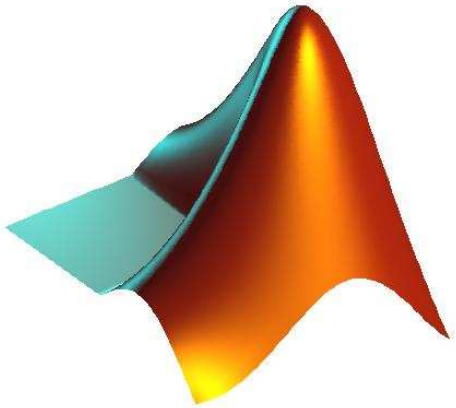
Matlab Tutorial

15.10.2008



Jun. Prof. Matthias Hein

Roadmap of the Tutorial

- **Matlab basics (9.00-12.00):**
  - history, syntax, etc.
  - vectors, matrices and arrays
  - basic programming
  - basic plotting
  - how to write fast code in matlab
- **Practical part (14.00-17.00):**
  - learning by doing

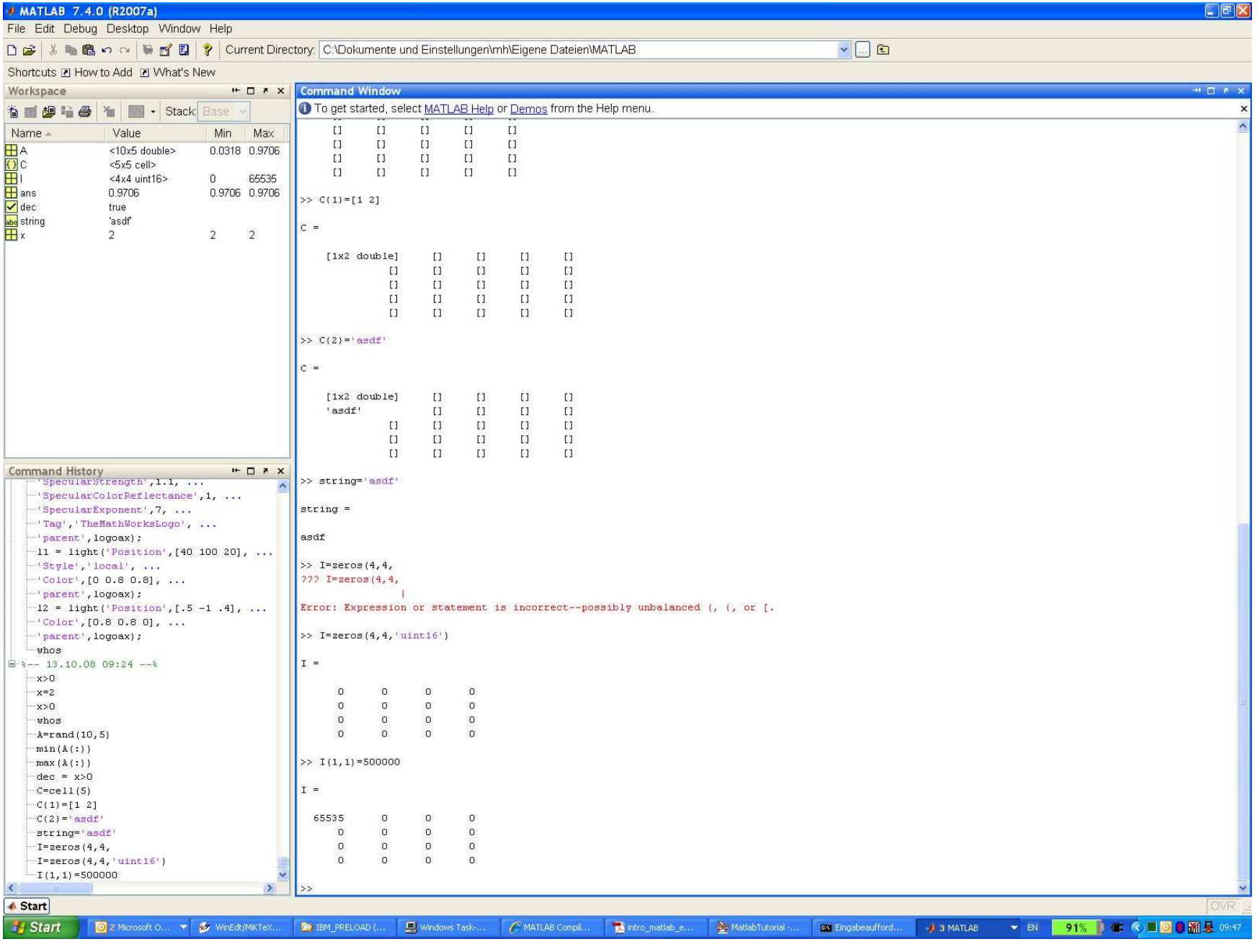
History of Matlab

- Developed by Cleve Moler in order to make the Fortran libraries LINPACK and EISPACK accessible from the command line.
  Today: LINPACK replaced by BLAS and LAPACK.
- Matlab stands for "Matrix Laboratory",
- 1984: MathWorks was founded,
- 9.10.2008: Matlab Version 7.7. (R2008b)

What is Matlab

- High-level language for technical computing
- interpreter (allows fast prototyping)
- scripting programming language
  - just-in-time compilation (since Matlab 6.5),
  - integration of C, C++, Java-Code,
  - object-oriented programming,
  - Matlab compiler to build executable or a shared library using "Matlab compiler runtime" in order to build applications which run without Matlab.

Matlab


MATLAB 7.4.0 (R2007a)
File Edit Debug Desktop Window Help
Current Directory: C:\Dokumente und Einstellungen\mh\Eigene Dateien\MATLAB
Shortcuts How to Add What's New
Workspace
Stack: Base
Name Value Min Max
A <10x5 double> 0.0318 0.9706
C <5x5 cell>
I <4x4 uint16> 0 65535
ans 0.9706 0.9706 0.9706
dec true
string 'asdf'
x 2 2 2
Command History
'SpecularStrength',1.1, ...
'SpecularColorReflectance',1, ...
'SpecularExponent',7, ...
'Tag','TheMathWorksLogo', ...
'parent',logoax);
l1 = light('Position',[40 100 20], ...
'Style','local', ...
'Color',[0 0.8 0.8], ...
'parent',logoax);
l2 = light('Position',[.5 -1 .4], ...
'Color',[0.8 0.8 0], ...
'parent',logoax);
whos
%-- 13.10.08 09:24 --%
x>0
x=2
x>0
whos
A=rand(10,5)
min(A(:))
max(A(:))
dec = x>0
C=cell(5)
C(1)=[1 2]
C(2)='asdf'
string='asdf'
I=zeros(4,4,
I=zeros(4,4,'uint16')
I(1,1)=500000
Command Window
To get started, select MATLAB Help or Demos from the Help menu.
>> C{1}=[1 2]
C =
[1x2 double]
>> C{2}='asdf'
C =
[1x2 double]
'asdf'
>> string='asdf'
string =
asdf
>> I=zeros(4,4,
??? I=zeros(4,4,
Error: Expression or statement is incorrect--possibly unbalanced (, {, or [.
>> I=zeros(4,4,'uint16')
I =
>> I(1,1)=500000
I =
65535
>>
Start
OVR
Start
2 Microsoft O...
WinEdt/MiKTeX...
IBM_PRELOAD (...
Windows Task-...
MATLAB Compil...
intro_matlab_e...
MatlabTutorial -...
Eingabeaufford...
3 MATLAB
EN
91%
09:47

Matlab syntax for entering commands:

```
>> x=1; y=2, plot(x,y,...
'r.')
y =
     2
>>
```

Syntax:

- ; - ends command, suppresses output
- , - ends command, allows output
- ... - allows to continue entering a statement in the next line

Variables:

- MATLAB variable names must begin with a letter (followed by any combination of letters, digits, and underscores),
- MATLAB distinguishes between uppercase and lowercase characters,
- Don't use function names for variables (you will not be able to call the function anymore),

  `>> which -all variable_name`

  checks if variable_name is already used as a function name,
- Don't use keywords for variables (like case, for, while, etc.)
- `i` and `j` are the imaginary numbers (can be overwritten)

```
>> i
ans =
    0 + 1.0000i
```

Types (dynamic typing in Matlab !):

- standard: every variable is either double or logical

  double: `>> x=pi`

  logical: `>> x= pi>2`

- other numerical variable types:

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

- with the command `format` one determines how many digits are shown

```
format short - 3.1416
format long - 3.141592653589793
```

- Matlab does **not** do exact arithmetic !

```
>> e = 1 - 3*(4/3 - 1)
e =
     2.2204e-016
```

Types (continued):

- Matlab allows infinity as value:

```
>> x = 1/0
x =
     Inf
```

```
>> x = 1.e1000
x =
     Inf
```

```
>> x = log(0)
x =
    -Inf
```

- other value `NaN` - not a number:

```
>> x = 0/0
x =
     NaN
```

```
>> x =inf/inf
x =
     NaN
```

```
>> x =inf*0
x =
     NaN
```

- **complex numbers:**

```
>> x = 3 + 5*i
>> x= exp(0.5*pi*i)
```

Basic types of Matlab are matrices

- using basic matrix operations yields very fast code (vectorization) !
- indexing of matrices is column-based.

column vector:	row vector:	general matrix
`>> x=[1; 2; 3]`	`>> x=[1 2 3]`	`A =[1 2 3; 4 5 6]`
`x=`	`x=`	`A =`
`1`	`1 2 3`	`1 2 3`
`2`		`4 5 6`
`3`		

Empty array: `A = []`

Useful functions to generate matrices: `zeros, ones, diag`

`>> A = zeros(2,3)`
`A=`
`0 0 0`
`0 0 0`
	`>> A = ones(2,3)`
	`A=`
	`1 1 1`
	`1 1 1`
		`>> A =diag([1 2])`
		`A =`
		`1 0`
		`0 2`

Caution : `ones(5000)` generates a 5000×5000 matrix of ones.

Concatenating matrices

Useful functions to deal with matrices: `horzcat, vertcat, repmat`

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

`>> C = horzcat(A,B)`
`C=`
`1 2 3 4 5 6`
 	
or
`>> C = [A,B];`
	`>> C = vertcat(A,B)`
	`C=`
	`1 2 3`
	`4 5 6`
	or
	`>> C=[A; B];`
		`>> C =repmat(A,2,2)`
		`C =`
		`1 2 3 1 2 3`
		`1 2 3 1 2 3`
		or
		`>> C=[A, A; A, A];`

Colon operator

Useful operator to create vectors: `:` `operator`

```
>> C = -3:2
C=
    -3 -2 -1  0  1  2
```

```
>> C = -2.5:2:2
C=
    -2.5000 -0.5000  1.5000
```

```
>> C = 5:-1:1
C=
     5  4  3  2  1
```

Caution:

- `9:1` is empty,
- also non-integer increments allowed:
  `-0.5:0.5:0.5` yields `-0.5000  0.0000  0.5000`,
- for a matrix `A(:)` returns entries in linear order,
- very useful for indexing (next slide) and plotting,

Indexing matrices: general use: `A(row, column)`

```
A =
    0.5494 0.5721 0.0492
    0.2741 0.3839 0.3954
```

`>> A(2,3)`	`>> A(2,:)`	`>> A(:,3)`
`ans =`	`ans =`	`ans =`
`0.3954`	`0.2741 0.3839 0.3954`	`0.0492`
		`0.3954`

- **Indices always start at 1 (zero is not allowed) !**
- `A(k,:)` returns the k-th row of `A`,
- `A(:,k)` returns the k-th column of `A`.

Indexing matrices II


```
A =
0.5494 0.5721 0.0492
0.2741 0.3839 0.3954
```

Linear indexing:

storage in column based linear order

```
0.5494 0.2741 0.5721 0.3839 0.0492 0.3954
```

`>> A(5)`	`>> A(1:2:5)`	`>> A(4:end)`
`ans =`	`ans =`	`ans =`
`0.0492`	`0.5494 0.5721 0.0492`	`0.3839 0.0492 0.3954`

- the colon operator works also for normal indexing
  `A(1,2:3)` gives entries `[A(1,2) A(1,3)]`.
- `end` is the maximal index possible for that entry
  `A(1:end,1)` maximal row index is 2
  `A(2,1:end)` maximal column index is 3

The matrix:

```
A =
    0.5494 0.5721 0.0492
    0.2741 0.3839 0.3954
```

Indexing on assignment:

```
>> B=zeros(1,5);
>> B(1,2:3)=A(2,1:2)
B=
    B= 0 0.2741 0.3839 0 0
```

In an assignment the dimensions indexed have to agree: `B=zeros(3,2,5)`

`B(1,:,1:3)=A` or `B(1,:,3)=A(1:2,2)` works

but not

`B(1:3,1:2,1)=A`.

Useful functions:

- `ind = find(X)` - locates all nonzero elements of array X (ind is the linear index)
  `[row,col] = find(X)` - returns the row and column indices of the nonzero entries in the matrix X

Usage:

```
A =
    0.5494 0.5721 0.0492
    0.2741 0.3839 0.3954
>> [r,c]=find(A>0.5)
    r = 1 1
    c = 1 2
```

Useful functions: `size,squeeze`

- `size` - returns the dimensions of an array

```
>> A=zeros(4,3,5);
>> size(A)
ans =
     4 3 5
```

- `squeeze` - eliminates **all** singleton dimensions

```
>> B = A(1,:,:);
>> size(B) returns 1 3 5
>> B = squeeze(B);
>> size(B) returns 3 5
```

Sparse matrix: A matrix is sparse if only a few elements are non-zero.

- Matlab uses **compressed column storage format** - let $A \in \mathbb{R}^{r \times s}$,
  - vector of nzmax elements containing all nonzero elements of A,
  - vector of nzmax elements containing the row indices of nonzero elements of A,
  - vector of s elements containing the pointers to the corresponding column

  memory needed: 8*nzmax + 4*(nzmax+s+1)
- create sparse matrix via: `sparse(r,c,v)` where
  - r is the list of row indices of the non-zero elements,
  - c is the list of column indices of the non-zero elements,
  - v contains the values of the non-zero elements.

  other ways: `spdiags`, `spones`, `sprand`, `sprandn`.

Arithmetic Operators

- $\pm$ used as

  `>> C = A ± B`

  adds/subtracts matrices `A,B`. A and B must be same size unless one of them is a scalar,

- `.*` used as

  `>> C = A.*B`

  is the element-wise product - A and B must be same size unless one of them is a scalar,

- `./` used as

  `>> C = A./B`

  is the element-wise division $C(i,j) = A(i,j)/B(i,j)$,

- `.^` used as

  `>> C = A.^p;`

  is the element-wise p-th power of A.

Arithmetic Operators II

- `*` used as

  `>> C = A*B`

  is the **matrix multiplication of A and B** (columns of A must equal rows of B) or one of them can be a scalar,

- `\` used as

  `>> X=A \ B`

  with $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{m \times s}$

  - $n = m$: the i-th column x_i of X is the **solution of the linear system** $A\,x_i = b_i$, where b_i is the i-th column of B,
  - $n \neq m$: the i-th column of x_i of X is the solution of the under-or overdetermined linear system $A\,x_i = b_i$ (in the least squares sense),

- `'` used as `>> C = A';` is the **transpose of the matrix A.**

All operations use the fast built-in functions (vectorization) !

Logical Operators

Logical type: true or false (1 or 0)

- &, |, ∼ are elementwise logical operators - given two arrays

```
x=[0.1 0.8 1.0];
y=[1.4 0.3 1.2];
```

`x > 0.5`	0	1	1
`y > 0.5`	1	0	1
`x > 0.5 & y >0.5`	0	0	1
`x > 0.5 | y >0.5`	1	1	1
`∼(x > 0.5)`	1	0	0

Logical values can be used to obtain values from an array:

```
>> x(x<0.5)
ans =
    0.1000
```

```
>> y(x<0.5)
ans=
    1.400
```

```
>> y(x<0.5 & y<0.5)
ans=
    Empty matrix:  1-by-0
```

Useful functions for linear algebra:

- `norm(A,p)`, computes the p-**norm** of a matrix/vector A (Caution !)
- `eye(n)`, creates the $n \times n$-**identity matrix**,
- `det(A)`, returns the **determinant** of a square matrix A,
- `rank(A)`, returns an estimate of the **rank** of A,
- `[U,V]=eig(A)`, returns the **eigenvalues** V **and eigenvectors** U of the square matrix A,
- `inv(A)`, returns the **inverse** of the square matrix A
  Warning: **never** use the inverse of a matrix to solve a linear system !
- `[U,S,V]=suv(A)`, returns the **singular value decomposition** of A.
- several other functions, factorizations etc.

Basics:

- Matlab scripts and functions have the file extension .m
- Matlab data files have the file extension .mat
- `help function` provides information on a Matlab function
- `type function` provides the Matlab-code if it is not a built-in function
- comments in a Matlab file start with %.
- save m-files not in the Matlab-directory

Difference between scripts and functions

- scripts are basically an extended command line statement (in particular all variables are shared between the common workspace and the script)
- functions work with their own workspace separate from that one at the command line.

Basic structure of a Matlab function:

```
function f = fact(n)
% Compute a factorial value.
% FACT(N) returns the factorial of N,
% usually denoted by N!

% Put simply, FACT(N) is PROD(1:N).
f = prod(1:n);
```

Elements:

- function definition line (actual name of the file counts as function name !)
- the first consecutive lines of comment following the function definition is shown when one uses

  `help fact`

A Matlab function II

Function input arguments

`function [x, y, z] = sphere(theta, phi, rho)` - specified input

`function sphere` - no input arguments

`function sphere(varargin)` - variable input

Function output arguments

`function [x, y, z] = sphere(theta, phi, rho)` - specified output

`function sphere(theta,phi,rho)` - no output

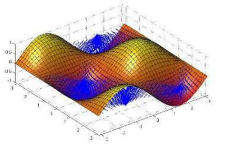
`function varargout = foo(n)` - variable output

General:

Function works also if less arguments are provided but not with more !

- `nargin, nargout`, contains number of provided input/output arguments
- function **calls by reference** but if input is changed copy (**call by value**) is created

Interpretation of names



Precedence Rules: When MATLAB is given a name to interpret, it determines its usage by using the following list (in that order)

- Variable
- Subfunction
- Private function
- Class constructor
- Overloaded method
- M-file in the current directory
- M-file on the path, or MATLAB built-in function

If two or more M-files on the path have the **same name**, MATLAB uses the M-file in the directory closest to the beginning of the path string.

Guide: Do not use names of existing variables or functions !

Interpretation of names II

File Precedence: If you refer to a file by its filename only (leaving out the file extension), and there is more than one file of this name in the directory, MATLAB selects the file to use according to the following precedence:

- built-in file,
- MEX-file (wrapper of C, C++ - Code),
- MDL-file (Simulink model file),
- P-Code file (pre-parsed m-file),
- M-file

Control flow (everything as in C++):

- `if, else, and elseif`
- `switch and case`
- `for`
- `while`
- `continue`
- `break`
- `try - catch`
- `return`

For the syntax use: help ...

Vectorization


```
function res =PNorm(x,p)
res=0;
for i=1:length(x)
  res=res+abs(x(i))^p;
end
```

```
function res =PNorm(x,p)
res = sum(abs(x).^p);
```

Effects of vectorization:

Code (avg. runtime in seconds) \ Input Size	1E3	1E5	1E7
Left Version (as m-file)	2.2E-4	4.8E-2	1.85
Left Version (command line)	3.0E-3	2.1E-1	12.5
Right Version (as m-file)	1.9E-4	3.1E-3	0.21
Right Version (command line)	1.3E-4	4.8E-3	0.35

$\Longrightarrow$ **JIT-compilation** in m-files

Memory management:

- `whos` shows the current content of the workspace
- `clear` - removes all variables from the workspace

  `clear functions` - removes compiled functions from memory

  `clear all` - removes functions and variables
- pre-allocate memory for (large) growing arrays

```
x=[1 1];
for i=1:100000
x = [x x(end)+x(end-1)];
end
```

```
a=1;
for k=1:100000
a(k) = k;
end
```

Problem: array is copied in each step !

Saving and loading variables to files:

```
>> X=[1 3]; Y=rand(10000,1);
>> save filename X Y; clear all;
>> load filename; whos
   Name   Size      Bytes  Class    Attributes
   X      1x2       16     double
   Y      10000x1   80000  double
```

Reading and saving binary or text files:

Use the C-like functions (almost same syntax):

`fopen, fread, fwrite, fclose, fseek, ftell, fscanf, fprintf`

Useful functions:

- **trigonometric functions:** `sin, cos, tan, asin, acos, atan, sinh,...`
- **exponential:** `exp, log, log2, log10, ...,`
- **rounding:** `fix, ceil, round, ...`
- **random number generator:** `rand, randn, randperm, ...`
- **data analysis:** `min, max, mean, median, std, var,...`
- **set functions:** `union, intersect, setdiff, unique, ...`
- **timer:** `tic, toc, cputime, ...`
- **other:** `abs, sign, sum, sort,...`

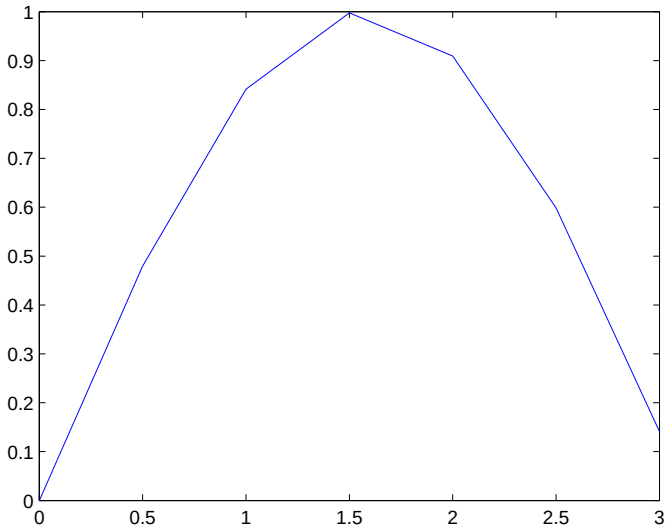
Useful to know:

- you can use **system commands** without having to leave Matlab
  just place an exclamation mark in front of the command
  `>> !dir` or `>> !grep ...`
- Matlab offers the usual **debugging** capabilities
  - standard breakpoints
  - conditional breakpoints
  - error breakpoints

Basic plotting: plotting arrays with `plot(X,Y)` or `plot(X,Y,'r.')`

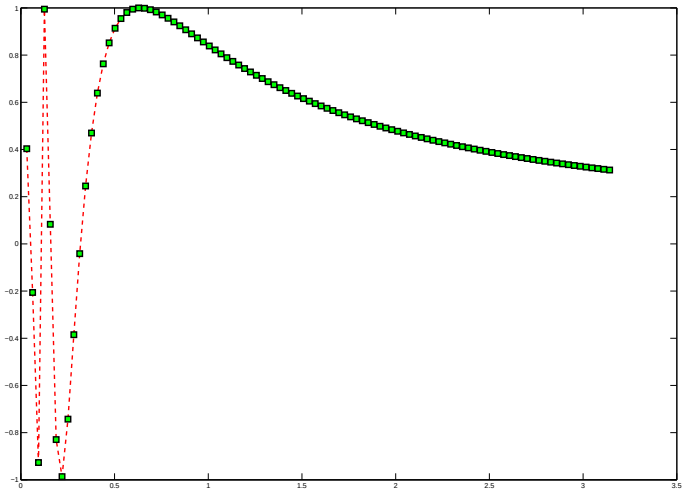
- standard: connects consecutive points in the vector X and Y with a line
- `'Linespec', -, -- , .- , :`
  `'Marker', +, o, ., *, x, >, <, ^, v, p, h`
  `'MarkerSize'`
  `'MarkerEdgeColor'`
  `'MarkerFaceColor'`

```
>> x=0:0.5:pi; y=sin(x);
>> plot(x,y);
```



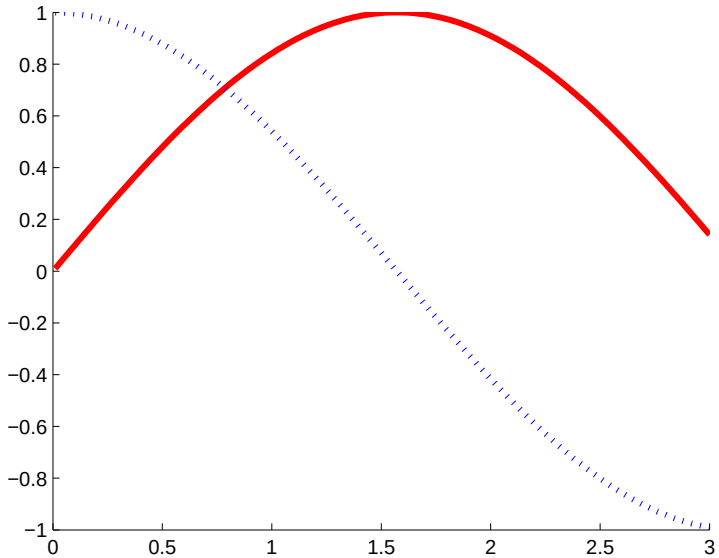
Specifying the linestyle:

```
>> x = pi/100:pi/100:pi;
>> y = sin(1./x);
>> plot(x,y,'--rs','LineWidth',2,.
'MarkerEdgeColor','k',...
'MarkerFaceColor','g',...
'MarkerSize',10)
```



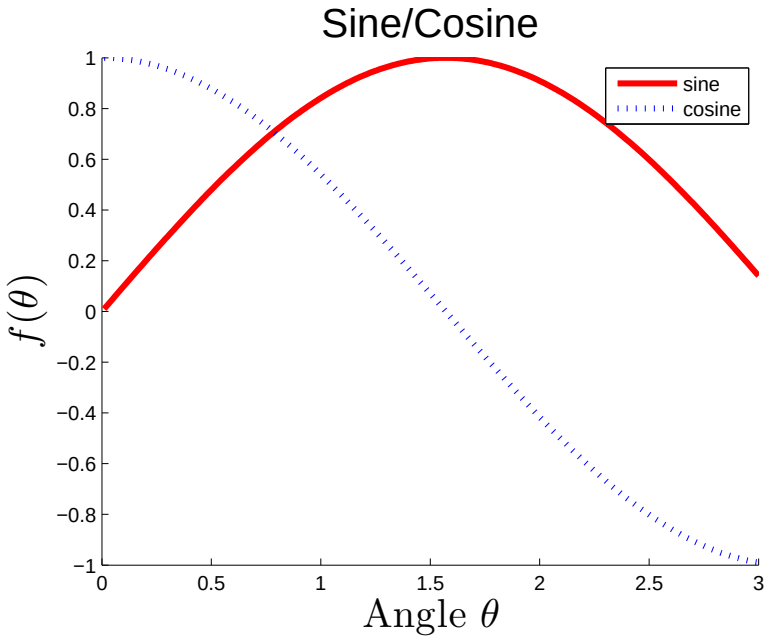
Several plots in one figure:

```
>> figure, hold on
>> x = 0.01:0.01:3;
>> y1 = sin(x); y2=cos(x);
>> plot(x,y1,'-r','LineWidth',3);
>> plot(x,y2,':b','LineWidth',3);
>> hold off
```



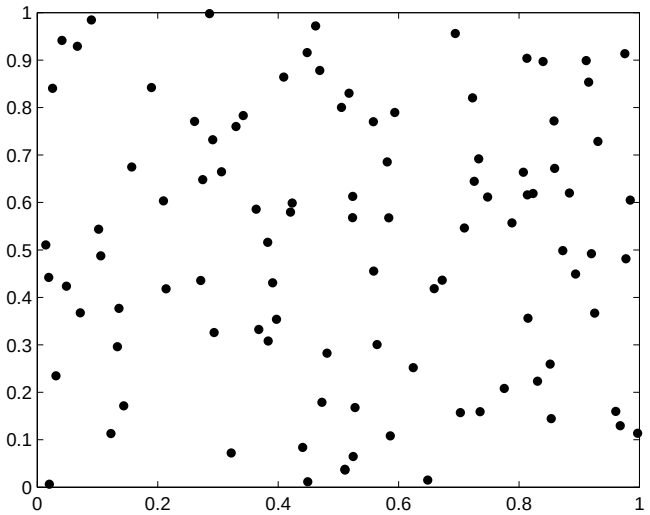
Adding title, axis labels, legend:

```
>> figure, hold on
>> x = 0.01:0.01:3;
>> y1 = sin(x); y2=cos(x);
>> plot(x,y1,'-r','LineWidth',3);
>> plot(x,y2,':b','LineWidth',3);
>> legend('sine','cosine');
>> xlabel('Angle $\theta$',...
'interpreter','latex');
>> ylabel('$f(\theta)$',...
'interpreter','latex');
>> title('Sine/Cosine');
>> hold off
```



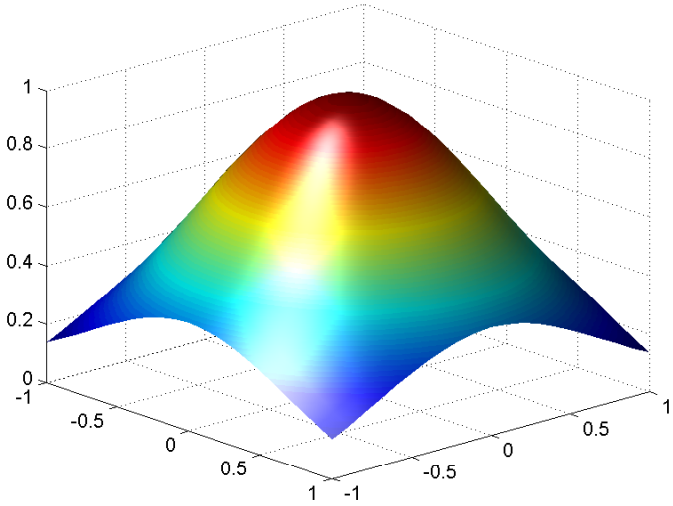
Plotting a set of points:

```
>> x = rand(100,1);
>> y = rand(100,1);
>> plot(x,y,'.k','MarkerSize',15)
```

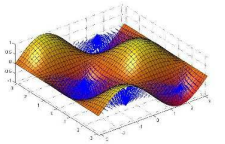


Plotting a 2D-function:

```
>> [X,Y]=meshgrid(-1:0.1:1,-1:0.1:1);
>> Z=exp(-(X.^2+Y.^2));
>> surf(X,Y,Z);
>> colormap jet; shading interp;
>> camlight left; lighting phong;
>> grid on
```



Plotting summary

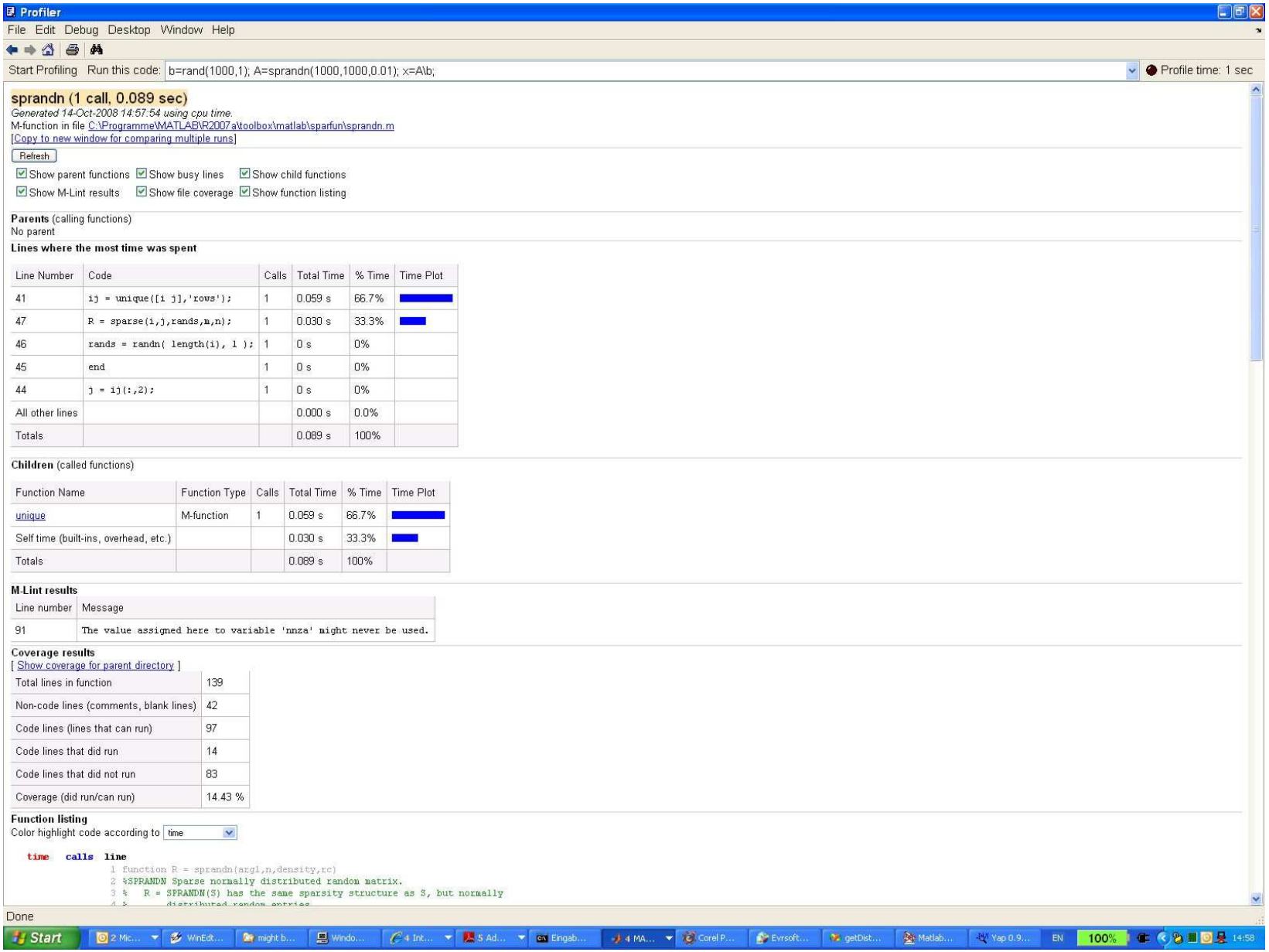


- `figure` - opens window
- `hold on, hold off` - between these commands everything is plotted in the same window
- `plot,plot3` - 2D and 3D plots
- `close fig_nr, close all` - closes figure windows
- `subplot` - create several different plots in one figure
- bar graphs: `pie, bar, hist, plotmatrix, ...`
- other functions: `scatter, scatter3, quiver, ...`
- save figures with `print -dbtyp filename` e.g. `dbjpg, dbmp, dbepsc`

Guidelines for Matlab-Programming:

- **use matrix-vector operations** instead of for-loops if possible (code is also often better readable),
- **pre-allocate memory** for large growing arrays,
- if matrices have only a few non-zero components **use `sparse` matrices**,
- **enable mulithreading** (**but:** due to problems with BLAS using multiple threads can be slower than a single thread),
- **changing the type of a variable is slow** - it is better to create a new one,
- **use the profiler** to identify bottlenecks and code them in C using Mex-files,
- **Do not use the built-in C(++) compiler for mex-files** - it is slow.

Matlab Profiler



Matlab Profiler II

Profiler

File Edit Debug Desktop Window Help

Start Profiling Run this code: b=rand(1000,1); A=sprandn(1000,1000,0.01); x=A\b; Profile time: 1 sec

Function listing

Color highlight code according to time

```
  time   calls  line
                 1 function R = sprandn(argl,n,density,rc)
                 2 %SPRANDN Sparse normally distributed random matrix.
                 3 %   R = SPRANDN(S) has the same sparsity structure as S, but normally
                 4 %       distributed random entries.
                 5 %
                 6 %   R = SPRANDN(m,n,density) is a random, m-by-n, sparse matrix with
                 7 %       approximately density*m*n normally distributed nonzero entries.
                 8 %       SPRANDN is designed to produce large matrices with small density
                 9 %       and will generate significantly fewer nonzeros than requested if
                10 %       m*n is small or density is large.
                11 %
                12 %   R = SPRANDN(m,n,density,rc) also has reciprocal condition number
                13 %       approximately equal to rc.  R is constructed from a sum of
                14 %       matrices of rank one.
                15 %
                16 %       If rc is a vector of length lr <= min(m,n), then R has
                17 %       rc as its first lr singular values, all others are zero.
                18 %       In this case, R is generated by random plane rotations
                19 %       applied to a diagonal matrix with the given singular values
                20 %       It has a great deal of topological and algebraic structure.
                21 %
                22 %   See also SPRAND, SPRANDSYM.
                23
                24 %   Rob Schreiber, RIACS, and Cleve Moler, MathWorks, 9/10/90.
                25 %   Revised 1/28/91, 2/12/91, RS; 8/12/91, CBM.
                26 %   Copyright 1984-2003 The MathWorks, Inc.
                27 %   $Revision: 5.12.4.1 $  $Date: 2003/05/01 20:43:14 $
                28
           1    29 if nargin == 1
                30    S = argl;
                31    [m,n] = size(S);
                32    [i,j] = find(S);
                33    R = sparse(i,j,randn(length(i),1),m,n);
           1    34 elseif nargin == 2
                35    error('MATLAB:sprandn:TwoInputs', 'Too many or not enough input arguments.')
           1    36 elseif nargin == 3
           1    37    m = argl;
           1    38    nnzwanted = round(m * n * min(density,1));
           1    39    i = fix( rand(nnzwanted, 1) * m ) + 1;
           1    40    j = fix( rand(nnzwanted, 1) * n ) + 1;
        0.06    1    41    ij = unique([i j],'rows');
           1    42    if ~isempty(ij)
           1    43       i = ij(:,1);
           1    44       j = ij(:,2);
           1    45    end
           1    46    rands = randn( length(i), 1 );
        0.03    1    47    R = sparse(i,j,rands,m,n);
                48 else    %    nargin == 4
                49    m = argl;
                50    nnzwanted = round(min(density,1)*m*n);
                51    minm = min(m,n);
                52    maxmn = max(m,n);
                53    lr = length(rc);
                54    if (lr > 1)  %  specified singular values
                55       if (lr > minm), lr = minm; end;
                56       %
                57       %   To start, put a random nonzero in every column / row if
                58       %   there is room enough
                59       %
                60       anz = zeros(maxmn, 1);
                61       anz(1:lr) = rc(1:lr);
                62       lr = minm;      %   if lr < minm then this adds some zero s.v.
                63       sqrt2o2 = sqrt(2)/2;
                64       while (lr < min(maxmn, nnzwanted))
```

A C mex-file


```
#include "mex.h"
#include <math.h>

void mexFunction(int nlhs, mxArray *plhs[],int nrhs, const mxArray *prhs[])
{
    int i, j, num, dim, k, index;
    double* RefPoint, * points, *outArray, norm, diff;

    // Test number of parameters.
    if (nrhs != 2 || nlhs != 1)
    { mexWarnMsgTxt("Usage: K = getDistVect(Point,Points)"); return;}

    // Parse parameters
    dim = (int)mxGetM(prhs[1]);   num = (int)mxGetN(prhs[1]);
    points = mxGetPr(prhs[1]);   RefPoint=mxGetPr(prhs[0]);
```

A C mex-file II


```
// Allocate memory for output
plhs[0] = mxCreateDoubleMatrix(1, num, mxREAL);
outArray = mxGetPr(plhs[0]);

// Compute Distance Vector
index=0;
for(i=0; i<num; i++) {
      norm = 0;
      for(k=0; k<dim; k++) {
            diff=RefPoint[k]-points[index];
            norm += diff*diff;
            ++index;
      }
      norm = sqrt(norm); outArray[i] = norm;
}
```

Pitfalls of Matlab-Programming:

- arrays start with index 1.
- element-wise product `A.*B` versus matrix-matrix product `A*B`.
- Never use `x=inv(A)*b` instead of `x=A\b`.
- when you work with complex numbers **never** use indices `i,j`.
- call your functions and variables different than existing Matlab functions (check with `which`) if you do not want to **explicitly override** them.
- some functions behave differently if they get a vector or matrix as input
  Examples: `min, max, sum, mean, median, var, std, ...`
  vector: applies to the whole vector - output a real number
  matrix: applies to each colum of the matrix - output a row vector
  $\Longrightarrow$ for the whole matrix `A` use e.g. `sum(A(:))`