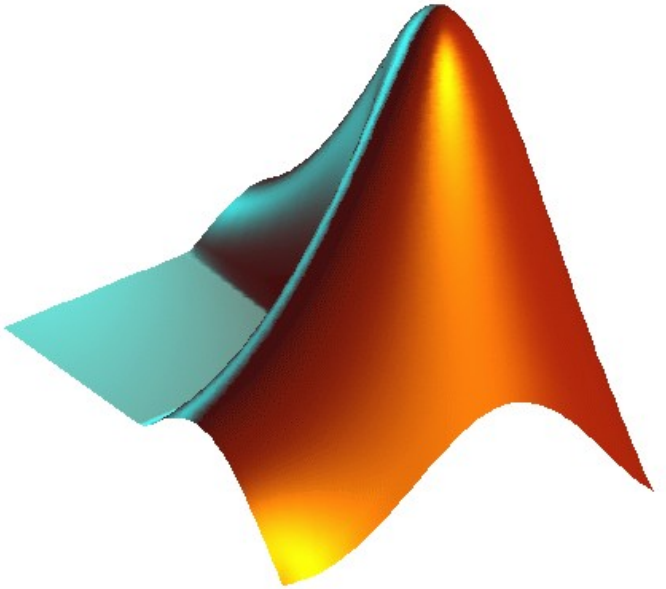
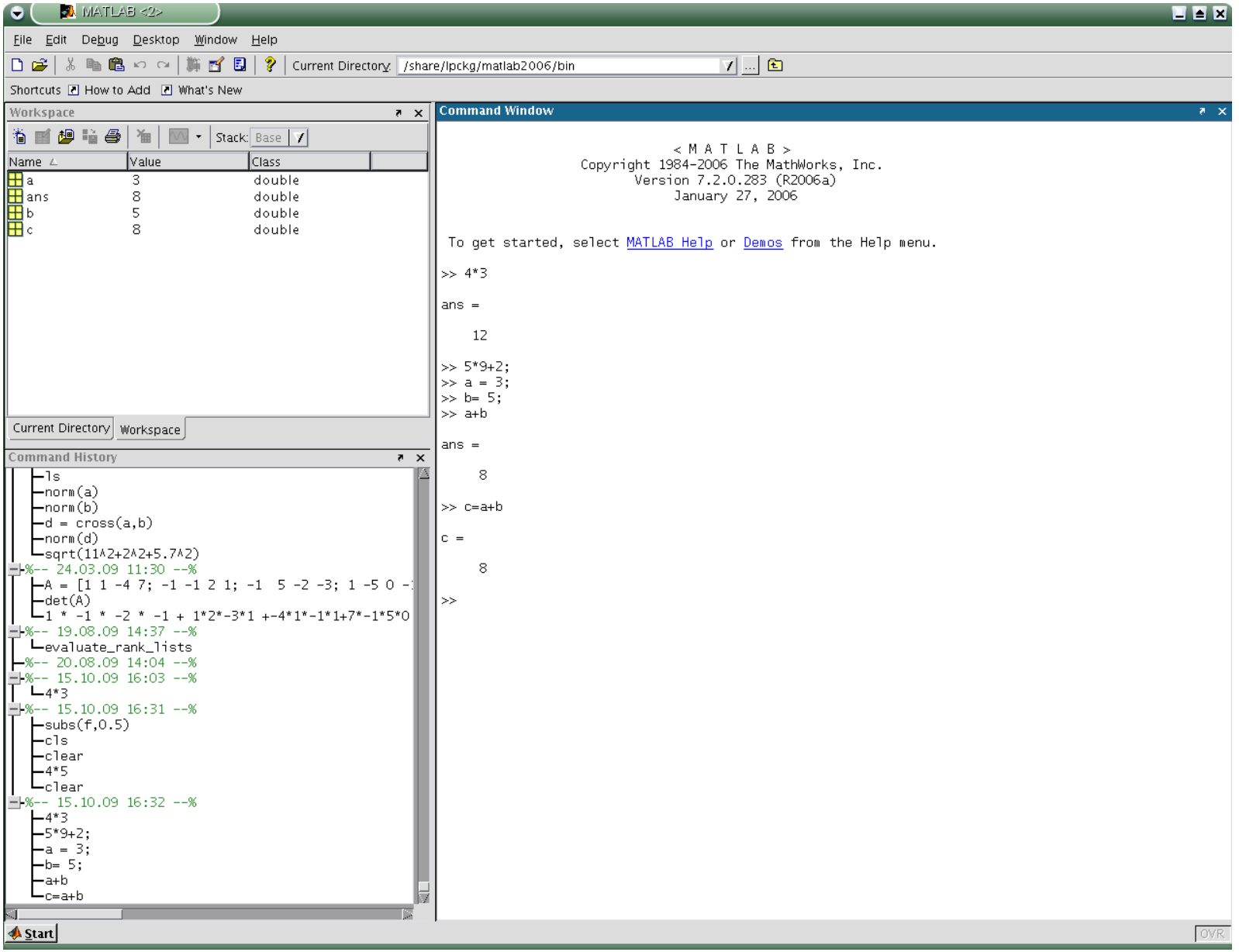
Getting started with Matlab

Topics:

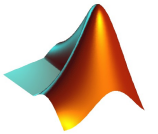
- User Interface
- Simple calculations
- Matrices/Vectors
- Functions
- Loops/conditions
- Plots



User Interface



Simple Calculations



Using Matlab as calculator:

```
>> 5*2
ans =
    10
```

(Matlab uses the Variable `ans` to store the result of an unassigned calculation)

```
>> (2+3)*5;
```

(A semicolon at the end of the command suppresses the output. Interally, the variable `ans` is set to 25.)

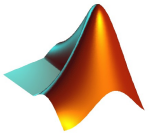
Defining Variables:

```
>> a = 3;
>> b = 5;
>> c = a+b;
```

(c = 8)

- Variables are available during your entire session and appear in the workspace
- The workspace can be cleared with the command `>> clear`
- Matlab has some pre-defined constants, e.g. `pi`

Simple Calculations



The way in which numbers appear

The function `format` controls the numeric format of the dispayed values.The function affects only how numbers are displayed, not how MATLAB computes or saves them.

```
>> format short;
```
5 digits (the default)

```
>> format long;
```
15 digits

```
>> format rat;
```
try to represent the answer as a rational number

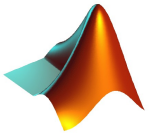
Examples for `pi` *:*

short: `3.1416;`

long: `3.14159265358979;`

rat: `355/133;`

Simple Calculations



Mathematical Functions:

Arithmetic functions: `+ , - , * , /`

Exponential functions: `^ , exp, log, log10`

Trigonomertic functions: `sin , cos, tan , asin, acos , atan`

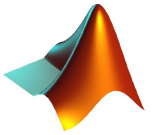
Other functions: `round, mod, abs, sign ...`

Examples:

```
>> 4*x^5;
```
(= $4x^5$)

```
>> exp(3)*sin(x);
```
(= $e^3 \cdot \sin(x)$)

```
>> mod(12,5);
```
(= 2)



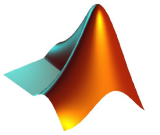
Exercise

Calculate the following expressions:

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

```
(for a=1, b=5, c=-2)
```

$$\log\left(e^{2+\cos\pi}\right)$$



Exercise

Calculate the following expressions:

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

(for a=1, b=5, c=-2)

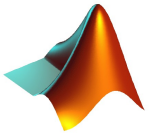
Solutions:

```
>> a = 1;
>> b = 5;
>> c = -2;
>> b-a/(b+(b+a)/(c*a))

ans =

       4.500
```

$$\log\left(e^{2+\cos\pi}\right)$$

```
>> log(exp(2+cos(pi)))

ans =

       1
```

Vectors / Matrices in Matlab



Matrices are the basic data element of Matlab (MATLAB = **MAT**rix **LAB**oratory)

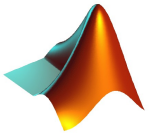
Entering Matrices:

- Separate the elements of a row with blanks or commas.
- Use a semicolon to indicate the end of each row.
- Surround the entire list of elements with square brakets [ ]

Example:

```
>> A = [3 2 4; 1 2 0; -2 4 3]

A =

     3     2     4
     1     2     0
    -2     4     3
```

Vectors / Matrices in Matlab



Row-vectors are 1 x n – matrices and column-vectors are n x 1 - matrices.

row-vector

```
>> a = [4 -1 2]
a =
     4 -1 2
```

column-vector:

```
>> b= [4; -1; 2]
b =
     4
    -1
     2
```

Creating Matrices from vectors:

column-vectors:

```
>> a = [4; -1; 2];
>> b = [2; 3; -3];
>> c = [1; 2; 0];

>> C = [a b c]

C =
     4     2     1
    -1     3     2
     2    -3     0
```

row-vectors:

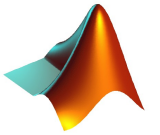
```
>> a = [4 -1 2];
>> b = [2 3 -3];
>> c = [1 2 0];

>> D = [a; b; c]

D =
     4    -1     2
     2     3    -3
     1     2     0
```

Concatenation of matrices:

```
E = [ C D; D C];
```

Note: Dimensions must be consistent!

Vectors / Matrices in Matlab



Since everything is a matrix in matlab, all scalar operations can also be applied to vectors/Matrices (as longs as the dimensions are consistent)

Examples:

```
>> A+4;
>> sin(A);
>> mod(A,2);
>> mod(A,B);
```

Scalar opreations are applied component-wise and a matrix of same dimension is returned.

Operations which are well-defined for matrices are NOT applied component-wise

Example: Matrix- Multiplication

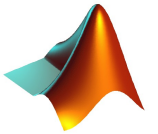
```
>> A = [1 2 3; 2 1 0; 1 2 1];  >> B = [1 2 1; 1 2 1; 1 2 1]
```

```
>> A*B

ans =  6    12     6
       3     6     3
       4     8     4
```

component-wise:

```
>> A.*B

ans =  1    4    3
       2    2    0
       1    4    1
```

Vectors / Matrices in Matlab



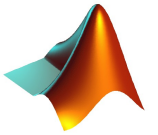
Various functions on matrices:

`A'`	Transpose-operator
`size(A)`	# of rows and colums (returned in a row vector of dimension 2)
`diag(A)`	diagonal of a square matrix (returned in a column vector)
`det(A)`	Determinant
`inv(A)`	Matrix-Inversion
`lu(A)`	LU-Decompostion of
...	

Generating matrices:

`zeros(m,n)`	nxm-matrix with all zeros
`ones(m,n)`	nxm-matrix with all ones
`eye(m)`	mxm-identity matrix
`rand(m,n)`	nxm-matrix with random values
`magic(m)`	mxm-magic matrix

Vectors / Matrices in Matlab



Accessing single matrix-elements:

`A(i,j)`	Element in row `i` and column `j` of the Matrix A
`A(i)`	Usually used to get the i-th component of a vector

Accessing multiple matrix-elements:

`A(:,j)`	`j` -th column vector of the Matrix A
`A(i,:)`	`i` -th row vector of the Matrix A
`A(1:3,j)`	first 3 elements of `j` -th column vector of the Matrix A

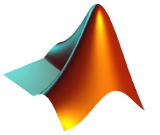
Additional Notes:

these operations can be used read AND to manipulate data.

`>> c = A(1,2);`	store the element *A12* in the variable `c`
`>> A(1,2) = -3;`	set A_{12} to the value `-3`
`>> A(:,2) = [];`	delete the second column of A

You can use the keyword end to access the last entry:

`>> A(end,2);`	last entry of the second column-vector



Exercise

Prerequisite: Define the Matrix $A = \begin{matrix} 1 & 1 & 3 & 2 \\ 2 & 4 & 7 & 0 \\ 1 & 0 & 2 & 5 \\ 8 & 3 & 1 & 2 \end{matrix}$

Tasks:

- Extract the upper left 3x3- submatrix of A

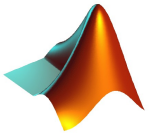
- Compute the dot product of the main diagonal of A and the fourth column vector of A.

- Define a Matrix B which arises from interchanging the 1st and the 4th row of A

- Set the entire 3rd column of A to '1'

- Insert the row-vector (1 2 3 4) at the bottom of A:

Vectors / Matrices in Matlab



Exercise

Prerequisite: Define the Matrix $A = \begin{matrix} 1 & 1 & 3 & 2 \\ 2 & 4 & 7 & 0 \\ 1 & 0 & 2 & 5 \\ 8 & 3 & 1 & 2 \end{matrix}$

Tasks:

Solutions:

- Extract the upper left 3x3- submatrix of A

```
>> A(1:3,1:3)
```

- Compute the dot product of the main diagonal of A and the fourth column vector of A.

```
>> diag(A)'*A(:,4)
```

- Define a Matrix B which arises from interchanging the 1st and the 4th row of A

```
>> B = [A(4,:); A(2:3,:);A(1,:)]
```

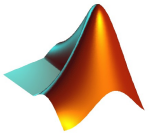
- Set the entire 3rd column of A to '1'

```
>> A(:,3) = 1;
```

- Insert the row-vector (1 2 3 4) at the bottom of A:

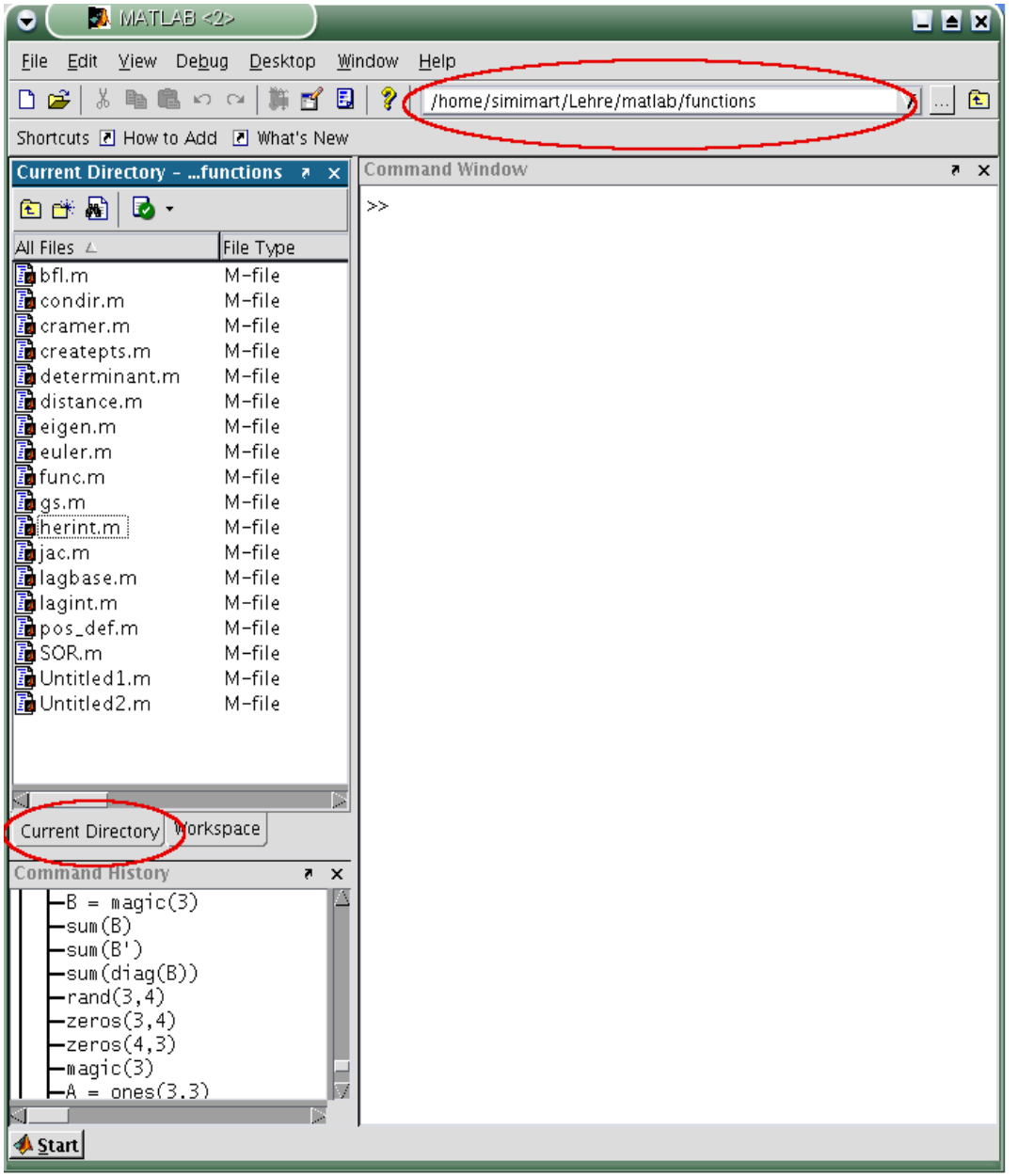
```
>> A(end+1,:) = [1 2 3 4];
```

Writing Functions in Matlab

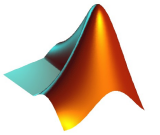


- A self-defined Function is stored in a file: <functionName>.m

- All .m-files in the current directory are available in your command window



Writing Functions in Matlab



Skeleton of a function:

```
 function [ output_args ] = <functionName>( input_args )
%FUNC Summary of this function goes here
%   Detailed explanation goes here
```

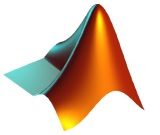
Example:

Function which returns the sum of the top-left and the bottom-right element of a matrix:

nonsense.m

```
function [ r ] = nonsense( A )

    r = A(1,1) + A(end,end);

end
```

Note: *Neither the type, nor the dimensions of the parameters has to be specified*



Functions with more than one Parameter:

Get the two different Products of two Matrices in one function call:

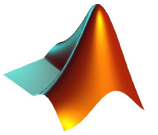
```
function [ AB BA ] = MatrixMult( A,B )

   AB = A*B;
   BA = B*A;

end
```

```
>> A = [1 2 3; 2 3 4; 1 0 2];
>> B = [0 1 2; 2 4 1; 5 3 1];
>> [M1,M2] = MatrixMult(A,B);
```

→

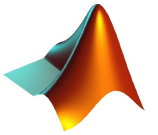
```
M1=   9   12   5
      6   14   7
     18   25   12
```

```
M2 =   8   11   4
      13   20   4
      14   23   7
```



Exercise:

Write a function ld(x) which returns the logarithm of a number x to the basis 2.

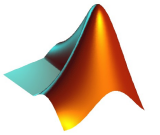


Exercise:

Write a function ld(x) which returns the logarithm of a number x to the basis 2.

Solution:

```
function [ r ] = ld( x )
 r = log(x)/log(2);
end
```



Loops

```
for i = N
      ...
   commands
      ...
end
```

repeats the commands for each element of a vector `N`. The currently processed vector element is available inside the loop via the running parameter (here: `i`)

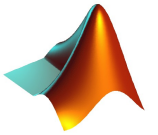
A vector with incremental elements can easily be created with the colon (:) -operator

```
>> t = 1:10
```
default-increment is 1

```
t = 1  2  3  4  5  6 7  8  9  10
```

```
>> s = 2:0.2:3
```
explicitly setting the default

```
s = 2.0  2.2  2.4  2.6  2.8  3.0
```



Conditions

```
if (expression)
        ...
    commands
        ...
end
```

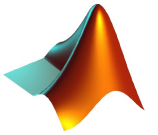
If the expression is true, the commands are executed, otherwise the program continues with the next command immediately beyond the `end` statement

You can create alternative command sections with `else` and `elseif`:

```
if (expression1)
        ...
elseif (expression2)
        ...
elseif (expression3)
        ...
else
        ...
end
```

as soon as one of the expressions is true, the respective commands are executed and the program continues beyond the `end` statement.

The command after an `else`-statement are executed when none of the conditions are true.



Conditions

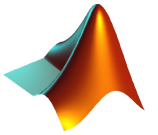
Examples for expressions:

`(a < b)`	True if a is less than b
`(a == b)`	True if a is equal to b
`(a ~= b)`	True if a is not equal to b

Compound statements:

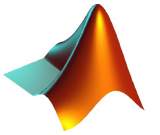
```
(x < 3 & y < 4)   Logical AND
(x < 3 | y < 4)   Logical OR
```

Loops / Conditions



Exercise: Write a function `fact_or_sum(n)` which computes $\sum_{i=1}^{n} i$ if n is even and *n! if n is odd (for n>0)*

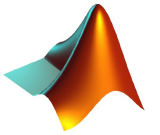
Loops / Conditions



Exercise: Write a function `fact_or_sum(n)` which computes $\sum_{i=1}^{n} i$ if n is even and *n! if n is odd (for n>0)*

Solution:

```
function [ res ] = fact_or_sum( n )

    res = 1;
    for i=2:n
        if (mod(n,2) == 1)
            res = res * i;
        else
            res = res + i;
        end
    end
end
```



Conditional Loops

```
while (condition)
      ...
   commands
      ...
end
```

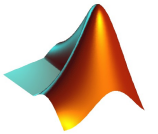
The loop is repeated as long as the condition is satisfied

The `break` command

`break` can be used to force any loop to end;

```
x = 1;
while (1 == 1)
   x = x+1;
   if x>10
      break
   end
   ...
end
```

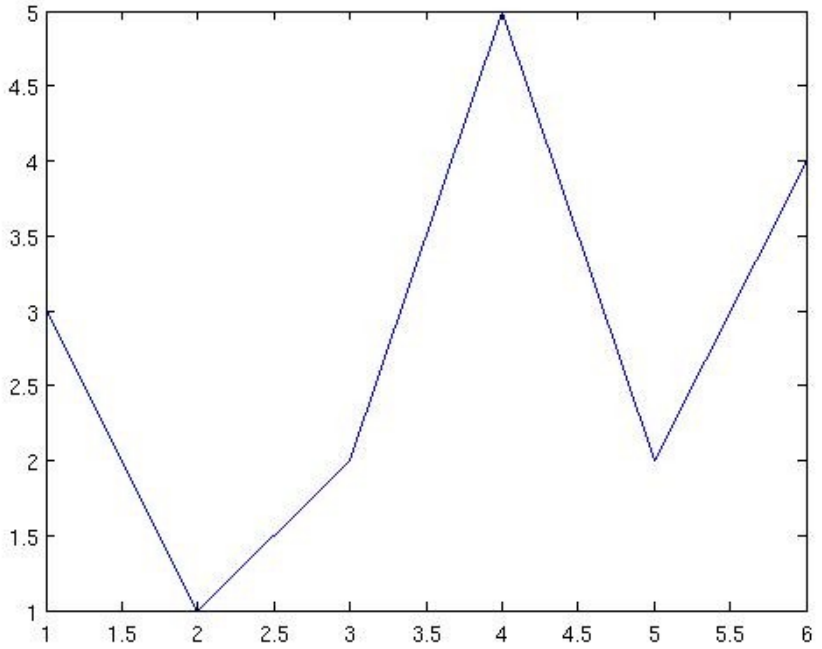
Plots



The plot function has different forms, depending on the input arguments.

- If `y` is a vector, `plot(y)` produces a piecewise linear graph of the elements of `y` versus the index of the elements

```
>> y = [3 1 2 5 2 4];

>> plot(y)
```



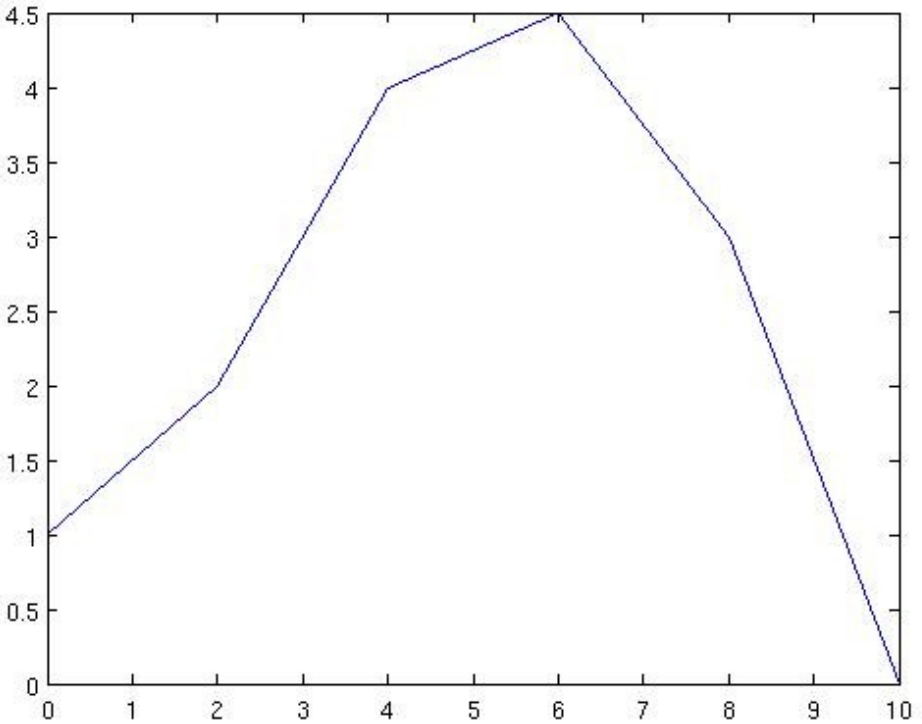
Note: Graphing functions automatically open a new figure window if there are no figure windows already on the screen. If a figure window exists, it is used for graphics output.

Plots

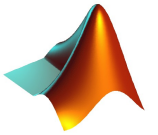
- If you specify two vectors as arguments, `plot(x,y)` produces a graph of `y` versus `x`.

```
>> x = [0:2:10];
>> y = [1 2 4 4.5 3 0];

>> plot(x,y)
```

x	0	2	4	6	8	10
y	1	3	4	4.5	3	0



Plots



Specifying line styles, colors and markers

```
plot(x,y,'color_style_marker')
```

color_style_marker is a string containing from one to four characters (enclosed in single quotation marks) constructed from a color, a line style, and a marker type. The strings are composed of combinations of the following elements:

Color:

'c'	cyan
'm'	magenta
'y'	yellow
'r'	red
'g'	green
'b'	blue
'w'	white
'k'	black

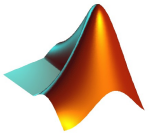
Line Style:

'-'	solid
'--'	dashed
':'	dotted
'.-'	dash-dot
	no line

Marker Type:

'+'	plus mark
'o'	unfilled circle
'*'	asterisk
'x'	letter x
's'	filled square
	no marker
...	

Plots



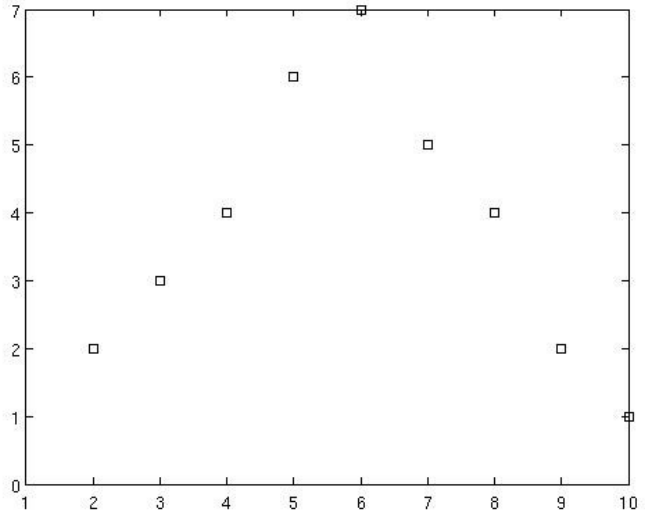
Examples:

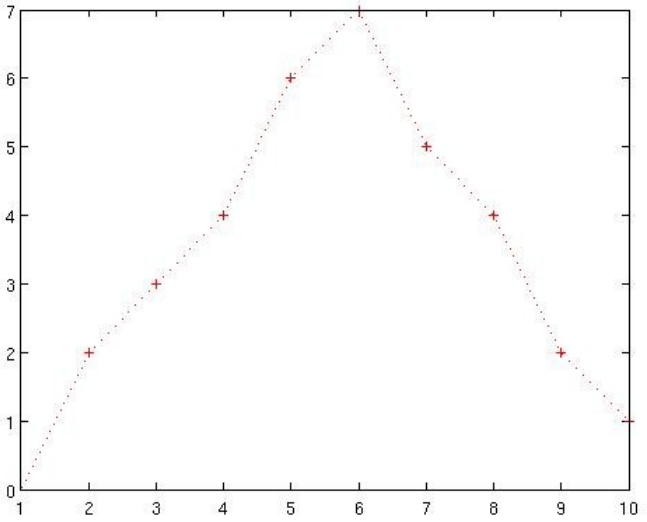
```
>> plot(x,y,'ks')
```

plots black squares at each data point, but does not connect the markers with a line.

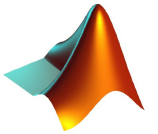
```
>> plot(x,y,'r:+')
```

plots a red-dotted line and places plus signs at each data point





Plots



Plotting multiple data sets in one graph

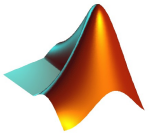
Multiple x-y pairs as arguments create multiple graphs with a single call to plot

```
>> plot(x,y,x,y2,x,y3)
```

Adding plots to an existing graph

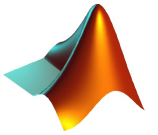
if you type the command `hold on` MATLAB does not replace the existing graph when you issue another plotting command; it adds the new data to the current graph, rescaling the axes if necessary.

```
>> plot(x,y);
>> hold on
>> plot(x2,y2);
>> hold off
```



Exercise

Create a plot of sin(x) with 100 data points in the interval $[0, 2\pi]$



Exercise

Create a plot of sin(x) with 100 data points in the interval [0,2π]

Solution:

```
>> s = 2*pi/99;

>> x = 0:s:2*pi;

>> y = sin(x);

>> plot(x,y,'r:+')
```