ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Analyse Numérique

Sémestre d'Autumne 2019

Prof. Fabio Nobile

2019

Introduction à Matlab® /Octave

1 Informations générales

Matlab® (MATrix LABoratory) est un logiciel commercial pour le calcul scientifique, qui comprend un langage de programmation et une interface graphique. En particulier, le langage est orienté vers les vecteurs et les listes de données; de plus, Matlab® est un langage interprété, c'est à dire que les commandes sont traduites en language machine à l'exécution; il ne nécessite donc pas de compilation. Toutes les informations concernant la distribution et l'installation de Matlab® sont disponibles sur le site officiel (`http://www.mathworks.com`).

GNU Octave est un langage qui peut remplacer Matlab® . Dans les séance d'exercices, nous utiliserons Matlab® , mais les commandes sont presque entièrement compatibles avec Octave. Octave est aussi un langage interprété; contrairement à Matlab® , Octave est un logiciel libre et gratuite, mais, dans sa version de base, il presente seulement une interface en ligne de commande. Une interface graphique est disponible séparément. Toutes les informations sur Octave sont disponibles sur le site officiel `http://www.gnu.org/software/octave/`.

Pour ouvrir Matlab® , cliquez sur *Start* → *Programmes* → *Matlab* sous Windows, ou tapez `matlab` dans un terminal sous Unix. Matlab® se présente avec un environnement interactif et un prompt (généralement >>) dans lequel on peut introduire des *commandes*. Par exemple, pour quitter Matlab® , tapez la commande:

```
>> quit
```

2 Aide en ligne

Matlab® propose une documentation complète en anglais. Vous avez également accès à une aide sur différents sujets en tapant la commande:

```
>> help commande
```

L'aide affiche un descriptif de la commande, par exemple :

```
>> help sum
 SUM Sum of elements.
    S = SUM(X) is the sum of the elements of the vector X. If
    X is a matrix, S is a row vector with the sum over each
    column. For N-D arrays, SUM(X) operates along the first
    non-singleton dimension.
    If X is floating point, that is double or single, S is
    accumulated natively, that is in the same class as X,
    and S has the same class as X. If X is not floating point,
    S is accumulated in double and S has class double.

    S = SUM(X,DIM) sums along the dimension DIM.

    S = SUM(X,'double') and S = SUM(X,DIM,'double') accumulate
    S in double and S has class double, even if X is single.

    S = SUM(X,'native') and S = SUM(X,DIM,'native') accumulate
    S natively and S has the same class as X.

    Examples:
    If   X = [0 1 2
              3 4 5]

    then sum(X,1) is [3 5 7] and sum(X,2) is [ 3
                                               12];

    If X = int8(1:20) then sum(X) accumulates in double and the
    result is double(210) while sum(X,'native') accumulates in
    int8, but overflows and saturates to int8(127).

    See also prod, cumsum, diff, accumarray, isfloat.
```

La commande `help` est **très utile**. Habituez-vous à l'utiliser chaque fois que vous avez besoin d'information sur la syntaxe et les paramètres d'entrée et sortie d'une commande.

3 Déclaration des variables

Matlab® permet de créer et d'initialiser des variables. Les déclarations suivent les règles suivantes:

a) toutes les variables sont des *matrices*;

b) les variables n'ont pas de type.

Exemples:

`a = 5`	*variable scalaire*	(1×1)
`b = [4.4 -6.9]`	*vecteur ligne*	(1×2)
`c = [-5; 2]`	*vecteur colonne*	(2×1)

`d = [2 3; −1 7]`	*matrice carrée*	(2×2)
`s = 'abc'`	*chaîne de caractères*	(1×3)
`t = 3+i`	*variable scalaire complexe*	(1×1)

Dans ces exemples, on a utilisé les opérateurs suivants:

a) séparateur de ligne: point-virgule ou *return* ;

b) séparateur de colonne: virgule ou espace blanc;

On a aussi utilisé l'unité imaginaire `i`, définie par $i = \sqrt{-1}$ (on peut utiliser de façon équivalente `j` ou `1i`). D'autres constantes fondamentales sont définies en Matlab® par

a) `pi`: le nombre $\pi = 3.14159265\ldots$

b) `exp(1)`: le nombre $e = 2.718281\ldots$

Il faut noter que Matlab® affiche les nombres avec quatre chiffres après la virgule, tandis que la représentation interne est faite avec 16 chiffres. Pour changer l'affichage des nombres en Matlab® , on utilise la commande `format`. Par exemple, si avant de taper `>> pi` (la variable `pi` est une approximation de π), nous écrivons

`format long`	nous obtiendrons	`3.14159265358979;`
`format short`	nous obtiendrons	`3.1416;`
`format long e`	nous obtiendrons	`3.14159265358979e+00;`
`format short e`	nous obtiendrons	`3.1416e+00.`

4 Workspace

Matlab® permet de connaître plusieurs informations sur les variables déclarées. Tapez:

`who`	pour afficher toutes les variables.
`whos`	pour afficher toutes les variables avec indication sur leurs tailles.
`size(A)`	pour afficher les dimensions de la matrice `A`.
`length(b)`	pour afficher la longueur du vecteur `b`.
`clear var`	pour effacer la variable `var`.
`clear` (ou `clear all`)	pour effacer toutes les variables.

5 Opérations fondamentales

Matlab® peut effectuer plusieurs opérations entre matrices. Les opérations fondamentales peuvent être partagées en deux catégories.

5.1 Opérations matricielles

Les opérations matricielles usuelles sont définies par `+, −, *, ^, \`

`C = A + B` est la somme matricielle, $C_{ij} = A_{ij} + B_{ij}$

`C = A * B` est le produit matriciel, $C_{ij} = \sum_k A_{ik} B_{kj}$

`C = A \ B` est la division matricielle, $C = A^{-1}B$

`C = A^3` est la troisième puissance matricielle (`C = A*A*A`)

Il faut remarquer que ces opérations sont bien définies seulement si les matrices ont des *dimensions cohérentes*. Pour l'addition `A+B`, `A` et `B` doivent avoir la même taille; pour le produit `A*B`, le nombre de colonnes de `A` et le nombre de lignes de `B` doivent être égaux; les opérations `A\B` et `A^3` demandent une matrice `A` carrée. Par exemple:

```
>> A = [1 2 3; 4 5 6]; B = [7 8 9; 10 11 12];
>> A + B
ans =

     8    10    12
    14    16    18

>> C = [13 14; 15 16; 17 18];
>> A + C
??? Error using ==> plus
Matrix dimensions must agree.
```

```
>> A * C
ans =

    94   100
   229   244

>> A * B
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

Notez que Matlab® renvoie un message d'erreur si les dimensions des matrices ne s'accordent pas avec l'opération commandée. Apprenez à lire ces messages d'erreur! Pour plus d'informations sur les messages d'erreur, voir Section 11.

5.2 Opérations élément par élément

Pour exécuter des opérations entre matrices élément par élément, il faut faire précéder l'opérateur d'un point. Les opérateurs élément par élément sont donc `.* ./ .^` et sont définis entre matrices

de même dimension ou entre un scalaire et une matrice

`C = A .* B` est le produit élément par élément, $C_{ij} = A_{ij}B_{ij}$

`C = A ./ B` est la division élément par élément, $C_{ij} = \dfrac{A_{ij}}{B_{ij}}$

`C = A.^3` est la troisième puissance élément par élément, $C_{ij} = A_{ij}^3$

6 Manipulation des matrices

Après avoir défini la matrice `A` (par exemple une matrice carrée `A` de taille `n`), Matlab® permet les opérations suivantes sur les *sous-matrices* de `A`.

6.1 Extraction de sous-matrices

`A(2,3)` extraction de l'élément A_{23}. Attention: le premier valeur d'une ligne/colonne/vecteur est indiqué par 1

`A(:,5)` extraction de la colonne $[A_{15}; \ldots; A_{n5}]$

`A(1:4,3)` extraction de la sous-colonne $[A_{13}; \ldots; A_{43}]$

`A(1,:)` extraction de la ligne $[A_{11}, \ldots, A_{1n}]$

`diag(A)` extraction de la diagonale $[A_{11}; \ldots; A_{nn}]$

6.2 Construction de matrices particulières

Matlab® permet, en plus, de définir des matrices particulières (une fois définis les entiers `n`, `m` et le vecteur `v`).

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

`A = diag(v)` matrice diagonale avec `v` comme diagonale

`A = diag(diag(B))` matrice diagonale, avec diagonale égale à celle de `B`

`A = zeros(n,m)` matrice de zéros avec `n` lignes et `m` colonnes

`A = ones(n,m)` matrice de un avec `n` lignes et `m` colonnes

`A = rand(n,m)` matrice aléatoire de nombres entre 0 et 1 avec `n` lignes et `m` colonnes

`x = debut:pas:fin` vecteur ligne de points équidistribués avec pas `pas` entre `debut` et `fin`

`x = linspace(debut,fin,n)` vecteur ligne de `n` points équidistribués entre `debut` et `fin`

6.3 Fonctions matricielles

Soit `A` une matrice, Matlab® permet d'effectuer directement les opérations suivantes.

`C = A'`	transposée de `A`, $C_{ij} = A_{ji}$. Si `A` est complexe, $C_{ij} = \overline{A_{ji}}$
`C = inv(A)`	inverse de `A` (matrice carrée), $C = A^{-1}$
`d = det(A)`	déterminant de `A` (matrice carrée)
`r = rank(A)`	rang de `A`
`nrm = norm(A)`	norme 2 de `A`
`v = eig(A)`	valeurs propres de `A` (matrice carrée)
`[V,D] = eig(A)`	`V`: matrice des vecteurs propres de A; `D`: matrice diagonale avec les valeurs propres de `A`.

Soit `A` une matrice carrée de taille `n` et soit `b` un vecteur de taille `n`, alors le vecteur `x`, solution du système linéaire `Ax = b` est défini par

```
>> x = inv(A) *  b
```

Cette commande est théoriquement correcte, cependant, si on est intéressé seulement à la solution `x` et pas à l'inverse `inv(A)`, il est préférable d'utiliser un *backslash*

```
>> x = A \ b
```

Cette commande résout le système linéaire avec des algorithmes fiables et optimaux.

7 Boucles de contrôle

Matlab® offre, comme plusieurs autres langages, quelques boucles de contrôle. Ces commandes sont spécialement utiles pour écrire des programmes Matlab® .

7.1 Boucle `for`

Si l'on veut exécuter des instructions pour chaque valeur

$$i = a,\ a+1,\ a+2,\ a+3,\ \ldots,\ a+n$$

d'une certaine variable i entre deux bornes assignées a et $a+n$, on utilise **`for`**. Par exemple pour calculer le produit scalaire `ps` entre deux vecteurs `x` et `y` de taille `n`, on pose $a = 1$ et on utilise:

```
>> n = length(x)
>> ps = 0;
>> for i = 1:n;
>>    ps = ps + x(i)*y(i);
>> end
```

Cette boucle est équivalente au produit matriciel entre le vecteur transposé de `x` et le vecteur `y`, en supposant que les facteurs soient deux vecteurs colonnes:

```
>> ps = x'*y;
```

Pour créer un vecteur dans une boucle, on peut faire:

```
>> % Creer un vecteur v = sqrt(range)
>> range = linspace(0,1,11);
>> v = [];                      % Defini un vecteur vide
>> for i = range
       v = [v ; sqrt(i)];       % Ajoute l'element sqrt(i) a la suite du vecteur v
   end
>> plot(range, v)               % Pour faire un graphe
```

7.2 Boucle `while`

Si l'on veut exécuter plusieurs fois des instructions pendant qu'une expression logique est vraie, on utilise **`while`**. Par exemple, le même calcul qu'on a considéré avec la boucle **`for`** peut être exécuté avec

```
>> n = length(x)
>> ps = 0;
>> i = 0;
>> while (i < n);
>>    i = i + 1;
>>    ps = ps + x(i)*y(i);
>> end
```

7.3 Instruction conditionelle `if`

Si l'on veut exécuter des instructions seulement si une expression logique est vraie, on utilise **`if`**. Par exemple, si l'on veut calculer la racine carrée d'une variable scalaire `r` seulement si `r` n'est pas négatif:

```
>> if (r > = 0)
>>   racine = sqrt(r);
>> end
```

On a plusieurs *opérateurs logiques* à disposition:

Opérateur	Action logique
`&&`	*and*
`||`	*or*
$\sim$	*not*
`==`	*equal to*

Taper par exemple `help &` pour une explication de la liste des opérateurs.

8 Scripts et fonctions

Matlab® permet l'exécution de listes de commandes sauvegardées dans un fichier. Les outils principaux sont les script files et les fonctions.

8.1 Script files

Un script file est une suite de commandes Matlab® . Les noms des script files doivent avoir l'extension ".m". On peut aussi écrire des lignes de commentaires, que Matlab® n'exécutera pas. Les lignes de commentaires sont introduites par %. Pour exécuter un script file tapez son nom, sans extension, dans le prompt Matlab® .
Une bonne pratique est de commencer un script avec les trois commandes `clc`, `close all`, `clear all`. Par exemple si le fichier `test.m` contient les commandes suivantes

```
clc
close all
clear all
a=0;
for i=1:5
   a=a+i^2;
end
a
```

en tapant

```
>> test
```

on aura

```
ans =
    55
```

8.2 Fonctions

Une fonction Matlab® est une suite de commandes qui nécessite un ou plusieurs input pour être exécutée et qui renvoie un ou plusieurs output. On a déjà vu plusieurs fonctions; en effet, la plupart des commandes introduites dans la Section 6 sont des fonctions. Par exemple, la fonction `size` renvoie comme output les dimensions d'une matrice.

8.2.1 Utiliser les fonction

Supposons que la fonction `helloworld` utilise comme input une scalaire et un vecteur et donne comme output trois variables. La syntaxe suivante permet d'appeler la fonction et de mémoriser ses outputs dans des variables:

```
>> helloworld(3,[pi −5]); % aucun output est memorise'.
>> out=helloworld(3,[pi −5]); % memorise le premier output.
>> [out1, out2]=helloworld(3,[pi −5]); % memorise le premier et deuxieme outputs.
>> [out1, out2, out3]=helloworld(3,[pi −5]); % memorise le trois outputs.
>> [∼, ∼, out3]=helloworld(3,[pi −5]); % memorise le troisieme output seulement
```

8.2.2 Ecrire une fonction

On a deux façons pour écrire une fonction en Matlab® :

a) @-fonctions;

b) .m functions.

@-functions

On peut définir rapidement des fonctions simples (c'est-à-dire des fonctions qu'on peut écrire en une ligne seulement) avec la commande `@(parameter1, parameter2, ...)`.

Pour définir une fonction `f` qui dépend de la variable `x`, on écrit:

```
>> f = @(x) log(x) - 1;
>> f(1)
ans =
    -1
```

Si `f` dépend de plusieurs variables, nous écrirons par exemple

```
>> f = @(x,p) log(x) - p;
>> f(1,1)
ans =
    -1
```

.m-functions

Pour écrire des fonctions plus longues qu'un seule ligne, on écrit les commandes de la fonction dans un .m file particulier, qui suit ces règles:

a) Le fichier doit commencer par:
`function` [*output arguments*] = `nom_fonction(`*input arguments*`)`
Par exemple, l'en-tête d'une fonction pour la méthode de la dichotomie pourrait être:

```
function [zero,res,niter]=bisection(fun,a,b,tol,nmax,varargin)
%BISECTION Find function zeros.
%   ZERO=BISECTION(FUN,A,B,TOL,NMAX) tries to find a zero ZERO
%   of the continuous function FUN in the interval [A,B]
%   using the bisection method.
            .
            .
            .  % istructions
            .
 zero = ...;
 res = ...;
 niter = ...;
            .
            .  % istructions
            .
            .

return          % fin du corps de la fonction
```

b) Le fichier doit avoir le même nom que la fonction (avec extension `.m`)

c) Les lignes de commentaires qui éventuellement suivent cet en-tête constituent le help de la fonction.

d) Toutes les variables internes à la fonction sont locales (c'est-à-dire, on pourra pas accéder à leur valuers dans la ligne de commande >>).

Par exemple, on peut écrire la fonction suivante dans le fichier `my_function.m`

```
function f = my_function(x);
f = x.^3 - 2*sin(x) + 1;
```

qui on peut évaluer comme

```
>> my_function(0)
ans =
    1
```

Attention: pour pouvoir exécuter un script ou une fonction, il faut que le fichier `.m` correspondant soit sauvegardé dans le *répertoire de travail*. Le répertoire de travail est normalement indiqué en haut de la fenêtre principale de Matlab®.

9 Mesurer les temps d'exécution

Pour mesurer le temps pour l'exécution d'une suite des commandes/fonctions Matlab®, il faut utiliser les commandes `tic` et `toc`.

```
>> tic
>> % des commandes ...
>> toc
Elapsed time is 0.000908 seconds
```

10 Graphisme 2D

La commande `plot(a,b)` reçoit comme input deux vecteurs `a` et `b` et crée un graphe dans un plan cartésien en utilisant les valeurs de `a` comme abscisses et les valeurs de `b` comme ordonnées. Par exemple, en tapant `plot([1 2 7 8],[5 2 −1 pi])` on obtient le graphe en Figure 1-gauche. Si les abscisses sont $1, 2, 3, \ldots$, on peut aussi utiliser la syntaxe `plot([5 2 −1 pi])`, qui est en fait équivalent à `plot([1 2 3 4],[5 2 −1 pi])` (voir Figure 1-droite).

Pour tracer le graphe de $f_1(x) = x^3 + 1 - 2\sin(\pi x)$ et $f_2(x) = x^3 + 1$ (voir Figure 2), il faut donc passer par les étapes suivantes:

a) Définir un *vecteur de points* dans l'intervalle donné:

```
>> x = −1:0.01:1;
```

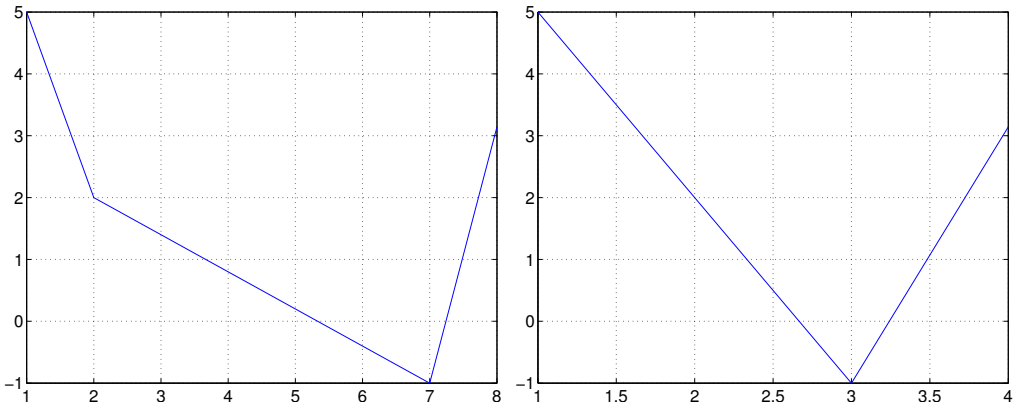

Figure 1: Gauche: résultat de la commande `plot([1 2 7 8],[5 2 −1 pi])`. Droite: résultat de la commande `plot([5 2 −1 pi])`. Observez que les abscisses sont différentes).

b) Créer des vecteurs contenants les évaluations des fonctions $f_1(x), f_2(x)$ pour chaque composant du vecteur `x`. Pour faire ça, on a deux possibilités: soit calculer immédiatement les vecteurs, avec les commandes

```
>> y1 = x.^3 + 1 − 2*sin(pi*x);
>> y2 = x.^3 + 1;
```

soit définir d'abord les fonctions $f_1(x), f_2(x)$:

```
>> f1 =  @(x) x.^3 + 1 − 2*sin(pi*x);
>> f2 =  @(x) x.^3 + 1;
```

et en suite évaluer les fonctions `f1` et `f2`:

```
>> y1 = f1(x);
>> y2 = f2(x);
```

Observez qu'on a utilisé `.^` (puissance élément par élément) et non pas `^` (puissance matricielle), parce qu'il faut évaluer la fonction en chaque élément du *vecteur* `x`.

c) Tracer le graphe:

```
>> plot(x,f1(x),'b')
>> hold on
>> plot(x,f2(x),'g')
>> hold off
```

L'option `'b'` ou `'g'` permet de spécifier la couleur de la ligne ("blue" ou "green"). Voir `>> help plot` pour une liste d'options. En utilisant les commandes `hold on` et `hold off`, tous les graphes tracés entre les deux commandes sont superposés au graphe précédent. Ceci permet de visualiser plusieurs fonctions dans la même fenêtre.

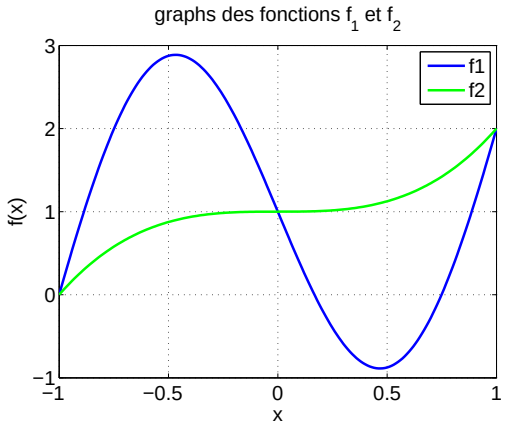

Figure 2: Graphes des fonctions $f_1(x)$ et $f_2(x)$

Il y a plein d'autres options graphiques en Matlab® . Voici une liste des commandes qui permettent d'avoir aussi une grille de repère (`grid`), un descriptif des axes (`xlabel` et `ylabel`), une légende (en spécifiant le nom de chaque ligne avec l'option `'DisplayName'` de la commande `plot`, et utilisant après la commande `legend show`) ainsi que un titre (`title`) et de spécifier aussi l'épaisseur des lignes du graphe (`'LineWidth',2`):

```
>> figure(1)
>> axes('fontsize',16)
>> plot(x,f1(x),'b','LineWidth',2,'DisplayName','f1')
>> hold on
>> plot(x,f2(x),'g','LineWidth',2,'DisplayName','f2')
>> grid on
>> xlabel('x')
>> ylabel('f(x)')
>> legend show
>> title('graphes des fonctions f_1 et f_2')
```

cette commande ouvre la fenêtre 1 avec le graphe. Le résultat de ces commandes est représenté dans la Figure 2. Pour plus de détails sur la fonction `plot` tapez `>> help plot`.

Matlab® donne aussi la possibilité de créer avec simplicité des graphes en échelle logarithmique. On a trois possibilités (voir Figure 3):

a) `semilogx(x,y)` trace un graphe dont l'échelle horizontale est logarithmique tandis que l'échelle verticale est lineaire. Cette fonction est idéale pour dessiner le graphe de données ou fonctions qui s'étendent sur des grands intervalles de `x`. Les valeurs de `x` peuvent être seulement positives (car on en prend le logarithme). En particulier, le graphe de la fonction $f(x) = \log(x)$ est une lignes droite dans un graphe `semilogx`.

b) `semilogy(x,y)` trace un graphe dont l'échelle horizontale est linéaire tandis que l'échelle verticale est logarithmique. Cette fonction est idéale pour dessiner le graphe de données ou fonctions qui ont des grandes variations sur des petits intervalles de `x`. Les valeurs de `y` peuvent être seulement positives. En particulier, les graphes des fonctions $f(x) = e^{kx}, k = 1, 2, 3 \ldots$ sont lignes droites à pente $1, 2, 3 \ldots$ dans un graphe `semilogy`.

c) `loglog(x,f1(x))` trace un graphe dont les deux échelles horizontales et verticales sont

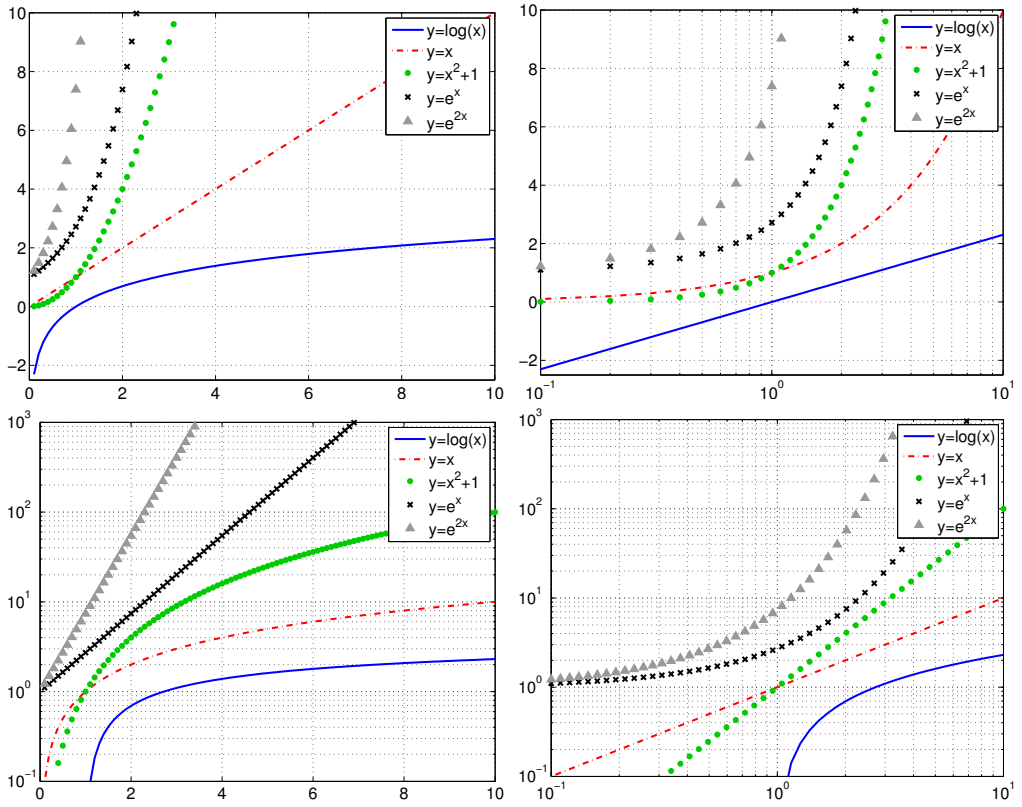

Figure 3: Graphiques en échelles linéaires (en haut à gauche), `semilogx` (en haut à droite), `semilogy` (en bas à gauche), `loglog` (en bas à droite) des fonctions $y = \log(x), y = x, y = x^2, y = e^x, y = e^{2x}$

logarithmiques. Les valeurs de `x` et de `y` peuvent être seulement positives. Les graphes des fonctions $f(x) = x^k, k = 1, 2, 3 \ldots$ sont lignes droites à pente $1, 2, 3 \ldots$ dans un graphe `loglog`. En conséquence, cette fonction est idéale pour reconnaître ce type de relation entre deux quantités.

11 Apprendre à lire les messages d'erreur

Comme déjà dit à la fin de la Section 5.1, chaque fois qu'on saisit une commande qui contient une erreur de syntaxe, Matlab® s'arrête et donne un message d'erreur. Ces messages donnent souvent des informations très claires sur l'erreur passé, et donc sur la correction des erreurs. Les erreurs les plus communes sont:

a) `Undefined function or variable 'b'`: on a essayé d'extraire la valeur d'une variable qui n'existe pas

```
>> clear all
>> b
```

b) `Index exceeds matrix dimensions`: on a essayé d'extraire une valeur d'une matrice, mais hors de la taille de la matrice.

```
>> a=[1 2 3]
>> a(4)
```

c) `Subscript indices must either be real positive integers or logicals`: on a essayé d'extraire une entreée d'une matrice en utilisant un indice qui n'est pas un entier supérieur ou égal à 1 (par exemple, 0 ou une valeur réelle):

```
>> a=[1 2 3]
>> b=3.2;
>> a(0)
>> a(1.5)
>> a(b)
```

d) `Error using +`
`Matrix dimensions must agree.`

On a essayé de additioner deux matrices de dimensions non cohérentes:

```
>> a=[1 2]
>> b=[3; 4]
>> a+b
```

e) `Error using *`

`Inner matrix dimensions must agree.`

On a essayé de multiplier deux matrices de dimensions non cohérentes. On a peut-être oublié le caractère "`.`" qui déclare l'opération élément-par-élément?

```
>> a=[1 5 5]
>> b=[3 2 6]
>> a*b
```

f) `Error using ^`

`Inputs must be a scalar and a square matrix.`

On a essayé de calculer la puissance d'une matrice non carrée. On a peut-être oublié le caractère "`.`" qui déclare l'operation élément-par-élément?

```
>> [1 2; 3 4; 4 5]^2
```

g) `Undefined function 'helloworld' for input arguments of type 'double'.` La fonction `helloworld` n'existe pas, ou elle n'est pas dans le *répertoire de travail*:

```
>> helloworld(4)
```

h) `Error using plot`

`Vectors must be the same lengths.`

Les vecteurs que l'on veut utiliser comme abscisse et ordonnée ont des longeurs différentes.

```
>> plot([1 2 3],[4 5])
```

i) `Error: Unexpected MATLAB operator.`

Matlab® n'arrive pas à interpréter la commande que l'on a saisi. Probablement il s'agit d'une faute de frappe. Une ligne verticale vous indiquera la position de l'erreur!

```
>> a=a+*5
a=a+*5
   |
Error: Unexpected MATLAB operator.
```

j) `Error using plot`

`Conversion to double from function_handle is not possible.`

On veut tracer le graphe d'une fonction, définie par `@`, mais on a oublié d'évaluer la fonction:

```
>> f=@(t) t.^2;
>> x=1:10;
>> plot(x,f) % au lieu de plot(x,f(x))
```

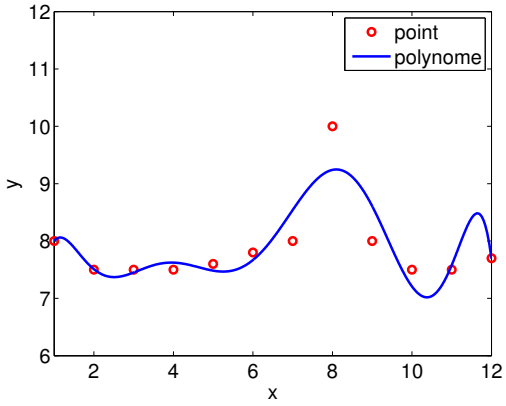

Figure 4: Données et polynôme d'interpolation

12 Polynômes en Matlab®

Considérons un polynôme de degré n:

$$f(x) = a_n x^n \ + a_{n-1} x^{n-1} \ + \ \ldots \ + a_1 x \ + \ a_0.$$

Matlab® représente les polynômes de degré n sous la forme d'un vecteur $p = [a_n, a_{n-1}, \ldots, a_0]$ de taille $n+1$ qui contient les coefficients en ordre décroissant par rapport au degré associé. Par exemple, le vecteur associé au polynôme $f(x) = 3x^3 - 4x^2 + x$ est

```
>> p = [3, −4, 1, 0];
```

Pour évaluer ce polynôme, on utilise la commande `polyval`:

```
>> x = 0:.1:1;
>> y = polyval(p, x);
```

Dans ce cas, les éléments du vecteur `y` sont les valeurs du polynôme calculées pour chaque élément du vecteur `x`.
Pour obtenir le vecteur associé au polynôme de degré `n` approximant un jeu de données, on utilise `polyfit`. Si la taille des données est plus grande que `n+1`, on a le polynôme approximant au sens des moindres carrés; si elle est égale à `n+1`, on a le polynôme interpolant. Par exemple, si on veut calculer le polynôme de degré 8 qui approche les données du vecteur `y` (au sens des moindre carrés) et en tracer le graphe, il faut utiliser le script suivant;

```
clc
clear all
close all


x = [1 2   3   4   5   6   7 8  9 10  11  12];
y = [8 7.5 7.5 7.5 7.6 7.8 8 10 8 7.5 7.5 7.7];

p     = polyfit(x, y, 8); % Calculons le polynome interpolant (degre=8)
```

```
x_int = x(1):0.01:x(end)
y_int = polyval(p, x_int)

figure(1)
axes('fontsize',16)
plot(x, y, 'or', x_int, y_int, 'b', 'linewidth', 2);
xlabel('x')
ylabel('y')
legend('point','polynome')
axis([1 12 6 12]);
```