Einführung in MATLAB

Sommer Campus 2004

Teil I

G.Brunner, B. Haasdonk, K. Peschke

Lehrstuhl für Mustererkennung und Bildverarbeitung
Uni Freiburg

Einteilung des Kurses

- Teil 1 (Mittwoch)
  - allgemeine Einführung in MatLab

  Klaus Peschke
- Teil 2 (Donnerstag)
  - Fortgeschrittene Technik (Optimierung, Profiling, Mex-Interface)

  Bernard Haasdonk
- Teil 3 (Freitag)
  - MATLAB Bibliotheken (Standard, Open-Source)

  Gerd Brunner

Contents

About MATLAB

- MATLAB is...
  - is a high-performance language for technical computing
  - is an interactive system whose basic data element is an array
  - an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation

- the name MATLAB stands for matrix laboratory
- MATLAB is very intuitive

Dokumentation / Help

- command line help:

```
help [functionname]
lookfor [functionname]
```

- very useful and comprehensive help navigator in the menu bar:
  - ➔ Help
    - ➔ Help Navigator
- on-line help at:

  http://www.mathworks.com/[1)]

[1)] diagrams and examples are copied from MATLAB 6.0.5

Dokumentation / Tutorials

- command line

```
demos
```

- over the menu bar:
  - ➔ Help
    - ➔ Demos
- on-line demos / tutorials at:

  http://www.mathworks.com/

  or via google

Getting started

- ...an example

```
A = [1 0 5];
zero_pos = find (A == 0);
```

Variable names / keywords

- Variables need not to be declared
- Variables are defined by assigning values to them
- MATLAB is case sensitve
- keywords are reserved names

```
>> iskeyword
ans =
    'break'
    'case'
    'catch'
    ...
```

- variable names that equals a function name will block function call to that function
  - check with *which*

```
>>which -all disp
disp is a build-in function.
...
```

Operators

- Matrix operators

```
help ops
  Operators and special characters.
  Arithmetic operators.
    plus       - Plus                    +
    minus      - Minus                   -
    mtimes     - Matrix multiply         *
    times      - Array multiply          .*
    power      - Array power             .^
    mldivide - Left matrix divide        \
    mrdivide - Right matrix divide       /
    ldivide    - Left array divide       .\
    rdivide    - Right array divide      ./
```

- no increment ++ operator !

Operators

- Relational operators

```
Relational operators.
  eq      - Equal                    ==
  ne      - Not equal                ~=
  lt      - Less than                <
  gt      - Greater than             >
  le      - Less than or equal       <=
  ge      - Greater than or equal    >=
```

- use: *help keyword* to get further help

```
help eq
```

Special characters

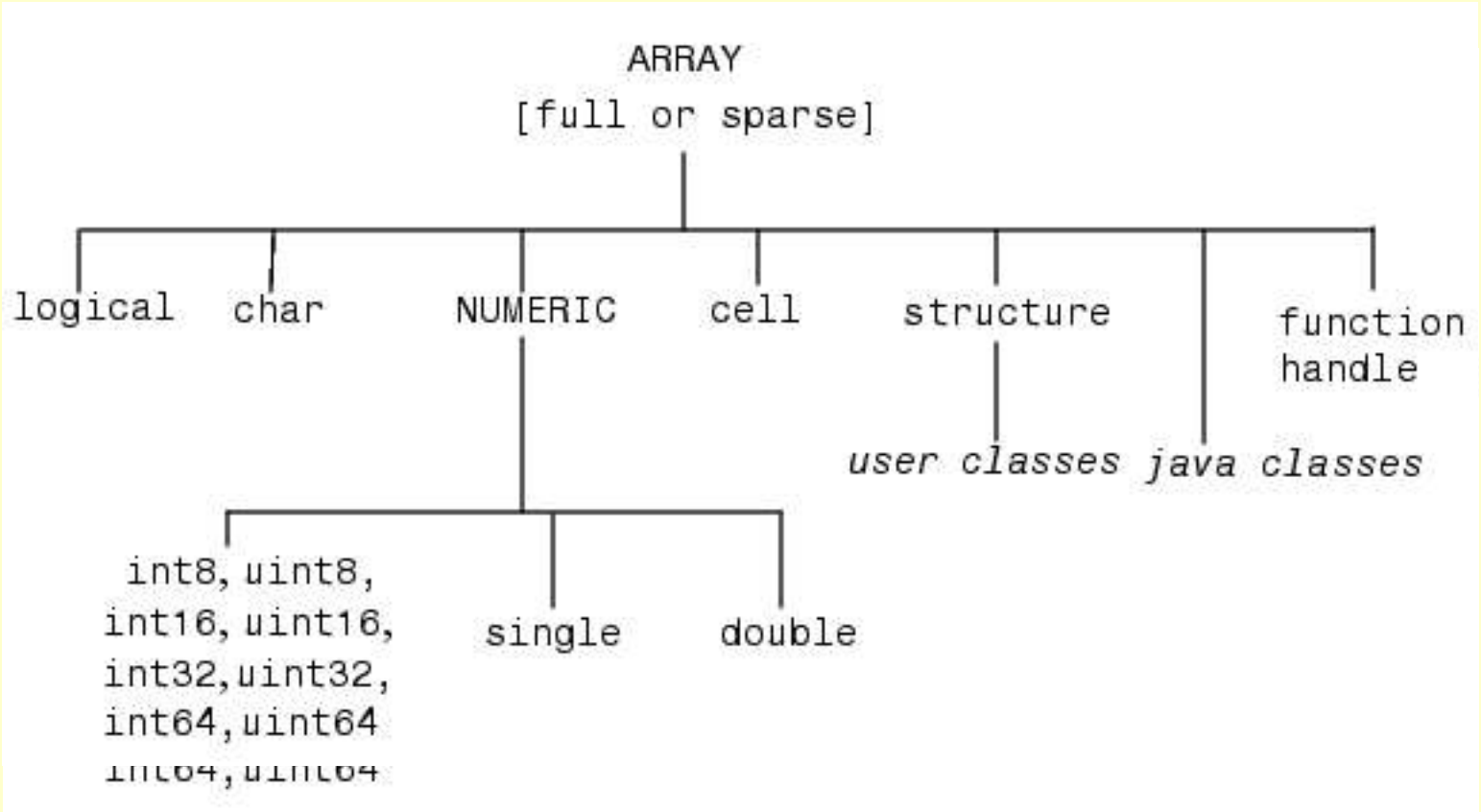
- Special characters

```
Special characters.
   colon      - Colon                                  :
   paren      - Parentheses and subscripting        ( )
   paren      - Brackets                            [ ]
   paren      - Braces and subscripting             { }
   punct      - Function handle creation             @
   punct      - Semicolon                            ;
   punct      - Comment                              %
   punct      - Invoke operating system command      !
   punct      - Assignment                           =
   punct      - Quote                                '
   ctranspose - Complex conjugate transpose          '
```

Data types

- all data is stored in arrays

```
A = [ data1, data2 ... ; ...]
```



Data types

- logical
  - contains true (1) and false (0)
- char
  - array of char = string
- numeric
  - double
  - (un-)signed integer

```
>> A=([1 2 3 4 5] >= 3)
A=
     0 0 1 1 1
```

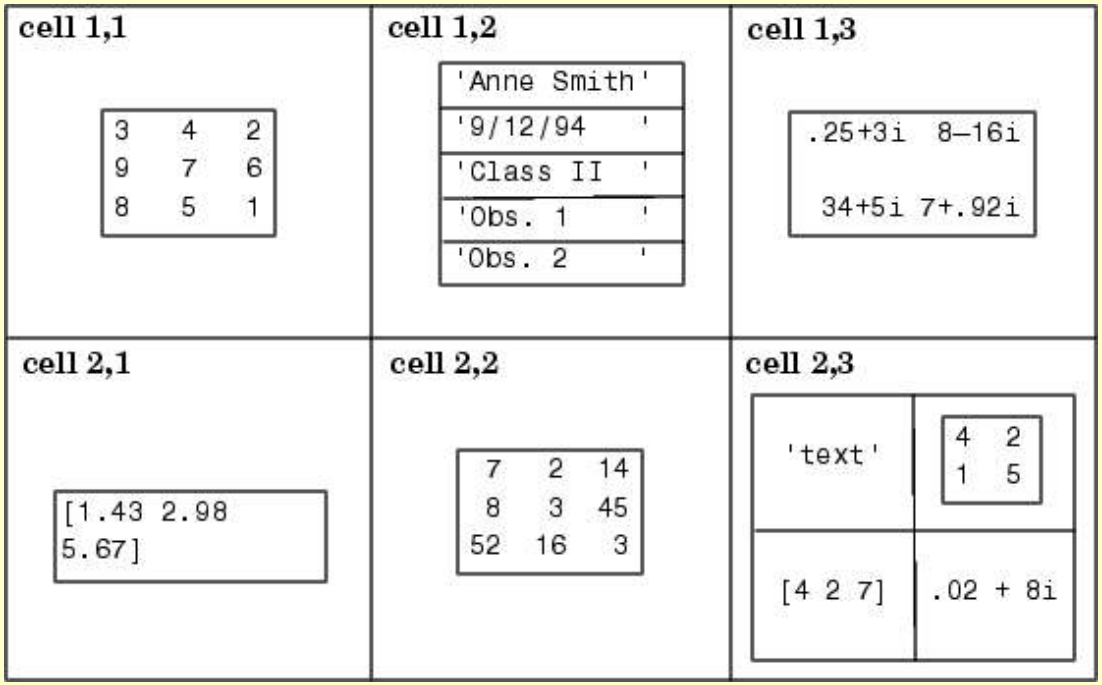
```
>> S = ['H'  'A'  'L'  'L' 'O']
S =
HALLO
```

```
>> T=[1 2 3 4 5; 6 7 8 9 10]
T =
     1  2  3  4  5
     6  7  8  9  10
```

```
>> U = int8 ( ones (3,2))
U =
     1  1
     1  1
     1  1
```

Cell arrays

- cell array to store different data structures



Multidimensional arrays / indexing

- accessing values with the *()*

index counting starts with 1
not 0 (like in C/C++) !!

```
A = [1 2 3 4 5];
a3 = A(3)
a3 =
     3
```

- the *:* means here "all"
  - accessing the rows
  - accessing the coloumns

```
A = [1 2 3 4 5; 6 7 8 9 10];
row2 = A(2,:)
row2 =
     6 7 8 9 10
```

```
A = [1 2 3 4 5; 6 7 8 9 10];
row2 = A(2,:);
A ( 1, :) = row2;
```

- the *end* keyword

  indexes the last element

```
A = [1 2 3 4 5; 6 7 8 9 10];
lastel = A( 1,end)
lastel =
     5
```

Single indexing vs. subscript values

- single indexing means index to a nx1 vector containing all the data (faster access)
- the nx1 vector is created columnwise (“Fortran style”)

```
A =    1    2    3    4    5
       6    7    8    9   10
sgA = A(:)
sgA =
    1
    6
    2
    7
    3
    8
    4
    9
    5
   10
```

```
A = [1 2 3 4 5; 6 7 8 9 10];
a6 = A(6)
a6 =
     8
```

```
A = [1 2 3 4 5; 6 7 8 9 10];
a6 = sgA(6)
a6 =
     8
```

Converting bw. subscript and linear indexing

- search and show all even elements

```
A = [1 2 3 4 5; 6 7 8 9 10];
[idx] = find( mod(A,2) == 0)
```

```
idx =
    2
    3
    6
    7
   10
```

- just using *idx* on A

```
A( idx)
ans =
     6
     2
     8
     4
    10
```

- convert to subscripts

```
[r_idx, c_idx] = ind2sub( size( A),idx);
```

```
r_idx =
     2
     1
     2
     1
     2
```

```
c_idx =
     1
     2
     3
     4
     5
```

Converting bw. subscript and linear indexing

- search and show all even elements (2)

```
A = [1 2 3 4 5; 6 7 8 9 10];
[r_idx c_idx] = find( mod(A,2) == 0)
```

```
r_idx =
     2
     1
     2
     1
     2
```

```
c_idx =
     1
     2
     3
     4
     5
```

- just using *r_idx* and *c_idx* on A

```
A(r_idx,c_idx)
ans =
     6     7     8     9    10
     1     2     3     4     5
     6     7     8     9    10
     1     2     3     4     5
     6     7     8     9    10
```

- use function sub2ind

```
idx = sub2ind(size(A),r_idx,c_idx);
A( idx)
```

Converting bw. subscript and linear indexing

- search and show all even elements (3)

```
A = [1 2 3 4 5; 6 7 8 9 10];
A( find( mod(A,2) == 0))
```

- very easy !! ;-)

Control statements

- if ...

```
if logical_expression
    statements
end
```

```
if n < 0
    disp('error')
elseif rem(n,2) == 0
    A = n/2;
else
    A = (n+1)/2;
end
```

- for ...

```
for index = start:increment:end
    statements
end
```

```
A = [1:10];
for h = 1:2:10
    A(h) = 0;
end
```

```
for matrix
    statements
end
```

```
A = [1:10];
idx = [1 3 5 7 9];
for h = idx
    A(h) = 0;
end
```

Control statements

- while ... if *expression*
  statements
  end

```
n = 0;
while prod(1:n) < 1e3
    n = n + 1;
end
```

- switch ... switch *expression*
  case
  statements
  case
  statements
  :
  otherwise
  end

```
switch invar
    case 'a'
        disp('vowel');
    case 'e'
        disp('vowel');
    otherwise
        disp('consonant');
end
```

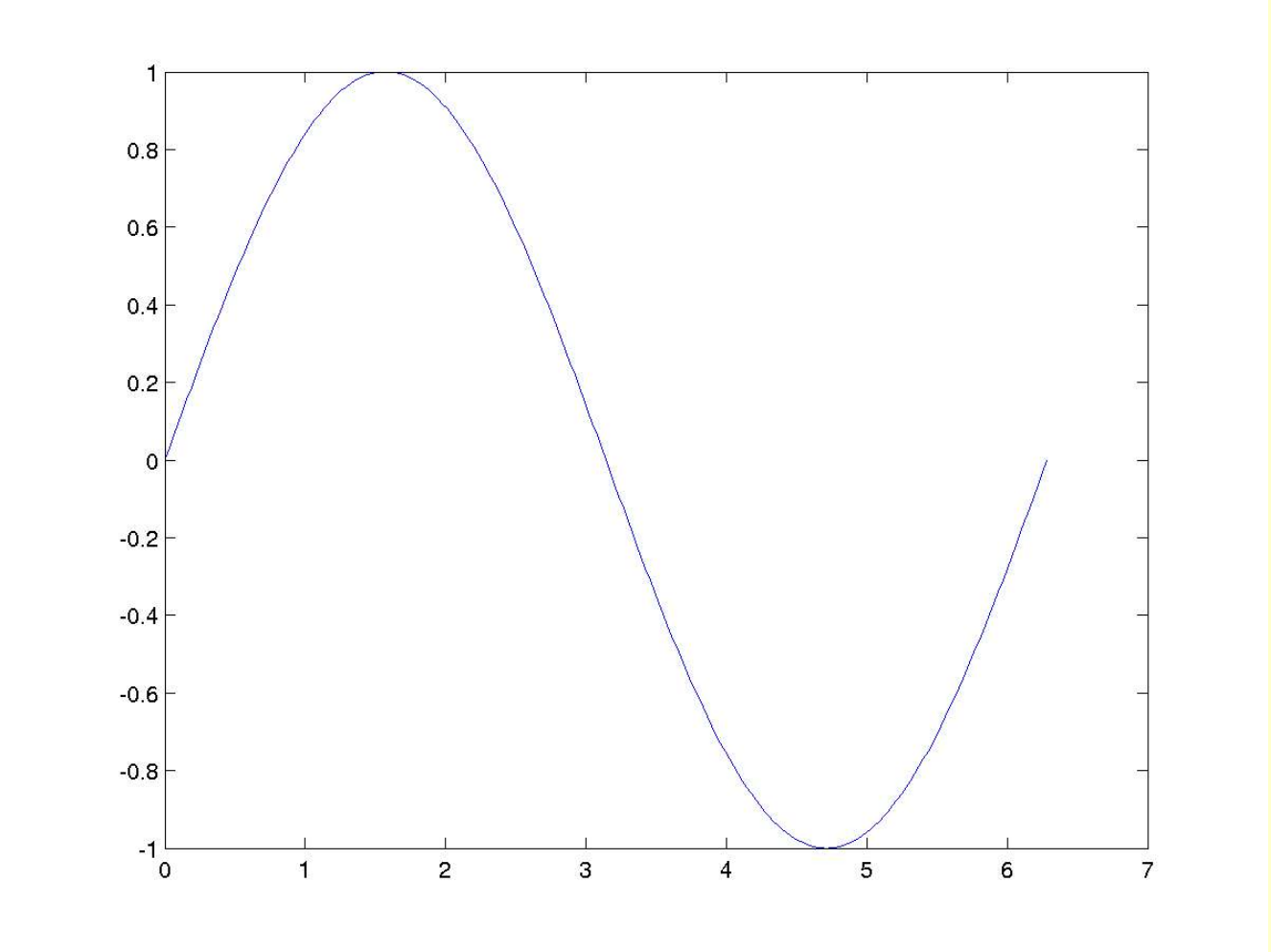
Control statements

- continue: skipps commands in a for or while loop
- break: terminates the execution of a for or while loop
- try...catch: error handling
- return: returns to the invoking function

Plotting

- making a diagram with: *plot (x1,x2,parameters)*

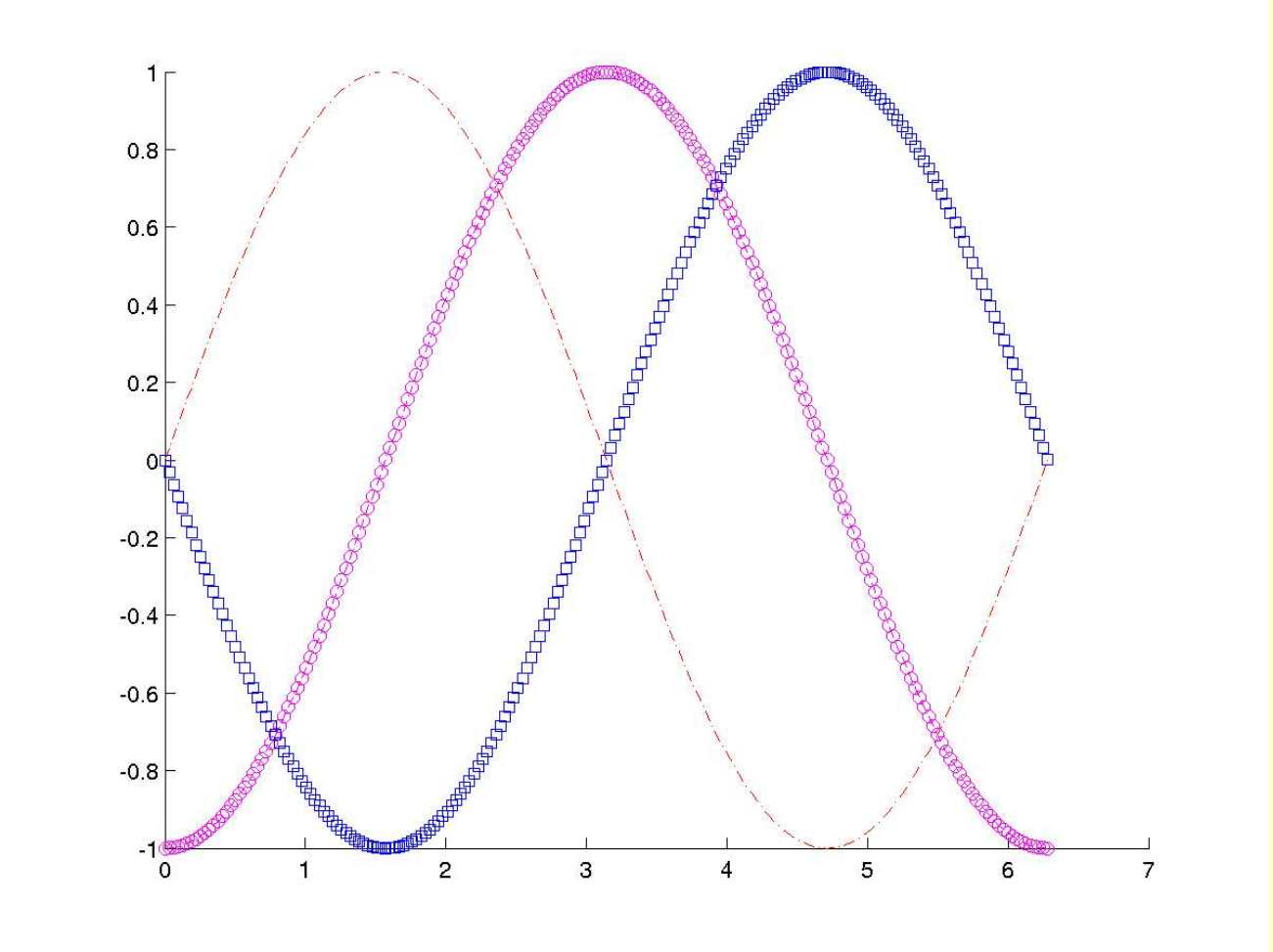
```
x=0:pi/100:2*pi;
y=sin(x);
plot (x,y);
```



Plotting

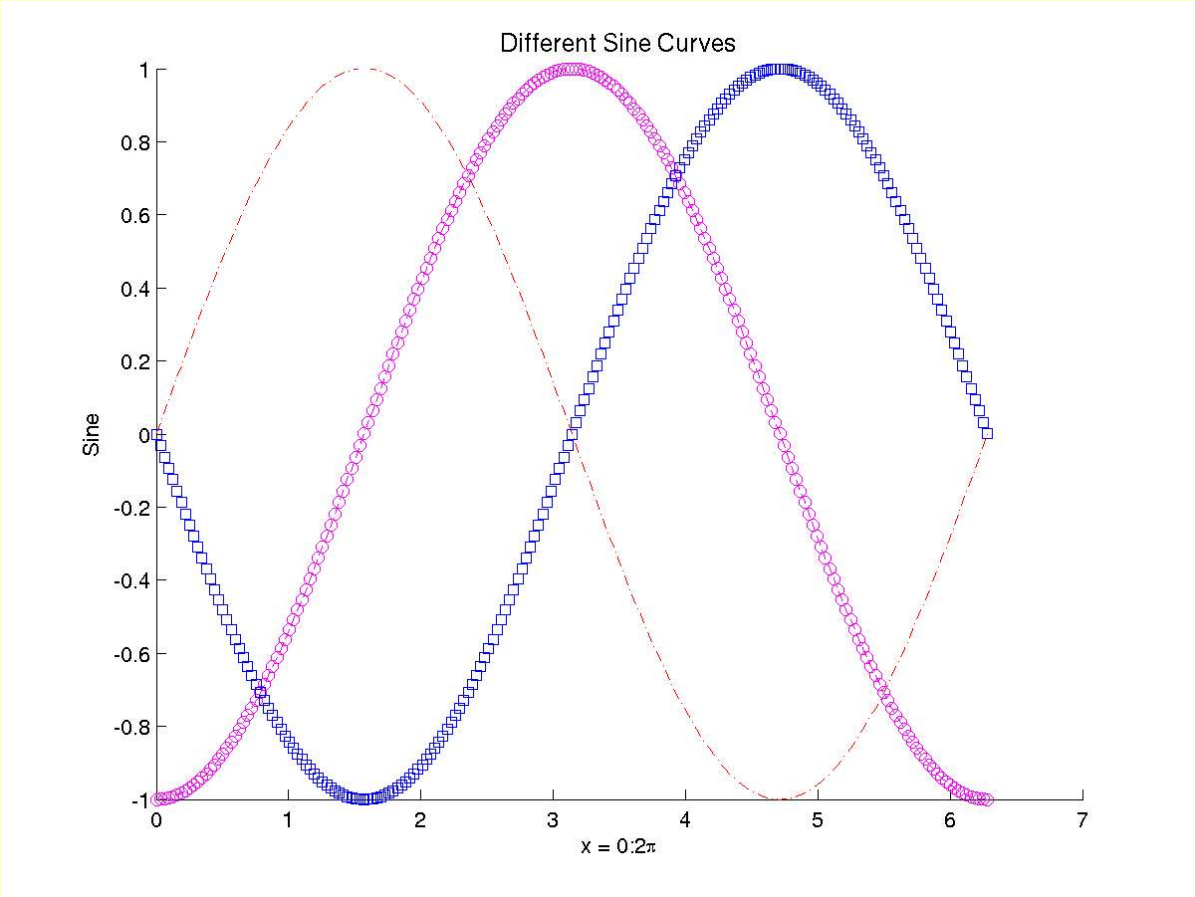
- ...using *linespec* parameter

```
x=0:pi/100:2*pi;
y1=sin(x);
y2=sin(x-pi/2);
y3=sin(x-pi);
hold;
plot (x,y1,'-.r');
plot (x,y2,'om');
plot (x,y3,'sb');
hold off
```



Plotting

- ...using labels




```
xlabel('x = 0:2\pi');
ylabel('Sine');
title('Different Sine Curves','FontSize',12);
```

Plotting images

- using function *imagesc (...)*

```
im=imread ('saturn.tif');
imagesc( im);
colormap ( gray);
```

- function imshow (...) to plot an image
  - function relates to image processing toolbox

Plotting

- ...plotting several diagrams/images in one window

  using subplot(rows, cols, idx)
- ...interactive editing the plot (menu)

- ...using graphic handles

```
plot (...);
set( gca, 'XDir','reverse') ;
```

- ...graphical input

```
[x y] = ginput ( npoints);
```

Writing / Reading data (file I/O)

- writing data with:

  save filename var1 var2 ... -option

```
a = 33;
b= 5;
save result.mat a b
```

  save ('filename','var1','var2',...'-option');

```
a = 33;
b= 5;
save ('res_ascii','a','b','-ASCII');
```

Writing / Reading data (file I/O)

- reading data with:

  load filename -option

```
load res_ascii -ascii      % returns a two dimensional array
load result                % returns variable a and b
```

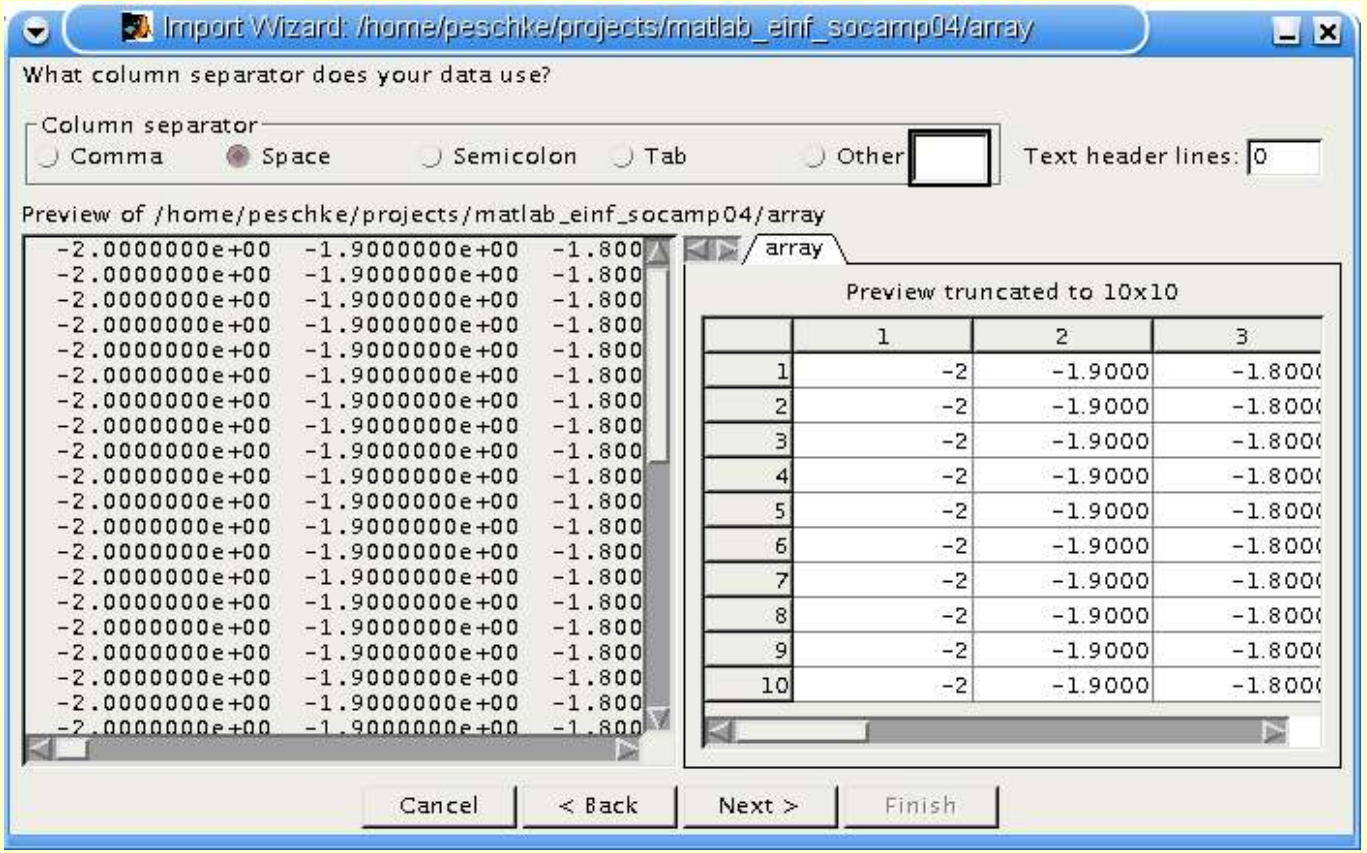
- also functional form:

  load ('filename','-option');

  e.g. when file name is given as a string or contains spaces etc.

Import Data

- import data (click right on file in file browser)...



Scripts and Functions

- Scripts are just commands (stored in a file) executed one after another in the global space
- Functions...
  - are stored in a separate .m-file, that has the same name as the function
  - operate on variables on their own work space
  - can accept input arguments
  - can return values

MATLAB functions

- Definition of a function
  - keyword function
  - function name
  - input argument
  - output argument

- Example: find all even elements of an array

```
function [r, c] = find_even_el( V)

[r, c] = find( mod(V,2) == 0);
```

Functions

- ...running a function

```
A = [1 2 3 4 5; 6 7 8 9 10];
[r,c] = find_even_el(A)
```

- or with a function handle

```
A = [1 2 3 4 5; 6 7 8 9 10];
[r,c] = feval (@find_even_el, A)
```

MATLAB functions

- more advance example

```
>> type mean
function y = mean(x,dim)
%MEAN   Average or mean value.
%   For vectors, MEAN(X) is the mean value of the elements in X. For
%   matrices, MEAN(X) is a row vector containing the mean value of
%   each column.  For N-D arrays, MEAN(X) is the mean value of the
...
if nargin==1,
  % Determine which dimension SUM will use
  dim = min(find(size(x)~=1));
  if isempty(dim), dim = 1; end
  y = sum(x)/size(x,dim);
else
  y = sum(x,dim)/size(x,dim);
end
```

MATLAB functions

```
>> help mean
 MEAN   Average or mean value.
    For vectors, MEAN(X) is the mean value of the elements in X. For
    matrices, MEAN(X) is a row vector containing the mean value of
    each column.  For N-D arrays, MEAN(X) is the mean value of the
    ...
```

```
>> lookfor mean
MEAN   Average or mean value.
MEAN2 Compute mean of matrix elements.
DMAE Mean absolute error performance derivative function.
DMSE Mean squared error performance derivatives function.
DMSEREG Mean squared error w/reg performance derivative function.
...
```

Running scripts faster

- loops are slow in MATLAB -> code should be vectorized

```
Y = 10 * rand ( 1000, 500);
V = [1:1000]';

Z1 = zeros(1000,500);
for  m = 1:size( Z, 2)
      Z1( :,m) = V .* Y( :, m);
end
```

takes 7,8 sec

```
Y = 10 * rand ( 1000, 500);
V = [1:1000]';

tic
Z2 = diag( V)*Y;
toc
```

takes 0,8 sec

- pre-allocate arrays

The repmat function

- use repmat (...) for big arrays

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

B = repmat(A,2,3);
B =
   1   2   3   1   2   3   1   2   3
   4   5   6   4   5   6   4   5   6
   1   2   3   1   2   3   1   2   3
   4   5   6   4   5   6   4   5   6
```

Übungen

- Aufgabenzettel
  - Aufgaben (pdf-Datei) und alle Daten unter:

    /misc/database/matlab_sommer_campus04/

- ...oder MATLAB Tutorials/Demos :-)