Introduction to MATLAB

Sirawich Pipatprathanporn
PhD Student
Geosciences Department, Princeton University

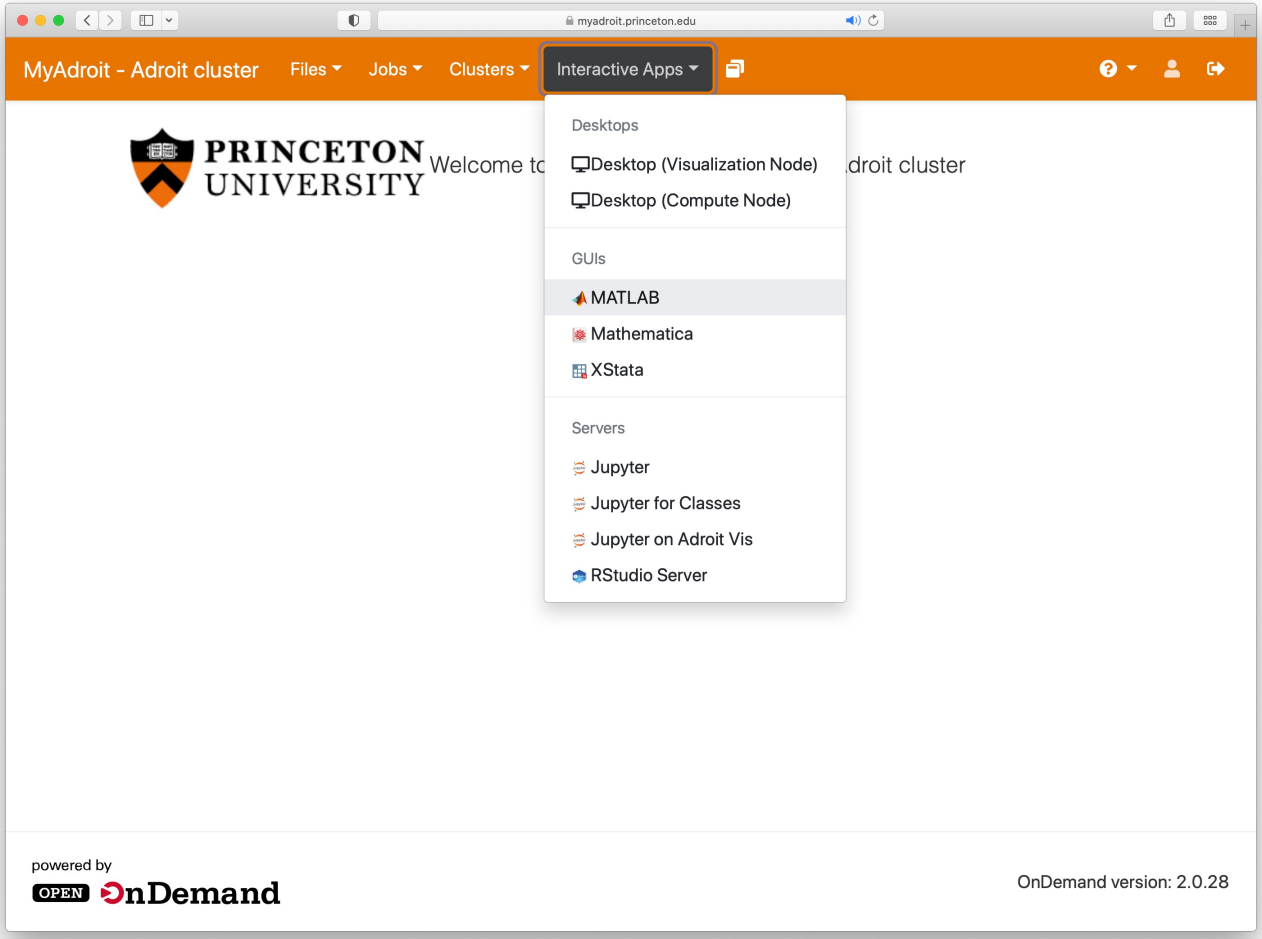
October 5, 2022

Outline

- Basics of MATLAB
- Matrices & other variable types
- Plots
- Loops & conditions
- Scripts & functions
- Search path
- File / Input output
- Environment variables

How to launch MATLAB on Adroit

1. Go to myadroit.princeton.edu and sign-in
2. Select Interactive Apps →MATLAB
3. Set the number of hours and the number of cores. For this session, 2 hours and 1 core is enough.
4. Click Launch
5. Once the job starts, click Luanch.



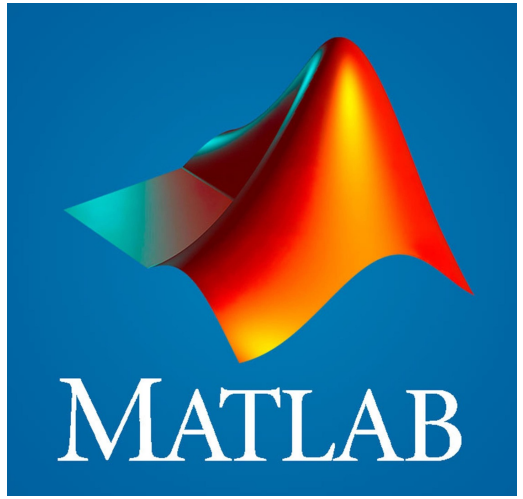
It may take a few minutes to launch

Objectives

- Introduce you to MATLAB environment
- Teach you basic commands for data processing and visualization
- Provide resource to learn more about MATLAB

What is MATLAB?

- Abbreviation of **MAT**rix **LAB**oratory
- A programming language that expresses matrix and array mathematics directly
- Capable for object-oriented programming

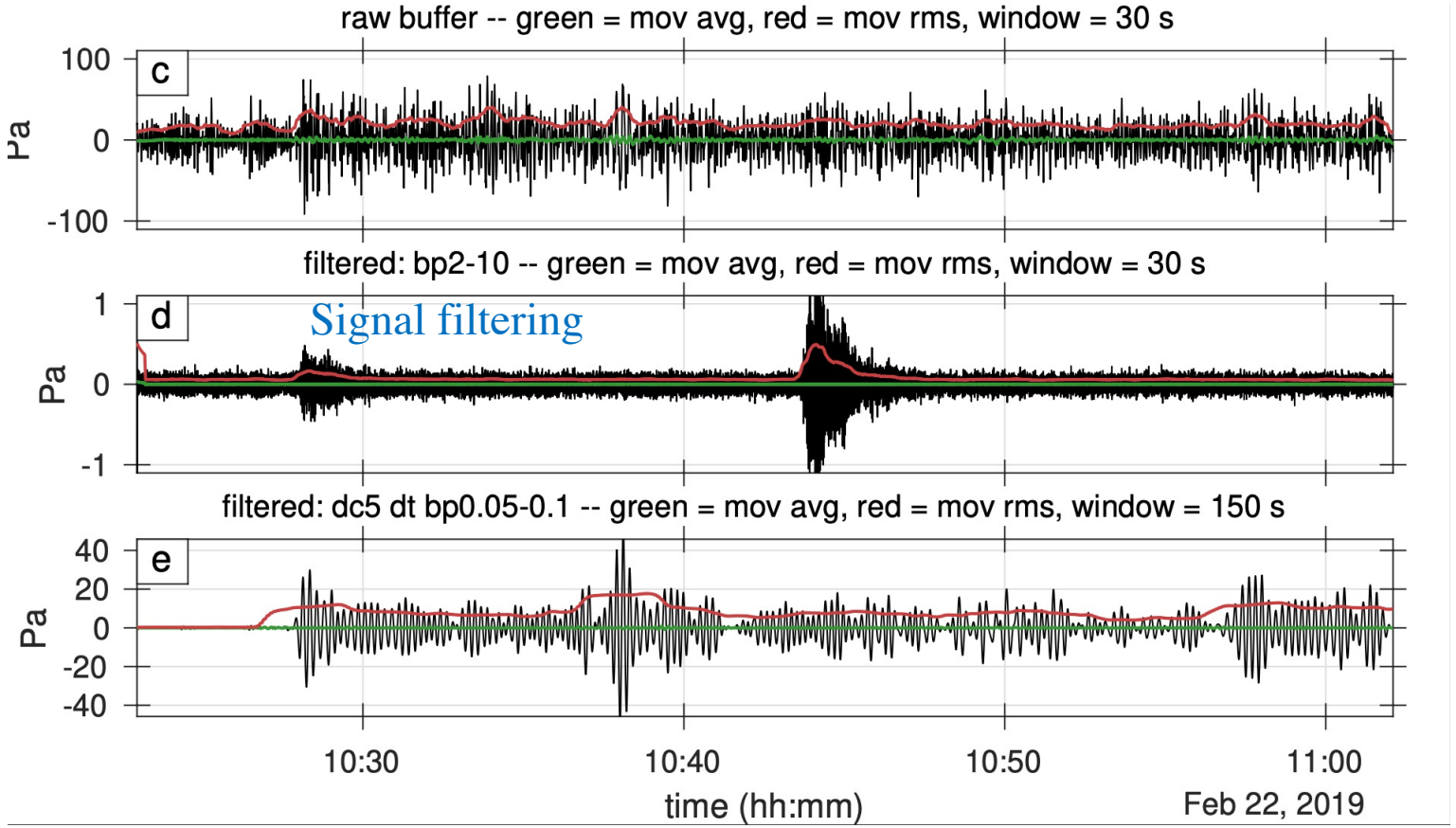


Why would you need MATLAB for research?

- Data analysis
- Signal processing

Read the data file and plot

Moving average and rms



Why would you need MATLAB for research?

- Data analysis
- Signal processing
- Data visualization
- And much more!

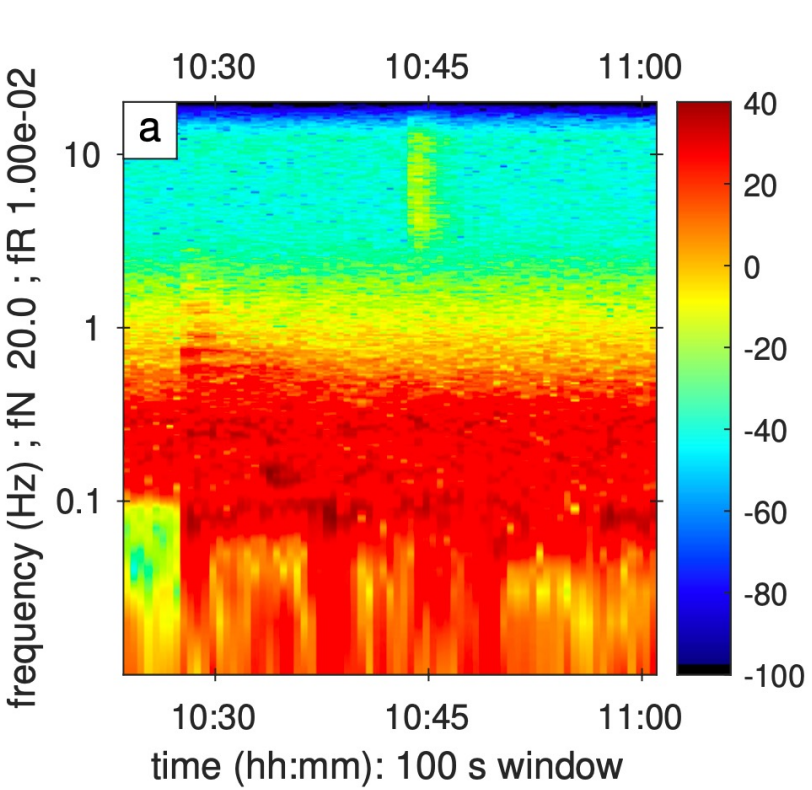
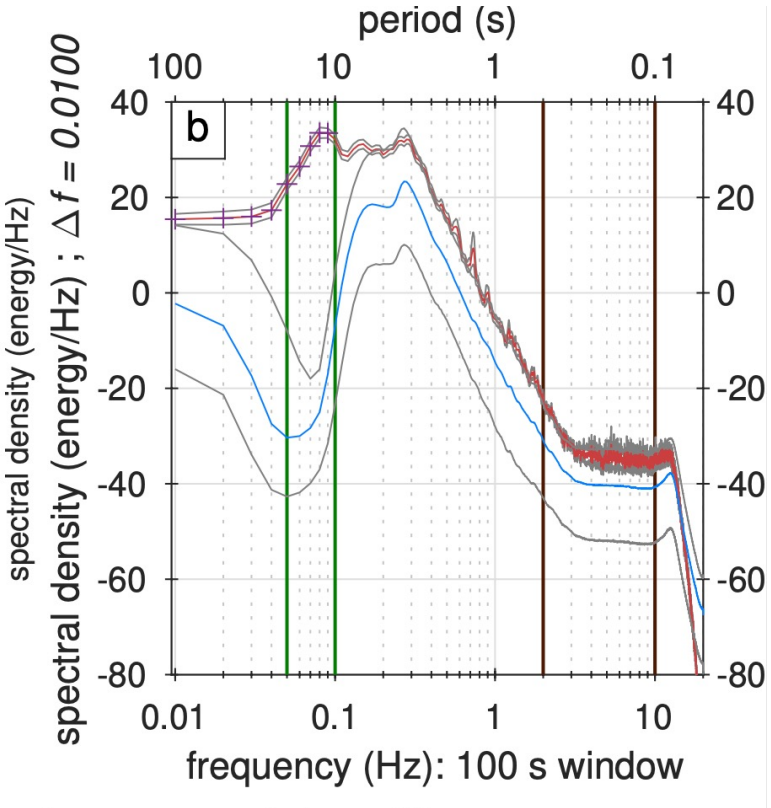


Image plot with colorbar

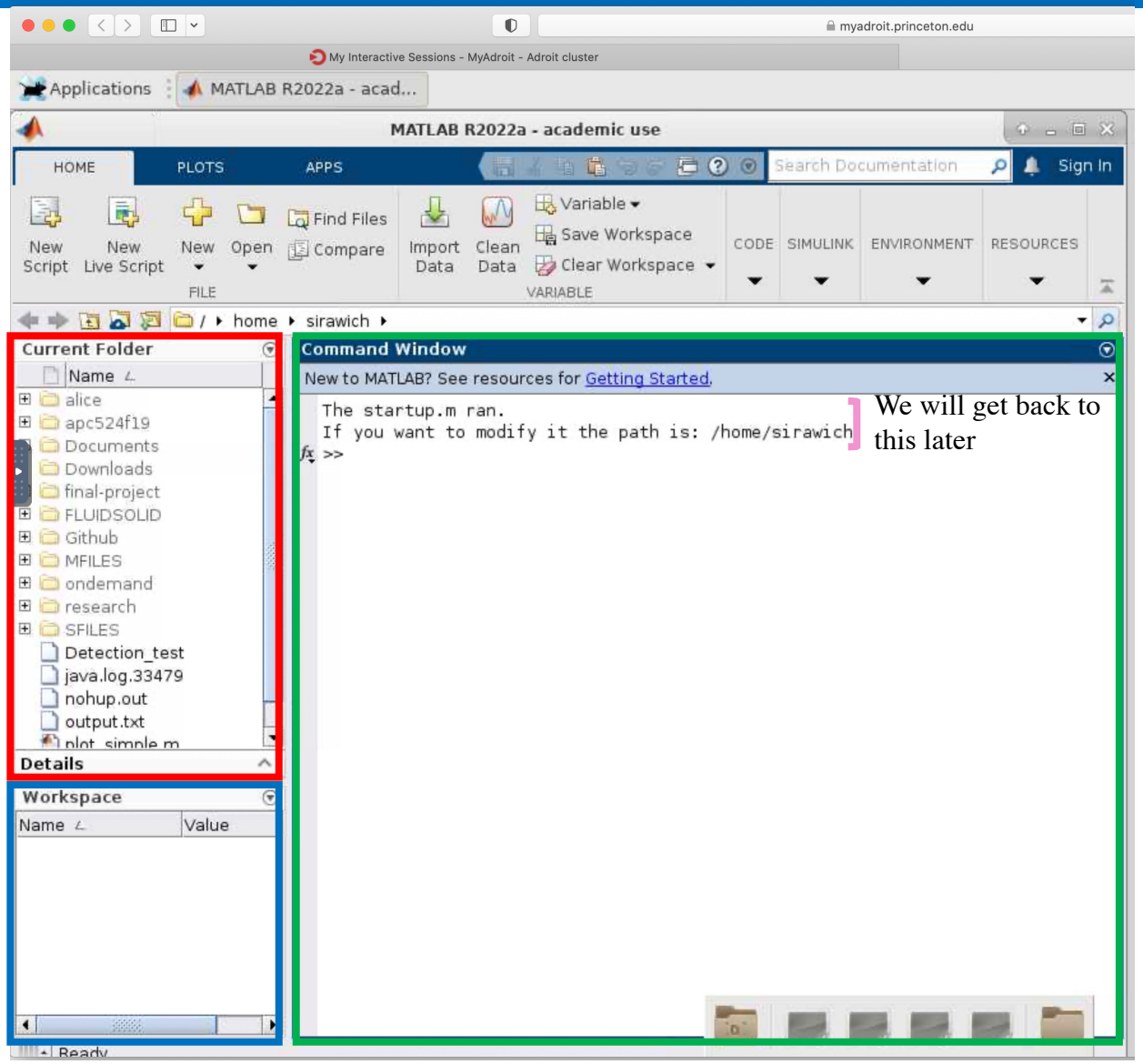


Convert to frequency domain using Fourier transformation

Desktop Basics

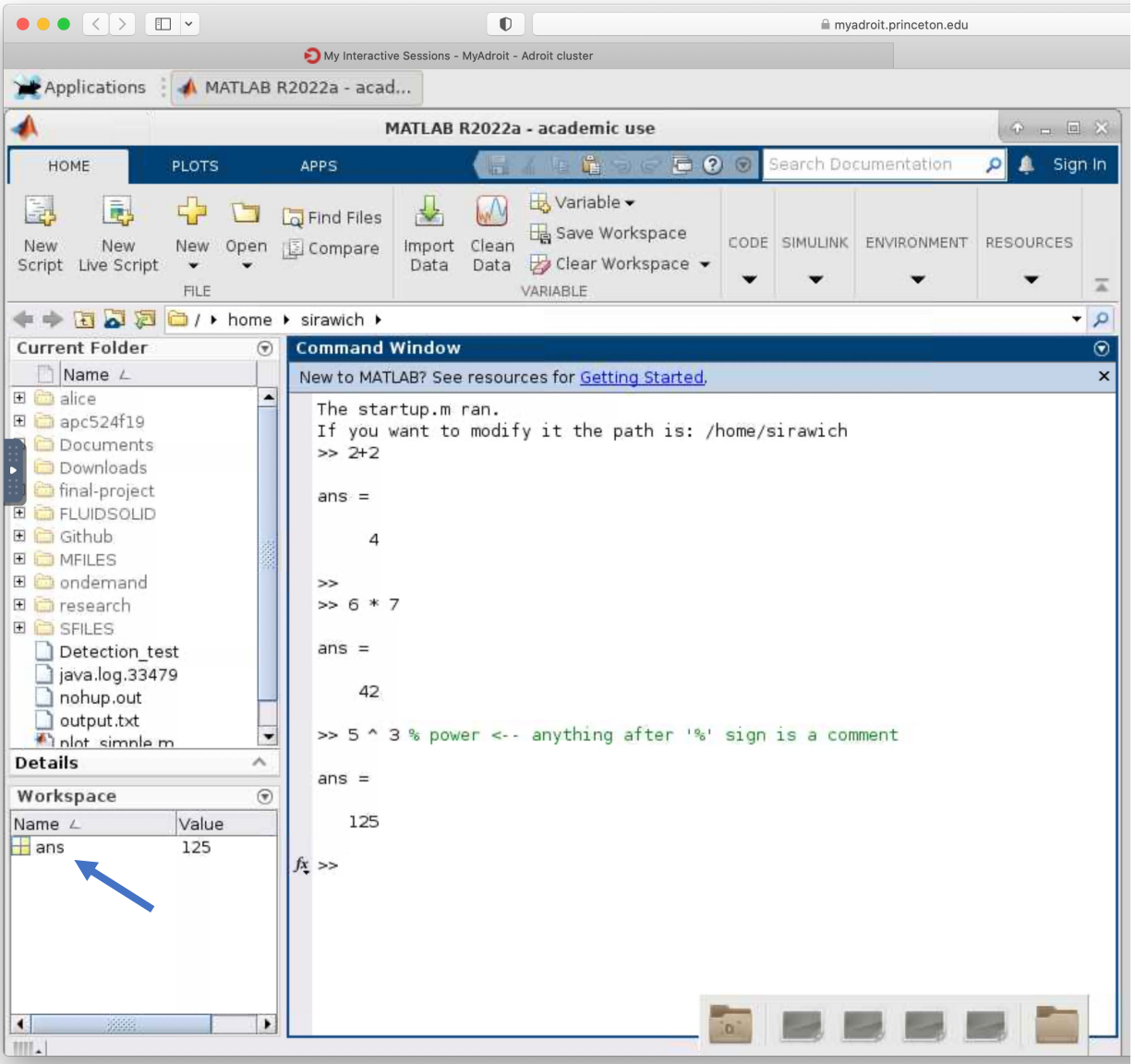
Current folder: access your files

Workspace: list of variables



Command window: enter the command at the command line

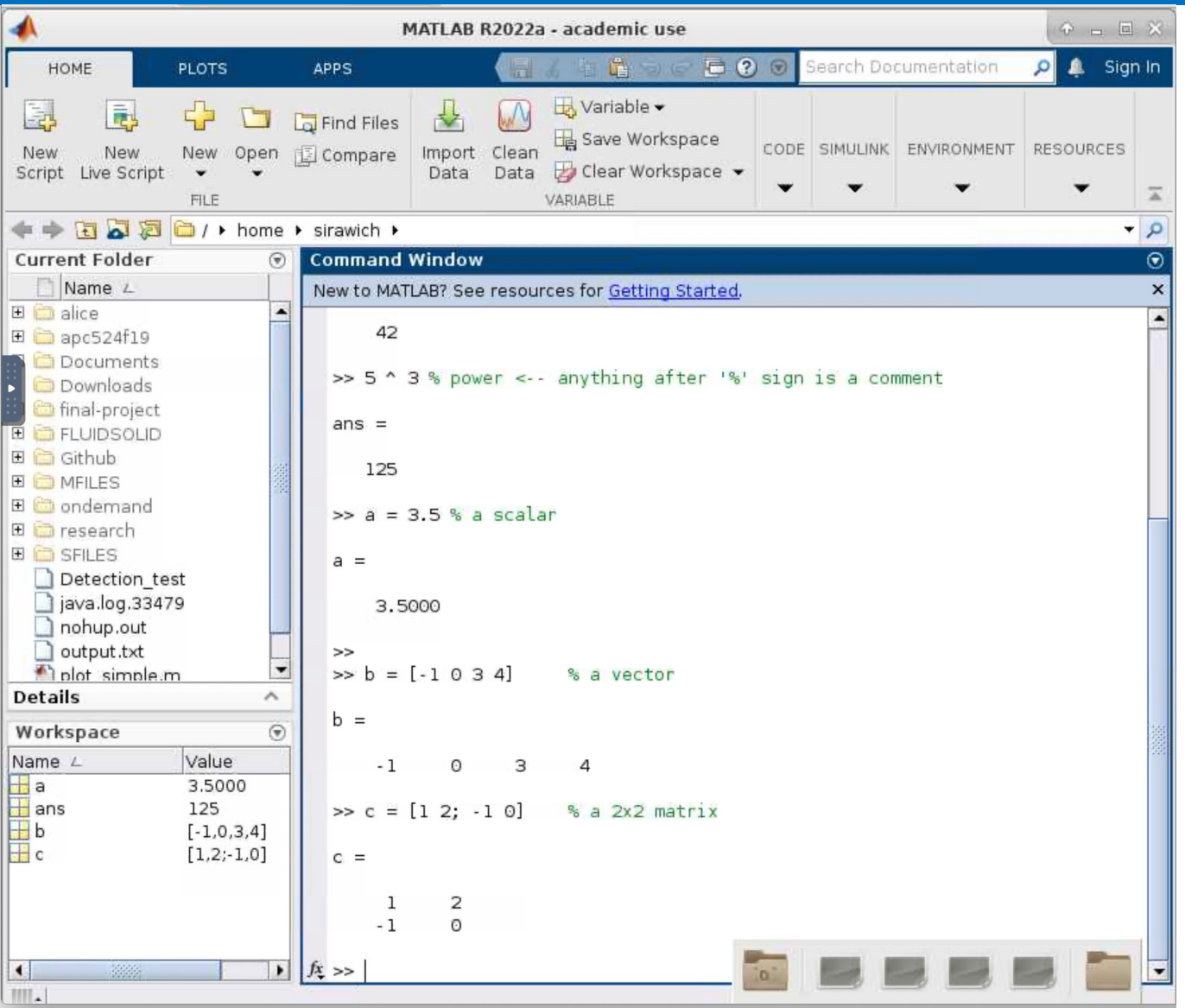
Let's enter some commands



- It can be as simple as 2 + 2
- Now MATLAB functions as a calculator

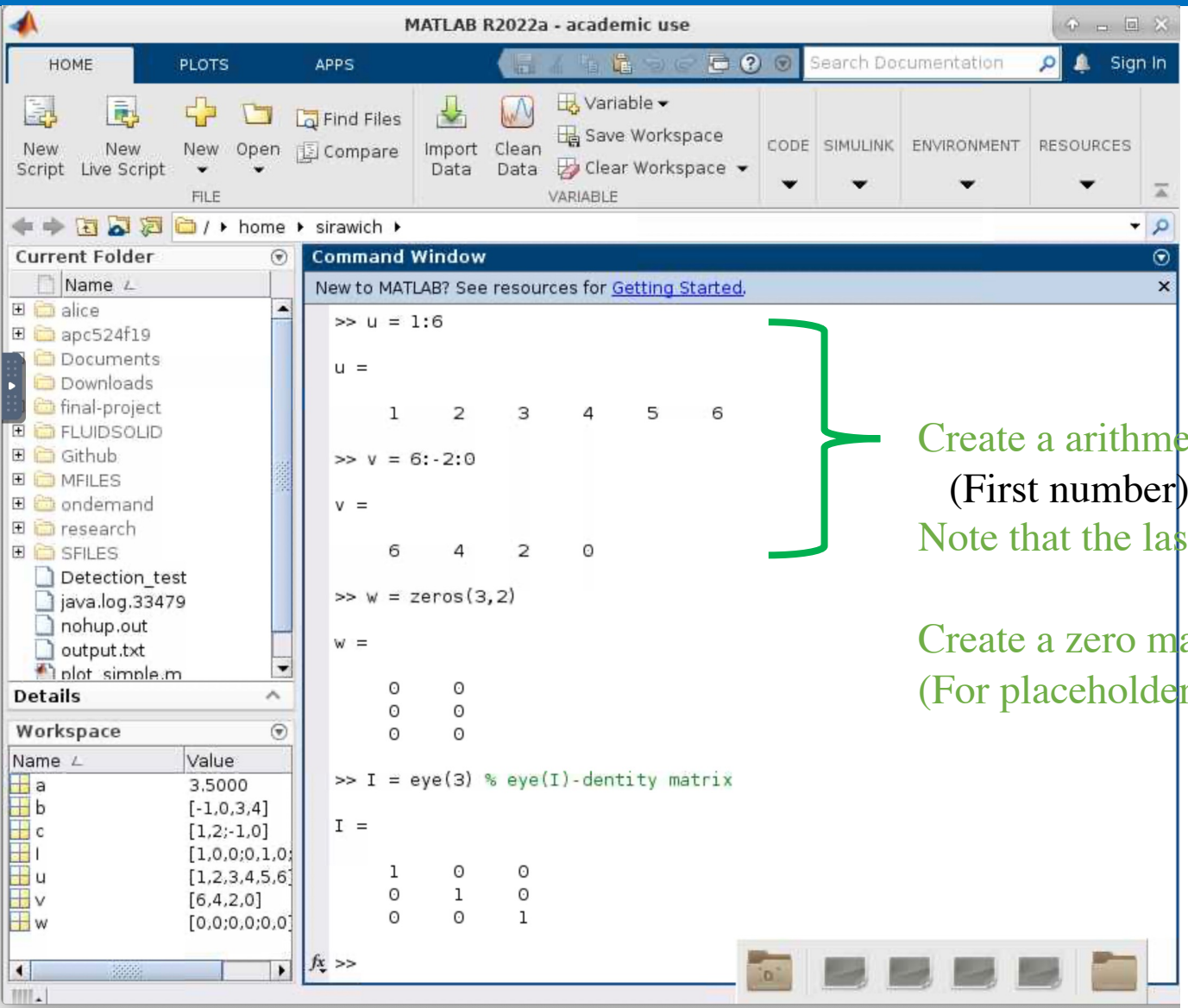
- Notice that a new variable appears in Workspace

Scalar, vector, and matrix



- Creating a scalar / vector / matrix by assigning a variable name to a value on the right hand side
- Use square brackets to indicate a vector / matrix
- Elements in a row are separated by a comma or a space
- Elements between rows are separated by a semicolon “;”

Alternative ways to create matrices



Create a arithmetic sequences; useful for indexing and loops

(First number) : (step size, default = 1) : (Last number)

Note that the last number is also included if it fits

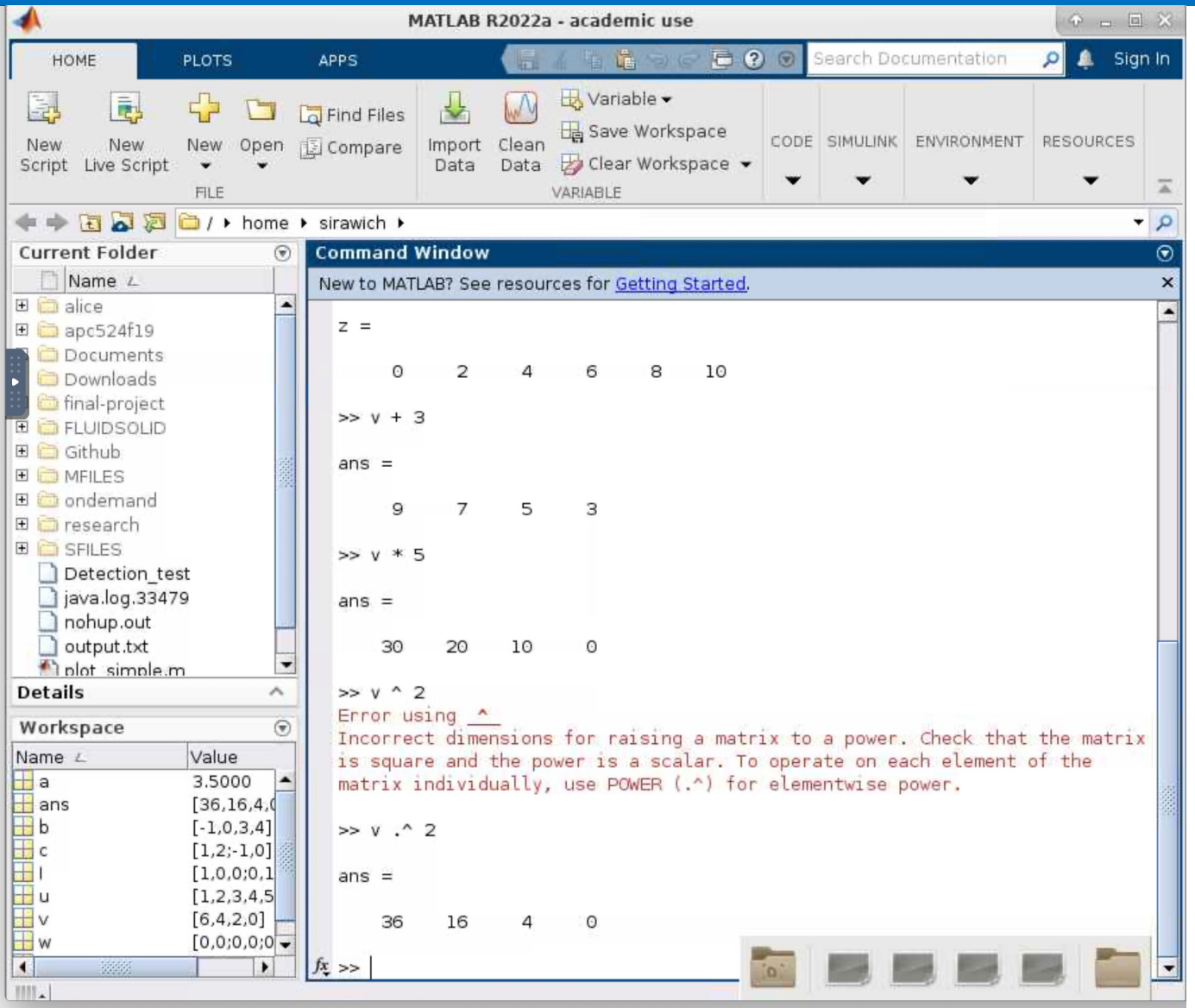
Create a zero matrix; useful for initialization

(For placeholder, consider nan instead.)

Create a equally-spaced matrix from 0 to 10 …

```
>> z = linspace(0, 10, 6)

z =

     0     2     4     6     8    10

fx >>
```

Matrix operations with a scalar



MATLAB allows processing a matrix with a scalar

For raising a matrix to a power, use `.^` instead of `^`

Matrix operations

```
>> v      % recall

v =

     6     4     2     0

>> v'     % transpose

ans =

     6
     4
     2
     0

>>
```

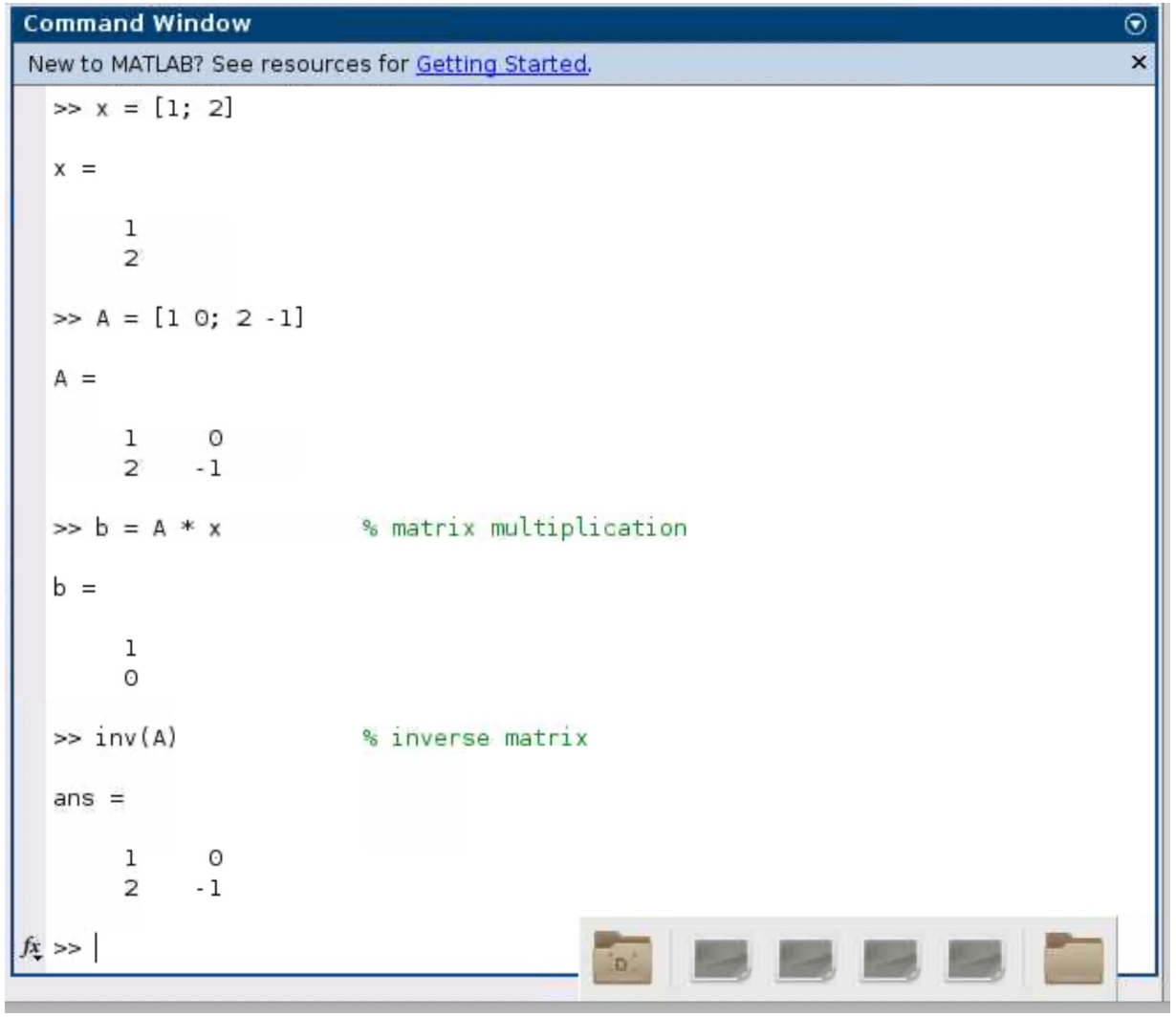
Useful when you want to create a row vector

```
>> x = [1 2 3]; y = [1 0 -1]; % use ';' to suppress the output
>> x - y

ans =

     0     2     4

>> x * y
Error using  *
Incorrect dimensions for matrix multiplication. Check that the number of
columns in the first matrix matches the number of rows in the second
matrix. To operate on each element of the matrix individually, use TIMES
(.*) for elementwise multiplication.

Related documentation

>> x .* y

ans =

     1     0    -3

>>
```

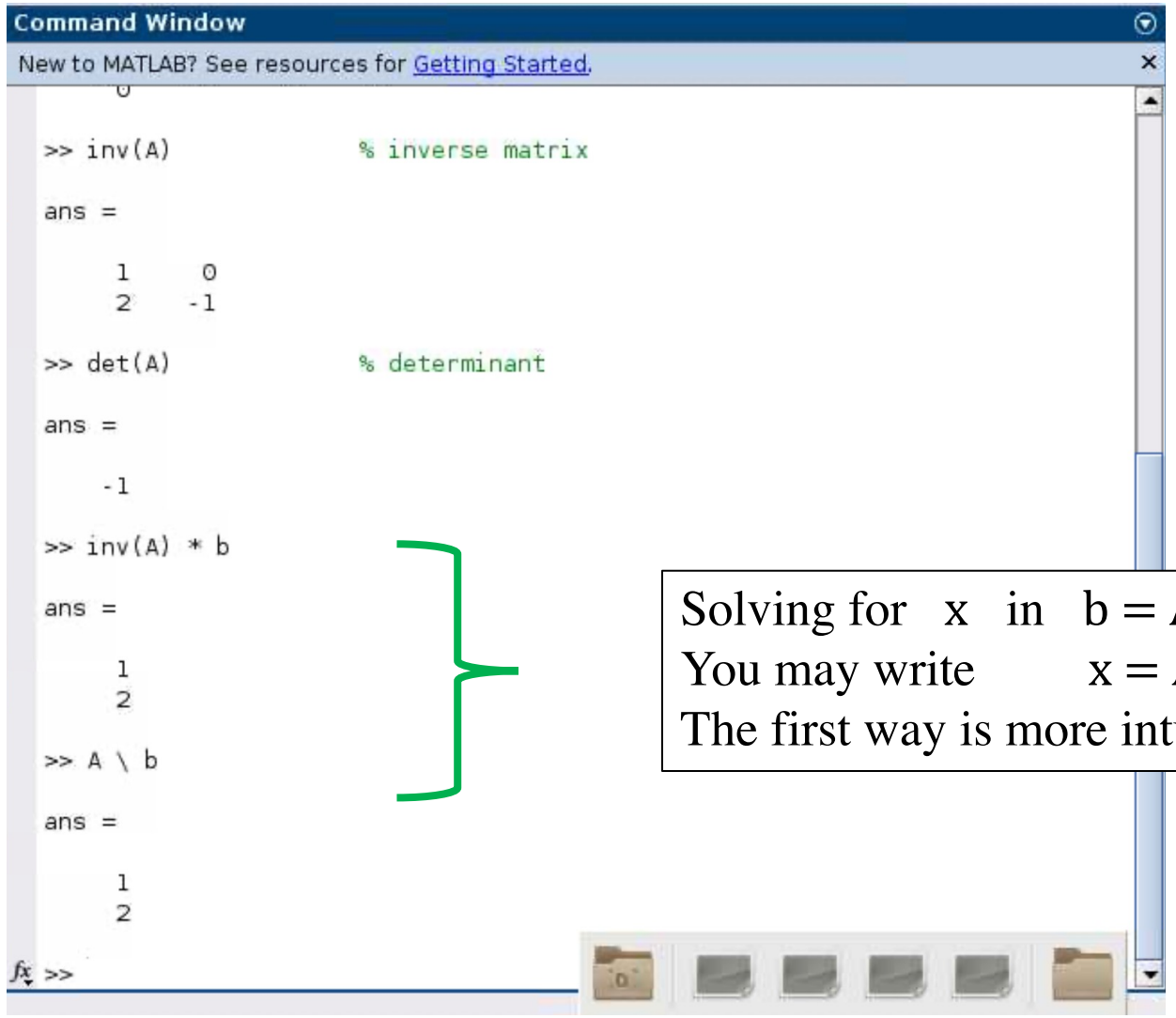
You can simply add/subtract two matrices

For element-wise multiplication, use `.*` instead of `*`

Matrix operations: Linear algebra



Matrix operations: Linear algebra

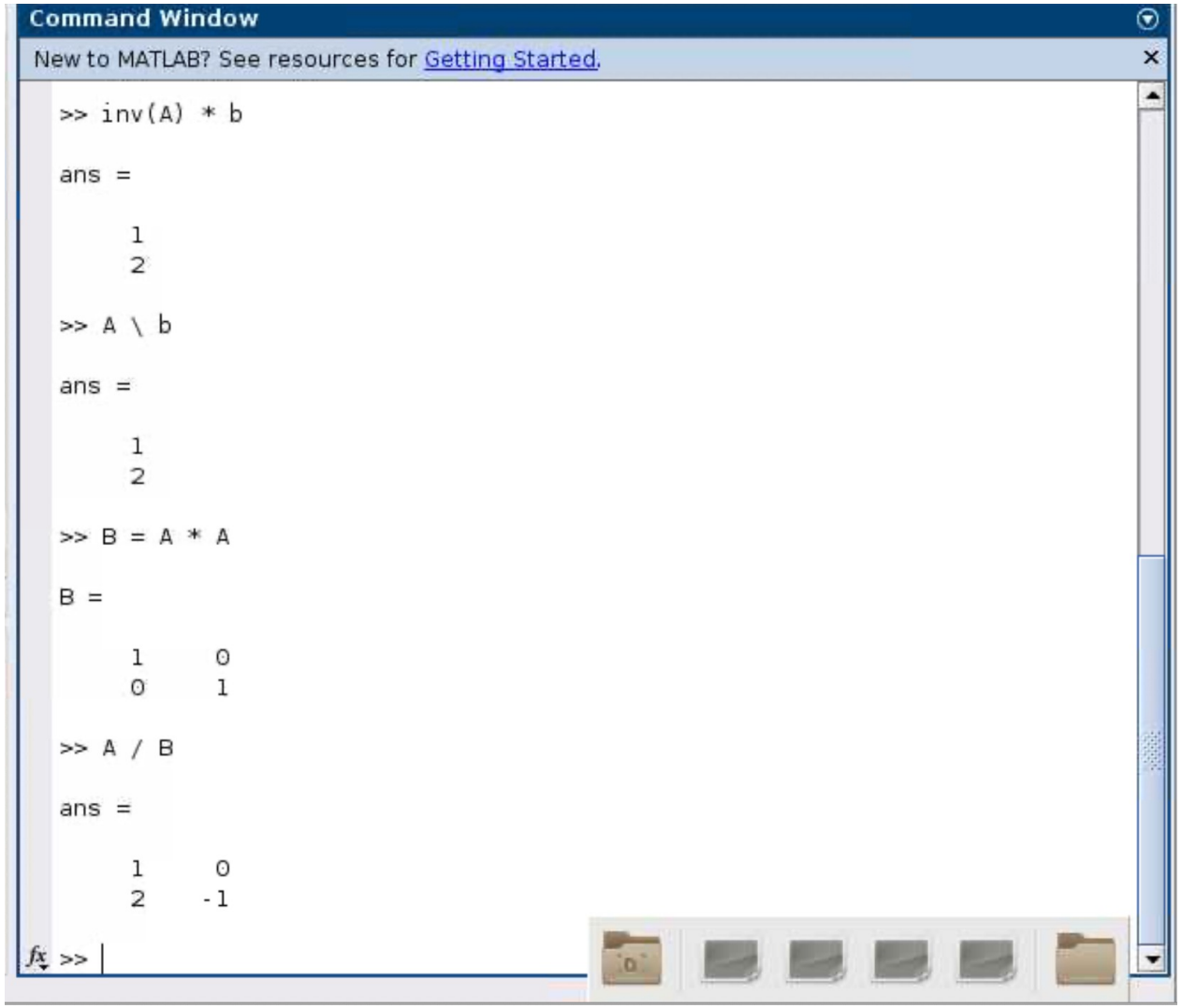


Solving for x in $b = Ax$

You may write $x = A^{-1}b$

The first way is more intuitive, but the second way is *faster*!

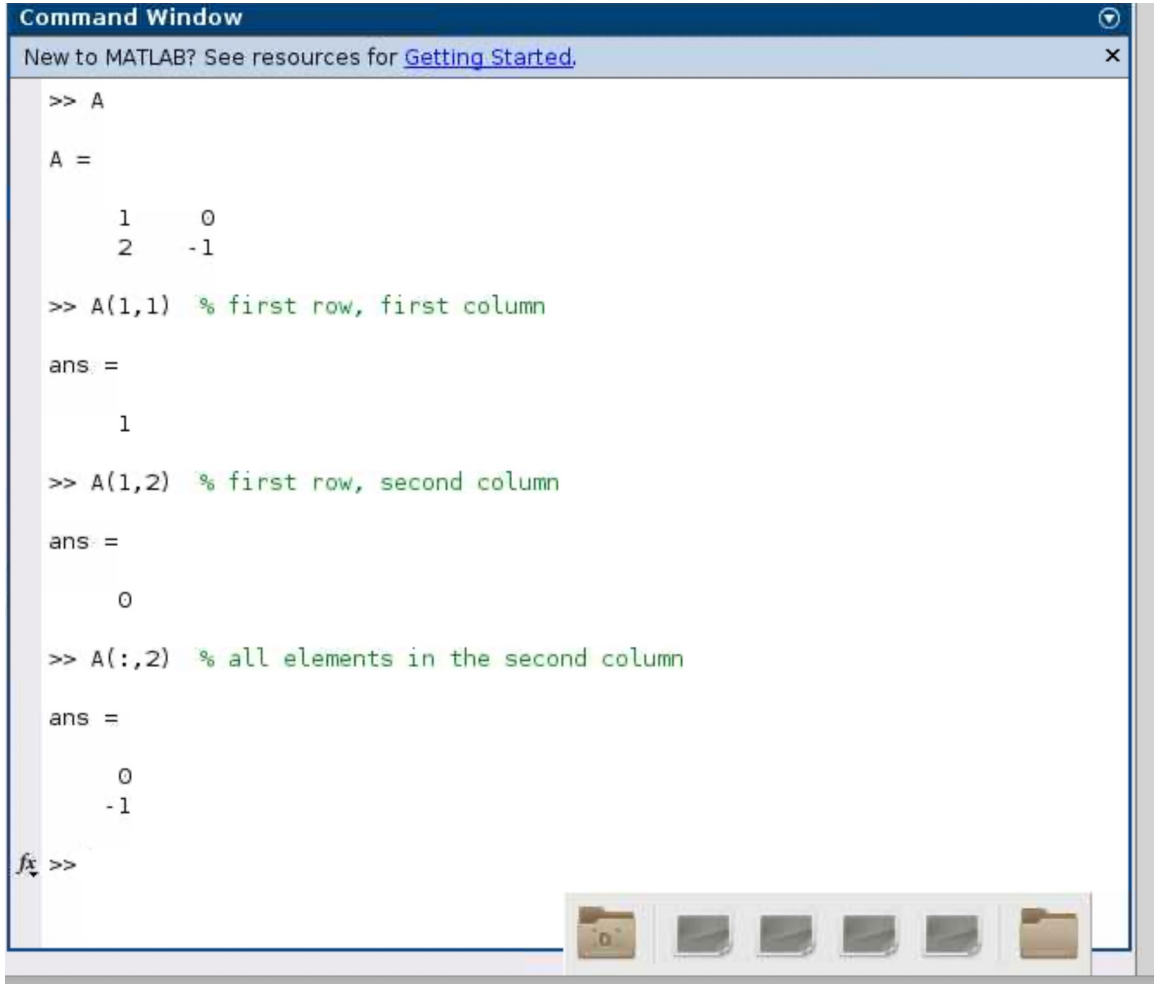
Matrix operations: Linear algebra



Matrix operations: Summary

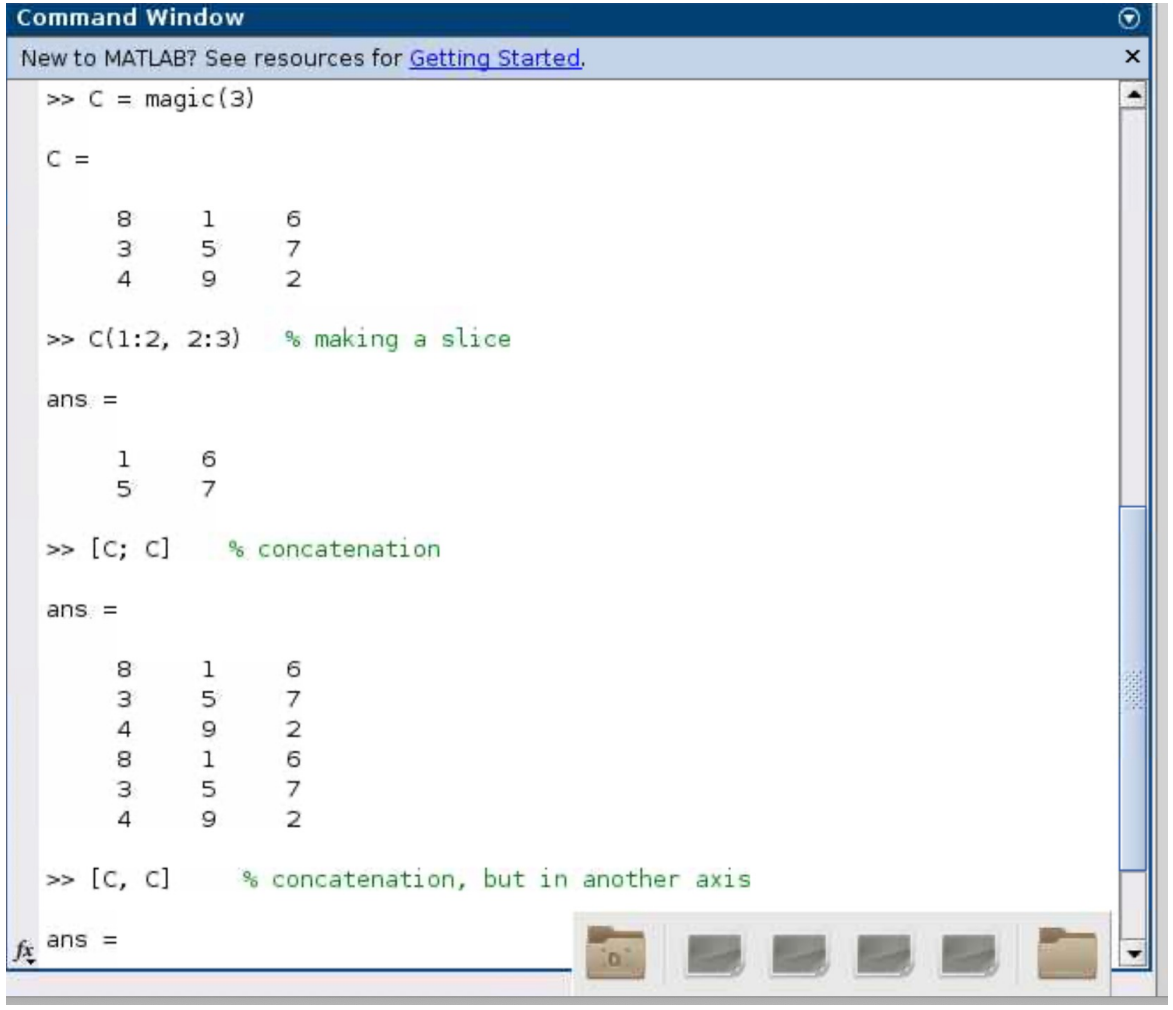
- Element-wise matrix operation
  - Feel free to modify a matrix (element-wise) with a scalar
  - Exception: use `.^` to raise a matrix to a power
  - For element-wise matrix multiplication and exponentiation, use `.*` and `.^`, respectively
- Linear Algebra operation
  - `'` – transpose
  - `*` – matrix multiplication
  - `/` – matrix division
  - `\` – left matrix division
  - `inv()` – matrix inverse
  - `det()` – determinant
  - `eig()` – eigenvalue and eigenvector
  - `dot()` – vector dot product
  - `cross()` – vector cross product

Matrix indexing



- Use parentheses to access elements in the matrix
- (row, column)
- Use colon ‘:’ to have all elements
- Index could be logical value

Matrix indexing

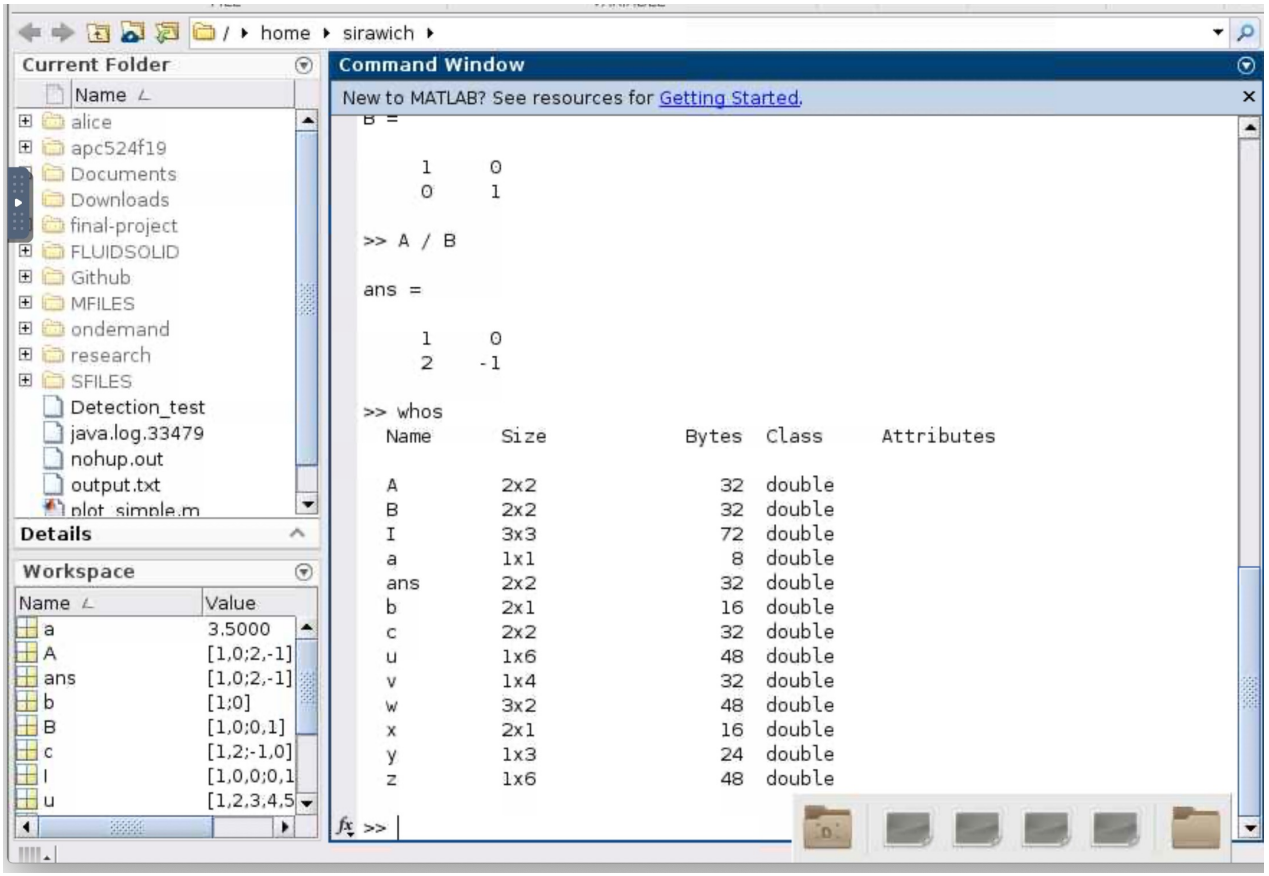


- Specifying number in front of and behind the colon to indicate the first and last elements (inclusive) for the slice

- Matrices can be concatenated in either vertically (using ‘:’) or horizontally (using ‘,’) as long as they have the same number of columns or rows, respectively.

Other variable types

- So far, all variables (scalars, vectors, and matrices) are the same type call double
- Other variable types include:
  - char – character e.g. c = ‘cat’
  - string – literal string e.g. d = “dog”
  - cell – a list of object
  - struct – data structure
- Use command `whos` to list variables in the workspace



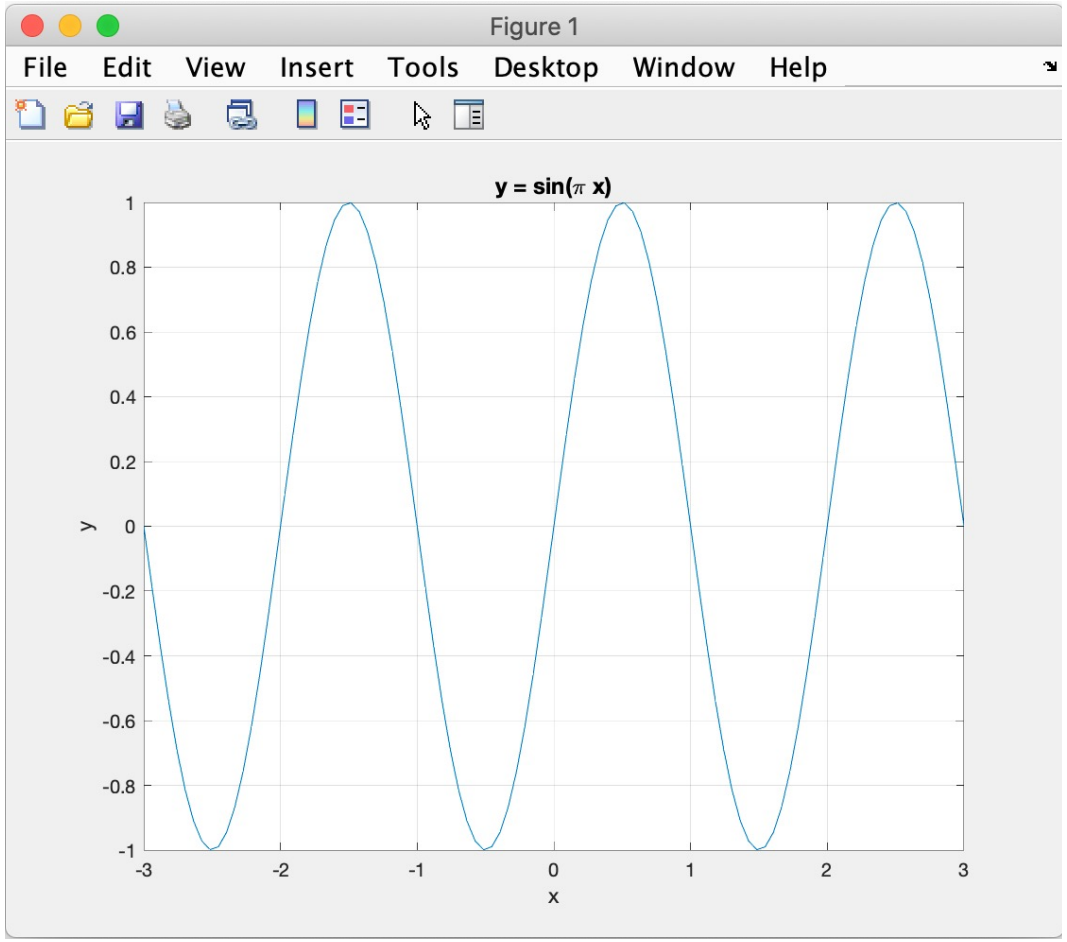
Useful commands

- whos – lists all variables in the workspace
- help [function name] – shows how to use a function
- doc [function name] – pops up a new window displaying documentation of a function (equivalent to a documentation page on mathworks.com)
- lookfor [keyword] – search all MATLAB files for keyword (1st line)
- which [item] – locate functions and files
- type [function name] – see the inside of a file

Visualization

- Here is an example on how to make a plot; It could be as simple as

```
x = linspace(-3, 3, 100);
y = sin(pi*x);
plot(x, y)
grid on
xlabel('x')
ylabel('y')
title('y = sin(\pi x)')
```



Graphic components

Figure – entire figure window

`.Parent` ⇅ `.Children`

Axes – a panel in a figure

`.Parent` ⇅ `.Children`

Objects – e.g. Line, Scatter, Histogram etc.

To access object, click the object and type `gco`

For axes and figures, type `gca` and `gcf`

To access the object, use `.get` command

Useful commands for plotting

`figure`	– create a new figure
`subplot`	– create a new subplot (axes)
`plot`	– make line plot
`scatter`	– make scatter plot
`hold on`	– allow adding second plot to the same axes
`gco/gca/gcf`	– get current object/axes/figure
`.Parent`	– access where the object belongs
`.Children(#)`	– access objects inside
`.get`	– get object properties
`set(obj,'Property','value')`	– set object properties

Need inspirations for your plots?

mathworks.com

MATLAB Plot Gallery

Search MathWorks.com

Overview | Getting Started | Features & Capabilities | What's New | For Students

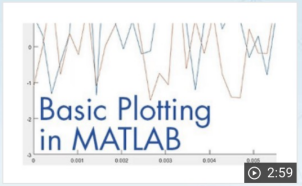
Trial software | Contact sales

MATLAB Plot Gallery

The MATLAB plot gallery provides various examples to display data graphically in MATLAB. Click **Launch example** below to open and run the live script examples in your browser with MATLAB Online™.

For more options, visit MATLAB Live Script Gallery to run live script examples from the MATLAB Community.

Learn the Basics

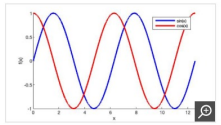


How to Use Basic Plotting Functions

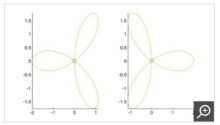
- Animation
- Contour Plots
- Data Distribution Plots
- Discrete Data Plots
- Geographic Plots
- Images
- Line Plots
- Polar Plots
- Scatter and Bubble Charts
- Surface and Mesh Plots
- Vector Fields
- Volume Visualization

Download code

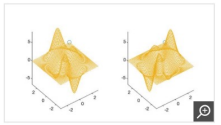
Animation



animatedline
Launch example

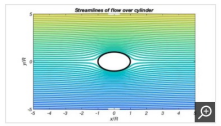


comet
Launch example

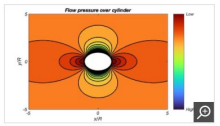


comet3
Launch example

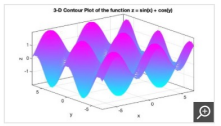
Contour Plots



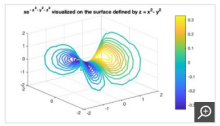
contour
Launch example



contourf
Launch example



contour3
Launch example



contourslice
Launch example

mathworks.com

Search MathWorks.com

Capabilities | What's New | For Students

Trial software | Contact sales

Images

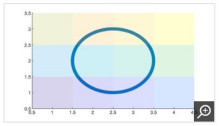
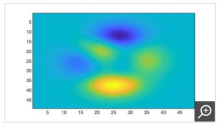
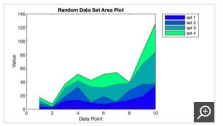


image
Launch example

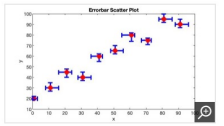


imagesc
Launch example

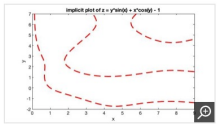
Line Plots



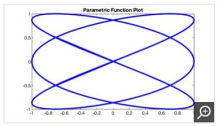
area
Launch example



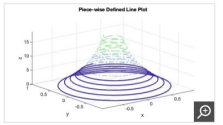
errorbar
Launch example



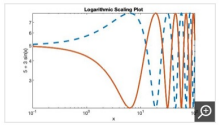
fimplicit
Launch example



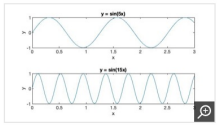
fplot
Launch example



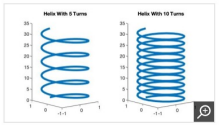
fplot3
Launch example



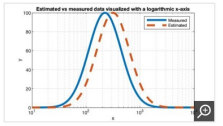
loglog
Launch example



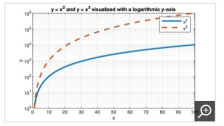
plot
Launch example



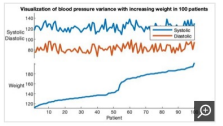
plot3
Launch example



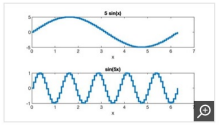
semilogx
Launch example



semilogy
Launch example



stackedplot
Launch example



stairs
Launch example

Loops and conditions

Command Window

New to MATLAB? See resources for Getting Started.

```
103.000000 is an odd number.
200.330000 is not an integer.
>> x = [0 1 3 4 8 12 27.5 103 200.33]

x =

  Columns 1 through 7

         0    1.0000    3.0000    4.0000    8.0000   12.0000   27.5000

  Columns 8 through 9

  103.0000  200.3300

>> for ii = 1:length(x)   % ii -- index variable
if mod(x(ii), 2) == 0
fprintf("%d is an even number.\n", x(ii));
elseif mod(x(ii), 2) == 1
fprintf("%d is an odd number.\n", x(ii));
else
fprintf("%f is not an integer.\n", x(ii));
end
end
0 is an even number.
1 is an odd number.
3 is an odd number.
4 is an even number.
8 is an even number.
12 is an even number.
27.500000 is not an integer.
103 is an odd number.
200.330000 is not an integer.
>>
>>
```

FOR loop

```
for ii = [1-d vector]
        statements
end
```

IF-ELSE statements

```
if [condition]
        statements
elseif [condition]
        statements
else
        statements
end
```

Loops and conditions

Command Window

New to MATLAB? See resources for Getting Started.

```
>> x

x =

  Columns 1 through 7

         0    1.0000    3.0000    4.0000    8.0000   12.0000   27.5000

  Columns 8 through 9

  103.0000  200.3300

>> ii = 1;
while ii <= length(x)
if mod(x(ii), 2) == 0
fprintf("%d is an even number.\n", x(ii));
elseif mod(x(ii), 2) == 1
fprintf("%d is an odd number.\n", x(ii));
else
fprintf("%f is not an integer.\n", x(ii));
end
ii = ii + 1;
end
0 is an even number.
1 is an odd number.
3 is an odd number.
4 is an even number.
8 is an even number.
12 is an even number.
27.500000 is not an integer.
103 is an odd number.
200.330000 is not an integer.
>>
>>
```

- While-loop version

[set initial state variables]

while [condition]

    statements

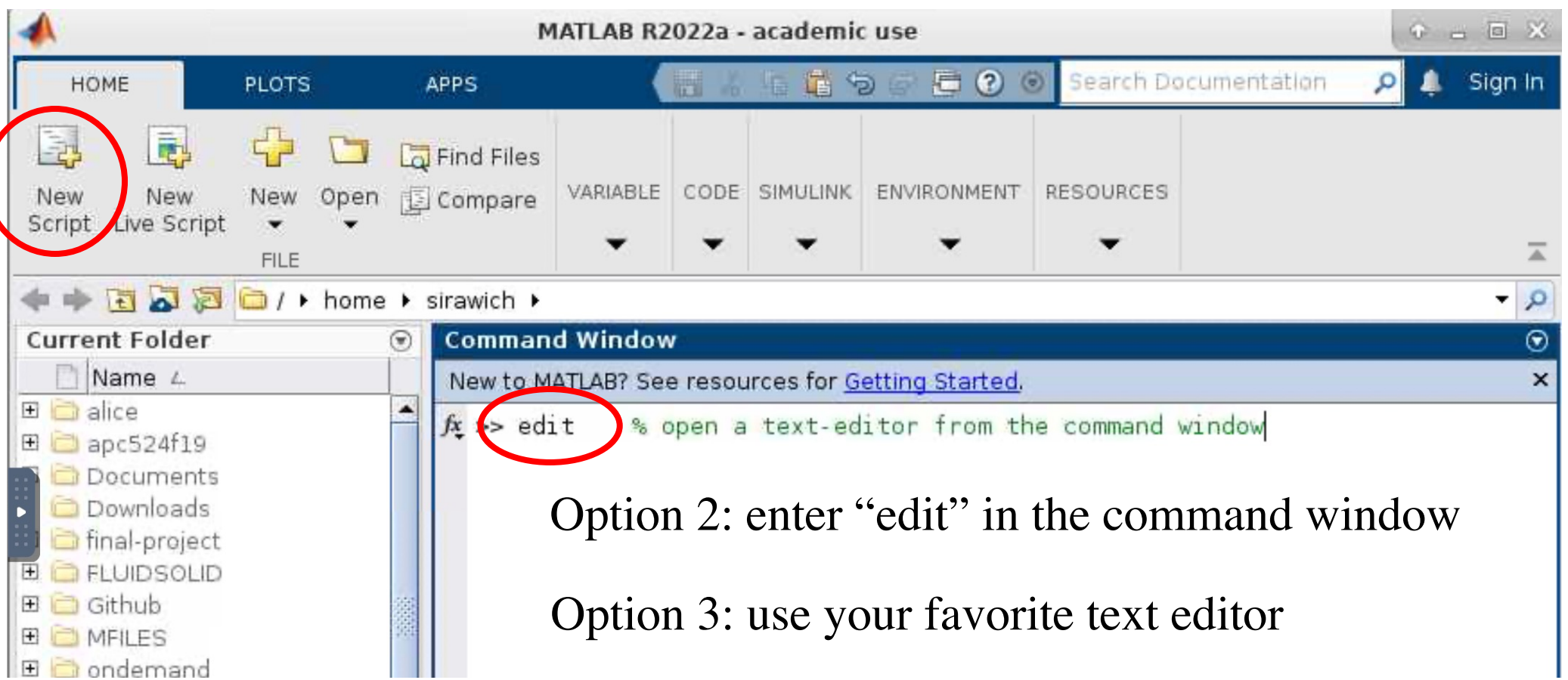
    [update state variables]

end

Instead of typing the same set of command again and again, why don't we turn it to a script

How to open the text editor

Option 1: Click "New Script"

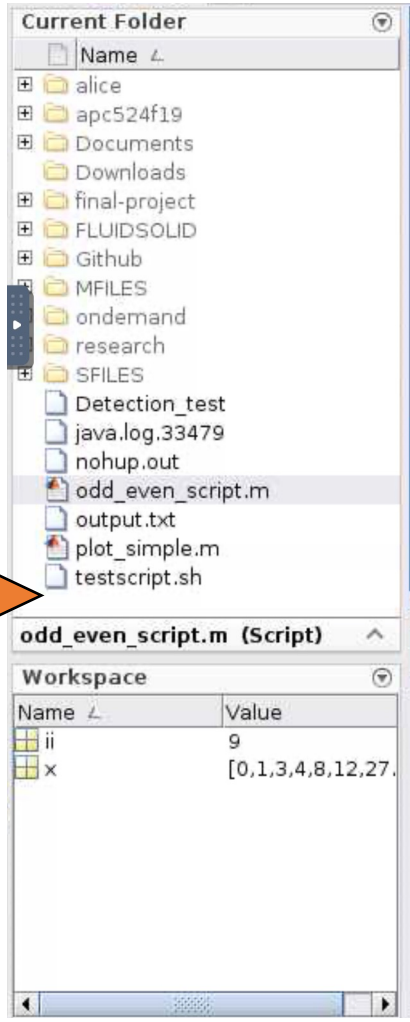


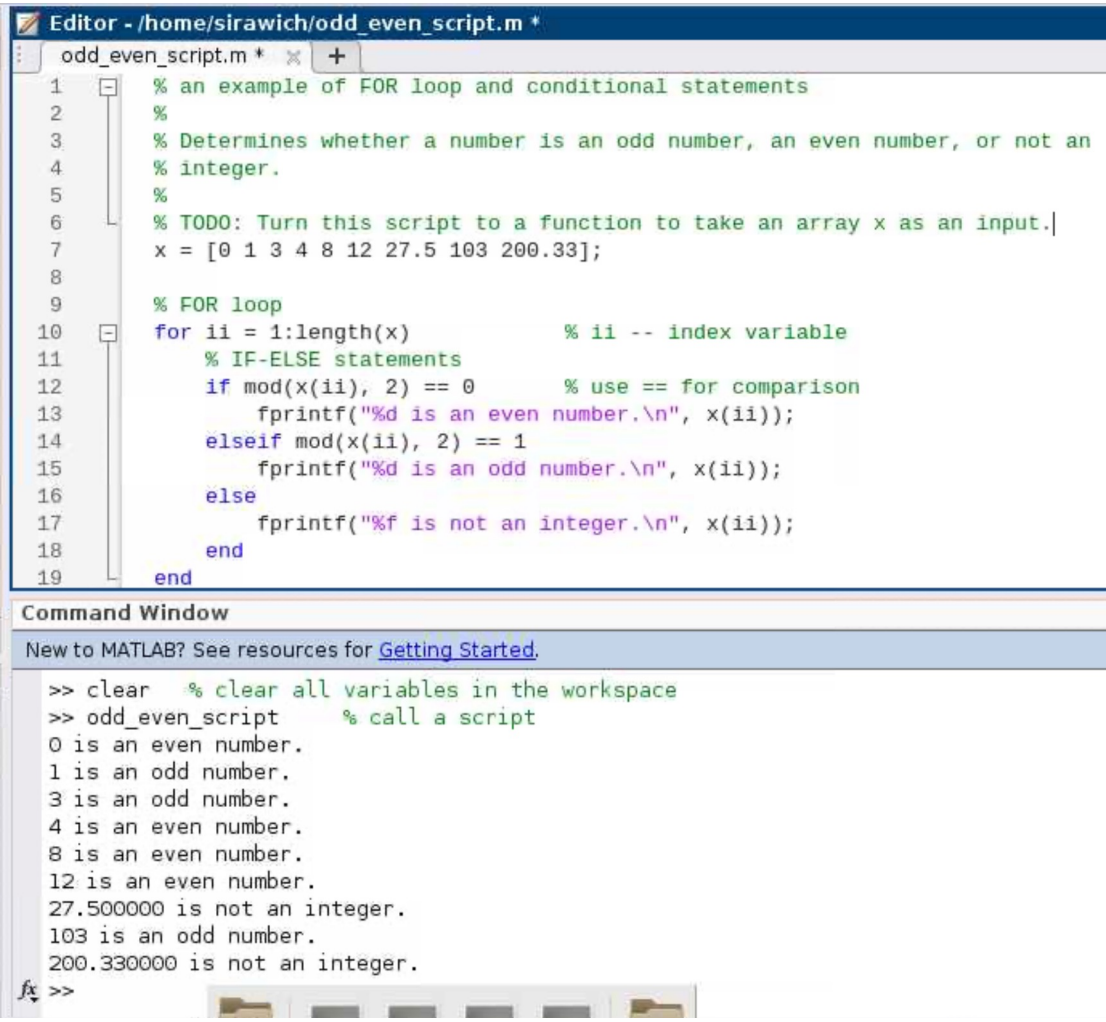
Option 2: enter "edit" in the command window

Option 3: use your favorite text editor

Script

```
>> x = [0 1 3 4 8 12 27.5 103 200.33]

x =

  Columns 1 through 7

         0    1.0000    3.0000    4.0000    8.0000   12.0000   27.5000

  Columns 8 through 9

  103.0000  200.3300

>> for ii = 1:length(x)   % ii -- index variable
if mod(x(ii), 2) == 0
fprintf("%d is an even number.\n", x(ii));
elseif mod(x(ii), 2) == 1
fprintf("%d is an odd number.\n", x(ii));
else
fprintf("%f is not an integer.\n", x(ii));
end
end
0 is an even number.
1 is an odd number.
3 is an odd number.
4 is an even number.
8 is an even number.
12 is an even number.
27.500000 is not an integer.
103 is an odd number.
200.330000 is not an integer.
>>
```





Function

Editor - /home/sirawich/odd_even.m

startup.m | odd_even_script.m | odd_even.m

```
1  function odd_even(x)
2  % ODD_EVEN(x)
3  %
4  % Determines whether a number is an odd number, an even number, or not an
5  % integer.
6  %
7  % INPUT:
8  % x         an array of numbers
9  %
10 % Last modified by sirawich-at-prineton.edu, 10/04/2022
11
12 % FOR loop
13 for ii = 1:length(x)            % ii -- index variable
14     % IF-ELSE statements
15     if mod(x(ii), 2) == 0       % use == for comparison
16         fprintf("%d is an even number.\n", x(ii));
17     elseif mod(x(ii), 2) == 1
18         fprintf("%d is an odd number.\n", x(ii));
19     else
20         fprintf("%f is not an integer.\n", x(ii));
21     end
22 end
```

Command Window

New to MATLAB? See resources for Getting Started.

```
>> odd_even(1:0.5:4);
1 is an odd number.
1.500000 is not an integer.
2 is an even number.
2.500000 is not an integer.
3 is an odd number.
3.500000 is not an integer.
4 is an even number.
fx >>
```

While script is useful for repeatedly running a set of commands, it is more useful if we can change the script to run with any array.

Imagine if you can call this script as

```
odd_even(x)
```

where x is a 1-d array!

This is where functions are useful.

Search path

- If you save your scripts/functions to other folders besides the current folder, you will find out that you cannot run your scripts/functions.

- But you would say, what's about the MATLAB built-in functions?

- Type `path` to print MATLAB's current search path

Adding a Path to Search Path

- To add a new path, enter

```
addpath path
addpath('path')
```

- To recursively add all paths under a given path, enter

```
addpath(genpath('path'))
```

- Now you may organize your files to any location you would like

File input / output

`save/load`	-- save/load variables from/to the workspace
`fopen`	-- open a file (for read/write)
`fclose`	-- close a file
`fprintf/fscanf`	-- write/read a text file
`fwrite/fread`	-- write/read a binary file

File input / output

- Saving variables from the workspace to a `.mat` file

```
save('filename.mat', 'varname1', 'varname2')
```

- Loading variables to the workspace

```
load filename.mat
load('filename.mat', 'varname1', 'varname2')
```

- Writing into a text file

```
fid = fopen('filename.txt');
fprintf(fid, "some text.\n");
fclose(fid);
```

Note: If `fid` is omitted, `fprintf` will print the text to the screen.

Environment variables

- MATLAB supports environment variables from the shell
- Environment variables can be accessed from any location: command window and inside any functions.

- To access an environment variable,

```
getenv('env_var_name')
```

- To create/replace an environment variable,

```
setenv('env_var_name', 'value')
```

startup.m

- Stores user-specified commands, executed when MATLAB starts
- I add paths to files for my research.
- Environment variables allow me to run this file on different machines where I might put files in different locations.

rgbcolor.m | startup.m

```
1  fprintf('The startup.m ran.\nIf you want to modify it the path is: %s\n', pwd)
2
3  addpath(genpath(getenv('PPICKER')))
4
5  addpath(genpath(getenv('OMNIA')))
6
7  addpath(genpath(fullfile(getenv('OMNIA'), 'changepoints')))
8
9  addpath(genpath(fullfile(getenv('OMNIA'), 'notmycode', 'MatTaup')))
10
11 addpath(genpath(fullfile(getenv('OMNIA'), 'mermaid')))
12
13 addpath(genpath(fullfile(getenv('OMNIA'), 'earthquakes'))) % this is for phangle.m
14
15 addpath(genpath(getenv('SLEPIANS')))
16
17 addpath(genpath(getenv('MERMAID')))
18
19 addpath(genpath(getenv('RFLEXA')))
20
21 addpath(genpath(getenv('MERMAID2')))
22
23 addpath(genpath(fullfile(getenv('MERMAID'), 'notmycode'))) % this is for focalmech.m
24
25 javaaddpath(fullfile(getenv('OMNIA'), 'notmycode', 'MatTaup', 'lib', 'matTaup.jar'))
26
27 % import irisFetch
28 addpath(genpath(getenv('IRISFETCH')))
29
30 javaaddpath(fullfile(getenv('IRISFETCH'), 'IRIS-WS-2.0.18.jar'))
31
32 % ensure that running this does not cause MATLAB to use other startup file
33 addpath(genpath(getenv('STARTUP')))
34
35 % graphic settings
36 %set(0, 'Units', 'inches', 'DefaultAxesXGrid', 'on', 'DefaultAxesYGrid', 'on');
```

Resources for learning MATLAB

- `help, doc` – to look for how to use a function
- `type` – to look inside a function … for inspiration for your functions
- https://www.mathworks.com – MATLAB official website
- https://www.tutorialspoint.com/matlab/index.htm – online tutorial

Challenge: Fitting the data

We are trying to figure out the gravitational acceleration (g) by measuring the period of oscillation of a pendulum at a few lengths. The period of oscillation is governed by

$$T^2 = \frac{4\pi^2}{g} L$$

Given the measurements as in the table below, make a scatter plot of T^2 against L with a best-fitted line. Then, use the slope to find g. You may find a function `polyfit` helpful.

Length of the pendulum, L (m)	Period of oscillation, T (s)
0.20	0.90
0.25	1.00
0.30	1.09
0.35	1.19
0.40	1.28
0.50	1.41