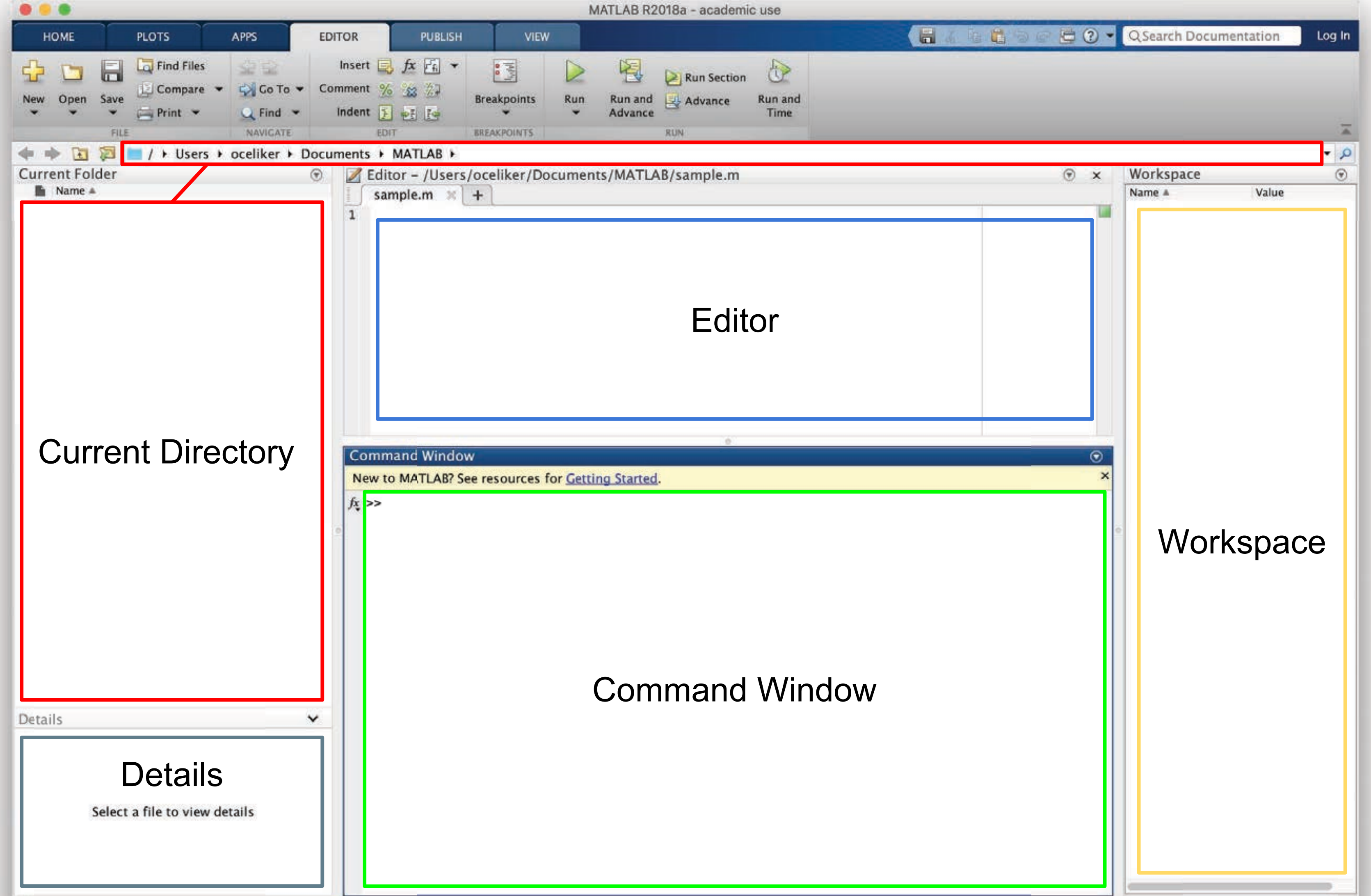
Introduction to MATLAB

Outline

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



/ ▸ Users ▸ oceliker ▸ Documents ▸ MATLAB ▸

Editor – /Users/oceliker/Documents/MATLAB/sample.m

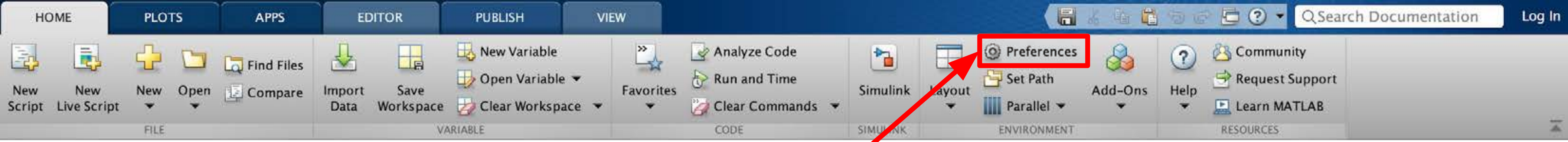
sample.m

New to MATLAB? See resources for Getting Started.

```
>>
```

Details

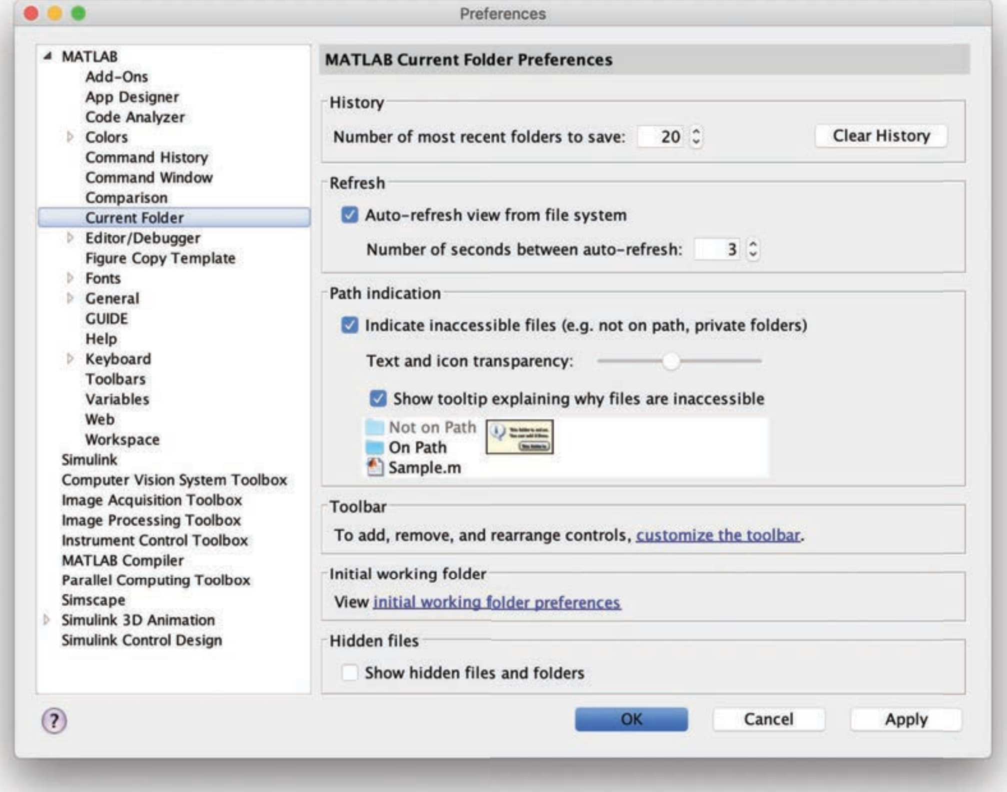
Select a file to view details



Customization

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

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



MATLAB Current Folder Preferences

History

Number of most recent folders to save: 20 Clear History

Refresh

☑ Auto-refresh view from file system

Number of seconds between auto-refresh: 3

Path indication

☑ Indicate inaccessible files (e.g. not on path, private folders)

Text and icon transparency:

☑ Show tooltip explaining why files are inaccessible

Toolbar

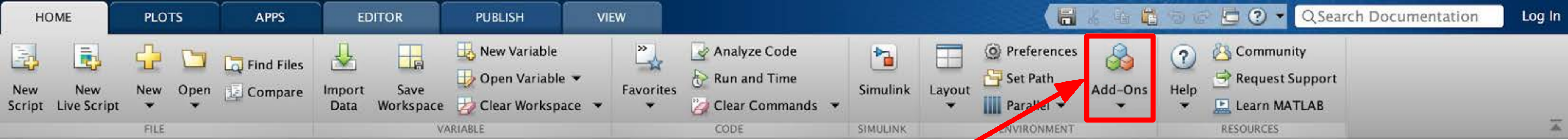
To add, remove, and rearrange controls, customize the toolbar.

Initial working folder

View initial working folder preferences

Hidden files

□ Show hidden files and folders

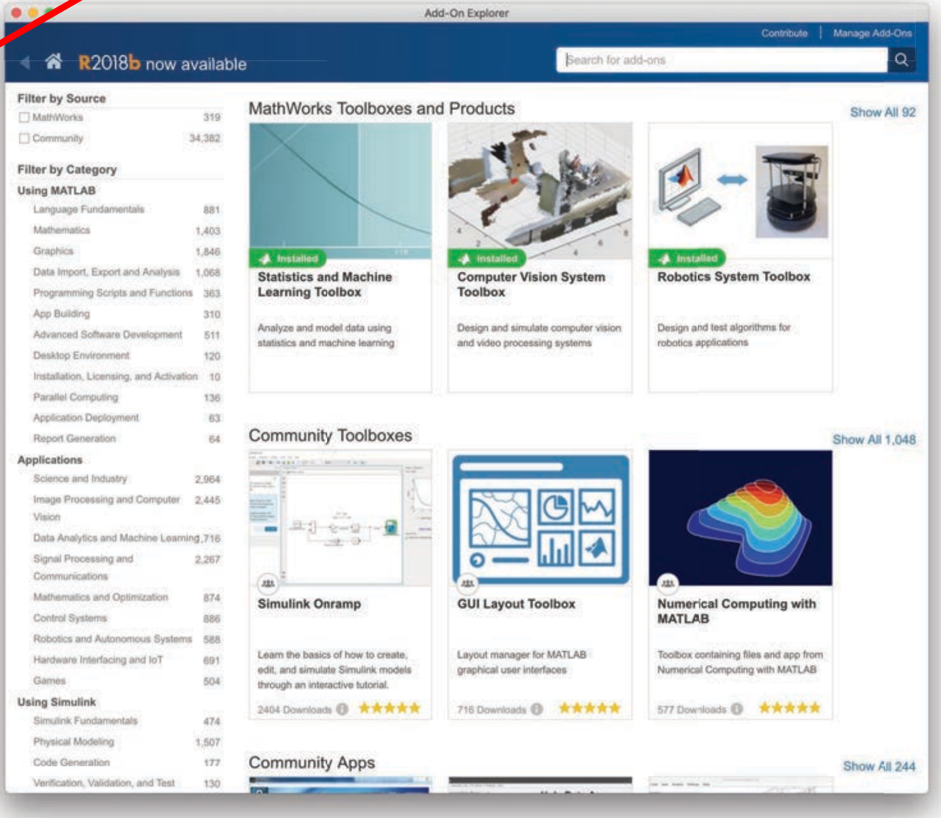


Installing Toolboxes

- 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!



Filter by Source

□ MathWorks	319
□ Community	34,382

Filter by Category

Using MATLAB	
Language Fundamentals	881
Mathematics	1,403
Graphics	1,846
Data Import, Export and Analysis	1,068
Programming Scripts and Functions	363
App Building	310
Advanced Software Development	511
Desktop Environment	120
Installation, Licensing, and Activation	10
Parallel Computing	136
Application Deployment	63
Report Generation	64
Applications	
Science and Industry	2,964
Image Processing and Computer Vision	2,445
Data Analytics and Machine Learning	,716
Signal Processing and Communications	2,267
Mathematics and Optimization	874
Control Systems	886
Robotics and Autonomous Systems	588
Hardware Interfacing and IoT	691
Games	504
Using Simulink	
Simulink Fundamentals	474
Physical Modeling	1,507
Code Generation	177
Verification, Validation, and Test	130

MathWorks Toolboxes and Products

Statistics and Machine Learning Toolbox
Analyze and model data using statistics and machine learning

Computer Vision System Toolbox
Design and simulate computer vision and video processing systems

Robotics System Toolbox
Design and test algorithms for robotics applications

Community Toolboxes

Simulink Onramp
Learn the basics of how to create, edit, and simulate Simulink models through an interactive tutorial.
2404 Downloads ★★★★★

GUI Layout Toolbox
Layout manager for MATLAB graphical user interfaces
716 Downloads ★★★★★

Numerical Computing with MATLAB
Toolbox containing files and app from Numerical Computing with MATLAB
577 Downloads ★★★★★

Community Apps

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

Outline

I. Getting Started
II. Scripts
III. Making Variables
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;`

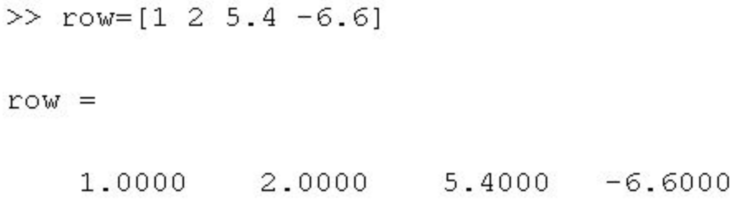
Arrays

- 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!

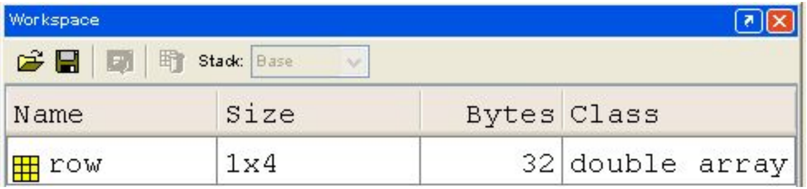
Row vectors

- 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:

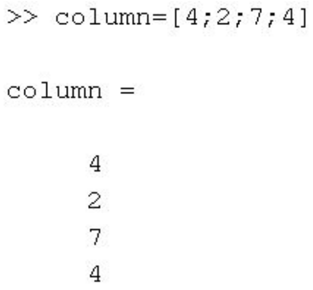


Name	Size	Bytes	Class
row	1x4	32	double array

Column vectors

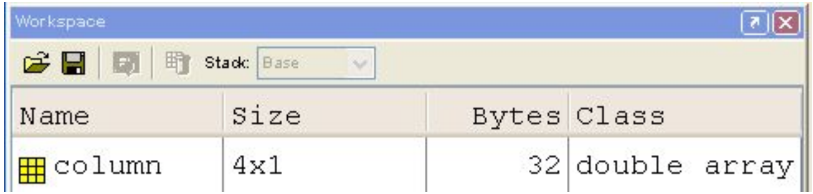
- Column vector: semicolon-separated values between square brackets
  - `col = [ 1; 2; 3.2; 4; 6; 5.4 ];`

- Command window:



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

column =

     4
     2
     7
     4
```

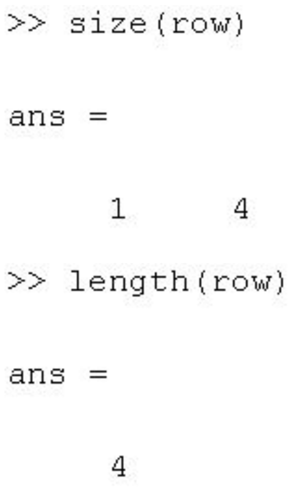
- Workspace:



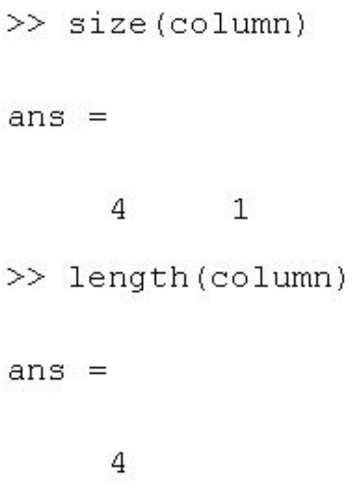
Name	Size	Bytes	Class
column	4x1	32	double array

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
```



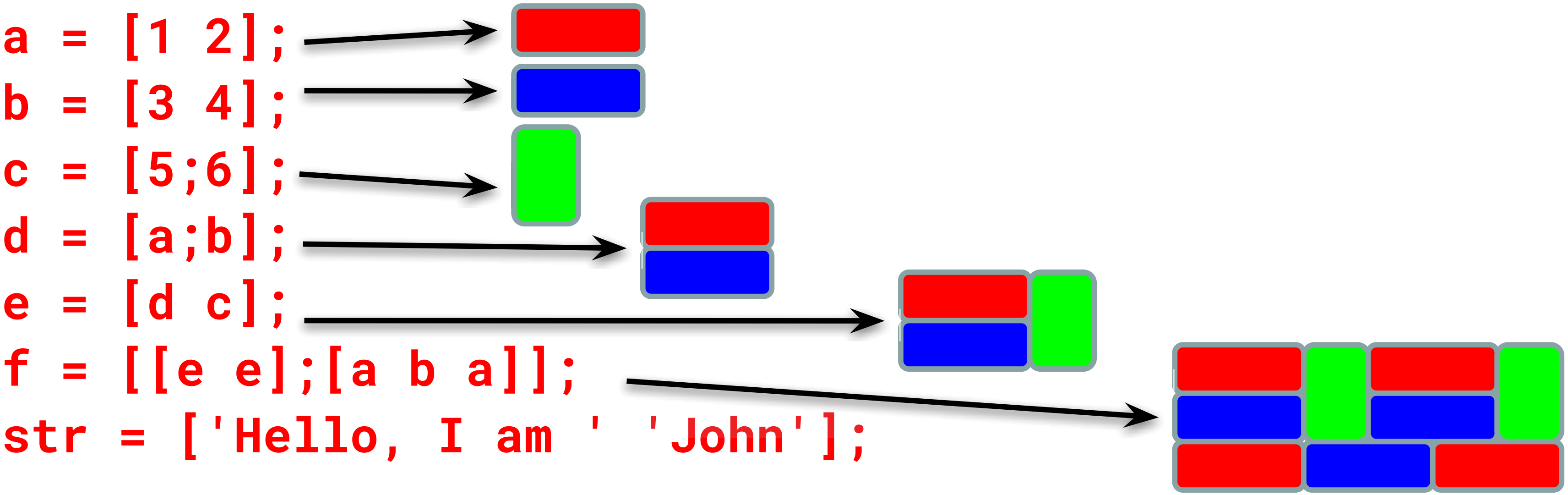
```
>> size(column)

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

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 enormous 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);`

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):

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

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

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)

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

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

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

$Must\ \ be\ square\ to\ do\ powers$

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

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

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

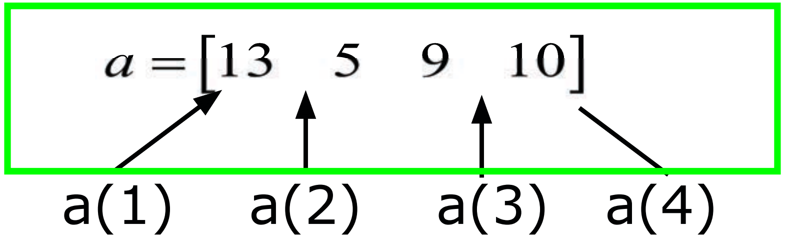
- To initialize a linear vector of values use **linspace**
  - » **`a=linspace(0,10,5)`**
    - ➢ Starts at 0, ends at 10 (inclusive), 5 values

- 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**

Vector Indexing

- MATLAB indexing starts with **1**, not **0**
  - ➢ We will not respond to any emails where this is the problem.
- a(n) returns the nth element



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

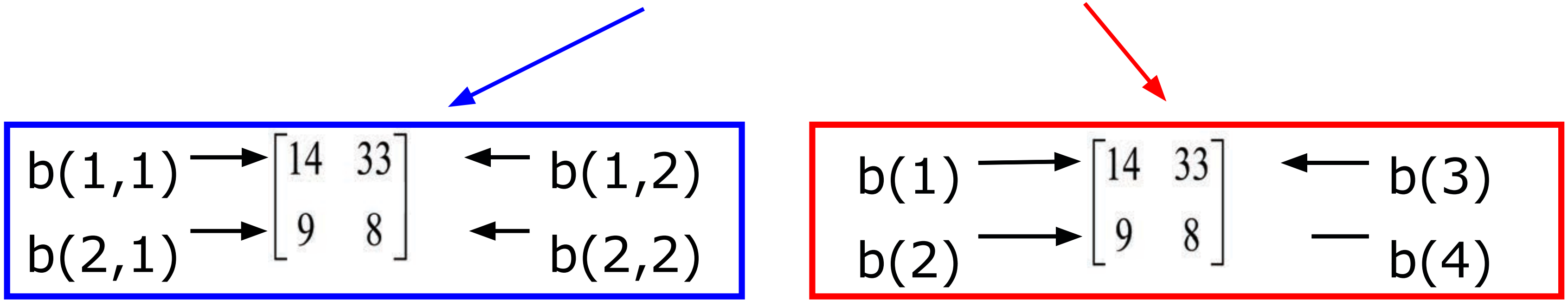
a(1) a(2) a(3) a(4)

- 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;`

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?

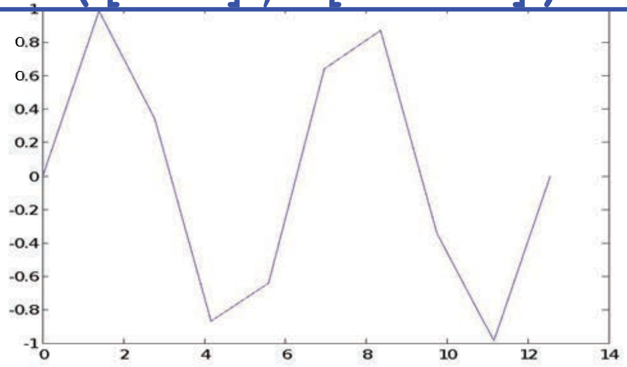
- **plot** generates dots at each (x,y) pair and then connects the dots with a line
- To make plot of a function look smoother, evaluate at more points

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

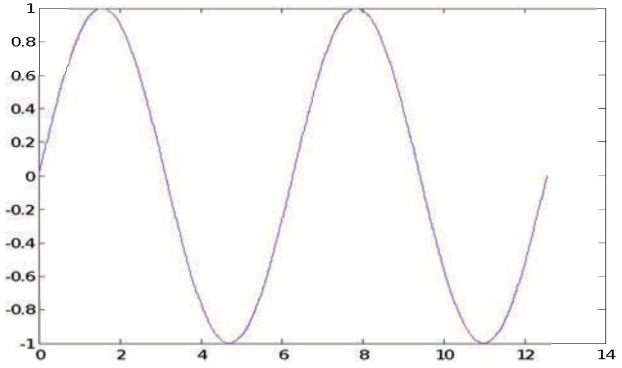
- x and y vectors must be same size or else you'll get an error

```
» plot([1 2], [1 2 3])
```

10 x values:



1000 x values:



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!!