Introduction à Matlab/Octave*

Ozan BAKIS

1 Introduction

GNU Octave[1] est un langage de haut niveau pour le calcul numérique. Il est libre: on peut l'utiliser, le distribuer et même le modifier librement à condition de respecter son licence (GNU GPL[2]). Un autre atout de GNU Octave (complété par Octave-Forge, http://octave.sourceforge.net/) est qu'il est presqu'identique à Matlab – le logiciel devenu *de facto* standard dans le domaine le calcul numérique – en terme de syntaxe[3].

1.1 Téléchargement et installation

A la base GNU Octave est développé pour GNU/Linux. Mais maintenant il existe aussi la version Windows à l'adresse: http://octave.sourceforge.net/. A cette adresse cliquez sur **Windows installer**. Ceci va télécharger GNU Octave. Pour l'installer executez ce dossier téléchargé (double-cliquez là dessus). Une étape importante est le choix des composants. Il faut accepter ce qui est proposé par défaut et en plus activez tous les packages de la section "Extra". Mais au cas où on aura besoin des paquetages d'Octave-Forge, nous allons voir ensuite comment installer un paquetages à partir d'Octave-Forge.

*Version préliminaire et incomplète...

[1] http://www.gnu.org/software/octave/

[2] http://en.wikipedia.org/wiki/GNU_General_Public_License

[3] Il n'existe pas de (tous les) "toolboxes" chez GNU Octave. Ce sont des (*package*s, i.e. groupe de fonctions) spécialisée pour des tâches spécifique comme optimisation, calcul symbolique, statistique... Mais les paquetates de Octave-Forge sont un substitut partiel pour ces "toolboxes". Comme le projet GNU Octave est un projet relativement jeune et en développement actif, on s'attend à ce que les différences actuelles entre Matlab et GNU Octave diminuent au cours du temps.

1.2 Démarrage

Une fois installé, vous lancez le programme de manière standard: soit à partir de l'icone sur le bureau, soit: Start → All programs → GNU Octave → Octave. Pour terminer GNU Octave, il faut faire **quite** ou **exit**.

1.3 Aide

Pour avoir l'aide sur une fonction, disons "sqp" ce qui est *successive quadratic programming* faites **help sqp**. Pour sortir de l'écran aide tapez **q**. Pour chercher un mot-clé, disons solve, dans toutes les fonctions la commande est **lookfor solve**. La touche TAB peut compléter les noms des fonctions et des variables. M-? (Pressez sur la touche Alt et tapez sur ?) va montrer toutes les possibilités pour compléter une fonction / variable.

1.4 Sauvergarder les sessions de GNU Octave

Pour enregistrer les commandes et les outputs de ces commandes dans un fichier la commande est **diary**. Pour sauvegarder a=4, b=5, a*b dans un fichier appelé `dene_diary` on émet les commandes suivantes:

```
diary dene_diary
a=4
b=5
a*b
exit
```

et si l'on ouvre le fichier `dene_diary` on voit

```
octave:13> a=4
a =  4
octave:14> b=5
b =  5
octave:15> a*b
ans =  20
octave:16> diary off
```

Si l'on ne donne pas un nom spécifique on doit utiliser les commandes **diary on** pour commencer à enregistrer et **diary off** pour terminer ce processus. Dans ce cas un fichier appelé "diary" va être généré automatiquement.

1.5 Caractères spéciaux

; → si le point-virgule est ajouté à la fin d'une commande, cette commande est exécutée mais son résultat n'est pas affiché. Le point virgle peut séparer aussi plusieurs commandes (comme a=4; b=5;) sur la même ligne.

`,` → la virgule sert à séparer plusieurs commandes lorsque l'on souhaite passer plusieurs commandes sur la même ligne. La différence entre a=4,b=5 et a=4; b=5;???

`...` → trois points seront utilisés en fin de ligne pour pouvoir continuer une série de commande /instruction sur la ligne suivante

`:` → opérateur de définition de séries. Par exemple, 1:4 définit la série "1 2 3 4"

`%` → pour entrer un commentaire..

`'` → caractère utilisé pour entrer une chaîne de caractère, comme 'univ gsu'. Ceci est aussi utilisé comme opérateur de transposition de matrice.

1.6 Affichage des nombres

```
Nombre       mantisse - exposant   MATLAB form
-----------  -------------------   ------------
789.34        7.8934 x 102         7.8934e2
0.0001        1 x 10-4             1e-4
4             4 x 100              4
400000000000  4 x 1011             4e11
```

1. `format short` (Par défaut) → notation décimale fixe à 5 chiffres, ex: 72.346
2. `format short e` → notation décimale flottante avec exposant, ex: 7.2346e+001
3. `format long` → précision maximale : 15 chiffres significatifs, ex: 72.346000000000000
4. `format long` → Ex: 7.23460000000000e+001
5. `format bank` → 2 chiffres après virgule, Ex:72.35
6. `format rat` → approximation par des expressions rationnelles. Ex: 3.333333 s'affichera 10/3

Essayons

```
s = [1/2 1/3 pi sqrt(2)];
format short; s
format long; s
format rat; s
format ; s
```

2 Notions de base

2.1 Généralités

- `workspace`→ espace de travail.
- `variable`→ élément de base dont le nom consiste en une lettre suivie d'un nombre quelconque de lettres, chiffres ou caractères souligné "_". Par ex. x_min, f56, un_nom_tres_long, etc. Mais les noms suivants ne sont pas valides: 6x_min, x-5,...
- pour appeler/sauvegarder plusiers variables en même temps on peut utiliser les charactères "*", "?". Supposons que nous avons
  - ax=5
  - abx=7
  - abcx=6

  who *x va lister toutes les trois variables, alors que who ?x va afficher seulement ax=7.
- expression: Une "expression" est une construction valide faisant usage de nombres, de variables, d'opérateurs et de fonctions. Par Ex: pi*r+2 est une expression.

Comment peut on faire calculer l'expression suivante à Matlab/Octave

$$b - \frac{a}{b + \frac{b+a}{ca}}$$

```
a = 3;
b = 5;
c = -3;
x = b-a/(b+(b+a)/(c*a));
x
```

2.2 Scalaires et constantes

Un scalaire est un nombre réeel, ou de manière équivalente un vecteur ayan un seul élément. Ex. a=4, b= sqrt(3), ...

Il existe aussi des constantes sont pré-définies.

1. pi

2. e
3. i
4. true
5. false
6. inf (ou Inf)
7. nan (ou NaN: indéterminé)
8. NA (valeur manquante)...

Comme les constantes, nous avons les variables spéciales de nom prédéfini.

1. ans → nom de variable par défaut pour les résultats
2. nargin → nombre d'arguments passés à une fonction
3. nargout → nombre d'arguments retournés par une fonction

2.3 Fonction prédéfinies

Soit **var** une variable quelconque. On a:

`sqrt(var)` : Racine carrée de var.

`exp(var)` : Exponentielle de var

`log(var)` et `log10(var)` : Logarithme naturel, resp. base 10, de var

Et d'autres fonctions trigonomètrique: cos(var), acos(var), sin(var), asin(var), sec(var), csc(var), tan(var), atan(var), cot(var), acot(var), cosh(var), acosh(var), sinh(var)...

`factorial(n)`: Factorielle de n

`rand` / `rand(n)` / `rand(n,m)`: Génère un nombre / une matice carrée de nxn / une matrice nxm de nombre aléatoires compris entre 0.0 et 1.0. Si l'on veut que la matrice créée soit distribuée selon la loi normale il faut uriliser `randn` au lieu de `rand`.

```
rand
rand(3)
rand(3,2)
rand(6,1)
randn
randn(3)
randn(6,1)
```

`fix(var)`: Arrondi à l'entier inférieur

`round(var)`: Arrondi à l'entier le plus proche, de var

`floor(var)`: Le plus grand entier qui est inférieur ou égal à var

`ceil(var)`: Le plus petit entier plus grand ou égal à var

`mod(var1,var2)`: Fonction var1 "modulo" var2

`rem(var1,var2)`: Reste ("remainder") de la division de var1 par var2

Remarques:

- var1 et var2 doivent être des scalaires réels ou des tableaux réels de même dimension

- rem a le même signe que var1, alors que mod a le même signe que var2

- les 2 fonctions retournent le même résultat si var1 et var2 ont le même signe

`abs(var)`: Valeur absolue (positive) de var

`sign(var)`: (signe) Retourne "1" si var>0, "0" si var=0 et "-1" si var<0

`real(var)`et `imag(var)`: Partie réelle, resp. imaginaire, de la var complexe

`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

Essayons:

```
a = [-1.9, -0.2, 3.4, 5.6]
ceil(a)%Round toward positive infinity
fix(a)% Round toward zero
floor(a)%Round toward negative infinity
round(a)% Round to nearest integer
```

2.4 Séries

a = 1:5 crée le vecteur a=[1 2 3 4 5] alors que x=1.7:4.6 crée le vecteur x=[1.7 2.7 3.7]. Si l'on ne précise pas un nom précis l'output sera nommé **ans**. 1:5 crée le vecteur ans=[1 2 3 4 5].

Si on veut donner aussi le pas d'incrémentation la commande est `-4:-2:-10.0` ce qui retourne le vecteur ans=[-4 -6 -8 -10]. Attention, la commande `-4:-2:-11.7` aussi retourne le vecteur ans=[-4 -6 -8 -10]

Lorsqu'on connait la valeur de début, la valeur de fin et que l'on souhaite générer des séries linéaires ou logarithmique de nbval valeurs, on peut utiliser les fonctions suivantes :

série = linspace(début,fin ,nbval)

Crée une série (vecteur ligne) de nbval éléments linéairement espacés de la valeur début jusqu'à la valeur fin. Si l'on omet le paramètre nbval, c'est une série de 100 éléments qui est créée Ex: `v=linspace(0,-5,6)`, `v=linspace(0,-5,3)`.

2.5 Vecteurs et Matrices

2.5.1 Les bases

Pour entrer un vecteur on utilise les "[ ]". Pour séparer les éléments d'un vecteur on peut utiliser soit "," soit un espace. Ex:

```
v1=[1 -4 5], v2=[-3,sqrt(4)] et v3=[v2 v1 -3]
```

Des vecteurs colonnes peuvent êtres crées en séparant ses éléments par point-virgule, Ex:

```
v4=[-3;5;2*pi], v5=[11 ; v4]
```

ou en transposant un vecteur ordinaire, Ex:

```
v6=[3 4 5 6]'
```

la commande `vec(i:j)=[]` va détruire des éléments i à j du vecteur vec (qui est redimensionné en conséquence). Alors que

```
vec([k l m])=[]
```

détruit respectivement des k-ème l-ème et m-ème éléments.

Ex: soit `v10=(11:20)`

l'instruction v10(4:end)=[] redéfinit v10 à [11 12 13] alors que

```
 v10([1 3:7 10])=[]
```

redéfinit v10 comme [12 18 19]

`length(vec)` retourne la taille (nombre d'éléments) du vecteur ligne ou colonne vec.

Question: Construire un vecteur contenant des valeurs de zéro à 1 où la valeur d'incrément (ou le pas d'incrémentation) est 0.1.

```
x = [0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0];
x = 0:0.1:1.0;
x = linspace(0,1,11);
```

Pour entrer une matrice mat de n lignes et de m colonnes il faut entrer les éléments d'une ligne séparés par des <espace>, ou , (virgules), et indiquer la fin de chaque lignes par des ; (point-virgules) et/ou par la touche <Enter>. Il faut qu'il y aie exactement le même nombre de valeurs dans chaque ligne, sinon l'affectation échoue.

```
m1=[-2:0 ; 4 sqrt(9) 3]
```

définit la matrice de 2 lignes x 3 colonnes avant pour valeurs [-2 -1 0 ; 4 3 3]

Construction d'une matrice mat par concaténation de vecteurs: si `v1=1:3:7` et `v2=9:-1:7`, alors

```
m2=m2=[v2;v1]  %retourne la matrice [9 8 7 ; 1 4 7]
m2=m2=[v2,v1]  %retourne la matrice [9 8 7 1 4 7]
m2=m2=[v2 v1]  %retourne aussi la matrice [9 8 7 1 4 7]
```

Concaténation des matrices et vecteurs:

Par ex. l'ajout devant la matrice m2 ci-dessus de la colonne `v3=[44;55]` avec `m2=[v3,m2]`, (ou `m2=[v3 m2]`) donne m2=[44 9 8 7 ; 55 1 4 7]

`ones(n{,m})` : renvoie une matrice de n lignes x m colonnes dont tous les éléments sont égaux à "1". Si m est omis, crée une matrice carrée de dimension n

Ex: `c*ones(n,m)` renvoie une matrice n x m dont tous les éléments sont égaux à c

`zeros(n{,m})` : renvoie une matrice de n lignes x m colonnes dont tous les éléments sont égaux à "0". Si m est omis, crée une matrice carrée de dimension n

`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". Si m est omis, crée une matrice carrée de dimension n

`diag(vec)` : 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"

`diag(mat)` : 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

Ex: `repmat(eye(2),1,2)` retourne [1 0 1 0 ; 0 1 0 1]

`[n m]=size(var)` : size renvoie, sur un vecteur ligne, la taille (nombre n de lignes et nombre m de colonnes) de la matrice ou du vecteur var. Les fonctions rows et columns retournent respectivement le nombre n de lignes et nombre m de colonnes.

Ex: si `m=ones(3,3)` alors `mat2=eye(size(m))` définit une matrice identité "mat2" de même dimension que la matrice m.

`length(mat)` : Appliquée à une matrice, cette fonction analyse le nombre de lignes et le nombre de colonnes puis retourne le plus grand de ces 2 nombres (donc identique à max(size(mat))). Cette fonction est par conséquent assez dangereuse à utiliser sur une matrice !

`mat(i,j)` : Désigne l'élément (i,j) de mat, donc retourne un scalaire

`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

Notez bien les formes simplifiées très courantes de cette notation pour désigner des lignes ou colonnes entières d'une matrice :

- `mat(i,:)` : la ligne i
- `mat(i:j,:)` : les lignes i à j
- `mat(:,k)` : la colonne k
- `mat(:,k:m)` : les colonnes k à m

La notation d'indices entre crochets `[]` permet de désigner un ensemble continu ou discontinu de lignes et/ou de colonnes

Ex 1: si l'on a la matrice m3=[1:4; 5:8; 9:12; 13:16]

- `m3([2,4],1:3)` retourne [5 6 7 ; 13 14 15]
- `m3([1,4],[1,4])` retourne [1 4 ; 13 16]

`mat(i)` ou `mat(i:j)` : Lorsque l'on adresse une matrice à la façon d'un vecteur (en ne précisant qu'un indice i ou une série i:j pour cet indice), la recherche s'effectue en numérotant les éléments de la matrice colonne après colonne. La première forme retourne, sur un scalaire, le i-ème élément ; la seconde forme retourne, sur un vecteur ligne, les i-ème au j-ème éléments.

`mat(:)` : Retourne un vecteur colonne constitué des colonnes de la matrice (colonne après colonne).

Ex: si `m4=[1,2;3,4]`, alors `m4(:)` retourne [1 ; 3 ; 2 ; 4]

`mat(i:j,:)=[]` et `mat(:,k:m)=[]` : Destruction de lignes ou de colonnes d'une matrice (et redimensionnement de la matrice en conséquence). La première expression supprime les lignes i à j, et la seconde supprime les

colonnes k à m. Ce type d'opération ne permet de supprimer que des lignes entières ou des colonnes entières

Ex: en reprenant la matrice m3 ci-dessus, l'instruction `m3([1,3:4],:)=[]` réduit cette matrice à la seconde ligne [5 6 7 8]

```
size(m3)%dimension
rows(m3)%number of rows
columns(m3)%number of columns
numel(m3)%number of elements

A=[1,2,3;
   4,5,6;
   7,8,9]
```

Si l'on veut changer la troisième colonne par 6,12,18, la commande est

```
A(:,3)= [6,12,18]% essayer   A(:,3)= 6,12,18
A(:,3)= A(:,3)*2
```

Analysons les commands suivantes:

```
x = linspace(0,1,10)
y = x(1:end)
y = x(1:end/2)
y = x(2:2:end)
y = x(2:end-1)
```

2.6 Opérateurs

2.6.1 Opérateurs arithmétiques

1. * → Multiplication
2. / → Division, 6/3=2
3. → Division à gauche, 3 6=2
4. ^(ou **) → Puissance
5. () → Paranthèses vont changer l'ordre de précédence

```
s = 1:6;
t = 6:-1:1;
s+t
s-t
s.*t
s./t
s.^2
1./s
s/2
s+1
```

2.6.2 Opérateurs relationolles

Les opérateurs relationolles sont utilisés pour tester deux expressions

1. == (ou eq) → Test égalité; a==b, a eq b
2. ~= (ou ne) → Test de différence; a ne b
3. < (ou lt) → Test d'infériorité; a < b, a lt b
4. > (ou gt) → Test sepériorité; a > b, a gt b
5. <= (ou le) → Test d'infériorité ou égalité; a <=b, a le b
6. >= (ou ge) → Test de supériorité ou égalité; a >=b, a ge b

Essayons:

```
A = [1 2 ; 3 4]
A >= 2
A < 2
B = [1 3 ; 4 2]
A < B
```

2.6.3 Opérateurs logiques

Les opérateurs logiques .... Soit `a=1,b=0`

1. ~ : Négation logique; `~1` retourne 0.
2. `&` : Et logique; `a&b`retourne 0.
3. `|` : Ou logique `a|b` retourne 1.

Remarque: Short-Circuit Operators

The following operators `&&` and `||` perform AND and OR operations on logical expressions containing **scalar values**. They are short-circuit operators in that they evaluate their second operand only when the result is not fully determined by the first operand.

`&&` returns logical 1 (true) if both inputs evaluate to true, and logical 0 (false) if they do not.

`||` returns logical 1 (true) if either input, or both, evaluate to true, and logical 0 (false) if they do not.

```
A=[0 0 1 1],B=[0 1 0 1]
A&B % retourne le vecteur [0 0 0 1]
A|B % retourne le vecteur [0 1 1 1]
a=5,b=4;
(a<7)&&(b<2) % retourne 1
(a<3)&&(b<2) % retourne 0
(a<3)||(b<2) % retourne 1
```

L'usage typique de && est dans les fonctions ou les boucles. from mathworks: `You can also use the && and || operators in if and while statements to take advantage of their short-circuiting behavior.`

```
if (nargin >= 3) && (...)
{
  ...expressions...
}
```

Si le premier argument est faux (c.a.d nargin < 3) quelque soit le deuxième les expressions suivantes ne seront pas évaluées.

2.6.4 Opérateurs d'incrémentation

a++

++a

Les deux augmentent a de 1. Mais si l'on veut affecter la valeur de a à une nouvelle variable b, b prend l'ancienne valeur de a avec a++, et nouvelle valeur de a avec ++a.

a =5; b=a++

Ici, b=5 et a = 6; parce que l'affectation précède l'incrémentation. Et si maintenant l'on fait c = ++a, on obtient a = 7 = c parce que l'affectation suit l'incrémentation.

Cela suit une logique similaire pour a– et –a.

2.6.5 Opérateurs matricielles

Addition et soustraction: Les 2 arguments doivent être des vecteurs ou matrices de même dimension, sauf si l'un des arguments est un scalaire. Dans ce cas-ci l'addition/soustraction s'applique sur tous les élément du vecteur ou de la matrice.

Ex: [2 3 4]-[-1 2 3] retourne [3 1 1], et [2 3 4]-1 retourne [1 2 3]

Produit matriciel: Soient A et B deux matrices. Pour pouvoir aplliquer $A*B$ il faut que le nombre de colonnes de A soit égal au nombre de lignes de B. Si A ou B est un scalaire la multiplication s'applique à tous les élément de l'autre.

Ex: [1 2]*[3;4] ou [1 2]*[3 4]′ produit le scalaire "11" (mais [1 2]*[3 4] retourne une erreur!) 2*[3 4] retournent [6 8]

Produit/division/puissance éléments par éléments:

```
A=[1 2;4 6], B=[3 -1;5 3], b=[1;1], c=[3;4]
A.*B % retourne [3 -2 ; 20 18]
A*B % retourne [13 5 ; 42 14]
A*2 % ou $A.*2$ retournent [2 4 ; 8 12]
A\b % retourne [-2 ; 1.5]
b./c % et c.\b retourne [0.33 ; 0.25]
c./b % et b.\c retourne [3 ; 4]
A^2 % retourne [9 14;28 44]
A.^2 %  retourne [1 4;16 36]
```

Division matricielle à gauche Considérer le système linaire $A * X = B$. `A\B` donne la solution X de ce système.

Division matricielle (à droite)

B/A est la solution X du système $X * A = B$. Nous n'allons pas utiliser beaucoup cette commande...

Trouver x,y donnés qui résoud le système:

$$x - y = 3$$
$$x + y = 5$$

On peut réécrire le même système comme $Ax = b$ où `A=[1,-1;1,1]` et `b=[3;5]`. La commande de solution est `x=A\b`. Pour verifier on peut essayer `inv(A)*b`.

2.6.6 Fonctions matricielles

Pour transformer une matrice en un vecteur colonne la commande est `(:)`. Ex. `m=[7,4,6;5,6,3]`alors, `m(:)`retourne [7; 4; 6;5; 6; 3]

Pour trier dans l'ordre: Soit `m=[7,4,6;5,6,3]`

-
- `sort(m,2)` trie les éléments à l'intérieur des lignes

```
sort(m) % trie les éléments à l'intérieur des colonnes
        % par l'ordre croissant.
sort(m,1) % idem...
sort(m,2) % même chose mais pour les lignes...
sort(m,1,'descend') % ... par l'ordre décroissant.
sort(m,2,'descend') % ... par l'ordre décroissant.
```

Pour prendre en compte les autres valeurs de la matrice la commande est `sortrows(m)` , ce qui est équivalent à `sortrows(m,1)`, trie les éléments de la matrice `m` en fonction de la 1ère colonne. Pour trier en fonction de la colonne j, la commande est `sortrows(m,j)`.

```
m=[2,3,1; 4,5,6;9,8,7]'
sortrows(m,1)
sortrows(m,2)
sortrows(m,3)
```

Pour trouver l'inverse d'une matrice la commande est `inv()`. Par ex. pour `A=[1,2;3,4]`, `inv(A)` retourne [-3.0 1.0; 2.0 -0.50]. Et pour le déterminant on a `det(A)` qui retourne -2.

- `trace(A)` nous donne la somme des éléments de la diagonale
- `rank(A)` nous donne son rang, i.e. le nombre de lignes/colonnes linéairement indépendantes.
- `min(A)` et max(A) donneent respectivement l'éélement le plus petit et le plus grand de chaque colonne.
- `sum(A)` ou sum(A,1) retourne la somme des colonnes alors que sum(A,2) retourne la somme des lignes.
- `prod(A)` ou `prod(A,1)` retourne le produit des colonnes alors que prod(A,2) retourne le produit des lignes.
- `cumsum(A)` ou `cumsum(A,1)` retourne la somme cumulée des colonnes alors que `cumsum(A,2)` retourne la somme cumulée des lignes.
- `cumprod(A)` ou `cumprod(A,1)` retourne le produit cumulé des colonnes alors que `cumprod(A,2)` retourne le produit cumulé des lignes.
- `mean(A)` ou `mean(A,1)` retourne la moyenne des colonnes alors que `mean(A,2)` retourne la moyenne des lignes.
- `median(A)` ... `median(A,1)` ... `median`(A,2) ...
- `cov(A)` retourne la matrice de covariance...
- `eig(A)` retourne les valeurs propres de A; `[evect,eval]=eig(A)` retourne les vecteurs et les valeurs propres.

- `find(A)` retourne des indices des éléments non-nuls de A; `find(A>7)` retourne des indices des éléments de A dont la valeur est supérieure à 7. `A(find(A>7))=99` remplace les éléments de A supérieur à 7 par 99.
- `unique(A)`retourne les éléments de A triés dans un ordre croissant et sans répétitions.
- `intersect(A,B)` retourne les éléments communs sans répétitions.
- `setdiff(A,B)` retourne les éléments qui ne sont pas communs sans répétitions.
- `union(A,B)` retourne les éléments communs et non-communs sans répétitions.

Il existe aussi une catégorie des fonctions matricielles logiques

- `isequal(A,B)` retourne 1 si A et B sont égaux sinon 0.
- `ismember(A,B)` retourne 1 si un élément de A est aussi élément de B et 0 sinon.
- `any(A)` retourne 1 pour les colonnes dont au moins un élément est non-nul.
- `A(A>7)=100` va transformer les éléments de A supérieurs à 7 en 100. C'est l'équivalent de la commande `A(find(A>7))=100`.

2.7 Chaînes de caractères

Soit notre chaîne de caractère(string) str='Bonjour tout le monde'. On utilise des apostrophes pour les chaînes de caractères.

Dans octave on utilise aussi des guillemets mais cela signifie autre chose. En fait, cet usage permet de définir des caractères spéciaux :

`\t` pour le caractère <tab>

`\n` pour un saut à la ligne (<newline>) ; mais la chaîne reste cependant un vecteur ligne et non une matrice

`\"` pour le caractère "

`\'` pour le caractère '

`\\` pour le caractère \

Ex:

```
disp("Texte\ttabulé\net sur\t2 lignes")
```

Another example

```
for i=1:5                                                          1
    disp('Sampiyon Besiktas!!!')
    %disp command is used for strings. The default behavior is     3
    %to make a newline for each line
end                                                                5
for i=1:5
    fprintf('Sampiyon\t\\Besiktas!!!')                             7
    %fprintf is used for both strings and numbers.
    % newline will be made if explicitly demanded                  9
end
for i=1:5                                                          11
    fprintf('\"Sampiyon\" Besiktas!!!\n')
end                                                                13
for i=1:5
    fprintf('Sampiyon\\ Besiktas!!!\n')                            15
end
```

D'autres fonctions que nous allons utiliser sont:

`lower(string)` / `upper(string)` va convertir la chaîne string en minuscules/majuscules. Par ex. `lower('ABC)'` retourne abc.

`abs(string)` ou `double(string)` va convertir les caractères de la chaîne string en leurs codes décimaux selon la table ASCII ISO-Latin-1. Par ex. `abs('t')` retourne 116.

`char()` convertit les nombres de la variable var en caractères (selon table ASCII ISO-Latin-1). Par ex. `char(110)` retourne n.

`str2num(string)` convertit un string en valeur numérique. Par ex. `str2num('12,34;56,78')` retourne la matrice [12 34 ; 56 78].

Pour remplacer l'espace avec _ (ou un autre caractère) dans les chaînes de caractère la commande est `str(isspace(str))='_'` (ou autre caractère, voir les exemples).

```
str='Bonjour tout le monde'
str(isspace(str))='_'
str=str2='A Bcd'
str(isspace(str))=''
str2(isspace(str2))='*'
```

3 Variables

Une variable est la part de la mémoire qui a un nom que nous avons choisi librement. Par ex. x=5. On a la variable x qui a la valeur numérique 5. Il y a quelques reglès qu'il faut respecter:

- une variable peut être composée de n'importe quels caractère alphanumérique et le caractère _. Une exception à cette règle est que les mots clés utilisé par le programme –comme "for" par ex.– ne peuvent pas être utilisés. Pour voir la liste complete des mots mots clés la commande est `iskeyword`.
- le premier caractère d'une variable doit être une lettre et non un chiffre. Par ex. x15abc= 7 est bon alors que 15abc = 7 ne marche pas.
- il est déconseille d'utiliser _ comme le premier (et dernier) caractère...

3.1 Variables globales

http://www.mit.edu/people/abbe/matlab/globals.html

```
When you define a variable at the matlab prompt, it is defined inside
of matlab's "workspace." Running a script does not affect this, since
a script is just a collection of commands, and they're actually run from
the same workspace. If you define a variable in a script, it will stay
defined in the workspace. Functions, on the other hand, do not share
the same workspace. A function won't know what a variable is unless the
it gets the variable as an argument, or unless the variable is defined
as a variable that is shared by the function and the matlab workspace,
or a global variable.

To use a global variable, every place (function, script, or at the matlab
prompt) that needs to share that variable must have a line near the top
identifying it as a global variable, ie:
     global phi;
Then when the variable is assigned a value in one of those places, it will
have a value in all the places that begin with the global statement.

If you want more than one function to share a single copy of a variable, simply
declare the variable as global in all the functions. Do the same thing at the
command line if you want the base workspace to access the variable. The global
declaration must occur before the variable is actually used in a function...
```

Par exemple, dans la fonction suivante on a deux variables globales: A and B.

```
function h = myf(t)
global A B
h = A*t/2 + B^2;
```

Après dans "workspace" principale on pourrait avoir plusieurs valeurs de la fonction ci-dessus en seulement modifiant la valeurs des variables globales:

```
global A B
A = 12; B=4;
y = myf(5)
A = 6; B=2;
y = myf(5)
```

Then interactively enter the statements:

```
global x=10
function f ()
x = 1
end
x
f()
x
f()
```

Comme on voit ci-dessus, la valeur de x est 1 à l'intérieur de la fonction mais 10 à l'extérieur. C'est parce que les fonctions travaillent avec les copies des variables. Quand la valeur de cette copie change, la valeur de l'original n'est pas affectée.

Si l'on affecte une valeur à une variable globale lors de la création de la variable, on ne peut pas redéfinir la même variable comme une variable globale. Mais on peut le changer de manière ordinaire sans uiliser la commande "global". Par ex.

```
global x1=10
global x1=20
x1
x1=7
x1
```

Pour trouver si une variable est globale la commande est **isglobal(name)**

3.2 Variables *persistent*

mathworks.com:

```
Persistent variables differ from global variables in that persistent variables
are known only to the function in which they are declared. This prevents
persistent variables from being changed by other functions or from the
MATLAB command line.
```

```
global x = 1

    function f ()
    persistent x = 10;
```

```
      persistent x = 20; % it is said that  x should be 20 but it
          is not.
       % Since persistent variables are initialized only once as in
            C/C++.
      x=x+5;
      x
      end

      function f2 ()
        x = 3;
        x=x+4;
       x
      end

x
f ()
f2 ()
display("second calls")
x
f()
f2()
display("third calls x")
x
f()
f2()
```

3.3 Statut des variables

Pour voir le statut des variables on utilise la command who. Les options que nous pouvons ajouter sont

- all: tous les symboles
- builtins: les fonctions prédéfinies
- functions: les fonctions définies par l'utilisateur
- long: la liste détaillé des types et dimensions de tous les symboles.
- variables: les variables définies par l'utilisateur

La commande "whos" est l'équivalente de "who -long".

La commande "clear", par défaut, efface les variables. Il existe d'autres options:

- ?: remplace un caractère

- *: remplace plusieurs caractère.
- all: efface toutes les fonctions, variables, scripts, etc.

clear a?b efface les variables ayant seulement 3 caractères et commencant avec a et se terminant avec b.

clear a*b efface toutes les variables qui commencent avec a et se terminent avec b.

4 Structure de contrôle

Soit x= 11. la commande rem(x,n) où n est un nombre ingérieur à x va retourner le reste de la division de x par rapport à n.

4.1 Construction if-else-if-else...

```
x=11
if (rem(x,3)==0)                                                    2
    %disp('x is divisible by 3');% or equivalently
    fprintf('x is divisible by 3\n');                               4
end
```

```
x=11                                                                1
if (rem(x,3)==0)
    disp('x is divisible by 3');                                    3
else
    disp('x is not divisible by 3');                                5
end
```

$$f(x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } 0 \le x \le 1, \\ 2 - x & \text{if } 1 < x \le 2, \\ 0 & \text{if } x > 2. \end{cases}$$

```
if x >= 0 & x <= 1
   f = x;                                                           2
elseif x > 1 & x <= 2
   f = 2-x;                                                         4
else
   f = 0;                                                           6
end
```

Consider the following code segment:

```
if (x < 50)
   Grade = 'F'
elseif (x < 60)
   Grade = 'D'
elseif (x < 70)
   Grade = 'C'
elseif (x < 80)
   Grade = 'B'
else
   Grade = 'A'
end
```

First, if x is less than 50, 'F' is assigned to Grade. If x is greater than or equal to 50, the execution continue with the first ELSE IF where x < 60 is tested. If it is .TRUE., 'D' is assigned to Grade. Note that one can reach the test of x < 60 simply because the test x < 50 is .FALSE.. Therefore, when reaches x < 60, we are sure that x >= 50 must hold and as a result, Grade receives 'D' if x is greater than or equal to 50 and is less than 60.

4.2 Construction switch

```
x=11
switch (rem (x, 5))                                                    2
  case 0
    fprintf ('x is divisible by 5 \n');                                4
  case {1,2} % on peut regrouper plusieurs cas
    fprintf ('x is not divisible by 5 and the remainder is 1 or 2\     6
       n');
  case {3} % les elements singulier peuvent etre entres entre les
     accolades aussi
    fprintf ('x is not divisible by 5 and the remainder is 3\n');      8
  otherwise
    fprintf ('x is not divisible by 5 and the remainder is 4\n');      10
end
```

4.3 Boucle while

On sait que la suite suivante converge vers $\sqrt{3} = 1.7321$. Si'lon veut faire afficher chaque étape pendant l'itération vers ce cible:

$$x_{n+1} = \frac{x_n}{2} + \frac{3}{2x_n}$$

```
x = 1;
while x^2 <= 100
     disp(x)
     x = x+1;
end
```

wikipédia:

La suite de Fibonacci est l'une des suites mathématiques les plus connues. Elle doit son nom au mathématicien italien Leonardo Pisano, plus connu sous le pseudonyme de Fibonacci (1175 - 1250). Dans un problème récréatif posé dans un de ses ouvrages, le Liber Abaci, Fibonacci décrit la croissance d'une population de lapins :

Possédant initialement un couple de lapins, combien de couples obtient-on en douze mois si chaque couple engendre tous les mois un nouveau couple à compter du second mois de son existence ?

```
fib = ones (1, 12);
i = 3;
while (i <= 12)
    fib (i) = fib (i-1) + fib (i-2);
    i++;
end
```

```
% do until is available only in octave...
fib = ones (1, 12);
i = 2;
do  %A newline is not required but using one is better.
     i++;
     fib (i) = fib (i-1) + fib (i-2);
until (i == 12)
```

4.4 Boucle for

```
for i=1:10% basic example
disp('Sampiyon Besiktas!!!')
end
```

```
fib = ones (1, 10);
for i = 3:10
    fib (i) = fib (i-1) + fib (i-2);
end
```

Remarque: Pour des raisons de performance il est conseillé de vectoriser le plus possible. Par example, au lieu de

```
i=0;
for t=0:10
   i=i+1;
   y(i)=sin(t)
end
```

il est mieux de faire

```
t=0:10;
y=sin(t);
```

4.5 Break

```
% only inside for or while loops
num = 103;
div = 2;
while (div*div <= num)
  if (rem (num, div) == 0)
    break;
        % break finishes the program while exit finishes octave/
            matlab session.
  end
  div++;
end
if (rem (num, div) == 0)
    fprintf ('Smallest divisor of %d is %d\n', num, div)
else
    fprintf ('%d is prime\n', num);
end
```

```
% only inside for or while loops
for count=1:10
   if (count == 5)
     break;% if x=5 then we will break out of loop
   end
   fprintf ('%d ', count);
end
```

4.6 Continue

```
% only inside for or while loops
for count = 1: 10
    if (count == 5) % if count is 5,
       continue; % skip remaining code in loop
```

```
    end % end is before the command set...
    fprintf('%d', count); % the command that will be skipped by    6
        continue when if condition is respected.
end
```

4.7 D'autres commandes de contrôle

Construction try-catch :.... A completer....

return: Cette commande termine l'exécution d'une fonction ou d'un script.

remarque: Il ne faut pas sortir d'une boucle avec exit ou quit; parceque ceci terminerait aussi la session Matlab/Octave et pas seulement le script!

nargin: la commande nargin retourne le nombre d'arguments d'entrée passés à une fonction lors de l'appel à cette fonction.

Ex: mafonction(param1,param2) : nargin retourne 2, etc...

nargout: cette commande retourne le nombre de variables créées/affectées par la fonction.

Ex: mafonction(...) : nargout est 0; out1=mafonction(...) : nargout est 1; [out1 out2]=mafonction(...) : nargout est 2, etc...

mathworks.com::

```
Short-Circuit Operators

The following operators perform AND and OR operations on logical
expressions containing scalar  values. They are short-circuit
operators in that they evaluate their second operand only when the
result is not fully determined by the first operand.

The statement shown here performs an AND of two logical terms, A and B:
A && B
If A equals zero, then the entire expression will evaluate to logical 0
(false), regardless of the value of B. Under these circumstances, there
is no need to evaluate B because the result  is already known. In this
case, MATLAB short-circuits the statement by evaluating only the first
term.

A similar case is when you OR two terms and the first term is true.
Again, regardless of the value of B, the statement will evaluate to
true. There is no need to evaluate the second term, and MATLAB does
not do so.

You can use the short-circuit operators to evaluate an expression only
when certain conditions are satisfied. For example, you want to execute
an M-file function  only if the M-file resides on the current MATLAB path.
```

```
Short-circuiting keeps the following code from generating an error
when the file, myfun.m, cannot  be found:

comp = (exist('myfun.m') == 2) && (myfun(x) >= y)

In the following statement, it doesn't make sense to evaluate the
relation on the right if the divisor, b, is zero. The test on the left
is put in to avoid generating a warning under these circumstances:

x = (b ~= 0) && (a/b > 18.5)

By definition, if any operands of an AND expression are false, the
entire expression must be false.  So, if (b ~= 0) evaluates to false,
MATLAB assumes the entire expression to be false and terminates its
evaluation of the expression early. This avoids the warning that would
be generated if MATLAB were to evaluate the operand on the right.
```

5 Fonctions et les fichiers "script"s

Les "m-files" sont des fichiers qui contiennent des commandes MATLAB/Octave. Ils ont l'extension ".m". Il existe deux types de "m-files": les fonctions et les scripts (de commande). En effet les fonctions sont des scripts de commande particuliers qui acceptent des *inputs* et qui retournent des *outputs*. On peut avoir la définition d'une fonction dans un fichier de script (avec d'autres commandes en même temps), ou on peut réserver un "m-file" entier uniquement pour une fonction.

Pour exécuter un script appelé abc.m, la commande est:
`run('abc')`sous matlab
`run('abc.m')` / `source('abc.m')` sous octave

Pour utiliser une fonction $f(x)$, on fait $f(n)$ où n est le nombre d'intérêt. Pour exécuter un script il faut que le script soit dans le "*path*" de recherche. Supposons que nous avon un m-file apelé `abc.m`. Normalement, matlab/octave recherche ce dossier dans:

- 1. le *workspace* ; s'il n'a pas trouvé...
- 2. on vérifie si abc.m n'est pas une fonction built-in (définie au niveau du noyau MATLAB/Octave) ; s'il n'a pas trouvé...
- 3. dans le répertoire courant de l'utilisateur [...]
- 4. les différents répertoires définis dans le "path de recherche"
- 5. si rien n'est trouvé: Oops, erreur!

Pour connaitre le *path* de recherche la commande est `path`. Pour savoir où l'on trvaille la commande est `pwd`. Pour changer de répertoire la commande est `cd`.

```
path
addpath('c:\abc\mes trucs')% pour windows ou
addpath('/home/yourname/mes trucs')% pour GNU/Linux
cd 'c:\abc\mes trucs'

computer% returns the machine type of the computer
version% returns the version of running matlab/octave
ver% more detailed information about computer and version
```

5.1 Fonctions simples

Les fonctions sont des "m-files" acceptant des inputs et retournant des outputs. Les noms des variables suivent les mêmes règles que les noms des variables. Pour expliquer ce que c'est une fonction nous allons suivre un example concret.

```
function y = myfactor(x)
% function: keyword that is used to define a function
% y: output argument that will be used to send result
% myfactor: function name. it must be the same as with the file
    name.
% x: input argument
if (x < 0)
    fprintf('\nError: Oops! negative integer\n');
    break;
end

res=1;
for i = x:-1:1
    res=res*i;
end
y=res;
end%optional
```

Listing 1: Une fonction...

Il est obligatoire de donner le même nom au "m-file" et à la fonction quand il y une seule fonction dans m-file. S'il y en a plusieurs le nom de la première fonction doit être pareil que le nom du m-file.

Les variables créées à l'intérieur du m-file qui définit une fonction sont des variables "locales". Elles ne sont pas accessible en dehors de la fonction.

Le m-file définissant la fonction doit commencer par le mot clé `function`.

Pour voir les m-files/scripts il faut faire `what`. Pour afficher ce qu'il y a dans m-file "myfactor.m" la commande est `type myfactor`. Finalement, pour utiliser la nouvelle fonction pour, disons 5, il faut `myfactor(5)`.

Exo: Essayer `help myfactor`, qu'est-ce que vous voyez?

```
function discount
% discount: function name. it must be the same as with the file
    name.
% x: interest rate, N: years, P= asset value, PV: present value
% function call is like: discount
N=input("Vadeyi yil olarak giriniz:")
P=input("Vade sonundaki menkul kiymet tutarini giriniz:")
x=input("Yillik faiz oranini giriniz")
if (P < 0)
    fprintf('\nError: Oops! negative values are not permitted\n');
    break;
end
PV=P/(1.0+x)^N;
disp(['Bugunku deger= ' num2str(PV) ])
end
```

Listing 2: Une fonction sans argument qui demande les inputs et ne retourne rien

```
function y = discount2
% y: output argument that will be used to send result
% discount: function name. it must be the same as with the file
    name.
% x: interest rate, N: years, P= asset value, PV: present value
% function call is like: discount2
N=5;
P=100;
x=0.03;
if (P < 0)
    fprintf('\nError: Oops! negative values are not permitted\n');
    break;
end
PV=P/(1.0+x)^N;
y=PV;
fprintf('Bugunku deger= %f\n',PV)
end
```

Listing 3: Une fonction sans argument qui retourne une valeur

```
function y = discount3(P,N,x)
% y: output argument that will be used to send result
% discount: function name. it must be the same as with the file
    name.
```

```
% x: interest rate, N: years, P= asset value, PV: present value
% function call is like: discount3(5,100,0.12)
if (P < 0)
    fprintf('\nError: Oops! negative values are not permitted\n');
    break;
end
PV=P/(1.0+x)^N;
y=PV;
fprintf('Bugunku deger= %f\n',PV)
end
```

Listing 4: Une fonction à 3 arguments qui retourne une valeur

```
function [bd,gd] = discount4(P,N,x)
% [bd,gd]: output array that will be used to send result
% discount: function name. it must be the same as with the file
    name.
% x: interest rate, N: years, P= asset value, PV: present value
% function call is like: [bd,gd]=discount4(5,100,0.12)
if (P < 0)
    fprintf('\nError: Oops! negative values are not permitted\n');
    break;
end
bd=P/(1.0+x)^N;
gd=P/(1.0+x)^(N-1);
fprintf('Bugunku deger= %f ve gelecek yilki deger= %f\n',bd,gd)
end
```

Listing 5: Une fonction sans argument qui retourne deux valeurs

5.2 Fonctions avancée et scripts

5.2.1 Fonctions *inline*s

Des fois il est utile de définir des fonctions *inline*s

```
f = inline('sqrt(x.^2+y.^2)','x','y')
f(3,4)% retourne 5
```

Une commande utile qui peut être utilisée avec les fonctions *inline*s est `vectorize()`.

```
x='a*b'
vectorize(x)%retourne a.*b
f=inline('a^b','a','b')
vectorize(f)%retourne a.^b
```

5.2.2 Sousfonctions

On peut créer une fonction qui appelle d'autres fonctions..

```
function f () % necessary to make this m-file a function
        % functions will be loaded and executed...
  fprintf ("in f, calling g\n");
  g ()
end
function g () % subfunction
  fprintf ("in g, calling h\n");
  h ()
end
function h () % subfunction
  fprintf ("in h\n")
end
```

Listing 6: f.m

Quand l'on a plusieurs définitions de fonctions dans le même m-file de fonction, seulement la première fonction (la fonction primaire) sera reconnue. Par exemple, dans `f.m`, on a trois fonctions. La commande `f()`, sera reconnue alors que `g()` et `h()`non!

```
disp('aaa'); % necessary to make this m-file a script
% the functions will be loaded but not executed
% see f.m
function f () % function
  fprintf ("in f, calling g\n");
  g ()
end
function g () % function
  fprintf ("in g, calling h\n");
  h ()
end
function h () % function
  fprintf ("in h\n")
end
```

Listing 7: f2.m: Un script...

5.2.3 Fonctions pointeurs

Normalement, les sousfonctions sont accédées seulement par la fonction principale et les autres sousfonctions. Si l'on veut qu'une sousfonction soit acédée par des fonctions définies dans d'autres "M-files" on doit créer des fonctions pointeurs (*function handles*).

mathworks.com:

```
By design, only functions within an M-file are permitted to access
subfunctions defined within that file. However, if, in this same M-file,
you were to construct a function handle for one of the internal
subfunctions, and then pass that handle to a variable that exists outside
of the M-file, access to that subfunction would be essentially
unlimited... ...A function handle is a callable association to a MATLAB
(and GNU octave) function. It contains an association to that
function that enables you to invoke the function regardless of where you
call it from. This means that, even if you are outside the normal scope of
a function, you can still call it if you use its handle.
```

Remarque: Notez qu'une fonction peut toujours accéder à une fonction dans le "path" de matlab. Par exemple, si l'on a les deux fonctions suivantes

```
function y=myfun1(x) % myfun1.m
y=x+5;
end

function y=myfun1(x) % myfun2.m
y=2*x;
end
```

`myfun2` peut accéder à `myfun1` par `myfun2(myfun1(5))` où le résultat est 20. Avec des fonctions pointeurs on peut accéder aussi aux sousfonctions!!!

Il ya des opérateurs fonctionnels (*functional operators*) qui acceptent seulement des fonctions pointeurs comme input. Une liste incomplete est

```
fplot, fminbnd, fminsearch, quad, quadv, dblquad
```

Par example, si l'on veut intégrer une fonction donnée, disons $f(x) = 1/(x^3 - 2 * x - 5)$, de 0 à 2, on aura besoin des fonctions pointeurs. La commande qui calcule une approximation de l'intégrale ci-dessus est

```
function y = myfun(x)% myfun.m
y = 1./(x.^3-2*x-5);
end
Q = quad(@myfun,0,2)
```

On pourrait définir aussi des fonctions anonymes sans être obligé d'utiliser des m-files grâce aux fonctions pointeurs.

```
kare = @(x) x.^2;
kare(5) %retourne 25
kare2 = @(x,y) x.^2.*y.^2;
kare2(5,2)%retourne 100
```

L'example ci-dessus avec quad pourrait s'écrire

```
myf=@(x) 1./(x.^3-2*x-5);
Q = quad(myf,0,2)
```

La raison de cette écriture est que maintenant `myf` est déjà une fonction pointeurs.

On a déjà précisé que quand il y a plusieurs sousfonctions dans le même m-file, seulement la première fonction (la fonction primaire) sera reconnue. Pour que les autres sousfonctions (ou des fonctions définies dans les fichier scripts) soient reconnues par d'autres fonctions, il faut utiliser soit *function handle*, soit la commande `feval()`. Pour un example sur feval() voir les encadrés (**??**) et (**??**) dans la section "Résolution et optmisation". Maintenant, on va voir un example sur les fonctions pointeurs.

Soit une fonction appelé `myrooth` qui accepte une fonction et un vecteur comme arguments, et qui fait:

```
function y=myrooth(fh,x) % 1 inputs
  a=sqrt(fh(x)); % a is assigned to squared fh(x)                    2
  y=a/sum(x);% but we don't know what is returned...
                    %because we do not know what fh is...           4
end
```

Listing 8: myrooth.m

Normalement, seulement les fonctions qui sont accessiple par matlab/octave seront reconnue par `myrooth` . par exemple, une fonction peut appeler seulement les fonctions dans le *path* de matlab/octave et une sousfonction, ou une fonction prédéfinie (*built-in function*) comme `log`,`sin`, etc.

Dans le script `denehandle.m` la fonction `mysq()` sera reconnue. Mais si l'on veut que cette fonction sera reconnue par `myrooth.m`, nous devrons créer un pointeur de cette fonction (a *function handle*).

```
% an example to show function handles...                           1
x=[3; 4];
function y=mysq(x)                                                  3
  z1=x(1)*x(1)+x(2)*x(2);
        z2=myx(x);%we can call a subfunction in a function.         5
        y=z1+z2;
end                                                                 7
function y=myx(x)
  y=x(1)*x(2);                                                      9
end
mysq(x) % returns 37                                                11
h = @mysq;% function handle for mysq
myrooth(h,x) % it works because h is a function handle.             13
myrooth(@myx,x) %it works because @myx is a function handle.
% but myrooth(myx,x) --and also myrooth(mysq,x)-- returns an error  15
```

```
% because myx and mysq are not subfunction for myrooth!
```

Listing 9: denehandle.m

La raison est que `mysq()` est inclue dans le **script** `denehandle.m` qui n'est pas un m-file de fonction.

Remarque: On aurait pu réaliser cette tâche avec le script suivant

```
function y=myroot(x) % can be called like myroot(x[3;4])
  a=fh(x); % a is assigned to fh(x)                              2
  y=sqrt(a)/sum(x);% we don't know what is returned...
end                                                              4
function z=fh(x)
  z=x(1)*x(1)+x(2)*x(2);                                         6
end
```

Listing 10: myroot.m

6 Entrée - sortie

On peut entrer les chiffres et les textes à partir du clavier de manière intéractive ou bien à partir d'un fichier. De même manière, on peut demander à un programme d'afficher les résultat immediatement sur l'écran ou sur un fichier que nous pouvons travailler ensuite. C'est le premier classement concernant les fonctions entrée - sortie. Ensuite on peut faire un dexième classement d'entrée - sortie par rapport au type, plus particulièrement au "format" des entrées et des sorties. Puisque chaque variable a un mode associé (vecteur, matrice, chaîne de caractère...) il est possible d'informer matlab/octave du mode des variables en question pour lecture/écriture de ces variables sur (ou bien à partir) d'un fichier/du clavier.

6.1 Entrée - sortie intéractive

6.1.1 Lecture/écriture non-formatée

display(x) va afficher x sur l'écran et passer à une nouvelle ligne.

Pour entrer un texte ou un chiffre à partir du terminal il faut utiliser la commande input(). Par ex. input('Entrer votre nom! ').

Les exemples sont:

```
nom = input('Entrer votre nom: ', 's');%to enter a string
age = input('Entrer votre age: ');%to enter a number
fprintf('Votre nom est: %s et votre age est: %d\n',nom,age);
```

6.1.2 Lecture/écriture formatée

Pour écrire le résultat sur un fichier ou bien pour lire les données à partir d'un fichier on utilise des "formats" prédéfinis. Il s'agit d'un ensemble de commandes qui controle comment matlab/octave lit et écrit sur les fichiers. Par exemple, combien de chiffres on veut mettre après la virgule. Ces "formats" de matlab/octave se sont basés sur des spécifications de formats du langage C.

`%u` et `%nu`: entier positif et entier positif ayant au moins n caractères de longueur.

`%d/%i` et `%nd/%ni`: même chose avec entier positif ou négatif....

`%f` et `%n.mf`: nombre réel et nombre réel ayant n caractères avant virgule, m caractères après virgule

`%e` et `%n.me`: même chose avec notation scientifique.

`%g` et `%ng`: même chose avec notation scientifique compacte.

`%s` et `%ns`: chaîne de caractères

Les commandes utiles sont `fprintf(),fscanf(),sscanf()`

```
x=1:4
                                                                    2
y=linspace(1,2,4)
                                                                    4
for i=1:4
    fprintf('\n%u %f\n', x(i),x(i));                                6
end
for i=1:4                                                           8
    fprintf('\n%u %f\n', y(i),y(i));
end                                                                 10
```

On voit que matlab/octave affiche un résultat non-désiré (pour ne pas dire faux). x étant un entier, l'affichage est bon; mais y n'est pas affiché correctement en "format" entier...

Il n'existe pas de commande pour lecture formatée à partir du clavier pour matlab. Pour octave nous avons la commande scanf(). Une première solution, pour matlab, est de créer une chaîne de caractère et après lire à partir de cette chaîne de caractère. Pour entrer un texte formaté à partir du clavier on peut utiliser la commande

```
res=input('','s')
```

Supposons que vous avez entré le nombres 1 2 3 4. Pour récupérer les nombres et les stocker dans une matrice B la commande nécessaire est

```
B = sscanf(res,'%f %f %f %f')
```

Comme on l'a déjà vu pour lecture de données à partir d'une chaîne de caractère la commande est sscanf(). L'exemple suivant crée un vecteur A contenant les valeurs 2.7183 et 3.1416:

```
s = '2.7183  3.1416';
A = sscanf(s,'%f')
```

Remarque[4]**:** *Avec la fonction sscanf() la lecture s'effectue en "format libre" en ce sens que sont considérés, comme séparateurs d'éléments dans la chaîne, un ou plusieurs <espace> ou <tab>. Si la chaîne renferme davantage d'éléments qu'il n'y a de "spécifications de conversion" dans le format, le format sera "réutilisé" autant de fois que nécessaire pour lire toute la chaîne.* ***Si, dans le format, on mélange des spécifications de conversion numériques et de caractères, il en résulte une variable de sortie (vecteur ou matrice) entièrement numérique dans laquelle les caractères des chaînes d'entrée sont stockés, à raison d'un caractère par élément de vecteur/matrice, sous forme de leur code ASCII...***

On a le choix entre choisir un vecteur ou une matrice pendant la lecture avec sscanf.

```
v=sscanf('1 2 3 4 5 6', '%f', 4)
[m,c]=sscanf('1 2 3 4 5 6', '%f', [3,2])
```

Dans la code ci-dessus v est un vecteur ayant 4 réels; m est une matrice composée des réels de dimension $(3, 2)$, i.e. 3 lignes et 2 colonnes; et finalement c est le nombre total des éléments lu par la fonction sscanf().

Remarque: Sous octave on a une autre version de sscanf() de forme

```
[var1, var2, var3 ...] = sscanf(string, format, 'C')
```

qui est beaucoup plus souple:

```
[s1,n1,n2,s2]=sscanf('abcde 12.34 45.3e14 fgh', '%s %f %f %s', 'C')
[s1,s2,n1,n2]=sscanf('abcde 12.34 45.3e14 fgh', '%3c %s %f %f', 'C')
[s,n1,n2]=sscanf('abcde 12.34 45.3e14 fgh', '%5c %f %f', 'C')
[s,n1,n2]=sscanf('abcde 12.34 45.3e14 fgh', '%3c %*s %f %f', 'C')
```

6.2 Entrée - sortie de fichiers

Pour lecture ou écriture nous avons besoin d'attacher le fichier à matlab/octave. Pour créer un fichier s'appelant yaz.txt les commandes nécessaires sont:

```
fid = fopen('yaz.txt','w');
.....
fclose(fid);
```

[4] from http://enacit1.epfl.ch/cours_matlab/

Pour lire à partir d'un fichier s'appelant oku.txt les commandes nécessaires sont:

```
fid = fopen('oku.txt','r');
.....
fclose(fid);
```

6.2.1 Lecture/écriture non-formatée de fichiers

```
fid = fopen('yaz.txt','w');
fputs (fid, 'Yer: Besiktas, tarih: 1903');
fclose(fid);

fid = fopen('yaz.txt','r');
text = fgetl (fid)
fclose(fid);
```

text sera une chaîne de caratère.

6.2.2 Lecture/écriture formatée de fichiers

Nous aurons à traiter les données numériques en économie. Pour lire ce type de données les commandes le plus utilisées sont csvread, dlmread, load,

Des fois les données numériques contiennent des chaînes de caractère (noms des variables, etiquettes, etc.). On dit que ce type de données est mixte: il y a à la fois des valeurs numériques et des chaînes de caractère. Pour ce type de donnée on a: textread, fscanf.

Soit notre fichier ayant seulement les valeurs numérique

```
1  5  4  16  8                                                    1
5  43  2  6  8
6  8  4  32  1                                                    3
```

Listing 11: datanum.txt

Pour lire le contenu de ce fichier et le nommer matrice A, les commandes sont

```
A=load('datanum.txt')
```

La commande load() suppose que les données sont délimitées par un espace. Si, au lieu de l'espace, le délimiteur est un point-virgule ';' ou une virgule ',' alors on devra utiliser dlmread() en précisant le délimiteur approprié. Quand le délimiteur est une virgule on peut aussi utiliser

```
90;7;8;7;6
5; 9; 81; 2; 3
```

Listing 12: datanum2.txt

```
A=dlmread('datanum2.txt',';')
```

```
90,7,8,7,6
5, 9, 81, 2, 3
```

Listing 13: datanum3.txt

```
A=dlmread('datanum3.txt',',')
A=csvread('datanum3.txt')
```

Soit

```
90,7,8,7,6,
5, 9, 81, 2, 3
```

Listing 14: datanum4.txt

```
A=csvread('datanum4.txt')
```

Qu'est-ce qui est faux?

Un exemple illustratif pour load():

Soit notre base de données originale:

```
year   A      B
1989   200  15
1990   220  18
1991   230  17
```

Nous allons la modifier de façon à supprimer les noms des colonnes

```
1989   200  15
1990   220  18
1991   230  17
```

Le fichier script...

```
X=load('veri2.dat');
year=X(:,1);
AA=X(:,2);
BB=X(:,3);
```

```
CC = AA .* BB;  %%%% attention to .*

disp('  You can put here what you want')
disp('  ------------------------------')
format = 'for %5u total cost is $%8.3f\n' ;

for no=1:1:length(year)
  fprintf(format, year(no), CC(no));
end
```

Listing 15: veriio.m

***Un autre exemple*:**

```
x=1:5% row vector
y=linspace(1,2,5) % row vector
function son=f(x,y)
  son=x.*y.^2;  %row vector
end
z=ones(1,5) %row vector
fid = fopen ('io2a.out', 'w');
fprintf(fid, 'x         y              z\n')
for i=1:5
    z(i)=f(x(i),y(i)); %row vector
    fprintf(fid, '%u\t %f \t %f\n', x(i),y(i),z(i));
end
fclose(fid);
% use vectorization!!!!
fid = fopen ('io2b.out', 'w');
fprintf(fid, 'x         y              z\n')
z=f(x,y)% all vectors are row vector!!!
fprintf(fid, '%u\t %f \t %f\n', x,y,z);
fclose(fid);

% usage of dlmwrite
% first, concatenate vectors into a matrix by transposition
A=[x' y' z'] % or A=[x', y', z']
dlmwrite('io2c.out',A,',', 'precision', '%.2f')
% usage of csvwrite
csvwrite('io2d.out',A)% precision option is not permitted
```

Listing 16: io2.m

Attention: fid, fprintf(), fclose()...

Nous avons le résultat suivant pour à la fois io2a.out et io2b.out

```
x         y              z
1   1.000000      1.000000
2   1.250000      3.125000
3   1.500000      6.750000
```

```
4   1.750000        12.250000
5   2.000000        20.000000
```

Listing 17: io2a/b.out

```
1.00,1.00,1.00
2.00,1.25,3.12
3.00,1.50,6.75
4.00,1.75,12.25
5.00,2.00,20.00
```

Listing 18: io2c.out

```
1,1,1
2,1.25,3.125
3,1.5,6.75
4,1.75,12.25
5,2,20
```

Listing 19: io2d.out

Remarque: `disp(sprintf())` est équivalent à `fprintf()`.

Pour les fichier de données ayant à la fois les chaînes de caractères et les valeurs numériques une autre possibilité est d'utiliser `fscanf()`.

```
abc 1 2
def 3 4
ghi 5 6
jkl 7 8
```

```
file_id = fopen('dene1.txt', 'rt') ;
no = 1 ;
while ~ feof(file_id)
 ad{no,1}    = fscanf(file_id ,'%s',1) ;
 fiyat(no,1) = fscanf(file_id ,'%u',1) ;
 miktar(no,1)  = fscanf(file_id ,'%u',1) ;
  no = no + 1 ;
end
status = fclose(file_id) ;
```

L'exemple suivant vient de "Help-octave mailing list" et il ne va pas marcher probablement sous Matlab. Supposons que nous avons un fichier temp.dat qui contient

```
time temperature1 temperature2
16:15:07 15,3 32,4
16:15:17 15,6 45,4
16:15:27 17,5 56,6
```

Comment peut-on lire ce fichier en tenant en compte les virgules utilisées comme séparateur décimal. La solution proposée par A. Richardson est from https://www-old.cae.wisc.edu/pipermail/help-octave/2009-September/016239.html

```
fid = fopen("temp.dat","r");
%Read header
h = fgetl(fid);
index = 1;
t1 = [];
t2 = [];
time = ["          "];
while(1)
  [t n1 n2 n3 n4 count] = fscanf(fid,"%s %d,%d %d,%d" ,"C");
  time(index,:) = t;
  t1(index) = [n1+n2/10];
  t2(index) = [n3+n4/10];
  index++;
  if (feof(fid)) break; endif
endwhile
fclose(fid);
t1
t2
time
```

Listing 20: un exemple difficile

7 Graphiques

Il existe différentes commandes pour faire des graphiques

7.1 plot 2D

Si x est un vecteur (ou une matrice) la commande `plot(x)` retourne le graphique du vecteur x (ou des colonnes de la matrice x) en ordonnée où l'indice de ce vecteur est en abcisse. En revanche, la commande `plot(t,x)` crée le graphique de x vs. t où t est en abcisse. On peut aussi tracer le graphique des pairs de t, x de façon suivante: `plot(t1,x1,t2,x2,3,x3)`. Etant donné

```
t = 0:0.025:6;
y = sin(t);
y1=sin(t-0.25);
y2=sin(t-0.5);
```

Comparer:

```
plot(y)
grid on
```

```
plot(t,y)
grid on
```

et/ou

```
plot(t,y,t,y1,t,y2)
grid on
```

On pourrait spécifier différents symboles et couleurs pour les lignes...

```
plot(t,y,'*',t,y1,'.',t,y2,'-')
```

ou encore on peut spécifier les couleurs

```
plot(t,y,'*g',t,y1,'.k',t,y2,'-m')
```

La liste complète des couleurs et des types de ligne est:

```
*********color specifier***
r    red
g    green
b    blue (default)???
c    cyan
m    magenta
y    yellow
k    black
w    white
*********line styles*******
-    solid line (default)
--   dashed line
:    dotted line
-.   dash-dot line
none no line
*********marker types******
+    plus sign
o    circle
*    asterisk
.    point
x    cross
none no marker (default)
```

Avec la commande `hold on` on peut tracer plusieurs lignes sur le même graphique.

```
t = 0:0.25:6;
plot(t,sin(t),'-.r*')
hold on
plot(t,sin(t-pi/2),'--mo')
plot(t,sin(t-pi),':bs')
hold off
```

from: http://enacit1.epfl.ch/cours_matlab/:

```
x=log(1:0.5:12);
y=sin(x);
bar(x,y);
axis([0 2.6 0 1]);
grid('on');
```

L'example ci-dessous montre comment créer la légende des variables, le titre du graphique et le nom des axes. from: http://enacit1.epfl.ch/cours_matlab/:

```
x=0:0.1:10*pi;
y1=sin(x); y2=sqrt(x);
plot(x,y1,x,y2);
axis([0 30 -6 6]); % 0<x<30 et -6<y<6
title('Titre est ici!!!');
xlabel('temps'); ylabel('Y=une fonc. de (X)');
legend('sinus(x)','racine(x)',4);
```

La position de la légende peut être changée selon: 1= angle haut/droite, 2= haut/gauche, 3= bas/gauche, 4=bas/droite, -1= en dehors à droite de la zone graphée.

L'example ci-dessous montre les subplots. from: http://enacit1.epfl.ch/cours_matlab/:

```
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');
```

`clf`: clear figure

Un graphique utile:

```
x = (1:40)/10;
g = (x-1).*(x-2).*(x-3);
izero = find(g==0);
ii = find(g~=0);
f(izero) = NaN;
f(ii) = 1./g(ii);
plot(x,f)X
```

7.2 plot 3D

La commande `plot3(X,Y,Z)` va relier chaque élément de X, Y, Z sur un graphique où X, Y, Z sont des vecteurs:

```
t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t)
axis square; grid on
```

from: http://enacit1.epfl.ch/cours_matlab/:

Supposons que nous avons des vecteurs x, y tels que $0 \leq x \leq 10$ et $2 \leq y \leq 5.5$. Et que nous voulons obtenir le graphique de $z = sin(x/3) * cos(y/3)$. Ici, on voudrait tracer le graphique de z pour chaque valeur de x et chaque valeur de y. On a besoin d'une grille des valeurs de x et y.

Pour représenter graphiquement cette surface, il s'agit au préalable de calculer une matrice z dont les éléments sont les "altitudes" z correspondant aux points de la grille. Cette matrice aura donc (dans le cas du présent exemple) la dimension 7 x 11 (respectivement length(y) lignes, et length(x) colonnes).

```
    x  0 1 2 3 4 5 6 7 8 9 10
y     ------------------------
2    |                        |
2.5  |                        |
3    |       matrice  z       |
3.5  |        (7 x 11)        |
4    |                        |
4.5  |                        |
5    |                        |
      ------------------------
```

```
x=0:1:10;  y=2:0.5:5;  % domaine des valeurs de la grille en X et Y
for k=1:length(x)      % parcours de la grille, colonne après colonne
  for l=1:length(y)    % parcours de la grille, ligne après ligne
    z1(l,k)= sin(x(k)/3)*cos(y(l)/3); % calcul de z, élément par élément
  end
end
mesh(x,y,z1);  %ou plot
% Essayer aussi surf(x,y,z1);
% Essayer aussi meshc(x,y,z1);

x=0:1:10;  y=2:0.5:5;     % domaine des valeurs de la grille en X et Y
[Xm,Ym]=meshgrid(x,y);
z2=sin(Xm/3).*cos(Ym/3); % calcul de z en une seule instruction vectorisée
     % notez bien que l'on fait produit .* (élém. par élém.) et non pas * (vectoriel)
mesh(x,y,z1);
% Essayer aussi surf(x,y,z1);
% Essayer aussi meshc(x,y,z1);
```

```
    -------------(7x11)---------------
   | 0  1  2  3  4  5  6  7  8  9  10 |
   | 0  1  2  3  4  5  6  7  8  9  10 |
   | 0  1  2  3  4  5  6  7  8  9  10 |
Xm=| 0  1  2  3  4  5  6  7  8  9  10 |
   | 0  1  2  3  4  5  6  7  8  9  10 |
   | 0  1  2  3  4  5  6  7  8  9  10 |
   | 0  1  2  3  4  5  6  7  8  9  10 |
    ---------------------------------

    ------------------------(7x11)-------------------------
   | 2    2    2    2    2    2    2    2    2    2    2   |
   | 2.5  2.5  2.5  2.5  2.5  2.5  2.5  2.5  2.5  2.5  2.5 |
   | 3    3    3    3    3    3    3    3    3    3    3   |
Ym=| 3.5  3.5  3.5  3.5  3.5  3.5  3.5  3.5  3.5  3.5  3.5 |
   | 4    4    4    4    4    4    4    4    4    4    4   |
   | 4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5 |
   | 5    5    5    5    5    5    5    5    5    5    5   |
    ------------------------------------------------------
```

Il est aussi possible de passer seulement un vecteur à la commande `meshgrid()`:

```
x = -2:0.2:2;
[X,Y] = meshgrid(x);
% ou bien on peut utiliser
%y=x;  [X,Y]=meshgrid(x,y);
Z = 100*sin(X).*sin(Y).* ...
    exp(-X.^2 + X.*Y - Y.^2);
surf(X,Y,Z)
```

Il est possible d'obtenir des champs de vecteurs:

```
x = -2:0.2:2;
[X,Y] = meshgrid(x);
Z = 100*sin(X).*sin(Y).*exp(-X.^2 + X.*Y - Y.^2);
[dx,dy]=gradient(Z,0.2,0.2);
% calcul champ vecteurs(dZ/dx, dZ/dy) pour dx=dy=0.2
quiver(X,Y,dx,dy,1.5)% affichage champ vecteurs
```

```
%source('pl1.m') works but run('pl1') does not...                    1
function Y = myf(x) %Create an m-file named myfun,...
Y(:,1) = sin(x(:))./x(:);%....that returns...                        3
Y(:,2) = x(:).^0.2;% a two-column matrix
end                                                                  5
fh = @myf;%Create a function handle pointing to myfun
fplot(fh,[0.1 10])%Plot. It does not work for an interval that       7
              %goes from - to +. Because for 0 it yileds a NaN!
```

8 Evaluation

```
eval ('a = 3')% will assign 3 to a
a*a% returns 9
y=feval('log',10)% will assign 2.30 to y
```

Cette commande est utile quand on ne connait pas le nom d'une fonction (ou variable) jusqu'au moment de l'execution.

Considérons la fonction suivante:

$$m(x) = 2\sin^2(x) + 3\sin(x) - 1$$

En fait, cette fonction est la combinaison de deux fonctions: $f(x) = 2x^2+3x-1$ et $g(x) = \sin(x)$ telle que l'on a $m(x) = f(g(x))$.

On pourrait définir $m(x)$ de par "m-file" `fev.m`:

```
function y=fev(fnc,x)
  z=feval(fnc,x)
  y=2.*z.^2+3*z-1
end
```

Pour l'utiliser il suffit de faire `f('sin',pi/2)` ou `f(@sin,pi/2)`. Évidemment, on pourrait remplacer la fonction `sin` par n'importe quelle fonction que l'on voudrait.

9 Equations linéaires

$$Ax = b \Rightarrow \begin{pmatrix} 3 & 1 \\ 4 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 9 \end{pmatrix}$$

où la solution est $x_1 = 2, x_2 = -1$.

```
A=[3, 1; 4,-1];
b=[5;9];
x=A\b
```

10 Equations nonlinéaires

10.1 Résolution des équations et les racines des polynômes

La commande `fzero()` résoud une équation nonlinéaire à une variable, i.e. trouve une zéro de l'équation en question. Il faut qu'il y ait une seule variable indépendente, i.e. $f(x)$. Il faut aussi donner une valeur initiale pour la variable indépendante comme une solution approximative qui n'est pas trop loin de la solution exacte.

```
f = @(x)x.^5-2*x-4;
z = fzero(f,2)% a first guess for solution...
```

Pour passer plusieurs paramètres en même temps une solution possible est d'utiliser les fonctions pointeurs comme dans l'exemple suivant.

```
% the purpose is to show how to pass aditional parameters to
% a function f(x), such that we pass f(x,a,b)...
b = 2;
c = 3.5;
x = fzero(@(x) x^3 + b*x + c,0)
% and now if we want to solve for different a, b
b = 4;
c = -1;
x = fzero(@(x) x^3 + b*x + c,0)
```

Listing 21: Passer plusieurs paramètres à une fonction

Le pôlynome $0.5x^3 - 11x^2 - 9x + 100$ suivant peût être représenté et résolu de façon suivante;

```
p = [0.5 -11 -9 100]
r = roots(p)
```

10.2 Système d'équations nonlinéaires

Pour résoudre un système d'équations nonlinéaires on utilise la commande `fsolve()`. Nous allons suivre l'exemple dans le manuel de GNU Octave.

$$6 = -2x^2 + 3xy + 4\sin(y)$$
$$-4 = 3x^2 - 2xy^2 + 3\cos(x)$$

```
function nonlin1 % example for system of nonlinear eq.
%[x, info] = fsolve ('fnl', [1; 2])%works in octave; not in matlab
[x, info] = fsolve (@fnl, [1; 2])%works in both
% an initial guess,[1;2], in column vector notation is required.
% x is the solution vector
% info = 1 indicates that the solution has converged
end
function y = fnl(x)
z=x(1); % z yi x(1) olarak tanimlayabiliriz...
w=x(2); % w yi x(2) olarak tanimlayabiliriz...
    y(1) = -2*z^2 + 3*z*w   + 4*sin(w) - 6;
    y(2) =  3*z^2 - 2*z*w^2 + 3*cos(z) + 4;
% variables are stored in vectors and equations are
% rewritten such that right hand side is zero, i;e. f(x)=0
end
```

Listing 22: Solution d'un système d'équations nonlinéaire: nonlin1.m

Il existe une différence importante entre Matlab and Octave. Pour Matlab chaque fonction doit être définie dans un M-file séparé, tandis que pour Octave on pourrait définir les fonctions même dans les "scripts". Analysons l'exemple suivant:

```
1;%necessary to make this m-file a script!
function y = fnl(x)
z=x(1); % z yi x(1) olarak tanimlayabiliriz...
w=x(2); % w yi x(2) olarak tanimlayabiliriz...
    y(1) = -2*z^2 + 3*z*w   + 4*sin(w) - 6;
    y(2) =  3*z^2 - 2*z*w^2 + 3*cos(z) + 4;
% variables are stored in vectors and equations are
% rewritten such that right hand side is zero, i;e. f(x)=0
end
[x, info] = fsolve ('fnl', [1; 2])%works in octave; not in matlab!
% or equivalently we can use function handle property as in
%[x, info] = fsolve (@fnl, [1; 2])%works in octave; not in matlab!
% in matlab we get: ??? Error: File: nonlin2.m Line: 2 Column: 1
% Function definitions aren't permitted at the prompt or in
    scripts.
% They must be in separate "m-files". One short-cut is defining
% the script file as a function. See nonlin1.m function!!!
```

Listing 23: Fonction définie dans un script: nonlin2.m

11 Optimisation linéaire

Le problème standard de l'optimisation (programmation) linéaire de peut s'écrire où x, c, b sont des vecteurs et A est une matrice

$$\begin{aligned} \max L(x) &= c^t x \\ s.c \quad Ax &\leq b \\ lb &\leq x \leq ub \end{aligned}$$

Pour les problèmes de minimisation il faut réécrire le même problème comme maximiser $-L(x)$. Sous octave on ne réécrit pas, on précise si le problème est un problème de minimisation ou maximisation. Soit:

$$c = \begin{pmatrix} 10 \\ 6 \\ 4 \end{pmatrix} \quad b = \begin{pmatrix} 100 \\ 600 \\ 300 \end{pmatrix} \quad A = \begin{pmatrix} 1 & 1 & 1 \\ 10 & 4 & 5 \\ 2 & 2 & 6 \end{pmatrix}$$

```
% linear programming example with glpk (octave only)...
c = [10, 6, 4]'; % a column vector
A = [ 1, 1, 1; % a matrix
      10, 4, 5;
       2, 2, 6];
b = [100, 600, 300]'; % a column vector
lb = [0, 0, 0]';% lower bound,column vector
ub = [];% upper bound, column vector
ctype = 'UUU';
%F: Free (the constraint is ignored).
%U: Variable with upper bound, A(i,:)*x <= b(i)
%S: Fixed Variable, A(i,:)*x = b(i)
%L: Variable with lower bound, A(i,:)*x >= b(i)
%D: Double-bounded variable, A(i,:)*x>=-b(i) and (A(i,:)*x<=b(i)
vartype = 'CCC';
% C: Continuous variable; I: Integer variable
s = -1;
%If s=1/s=-1 the problem is a minimization/maximization.
%The default value is 1 (minimization)
param.msglev = 1;
%0: No output; 1: Error messages; 2: Normal output; 3: Full output
param.itlim = 100;
% iterations limit; Negative value means no iterations limit.
[xmin, fmin, status, extra] = ...
     glpk (c, A, b, lb, ub, ctype, vartype, s, param);
xmin % the solution!!!
```

Listing 24: Optimisation linéaire

12 Optimisation nonlinéaire

La fonction `fminbnd()` sert à trouver le minimum d'une fonction à une variable.

```
1;%fminbnd exo..

function y=myf2(x)
y=x^2+4;
end
z = fminbnd(@myf2,-5,5);%search for min between [-5,5]
z
```

Listing 25: Minimisation à une variable

Souvent l'optimisation est multidimensionnelle et il existes des contraintes d'égalité et d'inégalité. L'exemple suivant montre ce cas plus général.

```
% successive quadratic programming example
% min_x phi (x); s.t.  g(x)=0, h(x)>=0;
% the objective function is of the form: y=phi(x)
% gradient and hession functions (if supplied) have
% the form g= gradient(x), h=hessian(x)
% If the problem does not have equality (or inequality)
% constraints, an empty matrix for cef (or cif) is used
% x: a vector
function r=g(x) % constraint
  r=[sumsq(x)-10; x(2)*x(3)-5*x(4)*x(5); x(1)^3+x(2)^3+1];
end

function obj=phi(x) % objectif to be maximized/minimized
   obj=exp(prod(x)) - 0.5*(x(1)^3+x(2)^3+1)^2;
end

x0=[-1.8; 1.7; 1.9; -0.8; -0.8];
[x, obj, info, iter, nf, lambda] = sqp (x0, @phi, @g, [])
```

Listing 26: Optimisation...

References

[1] http://www.gnu.org/software/octave/

[2] http://enacit1.epfl.ch/cours_matlab/

[3] https://staff.hti.bfh.ch/sha1/

13 Exercices

13.1 IO

Recréer des figures suivantes:
1-)

```
*
**
***
****
*****
******
```

```
%source('ioex1.m'), run('ioex1') and run('ioex1.m') work...
n=6;
for i=1:n
  for j=1:i
    fprintf('*');
  end
  fprintf('\n');
end
```

Listing 27: Exo-1

2-)

```
*
**
***
****
*****
******
*****
****
***
**
*
```

```
%source('ioex2.m'), run('ioex2') work...
n=6;
for i=1:n
  for j=1:i
    fprintf('*');
  end
  fprintf('\n');
end
for i=n-1:-1:1
  for j=1:i
    fprintf('*');
  end
  fprintf('\n');
end
```

Listing 28: Exo-2

3-)

```
      *
```

```
     ***
    *****
   *******
  *********
***********
```

```
n=6;
for i=1:n
  for j=1:n-i
    fprintf(' ');
  end
  for j=1:2*i-1
    fprintf('*');
  end
  fprintf('\n');
end
```

Listing 29: Exo-3