.

Applications
— Mathematica, Maple, Maxima, SAGE

Utilisation pour le calcul numérique
— Leur puissance est généralement liée au calcul symbolique,
— Transformer un problème numérique complexe en suite finie d'éléments de calcul numérique simples,
— évalués alors individuellement par des recettes numériques.

1.16

2.2 Calcul numérique

Calcul numérique

Langage de haut niveau
— Orienté autour du problème à résoudre.
— Utiliser les langues naturelles et des symboles mathématiques.
— Abstraction des caractéristiques techniques du matériel.

Interpréteur de commande
Exécution *à la volée* d'instructions via une fenêtre de commande ou l'exécution de fichiers-scripts

Compilation
Transformation d'un langage source vers un langage cible.
— Traduction vers un langage de bas niveau (assembleur, machine).
— Semi compilation : traduction dans un langage intermédiaire, sous forme binaire (code objet ou bytecode), avant d'être lui-même interprété ou compilé.

1.17

Matlab
— Créé à la fin des années 1970 autour des librairies Fortran EISPACK et LINPACK et réécrit en 2000 autour de la librairie LAPACK.
— Logiciel commercial (payant) développé par The MathWork. Licence propriétaire.
— Environnement : Windows, Linux, MacOS
— Site Web : `www.mathworks.fr`

1.18

Octave
— Créé à la fin des années 1980 pour faciliter l'utilisation de routines Fortran.
— Compatibilité du langage avec Matlab
— Environnement : Linux, MacOS, Sun Solaris, Windows
— Logiciel libre (gratuit). Licence GNU General Public License (GPL).
— Site Web : `www.octave.org`

1.19

Scilab
— Créé au début des années 2000 par l'INRIA.
— Environnement : Linux, MacOS, Sun Solaris, Windows
— Logiciel libre (gratuit). Licence CeCILL (CEA-CNRS-INRIA logiciel libre).
— Site Web : `www.scilab-enterprises.com`

1.20

Interface avec d'autres langages

— Appel de programme externe.
— Interface avec des modules compilés Fortran et C/C++
— Création de nouvelles routines avec des modules compilés

1.21

3 Comparatif Matlab-Octave-Scilab

Matlab

Les ☺ de Matlab :
— rapidité d'exécution (compilation possible du code) ;
— diversité des librairies ;
— programmation orientée objet
— IDE intégré et performant ;
— répandu dans le monde industriel.

Les ☹ de Matlab
— logiciel payant ;
— logiciel gourmand en ressources (disque et RAM)

1.22

Octave

Les ☺ d'Octave :
— compatibilité avec Matlab ;
— diversité des librairies ;
— communauté *open source* nombreuse et active ;
— logiciel gratuit ;

Les ☹ d'Octave
— pas de programmation orientée objet ;
— développement de GUI difficile ;
— lenteur relative d'exécution ;
— peu répandu dans le monde industriel.

1.23

Scilab

Les ☺ de Scilab :
— diversité des librairies ;
— IDE intégré ;
— Développement de GUI basique ;
— logiciel gratuit ;

Les ☹ de Scilab
— incompatibilité avec Matlab ;
— pas de programmation orientée objet ;
— lenteur relative d'exécution ;

1.24

4 Limites de Matlab-Octave-Scilab

Performance : calcul de π

Propriété
La série $\sum_0^{+\infty} \frac{(-1)^n}{2n+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \ldots$ converge vers $\frac{\pi}{4}$

Résultat (ordinateur Pentium à 700 MHz)
Pour une la somme des 1000000 premiers termes égale à 3.141594 :

Scilab 5 minutes (3 secondes 10000 répétitions).
— somme des 10000 premiers termes égale à 3.14169.

— La précision est moins bonne d'un facteur 100.

TurboPascal 3 secondes.

Matlab 12 secondes.

Octave 112 secondes.

1.25

Précision : calcul de e

Propriété

La série $\sum_0^{+\infty} \frac{1}{n!} = 1 + 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} \ldots$ converge vers e

Une boucle infinie ?

```
oldE=0;
estE=2;
ordre=1;
diviseur=1;
while(oldE != estE)
  ordre+=1;
  diviseur*=ordre;
  oldE=estE;
  estE+= 1/diviseur;
endwhile
```

Résultats

1. L'algorithme converge vers la valeur 2.71828182845905 après 17 itérations.
2. L'erreur relative est $2.2e-16$.
3. Le 18-ème terme est $\frac{1}{18!} = 1/6402373705728000 = 1.56e-16$
4. La précision machine est $4.44e-16$.

1.26

5 Installation de GNU Octave

Installation sur GNU/Linux

— La dernière version d'Octave est toujours disponible sur `ftp://ftp.gnu.org/gnu/octave`
  — Il est alors nécessaire de compiler les sources.
— Les versions d'Octave sont fournies par les distributions.
— Les distributions suivantes fournissent un paquet logiciel :
  — Debian/Ubuntu : `apt-get install octave octave-doc`
  — Fedora : `yum install octave octave-doc`
  — Gentoo et SuSE.
— Paquets créés par des volontaires indépendamment du projet Octave.
— Aucun délai garanti quant à la disponibilité d'une version d'Octave.
— Environnement de développement : éditeur de texte (GNU Emacs, vi, gedit, etc) et terminal ; un ou deux IDE (QtOctave))

1.27

Installation sur Mac OS X et Windows

Windows avec l'émulateur cygwin.

Mac OS X avec un gestionnaire de paquet.
— Fink, MacPorts, and Homebrew.

Compiler les sources ;)

Une méthode simple et efficace

Utiliser une machine virtuelle Linux !

1.28

Virtualisation

La virtualisation consiste à faire fonctionner un ou plusieurs systèmes d'exploitation / applications (comme un browser), sur un ou plusieurs ordinateurs au lieu d'en installer un seul par machine. On appelle machines virtuelles (virtual machines) ces ordinateurs virtuels.

Notions

Chaque outil de virtualisation met en œuvre une ou plusieurs de ces notions :
- couche d'abstraction matérielle et/ou logicielle
- système d'exploitation hôte (installé directement sur le matériel)
- systèmes d'exploitation « virtualisé(s) » ou « invité(s) »
- partitionnement, isolation et/ou partage des ressources physiques et/ou logicielles
- images manipulables : démarrage, arrêt, sauvegarde et restauration, gel, clonage,, sauvegarde de contexte, migration d'une machine physique à une autre
- réseau virtuel : réseau purement logiciel, interne à la machine hôte, entre hôtes et/ou invités

1.29

6 Premiers pas avec GNU Octave

Lancement et arrêt

- Pour *démarrer Octave*, taper la commande shell `octave` ou double-cliquer sur l'icône « Octave ».
- Après un message informatif, l'invite de commandes (*prompt*) s'affiche : `octave:1>`
- On peut taper des *commandes* qui seront *interprétées*.
- En cas de problème, on peut *interrompre Octave* en tapant `ctrl-C`.
- Pour *arrêter Octave*, tapez `quit` ou `exit`.

1.30

Obtenir de l'aide

- *Obtenir de l'aide*, tapez `help` or `doc`
- *Obtenir de l'aide* sur une *commande spécifique* (=fonctions prédefinies), tapez `help command`
  - Exemples : `help length`, `help size`, `help plot` ...
- *Obtenir de l'aide* sur le système d'aide `help help`
- Tapez `q` pour sortir du mode help

Coup du sombrero

```
help sombrero
sombrero(50)
quit
```

1.31

Remarque sur le système d'aide

- Dans le texte de l'aide, les noms de fonction apparaissent en *lettres minuscules* et les noms de variable apparaissent en *lettres majuscules*.

```
octave:20>help size
'size' is a built-in function

 -- Built-in Function:  size (A)
 -- Built-in Function:  size (A, DIM)
     Return the number of rows and columns of A.
[...]
```

— Ne vous méprenez pas ! la *convention de nommage* (sensible à la casse) autorise le *mélange de lettres majuscules et minuscules*.
  — `maVariable` et `mavariable` sont valides, mais différents.
— Par contre, les *fonctions prédéfinies* sont composées uniquement de *lettres minuscules* 1.32

Convention de nommage

Variables mélange de lettres majuscules et minuscules débutant par une lettre minuscule.
  — `linearity`, `credibleThreat`, `qualityOfLife`

Constantes lettres majuscules et séparation des mots par des tirets bas _ (underscore).
  — `MAX_ITERATIONS`, `COLOR_RED`

Fonctions uniquement des lettres minuscules ;
  — `getname(.)`, `computetotalwidth(.)`
  avec séparation des mots par des tirets bas _ (underscore) ;
  — `get_name(.)`, `compute_total_width(.)`

— *Les noms doivent indiquer la signification ou l'utilisation.*
— On peut utiliser le français, mais l'anglais est plus répandu. 1.33

7 Variables et types de données

Types de données

Matrices réelles et complexes

Strings chaînes (matrices) de caractères

Structures

— Vecteurs ? une matrice avec une seule ligne/colonne
— Scalaires ? une matrice 1×1
— Entiers ? c'est un flottant (double précision)
— Booléens ? c'est un entier (non-nul = true,0=false)
— Objets ? Seulement en Matlab 1.34

Définition de variables

Tapez simplement :

```
octave:1> format long g
octave:2> a=104348
a = 104348
octave:3> b=33215;
octave:4> res=a/b
res = 3.14159265392142
octave:5> pi
ans = 3.14159265358979
octave:6> (res-pi)/pi
ans = 1.05560489507986e-10
octave:7> res/pi -1
ans = 1.0556044927057e-10
```

— Déclaration dynamique des variables
— Typage dynamique des variables
— Opérations algébriques
— **Erreur d'arrondi** : Les résultats des calculs équivalents en lignes 6 et 7 sont différents. 1.35

Création d'une matrice

Tapez simplement :

```
octave:1> A = [8 2 3 ; 3 -1 7 ; 1 5 3]
```

Octave répondra par l'affichage de la matrice : :

```
A =

  8 2 3
  3 -1 7
  1 5 3
```

Syntaxe

— Utiliser une *virgule ou* un *espace* pour délimiter les *colonnes*
— Utiliser un *point virgule* pour délimiter les *lignes*

Les expressions suivantes sont équivalentes :

```
A = [8 2 3;3 -1 7;1 5 3]
A = [8,2,3;3,-1,7;1,5,3]
```

1.36

Chaînes de caractères

Tapez simplement :

```
octave:1> str='coucou'
```

— Octave supporte les guillemets simples et doubles.
— Matlab ne supporte que les guillemets simples.
— Pour des raisons de compatibilité, **utilisez des guillemets simples.**

Commandes fréquemment utilisées

— `strcat` : concaténation de chaînes de caractères
— `int2str` : conversion d'un entier en chaîne de caractères
— `num2str` : conversion d'un nombre en chaîne de caractères
— `sprintf` : écrit des données formattées (e.g. C++)

```
printf(strcat("La␣fraction␣%d/%d␣est␣une␣approximation␣de",upper("␣pi"),"␣
    avec␣une␣erreur␣relative␣de␣%.2e"),a,b,res/pi -1)
La fraction 104348/33215 est une approximation de PI avec une erreur
    relative de 1.06e-10
```

1.37

Chaînes de caractères

Octave/Matlab possède virtuellement toutes les fonctions communes pour la manipulation des chaînes de caractères.

— `strcmp`, `strncmp`, `strmatch`, `char`, `ischar`, `findstr`, `strfind`,
— `str2double`, `str2num`, `num2str`, `strtrim`, `upper`, `lower`,
— et plus encore . . .

1.38

Structures

— Création d'une structure

```
octave:1> data.id=0 ; data.timestamp=time
data =

  scalar structure containing the fields:
    id = 0
    timestamp = 1347983702.21123
```

— Création d'un tableau de structures

```
octave:2> data(2).id=1 ; data(2).timestamp=time ; data
data =

  1x2 struct array containing the fields:
    id
    timestamp
```

Exercice 2. Quelles sont les différences entre une structure et un objet ?

1.39

Contrôle de l'affichage

Afficher une variable

Tapez simplement son nom :

```
octave:2> a
a = 1
```

Supprimer l'affichage

Ajoutez un point-virgule :

```
octave:2> a;
octave:3> sin(0);
```

Marche aussi pour les appels de fonction

1.40

Remarques

— Les *variables* ont un *typage dynamique*. `s=3` suivi de `s='octave'` est accepté.

— Utiliser la commande *who* (ou `whos` pour plus de détail) pour connaître les variables actuellement définies.

```
octave:13> whos
Variables in the current scope:

  Attr Name Size Bytes Class
  ==== ==== ==== ===== =====
      A 2x2 32 double
      a 1x1 8 double
      ans 1x1 8 double
      s 1x6 6 char
      v 1x10 24 double

Total is 22 elements using 78 bytes
```

1.41

Précision numérique avec Octave

Les nombres sont stockés avec un format *double précision* respectant la *norme IEEE* sur les nombres flottants

realmin le plus petit nombre à virgule flottante (2.23×10^{-308})

realmax le plus grand nombre à virgule flottante (1.8×10^{308})

eps la précision relative (distance entre deux nombres flottants) `eps(1)`=$2.23 \times 10{-}16$ (approximativement)

1.42

Formatage des flottants

Contrôler l'affichage des flottants (`format options`)

short format à virgule fixe avec 5 décimales

long format à virgule flottante avec 5 décimales

short e format à virgule flottante avec 5 décimales

long e format à virgule flottante avec 15 décimales

short g meilleur format (virgule fixe ou flottante) avec 5 décimales.

long g meilleur format (virgule fixe ou flottante) avec 15 décimales

Voir `help format` pour des informations plus détaillées

```
octave:1> format short
octave:2> pi
ans = 3.1416
octave:3> format long
octave:4> pi
ans = 3.14159265358979

octave:5> format short e
octave:6> pi
ans = 3.1416e+00
octave:7> format short g
octave:8> pi
ans = 3.1416
```

1.43

Arrondir les flottants

Contrôler le mode d'arrondi

ceil(x) arrondit au plus petit entier supérieur à x.

floor(x) arrondit au plus grand entier inférieur à x.

round(x) arrondit à l'entier le plus proche de x.

fix(x) arrondit vers 0.

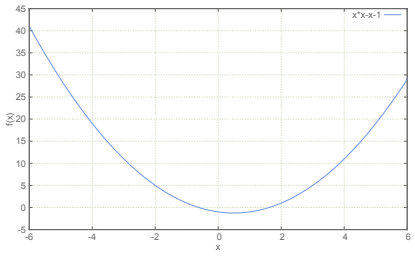
```
octave:1> [pi floor(pi) round(pi) ceil(pi)]
ans =
3.1416 3.0000 3.0000 4.0000

octave:2> fix(pi) fix(-pi)]
ans =
3 -3
```

1.44

Résolution de $x^2 - x - 1$

```
octave:1> p=[1 -1 -1]
p =
  1 -1 -1
octave:2> polyout(p)
1*s^2 - 1*s^1 - 1
octave:3> polyval(p,0.5)
ans = -1.2500
octave:4> roots(p)
```

```
ans =
  -0.61803
   1.61803
octave:5> (1+sqrt(5))/2
ans = 1.6180
octave:7> polyval(p,(1+sqrt(5))/2)
ans = 2.2204e-16
octave:8> fplot('x^2-x-1', [-5 6])
octave:9> print -dpdf trinome.pdf
```



1.45

Lecture 2
Matrices

Notes du cours *Introduction à la programmation Matlab/Octave*
Présenté le 3 novembre 2015 par Arnaud Malapert
du Département d'informatique de l'Université Nice Sophia Antipolis

1 Création d'une matrice

Création d'une matrice

Tapez simplement :

```
octave:1> A = [8 2 3 ; 3 -1 7 ; 1 5 3]
```

Octave répondra par l'affichage de la matrice : :

```
A =

  8 2 3
  3 -1 7
  1 5 3
```

Syntaxe

— Utiliser une *virgule ou* un *espace* pour délimiter les *colonnes*
— Utiliser un *point virgule* pour délimiter les *lignes*

Les expressions suivantes sont équivalentes :

```
A = [8 2 3;3 -1 7;1 5 3]
A = [8,2,3;3,-1,7;1,5,3]
```

2.46

Création d'une matrice à partir d'autres matrices

```
octave:1> A=[1 1 1;2 2 2]; B=[3;3];C=[4 4 4];
```

Ajout de colonne(s)

```
octave:2> C=[A B]
C =

  1 1 1 3
  2 2 2 3
```

Ajout de ligne(s)

```
octave:3> D=[A ; [4 4 4]]
D =

  1 1 1
  2 2 2
  4 4 4
```

En toute sécurité

```
octave:4> D=[A ; [4 4 4 0]]
error: vertical dimensions mismatch (2x3 vs 1x4)
```

2.47

Dimensions d'une matrice

`size(A,1)` renvoie le nombre de lignes de `A`

`size(A,2)` renvoie le nombre de colonnes

`size(A)` renvoie toutes les dimensions

`length(A)` renvoie la plus grande dimension

`numel(A)` renvoie le nombre d'éléments dans A

`isempty(A)` vérifie si `A` est la matrice vide `[]`

Pour Octave seulement

`rows(A)` renvoie le nombre de lignes

`columns(A)` renvoie le nombre de colonnes

2.48

Indexation

Les indices commencent à 1 !!!

```
octave:1> v=1:10;
octave:2> v(0)
error: subscript indices must be either positive integers or logicals
```

2.49

Indexation

Mémo

Par convention, on référence les lignes avant les colonnes.

- `A(i,j)` renvoie un élément
- `A(i,:)` renvoie une ligne
- `A(:,j)` renvoie une colonne
- `A(i:k,j:l)` renvoie une sous-matrice
- Une commande utile :*end*

```
octave:19> A=[1 3 5 7 11 13 17];
octave:20> A(3:end)
ans =

   5 7 11 13 17
```

2.50

Rôle du ' :'

Wildcard (ou joker)

Sélection d'une *ligne ou colonne entière* : `A(3,:)`, `A(5,:)`

Définition de plages : `start:increment:stop`

- `indices=1:5` : renvoie un vecteur ligne 1,2,3,4,5
- `steps=1:3:61` : renvoie un vecteur ligne 1,4,7,10,. . .,61
- `t=0:0.01:1` : renvoie un vecteur ligne 0,0.01,0.02,. . .,1

Astuce

Commande utile pour définir des plages : `linspace`

2.51

Pré-allocation de mémoire

Principe

— Prévoir l'espace mémoire nécessaire avant l'exécution du programme, en spécifiant la quantité nécessaire dans le code source.
— Au chargement du programme en mémoire, juste avant l'exécution, l'espace réservé devient alors accessible.

Matrices spéciales

`zeros(n,m)` matrice `n` $\times$ `m` ne contenant que 0

`ones(n,m)` matrice `n` $\times$ `m` ne contenant que 1

`eye(n)` matrice identité `n` $\times$ `n`

`diag([a b c])` matrice diagonale 3×3 dont la diagonale principale contient `a`,`b` et `c`.

2.52

Création d'une matrice de nombres aléatoires

`rand(m, n)` matrice $m \times n$ de nombres aléatoires suivant une distribution uniforme sur l'intervalle $[0, 1]$.

`randn(m, n)` matrice de nombres aléatoires suivant une distribution normale de moyenne 0 et variance 1.

`randperm(n)` un vecteur ligne contenant une permutation aléatoire des nombres 1 à `n`.

`randi(imax, m, n)` matrice de nombres entiers aléatoires suivant une distribution uniforme sur l'intervalle $[1, \texttt{imax}]$.

```
> rand(1,5)
ans =
   0.704514 0.489986 0.971673 0.040956 0.392909
> randn(1,5)
ans =
  -0.779900 0.029493 0.261672 1.118989 -0.227797
> randperm(5)
ans =
   5 4 1 2 3
> randi(10,1,5)
ans =
   6 9 6 2 2
```

2.53

2 Opérations sur les matrices

Modification d'une matrice

Affectation d'une ligne ou d'une colonne

Tous les éléments concernés sont modifiés.

```
octave:22> A=[1 3 5;7 11 13;17 19 23];
octave:23> A(3,:)=3;
```

Ajout d'une ligne ou d'une colonne

Si les éléments n'existent pas, ils sont ajoutés.

```
octave:24> A(:,4)=4
A =

   1  3  5  4
   7 11 13  4
   3  3  3  4
```

2.54

Suppression de lignes ou de colonnes

On affecte une matrice vide aux lignes et colonnes concernées.

```
octave:28> A(2,:)=[]
A =

  1  3  5  4
  3  3  3  4

octave:29> A(:,1:2:4)=[]
A =

  3  4
  3  4
```

2.55

Opérations sur les matrices

- `3*A` : multiplication par un scalaire
- `A*B + X - D` : multiplication, addition, soustraction
- `A'` : transposition
- `inv(A)` : inversion d'une matrice carré
- `v'*M*v` : mélange vecteurs et matrices
- `det(M)` : déterminant
- ...

2.56

Autres opérations

Opérations sur les vecteurs (colonnes)

- `v*v'` : produit scalaire
- `v'*v` : produit matriciel
- `v*v` : erreur.

Opérations membre à membre (vecteurs et matrices)

- `x.+x` : addition
- `x.-x` : soustraction
- `x.*x` : multiplication
- `x./x` : division
- `x.^ 3` : puissance

2.57

Exemple : rotation dans le plan euclidien

1. Construction de la matrice d'une rotation d'un angle de $\frac{\pi}{3}$:

```
octave:15> phi=pi/3
phi = 1.0472
octave:16> R = [cos(phi) -sin(phi) ; sin(phi) cos(phi) ]
R =

   0.50000 -0.86603
   0.86603  0.50000
```

2. L'image du point $(1,1)$ par une rotation de $\frac{\pi}{3}$ est ...

```
octave:17> x = [1 ; 1];
octave:18> R * x
ans =

  -0.36603
   1.36603
```

2.58

Quelques fonctions prédéfinies utiles

sum(v) somme des éléments de `v`

cumsum(v) sommes cumulées des éléments de `v`

prod(v) produit des éléments de `v`

cumprod(v) produits cumulés des éléments de `v`

diff(v) différences entre deux éléments consécutifs de `v`

```
> v=1:5;
> [ sum(v) prod(v)]
ans =
   15 120
> [ cumsum(v) ;cumprod(v) ]
ans =
    1 3 6 10 15
    1 2 6 24 120
> diff(v)
ans =
  1 1 1 1
```

```
> A= [v ; v];
> sum(A)
ans =
   2 4 6 8 10
> cumsum(A)
ans =
   1 2 3 4 5
   2 4 6 8 10
> diff(A)
ans =
  0 0 0 0 0
```

2.59

Fonctions statistiques

`mean(v)` moyenne des éléments de `v`

`std(v)` écart type à la moyenne des éléments de `v`

```
> v=1:5;
> [ mean(v) std(v) ]
ans =
  3.0000 1.5811
```

```
> A= [v ; v];
> mean(A)
ans =
  1 2 3 4 5
> std(A)
```

```
ans =
   0  0  0  0  0
```

2.60

Fonctions minimum et maximum

`min(v)` le plus petit élément de `v`

`max(v)` le plus grand élément de `v`

```
> A= [v ; v];
> v=1:5;
> [ min(v) max(v) ]
ans =
   1  5


> min(A)
ans =
   1  2  3  4  5
> min(A')
ans =
   1  1
```

2.61

Exercice : calcul d'une moyenne

Chaque ligne de la matrice `X` représente les trois notes d'un étudiant :

CC Contrôle Continu

TP Travaux Pratiques

CT Contrôle Terminal

1. Calculer les notes finales des étudiants avec les coéfficients suivants :
   — 40% CC ; 20% TP ; 40% CT.
2. Ajouter une colonne avec les notes finales.
3. Calculer les notes finales des étudiants avec les coéfficients suivants :
   — 20% CC ; 20% TP ; 40% CT ; 20% max(CC, CT) ;
4. Quelle système vous semble le plus avantageux ?
5. Ajouter une ligne pour
   — les moyennes,
   — les notes minimales,
   — les notes maximales.

2.62

Calcul d'une moyenne

— Détecter et expliquer l'anomalie observée durant l'exécution du programme.

```
> X=randi(20,5,3)
X =
   18  3  7
    1  5  9
   14 15  6
   17  4 15
    1 11  6
> F1 = [0.4 0.2 0.4] .* X
F1 =
   10.6000
    5.0000
   11.0000
   13.6000
```

```
   5.0000
> M=max(X(:,[1 3]),[], 2);
> F2 = [ 0.2 0.2 0.4] .* X + 0.2*M
F2 =
  10.6000


   6.6000
  11.0000
  13.6000
   6.0000
> find(F1>F2)
ans =
  1
  4
> X = [ X F2];
> [X ; min(X) ; max(X) ; mean(X) ]
ans =
  18.0000 3.0000 7.0000 10.6000
   1.0000 5.0000 9.0000 6.6000
  14.0000 15.0000 6.0000 11.0000
  17.0000 4.0000 15.0000 13.6000
   1.0000 11.0000 6.0000 6.0000
   1.0000 3.0000 6.0000 6.0000
  18.0000 15.0000 15.0000 13.6000
  10.2000 7.6000 8.6000 9.5600
```

2.63

Fonction de tri

`sort(v)` trie les éléments de `v`

```
> v=12:-2:1
v =
  12 10 8 6 4 2
> help randperm
...
-- Loadable Function: randperm (N)
...
   Return a row vector containing a random permutation of '1:N'.
...
octave:56> v=v(randperm(6))
ans =
  12 6 8 10 4 2
octave:57> sort(v)
ans =
   2 4 6 8 10 12


> A = [v;v];
> sort(A)
ans =
  12 6 8 10 4 2
  12 6 8 10 4 2
> sort(A,2)
ans =
   2 4 6 8 10 12
   2 4 6 8 10 12
```

2.64

Fonction de sélection

find(v)
- renvoie un vecteur contenant les indices des éléments non nuls de v.
- Très puissant en combinaison avec des tests conditionnels vectorisés

```
> v=rand(1,6)
v =
   0.079652 0.777096 0.065889 0.280128 0.254547 0.428377
> find(v>0.5)
ans = 2
> find(v>0.3)
ans =
   2 6
> v(find(v>0.3))
ans =
   0.77710 0.42838


> A = [v ; fliplr(v)]
A =
   0.079652 0.777096 0.065889 0.280128 0.254547 0.428377
   0.428377 0.254547 0.280128 0.065889 0.777096 0.079652
> find(A>0.3)'
ans =
    2 3 10 11
> A(find(A>0.3))'
ans =
   0.42838 0.77710 0.77710 0.42838
```

2.65

Exercice : carré magique

Exercice 3. `> magic(3)`

```
ans =
   8 1 6
   3 5 7
   4 9 2
```

Quelles sont les propriétés d'un carré magique d'ordre n ?

Solution. <2->
- n^2 nombres entiers *distincts*
- disposés de manière à ce que leurs *sommes*
  - sur chaque rangée,
  - sur chaque colonne et
  - sur chaque diagonale principale

  soient *égales*.
- Unique carré magique d'ordre 3 appelé *Lo Shu* (chine – 650 Av. J.-C.).

2.66

3 Matrices multidimensionnelles

Matrices multidimensionnelles

Les matrices peuvent avoir plus de deux dimensions !

Création d'une matrice à 3 dimensions

```
octave:1> A=ones(2,5,2)
A =

ans(:,:,1) =
```

```
   1  1  1  1  1
   1  1  1  1  1

ans(:,:,2) =

   1  1  1  1  1
   1  1  1  1  1
```

2.67

Matrices multidimensionnelles

Opérations sur les matrices multidimensionnelles
Toutes les opérations de création, d'indexation, d'ajout, de suppression s'appliquent directement.

Opérations sur une matrice à 3 dimensions

```
octave:2> size(A)
ans =

   2  5  2

octave:3> min(min(min(A)))
ans = 1
octave:4> A(1,2,1)
ans = 1


octave:5> A(:,1,:)
ans =

ans(:,:,1) =

   1
   1

ans(:,:,2) =

   1
   1
```

2.68

Transformation de matrices

`reshape(A,m,n)` *Changer la taille* de la matrice `A` afin qu'elle ait `m` lignes et `n` colonnes. Une erreur est levée si la matrice n'a pas `n` × `m` éléments.

`circshift(A,[ m,n])` *Décaler circulairement les éléments* de `A` de `m` lignes et `n` colonnes.

`shiftdim(A,n)` Décaler les dimensions de `A` par `n`. *transposition généralisée* pour les matrices dimensionnelles

```
octave:1> A=1:20;A=reshape(A,[2  5  2])
A =

ans(:,:,1) =

    1  3  5  7  9
    2  4  6  8  10
```

```
ans(:,:,2) =

  11  13  15  17  19
  12  14  16  18  20
```

2.69

Exemples de transformation de matrices

```
octave:2> circshift(A, [1 2])
ans =

ans(:,:,1) =

   8  10   2   4   6
   7   9   1   3   5

ans(:,:,2) =

  18  20  12  14  16
  17  19  11  13  15


octave:3> shiftdim(A, 1)
ans =

ans(:,:,1) =

   1  11
   3  13
   5  15
   7  17
   9  19

ans(:,:,2) =

   2  12
   4  14
   6  16
   8  18
  10  20
```

2.70

Lecture 3
Graphiques

Notes du cours *Introduction à la programmation Matlab/Octave*
Présenté le 3 novembre 2015 par Arnaud Malapert
du Département d'informatique de l'Université Nice Sophia Antipolis

Graphiques en 2D

- `plot(x,cos(x))` affiche un graphique en deux dimensions. Création automatique d'une fenêtre contenant la figure.
- `figure(n)` crée une fenêtre `n`. Si la fenêtre existe déjà, elle revient au premier plan.
- `figure` crée une nouvelle fenêtre et incrémente son identifiant.

3.71

Afficher plusieurs tracés

- Une série de tracés en deux dimensions : `plot(x1,y1,x2,y2,...)` e.g. `plot(x,cos(x),x,sin(x),x,x.^2)`
- Ajouter une `légende` au graphique : `legend` `legend('cos(x)','sin(x)','x^2')`
- `hold on` offre une solution alternative :

```
octave:1> x=0:0.05:pi;
octave:2> hold on
octave:3> plot(x,cos(x))
octave:4> plot(x,sin(x))
octave:5> plot(x,x.^2)
octave:6> legend('cos(x)','sin(x)','x^2','location','northwest')
```

3.72

Commandes utiles

`clf` ré-initialise la figure

`hold on` conserve les tracés (ne remplace pas les anciens tracés par les nouveaux)

`grid on` affiche un quadrillage

`grid off` enlève le quadrillage

`title('Titre')` affiche un titre pour le tracé

`xlabel('x')` affiche une étiquette pour l'axe des abcisses

`ylabel('f(x)')` affiche une étiquette pour l'axe des ordonnées

`subplot` trace une grille contenant plusieurs sous-figures

3.73

Contrôle des axes

`axis equal` définit des échelles identiques pour l'axe des abcisses et des ordonnées

`axis square` ajuste les échelles pour obtenir un graphique carré.

`axis tight` ajuste les axes en fonction des données

`axis` récupère les limites courantes des axes

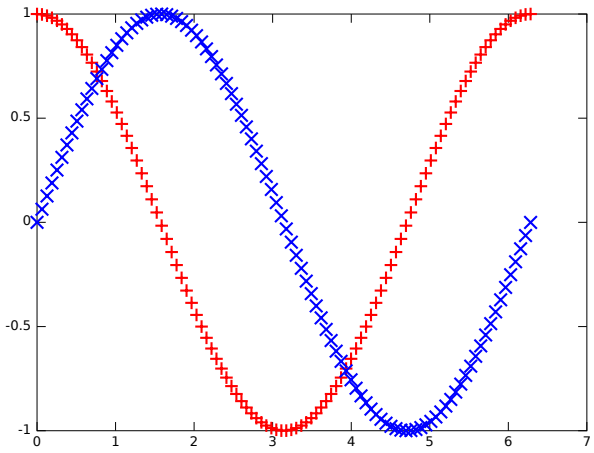


FIGURE 2 – `plot(x,cos(x),'r+',x,sin(x),'bx')`

`axis([0 1 0 2])` fixe les limites des axes

`axis off` supprime les axes

`box on` ajoute une encoche à chaque marque des axes

`box off` supprime toutes les encoches des axes

3.74

Choisir les symboles et les couleurs des tracés

— Dans `plot(x,cos(x),'r+')`, le *format* `'r+'` signifie *croix rouges*.
— Il existe un grand nombre de styles et couleurs prédéfinis : *`help plot`*

Exemple

```
octave:1> x=linspace(0,2*pi,100);
octave:2> plot(x,cos(x),'r+',x,sin(x),'bx')
```

3.75

3.76

Améliorer l'apparence

Ajuster les axes

```
octave:3> axis tight
```

Annoter le graphique

```
octave:4> legend('cos(x)','sin(x)', 'Location', 'Southwest')
octave:5> title('Trigonometric Functions')
octave:6> xlabel('x')
octave:7> ylabel('f(x)')
```

3.77

3.78

Améliorer l'apparence

Tentons un nouvel essai

```
octave:9> clf
```

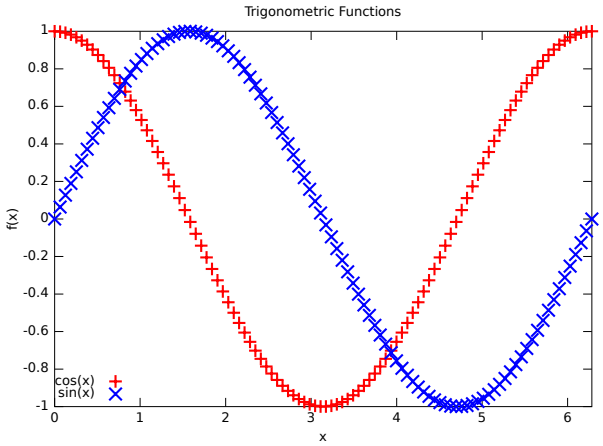


FIGURE 3 – `plot(x,cos(x),'r+',x,sin(x),'bx')` (annoté)

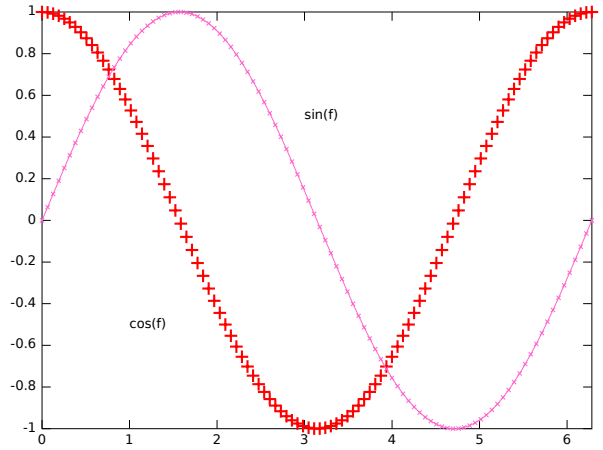


FIGURE 4 – `plot(x,cos(x),'r+',x,sin(x),'-x','Color',[1 .4 .8],'Markersize', 2)`

Controler la couleur et la taille des marqueurs

```
octave:10> plot(x,cos(x),'r+',x,sin(x),'-x','Color',[1 .4 .8],'Markersize'
    , 2)
octave:11> axis tight
```

Ajouter du texte libre

```
octave:12> text(1,-0.5,'cos(\phi)')
octave:13> text(3,0.5,'sin(\phi)')
```

On peut utiliser une syntaxe LaTeXavec certains terminaux.

3.79

3.80

Exporter les figures

Vous êtes satisfait de votre graphique . . . exportez le !

— `print -deps myPic-BW.eps` : fichier `.eps` en noir et blanc.
— `print -depsc myPic.eps` : fichier `.eps` en couleurs.
— `print -djpeg -r80 myPic.jpeg` : fichier `.jpeg` en 80 ppi.

— `print -dpng -r100 myPic.png` : fichier `.png` en 100 ppi.
— `print` peut aussi être appelée comme une fonction. Dans ce cas, les arguments et options sont lus comme une liste séparée par des virugles. E.g. : `print('-dpng','-r100','myPic.png');`

help print
pour obtenir des informations sur tous les terminaux de sortie dont certains sont spécialisés pour LaTeX.

3.81

À suivre

Un survol rapide
Nous ne présenterons pas en détail toutes les commandes graphiques pour Matlab/Octave.

— Jetez un œil à la documentation sur les commandes suivantes (`help command`) : `hist, bar, pie, area, fill, contour, quiver, scatter, compass, rose, semilogx, loglog, stem, stairs, image, imagesc` et bien d'autres encore . . .
— ou pour les graphiques 3D (la plupart des commandes 2D ont un équivalent en 3D) : `mesh, suft, plot3, bar3, pie3, fill3, contour3, quiver3, scatter3, stem3`

3.82

Lecture 4
Programmation, fonctions et scripts

Notes du cours *Introduction à la programmation Matlab/Octave*
Présenté le 3 novembre 2015 par Arnaud Malapert
du Département d'informatique de l'Université Nice Sophia Antipolis

Généralités

Les indices commencent à 1 !!!

```
octave:1> v=1:10;
octave:2> v(0)
error: subscript indices must be either positive integers or logicals
```

sensible à la casse

```
octave:2> V
error: 'V' undefined near line 2 column 1
octave:2> PLOT(v)
error: 'PLOT' undefined near line 2 column 1
```

Éditeur de texte

Utiliser un éditeur proposant au moins la coloration syntaxique :
— Gedit, GNU Emacs, QtOctave . . .

4.83

1 Structures conditionelles et répétitives

Structures conditionelles

Structure conditionelle : `if`

```
-- if (COND) ... endif
-- if (COND) ... else ... endif
-- if (COND) ... elseif (COND) ... endif
-- if (COND) ... elseif (COND) ... else ... endif
```

Exemple

```
x = 1;
if (x == 1)
 disp ('one');
elseif (x == 2)
 disp ('two');
else
 disp ('not one or two');
endif
```

Le cas du `switch`

```
x = 1;
switch x
 case 1
  disp ('one');
 case 2
  disp ('two');
 otherwise
  disp ('not one or two');
endswitch
```

4.84

if versus switch

Cas général : `if`

`switch` est une structure particulière destinée à tester la valeur d'une seule variable.

`if` only !

```
x = 1;
if (x > 0)
 disp ('positive');
elseif (x < 0)
 disp ('negative');
else
 disp ('nil');
endif
```

4.85

switch versus if

Éviter les répétitions : `switch`

`if` peut s'avérer ... quelque peu répétitif quand on teste la valeur d'une seule et même variable.

Complexe et répétitif

```
yesno = 'yes'
if( yesno == 'Yes' || yesno == 'yes' || yesno == 'YES' || yesno == 'y' ||
    yesno == 'Y')
  value = 1;
elseif(yesno == 'No' || yesno == 'no' || yesno == 'NO' || yesno == 'n' ||
    yesno == 'N')
  value = 0;
 else
  error ('invalid value');
endif
```

Simple et Concis.

```
yesno = 'yes'

switch yesno
 case {'Yes' 'yes' 'YES' 'y' 'Y'}
  value = 1;
 case {'No' 'no' 'NO' 'n' 'N'}
  value = 0;
 otherwise
  error ('invalid value');
endswitch
```

4.86

Utilitaire `menu`

— Affiche un titre suivi d'une série d'options numérotées.
— La valeur retournée est le numéro de l'option choisie par l'utilisateur.

```
octave:30> menu(morpheus, "Blue pill", "Red pill")
This is your last chance. After this, there is no turning back. You take
    the blue pill - the story ends, you wake up in your bed and believe
    whatever you want to believe. You take the red pill - you stay in
    Wonderland, and I show you how deep the rabbit hole goes.

  [ 1] Blue pill
  [ 2] Red pill

pick a number, any number: foo

error: input invalid or out of range

  [ 1] Blue pill
  [ 2] Red pill

pick a number, any number: 2
ans = 2
```

4.87

Structures répétitives (boucles)

Structures répétitives

```
  -- while (COND) ... endwhile
  -- do ... until (COND)
  -- for var=expression ... endfor
```

while

```
i=1;
N=10;
while(i<=N)
disp(i);
 i++;
endwhile
```

do-until

```
i=1;
N=10;
do
disp(i);
 i++;
until(i>N)
```

for

```
N=10;
for i=1:N
 disp(i);
endfor
%help linspace
```

4.88

Interruption et reprise des boucles

Instruction `break`

Sors de la boucle la plus profonde à laquelle appartient l'instruction.

Instruction `continue`

Saute à la fin de la boucle la plus profonde à laquelle appartient l'instruction (début d'un nouveau cycle).

Instruction à éviter

— augmente le risque de bug
— diminue la lisibilité du code

while

```
i=1;
N=10;
while(1)
if( i > 10)
 break
endif
disp(i);
 i++;
endwhile
```

do-until

```
i=1;
N=10;
do
disp(i);
 i++;
until(i>N)
```

4.89

2 Opérateurs

Opérateur d'incrémentation (Octave seulement)

Les opérateurs d'incrémentation augmente ou diminue la valeur d'une variable d'une unité.

`i++` incrémente un scalaire d'une unité

`i--` décrémente un scalaire d'une unité

`A++` incrémente tous les éléments d'une matrice d'une unité

`v--` décrémente tous les éléments d'un vecteur

4.90

Opérateurs de comparaison

Pas de type booléen en Octave

Un opérateur renvoie `1` en cas de *succès, ou 0* en cas d'*échec*.

— Pour une *comparaison entre matrices*, la comparaison est réalisé membre à membre. e.g. `[1 2;3 4] == [1 3;2 4]` renvoie `[1 0;0 1]`
— Pour une *comparaison entre une matrice et un scalaire*, le scalaire est comparée à chaque élément de la matrice. e.g. `[1 2;3 4] == 2` renvoie `[0 1;0 0]`

4.91

Autres opérateurs de comparaison

`any(v)` renvoie 1 si *au moins un élément* du vecteur `v` est *non nul*

`all(v)` renvoie 1 si *tous les éléments* du vecteur `v` sont *non nuls*

Pour les *matrices*, `any` et `all` renvoient un vecteur ligne avec le résultat des comparaisons correspondant aux colonnes de la matrice

`any(any(A))` renvoie 1 si *au moins un élément* de la matrice `A` est *non nul*

`all(all(A))` renvoie 1 si *tous les éléments* de la matrice `A` sont *non nuls*

```
octave:1> R = [1 2 3; 4 5 6] == [1 2 0 ; 4 0 0]
R =
   1 1 0
   1 0 0
octave:2> any(R)
ans =
   1 1 0
octave:3> all(R)
ans =
   1 0 0
octave:4> all(all(R))
ans = 0
```

4.92

Opérateurs relationnels

`x<y` est vraie si `x` est strictement inférieur à `y`

`x<=y` est vraie si `x` est inférieur ou égale à `y`

`x==y` est vraie si `x` est égal à `y`

`x>=y` est vraie si `x` est supérieur ou égale à `y`

`x>y` est vraie si `x` est strictement supérieur à `y`

`x~=y` est vraie si `x` est diffèrent de `y`

Pour Octave seulement

`x!=y` est vraie si `x` est diffèrent de `y`

`x<>y` est vraie si `x` est diffèrent de `y`

4.93

Opérateurs logiques

Opérateurs de négation

`~b1` renvoie 1 si l'expression est fausse

`!b1` renvoie 1 si l'expression est fausse (Octave)

Opérateurs logiques binaires

Les deux expressions sont toujours évaluées

`b1 & b2` renvoie 1 si les deux expressions sont vraies

`b1 | b2` renvoie 1 si une des deux expressions est vraie

Opérateurs logiques de court-circuit

`b2` n'est pas évaluée si elle ne peut pas modifier le résultat final.

`b1 && b2` renvoie 1 si les deux expressions sont vraies

`b1 || b2` renvoie 1 si une des deux expressions est vraie

4.94

3 Fonctions et scripts

Fonctions Octave/Matlab

— Des programmes Octave/Matlab complexes peuvent souvent être simplifiés en définissant des fonctions.
— Les fonctions sont définies dans un fichier différent (et donc utilisables pas d'autres programmes).
— Dans sa forme la plus simple, une définition de fonction est :

```
function name
body
end
```

— Définir une seule fonction par fichier `.m`.

4.95

Fonctions Octave/Matlab

— Pour passer des paramètres à une fonction ou
— définir les valeurs renvoyées par une fonction,
— taper simplement :

```
function [ret-var] = name(arg-list)
body
end
```

— `arg-list` est une liste d'arguments séparés par des virgules.
— `ret-var` est une liste de valeurs retournées séparées par des virgule.

4.96

Exemples : fonction

```
function [mu sigma] = calc_moments(data)
  mu = mean(data);
  sigma = std(data);
end

function [haspeaks i] = find_first_peak(data, thresh)
  indices = find(data > thresh);
  if isempty(indices)
    haspeaks = 0; i = [];
  else
    haspeaks = 1; i = indices(1);
  end
end
```

4.97

Fonction variadique

Fonction variadique

Une fonction variadique est une fonction d'arité indéfinie, c'est-à-dire qui accepte un nombre variable de paramètres.

`varargin` Récupérer tous les arguments dans un `cell array`. Accéder au i-ème argument avec `varargini`

`varargout` Récupérer tous les arguments dans un `cell array`. Accéder au i-ème argument avec `varargouti`

`nargin` Le nombre d'arguments en entrée.

`nargout` Le nombre d'arguments en sortie.

Voir `help varargin`, `help varargout` pour plus de détails

4.98

Fonction Octave/Matlab

— Un fichier texte avec l'extension `.m`.
— Le nom du fichier doit être celui de la fonction avec l'extension `.m`.
— Pour appeler la fonction, taper son nom sans l'extension `.m`.

```
[bool i] = find_first_peak(myreadings, 0.3);
```

— `% Ceci est un commentaire` ☺

4.99

Scripts Octave/Matlab

— Un fichier texte avec l'extension `.m`.
— Code principale à exécuter (c.f. *main* en C ou Java).
— Exécuter dans Octave en tapant son nom sans extension.
— `% Ceci est un commentaire` ☺

4.100

Écrire de la documentation

— Ajouter une aide pour vos scripts/fonctions accessible avec `help command`.
— Le premier bloc de commentaire d'un script/fonction `.m` est le texte d'aide.

```
## -*- texinfo -*-
## @deftypefn {Function File} {} shift (@var{x}, @var{b})
## @deftypefnx {Function File} {} shift (@var{x}, @var{b}, @var{dim})
## If @var{x} is a vector, perform a circular shift of length @var{b} of
## the elements of @var{x}.
##
## If @var{x} is a matrix, do the same for each column of @var{x}.
## If the optional @var{dim} argument is given, operate along this
## dimension.
## @end deftypefn

...

function y = shift (x, b, dim)
...
endfunction
```

4.101

Configurer les répertoires de scripts/fonctions

`path` afficher la liste des répertoires

`addpath('dir')` ajouter un répertoire au début de la liste.

`rmpath('dir')` supprimer un répertoire de la liste.

`savepath` sauvegarder la liste de répertoires.

4.102

Conventions de nommage

Fonctions mots séparés par des tirets-bas (underscore)
— Exemples : `intersect_line_circle.m`, `draw_robot.m`, `calc_probability.m`

Scripts mots séparés par des majuscules (premier mot en minuscule).
— Exemples : `LocalizeRobot.m`, `MatchScan.m`

Commandes Matlab/Octave en minuscules sans tirets ni tirets bas.
— Exemples : `continue`, `int2str`, `isnumeric`

4.103

4 Fonctions d'entrées/sorties

Fonction `save`

On a souvent besoin de sotcker résultats intermédiaires ou finaux sur le disque.

— `save my_vars.mat` Sauvegarder toutes les variables dans le fichier `my_vars.mat`
— `save results.mat resultdata X Y` Sauvegarder les variables `resultdata`, `X` et `Y` dans le fichier `results.mat`
— `save ... -ascii` Sauvegarder au format ASCII.
— `save ... -mat` Sauvegarder au format binaire MAT.

4.104

Fonction `load`

Et réciproquement, on a souvent besoin de lire des données sur le disque.

— `load my_vars.mat` Lire toutes les variables du fichier `my_vars.mat`
— `load results.mat X Y` Lire seulement `X` et `Y` du fichier `results.mat`
— `A = load('data.txt')` Lire un fichier texte ASCII contenant une matrice
  — colonnes séparées par un ou plusieurs espaces
  — lignes séparées par les lignes ;-)

4.105

Entrées/sorties à la C++

`open` Ouvrir ou créer un fichier pour lecture/écriture

`close` Fermer un fichier

`write` Écrire dans un fichier à la C++.

```
 v = randn(1000,1);
fid = fopen('gauss.txt','w');
fprintf(fid,'\%7.4f\n',v);
fclose(fid);
```

4.106

Remarque sur les entrées/sorties

— La précision de la fonction `printf` est cruciale si votre programme lit ou écrit des fichiers.
— Assurez-vous de toujours écrire les nombres avec une précision adéquates.
— L'exemple précédent, le format produira une mauvaise source de nombres aléatoires.

4.107

Entrées/sorties avancées

`textread` Lire du texte formaté depuis un fichier.

`fscanf` Lire du texte formaté depuis un fichier.

`fgetl` Lire une ligne d'un fichier.

`fread` Lire des données binaire d'un fichier

`imread` Lire une image dans un fichier (différents formats)

`imwrite` Écrire une image dans un fichier (différents formats)

4.108

Fonction `clear`

— `clear A` Supprimer la variable `A`.
— `clear A*` Supprimer toutes les variables dont le nom commence par `A`.
— `clear` Supprimer toutes les variables.
— `clear all` Ré-initialisation : variables, globales, fonctions, liens ...
— `close` Fermer la fenêtre graphique en premier plan.
— `close all` Fermer toutes les fenêtres graphiques
— `clc` Effacer la fenêtre de l'interpréteur.

4.109

Fonction `disp`

Afficher de jolis messages.

— `disp(A)` Afficher une matrice sans afficher son nom de variable.
— `disp(str)` Idem avec une chaîne de caractère.

Exemple

```
> disp('done')
done
> 'done'
ans = done
> sprintf('done')
ans = done
> disp ("The value of pi is:"), disp (pi)
The value of pi is:
 3.1416
```

4.110

5 En pratique

En pratique

— Naviguer efficacement dans l'*historique* de l'interpéteur.
— Génération reproductible de nombres aléatoires.
— *Lire et exécuter des fichiers* générés depuis des programmes Excel/C++/Python/Java/... avec une syntaxe Octave.
— *Générer des fichiers de données* pour un tableur (Excel, OpenOffice,...).
— Faire des *animations*.
— Appeler des *fonctions unix/dos* depuis un programme octave.
— Améliorer les *performances*.

4.111

Historique et complétion de commandes

Historique des commandes

— Naviguer dans l'historique des commandes avec les flèches verticales.
— Navigation réactive : tapez une ou plusieurs lettres pour restreindre la navigation aux commandes de l'historique commençant par ces lettres.

Complétion

Octave/Matlab a un mécanisme de complétion quand on presse la tabulation. Taper des lettres suivi de tabulation pour obtenir une liste des commandes débutant par ces lettres.

```
> file
file_in_loadpath fileattrib fileparts filesep
file_in_path filemarker fileread
```

4.112

Séquences reproductibles de nombres aléatoires

Générer de nombres aléatoires

Les fonctions de génération de nombres aleatoires (famille `rand`) initialise leur graîne initiale depuis l'horloge système.

Générer des séquences reproductibles/identiques

il faut fixer la graîne manuellement :

```
rand('seed',value)
```

4.113

Lire des fichiers de données

Méthode 1

— Écrire les données dans un fichier au format ASCII depuis un programme externe.
— Charger le fichier dans Octave avec la commande `load`.

Méthode 2

— Écrire les données dans un fichier texte (par ex. `.csv`).
— Charger le fichier dans Octave avec la commande `dlmread`.

4.114

Lire des fichiers de script

— Écrire les données et des commandes Octave dans un fichier `.m` depuis un programme externe.
— Exécuter le script dans *Octave* en tapant son nom sans l'extension `.m`.
— Exécuter le script dans un *Shell* en tapant son nom et en insérant la première ligne suivante :

```
#!/usr/bin/octave -qf
```

4.115

Animations

— `Matlab` a des commandes (`getframe`, `movie`) pour faire des animations à partir de graphiques.
— `Octave` ne supporte pas ces commandes, mais vous
— Cependant, il est possible de contourner le problème.
  — Exporter les graphiques dans un repertoire `frames`
  — Utliser `print` depuis une `for-loop`.
  — Créer une animation avec un programme externe (ImageMagick, Quicktime Pro).
  — Attention, il faut fixer les limites des axes pour l'animation.

4.116

Commandes Unix/Dos

— Intéragir efficacement avec des programmes externes.
— Pour les systèmes Unix/Linux/MacOSX, la commande Octave/Matlab `unix` permet d'exécuter des commandes Shell et renvoie leur résultat.

```
unix('ls -al')
unix('ftp < ftp_script')
unix('./myprogram')
```

— Pour Windows, la commande `dos` remplit un rôle similaire.

4.117

Commandes Octave pour le système

Commandes Unix prédéfinies

`pwd` Display current working directory

`ls` List directory. See also `dir`.

`cd` Change directory

`mkdir` Make new directory

`rmdir` Delete directory

Autres Commandes

`movefile` commande shell `cp`

`copyfile` commande shell `mv`

4.118

Optimisation des performances

Lenteur des programmes Octave/Matlab

Une faiblesse reconnu qui provient principalement du code utilisateur et pas des fonctions préfinies !

Rules of Optimization (M.A. Jackson)

Rule 1 Don't do it.

Rule 2 (for experts only) Don't do it yet.

Premature optimization (D. Knuth)

We should forget about small efficiencies, say about 97% of the time : *premature optimization is the root of all evil.*

4.119

Optimisation des performances

- ☹ Les boucles `for` sont démoniaques.
- ☺ Vectorisation
- ☺ Préallocation.
- ☺ Structure de tableaux
- ☹ Tableaux de structure.

4.120

Vectorisation

— Soit `phi = linspace(0,2*pi,100000)`, le code

```
for i = 1:length(phi),
sinphi(i) = sin(phi(i));
end;
```

— est *significativement plus lent que*

```
sinphi = sin(phi);
```

— Quasiment toutes les fonctions préfinies sont vectorisées.

☺ Pensez vectorisé

4.121

Préallocation

1. Si une boucle `for` ou `while` ne peut être évitée,
2. Ne pas augmenter la taille des structures dans la boucle.

```
for i = 1:100,
A(i,:) = rand(1,50);
end;
```

3. Pré-allouer intégralement la mémoire.

```
A = zeros(100,50); % preallocate matrix
for i = 1:100,
A(i,:) = rand(1,50);
end;
```

4. Si on ne connaît pas la taille à l'avance, pré-allouer la mémoire par bloc.

4.122

Structure of Arrays

— *Utiliser moins de mémoire*
— *Améliorer aussi la vitesse d'exécution*

☺ Structure of arrays

```
data.x = linspace(0,2*pi,100);
data.y = sin(data.x);
```

☹ Array of structure

```
people(1).name = 'Polly J Harvey';
people(1).age = 32;
people(2).name = 'Monica Lebowski';
people(2).age = 27;
```

4.123

Bibliographie

Bibliographie

Références

[1] Kai Arras. Octave/Matlab Tutorial, October 2009. University of Freiburg.

[2] Jean-Marie Chesneaux. Validité du logiciel numérique, 2006. Université Paris VI.

[3] Marc Daumas and Jean-Michel Muller. *Qualité des calculs sur ordinateur : vers des arithmétiques plus fiables ? [Jean-Claude Bajard, Olivier Beaumont, Jean-Marie Chesneaux,... et al.] coord. par Marc Daumas et Jean-Michel Muller*. Recherche en informatique. Masson, Paris, Milan, Barcelone, 1997.

[4] P.J.G. Long. Introduction to Octave, September 2005. University of Cambridge.

[5] Daniel Muller. Systèmes à microprocesseurs. http://tic01.tic.ec-lyon.fr/~muller/trotek/cours/, 1998. École Centrale de Lyon.

[6] Alfio Quarteroni, Fausto Saleri, and Jean-Frédéric Gerbeau. *Calcul scientifique : cours, exercices corrigés et illustrations en MATLAB et Octave*. Springer, Milan, 2006. Trad. de : Introduzione al calcolo scientifico : esercizi e problemi con MATLAB.

4.124