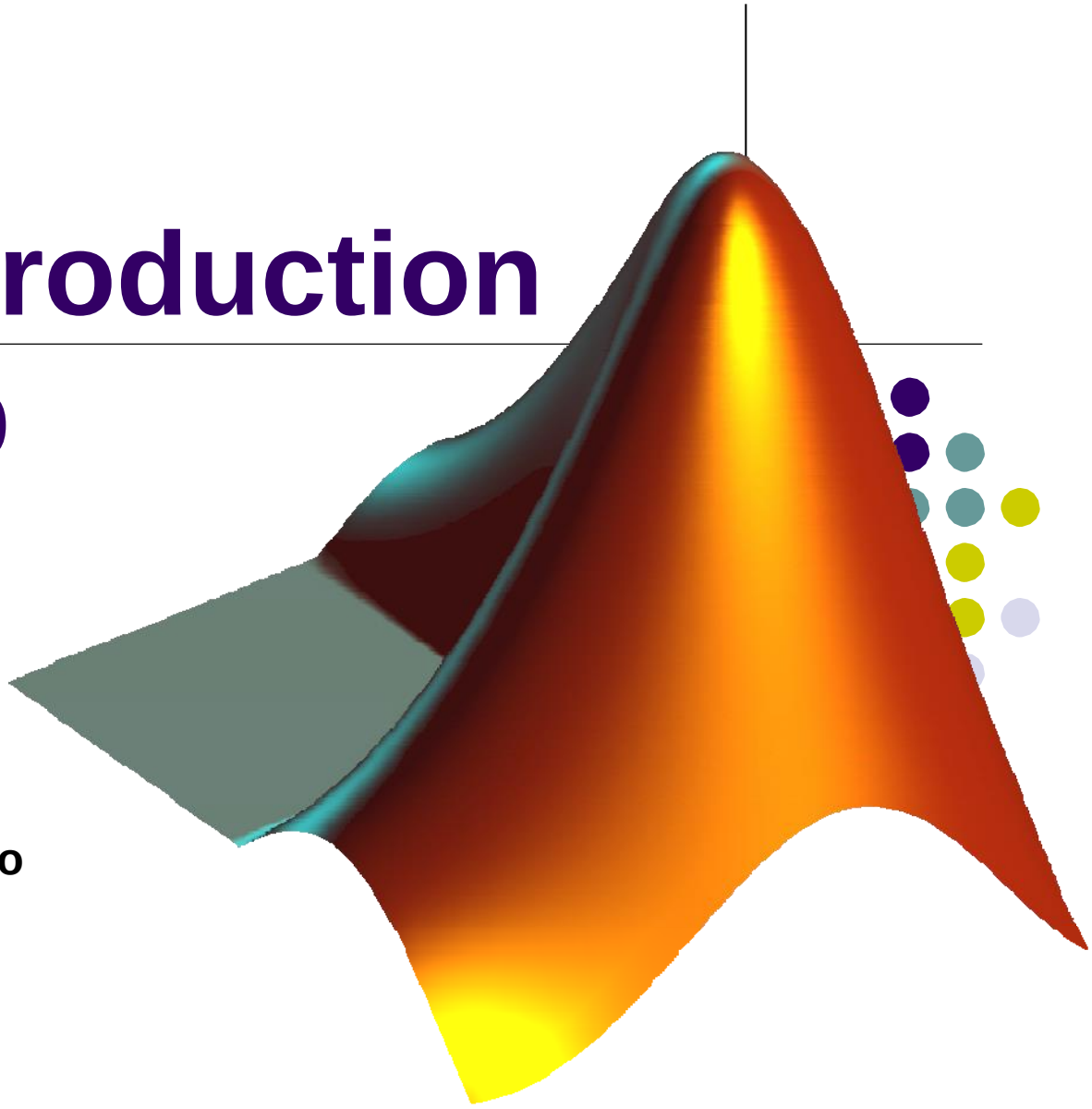
# Quick introduction to Matlab



**Edited by Michele Schiavinato**

# Outline

- Matlab introduction
- Matlab elements
  - Types
  - Variables
  - Matrices
- Scripts and functions
- Matlab Programming language
- Ploting

# Matlab introduction

- Matlab is a program for doing numerical computation. It was originally designed for solving linear algebra type problems using matrices.  It's name is derived from **MAT**rix **LAB**oratory.

- Matlab is also a programming language that currently is widely used as a platform for developing tools for Machine Learning

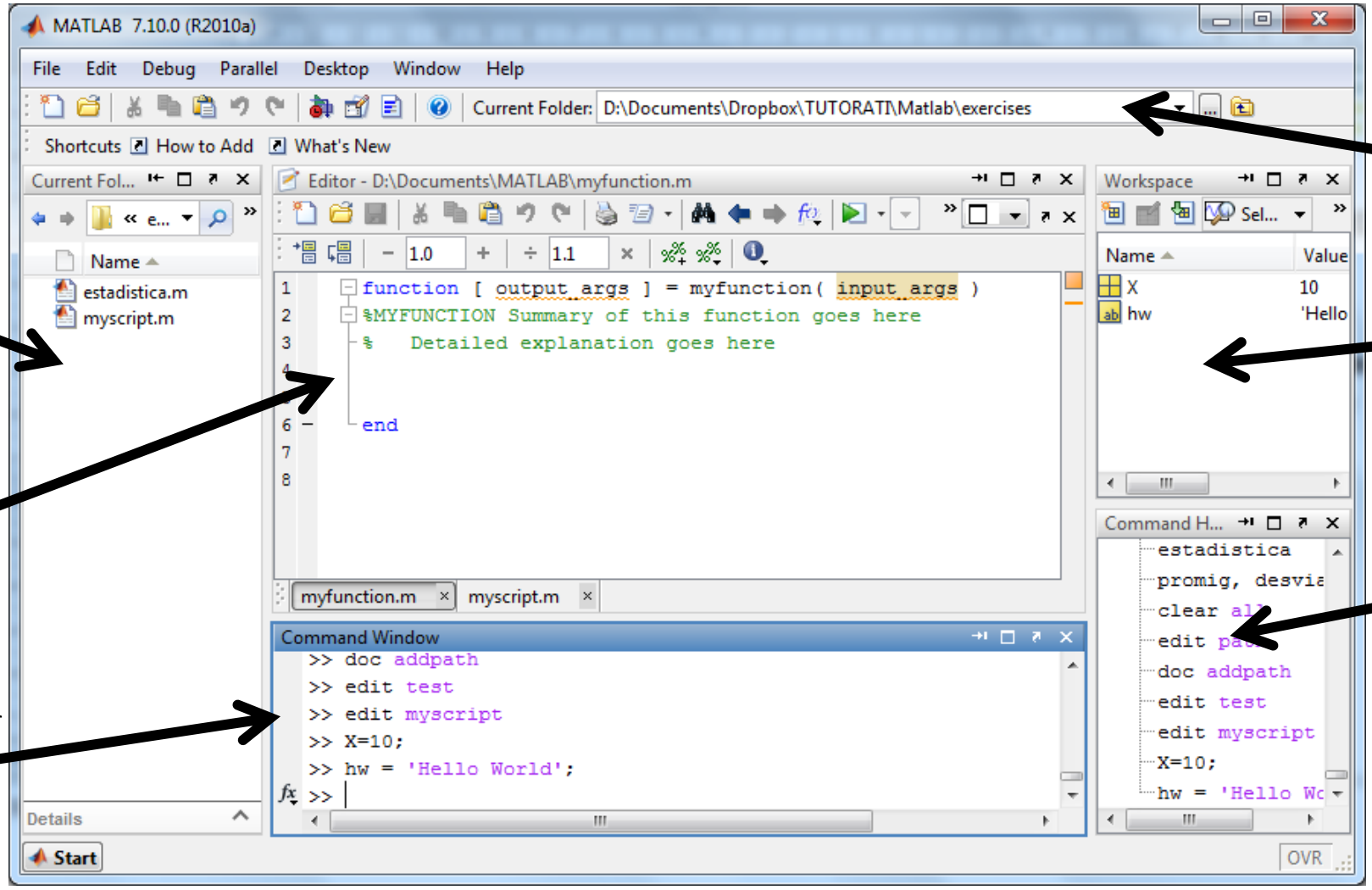
# Matlab main features

- Large toolbox of **numeric/image library functions**
- Very useful for **displaying/visualizing** data
- **High-level** coding**:** focus on algorithm structure, not on low- level details
- Allows **quick** prototype development of algorithms
- Powerful **debugging** features

# Matlab introduction

- Some other aspects of Matlab:
  - Matlab is an interpreter -> not as fast as compiled code
    - Typically quite fast for an interpreted language
    - Often used early in development -> can then convert to C (e.g.) for speed improvements
  - Can be linked to C/C++, JAVA, SQL, etc
  - Commercial product, but widely used in industry and academia
    - Many algorithms and toolboxes freely available

# Opening Matlab




Current Directory

Script Editor

Command Window

Working Path

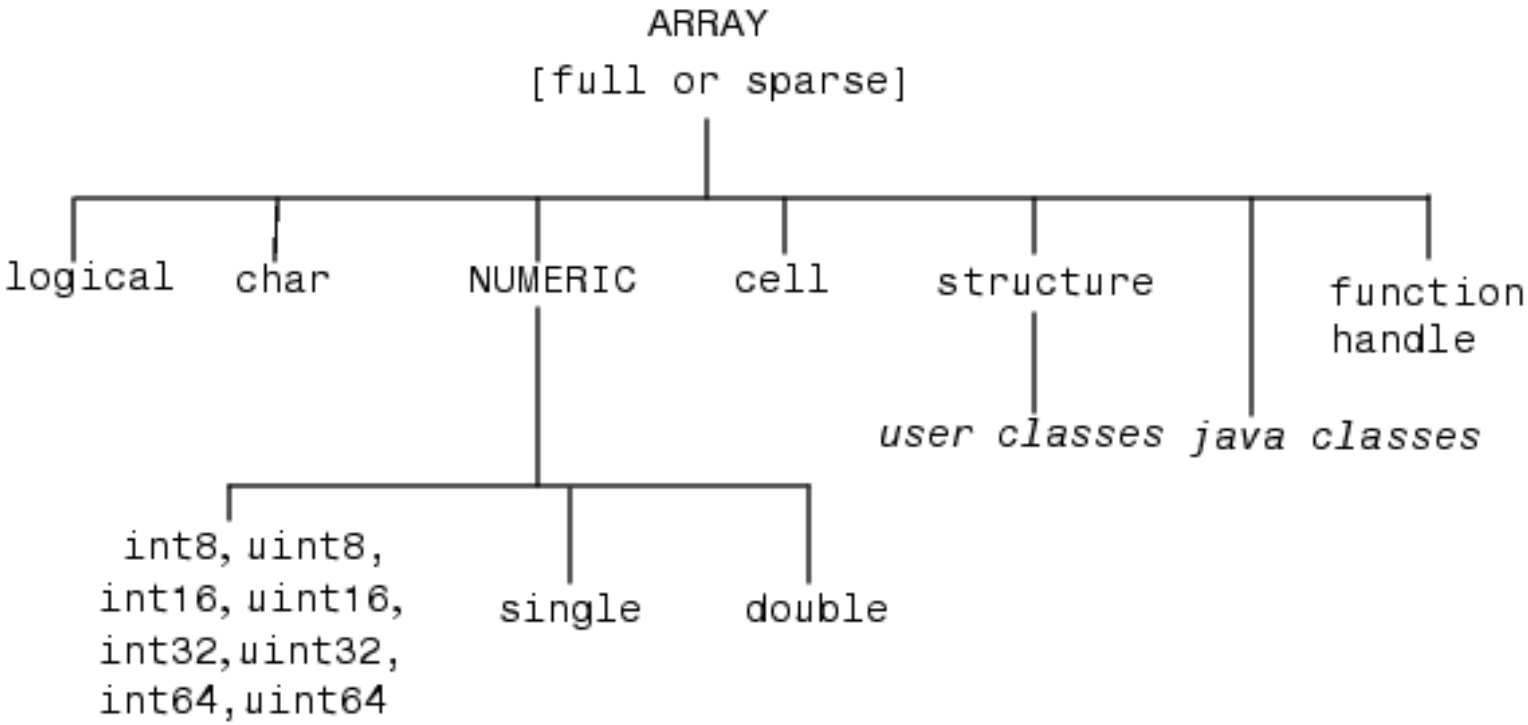
Working Memory

Command History

# Help

- Within Matlab
  - Type **help** at the Matlab prompt or **help** followed by a function name for help on a specific function
  - Type **doc** to get the graphical version of help
- Online
  - Online documentation for Matlab at the MathWorks website
    - http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html
  - There are also numerous tutorials online that are easily found with a web search.

# Data Types


ARRAY
[full or sparse]
logical
char
NUMERIC
cell
structure
function handle
user classes
java classes
int8, uint8,
int16, uint16,
int32, uint32,
int64, uint64
single
double

# Variables

- Have not to be previously declared
- Variable names can contain up to 63 characters
- Variable names must start with a letter followed by letters, digits, and underscores.
- Variable names are case sensitive

```
>> x = 10        --> x = 10
>> y = 3e-3      --> y = 0.0030
>> a = 'hello' --> a = hello
>> A  --> ??? Undefined function or
variable 'A'.
```

# Workspace

All the assigned variables are added to the workspace.

You can remove a specific variable form the workspace using:

```
>> clear 'var_name'
```

or remove all the variables  using:

```
>> clear
```

# Console output

We can see the value of a variable by typing is name on the command window:

```
>> z   --> z =     3
```

Terminating a line with a ; suppress the output of the assigned variable value:

```
>> x=10    -->    x =   10
>> x=10;   -->
```

In the expression is not an assignment its value is automatically assigned to the special variable **ans**:

```
>> 10      -->    ans =   10
```

# Console output

The default printing format shows only the first 4 decimal of a number:

```
>> x=1/3
x =
    0.3333
```

With the '**format**' command, you can set different output formats for numbers:

```
>> format long
>> x=1/3
x =
   0.333333333333333
```

You can **clean** all the current console typing the '**clc**' command.

# Matlab Assignment & Operators

| | | | | | | |
|---|---|---|---|---|---|---|
| Assignment | `=` | | `a = b` | (assign `b` to `a`) | | |
| Addition | `+` | | `a + b` | | | |
| Subtraction | `-` | | `a - b` | | | |
| Multiplication | `*` | or `.*` | `a*b` | or | `a.*b` | |
| Division | `/` | or `./` | `a/b` | or | `a./b` | |
| Power | `^` | or `.^` | `a^b` | or | `a.^b` | |

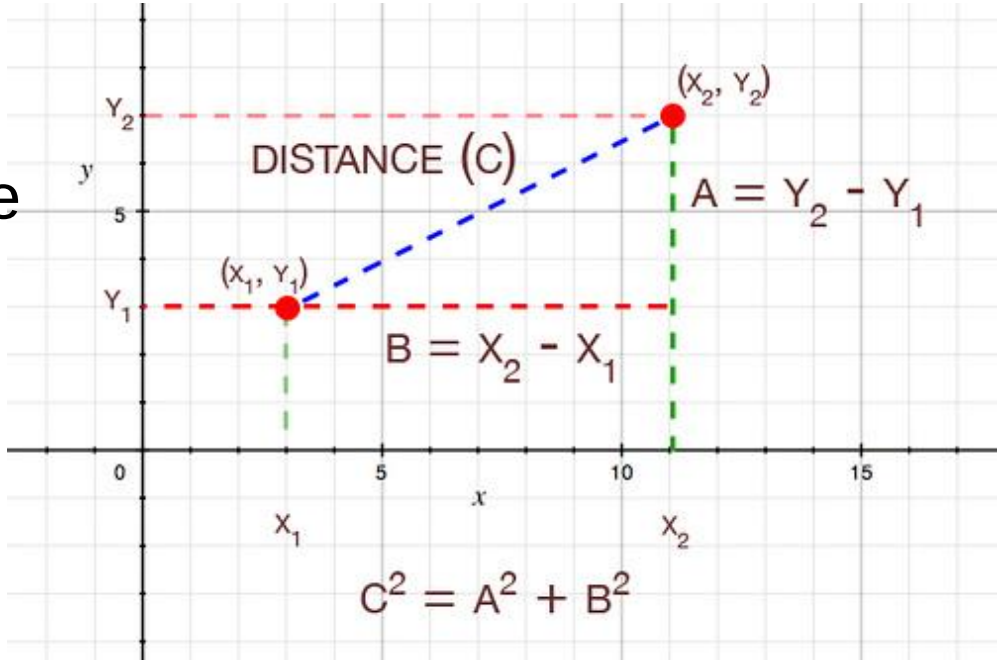
Function call:

```
func_name(p1, p2, …)
[o1, o2, …] = func_name(p1, p2, …)
```

# Operators exercise

Given two points in $\mathbb{R}^2$ with coordinates $x_1$,$y_1$ and $x_2$,$y_2$, compute their euclidean distance:

- Assign a value to $x_1$, $y_1$, $x_2$, $y_2$
- Use the operators provided by Matlab to compute the distance between the two points and assign it to a variable **d**



# Solution

```
>> x1 = 1;
>> y1 = 1;

>> x2 = 2;
>> y2 = 2;

>> d = ((x1-x2)^2 + (y1-y2)^2)^0.5

d =

   1.4142e+000
```

# Matlab Usefull Constants

| | |
|---|---|
| pi | Value of $\pi$ |
| eps | Floating-point relative accuracy |
| inf | Infinity |
| NaN | Not a number, e.g. 0/0 |
| realmin | The smallest usable positive real number |
| realmax | The largest usable positive real number |

The inbuilt Matlab constants can be overwritten

# Matlab Matrices

- Matlab treats all variables as matrices.  For our purposes a matrix can be thought as a bidimensional array, in fact, that is how it is stored.
- Vectors are special forms of matrices and contain only one row OR one column.
- Scalars are matrices with only one row AND one column.

# Matlab Matrices

- A matrix with only one row is called a row vector. A row vector can be created in Matlab as follows (note the commas):

```
>> rowvec = [12, 14, 63]

rowvec =

  12  14  63
```

# Matlab Matrices

- A matrix with only one column is called a column vector. A column vector can be created in MATLAB as follows (note the semicolons):

```
>> colvec = [12; 14; 63]

colvec =

   12
   14
   63
```

# Regularly spaced vectors

- A regularly spaced vector can be created using the colon ( : ) operator.

| | |
|---|---|
| j:k | is the same as [j,j+1,...,k], or empty when j > k. |
| j:i:k | is the same as [j,j+i,j+2i, ...,j+m*i], where m = fix((k-j)/i), i.e. the rounded value toward zero of (k-j)/i ratio.<br>This syntax returns an empty matrix when<br>▪ i == 0,<br>▪ i > 0 and j > k,<br>▪ i < 0 and j < k. |

>> 1:4 → ans = 1 2 3 4

>> 1:2:10 → ans = 1 3 5 7 9

# Matlab Matrices

- A matrix can be created in Matlab as follows (note the commas AND semicolons):

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

```
matrix =

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

# Selecting an element of a vector/matrix

We can access the *n*-th element of a vector by using the **(n)** operator.

**The indexes in Matlab start from 1**

To access the first element of a row vector:

```
>> rowvec(1) -->  ans = 12
```

To access the an element of a matrix we have to indicate its row and column indexes:

```
>> matrix(1,1)  --> ans = 1
```

# Extracting a Sub-Matrix

- A portion of a matrix can be extracted and stored in a smaller matrix by specifying the names of both matrices and the rows and columns to extract. The syntax is:

```
sub_matrix = matrix ( r1 : r2 , c1 : c2 ) ;
```

where r1 and r2 specify the beginning and ending rows and c1 and c2 specify the beginning and ending columns to be extracted to make the new matrix.

# Extracting a Column

- A column vector can be extracted from a matrix. As an example we create a matrix below:

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

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

- Here we extract column 2 of the matrix and make a column vector:

```
>> col_two=matrix(1:3,2)

col_two =
   2
   5
   8
```

# Extracting a Row

- A row vector can be extracted from a matrix.
- Here we extract row 2 of the matrix and make a row vector. Note that the 2 specifies the second row and the 1:3 specifies which columns of the row.

```
>> rowvec = matrix(2,1:3)

   rowvec =

       4     5     6
```

# Colon Operator

| | |
|---|---|
| **j:k** | is the same as [j,j+1,...,k]  is empty if j > k |
| **j:i:k** | is the same as [j,j+i,j+2i, ..,k]  is empty if i > 0 and j > k or if i < 0 and j < k |
| **A(:,j)** | is the j-th column of A |
| **A(i,:)** | is the i-th row of A |
| **A(:,:)** | is the equivalent two-dimensional array. For matrices this is the same as A. |
| **A(j:k)** | is A(j), A(j+1),...,A(k) |
| **A(:,j:k)** | is A(:,j), A(:,j+1),...,A(:,k) |
| **A(:,:,k)** | is the k-th page of three-dimensional array A. |
| **A(i,j,k,:)** | is a vector in four-dimensional array A. The vector includes A(i,j,k,1), A(i,j,k,2), A(i,j,k,3), and so on. |
| **A(:)** | is all the elements of A, regarded as a **single column**.<br>On the left side of an assignment statement, A(:) fills A, preserving its shape from before. In this case, the right side must contain the same number of elements as A. |

# Some matrix functions in Matlab

| | |
|---|---|
| • X = ones(r,c) | Creates matrix full with ones |
| • X = zeros(r,c) | Creates matrix full with zeros |
| • A = rand(r,c) | Creates a matrix with random numbers uniformally distributed in [0,1] |
| • B = diag(x) | Creates squared matrix with vector x in diagonal |
| • [r,c] = size(A) | Returns dimensions of matrix A |
| • + - * | Standard operations |
| • .+ .- .* ./ | Wise addition, substraction, … |
| • v = sum(A) | Vector with sum of columns |

# Transpose of a matrix

You can transpose a matrix using ' symbol:

For example, given a matrix A as follows:

```
>> A
A =
     1     2     3
     4     5     6

>> A'
ans =
     1     4
     2     5
     3     6
```

# Some matrix operations

- **Selecting the diagonal elements**

  `d = diag(A)`

  d is a vector containing the diagonal elements of A

- **Accessing Multiple Elements of a Matrix**

  `A(1,4) + A(2,4) + A(3,4) + A(4,4)`

  `sum(A(1:4,4)) or sum(A(:,end))`

  The keyword **end** refers to the *last* row or column.

- **Deleting Rows and Columns**

  to delete the second column of X, use `X(:,2) = []`

- **Concatenating Matrices A and B**

  `C=[A;B]` or `C=[A,B]`

# Exercise

Play with matrix indices and operators:

- Create a random 4x4 matrix
- Print the second column
- Subtract the first column from the diagonal

- Create a matrix of N vectors in $\mathbb{R}^2$ (Nx2)
- Compute and print the Euclidean norm of the vectors
- Consider the previous matrix as a matrix of points, select the points which distance from the origin is lower than the average distance (of all points from the origin).

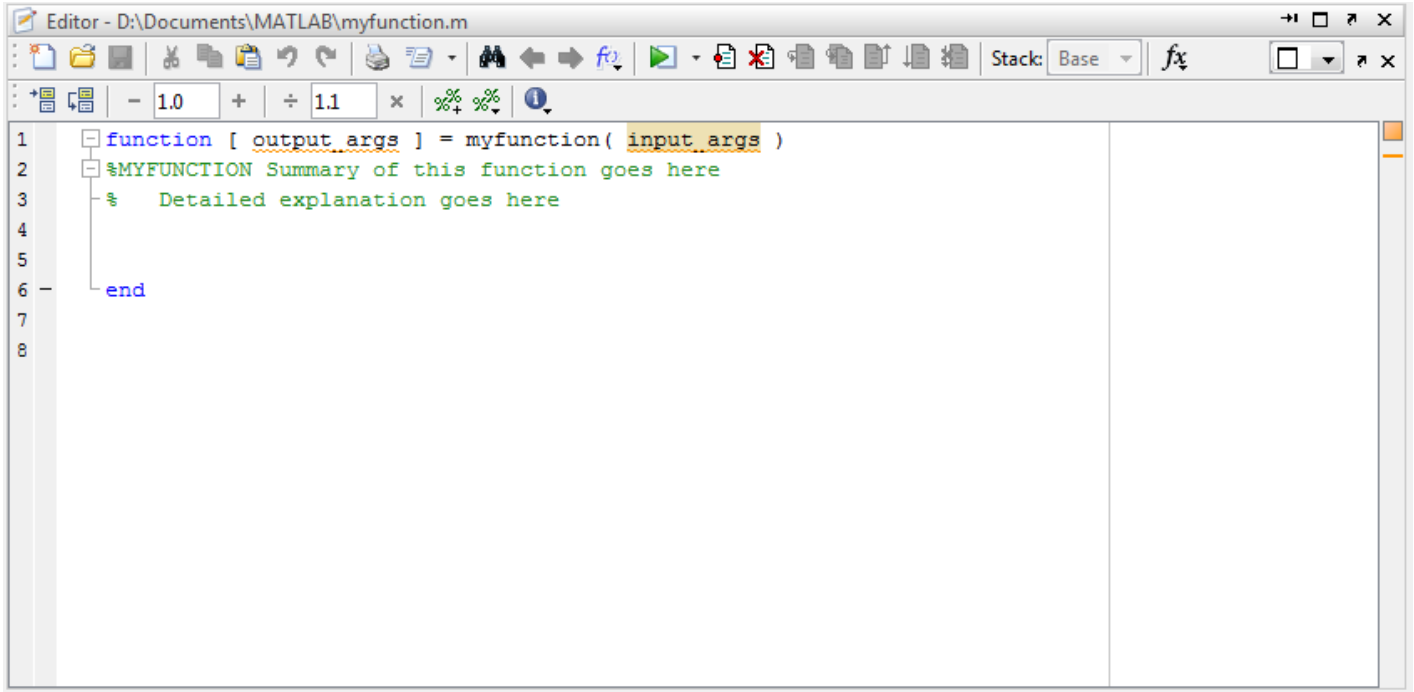
HINT: -use the Matlab help to learn about the **find** command;
-the function sqrt(A) compute the square root of each element of the matrix

# M-files

- M-Files are text files containing Matlab programs.
- Can be called from the command line or from other M-files.
- Present "**.m**" extension.
- Two kind of M-files:
  - **Scripts**
  - **Functions**

# Matlab Editor

- Matlab comes with its own text editor.
- To edit the file ***myscript.m*** enter the command **edit myscrip.**
- If the file *myscript.m* does not exist a new empty file will be created in the current directory.



# M-files: Scripts

- Without input arguments, they do not return any value.
- They are simply a **list of commands** that are executed in sequence.
- The commands of a script use the **current workspace.**

# M-files: Script Example

1) >> **edit** statistics.m
2) Write into the editor:

```
x = [4 3 2 10 -1];
n = length(x);
sum1 = 0; sum2 = 0;
for i=1:n
   sum1 = sum1 + x(i);
   sum2 = sum2 + x(i)*x(i);
end
mean_x = sum1/n;
stddev_x = sqrt(sum2/n – mean_x*mean_x);
```

3)Save the file
4) >> **run statistics**
5) >> mean_x, stddev_x

mean_x = 3.6000
stddev_x = 3.6111

# M-files: Functions

- With parameters and returning values.
- Only visible variables defined inside the function or parameters (they have their own workspace).
- Usually one file for each function defined.
- Structure:

```
function [out1, ..., outN] =
                        name-function (par1, ..., parM)
  sentence;
  …
  sentence;
end
```

# M-files: Functions Example

1)>> **edit** fstatistics.m

2) Write into the editor:

```
function [mean_x, stddev_x] = fstatistics(x)
    n = length(x);
    [sum1,sum2] = sums(x,n);
    mean_x = sum1/n;
    stddev_x = sqrt(sum2/n – mean_x.^2);
end
```

3) Save the file

4)>> **edit** sums.m

5) Write into the editor:

```
function [outsum1,outsum2] = sums(y,m)
    outsum1 = 0;
    outsum2 = 0;
    for i=1:m
        outsum1 = outsum1 + y(i);
    outsum2 = outsum2 + y(i)*y(i);
    end
end
```

6) Save the file

7) >> [p,d] = fstatistics([4 3 2 10 -1])

```
p = 3.6000
d = 3.6111
```

# M-file execution

- We can execute a m-file writing its name on the console:

```
>> sayHello
     hw = Hello World
```

sayHello.m

```
Hw = 'Hello World'
```

- We can run the current file in the editor pressing **F5**
- We can run the selected commands in the editor pressing **F9 or CTRL+Enter** (the latter does not show those commands are launched in console).
- In the m-file we can delimit some portion of the commands using two comment characters %%

# M-File location

We can run only m-files located in the Matlab Search Path or in current directory.

- We can add a folder temporarily to the Search Path using:

```
>> addPath('directory_path')
```

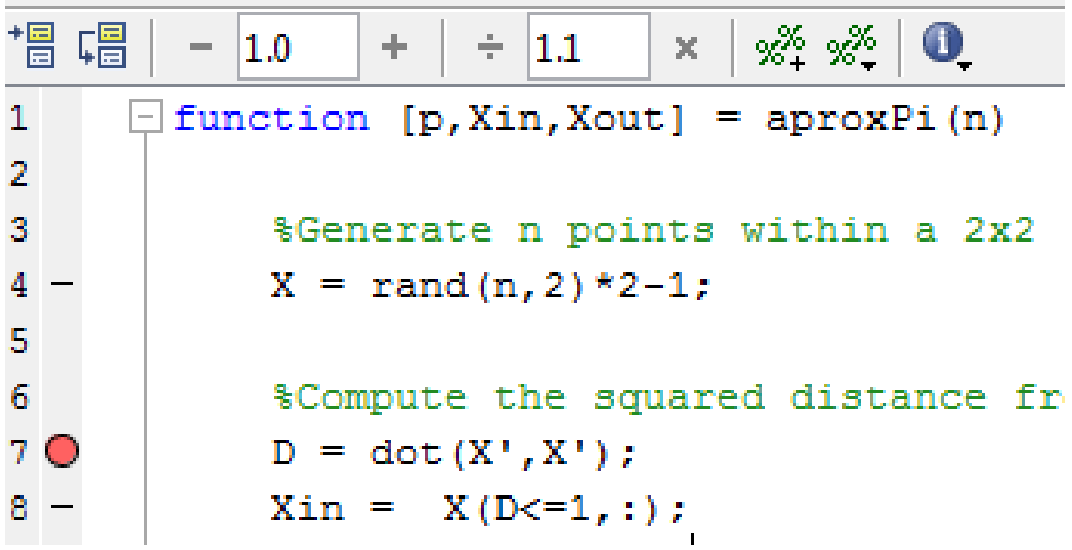
- Or permanently:
  1. Go to "File->Set Path" from within MATLAB or type "pathtool" at the MATLAB prompt.
  2. Use the "Add" button to add your desired folder(s) to the MATLAB path.
  3. Click "Save" so that this path is used in future MATLAB sessions.

# Debugging a .m file

Matlab have a powerfull debugger.

We can set/unset a breakpoints clicking on the right side of the line number.
The execution flow stops when a breakpoint is reached and we can:

- watch the workspace state of the function scope.
- change the value of the variables
- run commands that uses the current workspace.



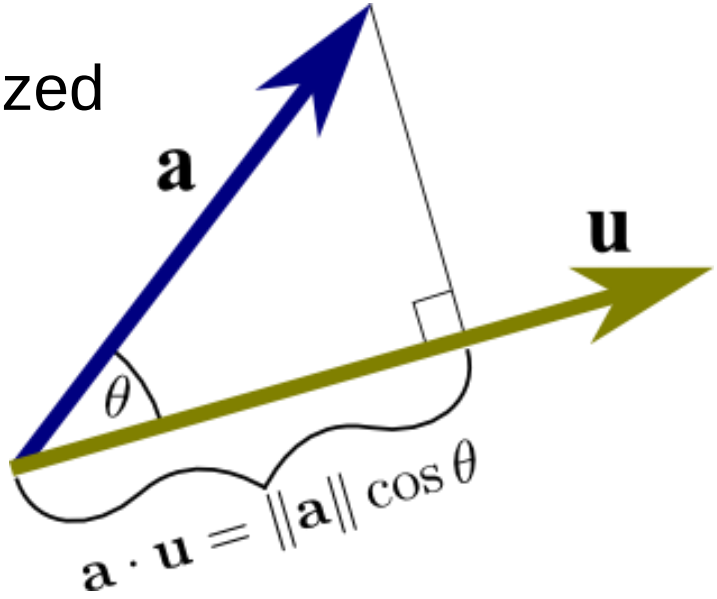
# Exercise

- Write a function that normalizes a given vector.

```
>> n_vec = normalized(vec);
```

- In a Matlab script generate 2 random vectors and measure the angle between them.

**Hint**: the dot product between two normalized vectors is equal to the cosine of the angle between them.



# Solution

```
>> edit normalized
```

```
function [ nvec ] = normalized( vec )
%NORMALIZED Returns the normalized vector
%  NV = normalized(V) returns the vector V./norm(vec)
    nvec = vec./norm(vec);
end
```

```
>> edit measureangle
```

```
% generate two random vectors
x1 = rand(2,1);
x2 = rand(2,1);
% The smallest angle between two normalized vectors is
% equal to the arcocosine of the dot product between them
angle = acos(normalized(x1)'*normalized(x2))*180/pi
```

```
>> measureangle
angle =
 17.9611
```

# Matlab programming language

- Elements of Matlab as a programming language:
  - Expressions
  - Flow Control blocks
    - Conditional
    - Iterations
  - Scripts
  - Functions

# Expressions: Matlab Relational Operators

- MATLAB supports six relational operators.
  - Less Than <
  - Less Than or Equal <=
  - Greater Than >
  - Greater Than or Equal >=
  - Equal To ==
  - Not Equal To ~=

# Expressions: Matlab Logical Operators

- MATLAB supports three logical operators.
  - not ~ highest precedence
  - and & equal precedence with or
  - or | equal precedence with and

# Expressions: Matlab Logical Functions

- MATLAB also supports some logical functions.

| | |
|---|---|
| any (x) | returns 1 if any element of x is nonzero |
| all (x) | returns 1 if all elements of x are nonzero |
| isnan (x) | returns 1 at each NaN in x |
| isinf (x) | returns 1 at each infinity in x |
| finite (x) | returns 1 at each finite value in x |

# Matlab Conditional Structures

```
if expression cond.
   sentences
elseif expr. cond.
   sentences
else
   sentences
end
```

```
a = input('valor1? ');
b = input('valor2? ');
if a == b,
   fprintf('a is equal to b\n');
elseif a > 0 && b > 0
   fprintf('both positive\n');
else
   fprintf('other case\n');
end
```

# Matlab Iteration Structures (I)

```
for variable = expr
   sentence;
    ...
   sentence;
end
```

```
M = rand(4,4); suma = 0;
for i = 1:4
  for j = 1:4
    suma = suma + M(i,j);
  end
end
fprintf('sum = %d\n',suma);
```

```
M = rand(10,10); suma = 0;
for i = [2, 5:8]        % rows 2, 5, 6, 7, 8
  for j = [1:5, 8:9]    % cols 1, 2, 3, 4, 5, 8, 9
    suma = suma + M(i,j);
  end
end
fprintf('sum = %d\n',suma);
```

# Matlab Iteration Structures (II)

```
while expr
   sentence;
    ...
   sentence;
end
```

```
M = rand(4,4);
i = 1; j = 1;
suma = 0;

while i <= 4
  while j <= 4
    suma = suma + M(i,j);
    j = j+1;
  end
  i = i+1;
end
fprintf('suma = %f\n',suma);
```

# (Optimizing code: vectorization)

- Loops should be avoided when possible:

```
for ind = 1:10000
    b(ind)=sin(ind/10)
end
```

Alternatives:

```
x=0.1:0.1:1000;
b=sin(x);
```

```
x=1:10000;
b=sin(x/10);
```

Most of the loops can be avoided!!!

# Exercise

Given two matrices of N points in $\mathbb{R}^2$ compute the average distance between each pair.

Compairs the execution time of two different implementations:

- using loops
- avoiding loops

Use the instructions **tic** and **toc** to measure the elapsed time.

```
>> tic;pause(0.1);toc;
Elapsed time is 0.105363 seconds.
```

# Solution

```
>> edit avoidloops
```

```
%generates 2 matrices of 100000 points
X1 = rand(2,1000000);
X2 = rand(2,1000000);

%loops version
tic
dists = zeros(1,1000000);
for i=1:1000000
    dists(i)=sqrt( (X1(1,i)-X2(1,i))^2 ...
              + (X1(2,i)-X2(2,i))^2);
end
fprintf('loop elapsed time: %fs\n',toc);

%matrices version
tic
dists = sqrt(sum((X1-X2).^2));
fprintf('matrices elapsed time: %fs\n',toc);
```

```
>> avoidloops
loop elapsed time: 1.300240s
matrices elapsed time: 0.026092s
```

# Exercise: Fibonacci

- Write a function which compute and return a vector containing the first n numbers of the Fiboncci series.

```
>> fib(10)
ans =
     1   1   2   3   5   8   13   21   34   55
```

- Write also a recursive implementation of the same function.

# Plotting with Matlab

- Matlab has a lot of function for plotting data.
- The basic version requires two input vectors, one for the abscissae ( x values) and one for the ordinates (y values).
- The vectors have to be the same length.

```
>> plot (time, dist)
```
plotting versus time

- We can give the plot function only the ordinates (y values). The vector indices are then considered as abscissae.

```
>>  plot (dist)
```
plotting versus index

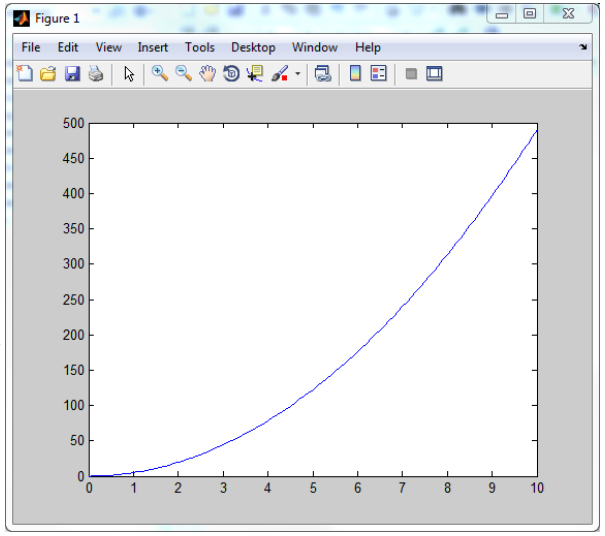
- To display multiple graphs at the same time we need to open a new window using the “**figure**“ command.
- The plot will be drawn in the last opened window.

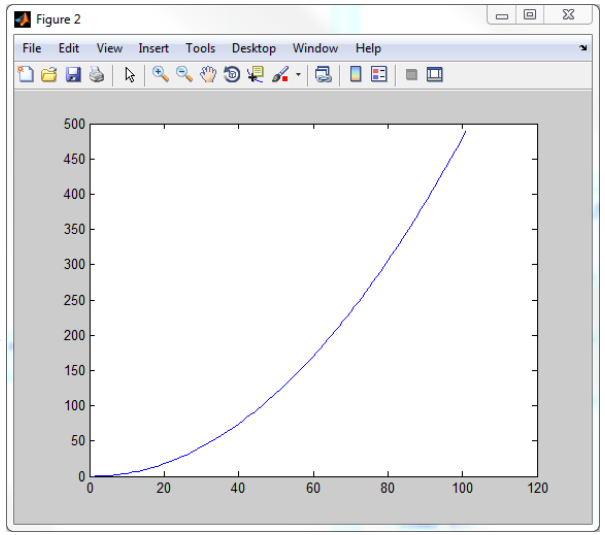
# Plotting with Matlab

```
time = 0:0.1:10;
dist = 0.5.*9.8.*time.^2;

%plot distance over time
plot(time,dist);

%open a new window
figure;
%plot distance over indices
plot(dist);
```

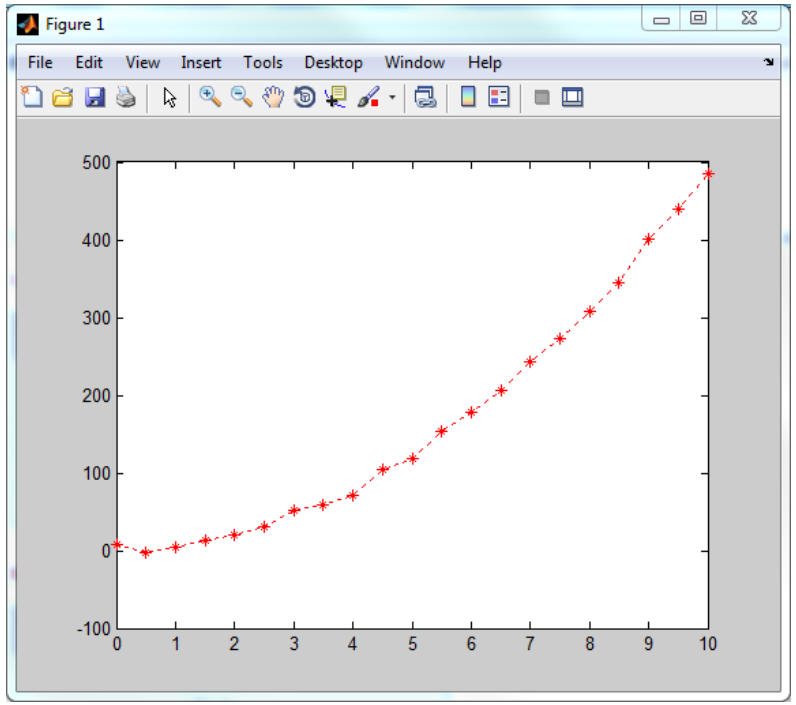


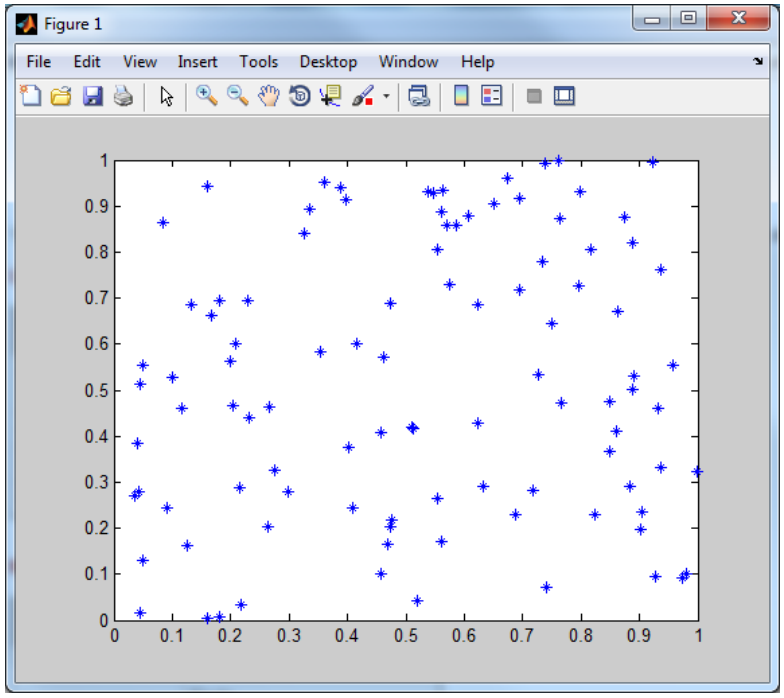



# Plotting in Matlab

We can specify the line and the marker style with an additional parameter of **plot** function.

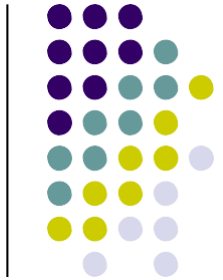
```
>> dist=dist+(rand(size(dist))-0.5)*20;
>> plot(time,dist,':*r');
```




```
x = rand(1,100);
y = rand(1,100);
plot(x,y,'*')
```

# Plotting in Matlab



```
>> doc LineSpec
```

### Line Style Specifiers

| Specifier | Line Style |
|---|---|
| - | Solid line (default) |
| -- | Dashed line |
| : | Dotted line |
| -. | Dash-dot line |

### Color Specifiers

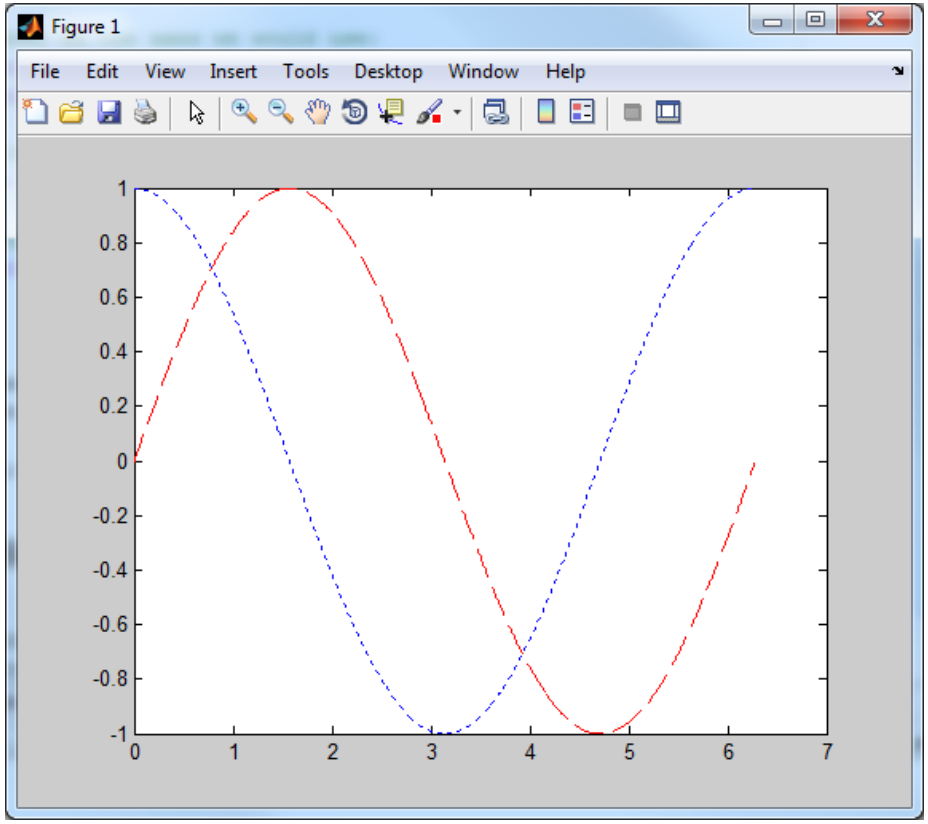
| Specifier | Color |
|---|---|
| r | Red |
| g | Green |
| b | Blue |
| c | Cyan |
| m | Magenta |
| y | Yellow |
| k | Black |
| w | White |

### Marker Specifiers

| Specifier | Marker Type |
|---|---|
| + | Plus sign |
| o | Circle |
| * | Asterisk |
| . | Point (see note below) |
| x | Cross |
| 'square' or s | Square |
| 'diamond' or d | Diamond |
| ^ | Upward-pointing triangle |
| v | Downward-pointing triangle |
| > | Right-pointing triangle |
| < | Left-pointing triangle |
| 'pentagram' or p | Five-pointed star (pentagram) |
| 'hexagram' or h | Six-pointed star (hexagram) |

# Plotting with Matlab

We can plot multiple functions on the same graph.

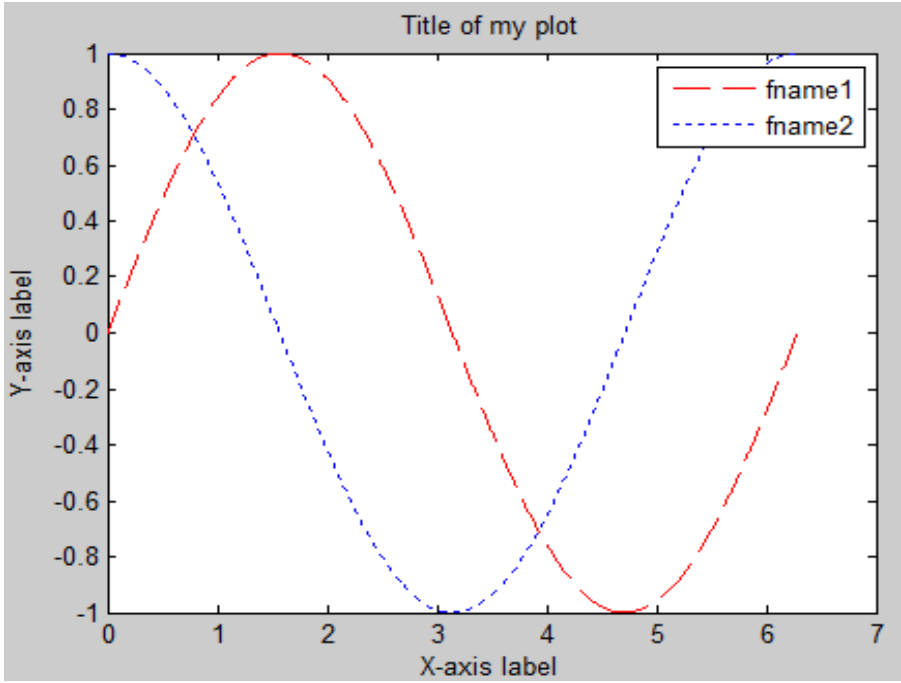



```
X=0:0.01:2*pi;
cosx=cos(X);
sinx=sin(X);
plot(X,sinx,'—r' , X,cosx,':b');
```

# Plotting with Matlab

- There are commands in Matlab to "annotate" a plot to put on axis labels, titles, and legends. For example:

```
% To put a title on the plot, we would use:
title ('Title of my plot')
% To put a label on the axes we would use:
xlabel ('X-axis label')
ylabel ('Y-axis label')
% To add a legend we should use:
legend('fname1','fname2');
```



# Save Plot

We can save the current plot to a file using:

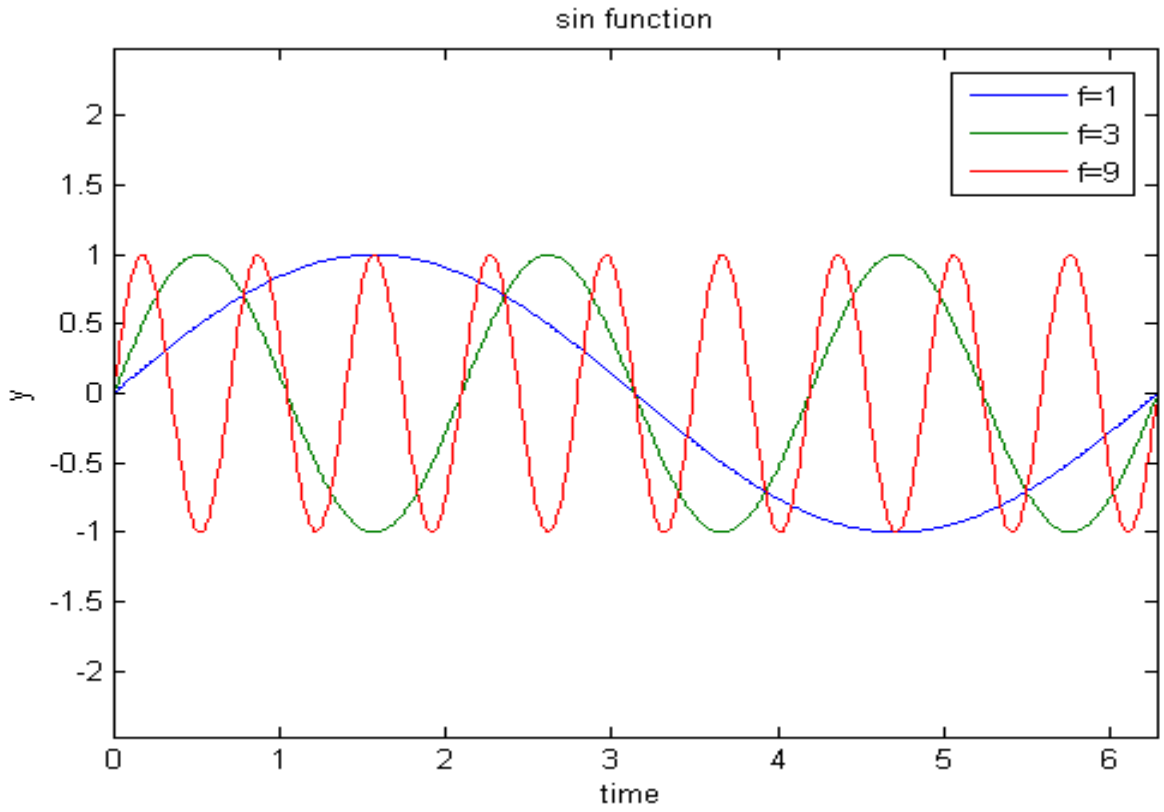
```
>> print -dpng 'filename'
```

We use different output formats, ie:
`-dpng`: save to a png image file (Rasterized)
`-dpdf`: save to a pdf file (Vectorized)
`-dsvg`: save to svg (vectorized)

# Exercise

Plot multiple sine functions over the time (t) with different frequencies (f):

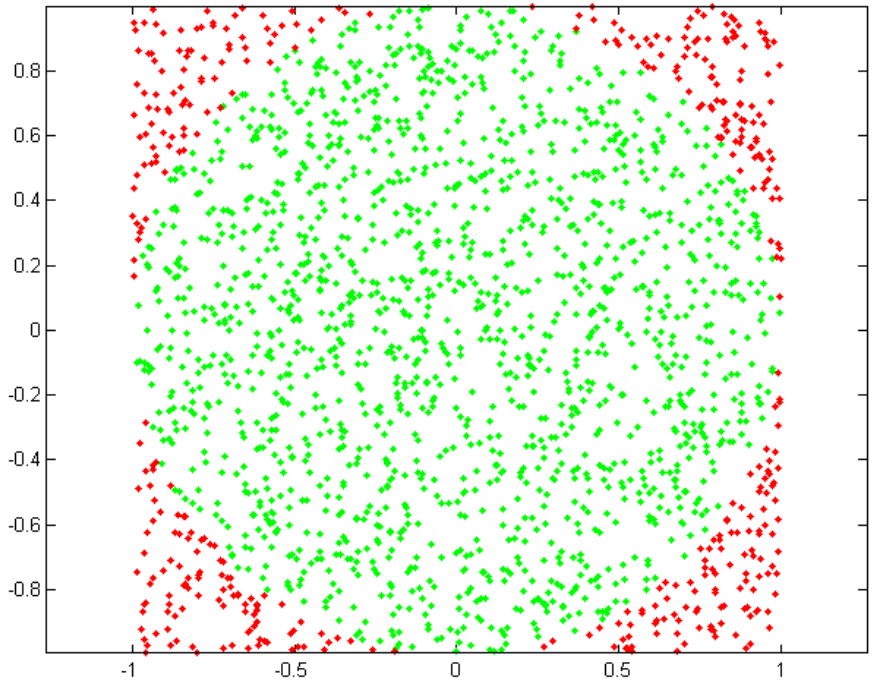
```
y = sin(f*t)
```



# Exercise: pi approximation

If we consider a circle with radius **r**, we can approximate its area randomly sampling the rectangle in which the circle is inscribed.
The approximate area of the circle is equal to the product of the **rectangle area** and the **probability of hitting the circle**.

- Write a function which takes the number of samples as input and returns the approximate value of $\pi$
- How many samples do I need to obtain an accuracy to the third digit of $\pi$?



# Solution

```
function p = approxpi(n)
%APROXPI Returns an approximation of Pi based on the
%statistical sampling of a circle inscribed
%on a rectangle area.
%  P = approxpi(N) returns the value of Pi computed using
N samples.

    %Generate n points within a 2x2 square centered in
the origin (hence the inscribed circle has r=1)
    X = rand(n,2)*2 - 1;

    %Compute the squared distance from the origin
    D = dot(X',X');
    Xin =  X(D<=1,:);
    Xout = X(D>1,:);
    p = 4*size(Xin,1)/n; %since r=1 the area of the
                         %circle is just the value of pi
    plot(Xin(:,1),Xin(:,2),'.g',Xout(:,1),Xout(:,2),'.r')
    axis equal
end
```