MATLAB Workshop

By: Ryan, Rachel, Harish

x MATH SOCIETY

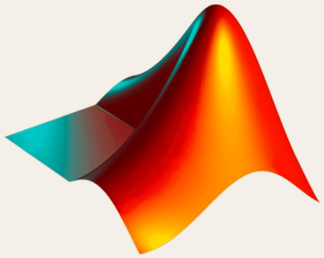
Meet the Team

Ryan
Y3 Math

Harish
Y3 DSA

Rachel
Y3 DSA

01 & 2 - Introduction to Matlab & Its Interface

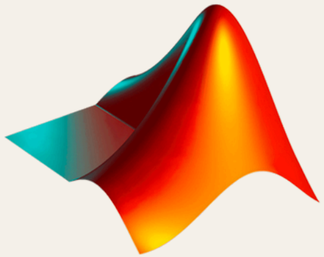


MATLAB®

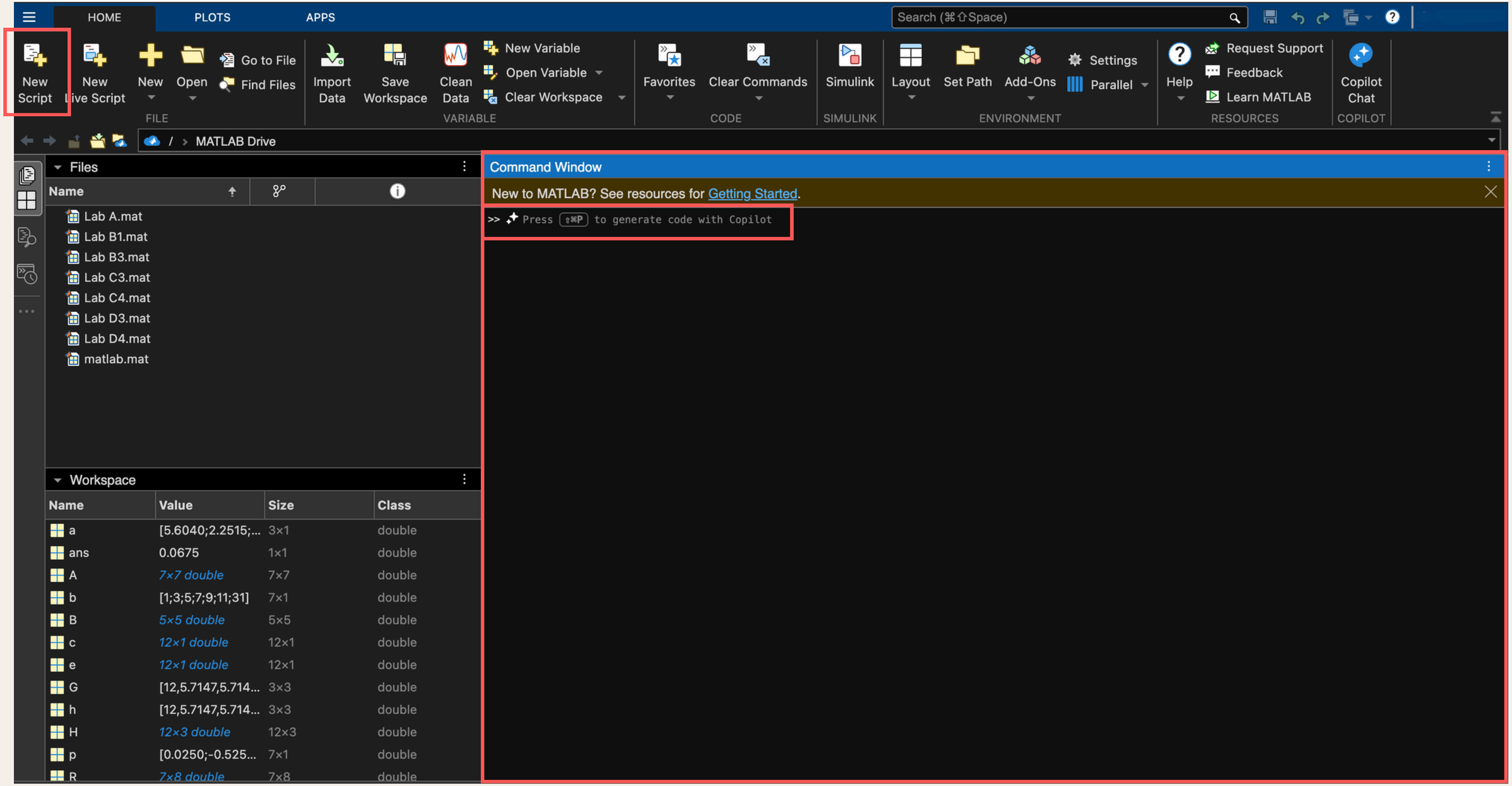
Getting Started with MatLab

MATLAB (Matrix Laboratory) is a high-level programming language and interactive environment from MathWorks. It is used for:

- technical computing
- data analysis
- algorithm development
- visualization
- numerical computation



MATLAB®


HOME
PLOTS
APPS
Search (⌘⇧Space)
New Script
New Live Script
New
Open
Go to File
Find Files
FILE
Import Data
Save Workspace
Clean Data
New Variable
Open Variable
Clear Workspace
VARIABLE
Favorites
Clear Commands
CODE
Simulink
SIMULINK
Layout
Set Path
Add-Ons
Settings
Parallel
ENVIRONMENT
Help
Request Support
Feedback
Learn MATLAB
RESOURCES
Copilot Chat
COPILOT
/ › MATLAB Drive
Files
Name
Lab A.mat
Lab B1.mat
Lab B3.mat
Lab C3.mat
Lab C4.mat
Lab D3.mat
Lab D4.mat
matlab.mat
Workspace
Name
Value
Size
Class
a [5.6040;2.2515;... 3×1 double
ans 0.0675 1×1 double
A 7×7 double 7×7 double
b [1;3;5;7;9;11;31] 7×1 double
B 5×5 double 5×5 double
c 12×1 double 12×1 double
e 12×1 double 12×1 double
G [12,5.7147,5.714... 3×3 double
h [12,5.7147,5.714... 3×3 double
H 12×3 double 12×3 double
p [0.0250;-0.525... 7×1 double
R 7×8 double 7×8 double
Command Window
New to MATLAB? See resources for Getting Started.
>> Press ⇧⌘P to generate code with Copilot

Tips:

- use a semicolon (;) at the end of a line if you don't want MATLAB to print every result immediately
- You can use the [up arrow key] (↑) to recall previously typed commands
- >>clc will clear your command window
- clear will clear all previously assigned variables. You can clear variables individually by typing >>clear variablename
- double clicking the variable name in the workspace window allows you to edit the variable

Statements

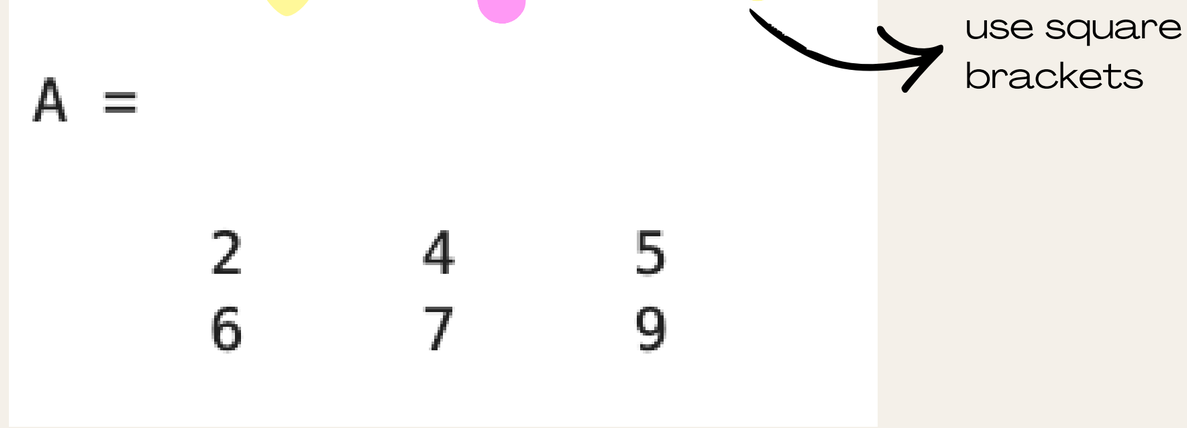
- **Assignment Form:**
  - Statements frequently follow variable = expression, which stores a result for future use
- **Simple Form:**
  - If you enter an expression alone, MATLAB assigns the result to a special default variable called ans (short for answer)
- **Command Help:**
  - Use the command help topic at the >> prompt to view usage information for any specific function

Creating and Managing Matrices

In MatLab, we can key in A = [row 1; row 2; ...; row m] for a mxn matrix

See the example below!

$$A = \begin{pmatrix} 2 & 4 & 5 \\ 6 & 7 & 9 \end{pmatrix}$$

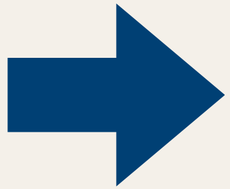


***impt: matlab is case sensitive! 'a' is not the same as 'A'**

Creating and Managing Matrices

(example from handout)

$$\mathbf{A} = \begin{pmatrix} 2 & 4 & 5 \\ 6 & 7 & 9 \\ 7 & 1 & 2 \end{pmatrix}$$



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

A =

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

Matrix Power Tools

`A(i,j)`	the (i,j)-entry of $\mathbf{A}$.
`A(i,:)`	the i^{th} row of $\mathbf{A}$.
`A(:,j)`	the j^{th} column of $\mathbf{A}$.
`A+B`	matrix addition.
`A-B`	matrix subtraction.
`t*A`	scalar multiplication if t is a real number, i.e. $t\mathbf{A}$.
`A*B`	matrix multiplication.
`A^n`	raising a square matrix $\mathbf{A}$ to a positive integral power n, i.e. $\mathbf{A}^n$.
`A'`	transpose of $\mathbf{A}$, i.e. $\mathbf{A}^\top$.
`A.'`	*conjugate* transpose of $\mathbf{A}$, i.e. $\mathbf{A}^*$, That is, the matrix is transposed and all the entries are complex-conjugated
`inv(A)`	inverse of an invertible square matrix A, i.e. $\mathbf{A}^{-1}$.
`zeros(n)`	the $n \times n$ zero matrix, i.e. $\mathbf{0}_{n\times n}$.
`zeros(m,n)`	the $m \times n$ zero matrix, i.e. $\mathbf{0}_{m\times n}$.
`eye(n)`	the $n \times n$ identity matrix, i.e. $\mathbf{I}_n$.
`rref(A)`	the reduced row echelon form of $\mathbf{A}$.
`det(A)`	the determinant of $\mathbf{A}$, i.e. $\det(\mathbf{A})$.

Precision & Numeric Display

Command	Description	Example Output of π	Best Use Case
format short	scaled fixed point format with 5 digits (default setting)	3.1416	Best for general work where you don't need high precision
format long	scaled fixed point format with 15 digits	3.141592653589793	Use this when viewing more digits to inspect results
format shorte	floating point format with 5 digits	3.1416e+00	Best for very large values (like the speed of light) or tiny ones (like the mass of an electron)

Precision & Numeric Display

Command	Description	Example Output of π	Best Use Case
format longe	floating point format with 15 digits	3.141592653589793	Use for extreme precision involving very large or small scales
format rat	approximate fractions.	355/113	When working with ratios, probability, or needing exact fractional representations of data

MATLAB will approximate decimals with rational numbers when you use format rat. Sometimes, this may cause unexpected results. Occasionally, an asterisk * may appear when you expect the quantity to be 0.

Now, let's try out the handout exercises!

(Ex 2.1, 2,2 & 2,3)

Exercise 2.1 — Variables, precedence, and ans

Compute

$$y = \frac{(3+5)^2}{7} - 10 + (2 \times 2).$$

in the MATLAB terminal and store it. Then compute y^2 without assigning it.

Remark. You can also use the `ans` variable as MATLAB always stores the unassigned outputs in `ans`

Solution:

```
>> y = ((3+5)^2)/7 - 10 + 2*2

>> y^2
```

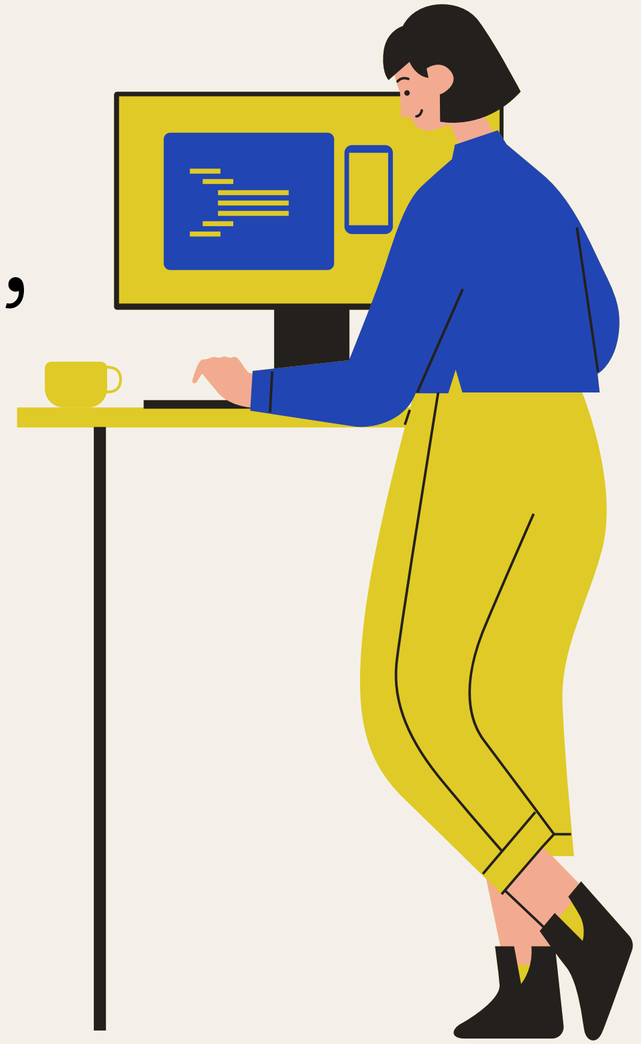
Exercise 2.2 — Rounding and numeric display

Let `x = pi/7`. Display it with 5 digits, 15 digits, scientific notation, and as a rational approximation. Then compute `round(x,4)`.

Solution:

```
x = pi/7;

format short % ~5 digits (default short display)
x

format long % ~15 digits
x

format shortE % scientific notation
x

format rat % rational approximation display
x

format short % switch back (recommended)
round(x,4) % round to 4 decimal places
```

03 - Arithmetic, variables, and built-in functions



Arithmetic Operators & Essential Built-in Functions

function	description
+, -, *, /	plus, minus, multiply, divide
^	power
sqrt(), exp(), log(), and log10()	square root, exponential, log base e, log base 10
sin(), cos(), and tan()	trigo functions (available for scalar or vector inputs)

Let’s try out the handout exercises!

(Ex 3.1–3.5)

Useful Functions for Ex3

1. Indexing (matrix operation)
- 2D indexing for matrices
  - syntax: `matrix(row, column)`
- colon operator (:)
  - generate a list of numbers without typing them all out
    - e.g.: 1:5 → [1,2,3,4,5]
    - e.g.: 1:2:10 → [1, 3, 5, 7, 9]
  - in matrices, it acts as a placeholder for "every index in this dimension"
    - e.g.: A(2, :)

2. `abs()`
- absolute

3. `dot()`
- scalar dot product

Useful Functions for Ex3

4. `norm()`
   - norm of a vector (magnitude or length)
   - uses Euclidean distance

5. `reshape()`
   - changes the dimensions of a matrix without changing its data
   - takes the elements of an existing matrix and change them into a new shape
   - impt: total number of elements must stay exactly the same

6. `acos()`
   - inverse cosine function
   - for finding angles between vectors
   - takes a value (usually between -1 and 1) and returns the angle in radians

Exercise 3.1

Create a vector $\mathbf{v} = (2, 4, 6, 8, 10)^\top$. Extract the second and fourth entries, and replace the last entry with 99.

Solution:

```
>> v = [2; 4; 6; 8; 10];
>> second = v(2);
>> fourth = v(4);
>> v(5) = 99; % you can also use v(end) instead

>> v % verify you have replaced it
```

(matrix operations question)

Exercise 3.2

Create the matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

Extract row 2 and column 3

Solution:

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

>> row2 = A(2,:);

>> col3 = A(:,3);
```

```
>> col3 = A(:,3);
>> col3

col3 =

     3
     6
     9
```

```
>> row2 = A(2,:);
>> row2

row2 =

     4     5     6
```

Exercise 3.3

Make the vector $\mathbf{v} = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)^\top$, reshape into a 3×4 matrix by columns, then transpose it.

Solution:

```
>> v = 1:12
>> M = reshape(v, 3, 4)
>> Mt = M'
```

```
>> v = 1:12

v =

     1     2     3     4     5     6     7     8     9    10    11    12

>> M = reshape(v, 3, 4)

M =

     1     4     7    10
     2     5     8    11
     3     6     9    12

>> Mt = M'

Mt =

     1     2     3
     4     5     6
     7     8     9
    10    11    12
```

Exercise 3.4

Let `u=[1; -2; 3];` and `v=[4; 0; -1];`. Compute the dot product, norms, and angle between them.

Tip: Use format short, format long, format rat to control display.

Solution.

```
u = [1; -2; 3];
v = [4; 0; -1];

dp = dot(u,v);
nu = norm(u);
nv = norm(v);

theta = acos( dp/(nu*nv) ); % radians
theta_deg = theta*180/pi;
```

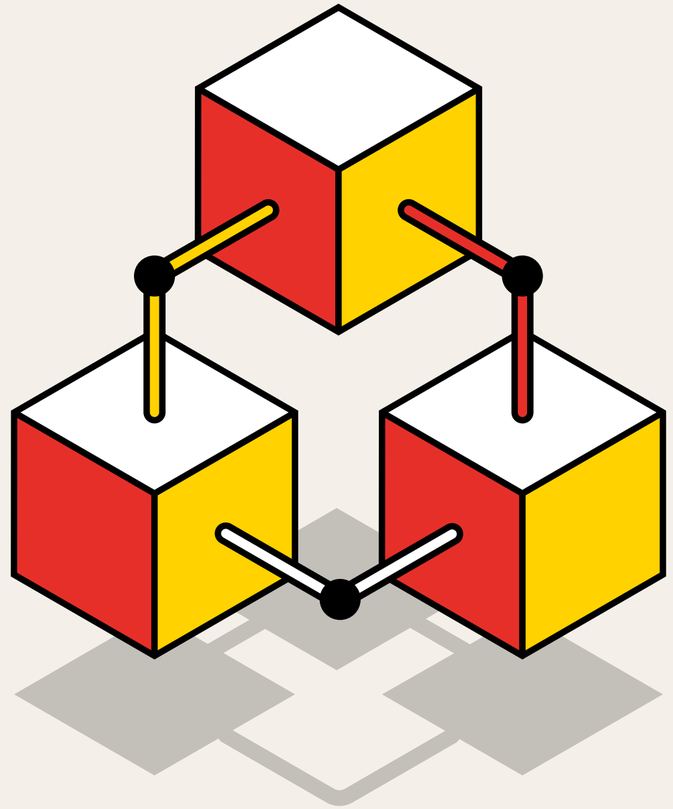
Exercise 3.5

For `t = 2.4`, compute `sin(t)`, `cos(t)`, `tan(t)`, `exp(t)`, `log(t)`, `sqrt(t)`, and verify that $\sin^2(t) + \cos^2(t) \approx 1$.

Solution:

```
>> t = 2.4;

>> sin(t)
>> cos(t)
>> tan(t)
>> exp(t)
>> log(t) % natural log
>> sqrt(t)

>> val = sin(t)^2 + cos(t)^2;
>> err = abs(val - 1) % should be very small (floating-point error)
```

04 - Arrays, Vectors, Matrices and Indexing



Creation

Row Vector: [1 2 3 4];

Column Vector: [1; 2; 3; 4];

Matrix (3x3): [1 2 3; 4 5 6; 7 8 9];

Identity matrix: eye(3)

Zero matrix: zeros(n, m)

Random matrix: rand(n,m)

Indexing and Slicing

Extracting an entry : A(2,3)

Extracting a row 2: A(2,:)

Extracting a col 3: A(:,3)

Extracting a Submatrix: A(1:2,2:3)

Practice question

(Ex 4.1)

Exercise 4.1

Input the following three matrices

$$A = \begin{pmatrix} 7 & -2 & 0 & -2 \\ -3 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 \\ 16 & -3 & 0 & -2 \end{pmatrix}, \qquad B = \begin{pmatrix} -2 & 5 & 17 & -2 \\ 2 & 6 & -15 & -1 \\ -2 & 6 & 19 & -2 \\ 1 & 2 & -6 & 0 \end{pmatrix}, \qquad C = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

(i) Using MATLAB, compute the products $(\mathbf{AB})^{-1}$, $\mathbf{A}^{-1}\mathbf{B}^{-1}$ and $\mathbf{B}^{-1}\mathbf{A}^{-1}$. What is the relation between these matrices?

Solution:

$\mathbf{B}^{-1}\mathbf{A}^{-1} = (\mathbf{AB})^{-1} \neq \mathbf{A}^{-1}\mathbf{B}^{-1}$.

Exercise 4.1

Input the following three matrices

$$A = \begin{pmatrix} 7 & -2 & 0 & -2 \\ -3 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 \\ 16 & -3 & 0 & -2 \end{pmatrix}, \qquad B = \begin{pmatrix} -2 & 5 & 17 & -2 \\ 2 & 6 & -15 & -1 \\ -2 & 6 & 19 & -2 \\ 1 & 2 & -6 & 0 \end{pmatrix}, \qquad C = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

(ii) Compute $\mathbf{C}^{-1}$. What result do you get? Why?

Solution:

Error message, because $\mathbf{C}$ is not invertible

Exercise 4.1

Input the following three matrices

$$A = \begin{pmatrix} 7 & -2 & 0 & -2 \\ -3 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 \\ 16 & -3 & 0 & -2 \end{pmatrix}, \qquad B = \begin{pmatrix} -2 & 5 & 17 & -2 \\ 2 & 6 & -15 & -1 \\ -2 & 6 & 19 & -2 \\ 1 & 2 & -6 & 0 \end{pmatrix}, \qquad C = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

(iii) Compute $(\mathbf{A}+\mathbf{B})^{-1}$ and $\mathbf{A}^{-1}+\mathbf{B}^{-1}$. Are these matrices equal?

Solution:

The matrices are not equal

Bonus activity

Enter the following statements in the MATLAB command window and observe the outputs. Describe what MATLAB has done on the right.

Commands	**Describe the action**
`» x = [ 1  2  3 ]`	create a 1 × 3 row vector
`» b = [ 1;  2;  3 ]`	create a 3 × 1 column vector
`» c = A(:,3)`	extract column 3
`» format rat`	show rational display

05 - Gaussian elimination & solving linear systems

Gaussian Elimination

Ax =b can be solved by Gaussian Elimination (row operations)

In MATLAB:

- `x = A\b;` solves Ax=b
- `rref([A b])` produces reduced row echelon form

Practice question

(Ex 5.1, 5.2)

Exercise 5.1

Consider the following system of linear equations

$$\begin{aligned} x + y + 2z &= 1 \\ 3x + 6y - 5z &= -1 \\ 2x + 4y + 3z &= 0 \end{aligned}$$

1. Enter the matrix $\mathbf{A} = \begin{pmatrix} 1 & 1 & 2 \\ 3 & 6 & -5 \\ 2 & 4 & 3 \end{pmatrix}$ and $\mathbf{b} = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$.
2. Type `inv(A)*b` to get a solution for this system.
3. Type `A\b`. What do you get?
4. Type `rref([A b])` to get the **reduced** row-echelon form of the augmented matrix.
5. Use the three different methods you learnt to solve the following system:

$$\begin{aligned} x + 2y + z &= 1 \\ x + 2y + 2z &= 1 \\ 2x + 4y + z &= 2 \end{aligned}$$

Solution:

```
A = [1 1 2; 3 6 -5; 2 4 3];
b = [1; -1; 0];

x_inv = inv(A)*b
x_bs = A\b
rref([A b])
```

Unique solution:

$$(x, y, z) = \left(\frac{33}{19}, -\frac{18}{19}, \frac{2}{19}\right).$$

For the second system (singular matrix), `inv` fails, but `\` and `rref` show infinitely many solutions:

$$(x, y, z) = (1 - 2t,\ t,\ 0), \quad t \in \mathbb{R}.$$

Exercise 5.2

(a) Consider the following system of linear equations

$$\begin{aligned} 3x_1 + 6x_2 + 9x_3 + 5x_4 + 25x_5 &= 53 \\ 7x_1 + 14x_2 + 21x_3 + 9x_4 + 53x_5 &= 105 \\ -4x_1 - 8x_2 - 12x_3 + 5x_4 - 10x_5 &= 11. \end{aligned}$$

Let $[\mathbf{M} \ \ \mathbf{w}]$ be the augmented matrix representing the system. Clearly $\mathbf{M}^{-1}$ does not exist. Does `M\w` yield a solution? How do you confirm that this is indeed a solution for the system?

(Remark: You are not required to know how matrix left division "\" works for this course. You may type "`help mldivide`" in MATLAB to find out more about this operator.)

(b) Find a general solution to this system using `rref([M w])`.

Solution:

```
M = [ 3 6 9 5 25;
      7 14 21 9 53;
     -4 -8 -12 5 -10];
w = [53; 105; 11];

x = M\w
res = norm(M*x - w)
R = rref([M w])
```

General solution (free parameters x_2, x_3, x_5):

$$x_1 = 6 - 2x_2 - 3x_3 - 5x_5, \qquad x_4 = 7 - 2x_5.$$

06 - Gram–Schmidt orthonormalization

Objective

Given independent vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \mathbb{R}^n$, Gram–Schmidt constructs orthonormal vectors $\mathbf{q}_1, \ldots, \mathbf{q}_k$ spanning the same subspace.

Formulae

For $j \geq 2$, $\mathbf{u}_j, \mathbf{v}_j, \mathbf{q}_i \in \mathbb{R}^n$

$$\mathbf{u}_j = \mathbf{v}_j - \sum_{i=1}^{j-1} (\mathbf{v}_j \cdot \mathbf{q}_i)\, \mathbf{q}_i, \qquad \mathbf{q}_j = \frac{\mathbf{u}_j}{\|\mathbf{u}_j\|}$$

where $\cdot$ denotes the dot product between two vectors.

Practice question

(Ex 6.1)

Exercise 6.1 (Needs some Linear Algebra knowledge)

Orthonormalize the columns of the matrix

$$\mathbf{V} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

using Gram-schmidt. Compare the output with `orth(V)`.

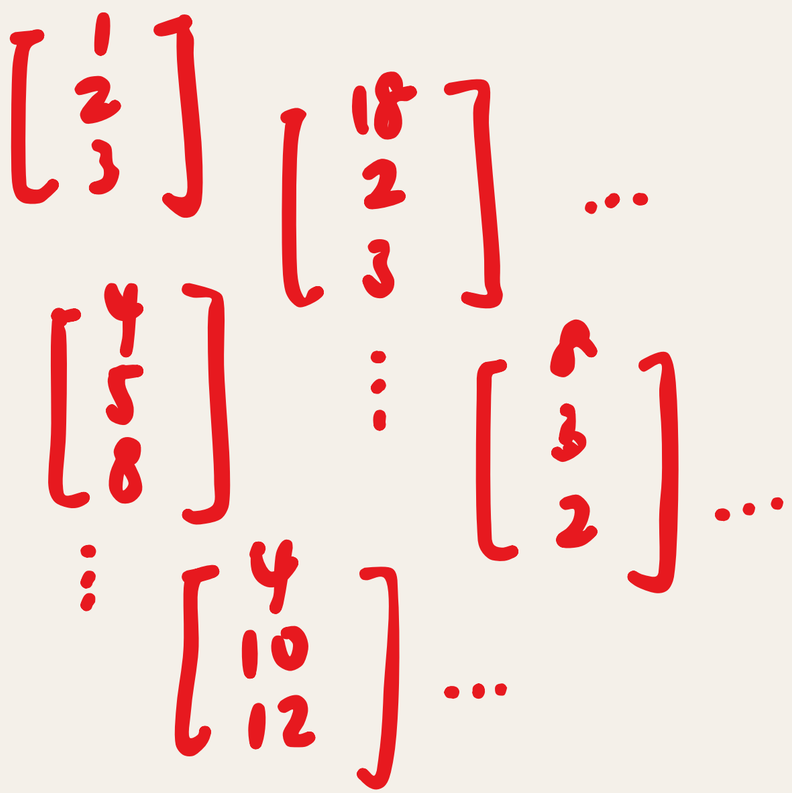
Solution:

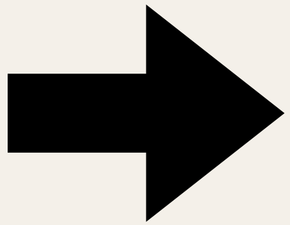
```
V = [1 1; 1 0; 0 1];

v1 = V(:,1);
e1 = v1 / norm(v1);

v2 = V(:,2);
u2 = v2 - dot(v2,e1)*e1;
e2 = u2 / norm(u2);

Q = [e1 e2];
Q'*Q
orth(V)
```

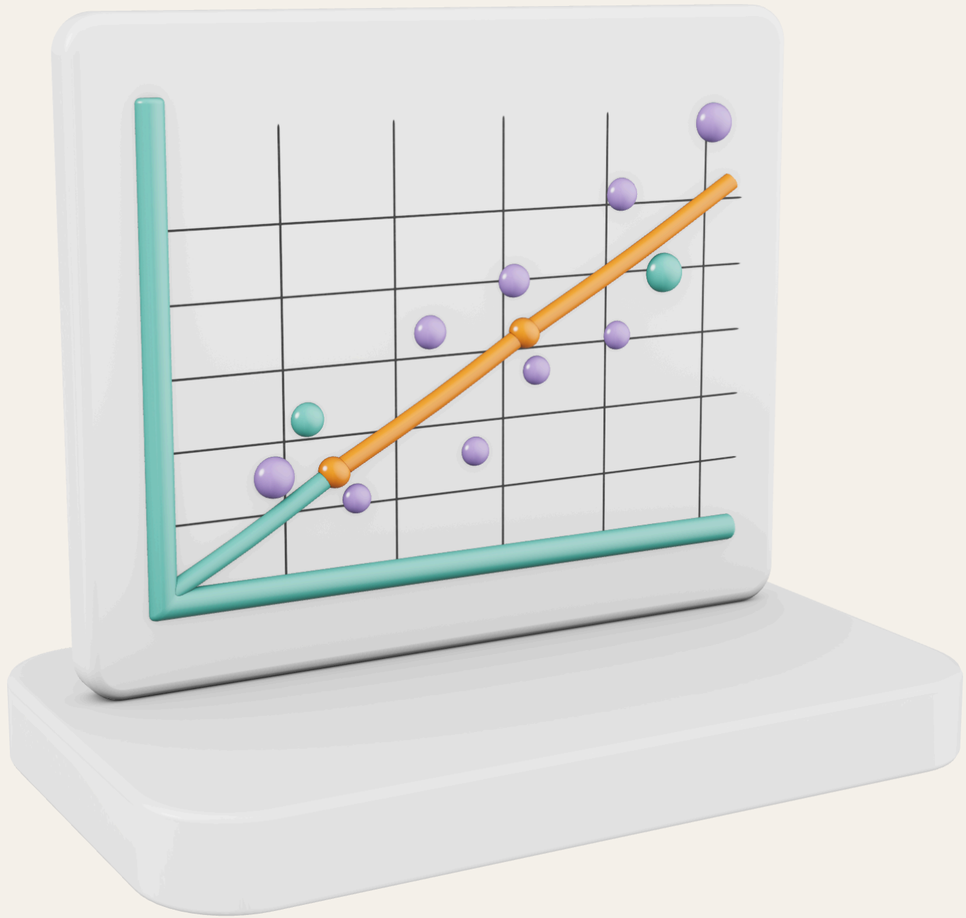
$$e_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \qquad e_2 = \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix}.$$

07 - Data Visualization and Logic

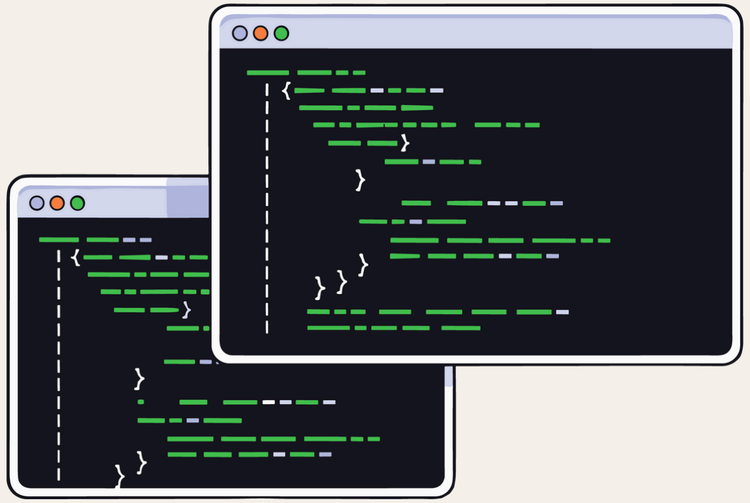
Why Plotting Matters



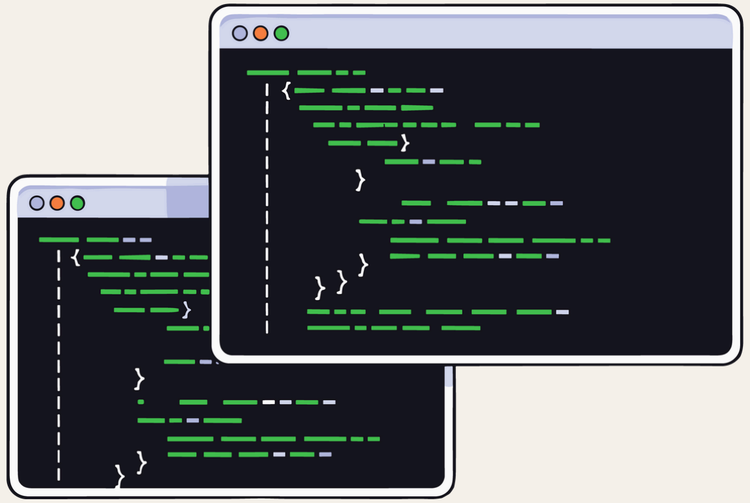




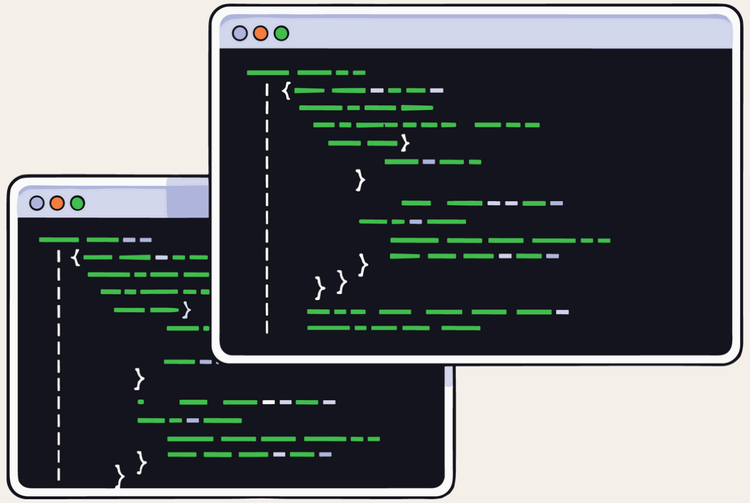
But why not python/R?



Workflow



Basic Plotting in MATLAB



Basic Plotting

1. `plot()`: creates 2D line plot of data to visualize the relationship between two sets of data
   a. X and Y: vectors of the same length
   b. LineSpec: line style, marker, colour

Line Style	Description	Resulting Line
"-"	Solid line	————
"--"	Dashed line	- - - - -
":"	Dotted line	.....
"-."	Dash-dotted line	-.-.-.-.-

Marker	Description	Resulting Marker
"o"	Circle	○
"+"	Plus sign	+
"*"	Asterisk	✳

   c. LineWidth: Name-Value argument that tells MATLAB you want to modify the thickness of the line
      i. must be typed out as a "Name" followed by a "Value"

```
>> plot(x, y1, 'LineWidth', 3)
>> plot(x, y2, 'LineWidth', 2)
```

Basic Plotting

2. `xlabel()` & `ylabel()`: add text descriptions to the horizontal and vertical axes of your plot

xlabel('text')	Labels the horizontal axis (X-axis)
ylabel('text')	Labels the vertical axis (Y-axis)

Basic Plotting

3. `title('text')`: places a header at the top of your plot to explain what the overall visualization represents – ie. title of the plot

4. `legend(LabelStrings, 'Location', 'Best')`:
   a. Label Strings: must provide one string for every line you have plotted (e.g. `legend('sin(x)', 'cos(x)')`
   b. specifies where the legend box sits on the axes – common values: '**best**', 'northeast' etc.
   c. refer to your handout for more arguments (p. 15-17)

Tip: always use 'Location', 'Best' in legend so that MATLAB finds the best location to input the legend!

Basic Plotting

(example from handout)

generate a vector of linearly spaced points between two specific values

```
>> x = linspace(-2*pi, 2*pi, 400);
y1 = sin(x);
y2 = cos(x);
plot(x,y1,'LineWidth',1.5); hold on;
plot(x,y2,'LineWidth',1.5);
xlabel('x'); ylabel('y');
title('Sine and Cosine');
legend('sin(x)','cos(x)','Location','best');
grid on; hold off;
```

use plot() to start plotting

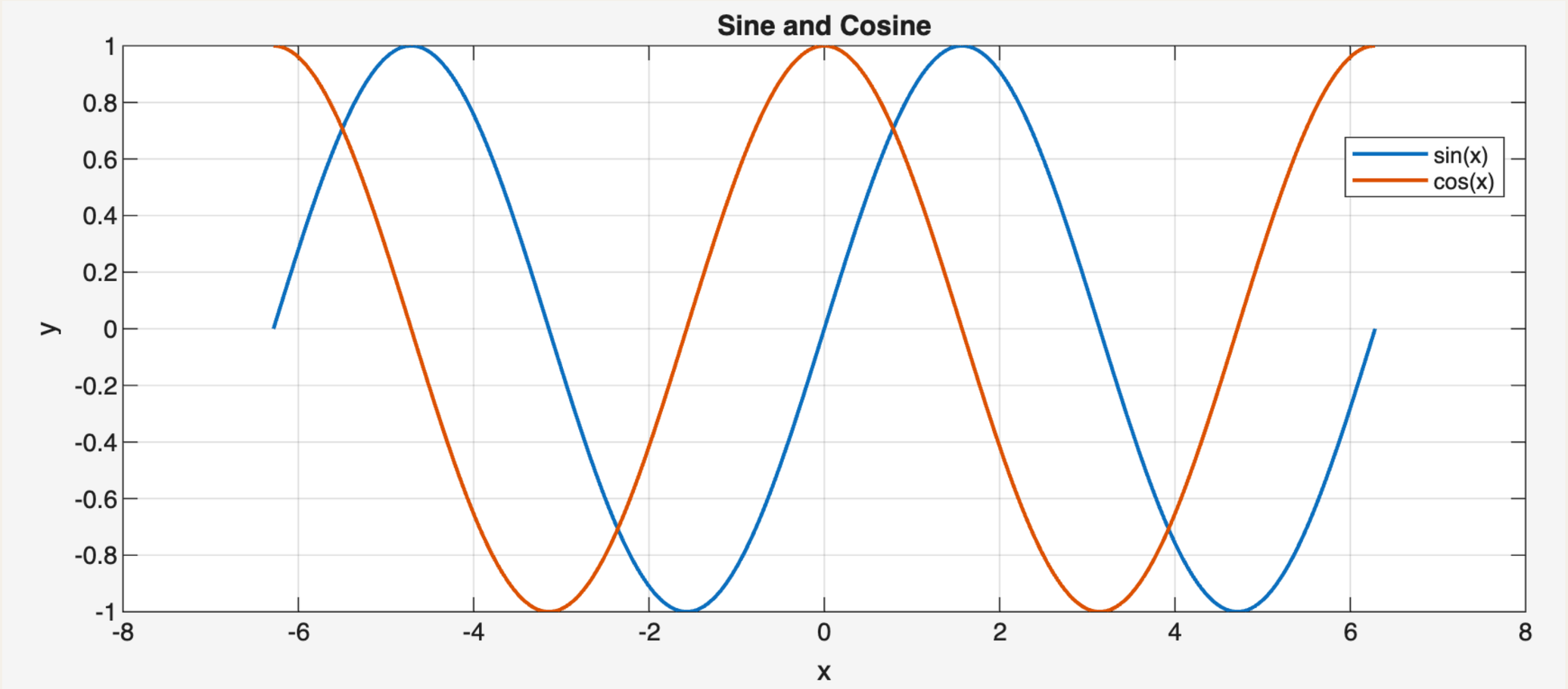
retains current plot
→ Both Sine and Cosine appear together

Discards current plot for new ones
→ Prevents future lines from cluttering this graph

overlays a grid that aligns with the "ticks" on your X and Y axes
→ adds the faint vertical and horizontal lines in the background of the graph

Basic Plotting

(example from handout)



Basic Plotting

(warm up from handout)

Plot $y = x^2$ and $y = 2x + 1$ on the same axes for $x \in [-3, 3]$. Include labels, title, legend, and a grid.

`grid on`

`title()`

`legend`

`xlabel, ylabel`

used to generate a vector of linearly spaced points between two specific values

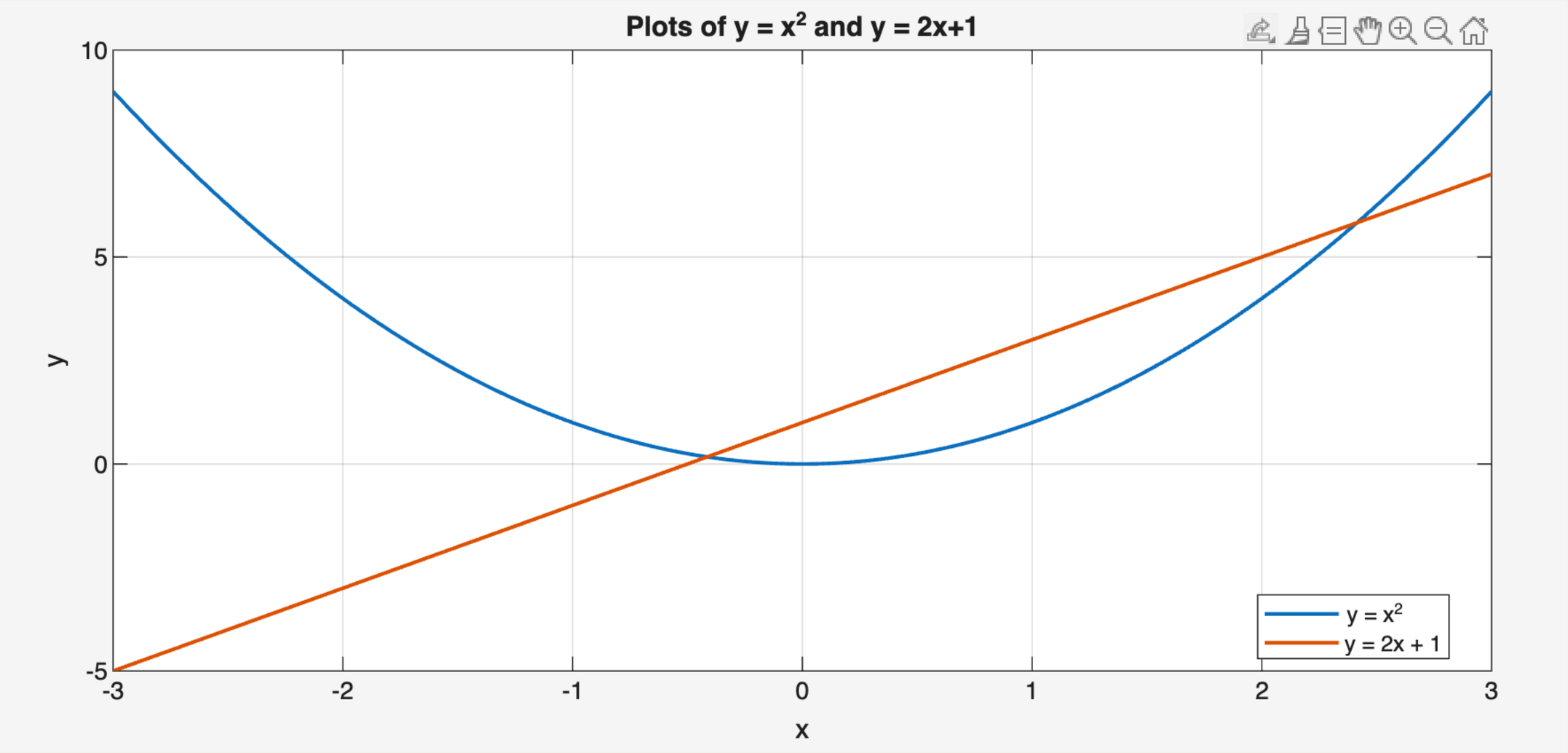
```
x = linspace(-3, 3, 400);
y1 = x.^2;
y2 = 2*x + 1
plot(x, y1, 'LineWidth', 1.5);
hold on;
plot(x, y2, 'LineWidth', 1.5);

grid on;
xlabel('x'); ylabel('y');
title('Plots of y = x^2 and y = 2x+1');
legend('y = x^2', 'y = 2x + 1', 'Location', 'best');

xlim([-3 3]);
```

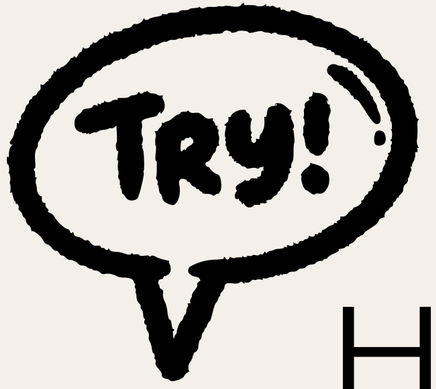
- calculate 400 individual points between those boundaries.
- high number like 400 ensures that the curves look smooth rather than jagged

"crop" the graph window so that it only displays values between -3 and 3

Basic Plotting

(warm up from handout)





Hands-on Activity!

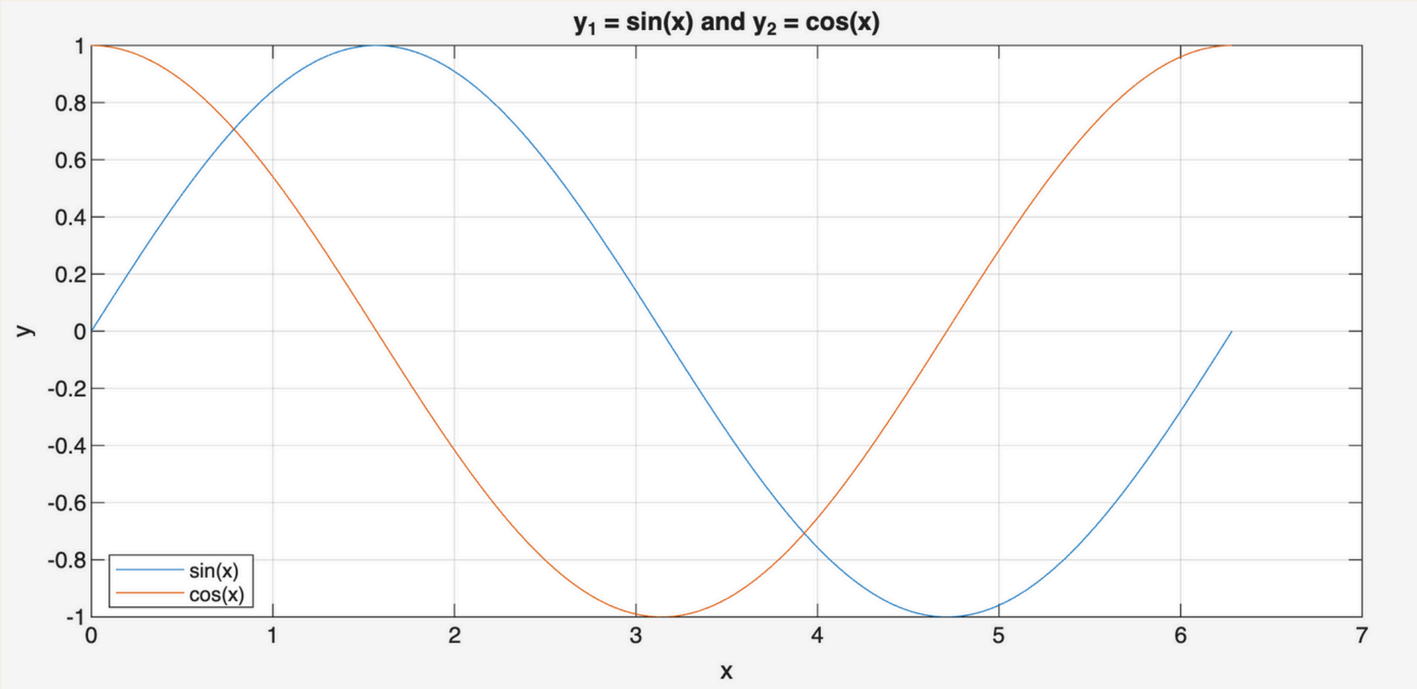
(Ex 7.1 & 7.2)

Exercise 7.1

Plot the curves $y_1 = \sin x$ and $y_2 = \cos x$ for $0 \leq x \leq 2\pi$. Add title, labels, legend and grid.

Solution.

```
x = linspace(0, 2*pi, 500);
y1 = sin(x);
y2 = cos(x);

plot(x, y1, x, y2);
grid on;
xlabel('x'); ylabel('y');
title('y_1 = sin(x) and y_2 = cos(x)');
legend('sin(x)','cos(x)','Location','best');
```



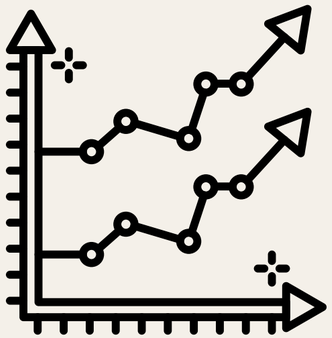
Slightly Advanced Plotting

(useful for the next exercise!)

1. `randn()`
   - generate random numbers that follow a **standard normal distribution**
   - used to add noise → can test model robustness

2. `polyfit(x, y, n)`
   - finds the coefficients for a polynomial of degree ‘n’ that best fits your data points (x,y) in a least-squares sense
   - e.g: for linear models, n = 1

3. `polyval(p, x):`
   - uses the coefficients found by polyfit to calculate the predicted y-values for a given x
   - used to draw the smooth best fit line over your noisy scatter plot



Other Plots

1. Histogram
   - `histogram(x)`

2. Scatter Plot
   - `scatter(x, y)`

3. Bar Plot
   - `bar(y)`
   - `bar(x, y)`

https://www.mathworks.com/help/matlab/creating_plots/types-of-matlab-plots.html

Exercise 7.2

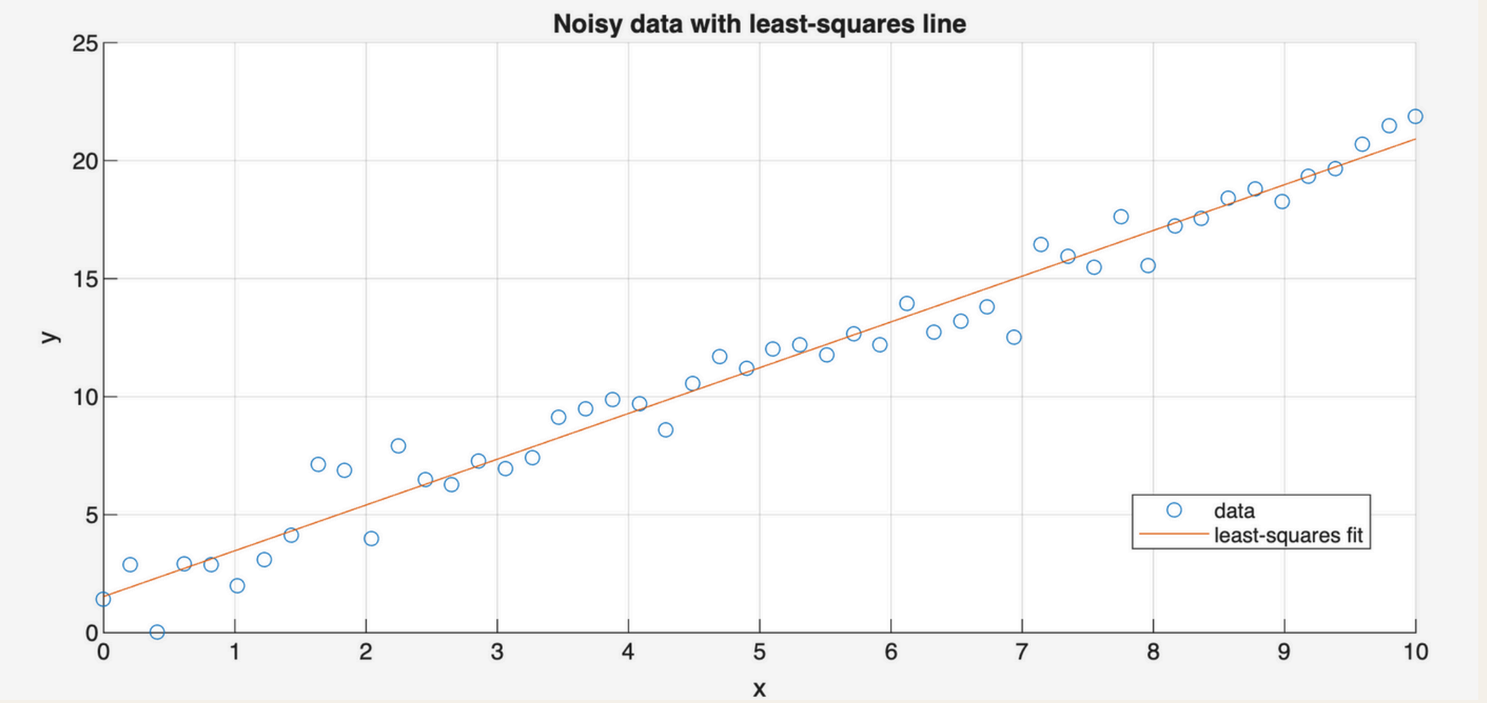
Generate noisy data $y \approx 2x + 1$, scatter it, and overlay the least-squares line.

scaling the noise:

- 0.8: multiplier to scale the noise
- size(x): creates exactly enough random numbers to match your data points

Solution.

```
x = linspace(0, 10, 50).';
y = 2*x + 1 + 0.8*randn(size(x));

scatter(x,y); grid on; hold on;

p = polyfit(x,y,1);
yfit = polyval(p,x);
plot(x,yfit);

xlabel('x'); ylabel('y');
title('Noisy data with least-squares line');
legend('data','least-squares fit','Location','best');
```



Logical Operations & Basic Control Flow

(example from handout)

initialisation
(we fix value of a=3)

```
>> a = 3; if a > 0
       disp('positive');
elseif a == 0
       disp('zero');
else
       disp('negative');
end
positive
```

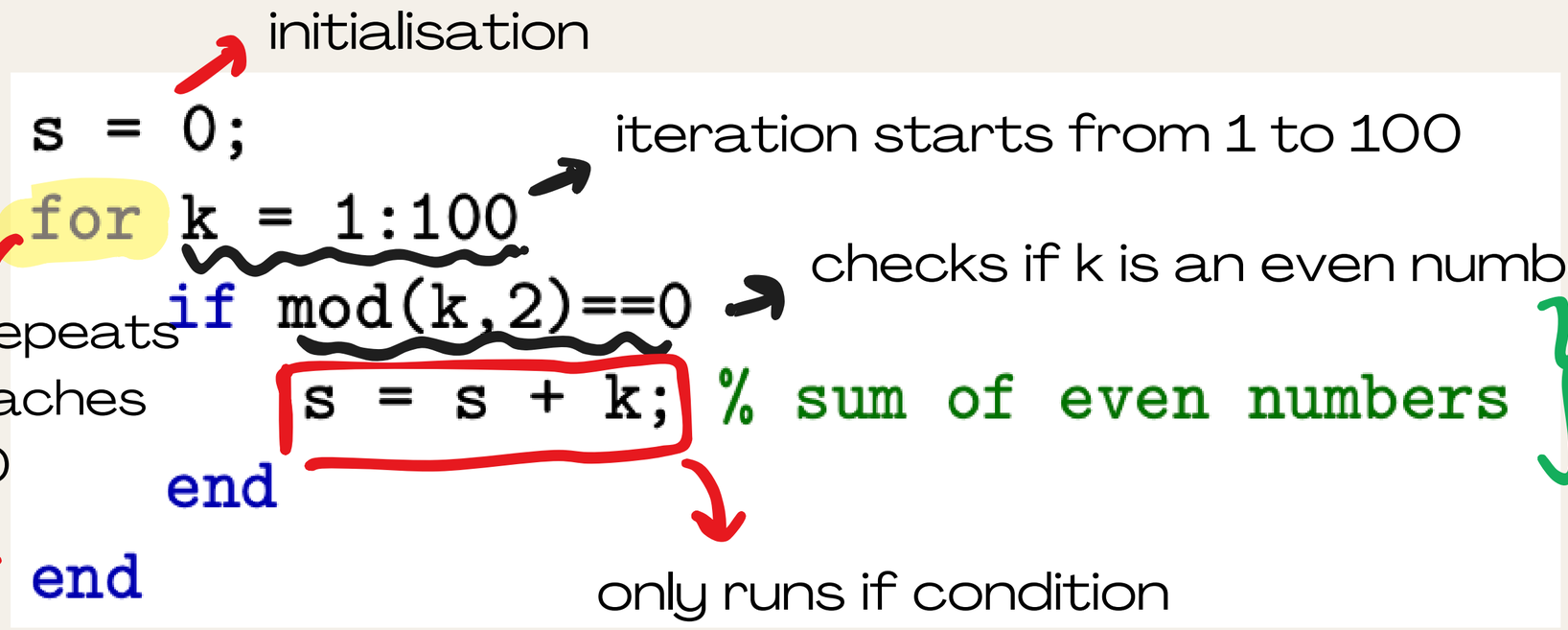
a =3, => display should show 'positive'

only if a=0

runs for any case **except** a>0 & a=0 (ie. a<0)

Logical Operations & Basic Control Flow

(example from handout)

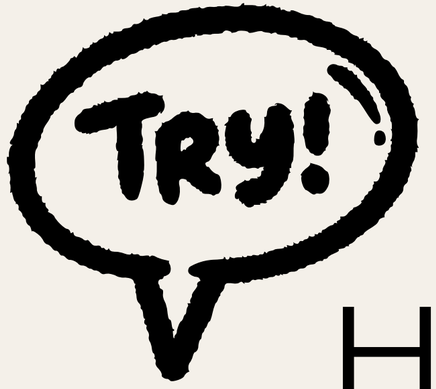


Logical Operations & Basic Control Flow

(cont.)

```
s = 0;
for k = 1:100
    if mod(k,2)==0
        s = s + k; % sum of even numbers
    end
end
```

Iteration	`k` value	`mod(k,2)==0`?	Action	New `s` value
1	1	No	Skip	0
2	2	Yes	`0 + 2`	2
3	3	No	Skip	2
4	4	Yes	`2 + 4`	6
…	…	…	…	…
100	100	Yes	Add 100	2550



Hands-on Activity!

(Ex 8.1–8.4)

Exercise 8.1

Compute $n!$ using a loop (don't use `factorial`).

Solution.

```
n = 10;
fact = 1;
for k = 1:n
    fact = fact * k;
end
fact
```

Exercise 8.2

Definition 1. (Newton-Raphson method) Let $f : \mathbb{R} \to \mathbb{R}$ be a differentiable function. Suppose we want to find a root of f, i.e., a solution of $f(x) = 0$ Starting from an initial guess x_0 with $f'(x_0) \neq 0$, define iteratively the next guess by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n = 0, 1, 2, \ldots$$

iteration formula

→ $x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$

Geometrically, x_{n+1} is the x-intercept of the tangent line to $y = f(x)$ at $x = x_n$. If $x_n \to r$ and $f'(r) \neq 0$, then r is a root of f.

Approximate $\sqrt{a}$ using the Newton-Raphson method until the change is below `1e-8`.

↳ objective: trying to find $\sqrt{a}$

→ function: $f(x) = x^2 - a$

stopping point
(when change is less than 10^-8)

While Loops

- creates an infinite loop that only stops when the break command is triggered
- useful when you don't know exactly how many steps it will take to reach the answer

Solution.

Use the definition to key in the formula into a conditional while-loop:

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right).$$

for this example, we will choose a = 10 for demonstration and verification

```
a = 10;
x = a;
tol = 1e-8;

while true
    xnew = 0.5*(x + a/x);
    if abs(xnew - x) < tol
        break
    end
    x = xnew;
end

xnew
sqrt(a)
```

initial starting guess

tolerance: 0.00000001

while loop

update x:
overwrites the "old" guess with the "new" guess

Exercise 8.3

Create 20 random numbers, then extract only those > 0.5 and set those < 0.2 to 0.

1

2

↳ can use x(...): parentheses tell MATLAB to look inside the vector x and grab only the values that met your rule

```
% 1. Create 20 random numbers
x = rand(20,1);

% 2. Extract only those > 0.5
pos = x(x > 0.5);

% 3. Set those < 0.2 to 0
x(x < 0.2) = 0;
```

Exercise 8.4

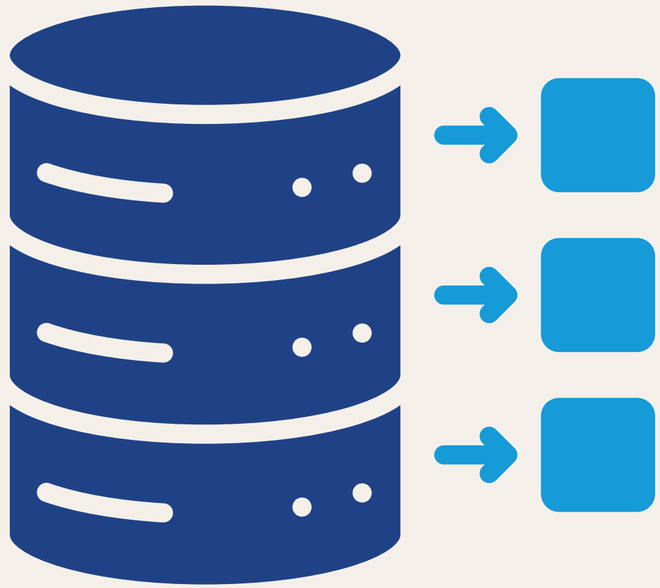
Write a script that:

1. creates a random vector `r` of length 20;
2. counts how many entries in `r` are > 0.5;
3. replaces entries ≤ 0.5 by 0 and entries > 0.5 by 1.

Solution.

```
% (1) random vector of length 20
r = rand(20,1);

% (2) count entries > 0.5
count = sum(r > 0.5);

% (3) threshold to 0/1
r(r <= 0.5) = 0;
r(r > 0.5) = 1;

count
r
```

08 - Data Science with MATLAB



Data Science

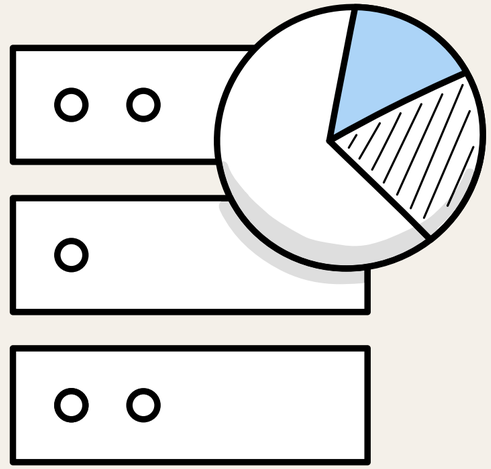
1. Powerful for Data Manipulation

- because MATLAB is a matrix based language
- in data science, dealing with arrays of data is commonplace
- => MATLAB simplifies operations that other languages would require complex loops and data structure manipulation

Data Science

2. Data Analysis and Visualization

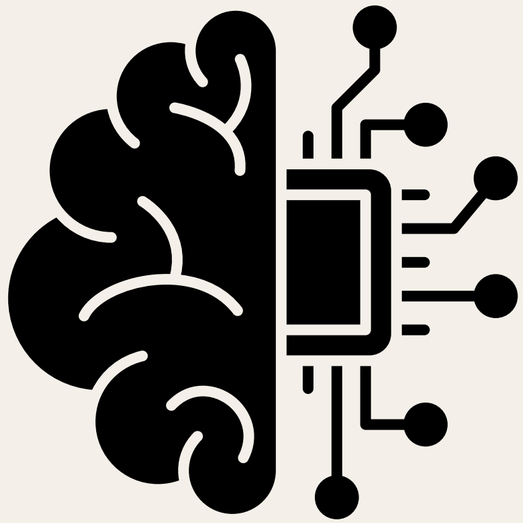
- MATLAB functions make it possible to perform complex data analysis tasks with relatively simple commands
- enable users to create clear and informative graphics that can help both in understanding the data and in presenting findings to others



Data Science

3. ML and DL

- offers built-in apps for designing, training, and evaluating machine learning models, which can significantly speed up the development process
- supports the creation and training of neural networks with just a few lines of code



09 - Final Challenge!

You are testing a simple vibration sensor on your device (e.g., an IMU-derived vibration metric or a piezo/vibration pickup). Assume that when the device is *not vibrating*, the sensor readings fluctuate around an unknown baseline value μ due to noise.

When the device is stationary, each sensor reading is modeled as

$$y_1, \ldots, y_n \sim \mathcal{N}(\mu, \sigma^2),$$

where σ is known (from the sensor datasheet or a previous calibration), and μ is the unknown baseline offset.

(a) Simulate $n = 200$ readings with `randn(200,1)`. The model that fits the data is proposed to be

$$y = 0.02 + 0.01\varepsilon, \qquad \varepsilon \sim \mathcal{N}(0,1).$$

Here, $\mu \approx 0.02$ is a small baseline offset, and $\sigma = 0.01$ is the noise level.

(b) Compute the maximum likelihood estimate (MLE) of the baseline:

$$\hat{\mu} = \frac{1}{200}\sum_{i=1}^{200} y_i.$$

Interpret $\hat{\mu}$ as the sensor offset you would subtract from future readings.

```
% Given
n = 200;
sigma = 0.01; % known noise std
mu_true = 0.02; % baseline offset in the proposed model
h = 1e-4; % step size for numerical second derivative

%% part (a)
y = mu_true + sigma * randn(n,1);

%% part (b) MLE of mu is the sample mean
mu_hat = mean(y);
```

Say you want to define a function that you can use later in MATLAB at your own convenience. But you can't exactly key in a function as MATLAB will only compute and output values. What can you do?

Answer: Use the concept of anonymous functions.

In MATLAB, defining an anonymous function is fairly straighforward.

If you want to define a function of one variable of, say $f(x) = \frac{1}{x}$ simply key in

```
>> f = @(x) 1/x
```

Once done, you are able to use this function as you please. For instance, say you wish to find the value of $\log\left(\frac{1}{\pi}\right)$

```
>> log(f(pi))
```

You may also key in functions of more than one variable. For instance, suppose you wish to define

$$h\left(x,y,z\right)=\sqrt{2x+\tan\left(\frac{y}{z}\right)}$$

You can simply key in

```
>> h = @(x,y,z) sqrt( 2*x + tan(y/z) )
```

Try it out for yourself before attempting the next part!

(c) Define the *log-likelihood function*

$$\ell(\mu) := -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu)^2,$$

using an *anonymous function* (also known as a lambda function).

```
%% part (c)
loglike = @(m) -(1/(2*sigma^2))*sum((y - m).^2);
```

(d) Choose a small step size, e.g. $h = 10^{-4}$, and approximate the second derivative at $\hat{\mu}$ by

$$\ell''(\hat{\mu}) \approx \frac{\ell(\hat{\mu}+h) - 2\ell(\hat{\mu}) + \ell(\hat{\mu}-h)}{h^2}.$$

Define the *observed information* as:

$$\mathcal{J}(\hat{\mu}) := -\ell''(\hat{\mu}).$$

(e) Use the curvature-based approximation

$$\widehat{\mathrm{Var}}(\hat{\mu}) \approx \frac{1}{\mathcal{J}(\hat{\mu})}, \qquad \widehat{\mathrm{SE}}(\hat{\mu}) = \sqrt{\widehat{\mathrm{Var}}(\hat{\mu})}.$$

Report $\hat{\mu}$ and $\widehat{\mathrm{SE}}(\hat{\mu})$

```
%% part (d)
ell_pp = ( loglike(mu_hat + h) - 2*loglike(mu_hat) + loglike(mu_hat - h) ) / h^2;
ell_pp % display 2nd derivative

% Observed information
J = -ell_pp;

J % display jacobian

%% part (e)
var_hat = 1 / J;
se_hat = sqrt(var_hat);

% display estimated variance and standard error
var_hat
se_hat
```

(f) A common engineering rule in Data Science is, if a future reading exceeds

$$\hat{\mu} + 3\,\widehat{\mathrm{SE}}(\hat{\mu}),$$

then the measurement is likely not just baseline noise (it suggests real vibration or a changed operating condition). Compute this threshold and state it clearly.

(g) The theoretical standard error of the mean is $\sigma/\sqrt{n}$. Compare your $\widehat{\mathrm{SE}}(\hat{\mu})$ with $\sigma/\sqrt{200}$ in MATLAB.

```
%% part (f)
threshold = mu_hat + 3*se_hat;

% display threshold
threshold

%% part (g)
se = sigma / sqrt(n);

% display SE
se

% show absolute difference
abs(se_hat - se)
```

ATTENDANCE & FEEDBACK FORM

Please help fill in so we can improve future workshops for you!

SLIDES WILL ALSO BE ON

Github

https://github.com/NUS-SDS-Workshops/AY-25-26-Public

All SDS workshops code and slides will be on there. Do consider ⭐ starring ⭐ to stay updated!

MORE WORKSHOPS COMING THIS SEMESTER 🥳

Week 5 (Thu): Product Club - Product Analytics A/B Testing

Week 6: Azure Cloud Platform for Data Science

Week 9: Collab with AI Society - RL/NLP Workshop

Week 10: MEGA WORKSHOP (will reveal soon)

So follow our telegram @nussds or instagram @nus.sds to stay updated!

Thank you!