MATLAB

Lecture 2

Making Folders

- Use folders to keep your programs organized
- To make a new folder, click the ‘Browse’ button next to ‘Current Directory’
- Click the ‘Make New Folder’ button, and change the name of the folder. Do NOT use spaces in folder names.
- Highlight the folder you just made and click ‘OK’
- The current directory is now the folder you just created
- To see programs outside the current directory, they should be in the Path. Use File-> Set Path to add folders to the path

MATLAB Basics

- MATLAB can be thought of as a super-powerful graphing calculator

- In addition it is a programming language
  - MATLAB is an interpreted language, like Java
  - Commands executed line by line

Help/Docs

- help
  - The most important function for learning MATLAB on your own

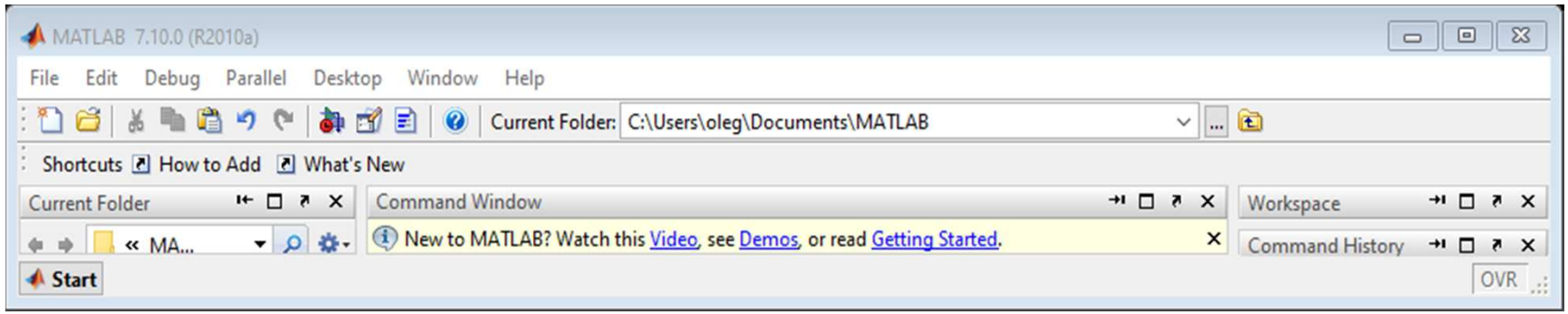
- To get info on how to use a function:
  - »help sin

  Help lists related functions at the bottom and links to the doc

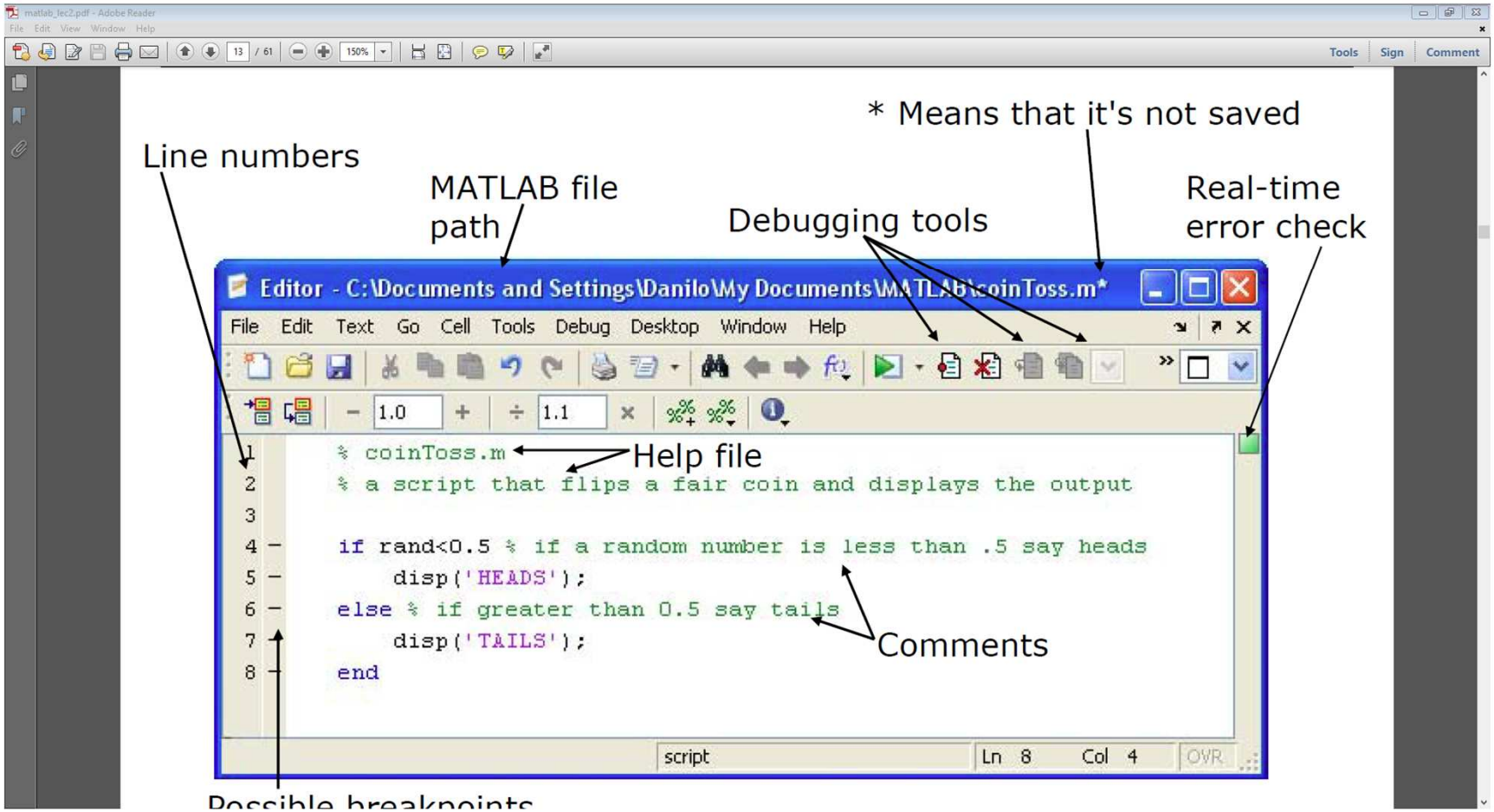
- To get a nicer version of help with examples and easy-to read descriptions:
  - »doc sin

Scripts: Overview

- Scripts are
  - collection of commands executed in sequence
  - written in the MATLAB editor
  - saved as MATLAB files (.m extension)
- To create an MATLAB file from command-line
  - »edit helloWorld.m
  - or click “page” button under “File”



Scripts: the Editor



Scripts: Some Notes

- COMMENT!
  - Anything following a %is seen as a comment
  - The first contiguous comment becomes the script's help file
  - Comment thoroughly to avoid wasting time later
- Note that scripts are somewhat static, since there is no input and no explicit output
- All variables created and modified in a script exist in the workspace even after it has stopped running

Scripts

- Make a *helloWorld* script
- Open the editor and save a script as helloWorld.m. This is an easy script, containing two lines of code:
- % helloWorld.m
- % my first hello world program in MATLAB
- disp('Hello World!');
- disp('I am going to learn MATLAB!');
- Command **disp** displays strings. Strings are written between single quotes, like 'This is a string'

Variables

- No need to initialize variables!
- MATLAB supports various types, the most often used are
- »3.84 64-bit double (default)
- »‘a’ 16-bit char
- Most variables you’ll deal with will be vectors or matrices of doubles or chars
- Other types are also supported: complex, symbolic, 16-bit and 8 bit integers, etc. You will be exposed to all these types through the homework

Naming variables

- **To create a variable, simply assign a value to a name:**
  - »var1=3.14
  - »myString='hello world'
- **Variable names**
  - first character must be a LETTER
  - after that, any combination of letters, numbers and _
  - CASE SENSITIVE! (var1 is different from Var1)

- **Built-in variables. Don't use these names!**
  - I and j can be used to indicate complex numbers
  - pi has the value 3.1415926…
  - Ans stores the last unassigned value (like on a calculator)
  - Inf and –Inf are positive and negative infinity
  - NaN represents 'Not a Number'

Variables

- A variable can be given a value explicitly
  - »a = 10
  - shows up in workspace!
- Or as a function of explicit values and existing variables
  - »c = 1.3*45-2*a
- To suppress output, end the line with a semicolon
  - »cooldude = 13/3;

Arrays

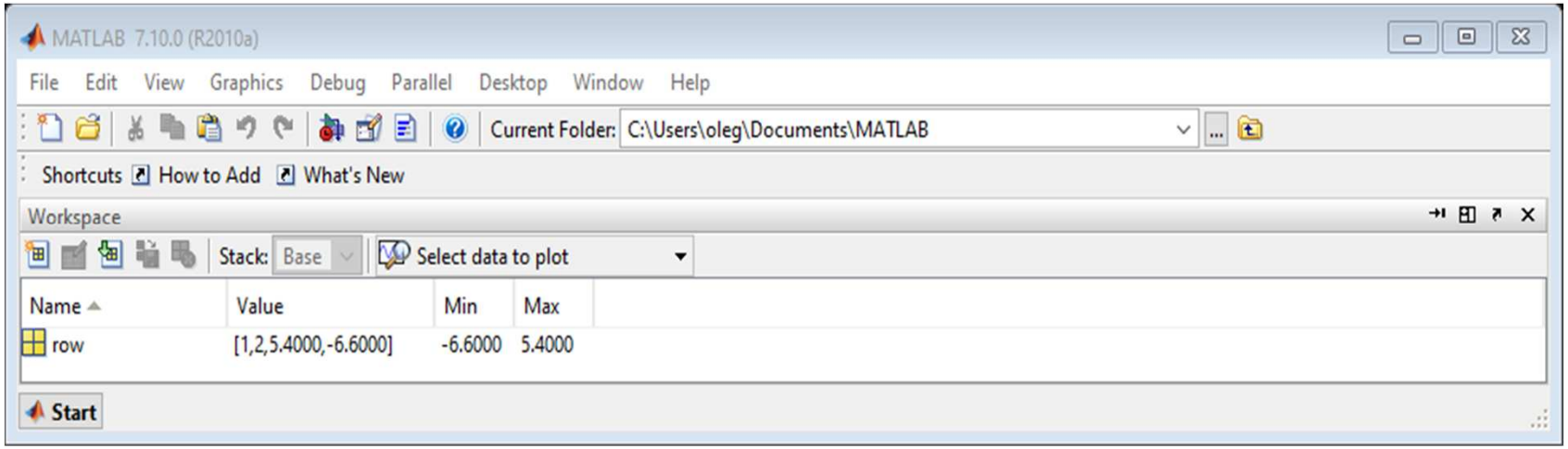
- Like other programming languages, arrays are an important part of MATLAB
- Two types of arrays
  - (1)matrix of numbers (either double or complex)
  - (2)cell array of objects (more advanced data structure)

Row Vectors

- Row vector: comma or space separated values between brackets
  - »row = [1 2 5.4 -6.6]
  - »row = [1, 2, 5.4, -6.6];
- Command window:
  - row =

    1.0000 2.0000 5.4000 -6.6000

Workspace:

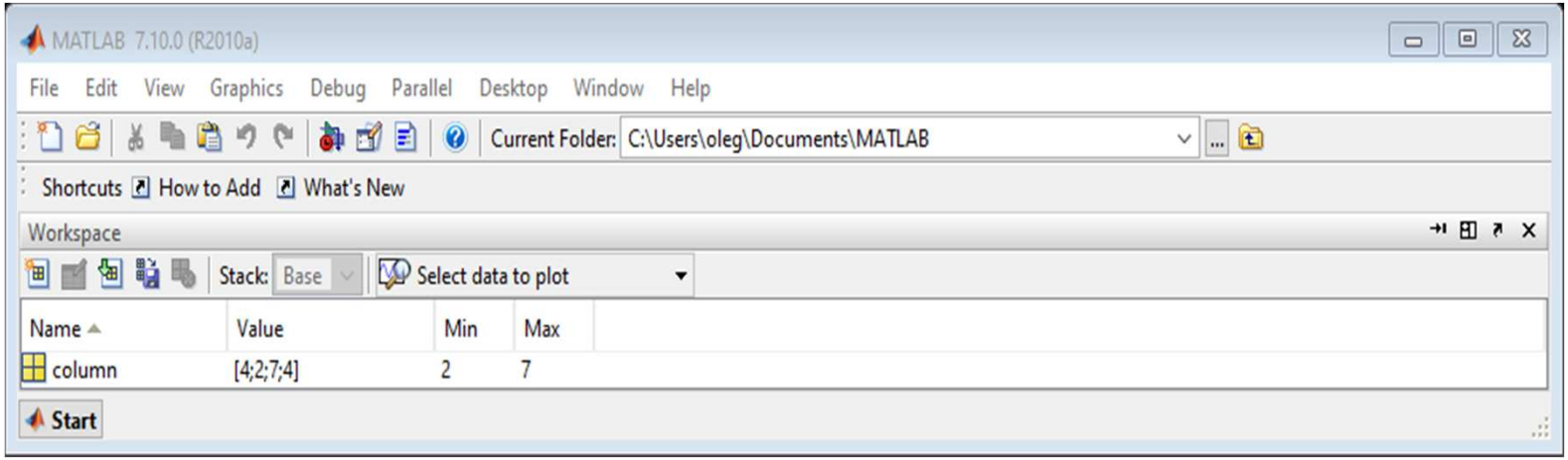


Column Vectors

- Column vector: semicolon separated values between brackets
  - »column = [4;2;7;4]
- Command window

```
» column =
»      4
»      2
»      7
»      4
```

- Workspace



size & length

- You can tell the difference between a row and a column vector by:
  - Looking in the workspace
  - Displaying the variable in the command window
  - Using the size function

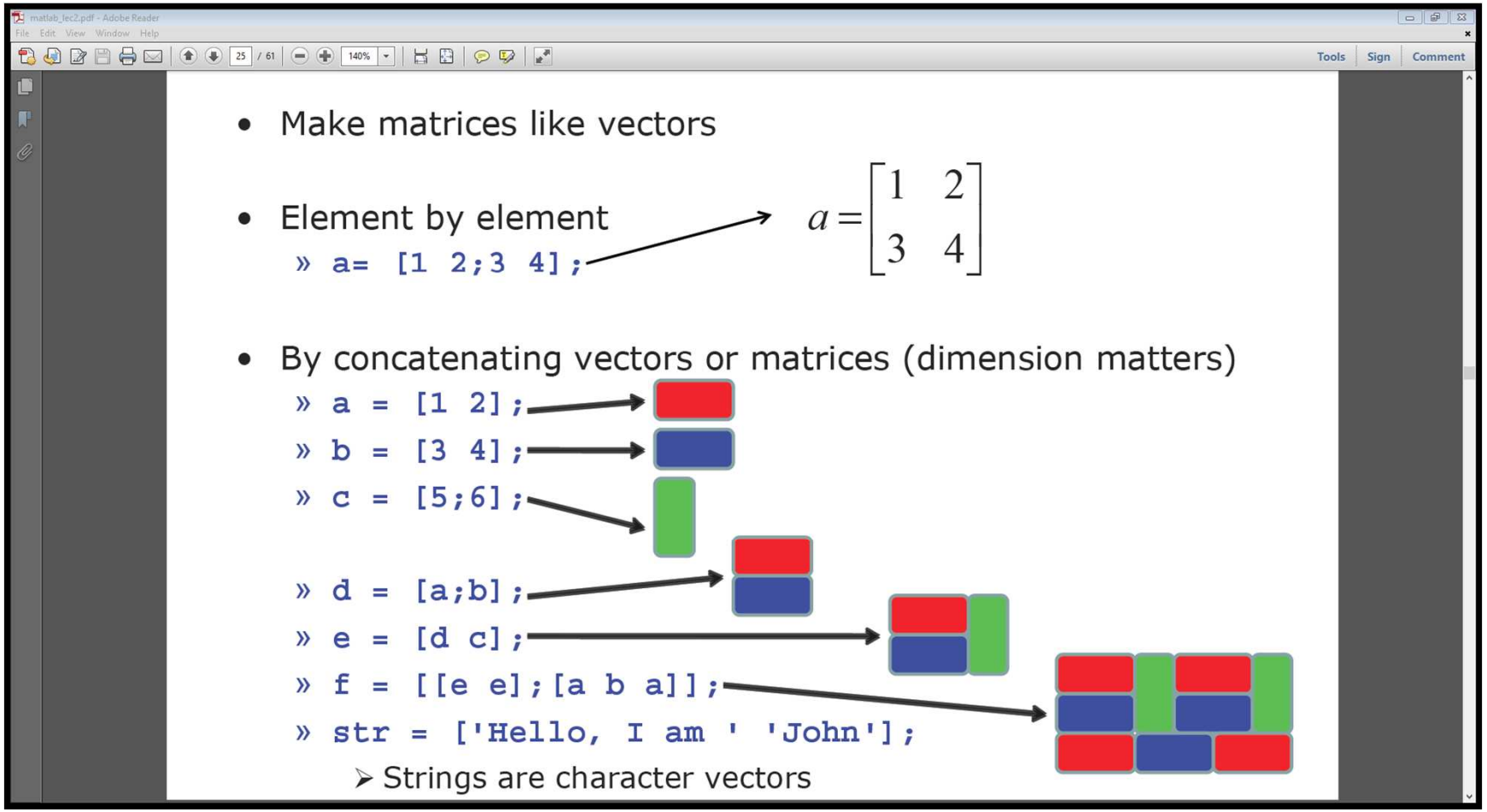
```
size(row)
ans =
  1    4
```

```
size(column)
ans =
  1    4
```

To get a vector's length, use the length function

```
length(row)
ans =
   4
```

Matrices


Tools
Sign
Comment
• Make matrices like vectors
• Element by element
» a= [1 2;3 4];
$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
• By concatenating vectors or matrices (dimension matters)
» a = [1 2];
» b = [3 4];
» c = [5;6];
» d = [a;b];
» e = [d c];
» f = [[e e];[a b a]];
» str = ['Hello, I am ' 'John'];
➢ Strings are character vectors

save/clear/load

- Use save to save variables to a file
  - »save myFile a b
    - saves variables a and b to the file myfile.mat
    - myfile.mat file is saved in the current directory
- Use clear to remove variables from environment
  - »clear a b
    - look at workspace, the variables a and b are gone
- Use load to load variable bindings into the environment
  - »load myFile
    - look at workspace, the variables a and b are back
- Can do the same for entire environment
  - »save myenv; clear all; load myenv;

Exercise: Variables

- Get and save the current date and time
  - Create a variable start using the function clock
  - What is the size of start? Is it a row or column?
  - What does start contain? See help clock
  - Convert the vector start to a string. Use the function datestr and name the new variable startString
  - Save start and startString into a mat file named startTime

Basic Scalar Operations

- Arithmetic operations (+,-,*,/)
  - »7/45
  - »(1+i)*(2+i)
  - »1 / 0
  - »0 / 0
- •Exponentiation (^)
  - »4^2
  - »(3+4*j)^2
- •Complicated expressions, use parentheses
  - »((2+3)*3)^0.1
- •Multiplication is NOT implicit given parentheses
  - »3(1+0.7) gives an error
- •To clear command window
  - »clc

Built-in Functions

- MATLAB has an enormous library of built-in functions
- Call using parentheses –passing parameter to function
  - »sqrt(2)
  - »log(2), log10(0.23)
  - »cos(1.2), atan(-.8)
  - »exp(2+4*i)
  - »round(1.4), floor(3.3), ceil(4.23)
  - »angle(i); abs(1+i);

Transpose

- The transpose operators turns a column vector into a row vector and vice versa
  - »a = [1 2 3 4+i]
  - »transpose(a)
  - »a'
  - »a.'
- The ‘ gives the Hermitian-transpose, i.e. transposes and conjugates all complex numbers
- For vectors of real numbers .'and 'give same result

Addition and Subtraction

- Addition and subtraction are element-wise; sizes must match (unless one is a scalar):

$$\begin{bmatrix}12 & 3 & 32 & -11\end{bmatrix} + \begin{bmatrix}2 & 11 & -30 & 32\end{bmatrix} = \begin{bmatrix}14 & 14 & 2 & 21\end{bmatrix}$$

$$\begin{bmatrix}12\\1\\-10\\0\end{bmatrix} - \begin{bmatrix}3\\-1\\13\\33\end{bmatrix} = \begin{bmatrix}9\\2\\-23\\-33\end{bmatrix}$$

- •The following would give an error
  - c = row + column
- Use the transpose to make sizes compatible
  - c = row'+ column
  - c = row + column'
- Can sum up or multiply elements of vector
  - s=sum(row);
  - p=prod(row);

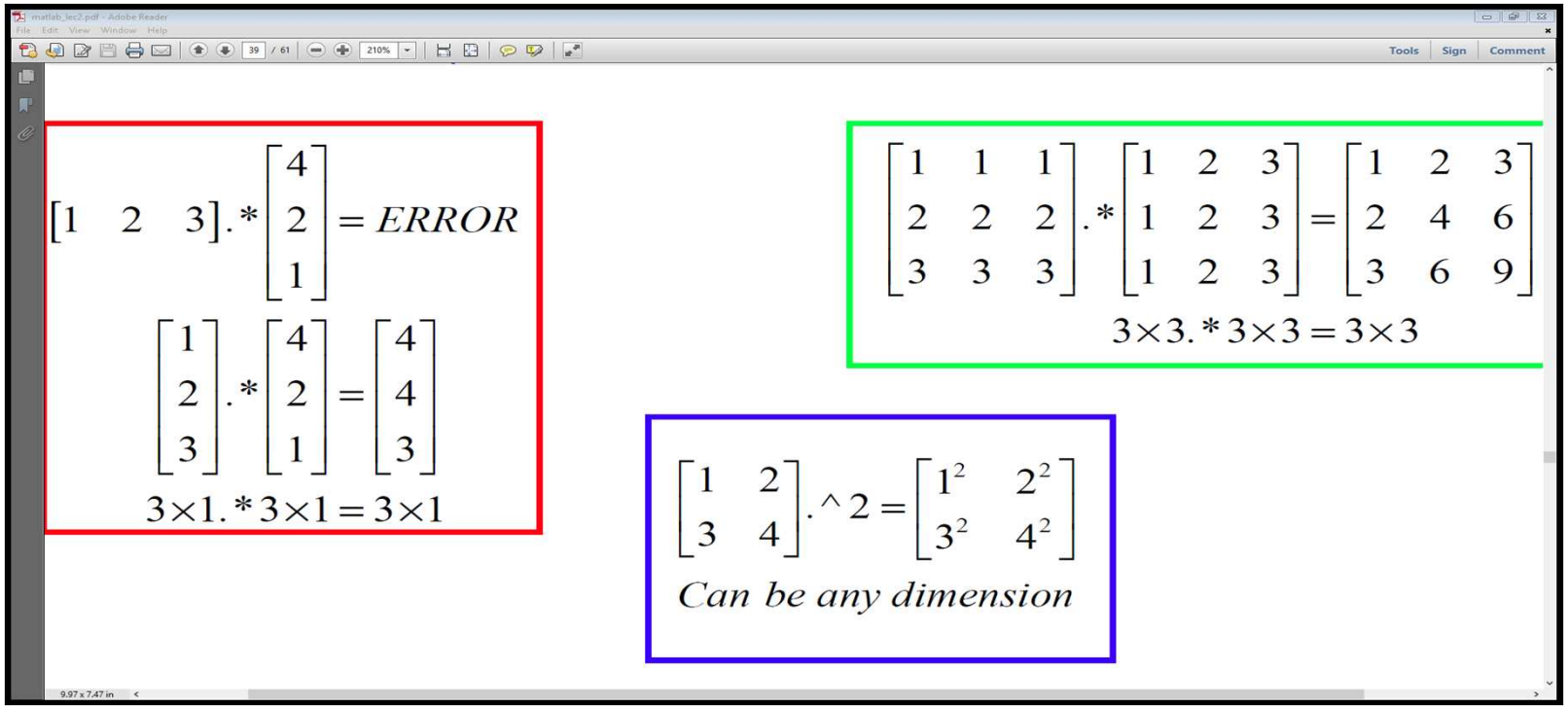
Element-Wise Functions

- All the functions that work on scalars also work on vectors
  - t = [1 2 3];
  - f = exp(t);
- is the same as
  - f = [exp(1) exp(2) exp(3)];
- If in doubt, check a function's help file to see if it handles vectors elementwise
- Operators (* / ^) have two modes of operation
  - element-wise

    standard

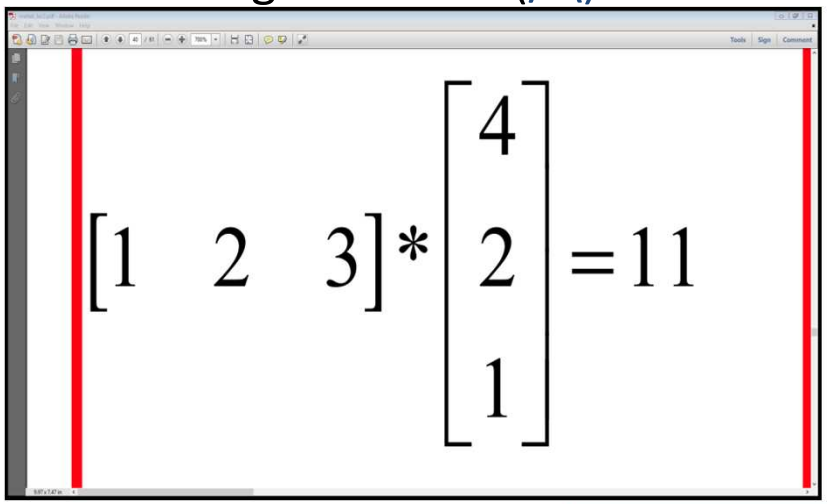
Operators: element-wise

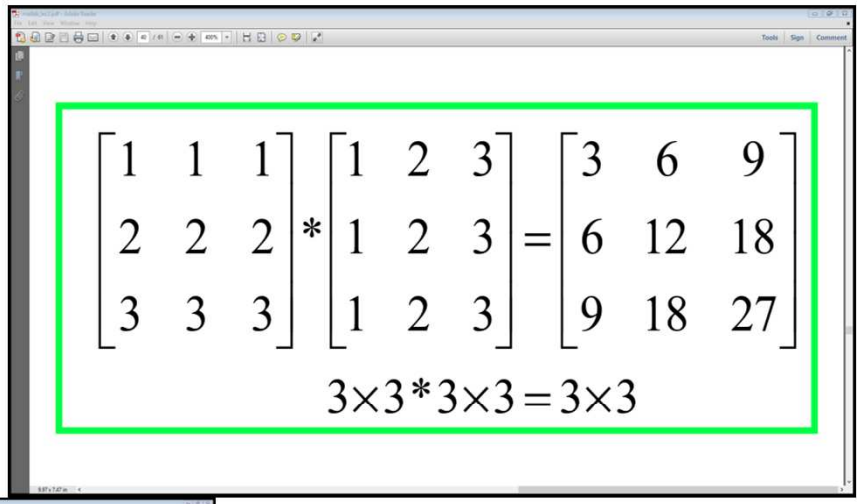
- To do element-wise operations, use the dot: .(.*, ./, .^). BOTH dimensions must match (unless one is scalar)!
  - a=[1 2 3];b=[4;2;1];
  - a.*b, a./b, a.^b all errors
  - a.*b', a./b', a.^(b') all valid

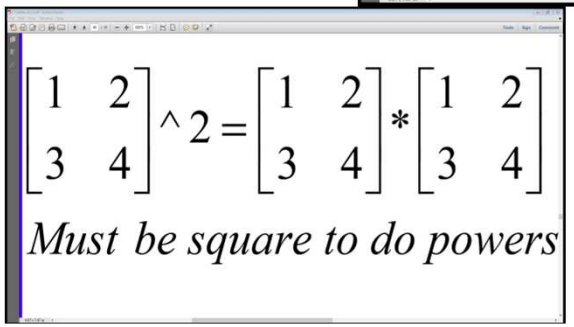


Operators: standard

- Multiplication can be done in a standard way or element-wise
- Standard multiplication (*) is either a dot-product or an outer product
- ¾Remember from linear algebra: inner dimensions must MATCH!!
- Standard exponentiation (^) can only be done on square matrices or scalars
- Left and right division (/ \) is same as multiplying by inverse







Automatic Initialization

- Initialize a vector of ones, zeros, or random numbers
  - o=ones(1,10)
- row vector with 10 elements, all 1
  - z=zeros(23,1)
- column vector with 23 elements, all 0
  - r=rand(1,45)
  - row vector with 45 elements (uniform [0,1])
  - n=nan(1,69)
- row vector of NaNs (useful for representing uninitialized variables)

- var=zeros(M,N);

- Number of rows Number of columns

Automatic Initialization

- To initialize a linear vector of values use linspace

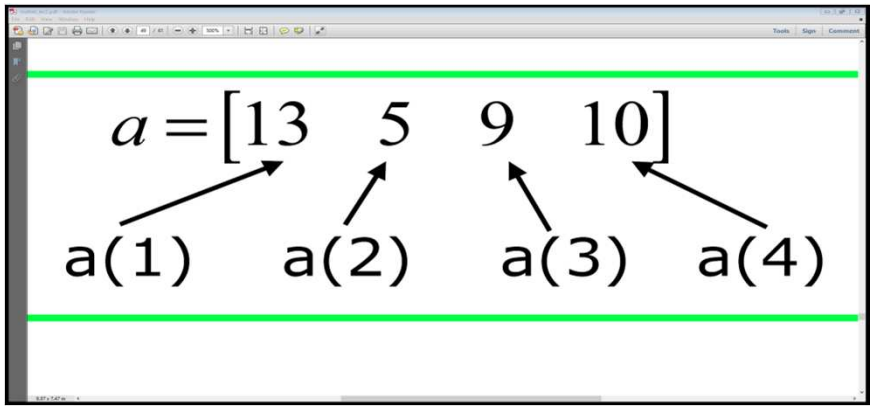
- a=linspace(0,10,5)
  - starts at 0, ends at 10 (inclusive), 5 values

- Can also use colon operator (:)
- b=0:2:10
  - starts at 0, increments by 2, and ends at or before 10
  - increment can be decimal or negative
- c=1:5
  - if increment isn't specified, default is 1

- To initialize logarithmically spaced values use logspace similar to linspace, but see help

Vector Indexing

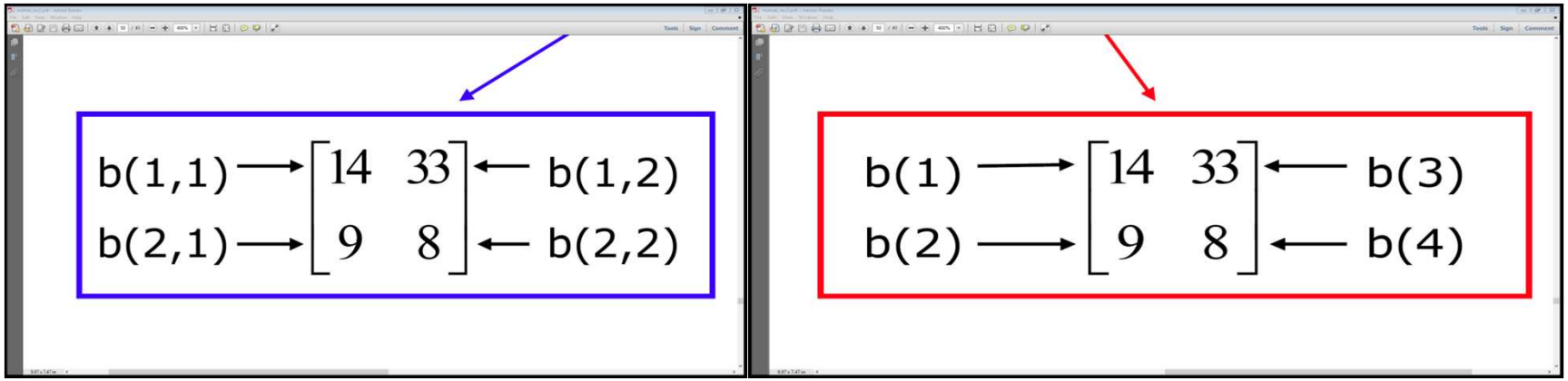
- MATLAB indexing starts with 1, not 0
- a(n) returns the n(th) element



- The index argument can be a vector. In this case, each element is looked up individually, and returned as a vector of the same size as the index vector.
- x=[12 13 5 8];
- a=x(2:3); ⟶ a=[13 5];
- b=x(1:end-1); ⟶ b=[12 13 5];

Matrix Indexing

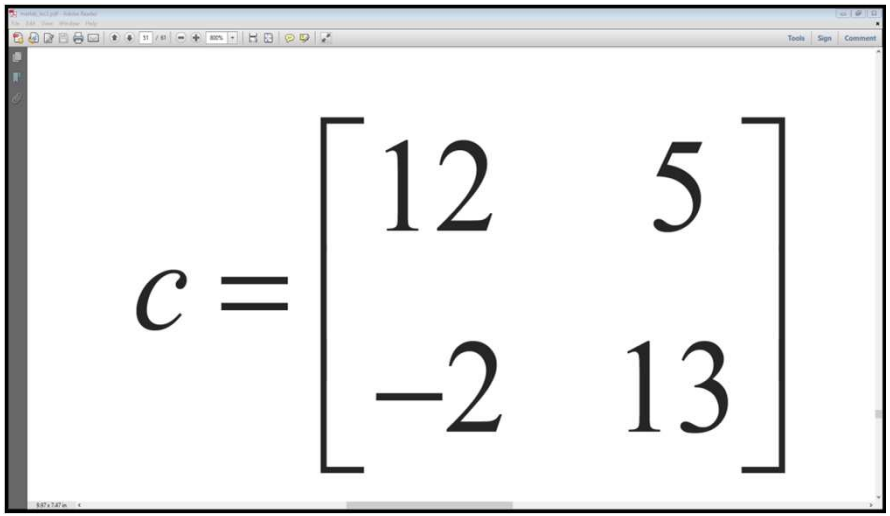
- •Matrices can be indexed in two ways
  - using subscripts(row and column)
  - using linear indices(as if matrix is a vector)
- Matrix indexing: subscripts or linear indices



- Picking submatrices
- A = rand(5) % shorthand for 5x5 matrix
- A(1:3,1:2) % specify contiguous submatrix
- A([1 5 3], [1 4]) % specify rows and columns

Advanced Indexing

- To select rows or columns of a matrix, use the :



d=c(1,:); ⟶ d=[12 5];

e=c(:,2); ⟶ e=[5;13];

c(2,:)=[3 6]; %replaces second row of c

Advanced Indexing

- MATLAB contains functions to help you find desired values within a vector or matrix

```
vec = [5 3 1 9 7]
```

- To get the minimum value and its index:

```
[minVal,minInd] = min(vec);
```

max works the same way

- To find any the indices of specific values or ranges

```
ind = find(vec == 9);
ind = find(vec > 2 & vec < 6);
```

- find expressions can be very complex, more on this later

Plotting

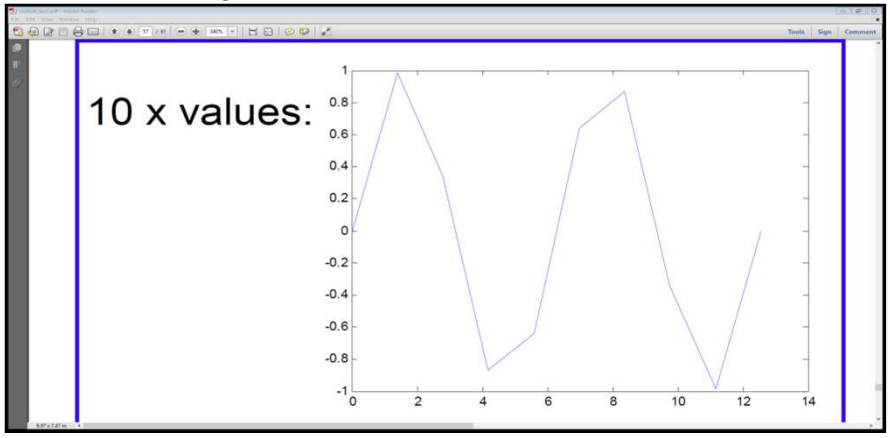
- Example
  - x=linspace(0,4*pi,10);
  - y=sin(x);
- •Plot values against their index
  - plot(y);
- Usually we want to plot y versus x
  - plot(x,y);

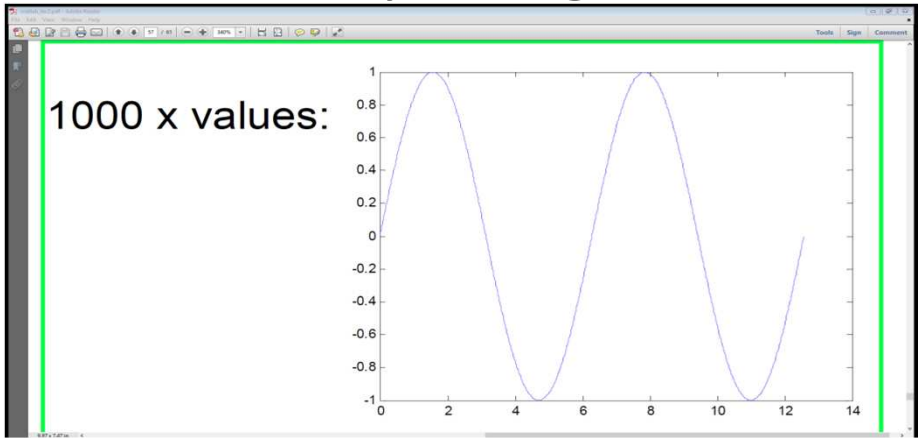
Plotting

- What does plot do?
- plot generates dots at each (x,y) pair and then connects the dots with a line
- To make plot of a function look smoother, evaluate at more points

```
x=linspace(0,4*pi,1000);
plot(x,sin(x));
```

- x and y vectors must be same size or else you'll get an error





Functions

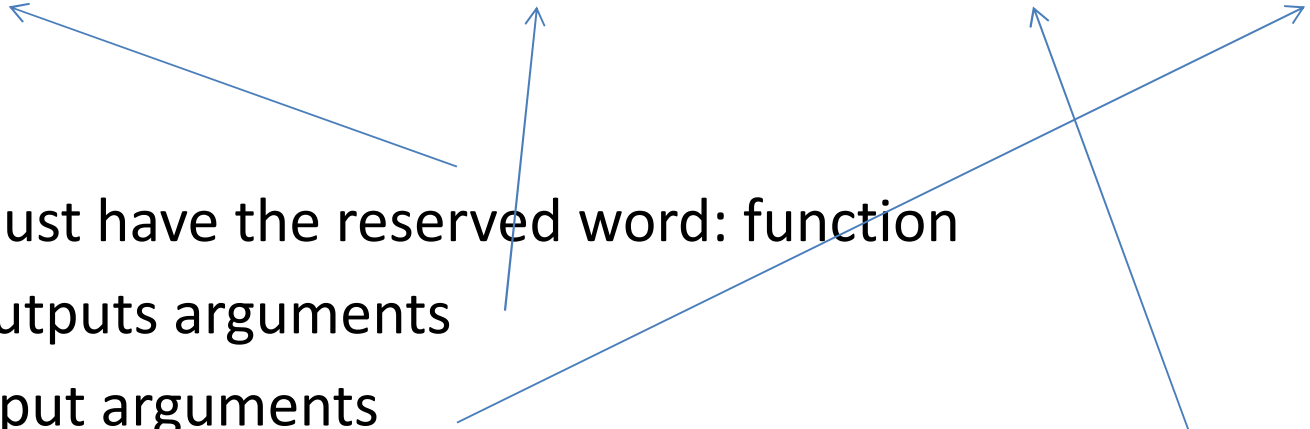
- Functions look exactly like scripts, but for ONE difference
  - Functions must have a function declaration

- We are using functions and scripts different way

Functions

- function declaration

- **function [output1, output2, output3,] = funName(input1, input2)**



Must have the reserved word: function

Outputs arguments

Input arguments

Function name should match MATLAB file name

Functions

- No need for return: MATLAB 'returns' the variables whose names match those in the function declaration

- Variable scope: Any variables created within the function but not returned disappear from memory after the function stops running

Functions examples

- Open new script and type :
  - function [output] = xsq(input)
  - output = input.^2;
- try saving it and the suggested name will be xsq.m. Save it as suggested.
- We can not run this file as a script!
- In a command window or in another(script) file type
  - x = 1:10;
  - y = xsq(x)
- This is a way how to use functions!

Functions examples

- Example with vector output

```
function [output] = func(x,y)
output = x.^2 + y.^2;
```

- Use it:

```
x= 0.0:pi/10:pi;y = x;
[X,Y] = meshgrid(x,y);
f = func(X,Y); contour(X,Y,f);
axis([0 pi 0 pi]); axis equal;
```

Functions examples

- Example with two outputs

```
function [sq,cub] = xpowers(input)
sq = input.^2; cub = input.^3;
```

- use

```
x = 1:10; [xsq,xcub] = xpowers(x);
```

- When the function is called we must know what form of output we expect!

Functions

- MATLAB functions are generally overloaded

- Can take a variable number of inputs
- Can return a variable number of outputs
- What would the following commands return:
  - **a=zeros(2,4,8); %n-dimensional matrices are OK**
  - **D=size(a)**
  - **[m,n]=size(a)**
  - **[x,y,z]=size(a)**
  - **m2=size(a,2)**

Functions

- All variables are local, unless they are declared global. If you use variable x in the main program, then it is not automatically passed into function

Functions

- Passing Parameters into Functions as variables
  - function [output] = myfunc(x,p)
  - output = p*x^2;
- Use it:
  - X= 0.0:pi/10:pi;P = 2;
  - F = myfunc(X,P); plot(X,F);
- Parameters as global variables
  - function [output] = myfunc(x)
  - global p q;
  - output = (x - p)*(x-q);
- use
  - global p q;
  - ...
  - z = fzero(@myfunc,x0);