An Introduction to Using MATLAB as a Research Tool

Instructor: Dirk Colbry, Ph.D.
Institute for Cyber-Enabled Research
colbrydi@msu.edu

"Learning your first computer language is like learning French poetry when you don't know French and you don't know poetry."
– Bill Punch, MSU Computer Science Professor

Agenda

- Motivation
- The MATLAB Interface
- MALTAB Command Syntax
- Programming with Scripts
  - Loop statements and block code
- Programming with Functions
- Loading and saving data

Sub-Agenda

- Where to find help with MATLAB
- Getting data inside of MATLAB
- Working with data in MATLAB
- Visualizing data using MATLAB

Motivation and Background

What is MATLAB?

- (Mat)rix (Lab)oratory
  - MATLAB is a high-level programming language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran.
  - This is accomplished by providing the user with extensive libraries of commonly used built-in functions. These functions allow users to focus on their research goals and avoid getting overrun by many unnecessary programming details.

Alternatives to MATLAB

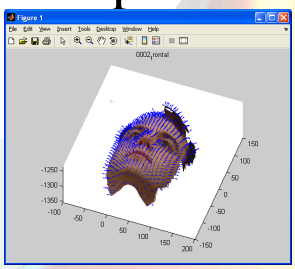
- R
- S-Plus
- SAS
- Mathematica
- Python
- Java
- C++
- Many more…

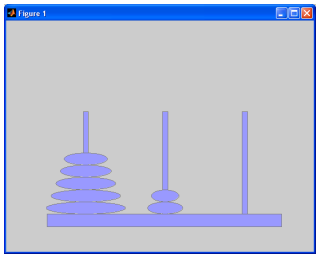
Why use MATLAB?

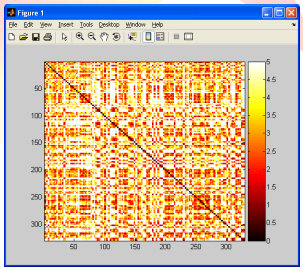
- MATLAB is designed to make it quick and easy to develop programs:
  - Uses an interpretive language, instead of a programming language that needs a compiler
  - Has an extensive library of existing functions
  - There are many existing resources online

A Few Examples

- Data Generation
- Data Analysis
- Data Visualization

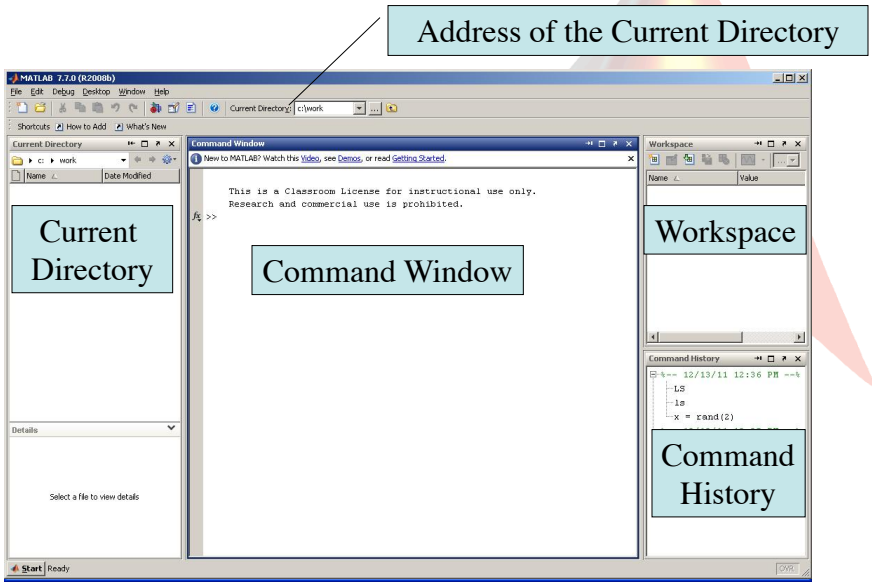






The MATLAB Interface

Navigating the Program



Interface Style

- You can drag and drop the different components of the MATLAB interface to make the program look and feel the way you want.
- You can use the button in the upper right corner of a component to "dock" a window or use the button to undock a window.
- You can always go back to the default interface arrangement by selecting Desktop→Desktop Layout→Default from the MATLAB menu.

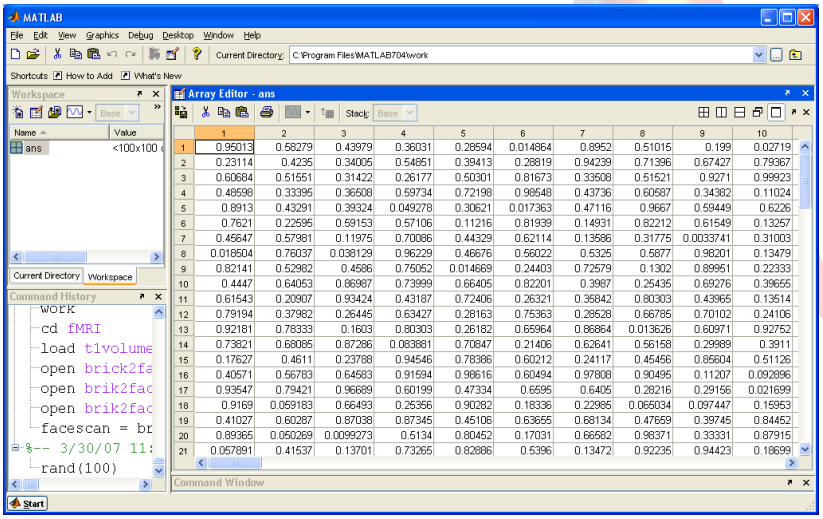
Using MATLAB as a calculator

- Try typing the following examples into the MATLAB command window:
  - » `10 + 20`
  - » `sqrt(99)`
  - » `r = 2`
  - » `C = 2*pi*r^2`
- What variables do you see in the workspace?

MATLAB Variable Editor

- Set up a basic variable:
  - » `x = 0;`
- Double click on the variable in the workspace.
  - The Variable Editor window will pop up.
- Cut and paste values to and from the Variable editor to Windows excel.

Variable Editor



Command Line Navigation

- The **>>** symbol is called the "command prompt."
- You can always double click on a command in the command history and the computer will run that line of code again.
- You can also use the up and down arrows to search though the command history.
- If you type the first few letters of a command and then use the up and down arrows, you will search only for commands starting with those letters.

Text Editor

- The editor is not in the workspace by default.
- You can start it by typing "edit" on the command line.
- Separate text regions by using the "%%" operator. (more about this later).

Language Syntax

Getting HELP!

- From the command line type:
  - » `help`
  - » `doc`
- If you do not know what a command does, type help and then the command name:
  - » `help plot`
  - » `doc datatypes`
- Do not be afraid to try the examples
  - – Copy and paste the example to the command line
- Use the following commands to start over:
  - » `close all; clear all; clc;`

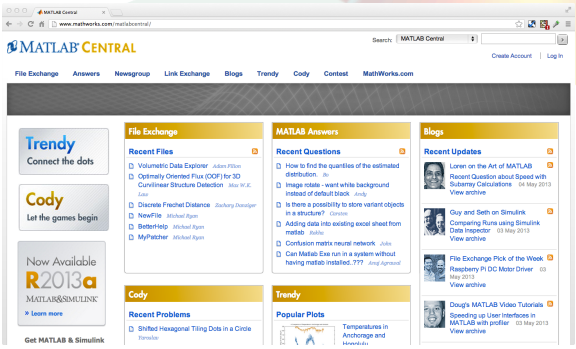
Doing a help Example

- Find a help message with an example:
  - » **`help avifile`**
- Copy the entire help message verbatim to the command window
- See the file this example created:
  - » **`ls`**

MATLAB Central

http://www.mathworks.com/matlabcentral/

- File exchange with free MATLAB software
- Newsgroups and online help



ceer Center for Engineering Education Research @ Michigan State University

- Resource developed as part of CSE 131 here at MSU

http://ceer.egr.msu.edu/matlab-resource

The MATLAB Interface (everything is text)

- Base expressions
  Numbers, Strings, `+, -, *, ^, /,` etc…
- Commands (functions and scripts)
  `help, plot, sqrt, rand,` etc.
- Variables
  `x, data, ans,` etc.
- Comments
  `% Ignored text.`

Basic Command Syntax

```
[output1, output2, …] = command(input1, input2, …);
```

- Command name
  - This is the name of the script or function.
  - Both functions and scripts have command names, however, scripts do not have inputs or outputs.
  - The command name is normally the same name as the file which defines the command.
  - Typing "help <command name>" will cause the help message for that command to appear.
  - The command name is case sensitive, but MATLAB will search for the closest match if the case sensitive one is not found.

Command Name Examples

- Example Commands:
  - » **`figure`**
  - » **`rand`**
  - » **`ls`**
- Type 'help' and then the command names.
- Type 'open' and then a command name.
  - Warning: you can edit commands that are open in the editor. Be careful to not make or save any changes to built-in MATLAB commands!
- Try adding capital letters to commands:
  - » **`LS`**
  - » **`RAND`**
  - » **`Figure`**

Basic Command Syntax

```
[output1, output2, …] = command(input1, input2, …);
```

- Inputs:
  - Comma separated list in parentheses.
  - A function is able to take different numbers of inputs and may perform differently for different numbers of inputs.
  - String inputs must be surrounded by single quotes.
  - If the inputs are all strings, the parentheses, commas and single quotes can be replace with white space.
    - Note: in this special case, no outputs will be assigned.
  - Note: scripts do not have inputs.

Input Examples

- Example commands with inputs:
  - » `rand(2);`
- Example of different behavior (overloading)
  - » `linspace(0,2*pi)`
  - » `linspace(0,2*pi,10)`
- Special case with strings as the only input
  - » `ls('c:\')`
  - » `ls c:\`
  - » `clear all`

Basic Command Syntax

```
[output1, output2, …] = command(input1, input2, …);
```

- Assignment and output
  - Comma separated list of variables in brackets.
  - A function may perform differently depending on the number of outputs that are requested.
  - If only one output is required, then the brackets and commas are not needed.
  - If the assignment and output variables are removed the system will automatically assign **`output1`** to **'`ans`'**, the default output variable.
  - Note: scripts do not have outputs.

Output Examples

- Example commands with outputs:
  - » `x = rand([1,2])`
  - » `f = figure`
  - » `im = imread('ngc6543a.jpg')`
  - » `h = image(im)`
  - » `[x, y] = ginput(1)`

Get 1 x,y input coordinate from the mouse. (click on the figure)

- Using the default assignment
  - » `rand(1)`
  - » `sqrt(26)`

Note: if you are working with images consider the image processing toolbox and the newer imshow command.

Basic Command Syntax

```
[output1, output2, …] = command(input1, input2, …);
```

- Display Output semicolon (Optional)
  - If the semicolon is not included, then MATLAB will automatically display the contents of the output variables to the terminal display.
  - If the semicolon is included, then the command will run "quietly" and not output to the terminal display.
- Semicolon also ends a command
  - Two commands can be placed on the same line of input

Semicolon Examples

- Display results
  - » `x = linspace(0,2*pi)`
- Do not display results
  - » `x = linspace(0,2*pi);`
- More than one command on a line
  - » `y = sin(x); plot(x,y);`

Overloading

- Functions can change what they do based on the type and number of inputs and outputs.

```
» x = linspace(1,100);
» y = rand([100 1]);
» y = sort(y);
» plot(x,y);
» plot(x, y, '*r');
```

Same function different numbers of inputs and different results.

Naming Commands and Variables

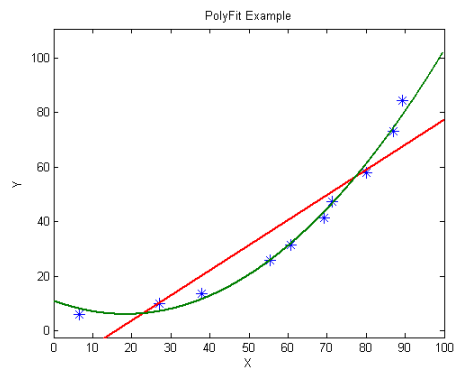
- There are special characters that cannot be used in names, including:

  <space> : \ * & + - ( ) [ ] { } # % @ etc…
- Names should be short and make sense
- Try not to reuse existing command and variable names
- Some good names include:
  - **beedata**
  - **timedata**
  - **videoplotfun**
  - etc.

Some bad names include:
- **sqrt**
- **var**
- **a, b, c, d, e,**
- etc.

Project 1: Fitting Polynomial Functions

- Use the following set of functions to input data, display them in a figure and fit a polynomial to the data. (Hint: use the help command.)
  - » **figure**
  - » **axis**
  - » **ginput**
  - » **plot**
  - » **polyfit**
  - » **hold**
  - » **ezplot**



Example Review

```
>> figure;
>> axis([0 100 0 100]);
>> [x y] = ginput(10);
>> plot(x,y,'dr');
>> p = polyfit(x,y,1)

p =

    0.8415    6.6390

>> hold on;
>> ezplot('0.8415*x + 6.6390', [0 100 0 100]);
>> hold off;
```

Function will wait until you click on your figure 10 times:

Results will vary depending on what points you clicked

Text and Title Commands

- `help title`
- Sometimes you get strange results
  - » `figure`
  - » `title('hello_world')`
- This is because MATLAB uses a tex interpreter to display mathematical functions
  - » `xlabel('2\pir^2');`
- Most of the time you do not want to use the tex interpreter.
  - » `ylabel('time_seconds', 'Interpreter', 'none');`
- If you want to learn how to use the tex interpreter, you can just Google tex or latex and read about the math environment.

Nesting

```
[output1, output2, …] = command(command2(), input2, …);
```

- The `output1` of one command can be the input to another command.
  - The value of the input will be the same as output1 of the nested command.
  - Nesting can continue as long as you like.

Example Nested Commands

- Here is an example of a non-nested command:
  - » `x = rand([100 1]);`
  - » `y = sort(x);`
  - » `plot(y);`
- Or using nested commands:
  - » `plot(sort(rand([100 1])));`
- Note: there is only one semi-colon.

Matrixes Assignments

- Basic Scalar Assignment:
  - » `x = 5`
- Basic Vector Assignment:
  - » `v = [1 2 3 7 8]`
- Basic Matrix Assignment:
  - » `m = [ 1 2 3 7 8; 5 2 4 5 3]`

Matrix Multiplication

- Inverse of a matrix
  - » `x = [1 2; 3 4]`
  - » `inv(x)`
- Transpose of x
  - » `x'`
- Matrix Multiplication
  - » `x * inv(x)`
- Item by item Multiplication
  - » `x .* inv(x) % notice the period`

Matrix Manipulation

- Vertical Concatenation
  - » `m2 = [ v; v; m]`
- Horizontal Concatenation
  - » `m3 = [ v v m]`
- Accessing only the first row of a matrix
  - » `x = m2(1,:)`
- Accessing only the first column of a matrix
  - » `y = m2(:,1)`

The : colon character

- It can be used to define a vector of numbers
  - » `X = 1:10`
  - » `Y = 1:2:20`
  - » `Z = 20:-1:1`
- It can also be used to index a matrix
  - » `x = rand(10)`
  - » `x(1:2, 3:5)`
  - » `x(1:2, :)`

Data Types (skipping)

Numeric (integer, single, double, unit8, etc)

- A double is the default numeric class in MATLAB
- Numeric operators include:
  (+ add) (- subtract) (* multiply) (/ divide) (^ power)
- The different numeric datatypes are needed to represent different classes of numbers:
  - Floating points
  - Negative numbers
  - Memory requirements
- A double will be able to do most of what you want. It can represent large floating point numbers with negative and positive values.

Casting

- Changing from one numerical type to another
- If you want to change from a floating point to an integer
  - **`round(5.6)`** or **`uint64(5.6)`**
- If you want to change an integer to a double you need to cast
  - **`double(x)`**

Memory Storage

- A bit is a one (1) or a zero (0)
- A byte is eight bits (a byte is the smallest amount of data represented in MATLAB)
- Different datatypes have different sizes

```
» clear all
» d = double(10);
» ui8 = uint8(10);
» ui32 = uint32(10);
» ui64 = uint64(10);
» s = single(10);
```

Examples

- Integers are required to index a matrix

```
» X = rand(5);
» X(1,2)
» X(1.5,2.5) %This causes an error
```

- Color images are normally represented by a three dimensional matrix (rows, columns, color) of uint8.
  - In other words: three, two dimensional arrays representing red, green and blue.
  - Each item in this 3D matrix is traditionally represented by a number from 0-255, which is an 8 bit binary number.

(Char)acter

- A char is a number between 0 and 65535.
  - How many bits is this?
- Each number is mapped to a specific letter in the alphabet; like a code.
- Different languages and fonts can have different mappings.
- ASCII is a universal standard for mapping the characters on a keyboard to one of the first 127 numbers.

ASCII – American Standard Code for Information Interchange

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Understanding Characters

- A string is just a vector of characters:
  - » `x = 'hello'`
  - » `y = 'world'`
  - » `x + y`
  - » `[x y]`
  - » `[x ' ' y]`
- An integer from 0-255 can be turned into a character:
  - » `x = [72 73];`
  - » `x = char(x)`
- Or you can change a character back into its number:
  - » `x = 'Hello World';`
  - » `double(x);`

Numbers and Character Paradox

- Here is an odd example:
  - » `x = '5'`
  - » `double(x);`
- Why does it print out 53 and not 5?
- We could subtract 48 and get the number.
- Or we can use a built in functions:
  - – `str2double` and `num2str`

String Compare - **strcmp**

- Compare two strings and return a boolean
  - » `h1='hello'; h2='world';`
  - » `h1==h2 % doesn't work`
  - » `sum(h1 == h2) % doesn't work`
  - » `sum(~(h1 == h2)) % doesn't work`
  - » `sum(~(h1 == h2)) == 0 % works`
  - » `sum(~(h1 == h1)) == 0 % works`
- Or use strcmp, which is much easier
  - » `strcmp(h1, h2)`
  - » `strcmp(h1, h1)`

Why doesn't this work?

- List of strings
  - » `x(1,:) = 'Hello everybody';`
  - » `x(2,:) = 'Ha Ha';`
  - » `x(3,:) = 'Thank you, come again';`
  - » `x(4,:) = 'Eat my shorts';`
  - » `x(5,:) = 'Excellent';`
  - » `x(6,:) = 'D''oh';`

Cells (note {curly} brackets)

- List of strings

```
» x{1} = 'Hello everybody';
» x{2} = 'Ha Ha';
» x{3} = 'Thank you, come again';
» x{4} = 'Eat my shorts';
» x{5} = 'Excellent';
» x{6} = 'D''oh';
```

Scalar → Vector → Matrix

- These are the most restrictive container class, but also the most widely used.
  - i.e., all of the components of the vector or matrix much be of the same data type and size.
- Accessing a Vector or Matrix:

  X(1,2) ← returns the component of the first row and the second column.

Cell → Cell Array

- A Cell is a container for any type of object. A Cell array allows you to make an array of objects that vary in type or size.
- Example cell array:

```
x = { '100' 100 10000 'hello world'}
```

- Accessing a cell array:

  `x{1}` ← returns the contents of the first cell

  `x(1)` ← returns the first cell as a cell

- Examples to try:

```
x{5} = 'bob';
x(5)
x{5}
```

Struct → Struct Array

- A struct is a structure of data types in MATLAB. These structures are also called objects.
- Example struct:

```
>> X.bob = 10;
>> X.cat = 20;
>> X.hello = 'Good day';
```

- Example struct array:

```
>> d = dir

13x1 struct array with fields:
    name
    date
    bytes
    isdir
```

- Accessing a struct array:

  d.name ← returns all of the names in the array.

  d(4).name ← only returns the name of the fourth struct.

Printing more complex output

» **`help sprintf`**

- There are special characters that can be used in a formatted string:
  - \t – tab
  - \n – new line
  - \\ – '\' backslash character
  - `''` – single quote
- Example:
  » `sprintf('Dirk''s email:\n\tdirk@colbry.com\n')`

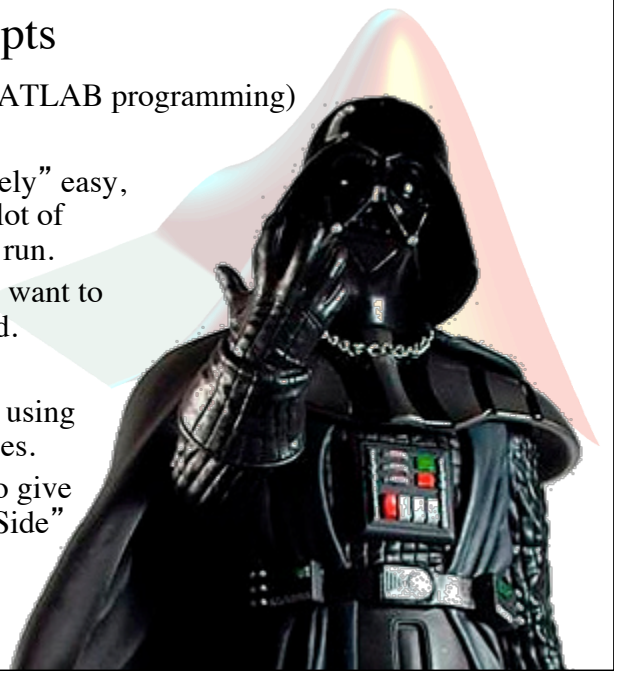
Programming With Scripts

Scripts

(The "Dark Side" of MATLAB programming)

- Scripts are "seductively" easy, but will cause you a lot of problems in the long run.
- Most of the time you want to use a function instead.

- However, we will be using scripts in our examples.
- Just remember, not to give in to the "Dark Side"



Scripts

- Put all of your commands in a single text file (you can use MATLAB's built-in editor).
- Name the file with the .m extension (filename.m).
- Type in the text file name to run the commands.
- Script do not have their own workspace. Instead, they use the current workspace. (I will explain this more when I talk about functions.)

Example Script

C:\Documents and Settings\Dirk\My Documents\CurrentWork\Teaching\PSY992_F06\testscript.m

```
% This is a comment.  The system will ignore anything with a comment.

% This is an example script program.
%
% This script plots some two dimensional data on the screen and then fits
% some curves to the data.

[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
meshc(X,Y,Z);

%Extra commands that are commented out.
%hold on;
%surf(X,Y,Z);
%hold off;

%colormap cool;
%axis([-3 3 -3 3 -10 5])
```

Crop Image Example

- A grayscale image is a matrix of values between 0 and 255.

```
im = imread('ngc6543a.jpg');
image(im);

im2 = im(70:530, 90:520, :);
image(im2);
```

- Note: Images can get warped
  - (type "`axis off equal;`" to see a clean image).

Block Code

"`if / else`" Statement

- If something is true do x, otherwise, do something else.

```
x = input('Enter a number and then enter ');
if(x > 9)
   % This code will only execute if x > 9
   disp('Number is greater than 9');
else
   % This code will only execute if x ~= 9
   disp('Number is less than 9');
end
```

Truth Statements

- Relationship Operators

  == - Equal
  ~= - Not equal
  < - Less than
  > - Greater than
  <= - Less than or equal
  >= - Greater than or equal

- Logical Operators

  & - logical AND
  | - logical OR
  ~ - logical NOT

"for" Statement

- Cycle though a vector one item at a time

```
figure;
hold on;
a = [0 100 0 100];
axis(a);
for i = 1:10
  [x(i) y(i)] = ginput(1);
  plot(x,y,'*');
  axis(a);
end
```

Group Practice

Lets turn this into a script (hint: use num2str)

```
>> figure;
>> axis([0 100 0 100]);
>> [x y] = ginput(10);
>> plot(x,y,'dr');
>> p = polyfit(x,y,1)

p =

    0.8415    6.6390

>> hold on;
>> ezplot('0.8415*x + 6.6390', [0 100 0 100]);
>> hold off;
```

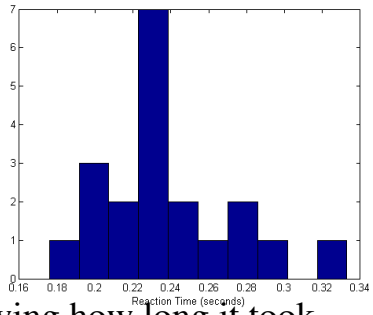
Response time experiment

- Write a script that measures the response time of a user.
- Outline of the task:
  - Describe research objective
  - Flow chart the program
  - Look up the necessary functions
  - Write the program

Project 2: Response time experiment

- Write a script that waits for a random amount of time between 1 and 2 seconds and then asks for user input (return key). Repeat 20 times.

```
for, end
rand
pause
tic, toc
beep
input
Hist
```



- Display a histogram showing how long it took between prompting the user and getting a response.

"while" Statement

- Keep doing something while a statement is true.

```
x = input('Type a number and then enter ');
while(x != 9)
  x = input('Type a number and then enter ');
end
```

Consecutive if statements

```
x = input('Type in a number and press <enter> ');
if(x == 1)
    disp('one');
else
    if(x == 2)
        disp('two');
    else
        if(x == 3)
          disp('three');
        else
            disp('more than three');
        end
    end
end
```

"switch / case" Statement

- Simple way to display a series of if statements.

```
x = input('Type in a number and press <enter> ');
switch(x)
    case(1)
        disp('one');
    case(2)
        disp('two');
    case(3)
        disp('three');
    otherwise
        disp('more than three');
end
```

"try / catch" Statement

- Try to do a command, if there is an error, address it and move on.

```
name = input('Type in an image file name with '' marks ');
try
    im = imread(name);
    image(im);
catch
    disp('could not open file');
end
disp('program did not exit');
```

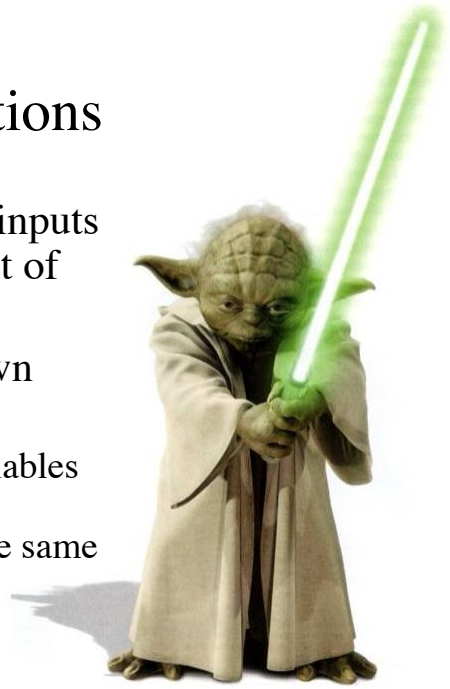
Block code Review

- **if / else** – Do the "if" block only if the statement is true. If the statement is not true, do the "else" block.
- **for** – Do block for a fixed number of times.
- **while** – Keep doing a block while a statement stays true.
- **switch/case** – Switch between blocks based on different cases of a variable.
- **try/catch** – Try a block. If the block fails, catch the error and do this other block.
- **end** – The end of a Block.

Function Programming

Functions

- Functions take a set of inputs and return a separate set of outputs.
- Functions have their own workspace.
  - This makes naming variables easier because different workspaces can have the same variable name.



Functions

- To change a script into a function the following line needs to be the first line in your file:

```
function [outputlist] = name(input list)
```

Example Function (functionList.m)

Output Variable(s)

Function Name (same as file)

Input Variable(s)

Function Declaration

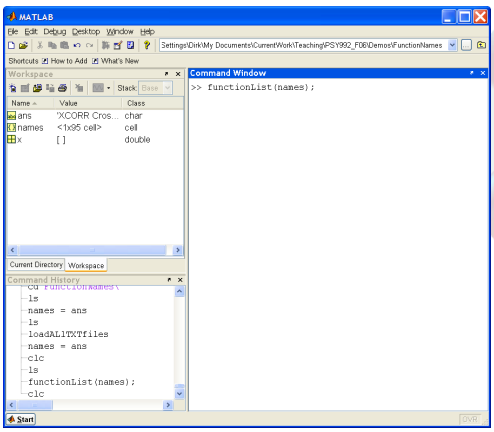
'Help' Comment Block

Program

```
function s = functionList(names)
% Written by Dirk Colbry
% 09-12-06
% Display the descriptions of a set of MATLAB commands
%

names = sort(names);
for i = 1:length(names)
    try
        h = help(names{i});
        s = strfind(h,10);
        s = h(1:s(1));
        s = strtrim(s);
        disp(s);
    catch
        disp(['<a href=""> Error - ' names{i} '</a>']);
    end
end
```

Every function has its own workspace



- When a function starts, its workspace only contains the inputs to the function (plus some special variables).
- When a function exits, only the output variables are in the main workspace.
- Variables that are inside and outside of the workspace are different, regardless of the variable names.
- For instance, if the variable 'x' is in the main workspace and there is also a variable named 'x' in my function workspace, they can have different values and it will not cause an error

Scripts vs. Functions

- Why Scripts are bad:
  - They share the same variable space (workspace) as the main program.
  - So, every time you need a new variable you have to make sure that you did not use the same name in the past or it could cause unwanted errors
- Why Functions are good:
  - Each function has its own variable space.
  - Functions make your code simple because any change you want to make only needs to be made once.
  - Functions help you organize your code.

Loading and Saving Data

File I/O

Saving and restarting MATLAB

- At any point you can save your MATLAB session:

```
>> save('mysession');
```

- Then you can exit MATLAB and reload your session latter:

```
>> load('mysession');
```

Types of files

- Just like variables, every file is a group of numbers.
- The program needs to know what the numbers mean in order to read the files.
- Since the numbers could mean anything, some standards have been adopted that make reading the file easier.
- There are generally two major classes of files, ASCII and Binary.

All files are given a file ID

- The **`fopen`** command opens a file and returns the file ID.
- Any command that can read or write to a file will normally take the file ID as an input.
  - **`fread, fwrite, fprintf, fgets, fgetl, fscanf, fseek, etc.`**
- After you are done accessing the file you should always use the **`fclose`** command.

fopen

- `fid = fopen(filename, permissions)`
- The permissions string can include:
  - `'r'` read
  - `'w'` write (create if necessary)
  - `'a'` append (create if necessary)
  - `'r+'` read and write (do not create)
  - `'w+'` truncate or create for read and write
  - `'a+'` read and append (create if necessary)
  - `'W'` write without automatic flushing
  - `'A'` append without automatic flushing

Example Function

```
function showfile(filename)
%SHOWFILE - display the contents of a file as ASCII

fid = fopen(filename, 'r');

while 1
    tline = fgetl(fid);
    if ~ischar(tline)
        break
    end
    disp(tline)
end
fclose(fid);
```

Text (ASCII) files

- In a text file, the list of numbers is taken from the ASCII table.
- Many programs can read text files (Notepad, MATLAB, etc).
- Some common text formats are:
  - Web pages (.html)
  - MATLAB programs (.m)
  - Text file (.txt)

Special ASCII files

- MATLAB can read any file. However, you need to tell MATLAB what you want it to mean.
  - Line Delimited files
  - Space Delimited files
  - Comma Delimited files

Binary files

- Binary files are more compact than text files. However, it is difficult to load binary files because the format of the file is unknown.
- Some binary files follow a know standard. The file extension tells the computer which standard is being used:
  - Image files (bmp, jpg, etc)
  - Sound files (mp3, wav, au, etc)
  - Proprietary formats (doc, pdf, mat, etc)

Specific I/O Commands

- General
  - load / save
- ASCII
  - csvread / csvwrite – comma separated data
  - dlmread / dlmwrite – ASCII delimitated data
  - textscan – specialized format data
- Binary
  - wk1read / wk1write – lotus notes spreadsheet file
  - xlsread / xlswrite – excel files
  - imread / imwrite – image files
  - aviread / aviwrite – movie files

Solution to Group Practice

```
figure
axis([0 100 0 100]);
[ x y] = ginput(10);
plot(x,y, 'dr');
p = polyfit(x,y,1);
hold on;
equ_str=[num2str(p(1)) '*x + ' num2str(p(2))];
ezplot(equ_str, [0 100 0 100]);
hold off;
```

Solution to Project 2

```
for i = 1:20
    pause(rand(1)*2);
    tic;
    x = input('press the (enter) key');
    t(i) = toc;
end
hist(t);
```