Introduction to Matlab

Gerasimos Chourdakis

(Slides by Arash Bakhtiari and SCCS)

Scientific Computing in Computer Science
Technical University of Munich

October 18^{th}-19^{th}, 2018

Schedule of the next two days

- **Thursday, October** 18**:**
  09:00 – 11:30 Interactive lecture
  11:30 – 12:45 Lunch break
  12:45 – 15:00 Interactive lecture
  15:00 – 15:15 Break
  15:15 – 17:30 Supervised individual work

- **Friday, October** 19**:**
  09:00 – 12:00 Interactive lecture
  12:00 – 13:00 Lunch break
  13:00 – 15:00 Supervised individual work
  15:00 – 15:15 Break
  15:15 – 16:30 Supervised individual work

Why MATLAB? From organizational perspective

- Matlab is needed for at least two lectures:
  Scientific Computing Lab and Numerical Analysis
- For Scientific Computing you will need to build teams of two (or three) and submit programs in Matlab
- anybody intends to use other software?
  (Octave, R, Phyton , ...)

What is MATLAB® and why do we use it?

- Matlab is a technical computing environment for high-performance numerical computations and visualisation.
- The name Matlab stands for *matrix laboratory*.
- Matlab provides a high-level programming language and an interactive technical computing (and debuging) environment.

- Simulation pipeline: Modeling, Discretization, Computation, Visualization
- Fast prototyping tool for: Algorithm development, Data analysis and visualisation, Numerical computations

Industries using Matlab

- Aerospace
- Automotive
- Bio-chem, Pharmaceutical, Medical
- Communication
- Financial Industry
- Electronics
- Semiconductors
- ...

Technical Preparations

Launch matlab (Rechnerhalle):

- Open Terminal and type in `matlab`
- You should have it in your program list (Scientific $\longrightarrow$ Matlab)

- If this does not work, then:
  `$ /mount/applic/packages/matlab/bin/matlab`

Technical Preparations

- Open a Linux terminal
- Create directories and download files:

```
$ cd ~ #change to your home directory

$ mkdir matlab #creates directory for your M-files

$ mkdir slide #creates directory for the slides

$ firefox & #Download these slides and the example files from
the course web page
```

- You can also connect to the Rechnerhalle from home:

```
$ ssh <Rechnerhalle_login>@lxhalle.in.tum.de
```

- You can get a license for Matlab from TUM (see TUMonline)
- There is also GNU Octave, a compatible free/open-source alternative

Outline Part I

Getting help

- Matlab documentation
  https://de.mathworks.com/help/matlab/index.html
- Help for functions in Command Window:

```
>> help
>> help demo
>> help lookfor
>> help doc
```

- command completion:
  TAB key
- previous command in the history:
  UP key
- next command in the history:
  DOWN key

Outline

Matrices and Arrays

- Scalar

```
>> n = 8;
```

- Vector

```
>> x = [1 2 3 4 5 6 7 8]
```

- Matrix

```
>> A = [1 2 3;4 5 6;7 8 9]
>> A = [
 1 2 3
 4 5 6
 7 8 9 ];
>> a = [1 2]
>> b = [3 4]
>> B = [a;b]
>> B(1,1) = 5;
>> B
```

Matrices and Arrays

- Colon notation:

```
>> [1:5]
>> [1:3:15]
>> clear x;
>> x = [1:3:15];
>> z = rand(1,10);
>> z2 = z(1:2:10)
>> clear z z2;
```

- Matrix building functions:

```
>> E = eye(3)
>> M = rand(3)
>> Z = zeros(3,2)
```

Exercise 1

Build a 6×3 -Matrix out of A and E !
Build a 3×6 -Matrix out of A and E !

Matrices and Arrays

- load from file

```
>> !echo "1 2 3">C.dat; echo "4 5 6">>C.dat; echo "7 8 9" >>C.dat
>> save BFile B
>> load('C.dat')
>> D = load('C.dat')
>> load BFile
```

Matrix and Array Operations

```
>> at = a'
>> B = [1 1 1; 2 2 2; 3 3 3];
>> A + B
>> A * B
>> A. * B
>> A^2
>> A.^2
>> n = 8
>> n * A
>> n + A

>> F = [1 2; 3 4]
>> c = [2; 2]
```

Exercise 2

Calculate the solution vector x of the the system $Fx = c$. Use the left division operator '\'. Verify your result.

Matlab Scripts

- a sequence of commands can be stored in a script file (% comment line)
- files are called 'M-files' (extension of the files is '.m')
- two types of M-files: script files and function files
- store the M-files in the directory ∼/matlab.
- The script will be executed if you call it in the Matlab command line.

```
>> edit % start the matlab editor
```

Exercise 3

Write your solutions of the previous exercises in M-files and execute them!

Outline

Conditional Statements

- If statement

```
d = 3.7; e = rand(1);
if (e ~= 0.0)
    f = d/e;
end
e ~= 0.0 % 1 -> true, 0 -> false}
```

- If-else statement

```
if (e ~= 0.0)
    f = d/e;
else
    f = 0;
end
```

Conditional Statements

- else-if statement

```
if (e < 0.5)
    f = -1;
elseif (e > 0.5)
    f = 1;
else
    f = 0;
end
clear f;
```

FOR Loops

```
z = [];
for (k=1:10)
    z = [z, rand];
end
z
clear z;
```

```
z = [];
for (k=10:-1:1)
    z = [z, rand];
end
z
clear z;
```

WHILE Loops

```
z = 9.7;
n = 0.0;
while (n+1 <= z)
    n = n + 1;
end
n
clear n z;
```

Breaking Loops

- breaking loops:

```
n = 10;
z = rand(1,n);
l = -1;
for (k=1:n)
    if (z(k)<0.5)
        l = k;
        break;
    end
end
l
clear n z l;
```

Exercise 4

Write an M-file that computes the factorial ($n!$) of a given integer number n !

Outline

Functions

- Matlab functions:

```
pi_4 = atan(1.)
sin(pi_4)
exp(1.)
```

- Anonymous functions:

```
f1 = @(x)(x.*x + 3);
f2 = @(x,y)(2*y - x);
x = 0:0.1:2;
y = 2:-0.1:0;
fr1 = f1(x);
fr2 = f2(x,y);
plot(x,fr1);
figure;
plot(x,fr2);
clear all;
```

Outline

2D Plots

```
f = sin(0:0.1:2*pi);
plot(f)
clear f;
z = 0:0.1:2*pi;
f = sin(z);
plot(z,f)
clear f z;
```

```
plot(sin(0:0.1:2*pi));
hold on
plot(cos(0:0.1:2*pi));
hold off
z = 0:0.1:2*pi;
plot(z,sin(z),'r-',z,cos(z),'b--')
clear z;
```

2D Plots

```
z = -2*pi:0.1:2*pi;
plot(z,sin(z),'r-',z,cos(z),'b--')
title('Sine and Cosine');
xlabel('angle');
ylabel('value');
legend('sine','cosine');
grid on
axis([-pi pi -1.5 1.5]);
clear z;
```

Exercise 5

Work though matlab graphics demo 2-D Plots, Line Plotting, and Axes Properties!

Outline Part II

Outline

Vector Functions

```
>> clear all
>> x = [2 8 3 4 -5 -3 7 -1]
>> y = [3 8 2 1 4 11 8 1.2]
>> A = [6 2 3;1 8 -9]
>> max(x)
>> z = max(x,y)
>> max(A)
>> max(A,[],1)
>> max(A,[],2)
>> [v,ii] = max(x',[],1);
>> v
>> ii
>> x(ii)
>> max(A,4)
```

Vector Functions

```
>> sum(x)
>> sum(A)
>> sum(A,1)
>> clear z v ii;
```

Exercise 6

Write an M-file that multiplies the elements in the rows of an 3×3-matrix (each row with a diffrent scalar, the scalars are in a vector) and stores the results in a new 3×3-matrix !

Matrix Functions

```
>> B = [x; y]
>> size(B)
>> max(size(B))
>> C = zeros(length(B));
>> whos
>> clear B C;

>> B = [x(1:3); y(3:2:length(y)); A(2,:)]
>> eig(B)
>> [V,D] = eig(B);
>> V
>> D
>> det(B)
>> rank([x;y;x])
```

Exercise 7

Write an M-file that calculates the inverse of a 3×3-matrix! Verify your result! (Verify the result from the [V D] = eig(B);)

Sub-Matrices and Colon Notation

```
>> B
>> B(1:2,2:3)
>> B(:,1)
>> B(2,:)
>> A
>> A2 = A(1:2,1:2);
>> A(1:2,1:2) = eye(2)
>> A(1:2,1:2) = A2(1:2,1:2);
>> C = [1 3; 2 4]
>> z = C(:)
>> n = B(3)
>> B(4)
>> clear n A2 z
```

Outline

Main Functions

Main function:

the first function in the file (visible to functions in other files or in command line)

```
function a = square_area(e)
% SQUARE AREA. Area of a square.
% SQUARE AREA(E) is the area of a square.
% E is the lenght of an edge.

a = e*e;

% end of square area
```

```
>> area = square_area(2.0)
>> help square_area
```

Main Functions

```
function [vol, diag] = cube_info1(e,d)
% CUBE INFO. Volume and length of the diagonal of
% a cube.
%
%  [VOLL,DIAG] = CUBE INFO(E,D) produces the volume of
%  a cube VOL and %  the length of diagonal of the cube.
%  Where E is the length of a edge of a D-dimensional cube.

vol = e^d;
diag = e * sqrt(d);

end
```

```
>> [vol, diag] = cube_info(2.0,3)
>> vol = cube_info(2.0,3)
>> [vol, diag] = cube_info(1.5)
```

Main Functions

```
function [vol, diag] = cube_info2(e,d)
% CUBE INFO. Volume and length of the diagonal of
% a cube.
%
%  [VOL,DIAG] = CUBE INFO(E) produces the volume of
%  a cube VOL and the length of diagonal of the cube.
%   Where E is the length of a edge of the cube.
%
%  [VOLL,DIAG] = CUBE INFO(E,D) produces the volume of
%  a cube VOL and the length of diagonal of the cube.
%  Where E is the length of a edge of a D-dimensional cube.

if (nargin < 2),
    d = 3;
end

vol = e^d;
diag = e * sqrt(d);

end
```

Main Functions

```
>> [vol, diag] = cube_info2(1.5)
```

```
>> type cube_info2
>> type tic
>> type rank
```

Local Functions

Local function:

additional functions within the file in any order after the main function (only visible to other functions in the same file)

```
function [avg] = mystats1(x)
    n = length(x);
    avg = mymean(x,n);
end

function a = mymean(v,n)
% MYMEAN Example of a local function.

    a = sum(v)/n;
end
```

Nested Functions

Nested function:

A nested function is a function that is completely contained within a parent function. Any function in a program file can include a nested function.

```
function [avg] = mystats2(x)
    n = length(x);
    avg = mymean();

    function a = mymean()
        a = sum(x)/n;
    end
end
```

Outline

3D Plots

```
function f = sinsinc (t,x,y)
    r = sqrt(x.^2+y.^2) + eps;
    f = cos(t)*sin(r)./r;
end
```

```
>> close all
>> [X,Y] = meshgrid(-10:.2:10, -10:.2:10);
>> Z = sinsinc(0.0,X,Y);
>> surf(X,Y,Z);
```

Exercise 8

Print the surface plot into an eps-file!

Movies

```
n = 15;
% Initialize the videowriter object
v = VideoWriter('sinsinc.avi');
v.FrameRate = 5;
open(v);
% simulation time interval (time step)
inc = 2*pi/(n-1);
[X,Y] = meshgrid(-10:.2:10, -10:.2:10);
for k = 1:n,
    t = inc * k;
    Z = sinsinc(t,X,Y); % output at k-th time step
    clf % clear image
    surf(X,Y,Z);
    axis([-10 10 -10 10 -1 1])
    colormap(copper) % set color scheme
                     % put the frame into movie object
    frame = getframe;
    writeVideo(v,frame);
end
close(v);
```

Exercise 9

Print the different figures to 'Portable Network Graphic (PNG)' files instead of creating the movie! Modify the name of the file according to the loop index.

Outline

Measuring the Execution Time

```
time = zeros(100,1);
for i = 1:100,
    A = rand(i);
    b = rand(i,1);
    tic
    x = A\b;
    time(i) = toc;
end
plot(time)
```

Debuging

- For prototyping a user friendly debugging is necessary
- Editor debuging features
- Command line debugging features

```
>> dbstop if error
>> help dbstop
```

Write in a script file and run:

```
a = [1 2 3]; i = 4;
a(i)
```