

A faint, dotted world map is visible in the background of the slide, centered behind the text.

Lecture 4: Complex Numbers Functions, and Data Input

What is a Function?

- A MATLAB Function (e.g. $y = \text{func}(x1, x2)$) is like a script file, but with inputs and outputs provided automatically in the command window.
- In MATLAB, functions can take zero, one, two or more inputs, and can provide zero, one, two or more outputs.
- There are built-in functions (written by the MATLAB team) and functions that you can define (written by you and stored in .m file).
- Functions can be called from command line, from within a script, or from another function.



Table 3.1–1 Some common mathematical functions

Exponential

exp (x) Exponential; e^x .

sqrt (x) Square root; \sqrt{x} .

Logarithmic

log (x) Natural logarithm; $\ln x$.

log10 (x) Common (base-10) logarithm; $\log x = \log_{10} x$.

Complex

abs (x) Absolute value; x .

angle (x) Angle of a complex number x .

conj (x) Complex conjugate.

imag (x) Imaginary part of a complex number x .

real (x) Real part of a complex number x .

Numeric

ceil (x) Round to the nearest integer toward ∞ .

fix (x) Round to the nearest integer toward zero.

floor (x) Round to the nearest integer toward $-\infty$.

round (x) Round toward the nearest integer.

sign (x) Signum function:

+1 if $x > 0$; 0 if $x = 0$; -1 if $x < 0$.

Functions are Helpful

- Enable “divide and conquer” strategy
 - Programming task broken into smaller tasks
- Code reuse
 - Same function useful for many problems
- Easier to debug
 - Check right outputs returned for all possible inputs
- Hide implementation
 - Only interaction via inputs/outputs, how it is done (implementation) hidden inside the function.



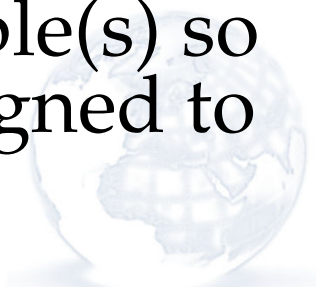
Finding Useful Functions

- You can use the `lookfor` command to find MATLAB functions that are relevant to your application.
- Example: `>> lookfor imaginary`
- Gets a list of functions that deal with imaginary numbers.
- `i` – Imaginary unit.
- `j` – Imaginary unit.
- `complex` – Construct complex result from real and imaginary parts.
- `imag` – Complex imaginary part.



Calling Functions

- Function names are case sensitive (meshgrid, meshGrid and MESHGRID are interpreted as different functions).
- Inputs (called function *arguments* or function *parameters*) can be either numbers or variables.
- Inputs are passed into the function inside of parentheses () separated by commas.
- We usually assign the output to variable(s) so we can use it later. Otherwise it is assigned to the built-in variable `ans`.



Rules

- To evaluate $\sin 2$ in MATLAB, we type `sin(2)`, not `sin[2]`
- For example `sin[x(2)]` gives an error even if `x` is defined as an array.
- Inputs to functions in MATLAB can be sometimes arrays.

```
>> x = -3 + 4i;
>> mag_x = abs(x)
mag_x =
     5

>> mag_y = abs(6 - 8i)
mag_y =
    10

>> angle_x = angle(x)
angle_x =
     2.2143

>> angle(x)
ans =
     2.2143

>> x = [5,7,15]
x =
     5     7    15

>> y = sqrt(x)
y =
     2.2361     2.6458     3.8730
```

Function Composition

- Composition: Using a function as an argument of another function
- Allowed in MATLAB.
- Just check the number and placement of parentheses when typing such expressions.
- `sin(sqrt(x)+1)`
- `log(x.^2 + sin(5))`



Which expression is correct?

- You want to find $\sin^2(x)$. What do you write?
- $(\sin(x))^2$
- $\sin^2(x)$
- \sin^2x
- $\sin(x^2)$
- $\sin(x)^2$
- *Solution:* Only first and last expressions are correct.



Trigonometric Functions

Trigonometric*

$\cos(x)$	Cosine; $\cos x$.
$\cot(x)$	Cotangent; $\cot x$.
$\csc(x)$	Cosecant; $\csc x$.
$\sec(x)$	Secant; $\sec x$.
$\sin(x)$	Sine; $\sin x$.
$\tan(x)$	Tangent; $\tan x$.

Inverse trigonometric†

$\arccos(x)$	Inverse cosine; $\arccos x = \cos^{-1} x$.
$\operatorname{arccot}(x)$	Inverse cotangent; $\operatorname{arccot} x = \cot^{-1} x$.
$\operatorname{arccsc}(x)$	Inverse cosecant; $\operatorname{arccsc} x = \csc^{-1} x$.
$\operatorname{arcsec}(x)$	Inverse secant; $\operatorname{arcsec} x = \sec^{-1} x$.
$\arcsin(x)$	Inverse sine; $\arcsin x = \sin^{-1} x$.
$\operatorname{arctan}(x)$	Inverse tangent; $\operatorname{arctan} x = \tan^{-1} x$.
$\operatorname{atan2}(y, x)$	Four-quadrant inverse tangent.

*These functions accept x in radians.

†These functions return a value in radians.



Hyperbolic functions

Hyperbolic

$\cosh(x)$	Hyperbolic cosine; $\cosh x = (e^x + e^{-x})/2$.
$\coth(x)$	Hyperbolic cotangent; $\cosh x / \sinh x$.
$\operatorname{csch}(x)$	Hyperbolic cosecant; $1/\sinh x$.
$\operatorname{sech}(x)$	Hyperbolic secant; $1/\cosh x$.
$\sinh(x)$	Hyperbolic sine; $\sinh x = (e^x - e^{-x})/2$.
$\tanh(x)$	Hyperbolic tangent; $\sinh x / \cosh x$.

Inverse hyperbolic

$\operatorname{acosh}(x)$	Inverse hyperbolic cosine
$\operatorname{acoth}(x)$	Inverse hyperbolic cotangent
$\operatorname{acsch}(x)$	Inverse hyperbolic cosecant
$\operatorname{asech}(x)$	Inverse hyperbolic secant
$\operatorname{asinh}(x)$	Inverse hyperbolic sine
$\operatorname{atanh}(x)$	Inverse hyperbolic tangent

User-Defined Functions

- Functions must be saved to a file with .m extension.
- Filename (without the .m) must match EXACTLY the function name.
- First line in the file must begin with a function definition line that illustrates inputs and outputs.

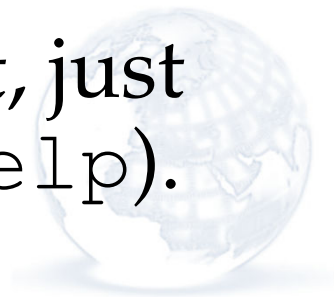
```
function [output variables] = name(input variables)
```

- This line distinguishes a function M-file from a script M-file.
- Output variables are enclosed in square brackets.
- Input variables must be enclosed with parentheses.



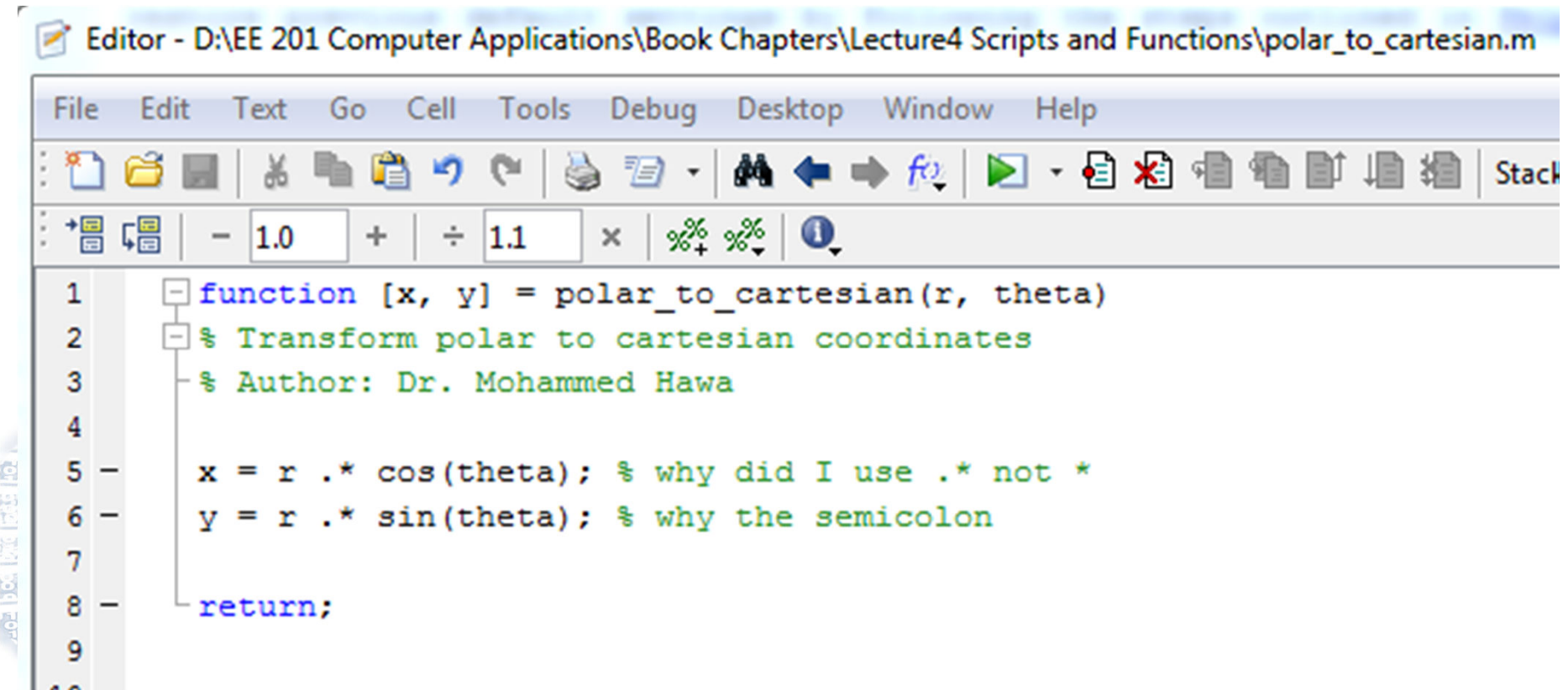
Functions Names

- Function names may only use alphanumeric characters and the underscore.
- Functions names should NOT:
 - include spaces
 - start with a number
 - use the same name as an existing command
- Consider adding a header comment, just under the function definition (for help).



Exercise: Your Own `pol2cart`

- Make sure you set your Current Folder to Desktop (or where you saved the .m file).



The screenshot shows a MATLAB editor window titled "Editor - D:\EE 201 Computer Applications\Book Chapters\Lecture4 Scripts and Functions\polar_to_cartesian.m". The window has a menu bar with "File", "Edit", "Text", "Go", "Cell", "Tools", "Debug", "Desktop", "Window", and "Help". Below the menu bar is a toolbar with various icons for file operations, editing, and execution. A numeric keypad is visible below the toolbar, showing values like 1.0 and 1.1. The main editing area contains the following MATLAB code:

```
1 function [x, y] = polar_to_cartesian(r, theta)
2 % Transform polar to cartesian coordinates
3 % Author: Dr. Mohammed Hawa
4
5 x = r .* cos(theta); % why did I use .* not *
6 y = r .* sin(theta); % why the semicolon
7
8 return;
```

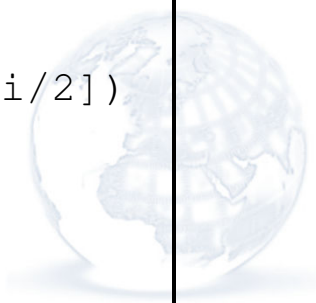
Test your newly defined function

```
>> [a, b] = polar_to_cartesian(3, pi)
a =
    -3
b =
    3.6739e-016

>> polar_to_cartesian(3, pi)
ans =
    -3

>> [a, b] = polar_to_cartesian(3, pi/4)
a =
    2.1213
b =
    2.1213

>> [a, b] = polar_to_cartesian([3 3 3], [pi pi/4 pi/2])
a =
   -3.0000    2.1213    0.0000
b =
    0.0000    2.1213    3.0000
```



MATLAB has pol2cart

```
>> help pol2cart
```

POL2CART **Transform polar to Cartesian coordinates.**

$[X,Y] = \text{POL2CART}(TH,R)$ transforms corresponding elements of data stored in polar coordinates (angle TH , radius R) to Cartesian coordinates X,Y . The arrays TH and R must be the same size (or either can be scalar). TH must be in radians.

$[X,Y,Z] = \text{POL2CART}(TH,R,Z)$ transforms corresponding elements of data stored in cylindrical coordinates (angle TH , radius R , height Z) to Cartesian coordinates X,Y,Z . The arrays TH , R , and Z must be the same size (or any of them can be scalar). TH must be in radians.

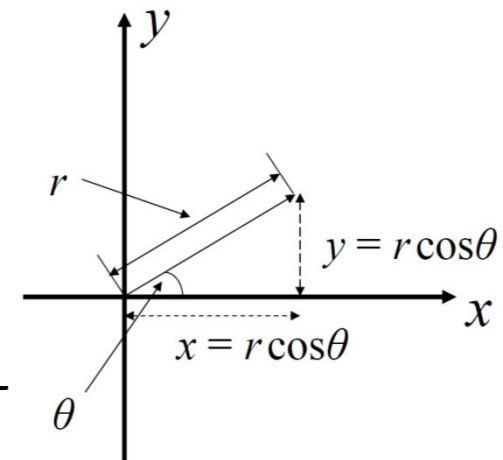
Class support for inputs TH,R,Z :

float: double, single

See also `cart2sph`, `cart2pol`, `sph2cart`.

Reference page in Help browser

`doc pol2cart`



Just like your code!

```
>> type pol2cart
```

```
function [x,y,z] = pol2cart(th,r,z)
```

```
%POL2CART Transform polar to Cartesian coordinates.
```

```
% [X,Y] = POL2CART(TH,R) transforms corresponding elements of data  
% stored in polar coordinates (angle TH, radius R) to Cartesian  
% coordinates X,Y. The arrays TH and R must the same size (or  
% either can be scalar). TH must be in radians.
```

```
%  
% [X,Y,Z] = POL2CART(TH,R,Z) transforms corresponding elements of  
% data stored in cylindrical coordinates (angle TH, radius R, height  
% Z) to Cartesian coordinates X,Y,Z. The arrays TH, R, and Z must be  
% the same size (or any of them can be scalar). TH must be in radians.
```

```
%  
% Class support for inputs TH,R,Z:  
% float: double, single
```

```
%  
% See also CART2SPH, CART2POL, SPH2CART.
```

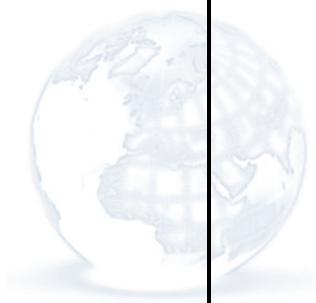
```
% L. Shure, 4-20-92.
```

```
% Copyright 1984-2004 The MathWorks, Inc.
```

```
% $Revision: 5.9.4.2 $ $Date: 2004/07/05 17:02:08 $
```

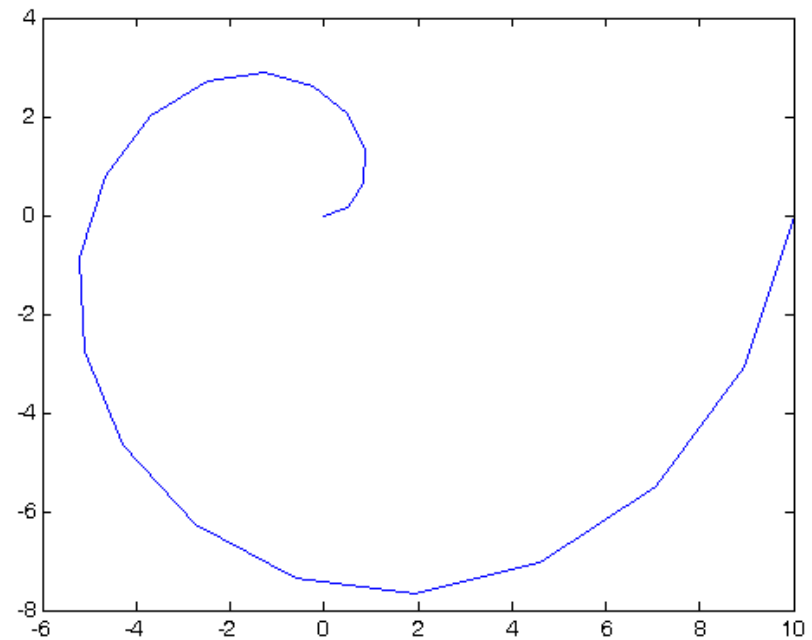
```
x = r.*cos(th);
```

```
y = r.*sin(th);
```



Exercise: Spiral

```
>> r = linspace(0, 10, 20);  
>> theta = linspace(0, 2*pi, 20);  
>> [x, y] = polar_to_cartesian(r, theta);  
>> plot(x, y);
```



Possible Cases

- **One input:**

```
function [o1, o2, o3] = myfunc(i1)
```

- **Three inputs:**

```
function [o1, o2, o3] = myfunc(i1, i2, i3)
```

- **No inputs:**

```
function [o1, o2, o3] = myfunc()
```

```
function [o1, o2, o3] = myfunc
```

- **One output:**

```
function [o1] = myfunc(i1, i2, i3)
```

```
function o1 = myfunc(i1, i2, i3)
```

- **No output:**

```
function [] = myfunc(i1, i2, i3)
```

```
function myfunc(i1, i2, i3)
```



Local Variables

```
function z = fun(x,y)

u = 3*x;
z = u + 6*y.^2;

% return missing is fine at end of file
```

- The variables x , y , u , z are **local** to the function `fun`, so their values will not be available in the workspace outside the function.
- See example below.

Example

```
>> x = 3;
>> b = 7;
>> q = fun(x, b);

>> x
x =
    3

>> y
??? Undefined function or variable 'y'.

>> u
??? Undefined function or variable 'u'.

>> z
??? Undefined function or variable 'z'.

>> q
q =
    303
```



Exercise

```
function show_date  
clear  
clc  
date
```

```
% how many inputs and outputs do we have?
```



Homework

```
function [dist, vel] = drop(v0, t)
% Compute the distance travelled and the
% velocity of a dropped object, from
% the initial velocity v0, and time t
% Author: Dr. Mohammed Hawa

g = 9.80665; % gravitational acceleration (m/s^2)
vel = g*t + v0;
dist = 0.5*g*t.^2 + v0*t;
```

```
>> t = 0:0.1:5;
>> [distance_dropped, velocity] = drop(10, t);
>> plot(t, velocity)
```

Local vs. Global Variables

- The variables inside a function are local. Their scope is only inside the function that declares them.
- In other words, functions create their own workspaces.
- Function inputs are also created in this workspace when the function starts.
- Functions do not know about any variables in any other workspace.
- Function outputs are copied from the function workspace when the function ends.
- Function workspaces are destroyed after the function ends.
 - Any variables created inside the function “disappear” when the function ends.



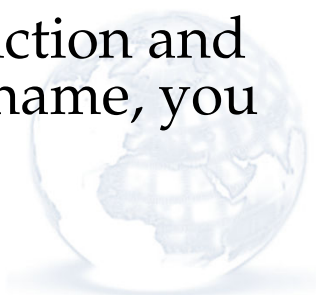
Local vs. Global Variables

- You can, however, define global variables if you want using the `global` keyword.
- Syntax: `global a x q`
- Global variables are available to the basic workspace and to other functions that declare those variables `global` (allowing assignment to those variables from multiple functions).



Subfunctions

- An M-file may contain more than one user-defined function.
- The first defined function in the file is called the *primary* function, whose name is the same as the M-file name.
- All other functions in the file are called *subfunctions*. They can serve as subroutines to the primary function.
- Subfunctions are normally “visible” only to the primary function and other subfunctions in the same file; that is, they normally cannot be called by programs or functions outside the file.
- However, this limitation can be removed with the use of function handles.
- We normally use the same name for the primary function and its file, but if the function name differs from the file name, you must use the file name to invoke the function.



Exercise

- The following example shows how the MATLAB M-function `mean` can be superceded by our own definition of the mean, one which gives the root-mean square value.

```
function y = subfun_demo(a)
y = a - mean(a);
```

```
function w = mean(x)
w = sqrt(sum(x.^2))/length(x);
```



Example

- A sample session follows.

```
>>y = subfn_demo([4 -4])  
y =  
    1.1716    -6.8284
```

- If we had used the MATLAB M-function mean, we would have obtained a different answer; that is,

```
>>a = [4 -4];  
>>b = a - mean(a)  
b =  
    4    -4
```

