Matlab Basics
Lecture 3

Heikki Apiola, Juha Kuortti

October 29, 2018

Contents:

More Graphics, 3D

Spacecurves, plot3

Spacecurves

Recall from Wed: ThreeDplots.m | plot3d.html

Start with a 2d-parametric curve, this is the former link:

```
t=linspace(0,10*pi,1000);
x=exp(-0.05*t).*cos(t);
y=exp(-0.05*t).*sin(t);
plot(x,y)
axis square
%% Let's take 3^{rd} dimension
z=t;
figure
plot3(x,y,z)
grid on
xlabel('x(t)');ylabel('y(t)');zlabel('z(t)')
```

Surfaces and contours, meshgrid

The function meshgrid is especially useful for 3d-graphics. Also in any computation where a function defined on a 2d-mesh is required.

Have a look at the m-file: meshscript.m

Here's a slight variation of the same example:

```
x=0:2;
y=3:6;
[X,Y]=meshgrid(x,y);
[X Y]    % X and Y side by side
                               y'     y'     y'
    x   0     1     2   |     3      3      3
    x   0     1     2   |     4      4      4
    x   0     1     2   |     5      5      5
    x   0     1     2   |     6      6      6
```

meshgrid (cont.)

Thus `X` consists of `length(y)` `(=4)` x-rows,

`Y` consists of `length(x)` `(=3)` y'-columns,

If you list `X` and `Y` in column order side by side, i.e.

`>> gridpoints=[X(:), Y(:)]` you will get a 3×4 rectangular grid of points, let's transpose the display to save space:

```
>> gridpoints'
   0    0    0    0    1    1    1    1    2    2    2    2
   3    4    5    6    3    4    5    6    3    4    5    6
```

```
plot(X(:),Y(:),'x')
axis([-.5 2.5 2.5 6.5])
grid on
title('xy-grid produced by meshgrid')
```

Some surface plots

With this data one gets:

```
    hold on
    mesh(x,y,Z)
    waterfall(x,y,Z)
```

Some variations with `mesh` using `subplot`:

```
  x=0:.1:pi; y=x;
  [X,Y]=meshgrid(x,y);
  Z=sin(Y.^2+X)-cos(Y-X.^2);
  subplot(2,2,1), mesh(Z)
  subplot(2,2,2), meshc(Z)
  subplot(2,2,3), mesh(x,y,Z),axis([0 pi 0 pi -5 5])
  subplot(2,2,4), mesh(Z),hidden off % No hidden ...
      line removal
```

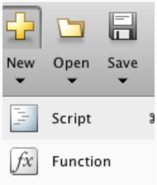
Functions

We have seen a number of MATLAB-functions in action.
Now we will see how to create our own functions, it is called programming.

User-defined functions

- Function handles, anonymous functions
  - One-liners, defined in the command window or in a script
    `>> f=@(x)x.^2` to be read: f is the function which "at x" returns the value x^2. (In math: $f = x \to x^2$)
    Several inputs allowed:
    `>> g=@(x,y,z)sqrt(x.^2+y.^2+z.^2).`
- Functions in m-files
  If more lines are needed, local variables, several output-values, control structures (`for`, `while`, `if - else`, etc.), then an m-file is needed.
- Inline-function is older, more restrictive version of function handle. We will not use them actively, the only reason to know about them, is old Matlab-codes. (`help inline`)

Examples of writing functions

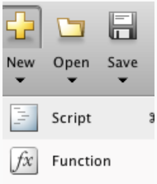


To start editing a function, open the editor on the top left “New”-button. Instead of script, this time click Function. Or on the command line:

```
>> edit myfunction
```

As our first example, let’s write a function that computes the mean of the components of the input vector.
Let’s first give some thought of the expression.

Examples of writing functions



To start editing a function, open the editor on the top left "New"-button. Instead of script, this time click Function. Or on the command line:

```
>> edit myfunction
```

As our first example, let's write a function that computes the mean of the components of the input vector.
Let's first give some thought of the expression.

```
x=1:10;
avg=sum(x)/length(x)
```

Example 1, mean of a vector

```
function y=mymean(x)
% Compute the mean (average) ox x-values.
% Call: y=mymean(x);
% Input: vector x
% Result: mean of x
% Exampe: r=mymean(1:10)
y=sum(x)/length(x);  % The only line of code
```

```
>> help mymean
```

Displays the 1^{st} contiguous comment block.

Built-in functions `mean`,`std` are available. Try `type mean` to see, how complicated data structure checks etc. make the code look complicated. **Beware of name conflicts**, remember: `which mean`.

Examples 2.: Make it work for matrices

Remember how `sum`, `min`,`max`, etc. work for matrices.

1. Write the function `mymean` as a function handle, call it `mymean1`.
2. Write a function `mymean2` that works for vectors as before and in addition for matrices columnwise (like sum, max, etc.)
   Note: The problem here is a row vector, you may need the `if` statement `(help if)`, to be discussed later.
   In this case function handle doesn't handle..., (unless you find some ingenious MATLAB trick)
3. How about the mean of all entries of a matrix?

mymean2.m

Example 2.: function stats

Standard deviation is given by:

$$\sigma = \sqrt{\frac{1}{N}\sum_{k=1}^{n}(x_k - \mu)^2}.$$

Write the code for the following function file:

```
function [avg,sd,range] = stats(x)
% Returns the average (mean), standard deviation
% and range of input vector x
 N=length(x);
 ...
 ...
```

Calling example function *stats*

Test your function using a script like the following:

```
%% Test script for function stats
x=linspace(0,pi);
y=sin(x);
[a,s,r]=stats(y)  % Function call
plot(x,y,'b')     % 'b' for blue
hold on
plot([0 pi],[a a],'k') % 'k' for blacK
shg               % show graphics
```

Solution: Listing of function stats

```
function [avg,sd,range] = stats(x)
% Returns the average (mean), standard deviation
% and range of input vector x
 N=length(x);
 avg=sum(x)/N;
 sd = sqrt(sum(x - avg).^2)/N);
 range=[min(x),max(x)];
```

Logical operators Flow Control.

1. Relational operators
2. Indexing, logical and numeric
3. `if,statements,end`
4. `if statements,elseif statements, else,end`

Relational Operators

Relational operators are used to compare variables. There are 6 comparison available:

- "equal to", using $==$
- "not equal to", using $\sim=$
- "less than", using $<$
- "less than or equal to", using $<=$
- "greater than", using $>$
- "greater than or equal to", using $>=$

The result of a comparison is either TRUE (1) or FALSE (0). Note that MATLAB makes difference between logical value and numerical one, but allows the usual arithmetic operations even with logical values.

Array comparisons

Suppose A and B are double arrays of the same size. Let **op** be any of the 6 relational operators $(==, \sim=, <, <=, >, >=)$.

Then the expression

$$\mathbf{A} \quad \mathbf{op} \quad \mathbf{B}$$

is a logical array of the same size. The relational operator is applied elementwise, comparing $A(i,j)$ to $B(i,j)$.

Example:

```
    >> A = rand(2,4);
    >> B = 0.5*ones(2,4);
    >> A<B
```

Excercise

Using `meshgrid` create a 2d space of matrices X and Y that covers the area $[-4,4] \times [-4,4]$.

Then find all the elements in X and Y for which $X(i,j)^2 + Y(i,j)^2 < 2$. Call the resulting logical array Z.

Finally `mesh(X,Y,Z)`.

Array comparisons, logical arrays

The result of a relational operation is a *logical array*.

- A logical array contains only 0's and 1's
- Internal representation is different from double arrays.

You can use a logical array in any numerical calculation similarly as a double array; the 0's and 1's behave normally. MATLAB automatically typecasts the logicals into doubles when arithmetic operations are applied.

```
A = [1 0 1 1]; B = logical(A);
whos
v=1:4;
v(A), v(B)
B2=2*B
whos
```

Logical indexing

- In addition to using numerical indexing, we can also extract entries using a bit pattern, i.e. a matrix of logical values.
- Only the entries corresponding to 1 are returned.
- Useful for selecting elements that satisfy some logical criteria formed by the above logical operators or "is"-functions such as isprime or isfinite.

```
A=magic(6)
B=A>30
```

Logical indexing (cont.)

We can then use this logical matrix to extract elements from A. In the following line, we repeat the call to A $>$ 30 but pass the result directly in, without first storing the interim result.

```
A=magic(6)
B1 = A(A > 30)                  % get all elements in ...
    A greater than 30
B = A(isprime(A) & (A > 30))    % get all prime ...
    elements in A greater than 30
```

Try also: `A.*(A>30)`

We could also achieve the same result using the **find** function, which returns the indices of all of the non-zero elements in a matrix. (find can be slightly slower.)

```
B2 = A(find(A > 30))
```

Using `find`

As noted, the command find returns the indices of the nonzero entries of a logical array. Another example:

```
    >> m = rand(6,1);
    >> m(find(m<0.5)) = 0;
```

Usually logical indexing will work just fine, so you can just do

```
    >> m = rand(6,1);
    >> m(m<0.5) = 0;
```

If (and only if) you need the numerical indices, use `find`.

More on `find`

Task: Find the min and max of

$$f(x) = \sin(3x) + 2\cos(5x)$$

```
x=linspace(0,4*pi,1000);
y=sin(3*x)+2*cos(5*x);
plot(x,y);grid on
[maxy,Ind]=max(y)  % max with 2 outputs does a ...
    ``find''
maxx=x(Ind)
[miny,Ind]=min(y)   % min with 2 outputs does a ...
    ``find''
minx=x(Ind)
```

Example continues

```
hold on
plot(maxx,maxy,'*r',minx,miny,'*b')
disp(['Peak value of y is ' num2str(maxy)])
title(['Peak value of y is ' num2str(maxy)])
```

This and a 2-d-example are found in:

minmaxexa.m, minmax2d.m

Excercise: Try to find the minimum point using the functions `fminsearch`, `fminbnd`, study with `help`.

All things are not equal

In finite precision arithmetic (MATLAB has about 17 digits of precision), it is not true that

$$(a+b)+c \text{ is equal to } a+(b+c)$$

In practice this means that when comparing doubles, equality is not a good test for similarity; instead we usually use `abs(x-y)<tol` to check for "equality". There are also other metrics — well learn them as we go.

Logical Operators

Logical operators are used to operate on logical variables. There are 3 binary operations

- "logical AND", using &
- "logical OR", using |
- "logical exclusive OR", using `xor`

There is also the unary operation

- "logical NOT", using $\sim$

For arrays, the operators are applied elementwise, and the results have logical values of TRUE (1) or FALSE (0)

In case of scalar values, there are also operators && and ||, that perform more efficiently.

Logical Operators

If A and B are scalars (double or logical), then

- $A \& B$ is TRUE (1) if A and B are both nonzero, otherwise it is FALSE (0)
- $A|B$ is TRUE (1) if either A or B are nonzero, otherwise it is FALSE (0)
- **xor**(A, B) is TRUE (1) if one argument is 0 and the other is nonzero, otherwise it is FALSE (0)
- $\sim A$ is TRUE if A is 0, and FALSE if A is nonzero.

For arrays, the operations are applied elementwise, so A and B must be the same size, or one must be a scalar.

If you wish to check if two arrays are same, use `all`, if you wish see whether they have any similarities, use `any`.

Control: `if`, `end`

To conditionally control the execution of statements, you can use

```
    if expression
         statements
    end
```

If the real part of all of the entries of `expression` are nonzero, then the statements between the if and end will be executed. Otherwise they will not be. If the `expression` is an array, then the check is implicitly `all(expression)`.

Execution continues with any statements after the end.

Control: `if`, `else`, `end`

```
if exp1
   statements1
else
   statements2
end
```

One of the sets of statements will be executed

- If `exp1` is TRUE, then `statements1` are executed
- If `exp1` is FALSE, then `statements2` are executed

Control:`if`, `elseif`, `end`

If you need to check for multiple cases, use `elseif`:

```
if exp_1
   statements1
elseif exp_2
   statements2
elseif exp_3
   statements3
end
```

MATLAB also contains a `switch,case,...,otherwise,end` structure, which can be seen as an alternative to `if,elseif,else,end`. See `>>help switch`, also `>> type why` as an amusing example.

Iterations, More control.

1. `for <cond>,statements,end`
2. `while <cond> ,statements, end`
3. `case, switch,...,end`

Control: `for, end`

Execute collection of statements a fixed number of times.

```
for x=expression
   statements(x)
end
```

The expression is in most cases a vector and the loop variable x runs through all the components of “expression”.

The most common “expression” would be the vector `1:N` or `1:h:N` with suitable h and N.

Control: `for`, `end`

Examples:

```
for k=1:n
   v(k)=k^2;
   end
   v
```

Produces the same as the vectorized form: `p=1:n; v=p.^2`

Example

Write a function to compute the amount owed for a loan amount L, given interest rate R, loan duration N (months) and fixed monthly payments of amount P.

To save time, I give the code on the next slide. Study the code carefully and write (copy/paste) it into an m-file loancalc.m and test it.

Write a script file (runloancalc.m for instance) including call to loancalc. Choose suitable values for the parameters (for instance planning your apartment loan or something). Suppose for instance that the time of loan is 15 years. What is the minimum monthly payment for making it with certain interest rate and loan amount. You can ask (and answer) other relevant questions as well.

Code for loancalc

```
function P = loancalc(L,R,N,MP)
% P = loancalc(L,R,N,MP) computes the
% history of amount owed on a loan of amount
% L, interest rate R, duration N, and fixed
% monthly payment MP.
P = zeros(N+1,1);
P(1) = L; % amount owed at Month=0
% interest rate R is annual, but applied
% monthly, yielding a 1+R/12 factor.
G = 1+R/12;
for i=2:N+1
   P(i) = P(i-1)*G - MP;
end
```

Control: `while`, `end`

If you need to execute commands for an undetermined number of times, use `while` loop

```
while expression
   statements
end
```

`while` evaluates `expression`, and if it is TRUE, then executes the `statements`, and repeats, otherwise it jumps to `end`.

Notice, that expression need not become FALSE ever, leading to an infinite loop.

Example: $\sqrt{a}$ with Newton, for-loop

Applying *Newton's method* to the equation $x^2 - a = 0$, leads to an iteration sequence:

$$x_0 = a, x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n}),$$

that converges to $\sqrt{a}$.

Write a MATLAB-script, that let's you examine this when $a = 5$.

As an initial guess you may use a itself, as suggested.

Give as a result a 3-column matrix T (just numbers, no text):

n	$x(n)$	error
0	a	$\sqrt{a} - 1$
1	$x(1)$	$\sqrt{a} - x(1)$
$\vdots$	$\vdots$	$\vdots$
N	$x(N)$	$\sqrt{a} - x(N)$

Use for-loop. Here a better solution would be while-loop.

Example: $\sqrt{a}$ with Newton, while-loop

Yes, do it! This time write a function `newtsqrt2` that just returns the result, pure scalar code to make it simple.

Sols:

newtsqrt.m , newtsqrt2.m

Polynomials

Polynomials, coefficient vector, value

MATLAB represents a polynomial as the vector of coefficients starting at the highest power.
Let

$$p(x) = x^4 - 15x^2 + 45x - 36.$$

To compute $p(x)$ at a vector of points x you should'n and needn't proceed as: `p=x.^4 -15*x.^2 + 45*x-36.5`. Instead there is a more efficient and easy-to-use function **polyval**.

```
c=[1 0 -15 45 -36]; % Note: 0 for a missing power
x=linspace(-6,6);   % 100 points on the interval [-2,2].
p=polyval(c,x);     % Values of ``c-polynomial''
plot(x,p); grid on;shg
```

Polynomials, roots

The roots of a polynomial equation can be obtained (numerically) by the function **roots**.

```
>> pzeros=roots(c)
pzeros =
  -5.0355 + 0.0000   % Real root
   1.8680 + 1.4184i  % complex conjugate roots
   1.8680 - 1.4184i  % (always with real polynomial)
   1.2996 + 0.0000i  % Real root
```

Note: One is tempted to use variable names such as `roots` or `zeros`. Both are names of Matlab's built-in functions (we just used `roots`). Check: `which roots`, `which zeros`.
Using such names may lead to "nonsense" error messages.

Polynomials, roots (continued)

To check how close to zero the values of the polynomial are at the computed zeros, we need the function `polyval`.

```
>> polyval(c,pzeros) % Values of p at pzeros
ans =
   1.0e-11 *         % Small enough
   0.1300 + 0.0000i
  -0.0043 - 0.0046i
  -0.0043 + 0.0046i
   0.0000 + 0.0000i
```

Find the real roots in the figure, zoom in (menu: “tools”).

Exercise

- Plot the values of the polynomial $p(x) = x^4 - 3x^3 + 8x + 2$ on the interval $x = [-3, 3]$.
- Find the roots of $p(x)$.
- Find the roots of $z^5 - 1$, and plot them on the complex plane. Plot the unit circle in the same figure and use `axis equal`.
  Note: plot works nicely in the complex plane, it just needs the vector of complex numbers. The shortest way to plot the unit circle. is
  ```
  t=linspace(-pi,pi); plot(exp(i*t));axis equal
  ```
  Here you have to do one plot at a time and use `hold on`.
- Construct a polynomial of degree 6, with roots $r_k = k$. (i.e., first root is 1, second 2 and so on). How high can you increase the degree, before the root-finding becomes inaccurate?
  Hint: `help poly`

Interpolation, curve fitting

How to model given data with a polynomial

MATLAB-functions: `polyfit,polyval`

https://se.mathworks.com/help/matlab/data_analysis/programmatic-fitting.html

- polyfit(x,y,n) finds the vector of *coefficients* of a polynomial p(x) of degree n that fits the y data (least-squares fit). Especially, if $n =$ (nr. of datapoints) -1, the polynomial passes through all the datapoints, this is called **polynomial interpolation**.

Interpolation, curve fitting, example

Given datapoints:

$t = 0, 0.3, 0.8, 1.1, 1.6, 2.3$

$y = 0.6, 0.67, 1.01, 1.35, 1.47, 1.25,$

fit polynomials of different degrees starting at the interpolation polynomial, which is of degree 5

```
clear;close all
t = [0 0.3 0.8 1.1 1.6 2.3];
y = [0.6 0.67 1.01 1.35 1.47 1.25];
plot(t,y,'o')
title('Plot of ydata Versus tdata')
n=length(t);
c=polyfit(t,y,n-1) % Coefficient vector of fitted ...
    polynomial
```

Interpolation example continued

Compute values of the polynomial at 100 or so points on the interval $[t_{min}, t_{max}]$.

```
clf                           % Clear figure
a=min(t);b=max(t);
tev=linspace(a-.1,b+.1); % Points of evaluation
yev=polyval(c,tev);      % Values of polynomial
plot(tev,yev,t,y,'o');grid on;shg
legend('fitted polynomial','datapoins')
title('Polynomial interpolation')
```

Spline interpolation example

Spline is a piecewise polynomial, most often consisting of cubic polynomial pieces. It avoids the extra bends and peaks often present with high degree polynomials. The basic use of MATLAB's `spline`-function is even (one step) easier than the `polyfit,polyval` combination above.

Continuing the previous example:

```
ysp=spline(t,y,tev); % Values of spline at tev (above)
hold on
plot(tev,ysp,'r')
```