MATLAB TUTORIAL
MATH/CS 375

1 Environment

1.1 Command Window (or Interpreter)

- quick access
- good for developing ideas
- good for accessing data in the workspace
- poor performance and editing/debugging capabilities

1.2 Scripts and Functions

```
1   function y = my_funky_fcn(x)
2
3   % Output parameter:     y (vector of outputs)
4   % Input parameters:     x (vector of values)
5
6 -   y = (sin(x)).^2;
```

- good test bed: all info in the workspace
- fast, linear
- function can be optimized and cached
- don't fill up the workspace with memory that you have to manage,
- reusable code
- will use these for HW's
- More on functions later…

2 Scalars, Vectors and Matrices

2.1 Scalars

```
a=2
```

```
a=2; % The semicolon ; suppresses output
```

2.2 Vectors

```
x= [1;2;3;4;5] %Column Vector
```

```
x = [1 2 3 4 5] %Row Vector.
```

Rows are separated by semicolons. The entries in a row are separated by spaces or commas

```
x(2)   %access the second component of the vector x
```

```
x(3) = 7; %set the third component of x as 7
x
```

2.3 Matrices

```
A = [1 2 3; 4 5 6; 7 8 9]
```

```
A(2,3) = -4;  %Modify the (2,3) element of A
A
```

3 Built-in Variables, Functions and Commands

```
pi  %3.14159
```

```
format long %Change the number of digits displayed
```

```
pi
```

```
format short
```

```
%Some useful built-in functions
```

```
size(A)    %size of a variable
```

```
length(x)  %length of a vector
```

```
max(x)     %returns the largest entry of a vector
```

```
min(x)     %returns the smallest entry of a vector
```

```
sin(a)      %returns sin of our variable a

sin(x)      %returns a vector of sin of entries of x
```

4 Creating matrices and vectors

```
x1 = 1:5                %Creates a Row Vector whole components increase by 1

x2 = 1:0.5:5            %Creates a Row Vector whole components increase by 0.5

x3 = 5:-0.5:1           %Creates a Row Vector whole components decrease by 0.5

x4 = linspace(1,5,9)    %Creates a vector from 1 to 5 with 9 equally spaced
                         entries

A = zeros(3,2)          %Creates a 3x2 matrix of all zeros

A = ones(2,4)          %Creates a 2x4 matrix of all ones

A = eye(3)              %Creates a 3x3 identity matrix

A = diag([1,2,3,4])     %Creates a 4x4 diagonal matrix with the vector [1,2,3,4] on
                         the diagonal
```

5 Operations on Vectors and Matrices

5.1 Addition, Multiplication, Etc

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

B = A + A               %Matrix addition

C = A*A                 %Matrix multiplication

D = A.*A                %Elementwise multiplication (NOT matrix-matrix
                         multiplication)

E = A^3                 % E = A*A*A

F = A.^3                %F = [1^3 2^3; 3^3 4^3]

G = inv(A)              %inverse of A

E = det(A)              %determinant of A

F = A'                   %Transpose of A
```

5.2 Accessing subvectors

```
x=0:0.1:1;
n=length(x)
x2=x(5:10)              % What is x2?
x2=x([1,3,4,11])        % What is x2?
x2=x(2:4:11)            % What is x2?
```

5.3 Accessing submatrices

```
a=[100 90 85; 117 110 108; 84 84 84; 96 90 88];
[m,n]=size(x)
a2=a(2,3)               % What is a2?
a2=a(:,2)               % What is a2?
a2=a(2,:)               % What is a2?
a2=a(2:3,:)             % What is a2?
a2=a(2:3,[1,3])         % What is a2?
```

6 Graphics

6.1 Plot command

```
x=0:.1:1;
y =sin(2*pi*x);
plot(x,y); % the two vectors have to have same dimensions
```

6.1.1 Labeling axis

```
xlabel('x');
ylabel('y');
```

6.1.2 Line type options:

```
plot(x,y,'-');
plot(x,y,':');
plot(x,y,'--');
plot(x,y,'-.');
```

6.1.3 Color options: y,m,c,r,g,b,w,k

```
plot(x,y,'g'); % green line (line is default)
```

6.1.4 Markeroptions: .,o,x,+,*,s,d,v,^,<,>,p,h

```
plot(x,y,'x'); % blue star (blue is default)
```

6.1.5 Using several options together

```
plot(x,y,'*-r'); % red line with star markers
```

6.2 Plotting several curves

6.2.1 Use the hold command

```
x=0:0.05:1;
y1=sin(2*pi*x);
y2=cos(2*pi*x);
plot(x,y1,'-b')
hold on
plot(x,y2,'--r')
```

6.2.2 Labeling

```
xlabel('x');
ylabel('y');
title('The Force Awakens is awesome')
legend('sin(x)', 'cos(x)')
```

6.2.3 Saving Figures as PDF or Other Formats

```
saveas(gcf,'episode_8.png')
saveas(gcf,'episode_9.pdf')
```

gcf is a builtin command to access the "current" figure

6.3 Other builtin plotting functions

```
close all %closes all the figures
```

Also see

```
loglog, semilogx, semilogy
%%%%%%
```

7 Control Structures

7.1 if statement

```
a = rand(1);  %Random value between 0 and 1
if a > 2/3
    disp('a>2/3')  %
elseif a < 1/3
    disp('a<1/3')
else
    disp('1/3 <= a <= 2/3')
end
```

built-in function disp displays its argument

```
disp(['a = ' num2str(a)])
```

7.2 logical operators:

<, >, <=, >=, ==, ~=

Note: 0 is false, while any non-zero value is considered true

7.3 for loop

7.3.1 Example 1:

```
for i = [2,4,6,8]
    disp(i^2);  %i takes values 2, 4, 6, 8
end
```

7.3.2 Example 2:

```
a=0;b=1;n=10; delx=(b-a)/n; x = zeros(n+1,1); % Set variables
for i=1:n+1
    x(i)=a+delx*(i-1); % index of x has to be an integer > 0 >> end
end
```

7.3.3 Example 3: for loop to display a table of values

```
a=0;b=1;n=10; delx=(b-a)/n; x = zeros(n+1,1); % Set variables
for i=1:n+1
    disp(sprintf('%d \t  %6.4f', i, x(i)));
end
```

Look at the syntax for using sprintf. It's a useful command!

7.3.4 Example 4: for loop to compute the sum of all elements of a vector x

```
x = 1 : 0.1: 10;
n = length(x);
s = 0;
for i = 1 : n
    s = s + x(i);
end
disp(s)

%note that the builtin function sum(x) does the same job
```

7.3.5 Example 5: Nested for loops for matrix operations

```
n = 10;
for i=1:n
    for j=1:n
        a(i,j) = 1/(i+j-1);
    end
end
```

7.4 while loop

```
a = 1;
while a <= 10   %while loop repeats as long as the given expression in front of
while is true
    disp(a)
    a = a+1;
end
```

8 Scripts

You can type a string of commands into a file whose name ends in .m, for example 'flnm.m'. If you then type

>> flnm

in your command window, it executes all the commands in the file flnm.m. Make sure you document your script files! Add a few lines of comments that state what the script does.

9 Functions

MATLAB Functions are similar to functions in Fortran or C. They enable us to write the code more efficiently, and in a more readable manner. The code for a MATLAB function must be placed in a separate .m file having the same name as the function. The general structure for the function is

function ⟨Output parameters⟩ = ⟨Name of Function⟩ (⟨Input Parameters⟩)
% % % Comments that completely specify the function
⟨function body⟩

A function is called by typing
>> variable = ⟨Name of Function⟩

When writing a function, somewhere in the function body the desired value must be assigned to the output variable!

9.1 Examples

9.1.1 Example 1: Function with two inputs and one output

Question: Write a function with input parameters x and n that evaluates the nth order Taylor approximation of e^x. Write a script that calls the function for various values of n and plots the error in the approximation.

Solution: The following code is written in a file called ApproxExp.m:

```
function y=ApproxExp(x,n);
% Output parameter: y (nth order Taylor approximation of e^x)
%Input parameters:  x (scalar)
%                   n (integer)
sumo = 1;
for k=1:n
    sumo = sumo + x^k/factorial(k);
end
y = sumo;
```

A script that references the above function and plots approximation error is:

```
x=4;
max_terms = 10; z = zeros(max_terms,1);
for n=1:max_terms
    z(n) =ApproxExp(x,n);
end
exact=exp(4); %use builtin function exp
plot(abs(exact-z)); xlabel('terms'); ylabel('error');
close all;
```

9.1.2 Example 2: Function with multiple outputs and builtin functions as arguments

```
function [d,err]=MyDeriv(f,fprime,a,h)
% Output parameter:     d (approximate derivative using finite difference (f(h+h)-
f(a))/h)
%                       err (approximation error)
% Input parameters:     f (function)
%                       fprime (derivative function)
%                       a (point at which derivative approx)
%                       h (stepsize)

d = (f(a+h)-f(a))/h;
err = abs(d-fprime(a));
```

A script that references the above function and plots the approximation error is given below.

```
a=1;
h=logspace(-1,-16,16);
n=length(h);
for i=1:n
    [d(i),err(i)]=MyDeriv(@sin,@cos,a,h(i));
end
loglog(h,err);
```

9.1.3 Example 3: Anonymous Functions

We define two anonymous functions in the code below. The anonymous functions are defined using the syntax @ as shown below:

```
h = 0.1;
g=@(x)(x.^2);
gprime=@(x)(2*x);
[d,err]=MyDeriv(g,gprime,a,h)
```

9.1.4 Example 4: Passing functions in files as arguments

We have seen how to pass in as an argument a function already defined in MATLAB (such as sin, cos), or an anonymous function (note difference in calling script). Alternatively, we can pass in a user specified function that is not inline. Consider functions f1 in file f1.m and df1 in file df1.m:

```
function y = f1(x)
y = (x.^2)

function y = df1(x)
y = 2*x;
```

Now you can call

```
    [d,err]=MyDeriv(@f1,@df1,1,.1)
```

9.1.5 Example 5: Function with one vector output

```
function y = my_funky_fcn(x)

% Output parameter:     y (vector of outputs)
% Input parameters:     x (vector of values)

y = (sin(x)).^2;
```

A script that references the above function is

```
x = 0:0.1:pi;
y = my_funky_fcn(x);
plot(x,y)
y2 = sin(x);
hold on;
plot(x,y2);
legend('sin(x)*sin(x)', 'sin(x)');
```

10 Other useful Matlab commands

save, load, clear all (Google or Bing them)