6.057
Introduction to MATLAB

Orhan Celiker, IAP 2019

Course Layout

Problem sets

- One per day, should take about 4 hours to complete
- Submit Word or PDF, include code and figures
- Some questions optional, but highly recommended!

Requirements for passing

- Attend 3/4 lectures (Friday is optional)
- Complete all problem sets (graded on a 3-level scale: -, √, +)...
- ... and achieve √ average

Prerequisites: You'll be fine!

MATLAB Basics

- MATLAB can be thought of as a super-powerful graphing calculator
  - Remember the TI-83 from calculus?
  - With many more buttons (built-in functions)

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

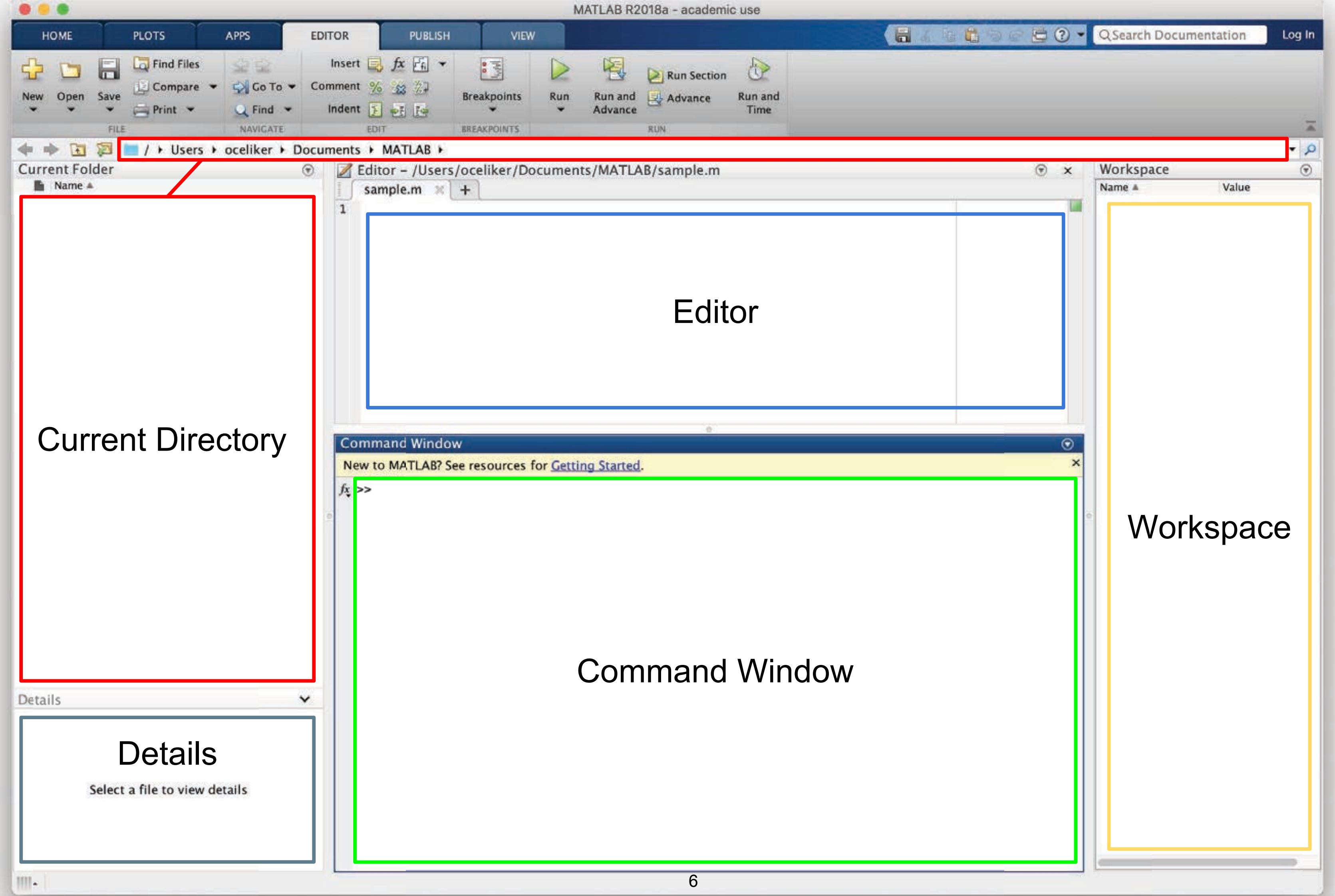
Outline

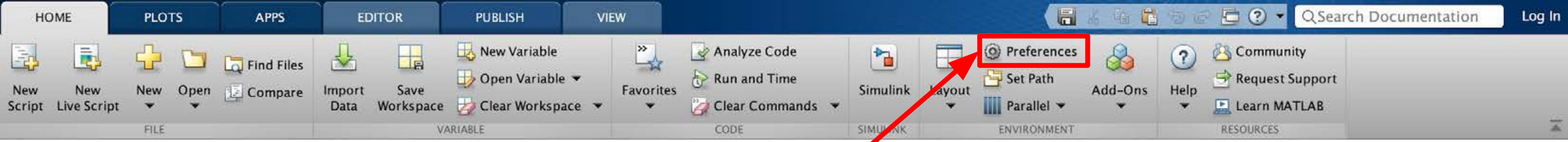
I. Getting Started
II. Scripts
III. Making Variables
IV. Manipulating Variables
V. Basic Plotting

Getting Started

- To get MATLAB Student Version for yourself

- You can also use MATLAB online
    - https://matlab.mathworks.com (requires Mathworks account with license)

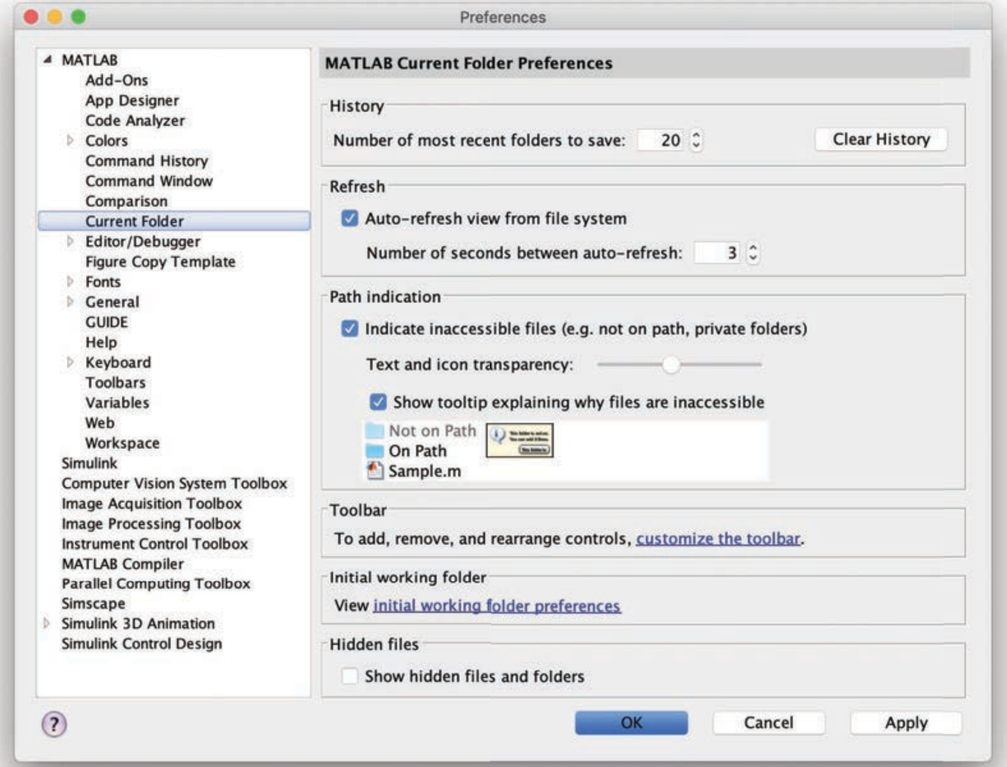

MATLAB R2018a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Log In
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
/ Users oceliker Documents MATLAB
Current Folder
Name
Current Directory
Details
Details
Select a file to view details
Editor - /Users/oceliker/Documents/MATLAB/sample.m
sample.m
Editor
Command Window
New to MATLAB? See resources for Getting Started.
>>
Command Window
Workspace
Name
Value
Workspace

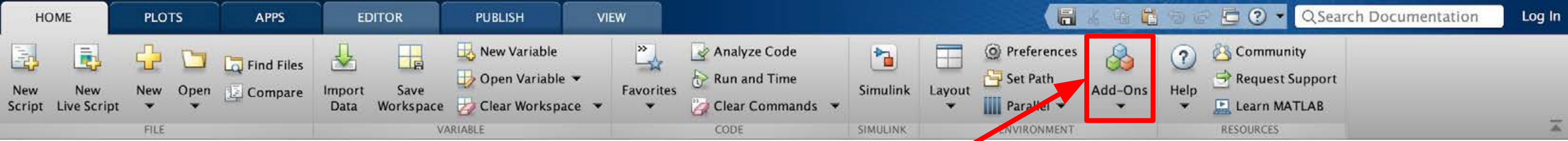


Customization

- In the top ribbon, navigate to:
  Home -> Environment -> Preferences

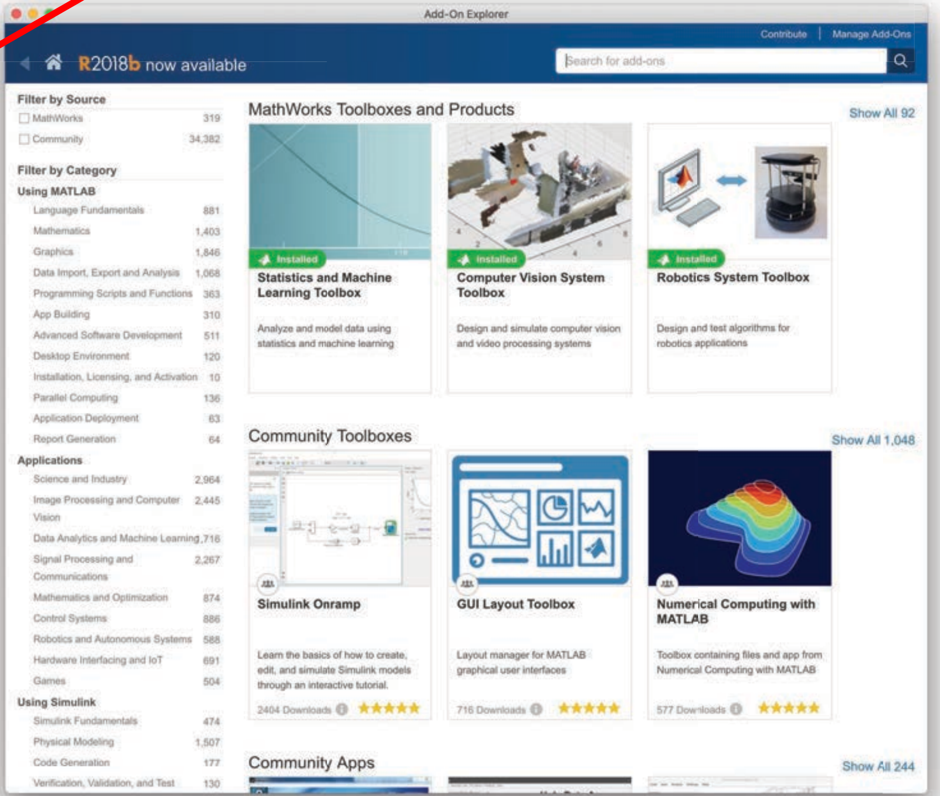
- Allows you to customize your MATLAB experience (colors, fonts, etc.)





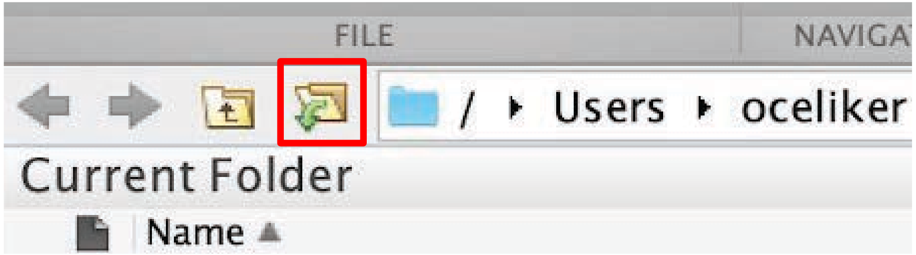
Installing Toolboxes

- In the top ribbon, navigate to: Home -> Environment -> Add-Ons
- Allows you to install toolboxes included with your license
- Recommended toolboxes:
  - Curve Fitting Toolbox
  - Computer Vision System Toolbox
  - Image Processing Toolbox
  - Optimization Toolbox
  - Signal Processing Toolbox
  - and anything related to your field!



Making Folders

- Use folders to keep your programs organized
- To make a new folder, click "Browse" next to the file path



- Click the Make New Folder button, and change the name of the folder. In the MATLAB folder (which should be open by default), make the following folder structure:

```
MATLAB
↳ IAP MATLAB
      ↳ Day1
```

Help/Docs

- `help`
  - The most important command 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 documentation
- To get a nicer version of help with examples and easy-to-read description:
  - `doc sin`
- To search for a function by specifying keywords:
  - `docsearch sin trigonometric`

Outline

I. Getting Started
II. Scripts
III. Making Variables
IV. Manipulating Variables
V. Basic Plotting

Scripts: Overview

- Scripts are
    - Collection of commands executed in sequence
    - Written in the MATLAB editor
    - Saved as m-files (.m extension)
- To create an m-file from the command line:
    - `edit MyFileName.m`
    - or click the "New Script" button on the top left

Scripts: Some notes

- COMMENT!
  - Anything following a % sign is interpreted as a comment
  - The first contiguous comment becomes the script's help file
  - Comment thoroughly to avoid wasting time later!
  - Mark beginning of a code block by using %%
- Note that scripts are somewhat static, with no explicit input and output
- All variables created or modified in a script retain their values after script execution

Exercise: Scripts

- Make a script with the name **helloWorld.m**
- When run, the script should show the following text:

```
Hello world!
I am going to learn MATLAB!
```

Hint: Use `disp(...)` to display strings. Strings are written between single quotes, e.g. `'This is a string'`

Outline

I. Getting Started
II. Scripts
III. <u>Making Variables</u>
IV. Manipulating Variables
V. Basic Plotting

Variable Types

- MATLAB is a "weakly typed" language
  - No need to initialize variables!
- MATLAB supports various types; the most popular ones are
  - 3.84
    - 64-bit double (default)
  - 'A'
    - 16-bit char
- Most variables you'll deal with are vectors, matrices, doubles or chars
- Other types are also supported: complex, symbolic, 16-bit and 8-bit integers (uint16 & uint8), etc.

Naming Variables

- To create a variable, simply assign a value to a name:

```
myNumberVariable = 3.14
myStringVariable = 'hello world!'
```

- Variable name rules
    - First character must be a LETTER
    - After that, any combination of numbers, letters and _
    - Names are CASE-SENSITIVE (e.g. `var1` is different than `Var1`)

Naming Variables (cont.)

Built-in variables (don't use these names for anything else!):

`i, j`: can be used to indicate complex numbers*

`pi`: has the value 3.1415...

`ans`: stores the result of the last unassigned value

`Inf, -Inf`: infinities

`NaN`: "Not a Number"

ops, use `ii, jj, kk`, etc. for loop counters.

Scalars

- 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:
  - Matrix of numbers (either double or complex)
  - Cell array of objects (more advanced data structure)

MATLAB makes vectors easy!
That's its power!

Row vectors

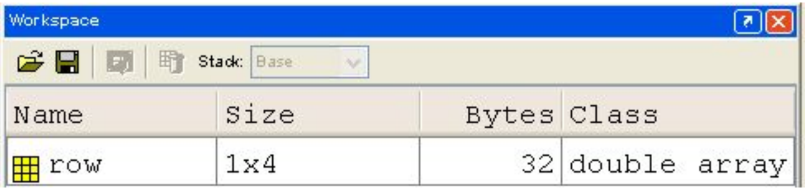
- Row vector: comma- or space-separated values between square brackets
  - `row = [ 1 2 3.2 4 6 5.4 ];`
  - `row = [ 1, 2, 4, 7, 4.3, 1.1 ];`
- Command window:

```
>> row=[1 2 5.4 -6.6]

row =

    1.0000    2.0000    5.4000   -6.6000
```

- Workspace:



Column vectors

- Column vector: semicolon-separated values between square brackets
  - `col = [ 1; 2; 3.2; 4; 6; 5.4 ];`

- Command window:

```
>> column=[4;2;7;4]

column =

     4
     2
     7
     4
```

- Workspace:

Workspace

Stack: Base

Name	Size	Bytes	Class
column	4x1	32	double array

Size and length

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

```
>> size(row)

ans =

     1     4

>> length(row)

ans =

     4
```

```
>> size(column)

ans =

     4     1

>> length(column)

ans =

     4
```

Matrices

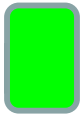
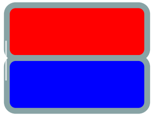
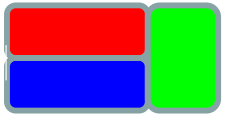
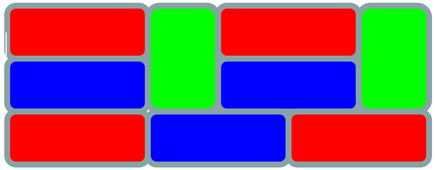
- Make matrices like vectors
  - Element by element
    - `a= [1 2;3 4];`

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

- 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 in the current directory
  - Default working directory is MATLAB unless you navigate to another folder
  - Make sure you are in the correct folder. Right now we should be in \MATLAB\IAP MATLAB\Day 1
- Use clear to save variables to a file
  - `clear a b`
  - Look at workspace: variables a and b are gone
- Use load to load variables into the workspace
  - `load myFile`
  - Look at workspace: a and b are back

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`

Exercise: Variables II

- In helloWorld.m, read in variables you saved using `load`
- Display the following text:

  I started learning MATLAB on [date, time]

- Hint: Use the `disp` command again
- Remember that strings are just vectors of characters, so you can join two strings by making a row vector with the two strings as sub-vectors.

Outline

I. Getting Started
II. Scripts
III. Making Variables
IV. <u>Manipulating Variables</u>
V. Basic Plotting

Basic Scalar Operations

- Arithmetic operations (+, -, *, /)
  - `7/45`
  - `(1+1i)*(1+2i)`
  - `1/0`
  - `0/0`
- Exponentiation
  - `4^2`
  - `(3+4*1j)^2`
- Complicated expressions: use parentheses
  - `((2+3)*3)^0.1`

Built-in Functions

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

Exercise: Scalars

`helloWorld` script:

- Your learning time constant is 1.5 days. Calculate the number of seconds in 1.5 days and name this variable `tau`
- This class lasts 5 days. Calculate the number of seconds in 5 days and name this variable `endOfClass`
- This equation describes your knowledge as a function of time t:

$$k = 1 - e^{-t/\tau}$$

- How well will you know MATLAB at `endOfClass`? Name this variable `knowledgeAtEnd` (use exp)
- Using the value of `knowledgeAtEnd`, display the phrase:

```
At the end of 6.057, I will know X% of MATLAB
```

Hint: to convert a number to a string, use `num2str`

Transpose

- The transpose operator 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
  - Transposes and conjugates all complex numbers
- For vectors of real numbers .' and ' give same result
  - For transposing a vector, always use .' to be safe

Addition and Subtraction

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

$$\begin{array}{r} [12 \quad 3 \quad 32 \quad -11] \\ +[2 \quad 11 \quad -30 \quad 32] \\ \hline =[14 \quad 14 \quad 2 \quad 21] \end{array}$$

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

Addition and Subtraction

- `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 element-wise
- Operators (* / ^) have two modes of operation
  - element-wise
  - standard

Element-wise functions

- 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

- Multiplication can be done in a standard way or element-wise
- Standard multiplication (*) is matrix product
    - Remember 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
    - Our recommendation: for now, just multiply by inverse (more on this later)

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} * \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} = 11$$

$$1 \times 3 * 3 \times 1 = 1 \times 1$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \wedge 2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$\textit{Must be square to do powers}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 9 \\ 6 & 12 & 18 \\ 9 & 18 & 27 \end{bmatrix}$$

$$3 \times 3 * 3 \times 3 = 3 \times 3$$

Exercise: Vector Operations

Calculate how many seconds elapsed since start of class

- In helloWorld.m, make variables called secPerMin, secPerHour, secPerDay, secPerMonth (assume 30.5 days per month), and secPerYear (12 months in year), which have the number of seconds in each time period
- Assemble a row vector called secondConversion that has elements in this order: secPerYear, secPerMonth, secPerDay, secPerHour, secPerMin, 1
- Make a currentTime vector by using clock
- Compute elapsedTime by subtracting currentTime from start
- Compute t (the elapsed time in seconds) by taking the dot product of secondConversion and elapsedTime (transpose one of them to get the dimensions right)

Exercise: Vector Operations

Display the current state of your knowledge

- Calculate currentKnowledge using the same relationship as before, and the t we just calculated:

$$k = 1 - e^{-t/\tau}$$

- Display the following text:
  At this time, I know X% of MATLAB

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 (representing uninitialized variables)

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 is not specified, default is 1

- To initialize logarithmically spaced values use **logspace**
    - ➢ Similar to **linspace**, but see **help**

Exercise: Vector Functions

Calculate your learning trajectory

- In helloWorld.m, make a linear time vector `tVec` that has 10,000 samples between 0 and `endOfClass`
- Calculate the value of your knowledge (call it `knowledgeVec`) at each of these time points using the same equation as before:

$$k = 1 - e^{-t/\tau}$$

Vector Indexing

- <u>MATLAB indexing starts with **1**, not **0**</u>
  - ➢ We will not respond to any emails where this is the problem.
- a(n) returns the n^{th} element

$$a = [13 \quad 5 \quad 9 \quad 10]$$

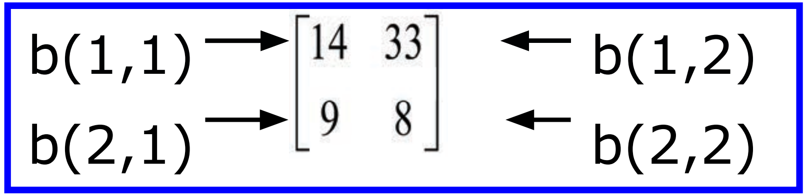
a(1) a(2) a(3) a(4)

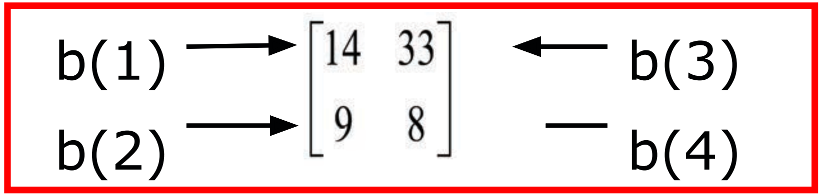
- The index argument can be a vector. In this case, each element is looked up individually, and returned as a vector of the same size as the index vector.


```
» x=[12 13 5 8];
```

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
```

Advanced Indexing 1

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

$$c = \begin{bmatrix} 12 & 5 \\ -2 & 13 \end{bmatrix}$$

```
» d=c(1,:);          d=[12 5];
» e=c(:,2);          e=[5;13];
» c(2,:)=[3 6];  %replaces second row of c
```

Advanced Indexing 2

- MATLAB contains functions to help you find desired values
  ```
  » vec = [5 3 1 9 7]
  ```

- To get the minimum value and its index (similar for `max`):
  ```
  » [minVal,minInd] = min(vec);
  ```

- To find the indices of specific values or ranges
  ```
  » ind = find(vec == 9); vec(ind) = 8;
  » ind = find(vec > 2 & vec < 6);
  ```
  - ➢ **find** expressions can be very complex, more on this later
  - ➢ When possible, **logical indexing** is faster than **find**!
  - ➢ E.g., `vec(vec == 9) = 8;`

Exercise: Indexing

When will you know 50% of MATLAB?

- First, find the index where **knowledgeVec** is closest to 0.5. Mathematically, what you want is the index where the value of ~ $\left|knowledgeVec - 0.5\right|$ is at a minimum (use **abs** and **min**)
- Next, use that index to look up the corresponding time in **tVec** and name this time **halfTime**
- Finally, display the string:
  Convert **halfTime** to days by using secPerDay. I will know half of MATLAB after X days

Outline

(1) Getting Started
(2) Scripts
(3) Making Variables
(4) Manipulating Variables
(5) **Basic Plotting**

Did everyone sign in?

Plotting

- Example
  - » `x=linspace(0,4*pi,10);`
  - » `y=sin(x);`

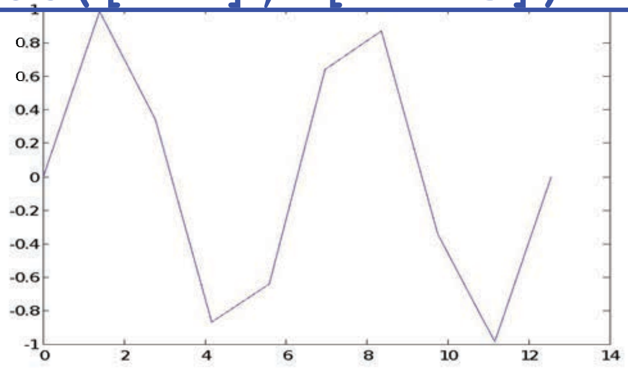
- Plot values against their index
  - » `plot(y);`
- Usually we want to plot y versus x
  - » `plot(x,y);`

MATLAB makes visualizing data fun and easy!

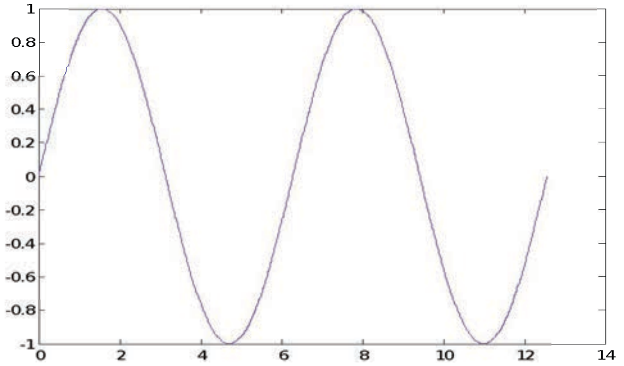
What does plot do?

- **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
  - » `plot([1 2], [1 2 3])`

10 x values:



1000 x values:



Exercise: Plotting

Plot the learning trajectory

- In helloWorld.m, open a new figure (use `figure`)
- Plot knowledge trajectory using `tVec` and `knowledgeVec`
- When plotting, convert `tVec` to days by using `secPerDay`
- Zoom in on the plot to verify that `halfTime` was calculated correctly

End of Lecture 1

(1) Getting Started
(2) Scripts
(3) Making Variables
(4) Manipulating Variables
(5)

Hope that wasn't too much and you enjoyed it!!

MIT OpenCourseWare
https://ocw.mit.edu

6.057 Introduction to MATLAB
IAP 2019