CM2202: Scientific Computing and Multimedia Applications
MATLAB Programming: 1. MATLAB Basics

Dr. Yukun Lai

School of Computer Science & Informatics

What is MATLAB?

MATLAB is:

An interactive, matrix-based system for scientific and engineering numeric computation and visualization.

You can solve complex numerical problems in a **fraction of the time** required with a programming language such as Java or C++.

The name MATLAB is derived from **MAT**rix **LAB**oratory.

MATLAB System (1)

The full MATLAB system consists of:

MATLAB language — high level interpreted language optimised for matrix/array operations and contains many other useful features.

Integrated Development Environment — MATLAB's IDE contains a number of tools for managing programs, editing files, saving workspace, command history, debugging and more.

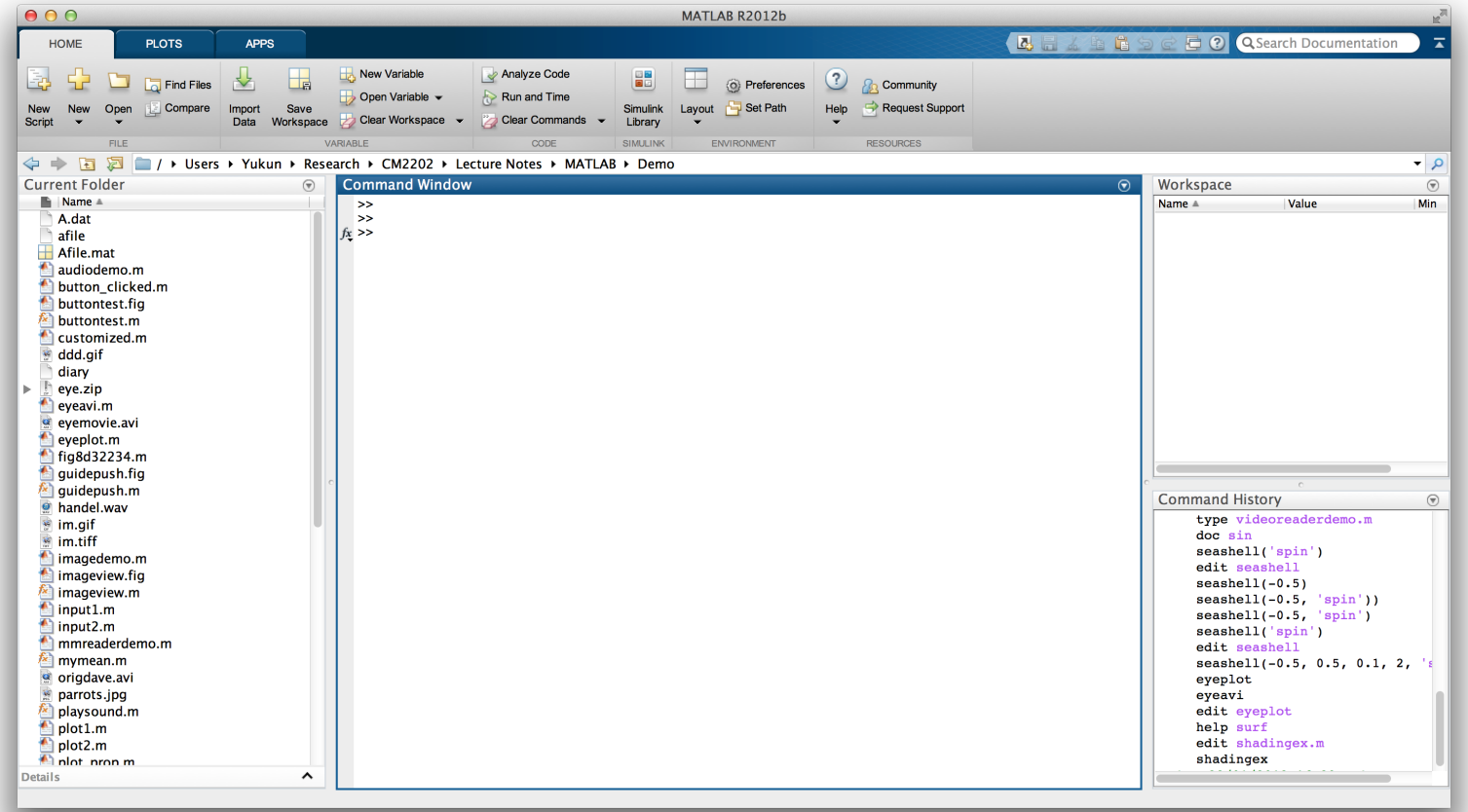
Handle Graphics® — Unparalleled suite of tools for both high-level graphing and display of data and basic graphic/image processing, as well as low-level commands for displaying and controlling graphics and building GUIs.

MATLAB System (2)

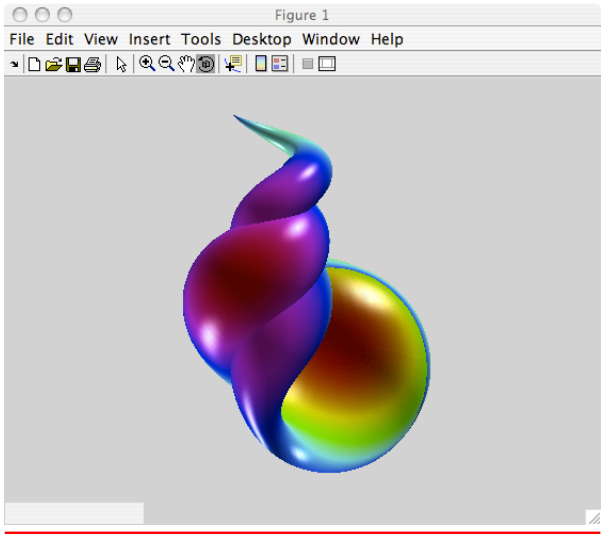
Extensible system — Vast array of Mathworks supplied, third party and freely available *toolboxes* to provide facilities for doing almost any sort of computation.

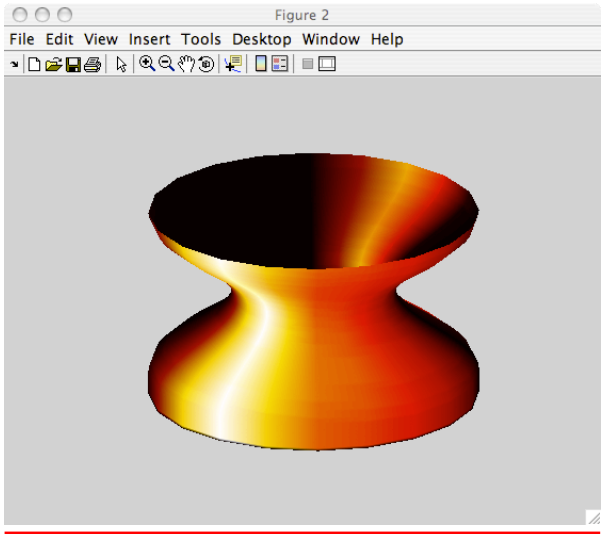
Application Program Interface — API link to Java/C/Fortran programs.

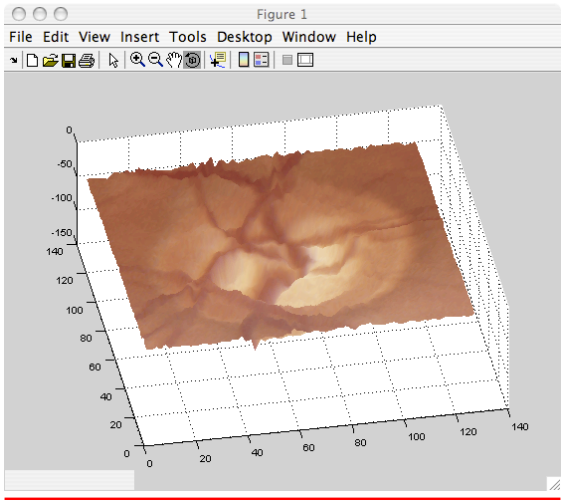
MATLAB IDE


MATLAB R2012b
HOME
PLOTS
APPS
Search Documentation
New Script
New
Open
Find Files
Compare
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
Analyze Code
Run and Time
Clear Commands
Simulink Library
Layout
Preferences
Set Path
Help
Community
Request Support
FILE
VARIABLE
CODE
SIMULINK
ENVIRONMENT
RESOURCES
/ ▸ Users ▸ Yukun ▸ Research ▸ CM2202 ▸ Lecture Notes ▸ MATLAB ▸ Demo
Current Folder
Name
A.dat
afile
Afile.mat
audiodemo.m
button_clicked.m
buttontest.fig
buttontest.m
customized.m
ddd.gif
diary
eye.zip
eyeavi.m
eyemovie.avi
eyeplot.m
fig8d32234.m
guidepush.fig
guidepush.m
handel.wav
im.gif
im.tiff
imagedemo.m
imageview.fig
imageview.m
input1.m
input2.m
mmreaderdemo.m
mymean.m
origdave.avi
parrots.jpg
playsound.m
plot1.m
plot2.m
Details
Command Window
>>
>>
>>
Workspace
Name
Value
Min
Command History
type videoreaderdemo.m
doc sin
seashell('spin')
edit seashell
seashell(-0.5)
seashell(-0.5, 'spin'))
seashell(-0.5, 'spin')
seashell('spin')
edit seashell
seashell(-0.5, 0.5, 0.1, 2,
eyeplot
eyeavi
edit eyeplot
help surf
edit shadingex.m
shadingex

MATLAB Graphics Examples







Why MATLAB for this module? (1)

MATLAB advantages

MATLAB is a platform-independent interpreted language optimised for numerical (matrix and array) computation:

- It allows one to perform numerical calculations easily: Simple High Level Syntax
- It allows one to visualize the results without the need for complicated and time consuming programming: One or two line of MATLAB code in most simple cases
- Optimised for matrix and array structures — all our multimedia data structures are arrays or matrices
- Simple yet powerful MATLAB Integrated Development Environment (IDE)

Why MATLAB for this module? (2)

MATLAB advantages

- Rich support of multimedia formats
  One or two line of MATLAB code reads audio and imagery direct to arrays for immediate simple processing
- Rich support of computational algorithms and tools
  Proprietary and freely available web *toolboxes* for Signal, Image, Video and many more processing

Summary: MATLAB allows its users to accurately solve problems, produce graphics easily and produce code efficiently, thus particularly suitable for Scientific Computing and Multimedia Applications.

Why MATLAB for this module? (3)

MATLAB disadvantages

Because MATLAB is an interpreted language

- It can be slow:
  But generally quicker development cycle than coding/debugging/(re)compiling normal languages
  Can easily port MATLAB to faster implementations later.
- Poor programming practices can make it unacceptably slow.
  Attend Lectures and Lab Classes to learn how to do it properly
- Cost: **not gnu! or open source/freeware**
  Student editions available — not full MATLAB toolbox support.
  Free alternatives: Octave, Scilab: similar syntax but *not* 100% compatible.

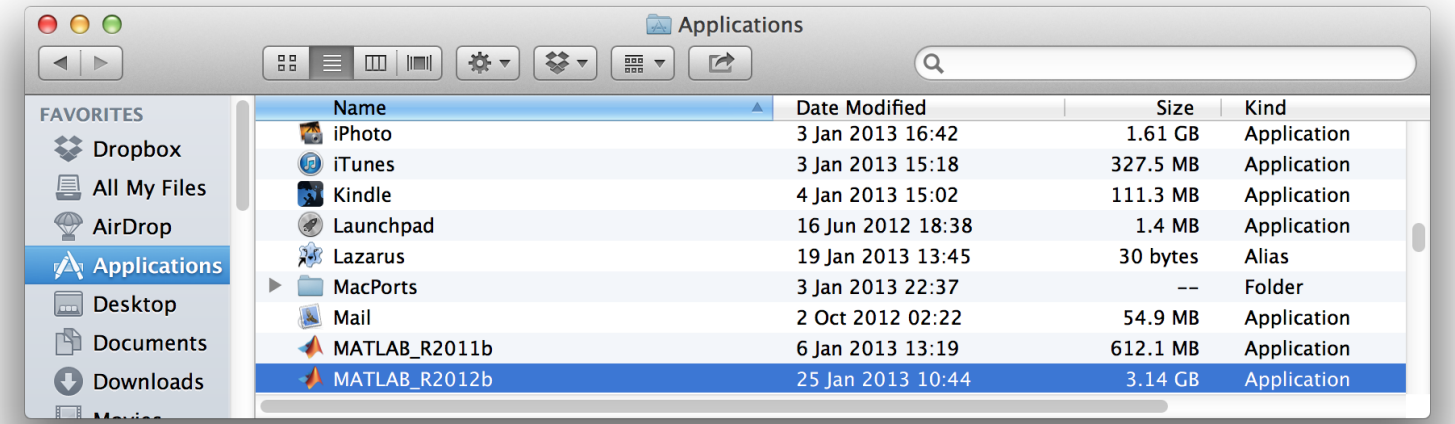
Useful Web Links

- MATLAB Primer — now a text book but older web page still relevant.
- Mathworks online MATLAB tutorials
- MATLAB — some basics
- UNH Math Dept's MATLAB Tutorials, Clarkson Univ.

Getting MATLAB Started

From Mac Finder:

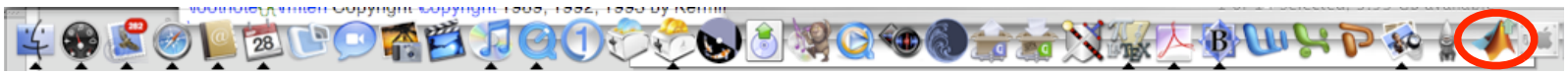
- From **Applications** folder double click on **MATLAB_R20XXa/b Folder**.



Getting MATLAB Started (cont.)

From Mac Dock:

- **Double click** on **MATLAB launcher application icon**.

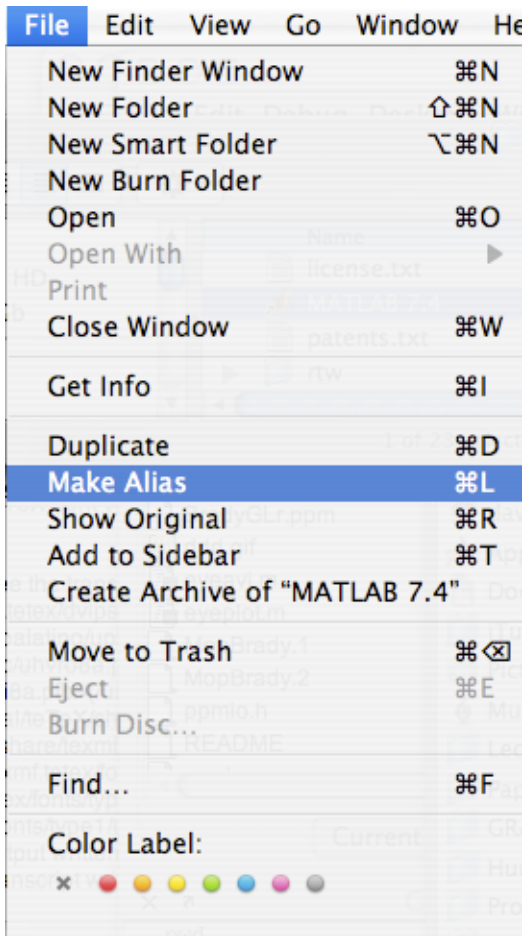


- To add this application to your dock, run application as above and then from dock **control/right mouse click** on the application icon, select **Keep in Dock**

Getting MATLAB Started (cont.)

From Mac Desktop:

- Place or add **MATLAB launcher application alias** into your Mac desktop.
- **Double click** on **MATLAB launcher application icon**.
- To add an alias to your desktop, find and select application in finder, from main menu **File** pull down menu select **Make Alias** (apple key+L).



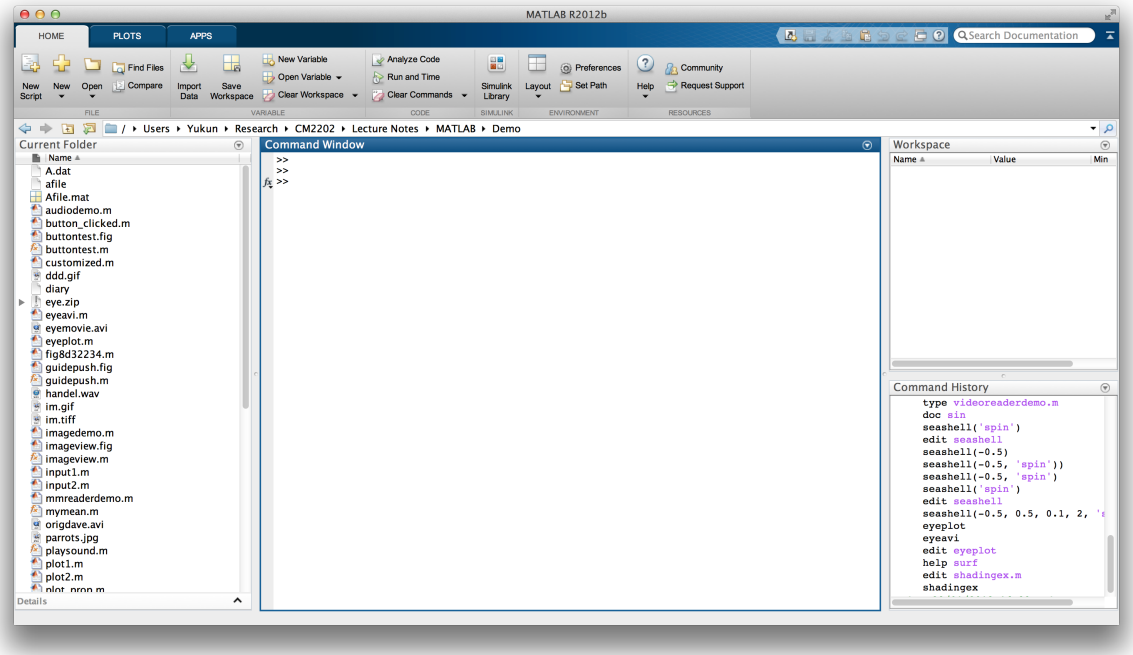
MATLAB Main Window

The MATLAB IDE has 4 Main sub panels:

- Current Folder
- Workspace
- Command History
- Command Window

It also has:

- Menu bar
- Toolstrip/Toolbar



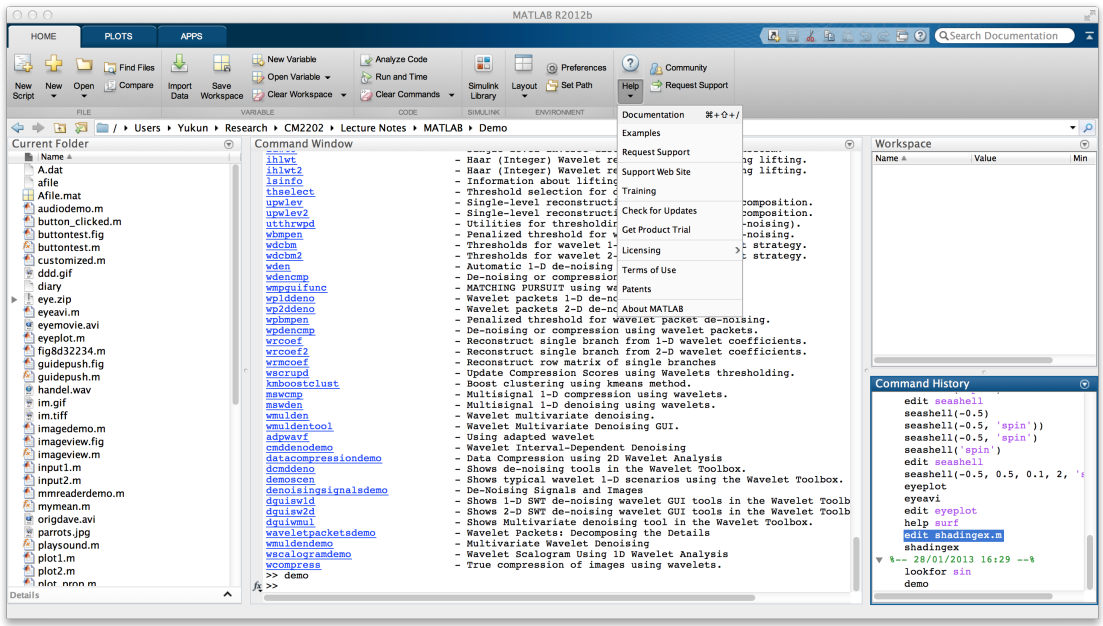
Getting Help

MATLAB provides a few ways to get help or more information about all its functions and utilities.

- From the command line,
  - type `help` or `help fn_name` for help text in Command Window.
  - type `doc` or `doc fn_name` for detailed document in Help Browser.
  - type `lookfor keyword` to search functions for keyword.

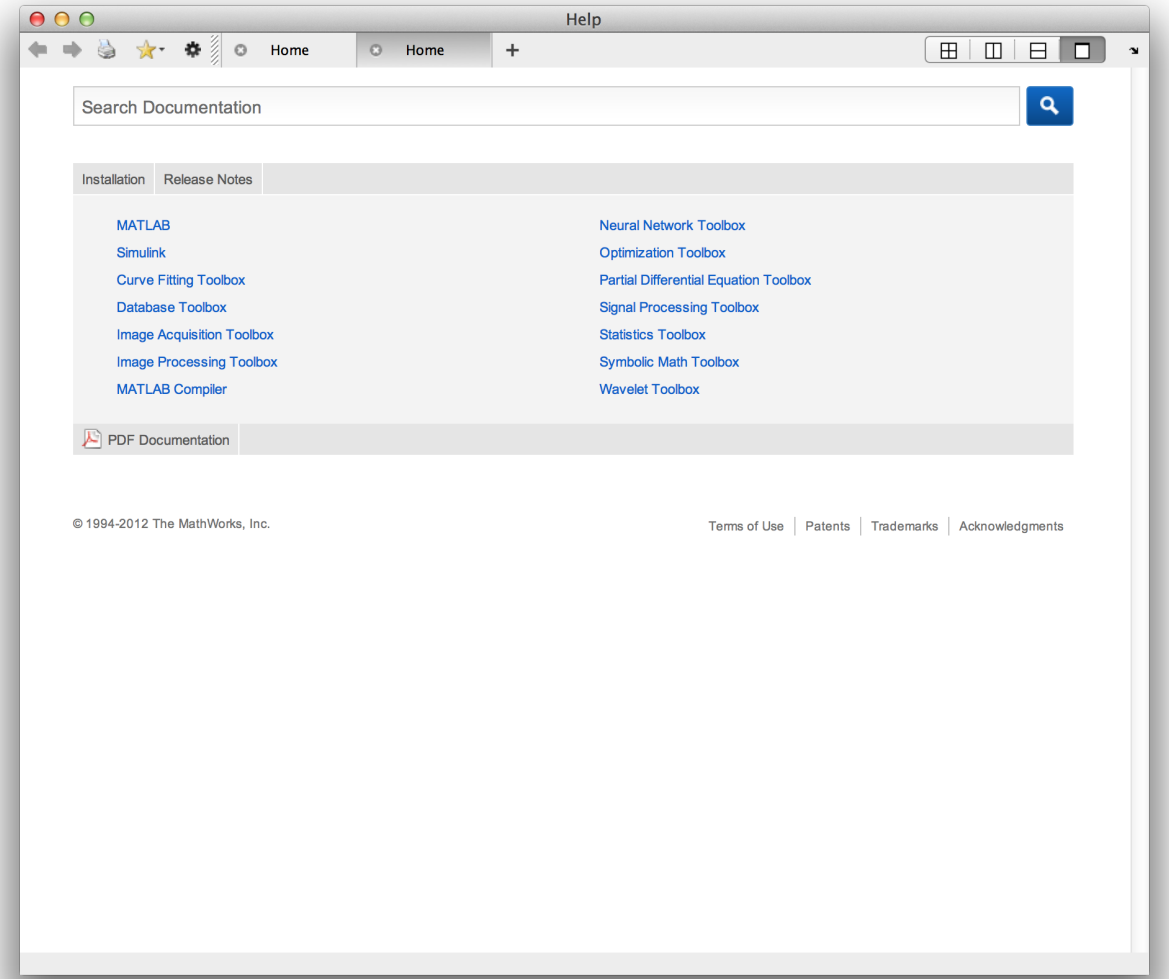
Getting Help (cont.)

- From the main toolstrip, select **Help** button
- Many options: Full documentation and Examples



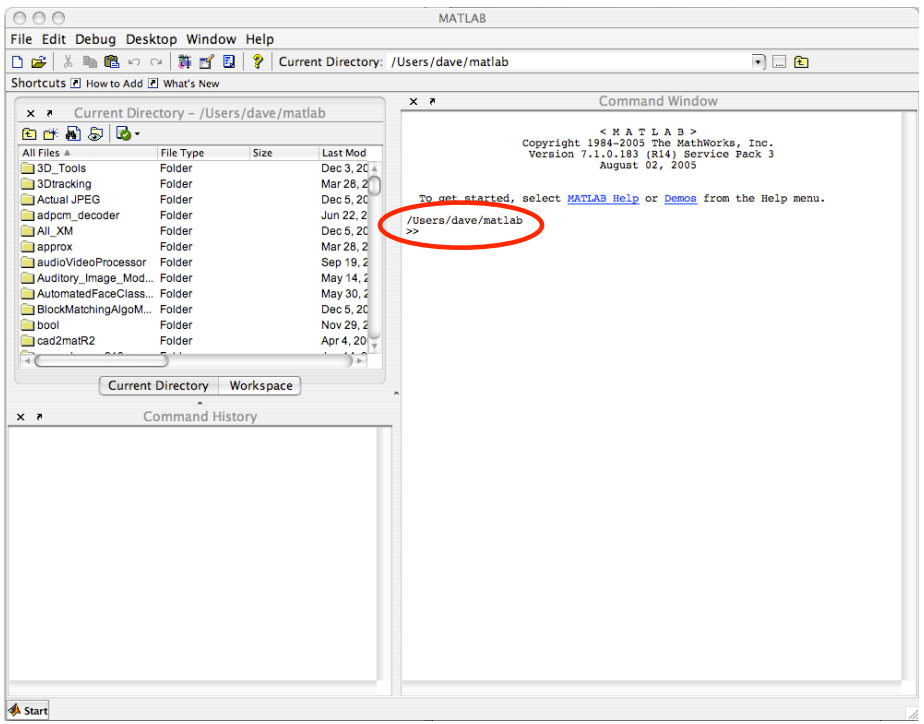
Getting Help (cont.)

MATLAB Help Window



MATLAB Programming

- The **command window** is the main area for entering and running commands.
- **Later** we will learn to use the MATLAB's editors (and other tools/GUIs) to also enter commands and make functions *etc.*



Entering Commands

- The MATLAB command prompt: >>
- At this prompt you can enter commands to:
  - Create or modify a variable: *E.g.* `A = 3`
  - Perform a computation: *E.g.* `1 + 2`
  - Call a function: *E.g.* `max([1 2 3])`
  - Get help: *E.g.* `help`, `help max`
  - Manage workspace: *E.g.* `who`
  - Save workspace variables: *E.g.* `save Afile A`
  - and some other things.

Scalars, variables and basic arithmetic

- Variables are declared at any time in your code.
- Simply use = operator *E.g.* `A=3`
- MATLAB has no notion of data types — a variable can be scalar one minute and an array or structure at another instance. Maybe this is what is required otherwise **be careful with change of type**
- MATLAB is much like any other language for performing basic arithmetic
- MATLAB can perform arithmetic directly at the command line: *E.g.* `1 + 2`
- **Strings** are declared using `''`: *E.g.* `S = 'string'`

Returning computation results

- MATLAB can return a computation to a variable: *E.g.*

```
>> B = A + 3
B =
     6
```

- MATLAB can perform arithmetic directly at the command line: *E.g.* `1 + 2`

```
>>1 + 2
ans =
        3
```

No **permanent** variable is assigned.
But the temporary variable `ans` is returned.
(This is an important basic notion of MATLAB)

Semicolon terminated commands

- Semicolon typically ends statements in MATLAB. Strictly speaking the return or newline ends the statement (forces evaluation)
- If semicolon is omitted then the result of the computation is echoed to the command window:
  - Ideal for code development and debugging
  - Annoying and time consuming in larger bodies of code!

```
>> 6+5
ans =
    11
>> 6+5;
>> ans;
>> ans
ans =
    11
```

```
>> A = 3
A =
     3
>> A = 3;
>> B = A + 3;
>> B
B =
     6
```

Matrices or Arrays

MATLAB works with essentially only one kind of object — **a rectangular numerical matrix** with possibly complex entries:

- All variables represent matrices.
- In some situations, 1-by-1 matrices are interpreted as scalars and matrices with only one row or one column are interpreted as vectors.

Entering Matrices

Matrices can be introduced into MATLAB in several different ways:

- Entered by an explicit list of elements,
- Generated by built-in statements and functions,
- Created in a text file with your local editor,
- Loaded from external data files or applications (**not dealt with here — see the User's Guide).**

Entering Matrices (cont.)

For example, either of the statements

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

and

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

creates the obvious 3-by-3 matrix and assigns it to a variable A.

Entering Matrices (cont.)

The elements within a row of a matrix may be separated by commas as well as a blank.

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

Entering Larger Matrices

- Best done in an ASCII file with your local editor, where errors can be easily corrected
- File should consist of a rectangular array of just the numeric matrix entries. If this file is named, say, `data.ext` (where `.ext` is any extension except `.mat`),
- The MATLAB command `load data.ext` will read this file to the variable *data* in your MATLAB workspace. *E.g.*

`A.dat:`

```
1 2 3
4 5 6
7 8 9
```

```
>> load A.dat
>> A
A =
     1     2     3
     4     5     6
     7     8     9
```

Built-in Matrix Creation Functions

Example functions that create simple matrices:

- `rand(n)` will create an $n \times n$ matrix with randomly generated entries distributed uniformly between 0 and 1, while `rand(m,n)` will create an $m \times n$ one.
- `magic(n)` will create an integral $n \times n$ matrix which is a magic square (rows, columns, and diagonals have common sum).
- `eye(n)`, `eye(m,n)` will create a square n or an $m \times n$ **Identity** matrix.
- `ones(n)`, `ones(m,n)` will create a square n or an $m \times n$ matrix of **ones**.
- `zeros(n)`, `zeros(m,n)` will create a square n or an $m \times n$ matrix of **zeros**.

Built-in Matrix Creation Functions Examples

```
>> eye(3,2)
ans =
     1     0
     0     1
     0     0
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
>> ones(3)
ans =
     1     1     1
     1     1     1
     1     1     1
>> zeros(3)
ans =
     0     0     0
     0     0     0
     0     0     0
>> ones(2,3)
ans =
     1     1     1
     1     1     1
>> rand(2)
ans =
    0.9501    0.6068
    0.2311    0.4860
>> magic(3)
ans =
     8     1     6
     3     5     7
     4     9     2
```

Multimedia Data as Vectors, Matrices, Arrays etc.

As we shall see in coming lectures, basic media data is simply represented as a matrix or array in MATLAB:

- Audio — 1-D array or vector of amplitudes of sound wave
- Image — 2-D array or matrix of colour intensities
- Video — 3-D array or matrix: Each frame a 2-D image, n frames per second

Audio Data as Vector

Reading Audio : MATLAB can natively read Wav and AU files via `wavread()` and auread(). *E.g.*
`[y,Fs] = wavread('handel.wav');`
A vector `y` is returned along with the sample rate, `Fs` of the input file.

Processing : Now we have the audio in a 1-D array or vector we can do things to it — we leave this for later.

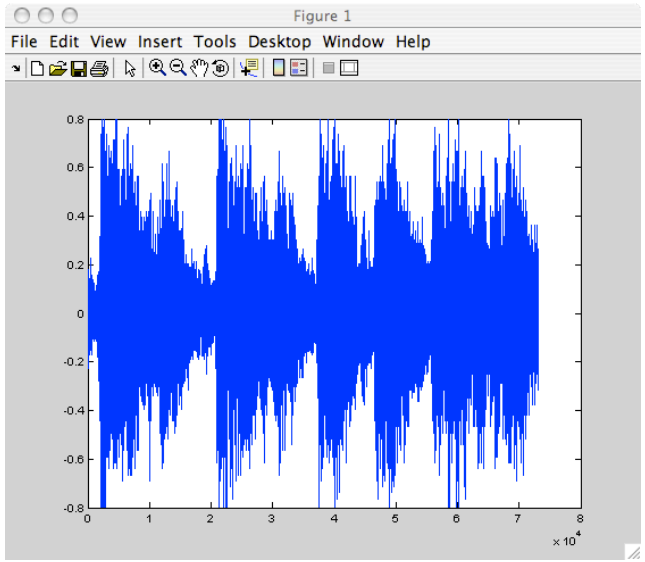
Display the waveform : More on graphics later but a simple `plot(y)` displays the audio data.

Playing the sound : Again more later. But a simple `audioplayer` and `play()` the sound will do for now.
Alternatively, you may use `sound(y, Fs)` or `soundsc(y, Fs)` to play the sound.

Audio Data as Vector (cont.)

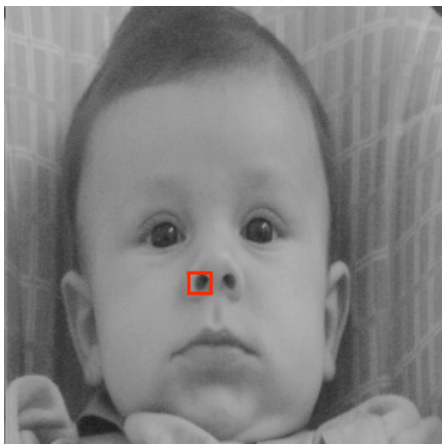
Output the data to a command window : `y` stores the data as a **long** MATLAB 1-D array. So we can simply type: `>>y` to look at the numbers (not that meaningful best to `plot()`?)

```
>> [y,Fs] = wavread('handel.wav'); % read in wav file
>> plot(y); % plot in figure
>> p = audioplayer(y, Fs); % create audioplayer
>> play(p, [1 n]); % play audio
>> y  % list  elements
ans =
         0
   -0.0062
   -0.0750
   -0.0312
    0.0062
    0.0381
    0.0189
   -0.0250
   -0.0312
   ............
```



Images as Matrices

- Images (uncompressed) are represented as a grid of pixels (intensities for greyscale images).



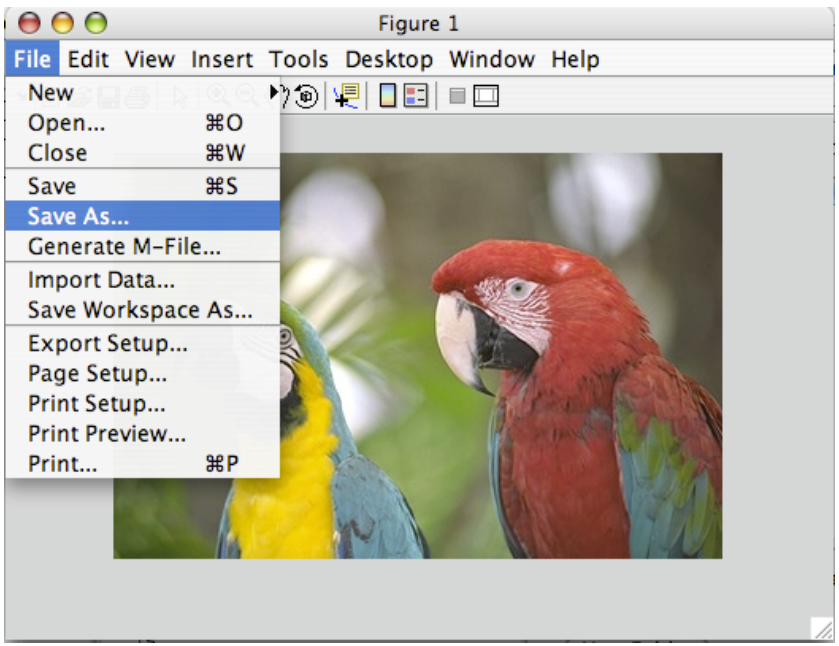
99	71	61	51	49	40	35	53	86	99
93	74	53	56	48	46	48	72	85	102
101	69	57	53	54	52	64	82	88	101
107	82	64	63	59	60	81	90	93	100
114	93	76	69	72	85	94	99	95	99
117	108	94	92	97	101	100	108	105	99
116	114	109	106	105	108	108	102	107	110
115	113	109	114	111	111	113	108	111	115
110	113	111	109	106	108	110	115	120	122
103	107	106	108	109	114	120	124	124	132

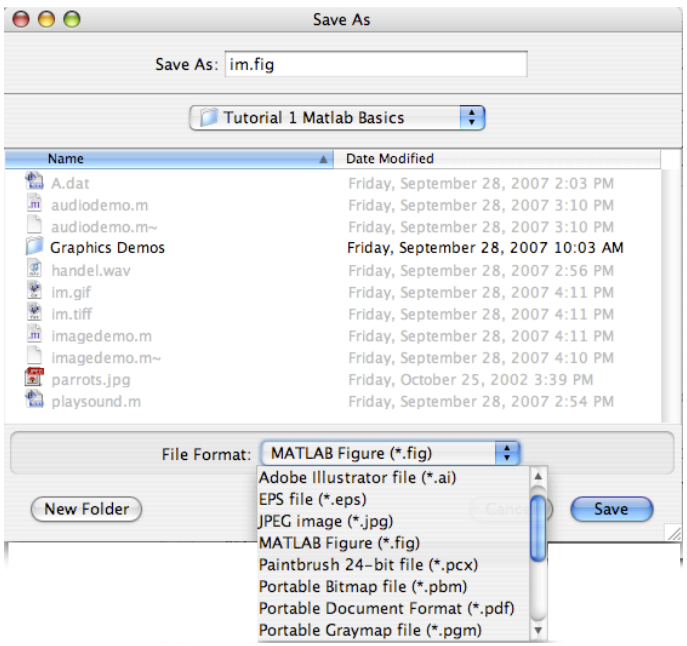
Reading/Writing Images in MATLAB

- `imread('filename')` reads a file and assigns it to an array variable:
  `im = imread('parrots.jpg');`
- MATLAB can read many file formats including: JPEG, GIFF, TIFF, BMP, PNG — **see** `help imread`
- This is a colour image so it has 3 images planes (RGB) — **more in lectures** Its actually a 3-D array in MATLAB.
- `size()` is useful to get the dimensions of the image:
  `[l m n] = size(im);`

Reading/Writing Images in MATLAB (cont.)

- `whos(var)` is also useful — it lists a given `var`iable in long form.
- `imwrite()` write an image to a specified file format.
- Images (indeed any graphics) may also be saved directly from the MATLAB figure window.

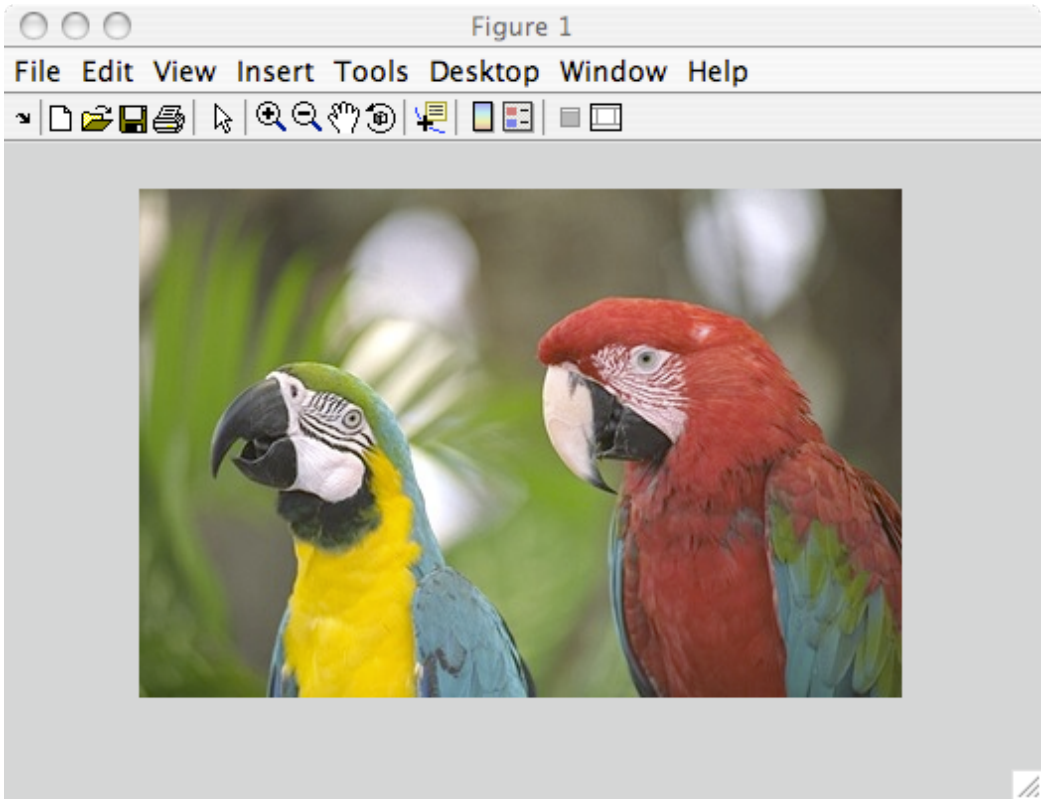




Reading/Writing Images in MATLAB

- `imread('filename')` reads a file and assigns it to an array variable:

  ```
  imshow(im);
  ```

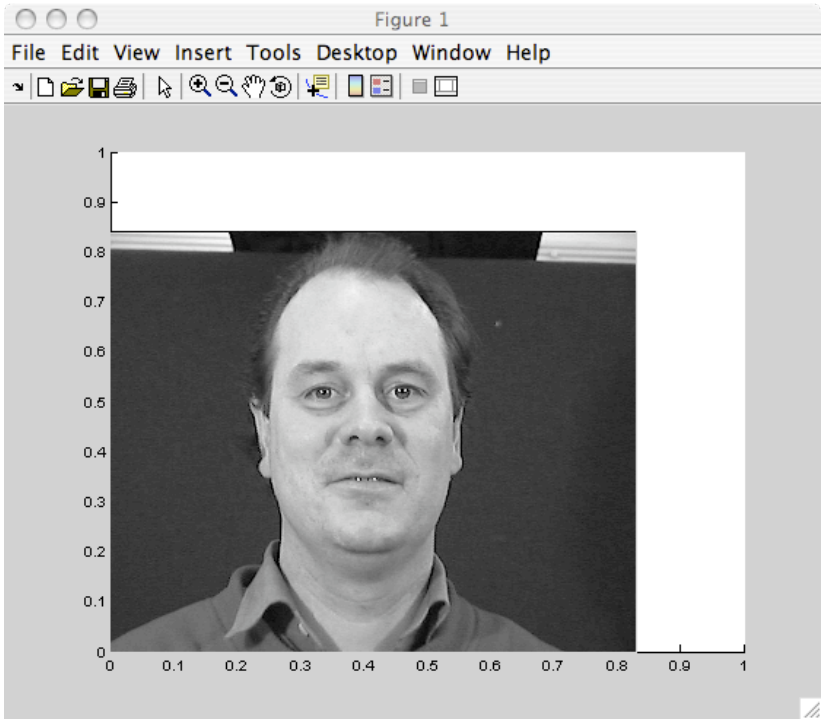


MATLAB Image Code Example

```
%read image
>>im = imread('parrots.jpg');

% get image size (note 3 colours)
>>[l m n] = size(im)
l =
   256
m =
   384
n =
     3

% list variable in long form
>>whos im
  Name        Size                Bytes  Class
  im        256x384x3           294912  uint8 array
Grand total is 294912 elements using 294912 bytes

% output as tiff image
>>imwrite(im,'im.tiff','tiff');
```

Videos as Matrices in MATLAB

- MATLAB has a `movie` structure: it is a 2D array of image frames over time (frame rate)
- `movie(movie_array)` will play the video in a figure.



- As with images and audio `size()` and `whos` are useful commands.

Reading Video using `VideoReader`

- `VideoReader` is a powerful tool to deal with video in MATLAB. It supports various formats, e.g. **AVI(.avi)**, **MPEG-1(.mpg)**, **Windows Media Video(.wmv, .asf, .asx)** on Windows, and **MPEG-4(.mp4, .m4v)** and **QuickTime Movie(.mov)** on Mac. Details of the video can also be obtained.
- Similarly use `VideoWriter` to create new videos.
- An example:

```
vr = VideoReader('origdave.avi');
```

Create a `VideoReader` object.

```
vidFrames = read(vr);
```

Read in all the frames. For true-colour video, `vidFrames` is a four dimensional array: Width $\times$ Height $\times$ Channels (3) $\times$ NumberOfFrames

Reading Video using `VideoReader` (cont.)

```
numFrames = get(vr, 'NumberOfFrames');
```

Obtain the number of frames.

This is equivalent to: `numFrames = vr.NumberOfFrames;`

```
for k = 1:numFrames
    mov(k).cdata = vidFrames(:, :, :, k);
    mov(k).colormap = [];
end
```

Construct a `movie` struct to hold the video data.

```
movie(mov, 1, get(vr, 'FrameRate'));
```

Play the movie. Here `get(vr, 'FrameRate')` obtains the frame rate (number of frames per second).

Indexing Matrices/Arrays

- Individual matrix and vector entries can be referenced with indices inside parentheses in the usual manner.
- For example, `A(2,3)` denotes the entry in the second row, third column of matrix `A`
- Another Example, `x(3)` denotes the third coordinate of vector `x`.
- A matrix or a vector will only accept *positive* integers as indices.

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

```
>> X = [1 2 3 4 5 6 7 8]
X =
  1  2  3  4  5  6  7  8
>> X(3)
ans =
     3
```

Matrix Operations

The following matrix operations are available in MATLAB:

$+$	addition
$-$	subtraction
$*$	multiplication
$\wedge$	power
$'$	conjugate transpose
$.'$	transpose
$\backslash$	left division
$/$	right division

These matrix operations apply to **scalars** (1-by-1 matrices) as well.

Matrix Operator Rules

Matrix Operators have to obey basic Math laws:

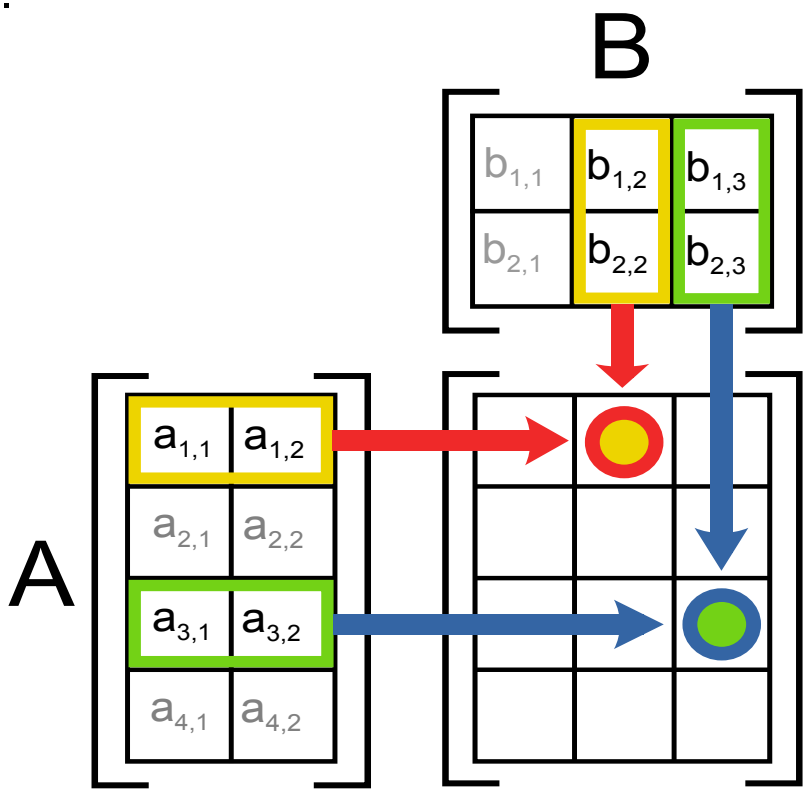
- If the sizes of the matrices are incompatible for the matrix operation, an error message will result:
  - For addition and subtraction matrices **must** have same dimension.
  - For multiplication **must** obey matrix product rule.
- **Exception**: The case of **scalar-matrix** operations (for addition, subtraction, and division as well as for multiplication) in which case each entry of the matrix is operated on by the scalar.

Note Array Operations are different — more soon

Matrix Multiplication Recap

According to mathematical definition, matrices $A_{m \times n}$ and $B_{p \times q}$ can be multiplied **if and only if** $n = p$, and the resulting matrix $C = A \cdot B$ is a matrix of size $m \times q$.

`A * B` and `B * A` are usually different (even if both are valid), i.e. *non-commutative*.



Matrix Division

- If A is an invertible square matrix and b is a compatible column, resp. row, vector, then
  $x = A\backslash b$ is the solution of $A * x = b$ and, resp.,
- $x = b/A$ is the solution of $x * A = b$.

Matrix Division Operators have to obey basic laws of linear algebra for solution of equations etc. these are not important here though.

Example: If A and B are invertible (full-rank) square matrices of the same size, and `C = A * B`. Then,

- `A\C` equals `inv(A) * C` and returns exactly B.
- `C/B` equals `C * inv(B)` and returns exactly A.

Array Operators

Note Array Operations are different to Matrix Operators

- There is a subtle change in syntax but big difference in result of a matrix *v.* array operator
- Array operators work in an element-by-element or entry-wise way.
- Matrix multiplication, division and power **do not**.
- To make array operator precede *similar* matrix operator with **.**

Matrix v Array Operators Examples

```
>> A
A =
     1     2     3
     4     5     6
     7     8     9
>> B = ones(3)
B =
     1     1     1
     1     1     1
     1     1     1
```

```
>> A*B
ans =
     6     6     6
    15    15    15
    24    24    24
>> A.*B
ans =
     1     2     3
     4     5     6
     7     8     9
```

Summary: Matrix v Array Operators

Matrix Operators	Array Operators	Operation
+	+	addition
−	−	subtraction
*	.*	multiplication
^	.^	power
/	./	right division
\	.\	left division
'		conjugate transpose
.'		transpose

Submatrices: Colon Notation

Submatrices (and Vectors of Matrices (= single rows or columns)) are often used in MATLAB to achieve fairly complex data manipulation effects.

- To appreciate the *usefulness* of these features, compare these MATLAB statements with a Java, FORTRAN, or C routine to effect the same.

Essential to grasp this principle: extensively used later in course

Colon Notation Advantages

Good MATLAB practice:

- Colon notation (and subscripting by integral vectors) are keys to *efficient manipulation of these objects*.
- Creative use of these features to vectorize operations permits one to minimize the use of loops etc.
  - Use of Loops to access matrices etc. slows MATLAB
- (Once syntax assimilated) Makes code simpler to read/understand.

Special effort should be made to become familiar with this.

Colon Notation: Accessing blocks of Matrix/Array Elements

The basic colon expression

`m:n` creates a sequence of values from `m` to `n`.

- It actually creates a row vector $m \ldots n$
- Can also be used in `for` statements — **more later**

Simple Example `1:5` is actually the row vector `[1 2 3 4 5]`.

```
>> 1:5
ans =
     1     2     3     4     5
```

Two Colons

The numbers need not be **integers** *nor* the **increment one**.

The syntax `m:step:n` generates values from $m \ldots n$ with given `step` size.

For example,

`0.2:0.2:1.2` gives

```
[0.2, 0.4, 0.6, 0.8, 1.0, 1.2],
```

`5:-1:1` gives

```
[5 4 3 2 1].
```

Accessing Submatrices

The colon notation can be used to access submatrices of a matrix. For example:

`A(1:4,3)` is the column vector consisting of the first four entries of the third column of A.

A colon by itself denotes an entire row or column:

`A(:,3)` is the third column of A, and

`A(1:4,:)` is the first four rows.

`end` can be used to save remembering the `end` of a vector or row/column or a matrix.

If x is an vector, what is the effect of the statement
`x = x(end:-1:1)`?

Arbitrary Integral Vectors

Arbitrary integral vectors can be used as subscripts:

`A(:,[2 4])` contains as columns, columns 2 and 4 of A.

Such subscripting can be used on both sides of an assignment statement:

`A(:,[2 4 5]) = B(:,1:3)` replaces columns 2,4,5 of A with the first three columns of B. Note:

- the *entire* altered matrix A is printed and assigned.
- (sub)matrices/arrays/vectors **must** be the same dimension for such operations to work.

`A(:,[2,4]) = A(:,[2,4])*[1 2;3 4]` Columns 2 and 4 of A can be multiplied on the right by the 2-by-2 matrix [1 2;3 4]

Further Examples

`A(:, 3:5) = []` removes columns 3 to 5 from A.

- only the *entire* row or column of matrix A can be removed.

`A(:, 3:5) = 10` sets every value to 10 in columns 3 to 5 from A.

It is possible to compose larger matrix from smaller submatrices. Assume `A1` and `A2` are two row vectors of the same length, `A = [A1; A2]` constructs a new matrix by putting `A1` and `A2` together. An example to flip/negate an image:

```
>> im=imread('parrots.jpg'); imshow(im);
>> size(im)
ans =
   256   384     3
>> im = im(:, end:-1:1, :); imshow(im);
>> im = 255 - im; imshow(im);
>> imwrite(im, 'test.tiff', 'tiff');
```

Summary

- Introduction to MATLAB concepts
- Basic MATLAB programming
- Handling multimedia (audio/image/video) using MATLAB
- Basic MATLAB operators