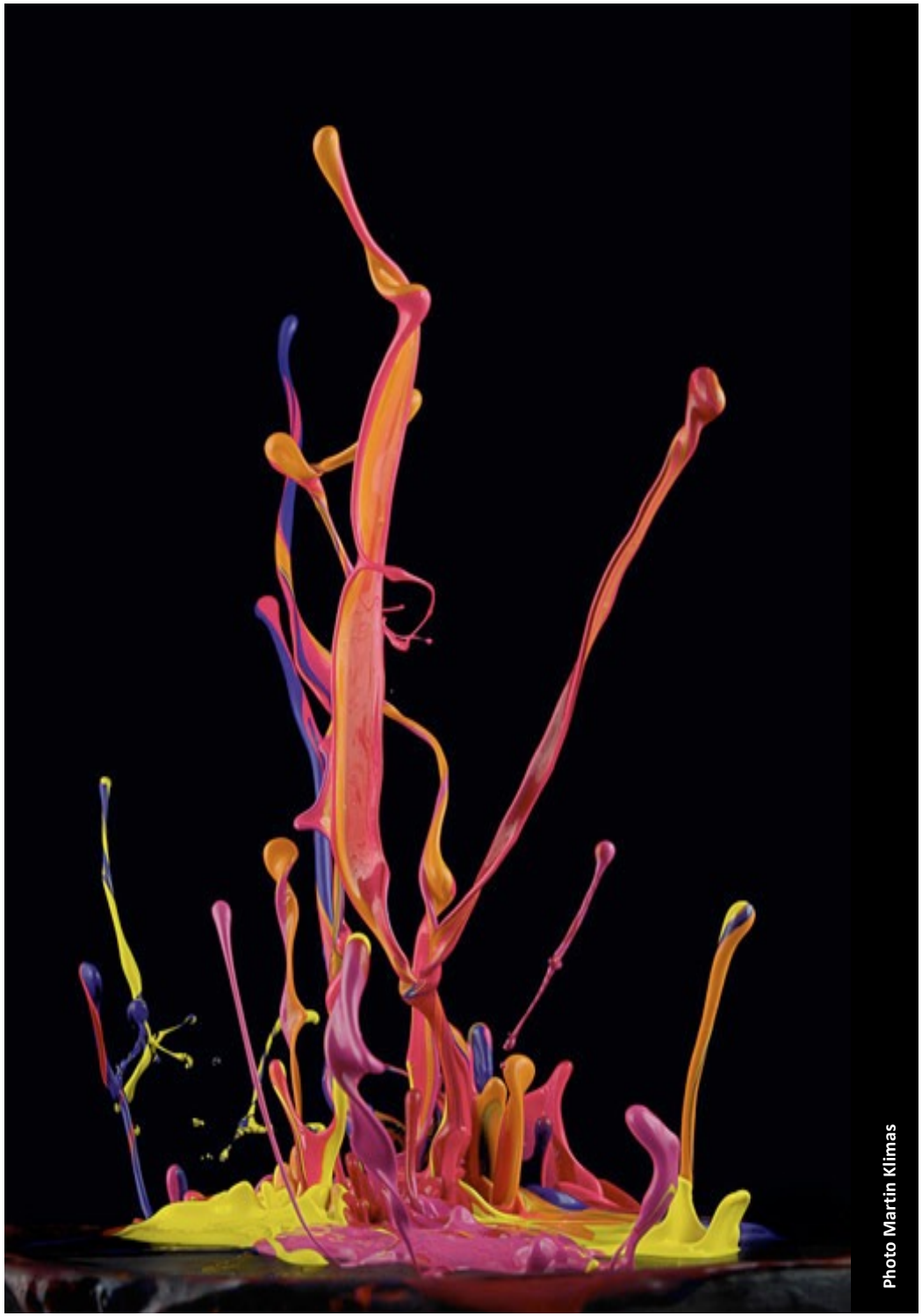


EPFL

Programmation pour Ingénieur

Matlab II

ME 3e semestre

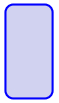
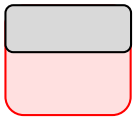
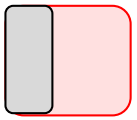
rev. 2025.r1

Christophe Salzmann

Laboratoire d'Automatique

Prev. module - recap

- Matlab mode interactif
- Repenser les problèmes en fonction de matrices/vecteurs pour éviter les boucles
- Les fonctions sont polymorphiques
- Δ **Les indices commencent à 1**
- Vecteur ligne [1 2 3]
- Vecteur colonne [1; 2; 3]
- 1e ligne d'une matrice M(1, :)
- 1e colonne d'une matrice M(: ,1)
- Δ **Accès matrice(ligneY, colonneX)**

Matlab recap

```
v = [1:0.5:5]

v(2)

v(10)

w=linspace(0,8,5)

w(6)=10

s=sum(ones(3,2))
```

```
m = [1 2 3; 4 5 6]

m(3)

m(1,3)

m(:,[1 3])=m(:,[3 1])
```

Matlab recap

```
v = [1:0.5:5]
      1.0   1.5   2.0   2.5   3.0   3.5 …
          4.0   4.5   5.0
v(2)
      1.5

v(10)
     ??? Subscripted assignment …
         dimension mismatch.

w=linspace(0,8,5)
      0   2   4   6   8

w(6)=10
       0   2   4   6   8   10

s=sum(sum(ones(2,3)))
         sum(sum([1,1,1;1,1,1])))
      6

s=sum(ones(2,3),'all')
      6
```

```
m = [1 2 3; 4 5 6]

      1      2      3
      4      5      6

m(3)
         2

m(1,3)
          3

m(:,[1 3])=m(:,[3 1])

      3      2      1
      6      5      4
```

Matlab recap

```
> sin(1) = 0.8415;
> sin(1)

ans =

    0.8415

> sin(2)
```

Matlab recap

```
> sin(1) = 0.8415;
> sin(1)

ans =

    0.8415

> sin(2)

       Index exceeds matrix dimensions.


> clear sin
> sin(pi)

ans =

   1.2246e-16
> eps
```

Matlab recap

```
m = [1 2 3; ...
     4 5 6; ...
     7 8 9]
```

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

```
m([1 2],[1 2])
```

```
     1     2
     4     5
```

```
m([1,2])
```

```
     1     4
```

Module 2

- Vectorisation + find()
- Programmer Matlab
- @ fonction anonyme, I
- Graphiques I

Vectorisation

Matlab est fait pour travailler avec des vecteurs/matrices. Les opérations itératives sur des éléments de vecteurs/matrices peuvent se transformer en une seule opération sur le vecteur ou la matrice.

```
x=1:1000000;
for n=1:length(x)
        y(n)=x(n)^3;
end
```

```
y= x.^3;
```

Accède aux *n* éléments du vecteur 1 à 1

115.850 ms

Accède au vecteur complet en 1 fois

18.779 ms

Demo

- Vectorization (2.2)

Vectorisation avec `find()`

La fonction `find(cond)` retourne les **indices** des éléments du vecteur/matrice pour lesquels la condition est vrai

On désire connaître le nombre d'éléments >0 d'un vecteur

```
x = [0.3, -0.5, -0.6, 0.8, 0.2, -0.1];

count=0;
for n=1:length(x)
    if x(n)>0
       count=count+1;
    end
end
```

```
count = length(find(x>0))
```

```
p = find(x>0)

  = [1 4 5]
```

Demo

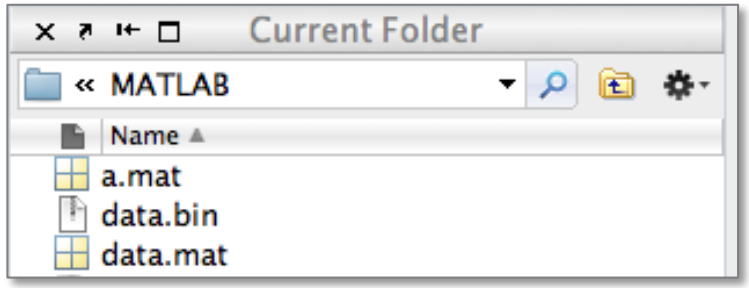
- Find() (2.3)

Vectorisation avec `find()`

```
>> x = rand(1,5)
x =
         0.1419    0.9157    0.7922     0.4218      0.9595

>> idx = find(x<0.5)  %   <-- indices
idx =
          1    4

>> y = x(idx) %   <-- valeurs
y=
       0.1419    0.4218

>> x(idx) = -x(idx); %   <-- modifie les valeurs
x =
      -0.1419    0.9157    0.7922   -0.4218    0.9595

x(idx) = 0;
x  =
       0.0000    0.9157    0.7922    0.0000    0.9595
```

Programme Matlab – script.m

- Un programme Matlab (ou script) est un fichier composé de commandes matlab. Il a un extension '**.m**'. Ces commandes sont éxécutées une à une, de la même manière que si elles avaient été entrées dans la ligne de commande, les variables sont sauvées dans le *workspace* et **des conflits sont donc possibles!**
- Matlab gère une table de chemins d'accès dans lesquels il va chercher les fichiers demandés. Si un fichier se trouve en dehors de cette table il sera invisible et matlab retournera une erreur. **De même les chemins relatifs le sont par rapport au dossier courant.**



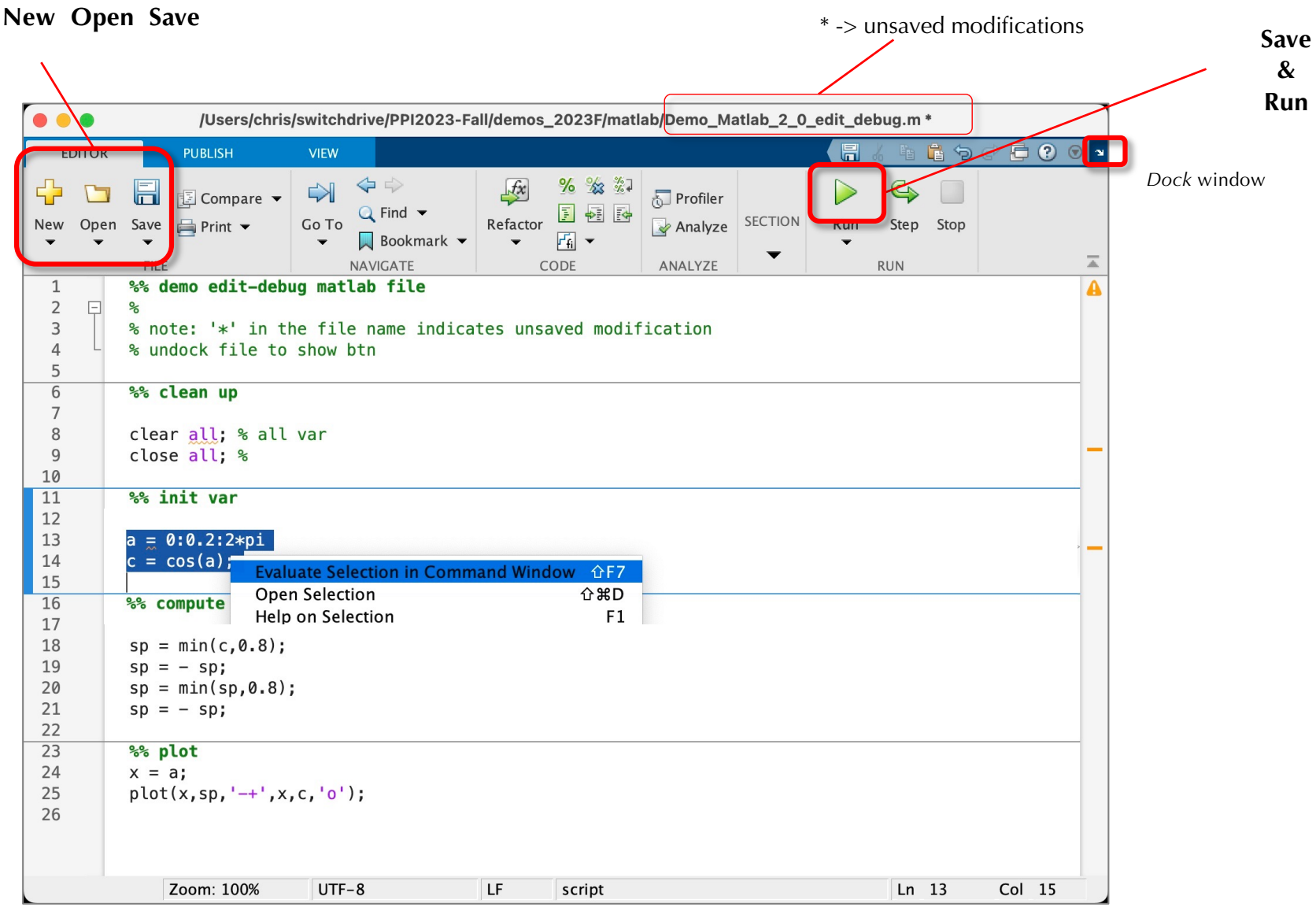
liste des chemins d'accès courants

```
>> path
```

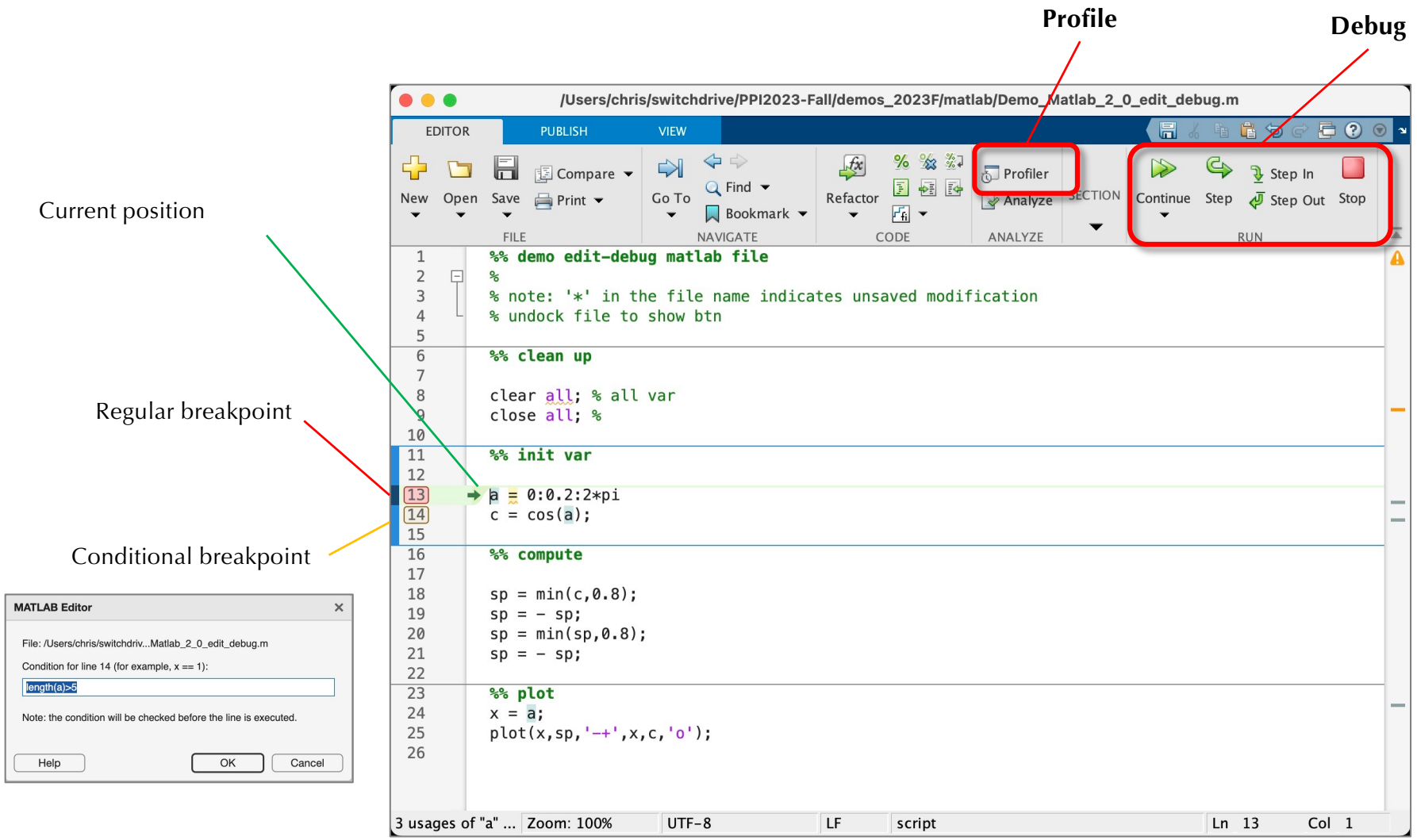
Demo

- .m script file (2.1)
  - Section %%
  - Debug
  - Profiler

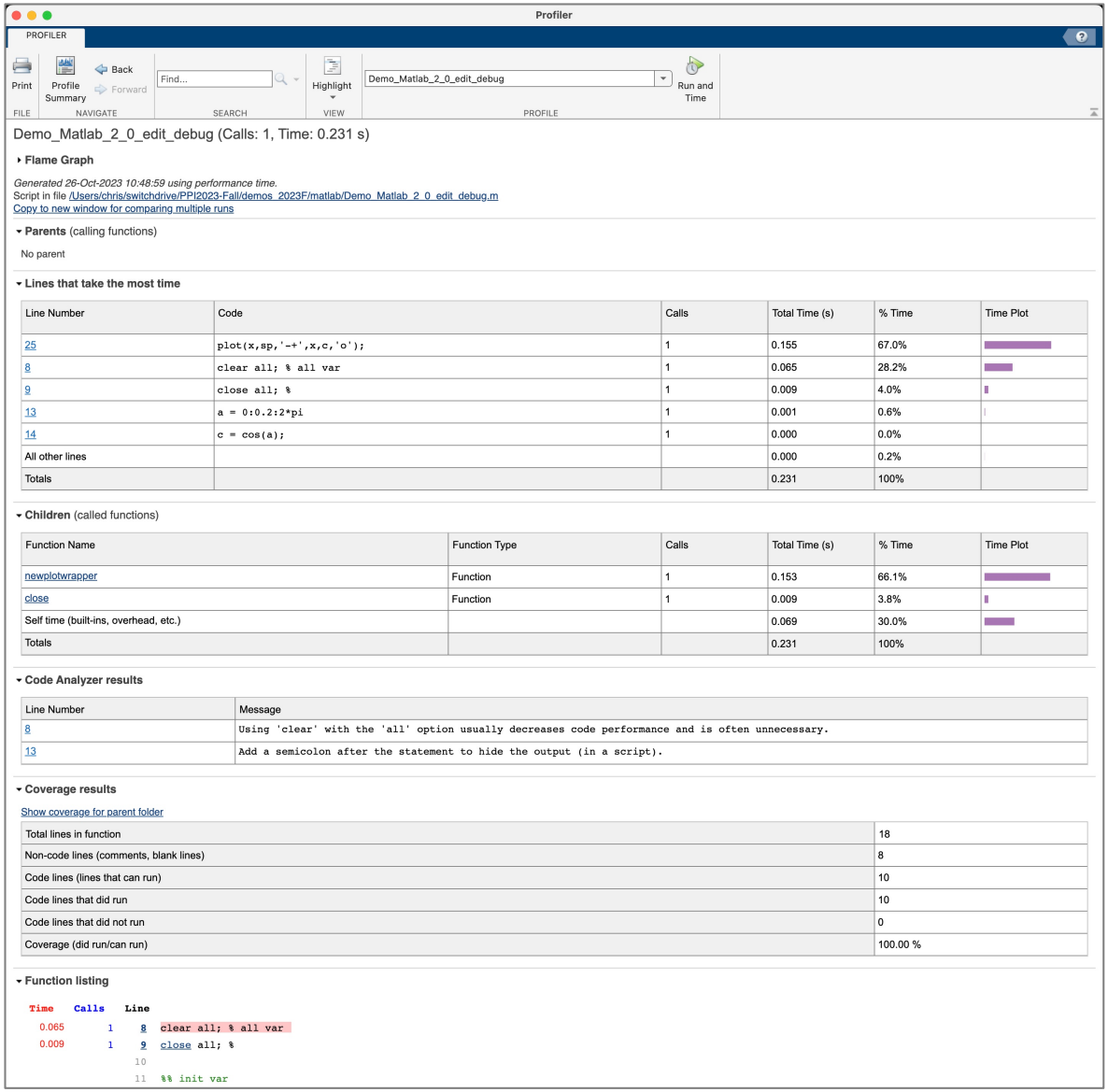
Script file – edit & run



Script file – debug



Script file - profiler



Simple profiling

```
>> tic % start timer
>> yourcode
>> toc % get delta
```

Profiler

Where you spent time

Coverage

Code that has been executed

Matlab flow control

- Matlab est similaire à c/c++/pascal
- La plus part des opérations ont la même signification avec quelques exceptions, ex. `!=` en c, devient `~=`
- La liste complète des opérateurs avec `help .`

point

```
if cond1
      commands1
elseif cond2
      commands2
else
      commands3
end
```

pas besoin de parenthèses

termine le bloc `if`

```
if (cond1) {
    commands1
    }
elseif (cond2) {
    commands2
    }
else {
    commands3
    }
```

Matlab flow control

```
switch switch_expression
   case case_expression
      statements
   case { expr1, expr2 }
      statements
    …
   otherwise
      statements
end
```

```
switch(expression) {
   case case_expression:
         statements;
         break;
   case case_expression:
         statements;
         break;
    …
   default
     statements;
}
```

Matlab flow control

i *ne serait pas un bon choix !*

```
for n=1:10
     commands1(n)
end
```

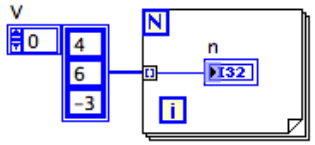
n *va prendre les valeurs de 1 à 10*

```
for n=1:0.5:2
     commands1 (n)
end
```

n *va prendre les valeurs de 1 à 2 avec un pas de 0.5*

```
for n=V
     commands1(n)
end
```

n *prendre les valeurs successives du vecteur V*




```
for
(n=1;n<=10;n++){
    commands1(n);
}
```

```
for (n=1;n<=2;n+=0.5){
     commands1(n);
}
```

```
for (n=1;n<=size_V;n++){
    commands1(V[n]);
}
```

Matlab flow control

```
while cond
      commands
end
```

`break, continue,` same as in c/c++

Boolean conditions

`0` -> false

`nonzero` -> true

```
While (cond){
    commands;
}
```

Matlab functions

- Une fonction Matlab est un fichier composé de commandes matlab. Il a un extension '.m'.
- Ce fichier doit avoir le **même nom** que la fonction qu'il contient sans l'extension'.m'.
- Le keywork `function` doit être présent dans le fichier, à la première ligne éxecutable.
- Le keywork `end` termine la fonction, il est optionnel mais fortement recommendé.
- Les **variables** à l'intérieur de la fonction sont **locales**, i.e ne modifie pas le contenu du *workspace*.
- Le fichier '.m' peut contenir plusieurs **sous-fonctions**, elles ne sont visibles que par la fonction principale.
- Les **sous-fonctions** peuvent se trouver à l'intérieur d'une autre fonction, à éviter car peu lisible.

Matlab functions, cont.

- Il est possible d'avoir des variables **globales** à l'aide du mot réservé global

Demo

- Fonction simple (myFunc.m)
- Sous fonction
- Multiple params de retour (myFunc2.m)
- Variable globale (myFuncGlobal.m)

Matlab functions

myFunc.m 

```
% nom du fichier = nom de la fonction + '.m'
% doc as usual, will be displayed with '>> help myFunc'

function out1 = myFunc(in1)
        a = in1+1                       % 'a' is a local var
        out1 = subFunc(a) + 2           % sub fcn  call
                                        % must be in the same file
end                                     % optional


function [out2] = subFunc(in1)
        out2 = -in1                     % 'out2 =' returns 'out2'
                                        % => return out2; in c
end
```

Matlab functions

Une fonction Matlab peut avoir **plusieurs** paramètres d'entrée et de sortie

```
function [out1, out2] = myFunc2(in1,in2)
   out1 = in1+in2              % "out1 = …" will be displayed
   out2 = (in2+in1)/2;         % ';' -> won't be displayed
end
```

Si omis, seulement le 1er paramètre de sortie esr retourné dans `ans`, dans le cas de `myFunc2` c'est `out1`

```
>> myFunc2(3,2)
out1 =
       5
ans
       5
```

```
>> [a, b]= myFunc2(3,2)
out1 =
       5
a =
       5
b =
       2.50
```

Matlab functions – var arg

Une fonction Matlab peut avoir un nombre variable de paramètres d'entrée et de sortie variable.
Le fonctionnement est similaire aux paramètres de la fonction main() en c/c++

```
function [out1, varargout] = myFcn2(in1, varargin)
   out1 = in1;                                    % in1, out1 -> fixed parameters
   nin  = nargin;                                 % nbr of input parameters
   nout = nargout;                                % nbr of output parameters
   fprintf('nb in param: %d, var: %d\n', nin, size(varargin,2));
   fprintf('nb out param: %d, var: %d\n', nout,nout-1);
   if nin>1
      for n=1:nin-1
        fprintf('Param in %d:\n', n);             % same a c syntax
        disp(varargin{n});                        % varargin is a Cell vector
      end
   end
   if nout>1
      for n=1:nout-1                              % minus the nbr of fixed out param
        varargout(n)={[1:n]};                     % varargout is a Cell vector
        fprintf('Param out %d:\n', n);
        disp(varargout{n});
      end
   end
end
```

@ fonction

Matlab permet de définir une référence sur une fonction à l'aide de *@myfunc*

Définir une référence (handle) sur une fonction permet par exemple de passer cette référence en paramètre d'une autre fonction.

Par exemple **`integral()`** calcule l'intégrale de la fonction passée en paramètre, pas besoin de réécrire **`integral()`** pour chaque nouvelle fonction à intégrer.

Exemple:

```
xmin = 0;
xmax = pi;

a = integral(@myF,xmin,xmax)        % utilise la reference à la fonction

myFHandler = @myF2;                 % myFHandler est une variable

b = integral(myFHandler,xmin,xmax) % utilise la variable qui contient la reference à la fonction

c = myFHandler(3)                   % myF2 est accessible via sa référence


function y = myF(x)
  y = x.^3;
end

function y = myF2(x)
  y = x.^2;
end
```

@ fonction anonyme

Une fonction annonyme est associée à une référence (handler). Le corps de la fonction est défini sur 1 ligne.

```
myHandler = @ (params) core
```

`myHandler` référence sur la fonction

`@` défini la fonction anonyme

`(params)` 0 ou plusieurs paramètres

`core` corps de la fonction, doit tenir sur 1 ligne

Exemples

```
>> mySQR= @(x) x.^2;
>> mySQR(5)
>> ans =  25

>> myHypot= @(x,y) sqrt(x.^2+y.^2);
>> myHypot (3,4)
>> ans =  5
```

Demo

- Demo anonymous functions (4.1a,b)

@ fonction anonyme

Les fonctions anonymes n'ont qu'**un** paramètre de retour, à moins qu'elles n'emploient des fonctions[#] qui en retournent plus d'un.

[#] *voir fonction* `deal()`

```
>> myMax= @(X) max(X);
>> myMax(rand(1:5))
>> 0.91234
>> [v,p]= myMax(rand(1:5))
>> v = 0.9706
>> p = 2
```

Les variables utilisées lors de la création de la fonction anonyme sont **mémorisées** dans la fonction

```
>> a= 123;
>> myF1= @(x) x+a; % 'a' var locale
>> myF1 (5)
>> ans =  128
>> clear a
>> a
>> Undefined function or variable 'a'
>> myF1 (5)
>> ans =  128 % 'a' est mémorisé dans la fonction anonyme
```

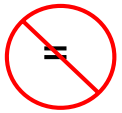
@ fonction anonyme et '='

Il n'est pas possible de faire une assignation explicite dans une fonction anonyme

- Ex.

NO!

```
FlipH = @(X) X(1:end)   =   X(end:-1:1)
```



- Why ?

La définition de la fonction anonyme comporte déjà une assignation implicite.

```
FlipH = @(X) X(end:-1:1)
```

peut se lire comme:

```
Resulat-de-FlipH(X)-est-égale-à:    X(end:-1:1)
```

@ fonction anonyme et '='

Ecrire une fonction anonyme qui replace la 2e valeur d'un vecteur par son inverse

```
Inv2 = @(X) X[2] = -X[2]
```

Une solution possible:

```
Inv2 = @(X) [X(1), -X(2), X(3:end)]
```

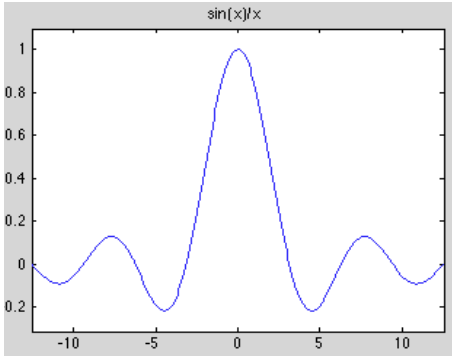
Demo

- anonymous myEzplot (4.2)

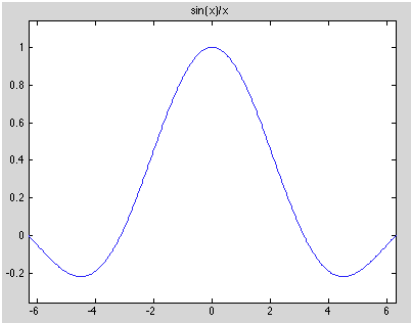
@ fonction anonyme

Beaucoup de fonctions matlab accèptent des fonctions annonymes en paramètre. Par exemple la fonction `ezplot()` qui permet d'afficher de manière simplifiée une fonction passée en paramètre soit comme une ***string*** soit comme un ***handle*** sur un fonction. Par défaut, la fonction `ezplot()` est évaluée pour le domaine [$-2\pi \leq x \leq 2\pi$]

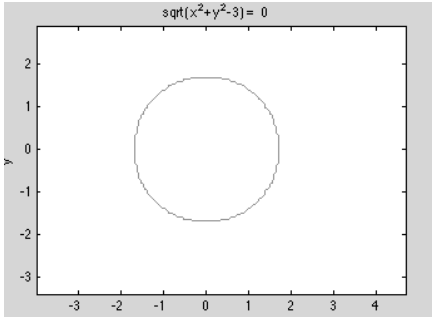
```
>> mySinX= @(x) sin(x)./x;
>> ezplot(mySinX,[-4*pi,4*pi])
```




```
>> ezplot('sin(x)/x')
```




```
>> fh = @(x,y) sqrt(x.^2 + y.^2)-3;
>> ezplot(fh)
>> axis equal
```



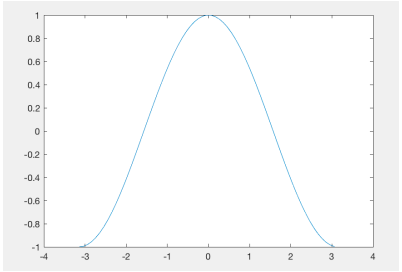
@ fonction anonyme

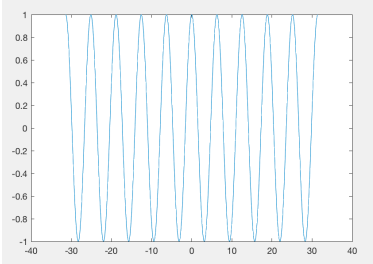
Passer une fonction anonyme en paramètre permet de ne pas forcément avoir besoin modifier la fonction qui l'appelle. Dans l'exemple ci-dessous **`myPlotFcn()`** évalue et *plot* une fonction passée en paramètre entre **`-pi:0.1:pi;`** si le range n'est pas défini, sinon utilise le range fourni.

Le corps de cette fonction ne change pas, uniquement les paramètres de la fonction!

```
function myPlotFcn(h, range)
if isa(h, 'function_handle')   % is h a function handle.
    if nargin < 2              % less than 2 input arg.
        range=-pi:0.1:pi;          % defines the range
    end
     A = h(range);             % evaluate the @ function within the range
  plot(range,A)                % Plot the resulting data.
end
```

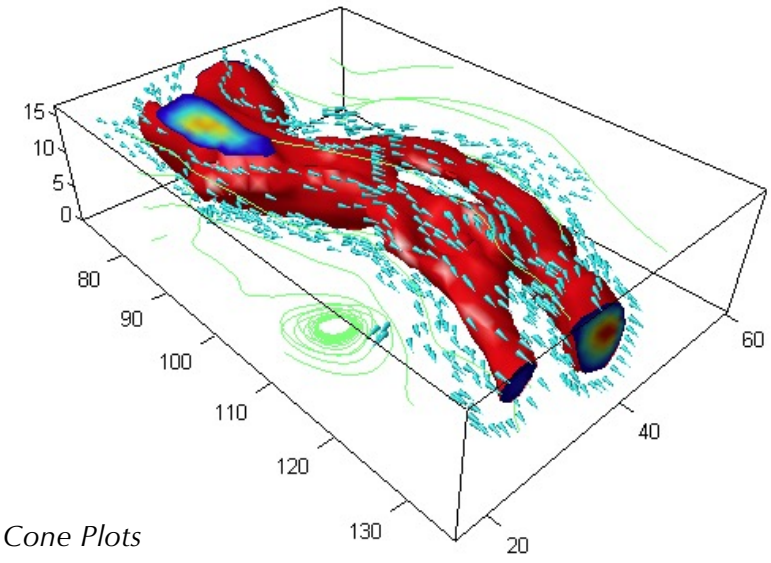
```
>> a= @sin;
>> myPlotFcn(a)
>> myPlotFcn(a,-10*pi:0.1:10*pi)
```





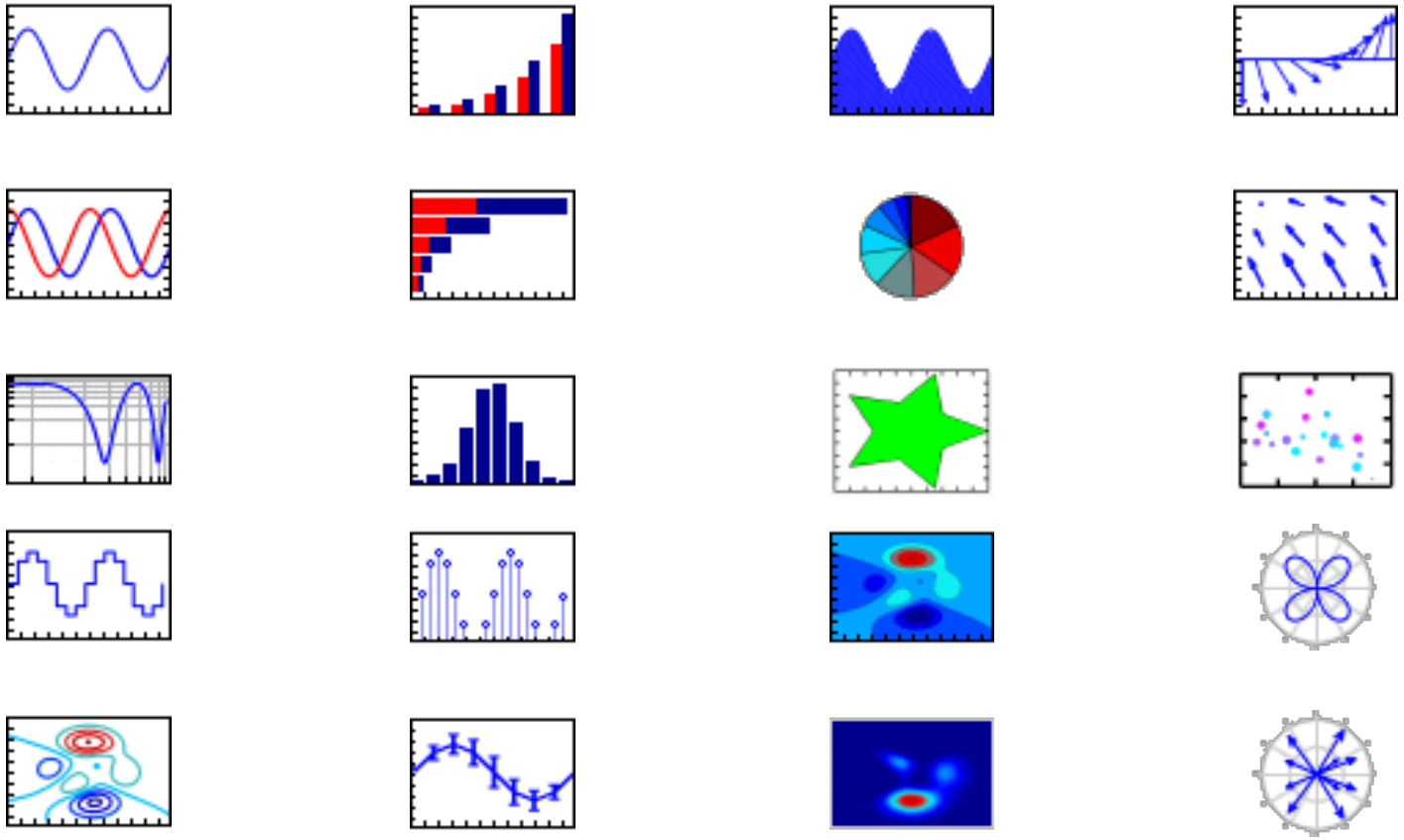
Plot *and more*

- En plus de son moteur de calcul, Matlab est un puissant outil de visualisation de données.
- Les figures Matlab peuvent êtres éditées/modifiées de manière interactive ou par programmation.
- Les figures Matlab peuvent êtres annimées.
- Les figures Matlab peuvent êtres sauvegardées dans différents formats (eps, PDF, etc).



Example – Vector Field Displayed with Cone Plots

2D-Plots



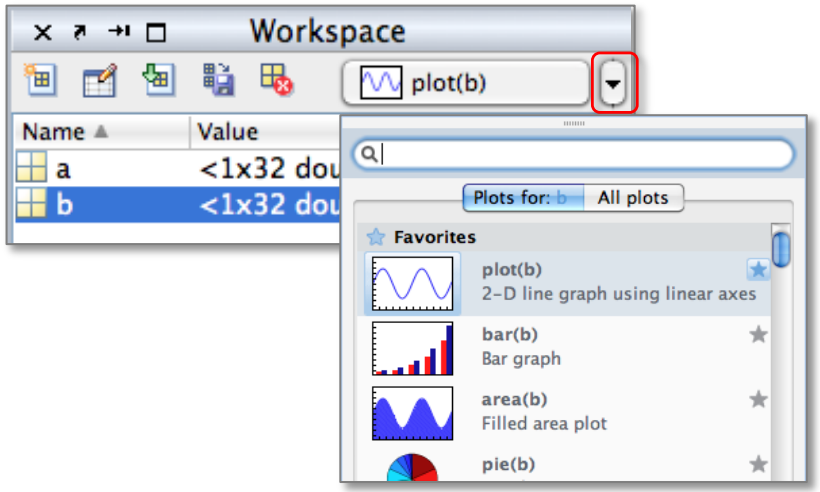
http://www.mathworks.com/help/techdoc/creating_plots/f9-53405.html

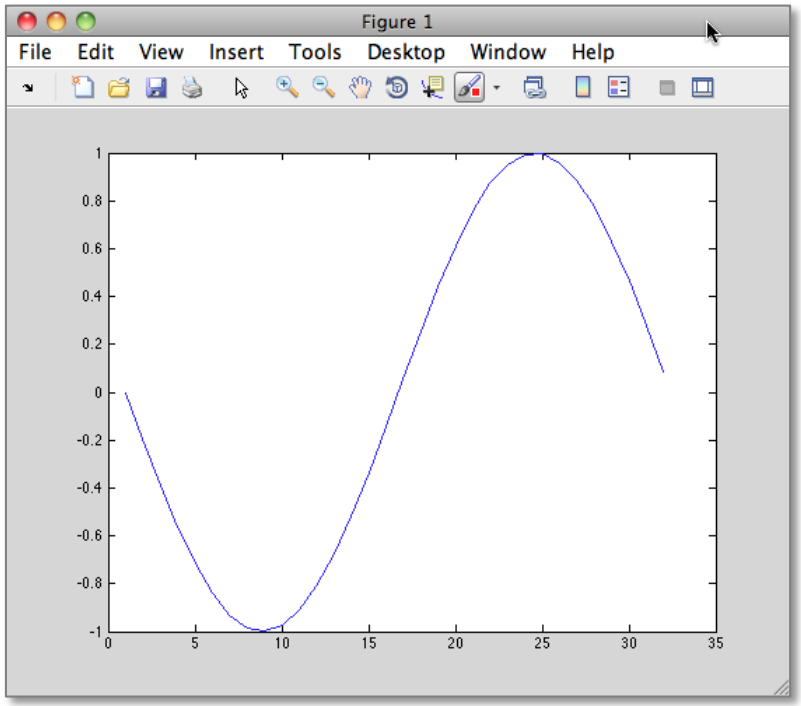
Very simple plot

La commande `plot` permet de créer une figure à partir de données numériques.

```
>> a = [-pi:0.2:pi]
>> b = sin(a);
>> plot(b);
```

ou





Demo

- Plot simple (2.4)
- Handle sur un plot (2.5)
- Edition interactive d'un plot + save
- ezplot

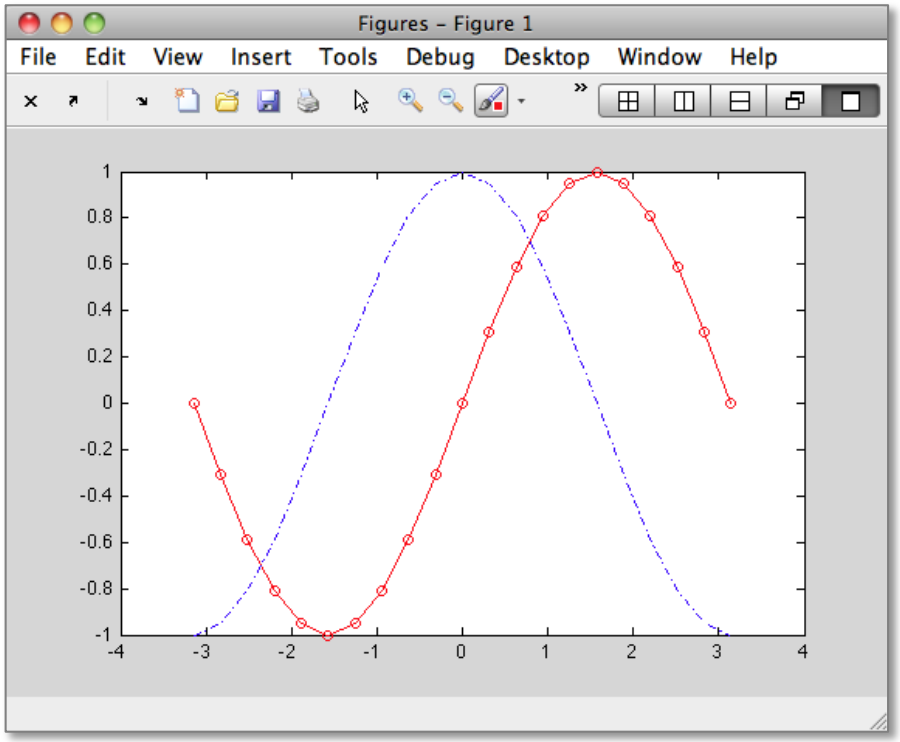
Plot()

Il existe plusieurs manières pour afficher plusieurs courbes dans le même plot. La première consiste à passer plusieurs vecteurs XY à la fonction plot.

```
>> x = -pi:pi/10:pi;
>> S = sin(x)
>> C = cos(x)
>> plot(x,S,'-ro',x,C,'-.b')
```

courbe 1

courbe 2



La deuxième consiste à superposer les courbes les une sur les autres.

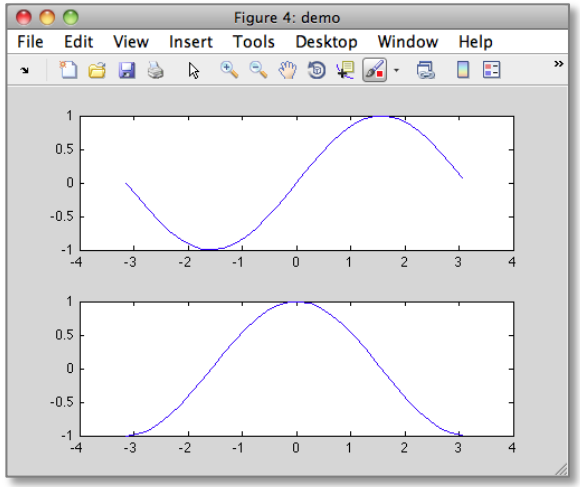
```
>> hold on
>> plot(x,S,'-ro')
>> plot(x,C,'-.b')
>> hold off
```

Propriétés

Figure

- Les *plots* vivent à l'intérieur d'une **figure**
- La référence sur la figure courante est `gcf`. Il est possible d'accéder et de modifier la figure et les plots associés au travers de cette référence (handle).
- Une figure peut contenir plusieurs plots supperposés (`hold on/off`)
- Une figure peut contenir plusieurs *sub-plots,* `Subplot(#raw,#col,idx)`

```
>> fh = figure('Name','demo')
>> x=-pi:0.2:pi
>> subplot(2,1,2); plot(x,cos(x))
>> subplot(2,1,1); plot(x,sin(x))
```



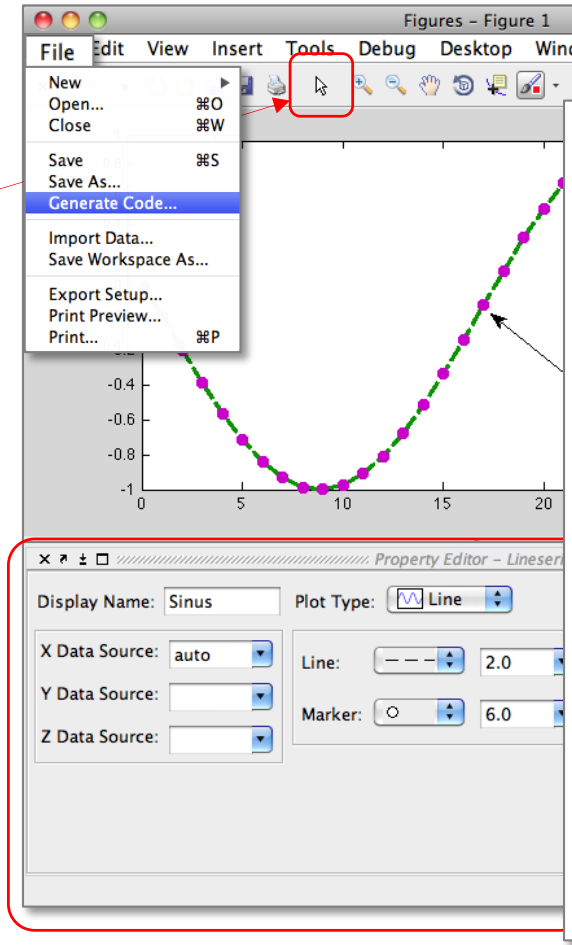
Edit plot interactively

Utiliser la flèche blanche pour séléctionner l'objet désiré sur votre figure, et modifier ses propriétés.

Une fois éditée il est possible de générer une fonction avec le code pour recréer la figure

Select object

Propriétés




```
function createfigure(Y1)
%CREATEFIGURE(Y1)
%  Y1:  vector of y data

%  Auto-generated by MATLAB on 03-May-2012 16:24:36

% Create figure
figure1 = figure('XVisual','');

% Create axes
axes1 = axes('Parent',figure1);
box(axes1,'on');
hold(axes1,'all');

% Create plot
plot(Y1,'Parent',axes1,...
    'MarkerFaceColor',[0.749019622802734 0 0.749019622802734],...
    'MarkerEdgeColor',[0.749019622802734 0 0.749019622802734],...
    'Marker','o',...
    'LineWidth',2,...
    'LineStyle','--',...
    'Color',[0 0.498039215803146 0],...
    'DisplayName','Sinus');

% Create legend
legend1 = legend(axes1,'show');
set(legend1,...
    'Position',[0.773390165111275 0.173247177658943 0.123571428571429
0.0636363636363637]);

% Create textarrow
annotation(figure1,'textarrow',[0.596984924623116 0.514572864321608],...
    [0.381352941176471 0.519607843137255],'TextEdgeColor',[1 1 1],...
    'FontWeight','bold',...
    'String',{'Very interessting point!'});
```

Commande Plot()

```
>> plot(Y)                          % plot 1 line, x 1:size(Y)
>> plot(X1,Y1,...,Xn,Yn)            % plot multiple X-Y lines
>> plot(X1,Y1,LineSpec,...,)        % specify line attributes
>> plot(...,'PropertyName',PropertyValue,...) % specify line attributes
>> plot(axes_handle,...)            % plot using specific axes handled (ref)
>> handle = plot(...)               % get a handle (ref) to plot elements
```

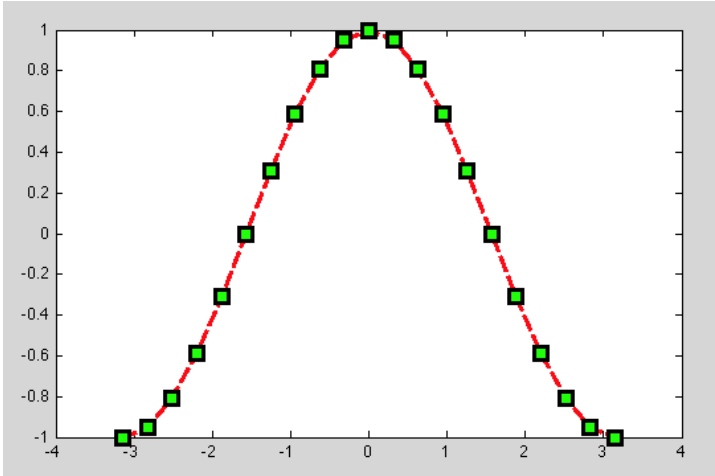
Exemple

```
>> x = -pi:pi/10:pi;
>> y = cos(x)
>> plot(x,y,'--rs','LineWidth',2,...
                  'MarkerEdgeColor','k',...
                  'MarkerFaceColor','g',...
                  'MarkerSize',10)
```

Style
--: dashed

Color
r: red

Marker
s: square

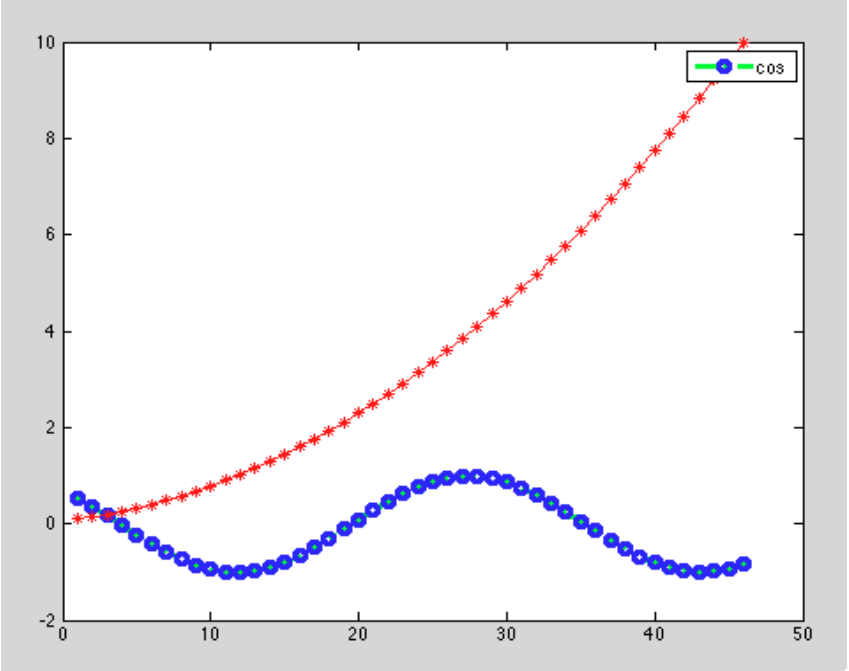


`>> doc linespec` pour l'entier des options

Plot reference

La fonction plot() retourne un **handle** (référence) sur la structure du plot. Il est ainsi possible de modifier le contenu et l'apparence du plot au travers de cette référence en utilisant `set(h,…)` et `get(h,…)`, `get(h)` retourne l'ensemble des propriétés.

```
>> x=1:0.2:10
>> h = plot(x)
>> set(h,'YData',sin(x))

>> set(h,'LineStyle','-.')
>> set(h,'LineWidth',2)
>> set(h,'Color',[1,0,0])
>> set(h,'Marker','o')
>> set(h,'MarkerEdgeColor','b')

>> set(h,'YData',cos(x))
>> ah = get(gcf,'CurrentAxes')
>> lh = legend(ah,'show');
>> set(lh,'String',{'cos'})
>> hold on
>> h1=plot(x.^2./10,'-r*');
```



gcf: référence sur la figure courante

Demo

- myEzplot (4.2)

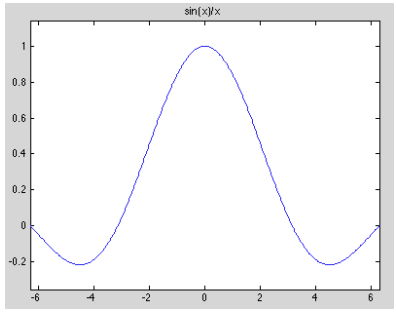
ezplot

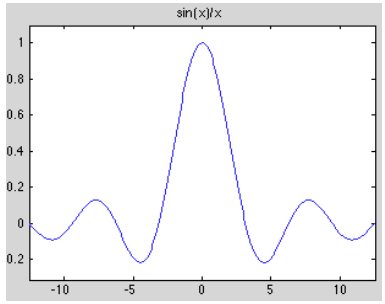
Pour la plus parts des plots, Matlab propose des manières simplifiées d'afficher une fonction passée en paramètre soit comme une *string* soit comme un *handle* sur une fonction.

La fonction est calculée par défaut sur le domaine [$-2\pi \leq x \leq 2\pi$]

```
>> ezplot('sin(x)/x')
```

```
>> ezplot('sin(x)/x',[-4*pi,4*pi])
```

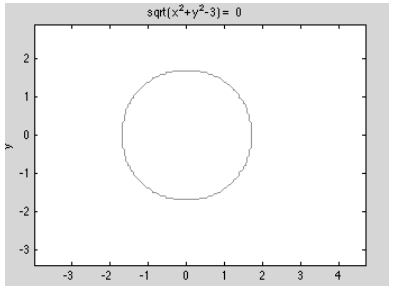




`@` defini une fonction annonyme

`fh` est un handler sur cette fonction

```
>> fh = @(x,y) sqrt(x.^2 + y.^2 -3);
% se lit: fh pointe sur la fonction sqrt(...)
>> ezplot(fh)
>> axis equal
```



Matlab 2 - recap

- Programmation très similaire à C/C++
- Fonctions
  - Contraintes sur le nom du fichier
  - Les variables sont locales
  - Plusieurs paramètres de retour possible
- Vectorization pour être rapide
- Fonction find() retourne les indexes des éléments ≠ 0
- @fonction anonyme – 1 line
- **h**=plot(), **h** est un *handle* pour modifier le plot
- Plot/ezplot pour un graphique simple