ENGR I1100
Engineering Analysis

INTRO TO MATLAB

Sept 17, 2025

Rahul Pandare
PhD Candidate
Dept. of Chemical Engineering
rpandar000@citymail.cuny.edu

Why MATLAB?

- Optimized for Matrices, vectors and equations – Ideal for Engineering problems

- Easy to solve ODEs, own integrated development environment (IDE) – Unlike Python or C

Why MATLAB?

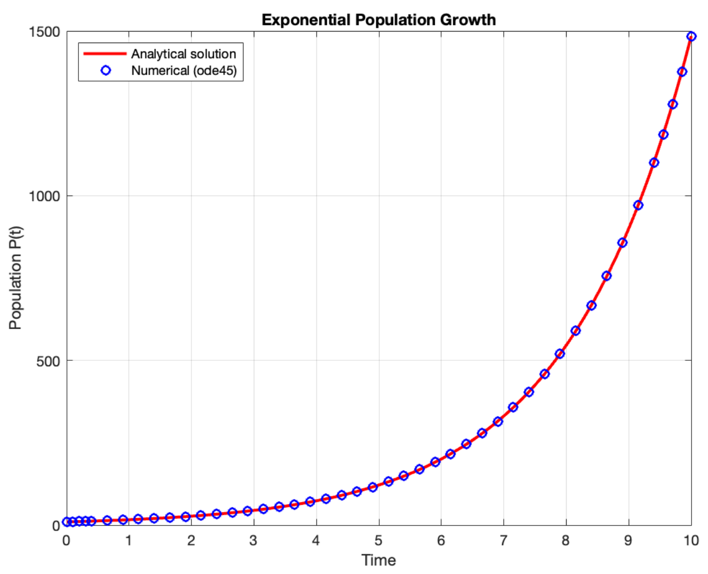
- Optimized for Matrices, vectors and equations – Ideal for Engineering problems

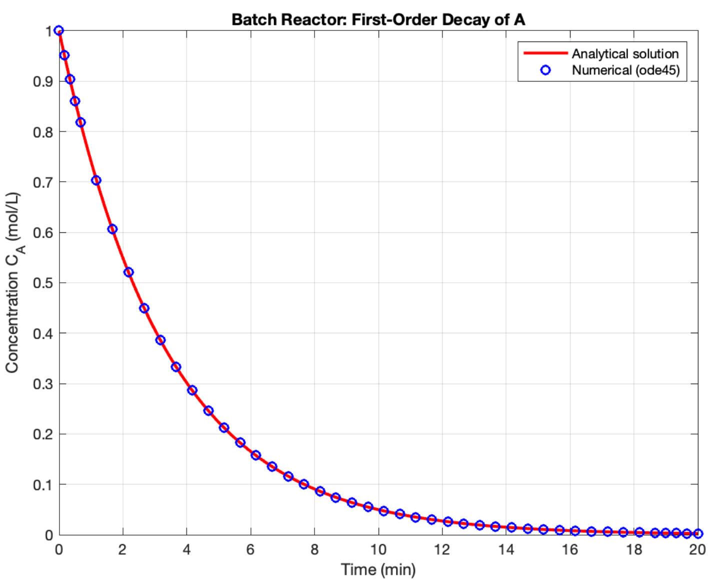
- Easy to solve ODEs, own integrated development environment (IDE) – Unlike Python or C

Examples:

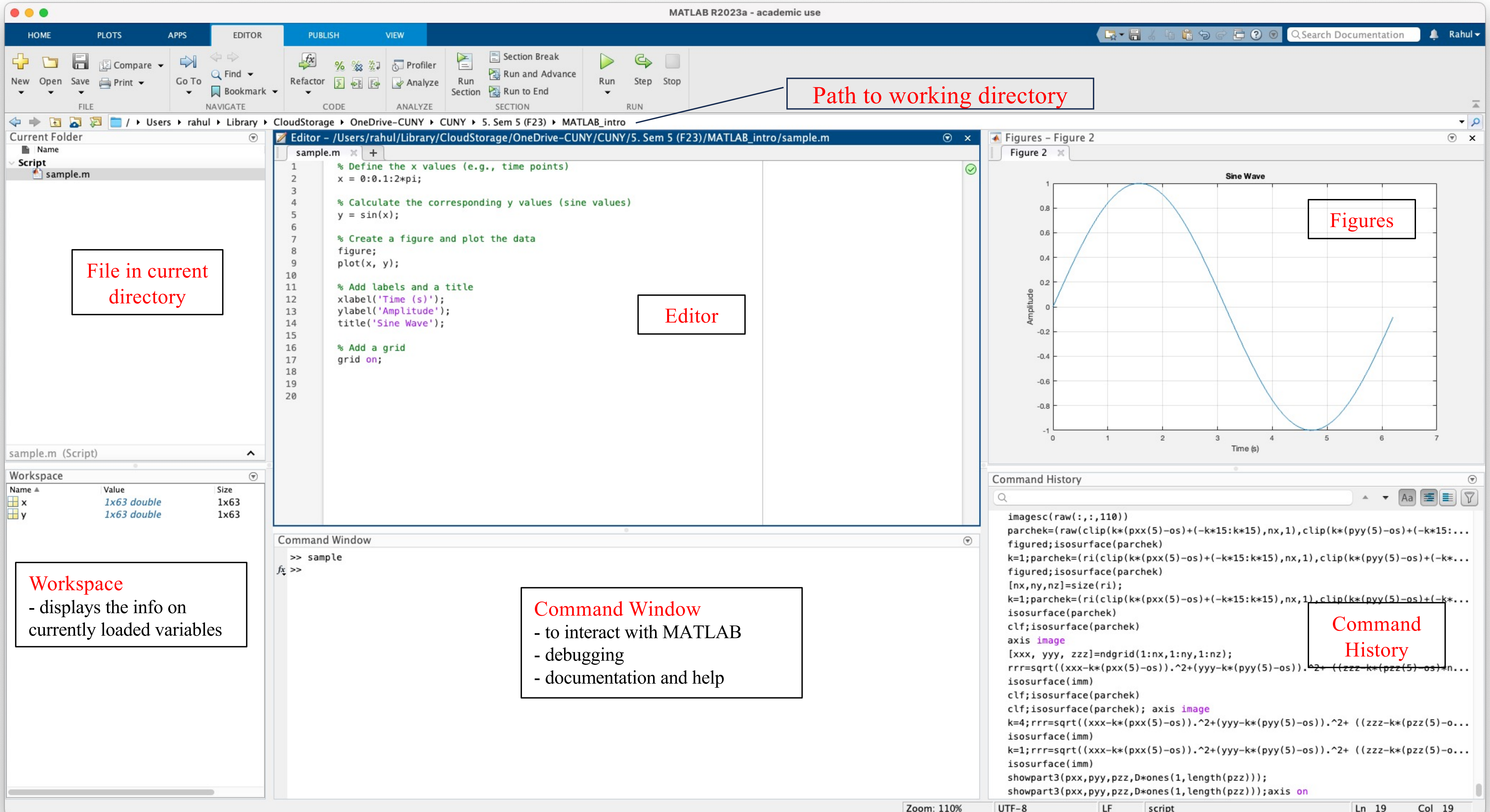
$$\frac{dP}{dt} = rP, \quad P(0) = P_0$$

$$A \longrightarrow \text{Products}, \quad \frac{dC_A}{dt} = -kC_A, \quad C_A(0) = C_{A0}$$





MATLAB screen overview



MATLAB syntax

Variables

Command Window

```
>> 1+2*3

ans =

     7

>> x=1+2*3

x =

     7

>> x=1+2*3;
>>
```

(adding semicolon does not print the output)

Using command window as a calculator

Expressing variable as a function

$$y = e^{-a} * sin(\pi/x^2) + 10 * \sqrt{y}$$

Command Window

```
>> a=5;x=2;y=8;
>> y=exp(-a)*sin(pi/x^2)+10*sqrt(y)

y =

   28.2890
```

Note:
In programming we use numerical variables which are different than symbolic variables used in conventional math

Table 2.1: Elementary functions

cos(x)	Cosine	abs(x)	Absolute value
sin(x)	Sine	sign(x)	Signum function
tan(x)	Tangent	max(x)	Maximum value
acos(x)	Arc cosine	min(x)	Minimum value
asin(x)	Arc sine	ceil(x)	Round towards $+\infty$
atan(x)	Arc tangent	floor(x)	Round towards $-\infty$
exp(x)	Exponential	round(x)	Round to nearest integer
sqrt(x)	Square root	rem(x)	Remainder after division
log(x)	Natural logarithm	angle(x)	Phase angle
log10(x)	Common logarithm	conj(x)	Complex conjugate

Matrix

Command Window

```
>> v=[1 3 5 8 12 13] % Row vector

v =

     1     3     5     8    12    13

>> w=[1;6;9;12;20;21] % Column vector

w =

     1
     6
     9
    12
    20
    21

>> v' % transpose of a vector

ans =

     1
     3
     5
     8
    12
    13
```

comment

Command Window

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

A =

     1     2     3
     4     5     6
     7     8     9

>> A(2,1) % extracting an element from a matrix

ans =

     4

>> B = randi([0,100],5,8) % creating a 5x8 matrix with random integers between 0-100

B =

    44    49    27    50    75    96    84    35
    38    45    68    96    25    55    25    19
    77    65    66    34    51    14    82    25
    80    71    16    59    70    15    24    62
    18    76    12    22    89    26    93    47

>> C = B(2:end,3:7) % slicing the boxed part from the B matrix

C =

    68    96    25    55    25
    66    34    51    14    82
    16    59    70    15    24
    12    22    89    26    93

>> B % the orignal matrix B remains unaltered

B =

    44    49    27    50    75    96    84    35
    38    45    68    96    25    55    25    19
    77    65    66    34    51    14    82    25
    80    71    16    59    70    15    24    62
    18    76    12    22    89    26    93    47
```

<u>Important</u>:
in MATLAB indexing starts from 1

Matrix operations

```
>> x=0:0.1:5; % vector from 0-5 with increament of 0.1
>> length(x)

ans =

    51

>> y=linspace(0,10,100); % linspace creates a vector from 0-10 with 100 subintervals
>> length(y)

ans =

   100
```

Colon operator

linspace operator

```
>> a=[1 2 3 4];b=[10 11 12 13];c=[5;6;7;8]; % a and b row vectors and c column vector
>> a*c %matrix multiplication

ans =

    70

>> a*b %incompatible dimensions for matrix multiplication
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

>> a.*b % elementwise multiplication

ans =

    10    22    36    52

>> b./a %elementwise division

ans =

   10.0000    5.5000    4.0000    3.2500
```

```
>> a+b % elementwise addition

ans =

    11    13    15    17

>> b-a % elementwise subtraction

ans =

     9     9     9     9
```

matrix operations

Plots

Plots

Q. Plot y as a function of x:

$y1 = 2\cos(x)$,

$y2 = \cos(x)$, and

$y3 = 0.5 * \cos(x)$,

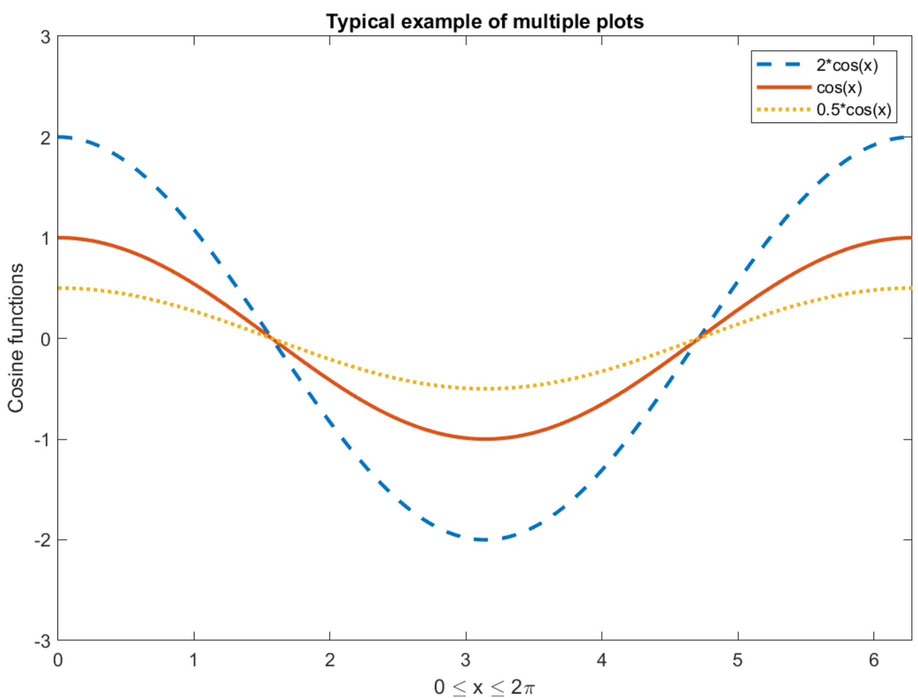
in the interval $0 \le x \le 2\pi$.

Command Window

```
>> x = 0:pi/100:2*pi; % or x=linspace(0,2*pi,201)
>> y1 = 2*cos(x); %function 1
>> y2 = cos(x); %function 2
>> y3 = 0.5*cos(x); %function 2
>> figure; %create a new figure handle
>> plot(x,y1,'--',x,y2,'-',x,y3,':') %plot takes in agruments: x, y and line style
>> xlabel('0 \leq x \leq 2\pi') %x label
>> ylabel('Cosine functions') % y label
>> legend('2*cos(x)','cos(x)','0.5*cos(x)') %legends
>> title('Typical example of multiple plots') % title of the plot
>> axis([0 2*pi -3 3]) % setting the axis bounds for plots
```

Function details

Plot details



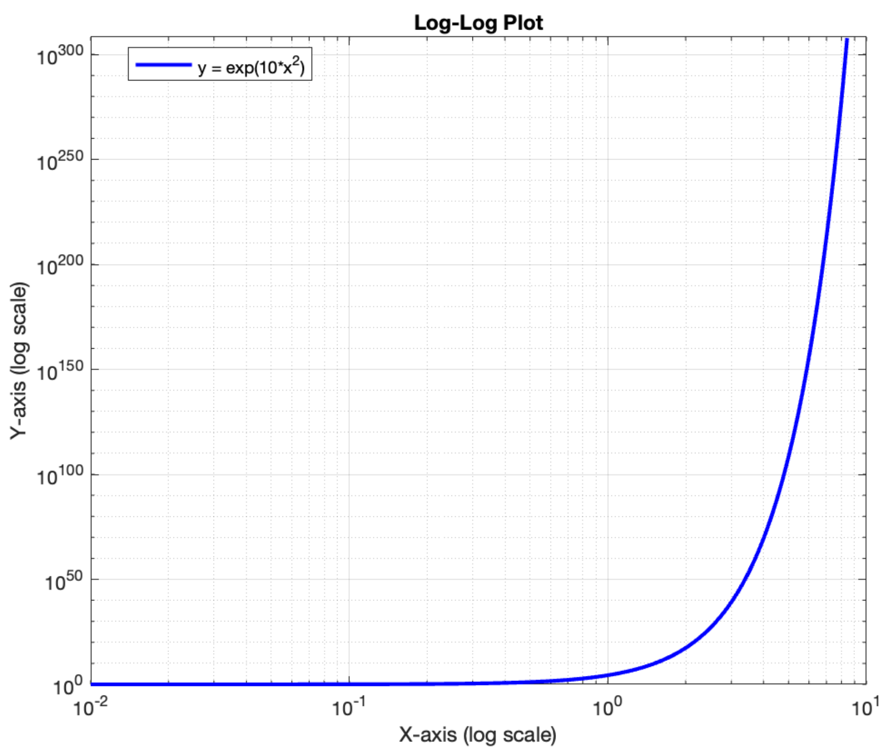
Note:

- Multiple (x, y) pairs arguments create multiple graphs with a single call to plot
- Plot function also has the attributes for line style, symbol color and line weight:

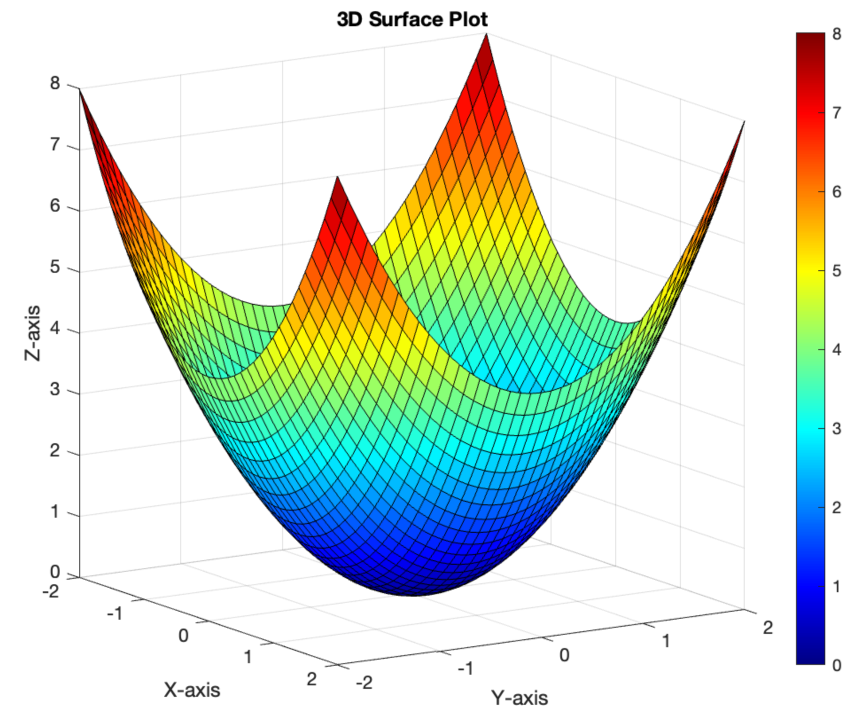
Table 2.3: Attributes for `plot`

Symbol	Color	Symbol	Line Style	Symbol	Marker
k	Black	–	Solid	+	Plus sign
r	Red	––	Dashed	o	Circle
b	Blue	:	Dotted	∗	Asterisk
g	Green	–.	Dash-dot	.	Point
c	Cyan	none	No line	×	Cross
m	Magenta			*s*	Square
y	Yellow			*d*	Diamond

Plots



Log-log plot



Surf plot

more plot types: scatter, bar, histogram, contour, heatmap etc.

Discretization

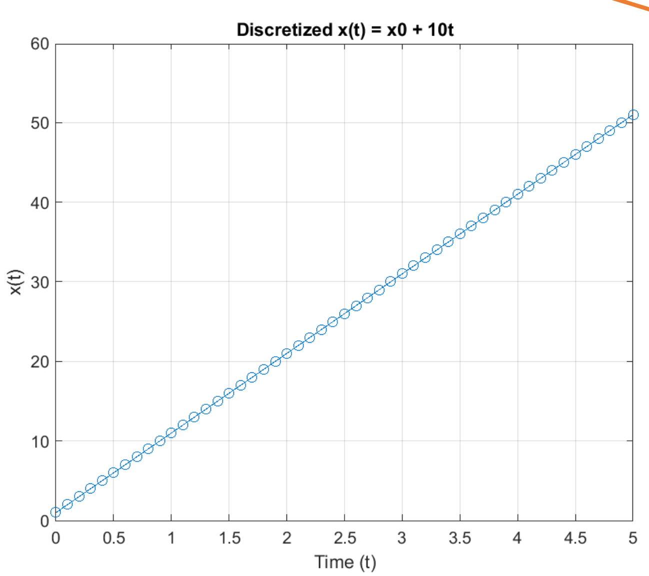
Discretization (example 1)

- Discretization is the process to convert a continuous equation into a form that can be used to calculate numerical solutions

Q. let' s say an object is traveling along the + x direction with a speed of 10 m/s, we would write its position vector as : **x (t) = x0 + 10t**. compute the position for time interval 0s to 5s.

We know that the particle will travel a distance of 10*Δt for a time interval of Δt, so that we can write:

$x(i) = x(i-1) + 10*\Delta t$



Editor – /Users/rahul/Library/CloudStorage/OneDrive-CUNY/CUNY/5. Sem 5 (F23)/MATLAB_intro/discretization_1.m *

discretization_1.m *

```
%% code for  discretization of eq: x(t) = x0 + 10t

clc;    % clearing all output in command window
clear;  % clearing all variables in workspace
clf;    % clearing only the current figure handle

% Define the initial position, time and time step
x0=1;                  %initial position
t0 = 0;                % Initial time
t_step = 0.1;          % Time step
tf = 5;                % Final time

%listing all the t values
t_values = t0:t_step:tf;

% Initialize array to store x values
x_values = zeros(1, length(t_values));

% Initial condition
x_values(1)=x0;

% Loop to discretize and calculate x(t)
for i = 2:length(t_values)
    x_values(i) = x_values(i-1) + 10 * t_step;
end

% Plot the results
plot(t_values, x_values, '-o');
xlabel('Time (t)');
ylabel('x(t)');
title('Discretized x(t) = x0 + 10t');
grid on;
```

Discretization (example 2)

$$\frac{dy}{dx} = -2y + x^2$$

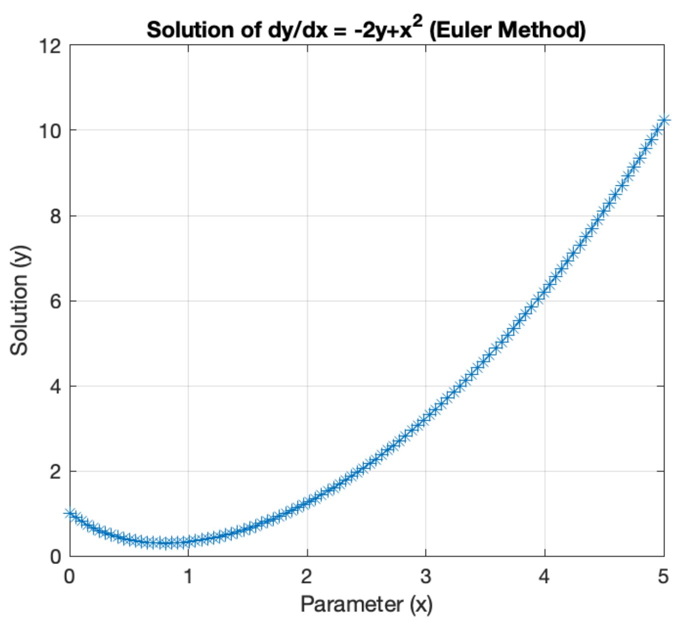
Q. Numerically solve the above-mentioned ordinary differential equation (ode) by discretization for the x interval [0,5]

Discretizing using the Euler's method:

$$y_{i+1} = y_i + hf(x_i, y_i)$$

where,

- y_{i+1} is the next estimated solution value;
- y_i is the current value;
- h is the interval between steps;
- $f(x_i, y_i)$ is the value of the derivative at the current (x_i, y_i) point.



We can write it as:

$$y_i = y_{i-1} + hf(x_{i-1}, y_{i-1})$$

Editor - C:\Users\panda\OneDrive - CUNY\CUNY\5. Sem 5 (F23)\MATLAB_intro\myode.m

discretization_2.m | myode.m

```
%% This function is an ode
% this function is used in the script discretization_2.m to evalute the
% derivation at a certain point.

function dydx = myode(x, y)
% Define the ODE dy/dx = -2y +x^2
dydx = -2*y+x^2;
end
```

discretization_2.m

```
%% code for discretization of ode: y'= -2y+x^2

clc;    % clearing all output in command window
clear;  % clearing all variables in workspace
clf;    % clearing only the current figure handle

%Define the ODE function
%we can define the ode as shown below or write it into a function and use
%it in this script. But here we use the function myode.m
%myode = @(x, y) -2 * y+x^2;

% Define the time span for the solution
xspan = [0, 5];  % Start at t=0 and end at t=5

% Define the initial condition
y0 = 1;

% Discretize the time span into discrete time points
x_values = linspace(xspan(1), xspan(2), 100); % 100 time points
h = x_values(2) - x_values(1); %delta x (spacing)

% Initialize array to position values (y)
y_values = zeros(1, length(x_values));

% Set the initial values
y_values(1) = y0;

% Apply the Euler method for numerical integration
for i = 2:length(x_values)
    y_values(i) = y_values(i-1) + h * myode(x_values(i-1), y_values(i-1));

end

% Plot the solution
plot(x_values, y_values, '-*', 'LineWidth', .5);
xlabel('Parameter (x)');
ylabel('Solution (y)');
title('Solution of dy/dx = -2y+x^2 (Euler Method)');
grid on;
```

ODE function

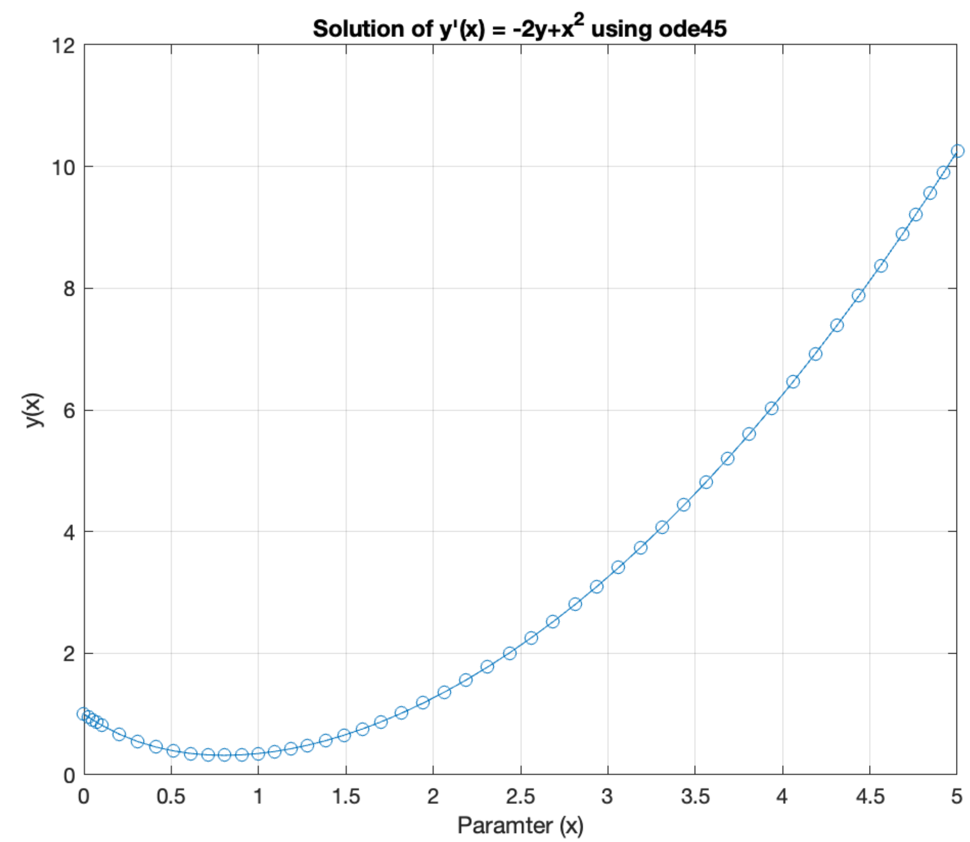
$$\frac{dy}{dx} = -2y + x^2$$

MATLAB has an inbuilt function called **ODE** to numerically solve ordinary differential equations. Here we use ode45 which uses Runge-Kutta approximation to solve the ode.

Editor – /Users/rahul/Library/CloudStorage/OneDrive-CUNY/CUNY/5. Sem 5 (F23)/MATLAB_intro/ode45_.m

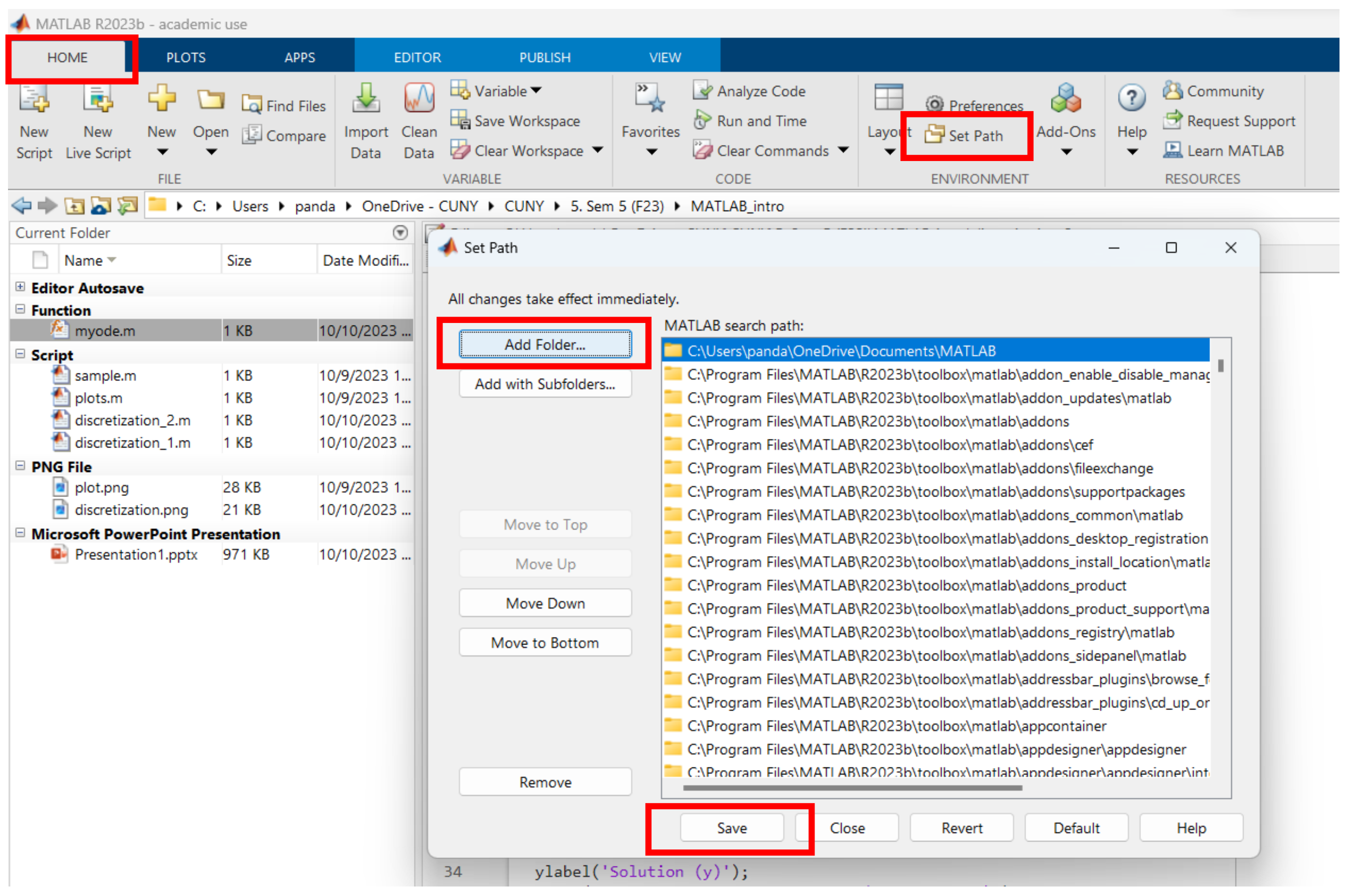
ode45_.m | myode.m

```
%% code for discretization of ode: y'= -2y+x^2

clc;    % clearing all output in command window
clear;  % clearing all variables in workspace
clf;    % clearing only the current figure handle

% Define the time span for the solution
xspan = [0, 5];  % Start at t=0 and end at t=5

% Define the initial condition
y0 = 1;

% Use the ode45 solver to solve the ODE using the custom ODE function
[x, y] = ode45(@myode, xspan, y0);

% Plot the results
plot(x, y, '-o');
xlabel('Paramter (x)');
ylabel('y(x)');
title('Solution of y''(x) = -2y+x^2 using ode45');
grid on;
```



Points to remember

1. When working with a script which requires a function (e.g., the ODE function) make sure the script and the function are in the same directory or else add the directory which contains the function file to the **PATH**



Adding a directory to the path:

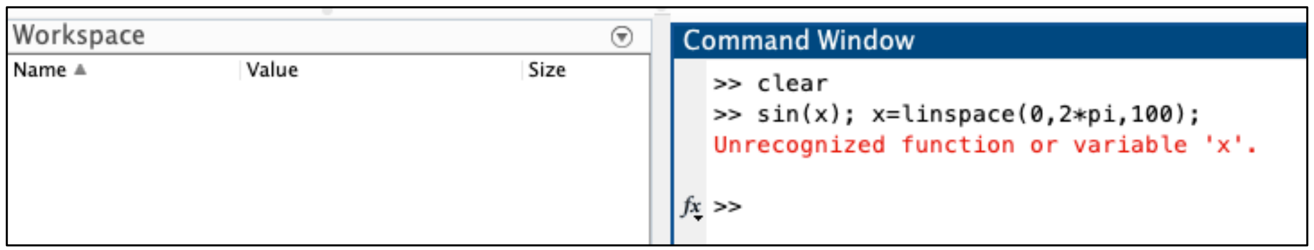
1. in the HOME tab
2. click on set path
3. add the folder you wish to work with
4. save

Points to remember

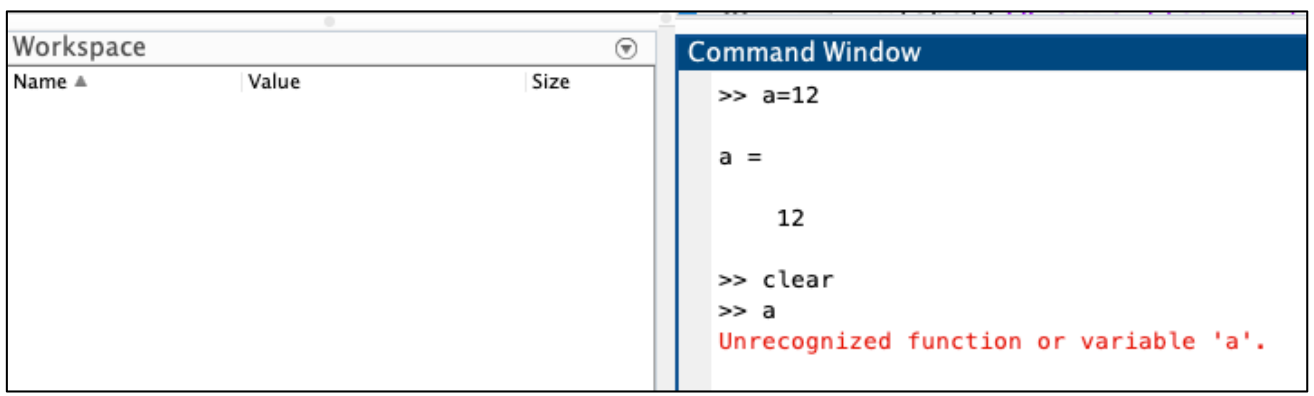
2. Make sure your workspace is clear before you execute your code so that the script does not use variables already in the memory.

```
clc;    % clearing all output in command window
clear;  % clearing all variables in workspace
clf;    % clearing only the current figure handle
```

3. You can only use variables after they are stored in the workspace



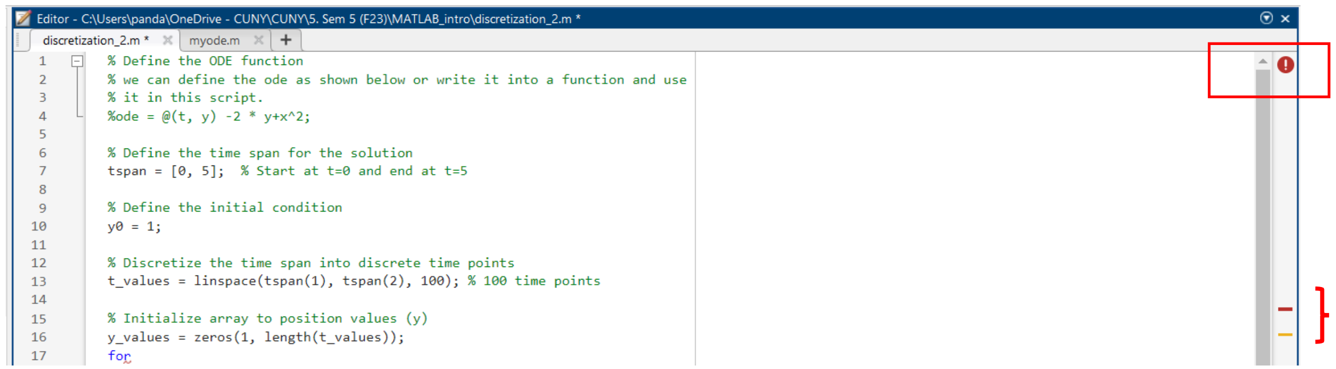
here we called in the variable x without defining it first. Also, the workspace has no variable stored



cannot call in a variable when it is not stored in workspace even though you see it in the command window.

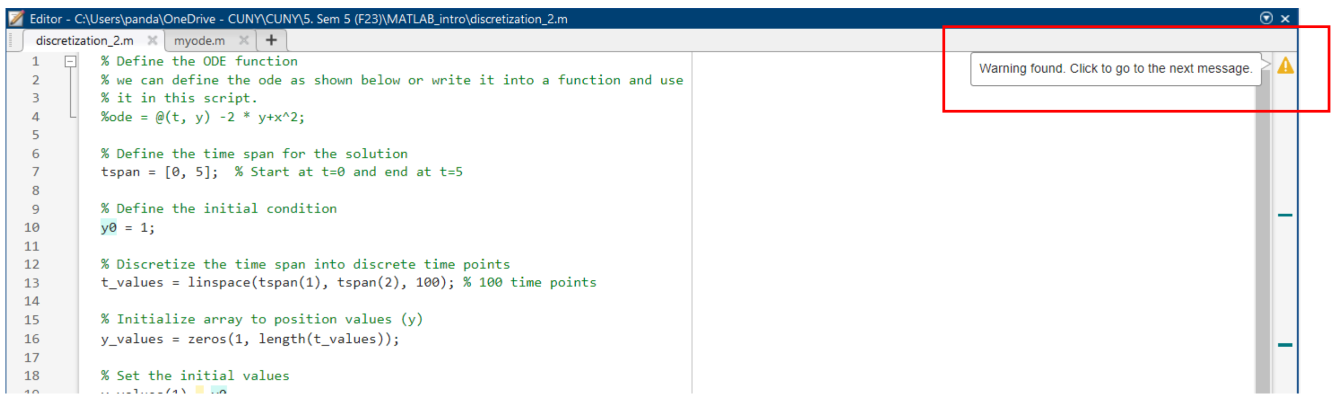
Points to remember

4. Good practice to look for warnings before executing the script

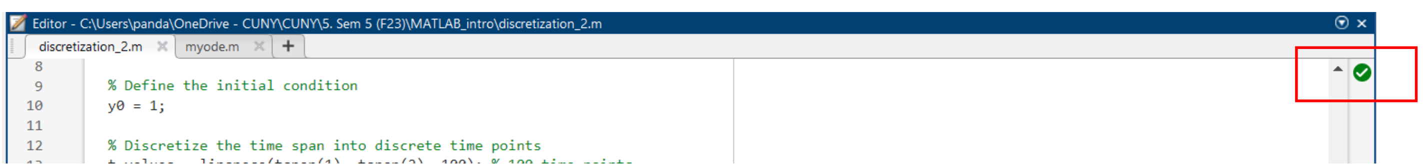


red exclamation – script won't run

click on these line markers to view more info on the error



yellow exclamation – script might execute but with some issue

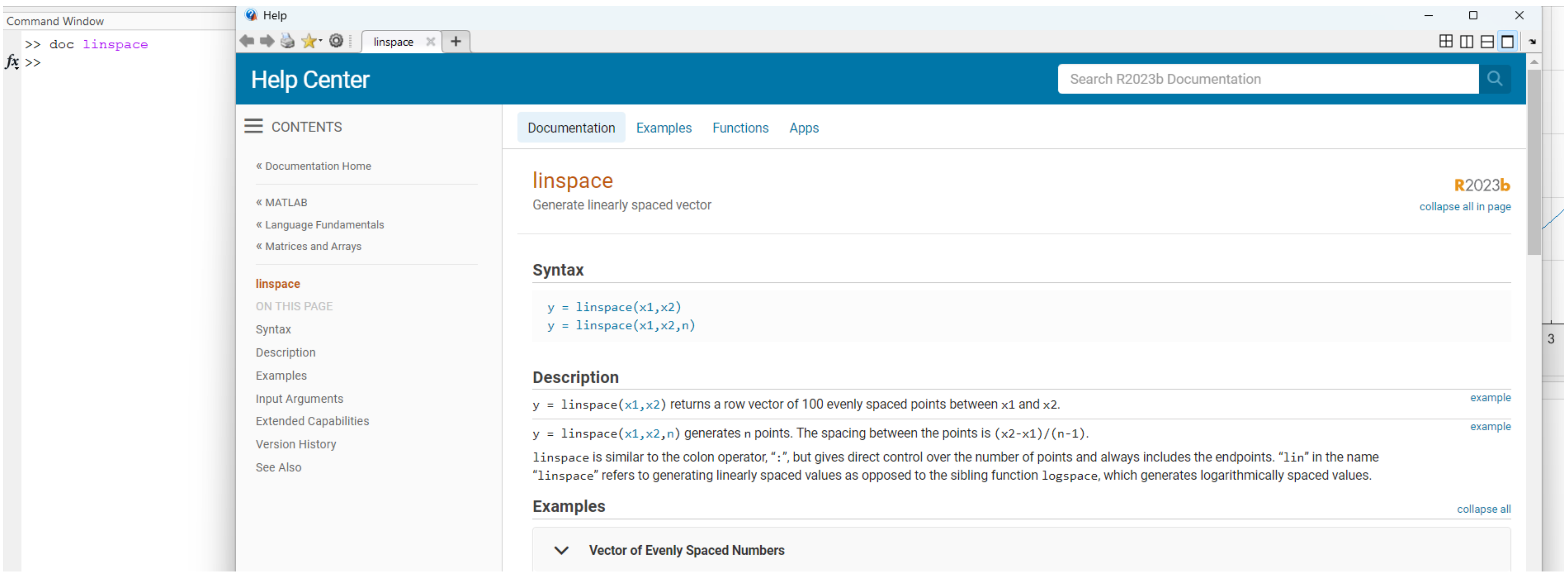


green tick – script is error free

Points to remember

5. To look up documentation on any MATLAB function/ feature just type :
doc <function/feature> OR help <function/feature> in the command window.
A new help window will pop up.

Eg.: Looking up info on **linspace**



Thank you!

Scripts used here and additional study material: https://rahul-pandare.github.io/teaching/matlab-intro

---

Project due – Dec 8

Office hours: Dec 2 & 3 (Tentative)

For question: rpandar000@citymail.cuny.edu ; Office: ST-305

Download MATLAB: https://www.mathworks.com/academia/tah-portal/city-university-of-new-york-1111017.html
(sign in with your cuny.edu email ID)