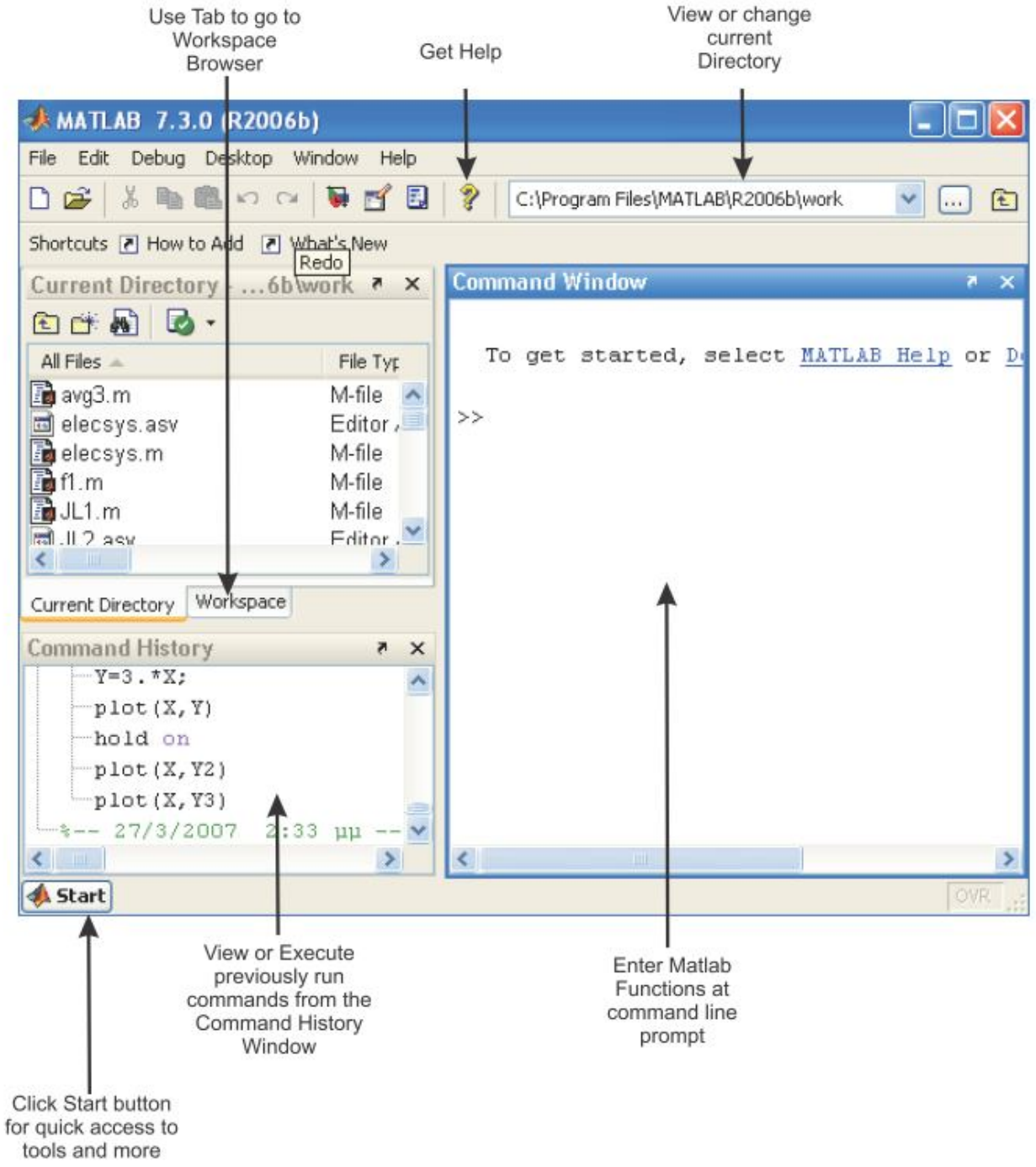
M1 BBS - EM8BBSEM

Simulation de Systèmes Biologiques

(#5)

Georges Czaplicki, UPS / IPBS-CNRS
Tél. : 05.61.17.54.04, email : cgeorge@ipbs.fr


Use Tab to go to Workspace Browser
Get Help
View or change current Directory
MATLAB 7.3.0 (R2006b)
File Edit Debug Desktop Window Help
C:\Program Files\MATLAB\R2006b\work
Shortcuts How to Add What's New
Redo
Current Directory ...6b\work
All Files File Typ
avg3.m M-file
elecsys.asv Editor
elecsys.m M-file
f1.m M-file
JL1.m M-file
Current Directory Workspace
Command History
Y=3.*X;
plot(X,Y)
hold on
plot(X,Y2)
plot(X,Y3)
%-- 27/3/2007 2:33 µµ --
Command Window
To get started, select MATLAB Help or D
>>
Start
OVR
View or Execute previously run commands from the Command History Window
Enter Matlab Functions at command line prompt
Click Start button for quick access to tools and more

```
% IMPORTANT : la commande la plus importante : 'help'


% Pour voir les possibilités du Matlab : 'helpwin' ou 'demo'
```

Introduction au langage Matlab

(copie du "diary" ; "diary off" pour terminer)

MATLAB always stores the result of the latest computation in a variable named *ans*. To store variables for a longer time we can specify our own names:

```
>> x = 5+2^2

x =

     9
```

and then we can use these variables in computations:

```
>> y = 2*x

y =

    18
```

These are examples of **assignment statements**: values are assigned to variables. **Each variable must be assigned a value before it may be used on the right of an assignment statement**. If you do not want to see the result of a statement in the Command Window, which is typically the case for intermediate statements, we can terminate the line with a semi–colon:

```
>> x = 5+2^2;
>> y = 2*x;
>> z = x^2+y^2

z =

   405
```

Notes:

- Other types of numerical variables can be defined explicitly if needed as:

  `char, single, int8, int16, int64, uint8, uint16, uint64.`

- MATLAB is accurate but does not do exact arithmetic !

```
>> 3*(5/3 - 4/3) - 1
ans =
     4.440892098500626e-16
```

- MATLAB identifies i, j, pi also as numbers and to some extent Inf and NaN:

```
>>  pi
ans = 3.1416

>> (-1)^(0.5)
ans = 0.0000 + 1.0000i

>> log(0)
ans =  -Inf

>> 0/0
ans = NaN
```

Logical variables and operators

A logical variable can be "true" or "false", or one and zero in binary system.

```
>>S = 2>4
S =     0
```

Boolean algebra can be done in MATLAB using the logical operators AND (&) , OR (|) and NOT ($\sim$). For example given an arbitrary boolean variable S:

```
>> S | ~S
ans =     1
```

Note: MATLAB implicitly "casts" data types to avoid syntax errors:

- Logical values are converted to 1 and 0 when used as numbers.

  ```
  >> (2 < 3) * 3
  ans =     3
  >> true * 3
  ans =     3
  ```

- All numbers hold "true" boolean value when used in logical expressions, except for 0 itself which is "false".

  ```
  >> false | -4.03
  ans =    1
  ```

Vecteurs :

```
% vecteur en 3D
» v=[1 2 3]
v =
     1     2     3

% un autre vecteur
» w=[4 5 6]
w =
     4     5     6

% pas d'affichage
» w=[4 5 6];
```

```
% produit scalaire
» v*w
??? Error using ==> *
Inner matrix dimensions must agree.
» v*w'
ans =
    32
» w*v'
ans =
    32

% la norme
» norm(v)
ans =
    3.7417
» norm(w)
ans =
    8.7750
```

```
% vecteur en tant qu'une colonne
» z=[7;8;9]
z =
     7
     8
     9
» v*z
ans =
    50

% cos(phi) entre vecteurs v et z
» cosphi=v*z/(norm(v)*norm(z))
cosphi =
    0.9594

% angle en radians
» acos(cosphi)
ans =
    0.2859

% angle en degrés
» acos(cosphi)*180/pi
ans =
   16.3801
```

Matrices

```
% créer une matrice
» a=[1 2 3;4 5 6;7 8 9]
a =
     1     2     3
     4     5     6
     7     8     9

% plus facile de voir ce qu'on entre
» a=[1 2 3
4 5 6
7 8 9];

% un élément de la matrice
» a(1,3)

ans =
     3
```

```
% sous-matrice de a par extraction
» b=a(1,:)
b =
     1     2     3

» c=a(:,1)
c =
     1
     4
     7

% sous-matrice par suppression d'une colonne
» c=a;
» c(:,2)=[]

c =
     1     3
     4     6
     7     9
```

```
% certaines déclarations prédéfinies
» m=eye(3)

m =

     1     0     0
     0     1     0
     0     0     1

» m=ones(4,3)

m =

     1     1     1
     1     1     1
     1     1     1
     1     1     1

» m=zeros(3,4)

m =

     0     0     0     0
     0     0     0     0
     0     0     0     0
```

Concatenating matrices are straightforward MATLAB as long as their dimensions are consistent.

```
>> A = [1, 2, 3];B = [4, 5, 6];
>> C = [A, B]
C =
     1     2     3     4     5     6
>> C = [A; B]
C =
     1     2     3
     4     5     6
```

```
% opérations : produit matriciel
» x=a*a

x =
    30    36    42
    66    81    96
   102   126   150

% équivalent à :
» x=a^2

x =
    30    36    42
    66    81    96
   102   126   150
```

```
% produit élément par élément
» x=a.^2

x =
     1     4     9
    16    25    36
    49    64    81

% division élément par élément
» x=x./a

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

```
% matrice inverse
» y=inv(x)

Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 2.055969e-018.

y =

  1.0e+016 *

   -0.4504    0.9007   -0.4504
    0.9007   -1.8014    0.9007
   -0.4504    0.9007   -0.4504

% il y a un problème, mais essayons quand même de calculer
% le produit de cette matrice et de son inverse
» x*y

ans =

     2     0     0
     8     0    -4
    16     0     8

% le résultat ne donne pas la matrice d'identité. Pourquoi ?
```

```
» det(x)

ans =

      0

% C'est une matrice singulière !
```

```
% Quel est le rang de cette matrice ?
» rank(x)

ans =

     2
% les lignes (ou colonnes) sont linéairement dépendantes.
```

Comment vérifier la dépendance linéaire ?

$$\lambda_1 \cdot [1 \quad 2 \quad 3] + \lambda_2 \cdot [4 \quad 5 \quad 6] + \lambda_3 \cdot [7 \quad 8 \quad 9] = [0 \quad 0 \quad 0]$$

$$\left\{ \begin{matrix} 1\lambda_1 + 4\lambda_2 + 7\lambda_3 = 0 \\ 2\lambda_1 + 5\lambda_2 + 8\lambda_3 = 0 \\ 3\lambda_1 + 6\lambda_2 + 9\lambda_3 = 0 \end{matrix} \right\} \rightarrow \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} \cdot \begin{bmatrix} \lambda_1 \\ \lambda_{21} \\ \lambda_3 \end{bmatrix} = \mathbf{0}$$

$$[\lambda_1 \quad \lambda_2 \quad \lambda_{31}] = [1 \quad -2 \quad 1] \cdot const$$

```
% essayons encore une fois…
% changeons un seul élément : x(3,3)

» x=[1 2 3
     4 5 6
     7 8 10]

x =

     1     2     3
     4     5     6
     7     8    10

% le determinant…
» det(x)

ans =

    -3

% si det(x) n'est pas 0, alors le rang sera égal à 3
» rank(x)

ans =

     3
```

```
% l'inverse de la matrice x
» y=inv(x)

y =

   -0.6667   -1.3333    1.0000
   -0.6667    3.6667   -2.0000
    1.0000   -2.0000    1.0000

% voyons ce que ça donne…
» x*y

ans =

    1.0000         0   -0.0000
         0    1.0000         0
         0         0    1.0000

» y*x

ans =

    1.0000         0    0.0000
         0    1.0000         0
   -0.0000   -0.0000    1.0000
```

Fonctions des matrices

```
% Nous savons calculer l'expression f(x), où x est un chiffre
% (un scalaire). Mais comment faire si x est une matrice ?


% exemple : calculer exp(-A), où A est une matrice

» A=[4 2 1
     3 5 2
     2 4 6]

A =

     4     2     1
     3     5     2
     2     4     6
```

% décomposition de la matrice A en vecteurs et en valeurs propres

```
» [V,D]=eig(A)

   V =

       0.3284    0.4063    0.4560
       0.5430    0.3274   -0.7415
       0.7728   -0.8531    0.4922


   D =

       9.6605         0         0
            0    3.5121         0
            0         0    1.8273
```

```
% vérifions si la décomposition est correcte :
% est-ce que A est égale au produit V * D * inv(V) ?

» B=V*D*inv(V)

B =

    4.0000    2.0000    1.0000
    3.0000    5.0000    2.0000
    2.0000    4.0000    6.0000

% si oui, la différence devrait être 0

» A-B

ans =

  1.0e-014 *

   -0.5329    0.0444   -0.1332
    0.3553    0.0888    0.3553
    0.0222   -0.0888    0.0888
```

```
% si A=V*D*inv(V), alors f(A)=V*f(D)*inv(V)

% calculons exp(-Dii) sur la diagonale de D
» for i=1:3; D(i,i)=exp(-D(i,i));end;
» D

D =

    0.0001         0         0
         0    0.0298         0
         0         0    0.1608


% reconstruisons la matrice dans le repère initial
» C=V*D*inv(V)


C =

    0.0762   -0.0501    0.0029
   -0.0938    0.0884   -0.0222
    0.0430   -0.0630    0.0261

% Et maintenant la même chose, mais de façon plus simple
» expm(-A)

ans =

    0.0762   -0.0501    0.0029
   -0.0938    0.0884   -0.0222
    0.0430   -0.0630    0.0261
```

```
% Tableaux utiles : un vecteur ligne de 101 valeurs
% également espacées
» t=0:pi/100:pi;
» size(t)
ans =
     1   101


% à quoi correspond « pi » ?
» pi

ans =
    3.1416

% Attention : tous les nombres sont en double précision,
% seul le format de sortie varie

» format long

» pi

ans =
   3.14159265358979


» help format
```

```
FORMAT Set output format.
   All computations in MATLAB are done in double precision.
   FORMAT may be used to switch between different output
   display formats as follows:
     FORMAT          Default. Same as SHORT.
     FORMAT SHORT    Scaled fixed point format with 5 digits.
     FORMAT LONG     Scaled fixed point format with 15 digits.
     FORMAT SHORT E  Floating point format with 5 digits.
     FORMAT LONG E   Floating point format with 15 digits.
     FORMAT SHORT G  Best of fixed or floating point format with 5
                      digits.
     FORMAT LONG G   Best of fixed or floating point format with
                      15 digits.
     FORMAT HEX      Hexadecimal format.
     FORMAT +        The symbols +, - and blank are printed
                     for positive, negative and zero elements.
                     Imaginary parts are ignored.
     FORMAT BANK     Fixed format for dollars and cents.
     FORMAT RAT      Approximation by ratio of small integers.

   Spacing:
     FORMAT COMPACT  Suppress extra line-feeds.
     FORMAT LOOSE    Puts the extra line-feeds back in.
```

```
% faisons le point sur les variables utilisées jusqu'à ici
» who
Your variables are:
a          b          cosphi     v          x
ans        c          t          w          z

» whos
  Name          Size              Bytes  Class
  a             3x3                  72  double array
  ans           1x1                   8  double array
  b             1x3                  24  double array
  c             3x2                  48  double array
  cosphi        1x1                   8  double array
  t             1x101               808  double array
  v             1x3                  24  double array
  w             1x3                  24  double array
  x             3x3                  72  double array
  z             3x1                  24  double array
Grand total is 139 elements using 1112 bytes


% on peut tout nettoyer
» clear
» who
```

```
% graphiques 2D : comment tracer la courbe x=sin(t)
» t=0:pi/100:2*pi;
» x=sin(t);
» plot(t,x)

% toutes les informations sur 'plot' :
» help plot

PLOT   Linear plot.
PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
then the vector is plotted versus the rows or columns of the
matrix, whichever line up.  If X is a scalar and Y is a vector,
length(Y) disconnected points are plotted.

PLOT(Y) plots the columns of Y versus their index.
If Y is complex, PLOT(Y) is equivalent to PLOT(real(Y),imag(Y)).
In all other uses of PLOT, the imaginary part is ignored.
```

```
Various line types, plot symbols and colors may be obtained with
PLOT(X,Y,S) where S is a character string made from one element
from any or all the following 3 colunms:

y     yellow          .     point               -     solid
m     magenta         o     circle              :     dotted
c     cyan            x     x-mark              -.    dashdot
r     red             +     plus                --    dashed
g     green           *     star
b     blue            s     square
w     white           d     diamond
k     black           v     triangle (down)
                      ^     triangle (up)
                      <     triangle (left)
                      >     triangle (right)
                      p     pentagram
                      h     hexagram

For example, PLOT(X,Y,'c+:') plots a cyan dotted line with a plus
at each data point; PLOT(X,Y,'bd') plots blue diamond at each data
point but does not draw any line.
```

```
% traçons la courbe en rouge et changeons les axes
» plot(t,x,'r')
» axis([0 2*pi -2 2])

% ajoutons une grille
» grid on
```

Making Plots

Matlab provides a variety of functions for displaying data as 2-D or 3-D graphics.

For 2-D graphics, the basic command is:

```
plot(x1, y1, 'line style', x2, y2, 'line style'...)
```

This command plots vector `x1` versus vector `y1`, vector `x2` versus vector `y2`, etc. on the same graph. Other commands for 2-D graphics are: **polar, bar, stairs, loglog, semilogx,** and **semilogy**.

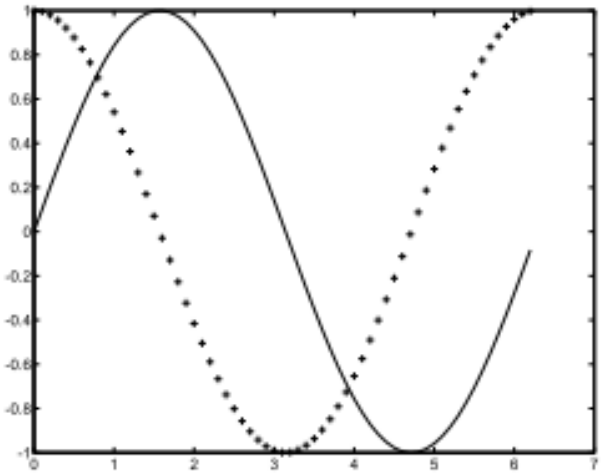
Example 1: Plot $y_1 = \sin(x)$ and $y_2 = \cos(x)$ with x in $[0, 2\pi]$ on the same graph. Use a solid line for $\sin(x)$ and the symbol + for $\cos(x)$. The first step is to define a set of values for `x` at which the functions will be defined.

```
x = 0 : 0.1 : 2*pi;
y1 = sin(x);
y2 = cos(x);
plot(x, y1, '-', x, y2, '+')
```

Note: Ordinarily *Matlab* prints the results of each calculation right away. Placing **;** at the end of each line directs *Matlab* to not print the values of each vector.

Another way to get multiple plots on the same graph is to use the **hold** command to keep the current graph, while adding new plots. Another **hold** command releases the previous one. For example, the following statements generate the same graph as in **Example 1**. *Matlab* remembers that the vector `x` is already defined.

```
plot(x, sin(x), '-')
hold
plot(x, cos(x), '+')
```

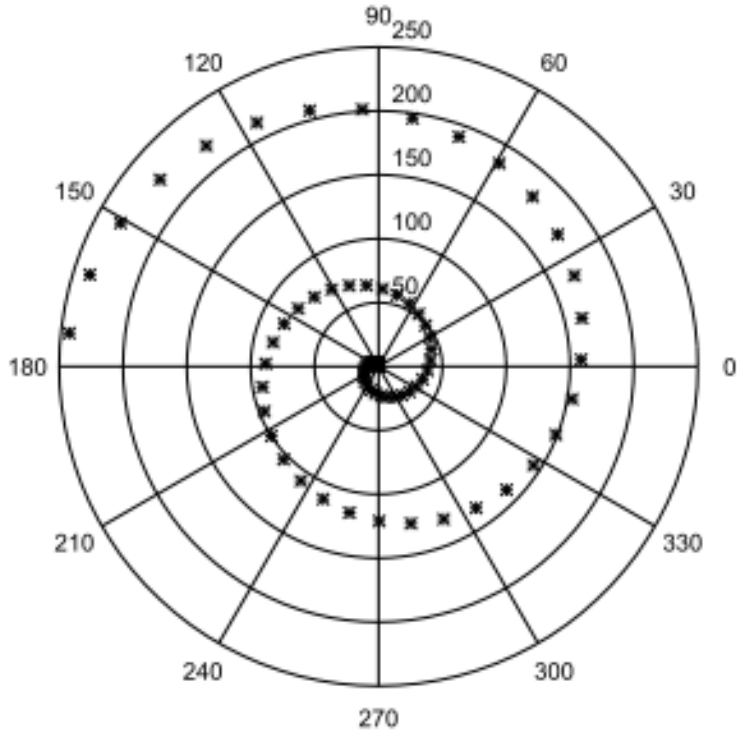


Example 1

The next example shows how *Matlab* generates a spiral using the polar coordinate system.

Example 2: Plot $\rho = \theta^2$ with $0 \leq \theta \leq 5\pi$ in polar coordinates.

```
theta = 0: 0.2: 5*pi;
rho = theta.^2;
polar(theta, rho, '*')
```



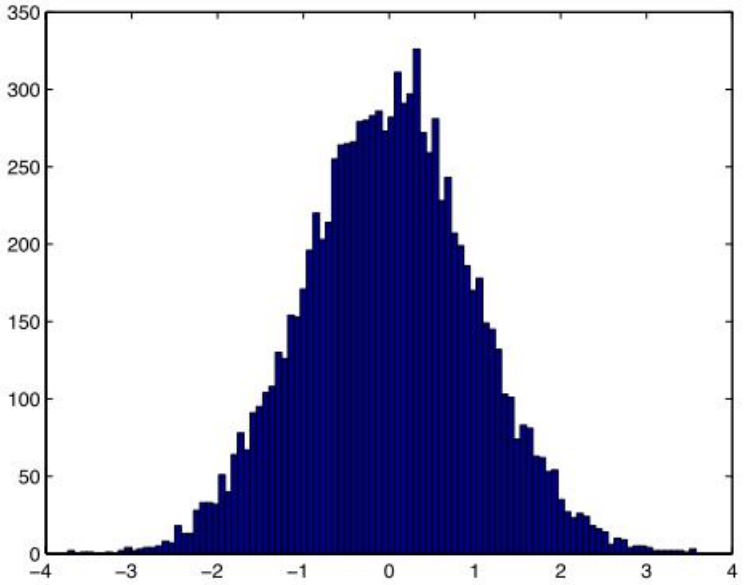
Example 2

The hist command

Histogram plots are one of the other basic types of plots in MATLAB and can be produced using the hist function. The easiest form of using this function is hist(x), in which **x** denotes a vector. The command divides the range of the values in **x** to ten bins and plots the distribution of the element counts in each bin using bar plots.

An additional scalar parameter can be added afterwards in order to tune the number of bins, for example the following commands produce $10,000$ normally distributed random numbers, plot their histogram using 100 bins:

```
>> x = randn(10000,1);
>> hist(x, 100)
```



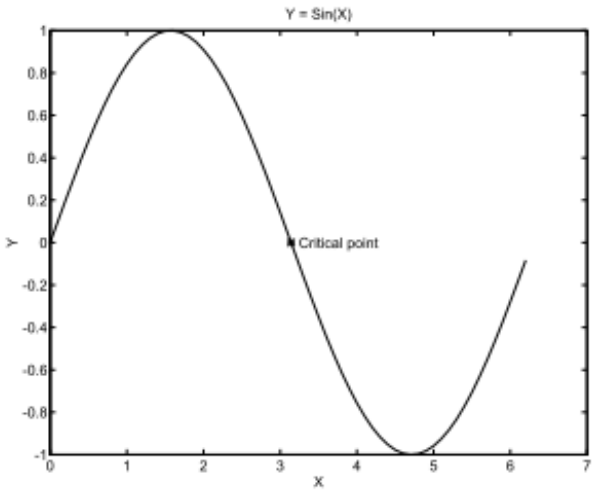
Title and Labels

You can add a title and labels for the axes with the commands; **title, xlabel, ylabel** and **zlabel**. You can also add contour labels to a contour plot by the command **clabel**. Other text can be added to the graph by using the **text** or **gtext** commands. With **text**, you specify a location where left edge of a text string is placed. With **gtext**, you position the text string with the mouse.

Here is an example which adds titles and labels to the graph of $f(x) = sin(x)$.

Example 4: Plot $y = sin(x); 0 \leq x \leq 2\pi$, with appropriate labels.

```
x = 0: 0.1: 2*pi; plot(x,sin(x))
title('Y = Sin(X)')
xlabel('X');   ylabel('Y')
hold
plot(pi,0,'*')
text(pi + 0.1, 0, 'Critical point')  % or gtext('Critical point')
hold
```

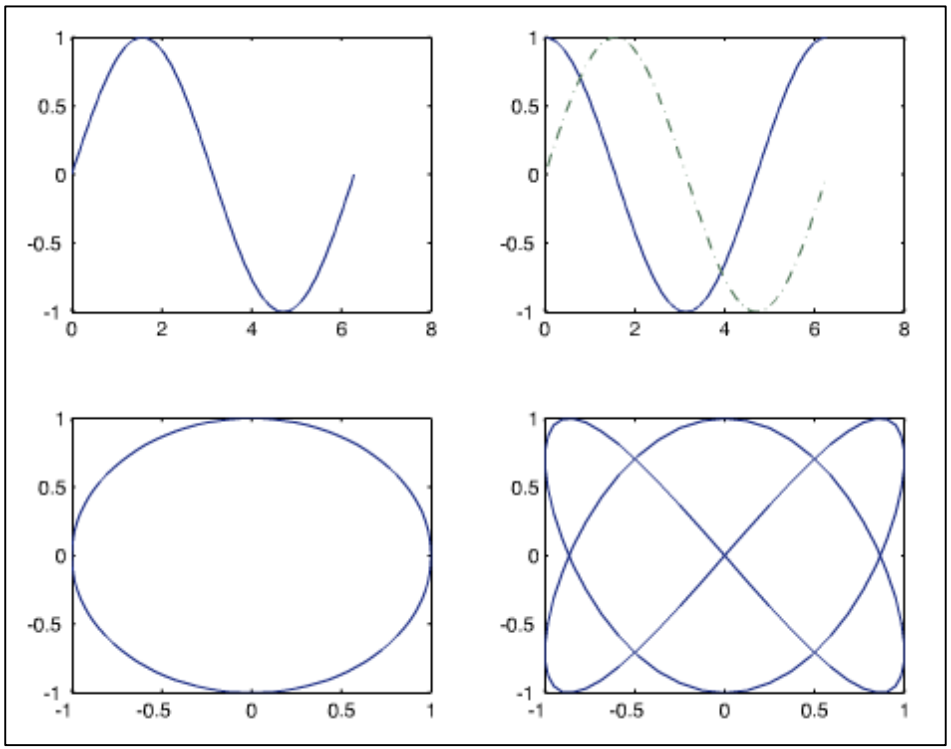


Example 4

- **Multiple Plots**

The command **subplot(m,n,p)** breaks the graph (or figure) window into an m-by-n matrix of small rectangular panes. The value of **p** is the pane for the next plot. Panes are numbered from left to right, top to bottom. To return to the default single graph per window, use either **subplot(1,1,1)** or **clf**.

You can have more than one graphics window on an X display. The *Matlab* command, **figure** opens a new window, numbering each new window. You can then use commands such as **clf, figure(h),** or **close** to manipulate the figure windows.

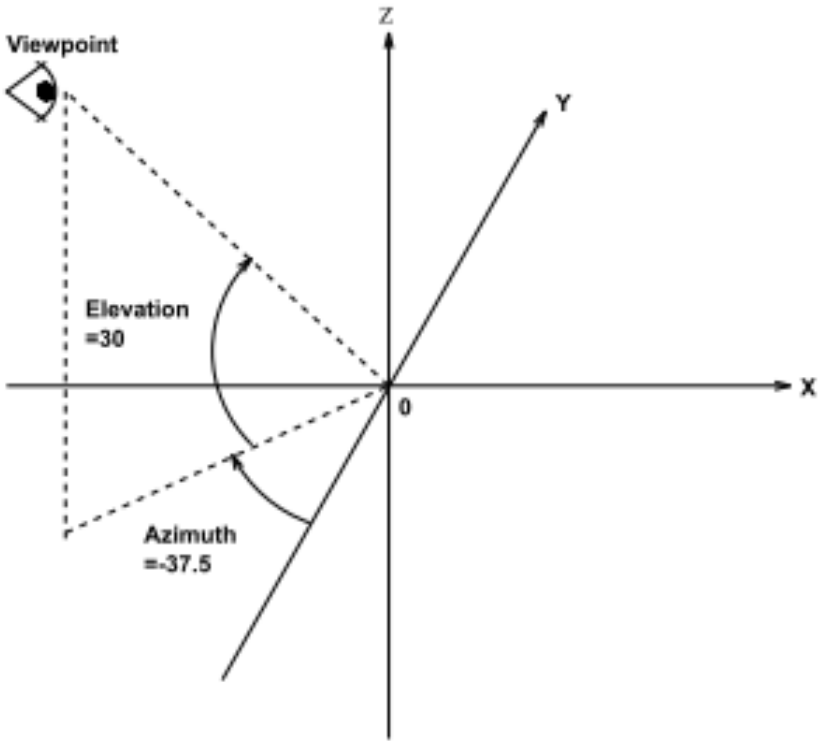


- **Viewpoint**

  You can set the angle of view of a 3-D plot with the command:

  ```
  view(az,el)
  ```

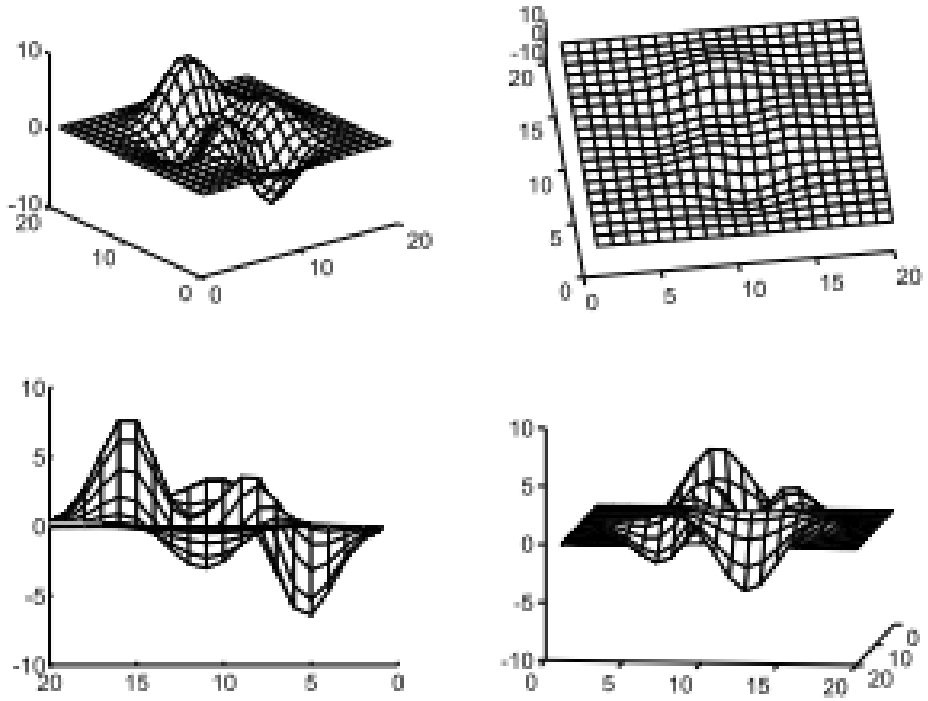
  `az` is the azimuth and `el` is the elevation of the viewpoint, both in degrees. See the *viewpoint* figure for an illustration of azimuth and elevation relative to the Cartesian coordiate system.



Default Viewpoint

Example 5: View the internal *Matlab* **peaks** matrix from 4 different viewpoints. The first one, (**view(-37.5,30)**, is the default viewpoint.

```
subplot(2,2,1); mesh(peaks(20)); view(-37.5,30)
subplot(2,2,2); mesh(peaks(20)); view(-7,80)
subplot(2,2,3); mesh(peaks(20)); view(-90,0)
subplot(2,2,4); mesh(peaks(20)); view(-7,-10)
```



Example 5

- **Controlling Axes**

  You can control the scaling and appearance of plot axis with the **axis** function. To set scaling for the x- and y- axes on the current 2-D plot, use this command:

  ```
  axis([xmin xmax ymin ymax])
  ```

  To scale the axes on 3-D plot, use this:

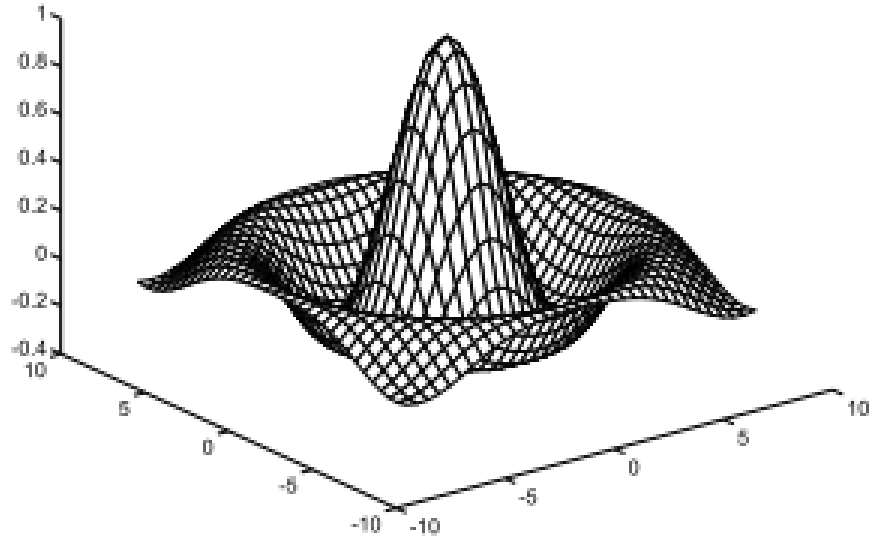
  ```
  axis([xmin xmax ymin ymax zmin zmax])
  ```

  In addition,

axis('auto')	returns the axis scaling to its default where the best axis limits are computed automatically;
axis('square')	makes the current axis box square in size, otherwise a circle will look like an oval;
axis('off')	turns off the axes
axis('on')	turns on axis labeling and tic marks.

```
% Graphiques 3D


» clear

% un exemple de la fonction sin(x)/x en deux dimensions (y=x).
% Tout d'abord, un vecteur ligne :
» x=-8:0.5:8;

% création de deux matrices carrées
% (meshgrid(x) = meshgrid(x,x)
» [X,Y]=meshgrid(x);

» whos
  Name        Size              Bytes  Class

  X          33x33              8712  double array
  Y          33x33              8712  double array
  x           1x33               264  double array
Grand total is 2211 elements using 17688 bytes

% MATLAB est sensible à la casse (x != X)
```

```
% calculons la distance par rapport au centre
% pour tous les nœuds de la grille
» R=sqrt(X.^2+Y.^2)+eps;

% que represente 'eps' ?
» eps
ans =
    2.220446049250313e-016

% calcul de la fonction pour le graphe
» Z=sin(R)./R;
» mesh(X,Y,Z)   (==> OK pour Matlab, mais pas pour FreeMat !)
» surf(X,Y,Z)
```



```
» help meshgrid

MESHGRID   X and Y arrays for 3-D plots.
[X,Y] = MESHGRID(x,y) transforms the domain specified by vectors
x and y into arrays X and Y that can be used for the evaluation
of functions of two variables and 3-D surface plots.
The rows of the output array X are copies of the vector x and
the columns of the output array Y are copies of the vector y.

[X,Y] = MESHGRID(x) is an abbreviation for [X,Y] = MESHGRID(x,x).
[X,Y,Z] = MESHGRID(x,y,z) produces 3-D arrays that can be used to
evaluate functions of three variables and 3-D volumetric plots.

For example, to evaluate the function  x*exp(-x^2-y^2) over the
range  -2 < x < 2,  -2 < y < 2,

     [X,Y] = meshgrid(-2:.2:2, -2:.2:2);
     Z = X .* exp(-X.^2 - Y.^2);
     mesh(Z)

MESHGRID is like NDGRID except that the order of the first two
input and output arguments are switched (i.e., [X,Y,Z] =
MESHGRID(x,y,z) produces the same result as [Y,X,Z] =
NDGRID(y,x,z)).  Because of this, MESHGRID is better suited to
problems in cartesian space, while NDGRID is better suited to N-D
problems that aren't spatially based.  MESHGRID is also limited to
2-D or 3-D.

See also SURF, SLICE, NDGRID.
```

```
% deux autres présentations possibles :
» surf(X,Y,Z)
» contour(X,Y,Z)

% impression ou sauvegarde dans un fichier
» print -dtiff -r300 Fig.tiff;

% ('help print' pour toutes les options)
```

For 3-D graphics, the most commonly used commands are:

```
plot3(x1, y1, z1, 'line style', x2, y2, z2, 'line style'...)
contour(x,y,Z)
mesh(x,y,Z), surf(x,y,Z)
```

The first statement is a three-dimensional analogue of `plot()` and plots lines and points in 3-D. The second statement produces contour plots of the matrix `Z` using vectors `x` and `y` to control the scaling on the `x-` and `y-` axes. For surface or mesh plots, you use the third statement where `x`, `y` are vectors or matrices and Z is a matrix. Other commands available for 3-D graphics are: **pcolor, image, contour3, fill3, cylinder,** and **sphere**.

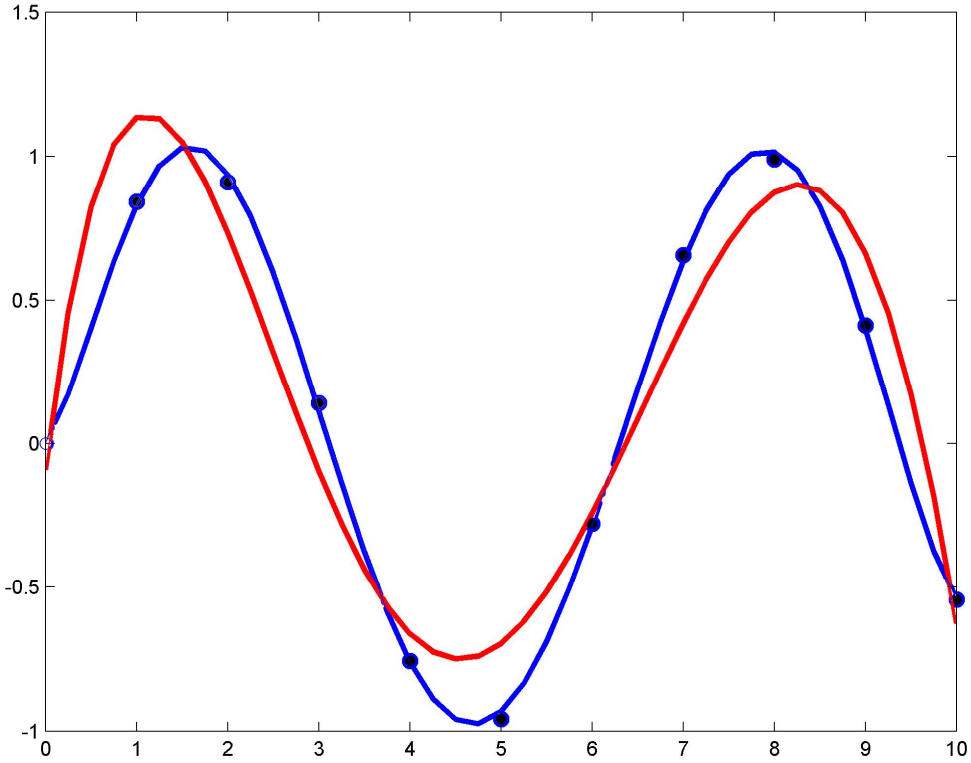
Cas de données discrètes - interpolation

Fonction `interp1`

```
x  =  0:10;
y  =  sin(x);
xi =  0:.25:10;
yi =  interp1(x,y,xi);   (interplin1 pour FreeMat)
plot(x,y,'o',xi,yi)
```

Fonctions polyfit et polyval

```
x  =  0:10;
y  =  sin(x);
xi =  0:.25:10;
n  =  5;    (ordre du polynôme ; meilleur choix ?)
p  =  polyfit(x,y,n);
y5 =  polyval(p,xi);
n  =  7;
p  =  polyfit(x,y,n);
y7 =  polyval(p,xi);
plot(x,y,'o',xi,y5,'-r',xi,y7,'-b')
```



MATLAB functions

Simple Built-in Functions

Here are some basic instances of the ready MATLAB functions:

- Trigonometric functions: `sin`, `cos`, `tan`, `asin`, `acos`, `atan` (their arguments should be in radians).
- Exponential: `exp`, `log`, `log2`, `log10`
- Random number generator: `rand`, `randn`, `randperm`, `mvnrnd`
- Data analysis: `min`, `max`, `mean`, `median`, `std`, `var`, `cov`
- Linear algebra: `norm`, `det`, `rank`, `inv`, `eig`, `svd`, `null`, `pinv`
- Strings: `strcmp`, `strcat`, `strfind`, `sprintf`, `sscanf`, `eval`
- Files: `save`, `load`, `csvwrite`, `csvread`, `fopen`, `fprintf`, `fscanf`
- Other: `abs`, `sign`, `sum`, `prod`, `sqrt`, `floor`, `ceil`, `round`, `sort`, `find`
- Master key: `help`, `doc`, and the external function Google!

Scripts et fonctions

```
% SCRIPT : programme sans arguments d'entrée ou de sortie,
%          utilise les données de l'espace de travail
% FONCTION : accepte des arguments et renvoie des valeurs,
%            les variables internes sont locales


% invocation de l'éditeur intégré
» edit


% taper dans la fenêtre de l'éditeur le contenu du programme :


% Programme traçant la surface z=x*y dans le carré [-1,1]x[-1,1]
x=-1:0.05:1;
y=x';
z=y*x;
mesh(x,y,z)
```

```
% sauvegarder sous le nom 'my_script.m'


% vérification
» type my_script

% Programme traçant la surface z=x*y dans le carré [-1,1]x[-1,1]
x=-1:0.05:1;
y=x';
z=y*x;
mesh(x,y,z)



% exécution du script
» clear
» my_script
» whos

  Name        Size              Bytes  Class
  x           1x41                328  double array
  y          41x1                 328  double array
  z          41x41              13448  double array
Grand total is 1763 elements using 14104 bytes
```

```
% exemple de function : my_norme.m

» type my_norme

% Fonction calculant la norme du max d'un vecteur
function z=my_norme(x)
z=0;
for n=1:length(x)
   if(abs(x(n))) > z
      z=abs(x(n));
   end
end


% exécution
» clear
» x=[-1 3 -4 2];
» y=my_norme(x)
y =
     4

» whos
  Name      Size            Bytes  Class
  x         1x4                32  double array
  y         1x1                 8  double array
Grand total is 5 elements using 40 bytes
```

Fonctions en Matlab

1. M-files

```
function [y1, y2, …, ym] = toto(x1, x2, …, xn)
…
```

Exemple avec une fonction utile (nargin) :

```
function c  =  testarg1(a,b)
if  (nargin ==  1)
c  =  2*a;
elseif  (nargin  ==  2)
c  =  a  +  b;
end
```

Instruction	Description
`nargin`	nombre d'arguments d'entrée d'une fonction
`nargout`	nombre d'arguments de sortie d'une fonction
`error`	interrompt l'exécution de la fonction, affiche le message d'erreur et retourne dans le programme appelant.
`warning`	imprime le message mais ne retourne pas dans le programme appelant
`pause`	interrompt l'exécution jusqu'à ce que l'utilisateur tape un `return`
`pause(n)`	interrompt l'exécution pendant n secondes.
`pause off`	indique que les `pause` rencontrées ultérieurement doivent être ignorées, ce qui permet de faire tourner tous seuls des scripts requièrant normalement l'intervention de l'utilisateur.
`break`	sort d'une boucle `while` ou `for`.
`return`	retourne dans le programme appelant sans aller jusqu'à la fin de la fonction.

As a matter of convention, you should always call the m-file and the function by the same name. However, make sure that it does not conflict with a name that is already taken by another function. Otherwise the original function is "overwritten", i.e., cannot be called any more.

2. Fonctions inline

```
>>angle=inline('180*atan(y/x)/pi')
angle  =
Inline  function:
angle(x,y) =  atan(y/x)

>>angle(5,4)
ans   =
0.6747
```

Flow control

Checking conditions: if, else, ifelse

Conditional statements check a given expression and based on the outcome execute certain parts of the code. You can also check several conditions in a sequential order with `elseif` statements.

```
if expression
    statements
elseif expression
    statements
elseif expression
    statements
...
else
    statements
end
```

Example: the following function computes the factorial $n! = \Pi_{k=1}^{n} k$ by recursive function calls.

```
function f = fac(n)
if n == 0
   f = 1;
else
   f = n*fac(n-1);
end
```

Iterations: for and while loops

Iterations are defined by for and while loops. The difference between both is that for-loops have a fixed number of cycles. While-loops stop until a certain condition is matched. The for loop has the general syntax:

```
for variable = vector
    statements
end
```

Example: the following function computes the Fibonacci series.

```
function f = fibonacci(n)
f = zeros(n,1);
f(1:2) = [1 1];
for l=3:n
   f(l) = f(l-2) + f(l-1);
end
```

The while loop format is

```
while condition
    statements
end
```

Example: the following function sorts the elements of a vector in ascending order.

```
function v = gsort(v)
pos = 2;
while pos <= length(v)
  if v(pos)>v(pos-1) % move to the next position
    pos = pos + 1;
  else % swap the values
      foo = v(pos-1);
      v(pos-1) = v(pos);
      v(pos) = foo;
      if pos > 2 % decrease position
          pos = pos - 1;
      end
  end
end
```

For and while loops can be interrupted with the `break` statement. Generally, `for` and `while` loops should be avoided in MATLAB as they are very slow. Instead you should try to use built-in MATLAB functions or vectorize the code with the dot-operator.

Optimization

Optimization plays a central role in parameter estimation. Whenever you want to find a parameter set that leads to an optimal fit of your model to some measurements, you will have to employ some kind of optimization procedure.

Finding roots and minima

Assume you want to find values for the arguments of a function for which it is zero. These are called the roots of a function. The `fzero` function can be used to find the root of a continuous function of one variable. You can either specify your own function

```
x = fzero(@myfun,x0);
```

where myfun is an M-file function that you defined before hand, or you specify an inline function (in MATLAB this is called an anonymous function) such as:

```
x = fzero(@(x)sin(x*x),x0);
```

Beside an input function, `fzero` requires a starting value, from where the search for the root starts. For example, the nearest roots of $\sin(x^2)$ starting from $x_0 = 2$ and $x_0 = 3$ are:

```
>> x = fzero(@(x)sin(x*x),2)

x =

    1.7725

>> x = fzero(@(x)sin(x*x),3)

x =

    3.0700
```

Note, that `fzero` only finds a root if your function is *continuous* and *crosses* the x-axis at the root (i.e. changes sign).

The minimum of a continuous function of one variable is computed by the `fminbnd` function:

```
x = fminbnd(fun,x1,x2)
```

where `x1` and `x2` are the boundaries within which the minimum is attained and `fun` is a function handle to an M-file function or a build in MATLAB function. For example,

```
>> x = fminbnd(@cos,3,4)

x =

    3.1416
```

find a numerical approximation to π. Similarly the minimum of an multi-variable, scalar function is computed by the `fminsearch` function.

```
x = fminsearch(fun,x0)
```

As an example consider the function $f(x, y) = ax^2 + by^2$ which has a uniquely defined minimum at $(x, y) = (0, 0)$ for $a, b > 0$. To solve for the minimum we need to define the function in a separate M-file.

```
function f = myfun(x,a,b)
f = a*x(1)^2 + b*x(2)^2;
```

Now the call

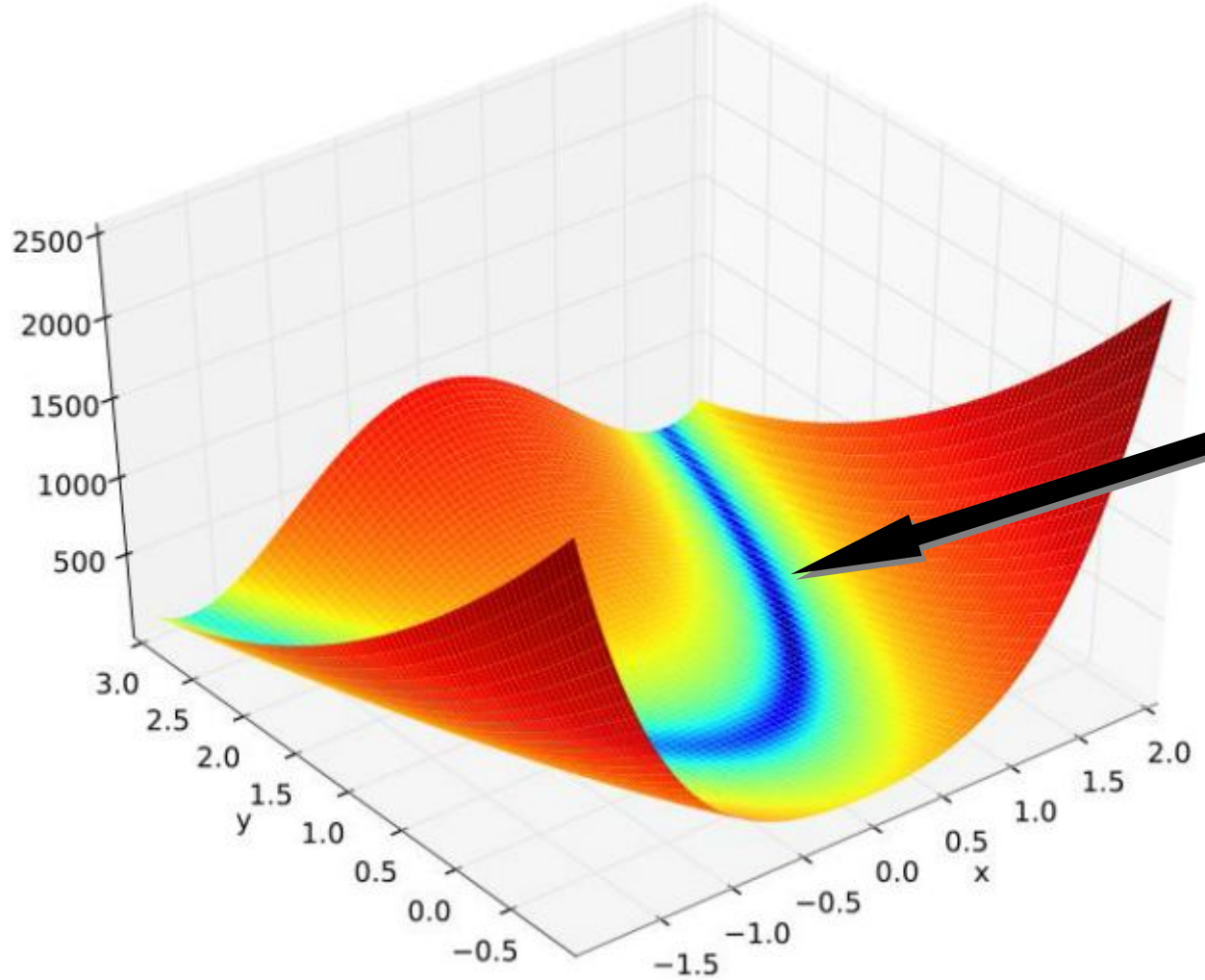
```
>> x = fminsearch(@(x) myfun(x,1,2),[0,1])

x =

   1.0e-03 *

    0.9298    0.0312
```

yields a numerical approximation to the true solution (starting at the value $(x, y) = (0, 1)$ and setting $a = 1, b = 2$). The result might come as a surprise, since we already started a the correct value of x and end up wit a deviation in the order of 10^{-3}. If we increase the default accuracy from 10^{-3} to 10^{-8} the deviation from the true value decreases accordingly.

```
>> opt = optimset('TolX',1e-8);
>> x = fminsearch(@(x) myfun(x,1,2),[0,1],opt)

x =

   1.0e-08 *

    0.1105    0.1613
```

The `optimset` function allows to adjust many more optimization settings. Consult the documentation for details!

Exemple : minimisation de la fonction de Rosenbrock (banane)



min = 0
au (1,1)

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2.$$

Minimizing Rosenbrock's Function with fminsearch

```
banana = @(x)100*(x(2)-x(1)^2)^2+(1-x(1))^2;
```

```
[x,fval,exitflag] = fminsearch(banana,[-1.2, 1])
```

This produces:

```
x =
    1.0000 1.0000

fval =
    8.1777e-010

exitflag =
    1
```

This indicates that the minimizer was found at [1 1] with a value near zero.

Ordinary differential equations in MATLAB

Differential equations are the most common tool to model time and state-continuous dynamical processes. Many of these ODEs have to be analyzed numerically. MATLAB provides a powerful suite for the numerical integration of ODEs. We will focus on two types of ODE solvers: `ode45` and `ode15s`. `ode45` is the first method of choice, but whenever the ODE is stiff, i.e., the dynamics happen on very different time scales, you should use `ode15s` for integration. The general call of the two ODE solvers is

```
[t,x] = ode45(@rhs_handle,time_span,x0,options,args)
[t,x] = ode15s(@rhs_handle,time_span,x0,options,args)
```

where `rhs_handle` is a handle to the right hand side defining the ODE (see below). The vector defining the lower and upper integration boundaries is `time_span`, the vector `x0` defines the initial conditions. Options like relative and absolute error tolerances can be specified within the `options` structure, which is generated with the `odeset` function. However, they can also be omitted (with `[]`) to use standard options. All additional parameters (`args`) are passed to the `rhs_handle` file.

As a first example consider the logistic differential equation, which is a model for the growth of populations.

$$\dot{x}(t) \quad = \quad rx(t)(K - x(t))/K \tag{1}$$

The time-dependent population density is $x(t)$, parameter r is the growth rate and K is called the carrying capacity (Can you explain why?). In order to implement this equation in MATLAB we have to define the right-hand-side of equation 1 as a MATLAB function.

```
function dxdt = logistic(t,x,r,K)
dxdt = r*x*(K-x)/K;
```

We create this function using the MATLAB editor and save it in the current working directory under the file name `logistic.m`. Lets assume the following parameter values: $r = 2, K = 10$ and as initial conditions $x_0 = 0.1$. We can now integrate the ODE for the time interval $0 \leq t \leq 10$ and plot the results with the call

```
[t,x] = ode45(@logistic,[0 10],0.1,[],2,10);
```

Note that we have left the options structure empty as we want to use the standard integrator options. If we want to get the solution $x(t)$ for specific time points, e.g., at each 1/10 unit of t, we can set `tspan = [0:0.1:10]`. However, the internal step size of the integrator is chosen automatically and cannot be controlled with the `tspan` argument.

```
[t,x] = ode45(@logistic,[0 10],0.1,[],2,10);
plot(t,x)
```

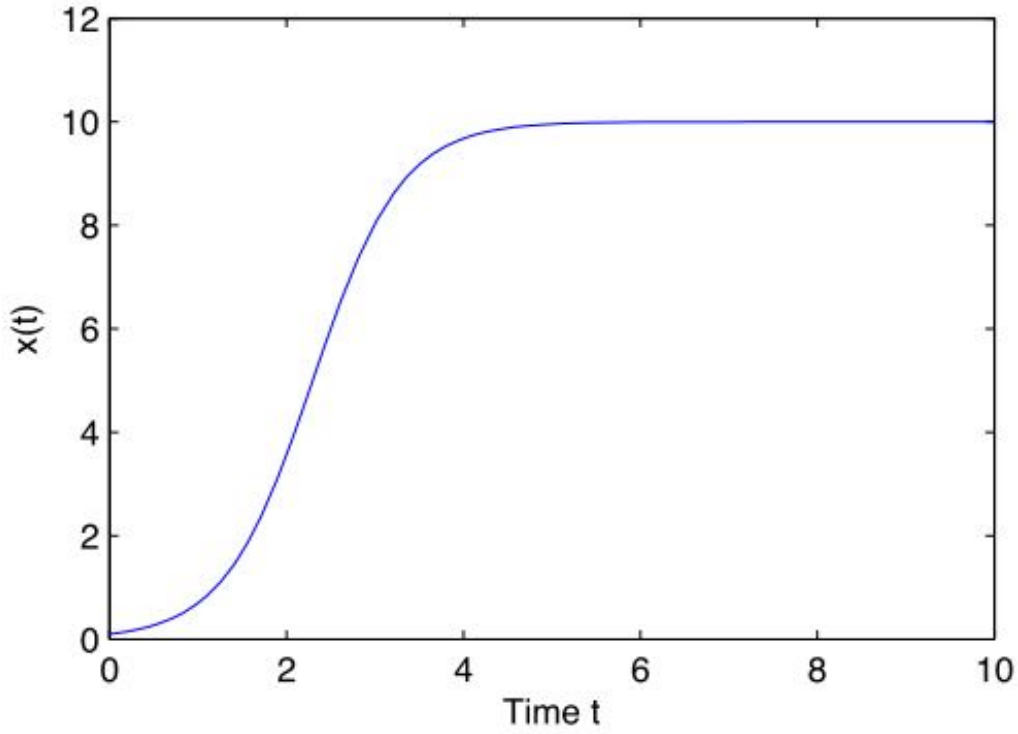


Figure 6: Solution of the logistic differential equation with $r = 2, K = 10$ and $x_0 = 0.1$.