

ENGR I1100
Engineering Analysis

INTRO TO MATLAB

Sept 11, 2024

Rahul Pandare

PhD Candidate

Dept. of Chemical Engineering

rpandar000@citymail.cuny.edu

MATLAB vs PYTHON

MATLAB

- Proprietary, high-level programming language primarily designed for numerical and matrix computations. Written in C/C++.
- MATLAB offers toolboxes for various domains: Image Processing Toolbox, Aerospace Toolbox, Deep Learning Toolbox, SimBiology Toolbox, etc.
- MATLAB provides its own integrated development environment (IDE)
- Common in academia, engineering, and simulation

Python

- Open-source, general-purpose programming language known for its versatility. Written in C.
- Rich ecosystem of libraries and packages: NumPy, Matplotlib, openCV, etc.
- Python offers multiple IDE options: Jupyter Notebook, PyCharm, Visual Studio Code, etc.
- Widely used in diverse fields – Data analytics, Web development, FinTech, AI

MATLAB screen overview

The image shows the MATLAB R2023a - academic use interface. The top menu bar includes HOME, PLOTS, APPS, EDITOR, PUBLISH, and VIEW. The top toolbar contains icons for New, Open, Save, Print, Go To, Find, Refactor, Profiler, Run, Step, and Stop. The top status bar shows the current directory path: /Users/rahu/Library/CloudStorage/OneDrive-CUNY/CUNY/5. Sem 5 (F23)/MATLAB_intro. The left sidebar shows the Current Folder (sample.m) and the Workspace (x, y). The main Editor window displays the script sample.m. The right sidebar shows the Figures window (Figure 2) and the Command History window.

Path to working directory

File in current directory

Editor

Workspace
- displays the info on currently loaded variables

Command Window
- to interact with MATLAB
- debugging
- documentation and help

Figures

Command History

```
imagesc(raw(:,:,110))
parc hek=(raw(clip(k*(p xx(5)-os)+(-k*15:k*15),nx,1),clip(k*(pyy(5)-os)+(-k*15:...
figured;isosurface(parc hek)
k=1;parc hek=(ri(clip(k*(p xx(5)-os)+(-k*15:k*15),nx,1),clip(k*(pyy(5)-os)+(-k*...
figured;isosurface(parc hek)
[nx,ny,nz]=size(ri);
k=1;parc hek=(ri(clip(k*(p xx(5)-os)+(-k*15:k*15),nx,1),clip(k*(pyy(5)-os)+(-k*...
isosurface(parc hek)
clf;isosurface(parc hek)
axis image
[xxx,yyy,zzz]=ndgrid(1:nx,1:ny,1:nz);
rrr=sqrt((xxx-k*(p xx(5)-os)).^2+(yyy-k*(pyy(5)-os)).^2+((zzz-k*(p zz(5)-o...
isosurface(imm)
clf;isosurface(parc hek)
clf;isosurface(parc hek); axis image
k=4;rrr=sqrt((xxx-k*(p xx(5)-os)).^2+(yyy-k*(pyy(5)-os)).^2+((zzz-k*(p zz(5)-o...
isosurface(imm)
k=1;rrr=sqrt((xxx-k*(p xx(5)-os)).^2+(yyy-k*(pyy(5)-os)).^2+((zzz-k*(p zz(5)-o...
isosurface(imm)
showpart3(pxx,pyy,pzz,D*ones(1,length(pzz)));
showpart3(pxx,pyy,pzz,D*ones(1,length(pzz)));axis on
```

Zoom: 110% UTF-8 LF script Ln 19 Col 19

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)
fx >>
```

Using it 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 +∞
atan(x)	Arc tangent	floor(x)	Round towards -∞
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 original 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
```

Important:
in MATLAB
indexing starts
from 1

Matrix operations

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

Colon operator

```
ans =
```

```
51
```

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

linspace operator

```
ans =
```

```
100
```

```
>> 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
```

matrix operations

```
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
```

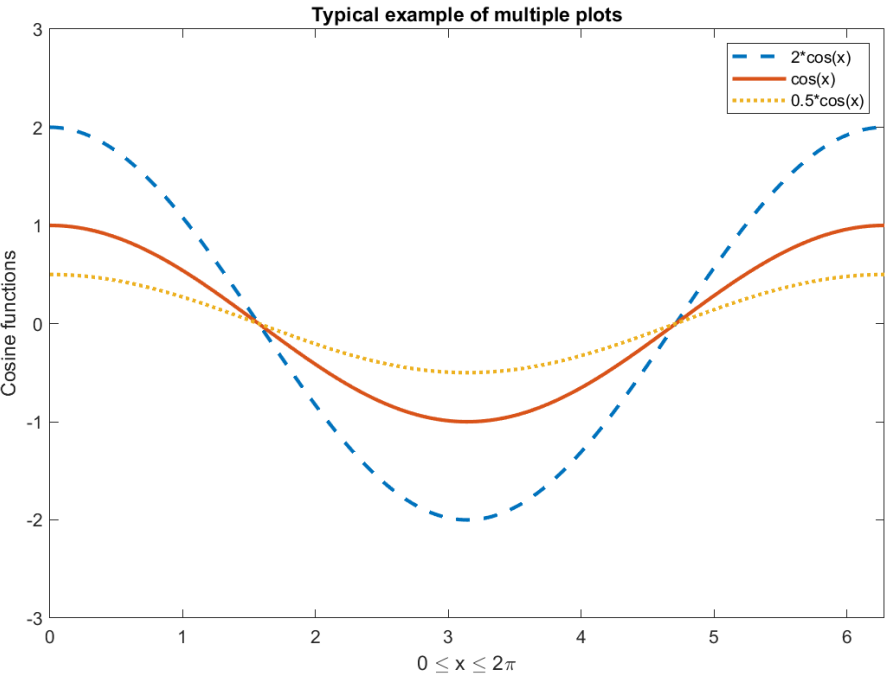
Plots

Plots

Q. Plot y as a function of x :
 $y_1 = 2 \cos(x)$,
 $y_2 = \cos(x)$, and
 $y_3 = 0.5 \cos(x)$,
in the interval $0 \leq x \leq 2\pi$.

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
```

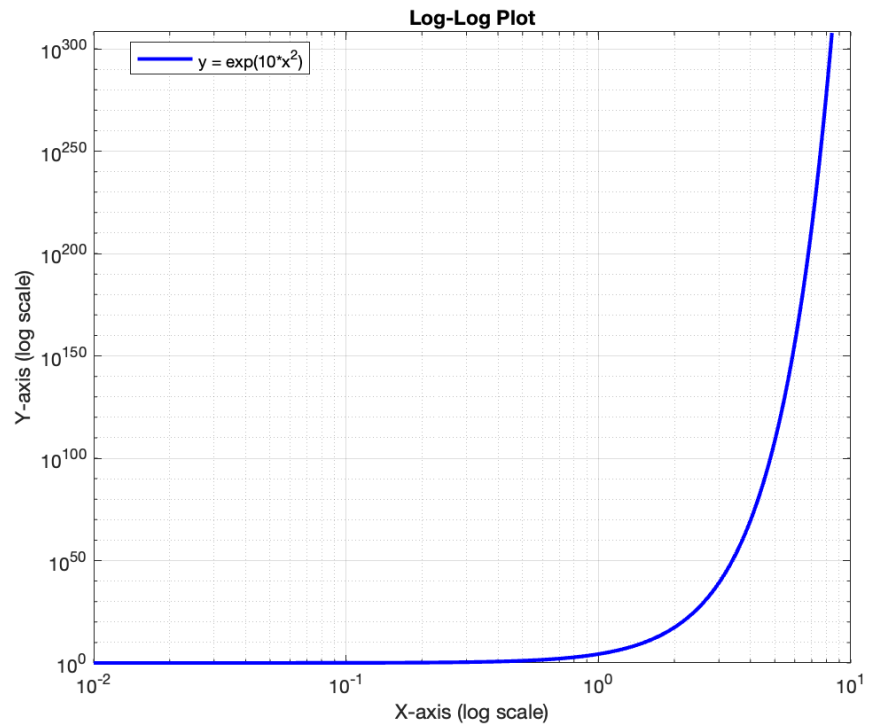


- Note:
- Multiple (x, y) pairs arguments create multiple graphs with a single call to plot
 - Plot function also has the attributes for line style, symbol color and line weight

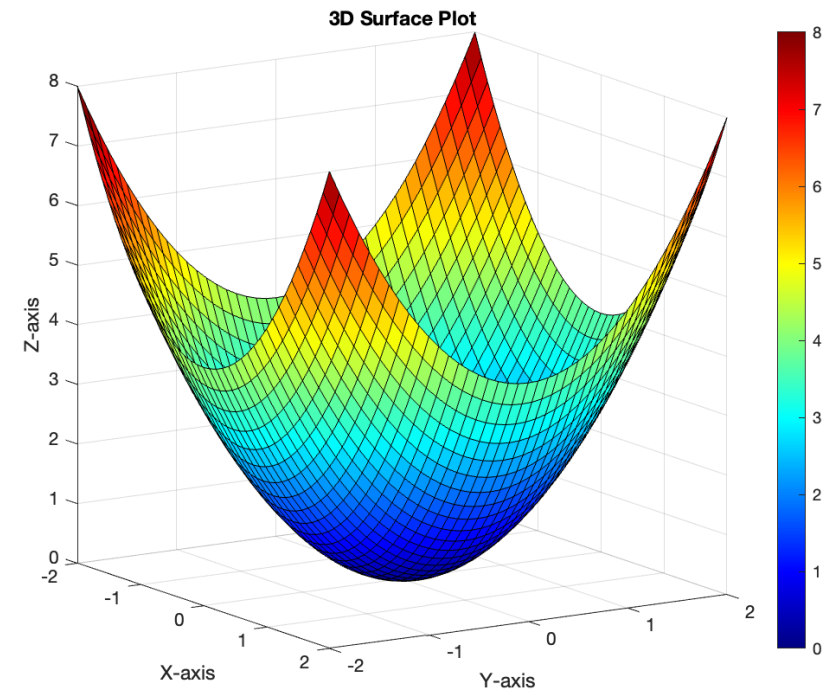
Table 2.3: Attributes for plot

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

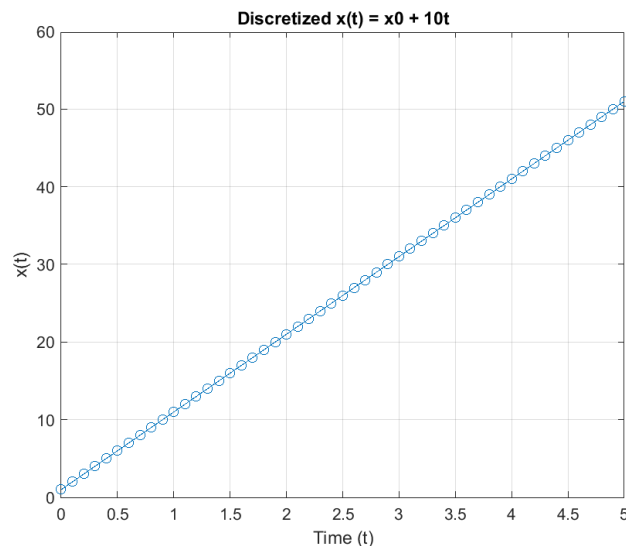
Discretization (example 1)

- 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 :
 $\mathbf{x}(t) = \mathbf{x}_0 + 10t$. compute the position for time interval 0s to 5s.

We know that the particle will travel a distance of $10 \cdot \Delta t$ for a time interval of Δt , so that we can write:

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



```
Editor - /Users/rahul/Library/CloudStorage/OneDrive-CUNY/CUNY/5. Sem 5 (F23)/MATLAB_intro/discretization_1.m *
discretization_1.m *
1  %% code for discretization of eq: x(t) = x0 + 10t
2
3  clc;    % clearing all output in command window
4  clear;  % clearing all variables in workspace
5  clf;    % clearing only the current figure handle
6
7  % Define the initial position, time and time step
8  x0=1;    % initial position
9  t0 = 0;   % Initial time
10 t_step = 0.1; % Time step
11 tf = 5;   % Final time
12
13 %listing all the t values
14 t_values = t0:t_step:tf;
15
16 % Initialize array to store x values
17 x_values = zeros(1, length(t_values));
18
19 % Initial condition
20 x_values(1)=x0;
21
22 % Loop to discretize and calculate x(t)
23 for i = 2:length(t_values)
24     x_values(i) = x_values(i-1) + 10 * t_step;
25 end
26
27 % Plot the results
28 plot(t_values, x_values, '-o');
29 xlabel('Time (t)');
30 ylabel('x(t)');
31 title('Discretized x(t) = x0 + 10t');
32 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:

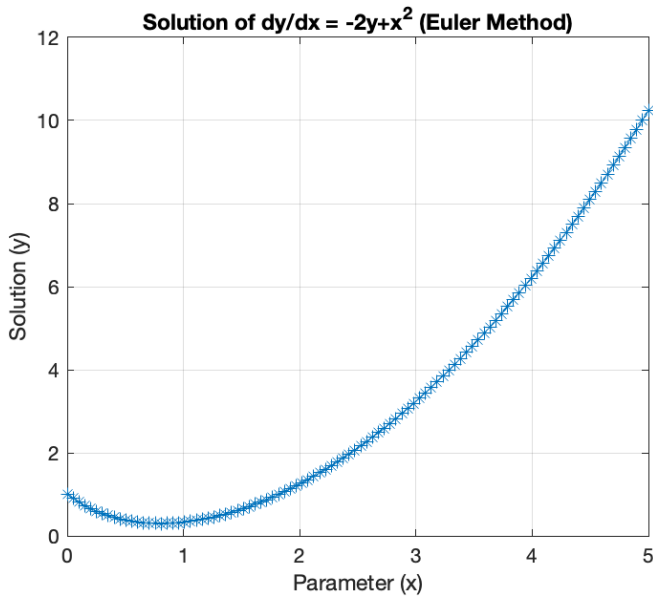
$$y_{i+1} = y_i + hf(x_i, y_i)$$

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:

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



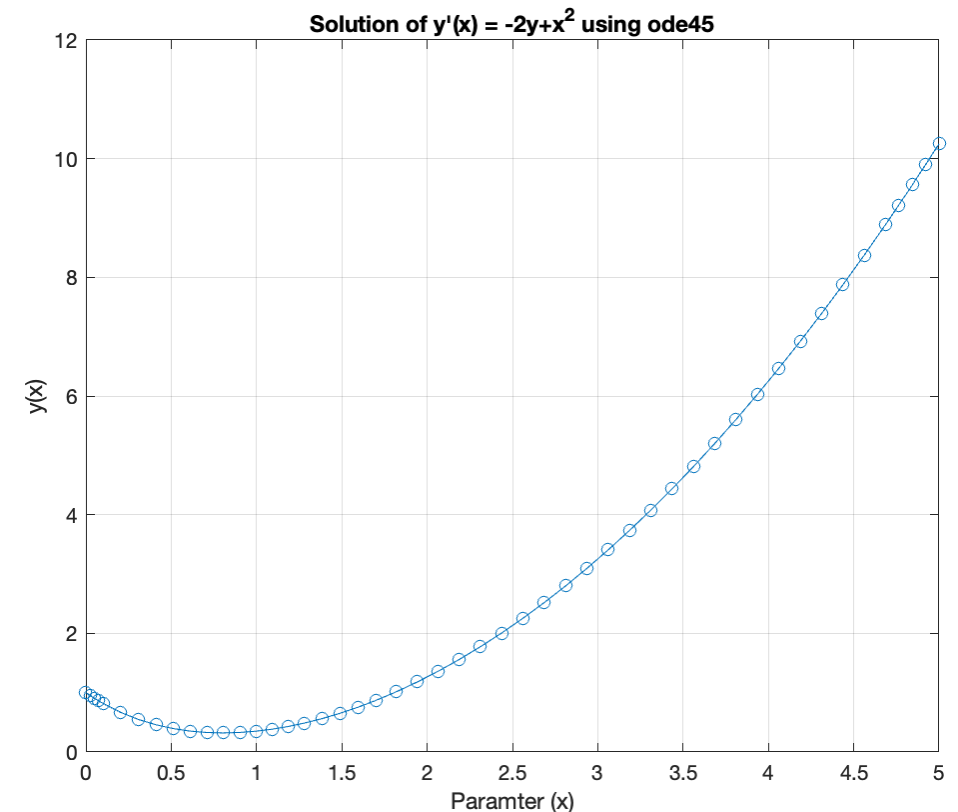
```
Editor - C:\Users\panda\OneDrive - CUNY\CUNY\5. Sem 5 (F23)\MATLAB_intro\myode.m
discretization_2.m  myode.m  +
1  %% This function is an ode
2  % this function is used in the script discretization_2.m to evaluate the
3  % derivation at a certain point.
4
5  function dydx = myode(x, y)
6  % Define the ODE dy/dx = -2y + x^2
7  dydx = -2*y+x^2;
8  end
discretization_2.m  +
1  %% code for discretization of ode: y' = -2y+x^2
2
3  clc; % clearing all output in command window
4  clear; % clearing all variables in workspace
5  clf; % clearing only the current figure handle
6
7  %Define the ODE function
8  %we can define the ode as shown below or write it into a function and use
9  %it in this script. But here we use the function myode.m
10 %myode = @(x, y) -2 * y+x^2;
11
12 % Define the time span for the solution
13 xspan = [0, 5]; % Start at t=0 and end at t=5
14
15 % Define the initial condition
16 y0 = 1;
17
18 % Discretize the time span into discrete time points
19 x_values = linspace(xspan(1), xspan(2), 100); % 100 time points
20 h = x_values(2) - x_values(1); %delta x (spacing)
21
22 % Initialize array to position values (y)
23 y_values = zeros(1, length(x_values));
24
25 % Set the initial values
26 y_values(1) = y0;
27
28 % Apply the Euler method for numerical integration
29 for i = 2:length(x_values)
30     y_values(i) = y_values(i-1) + h * myode(x_values(i-1), y_values(i-1));
31 end
32
33
34 % Plot the solution
35 plot(x_values, y_values, '-*', 'LineWidth', .5);
36 xlabel('Parameter (x)');
37 ylabel('Solution (y)');
38 title('Solution of dy/dx = -2y+x^2 (Euler Method)');
39 grid on;
```

ODE function

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

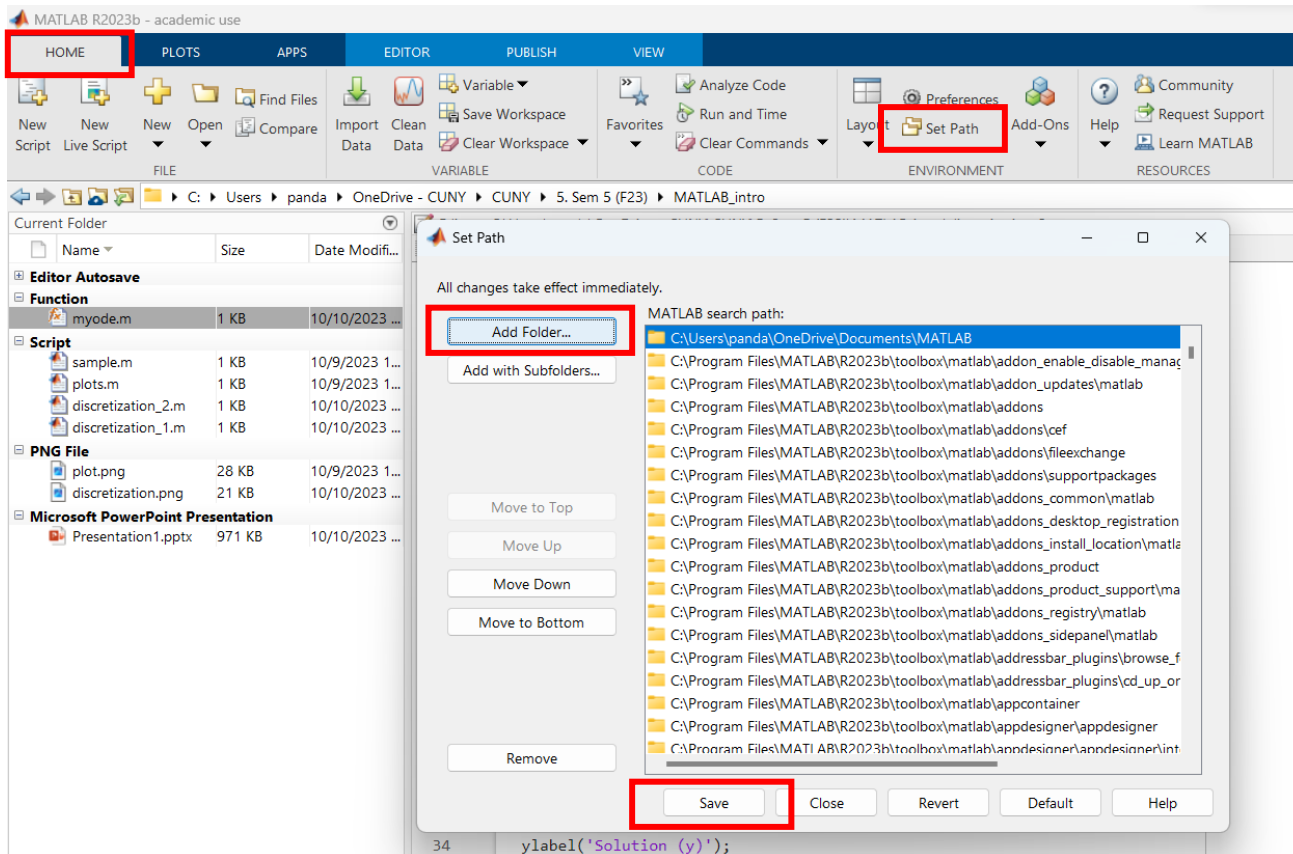
MATLAB has an inbuilt function called **ODE** to numerically solve ordinary differential equations. Here we use ode45 which uses Runge-Kutta approximation to solve the ode.

```
Editor - /Users/rahul/Library/CloudStorage/OneDrive-CUNY/CUNY/5. Sem 5 (F23)/MATLAB_intro/ode45_m
ode45_m x myode.m x +
1 %% code for discretization of ode: y' = -2y + x^2
2
3 clc; % clearing all output in command window
4 clear; % clearing all variables in workspace
5 clf; % clearing only the current figure handle
6
7 % Define the time span for the solution
8 xspan = [0, 5]; % Start at t=0 and end at t=5
9
10 % Define the initial condition
11 y0 = 1;
12
13 % Use the ode45 solver to solve the ODE using the custom ODE function
14 [x, y] = ode45(@myode, xspan, y0);
15
16 % Plot the results
17 plot(x, y, '-o');
18 xlabel('Paramter (x)');
19 ylabel('y(x)');
20 title('Solution of y''(x) = -2y+x^2 using ode45');
21 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:

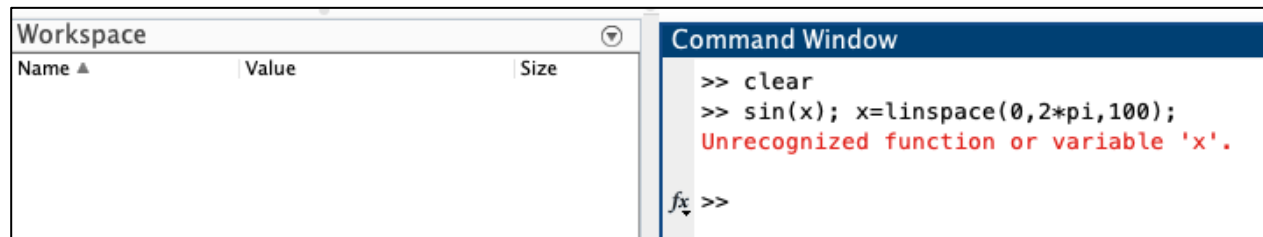
1. in the HOME tab
2. click on set path
3. add the folder you wish to work with
4. save

Points to remember

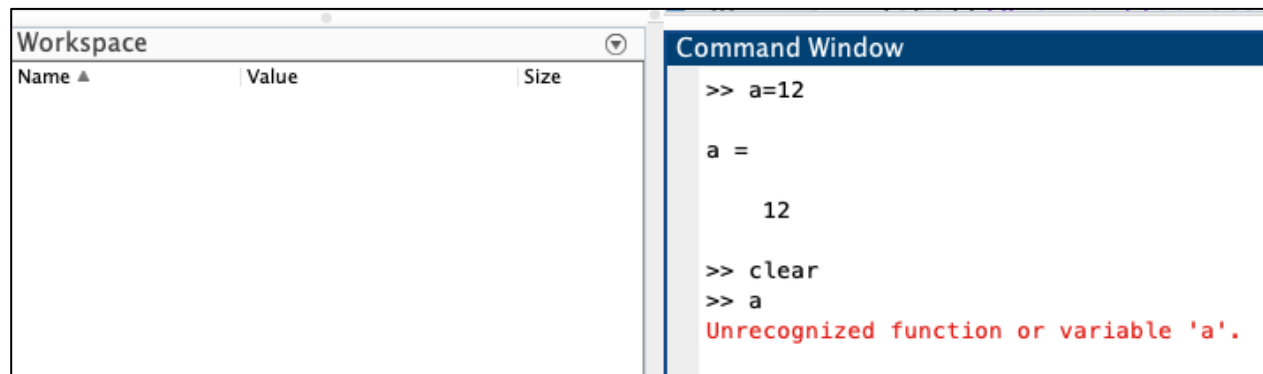
2. Make sure your workspace is clear before you execute your code so that the script does not use variables already in the memory.

```
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



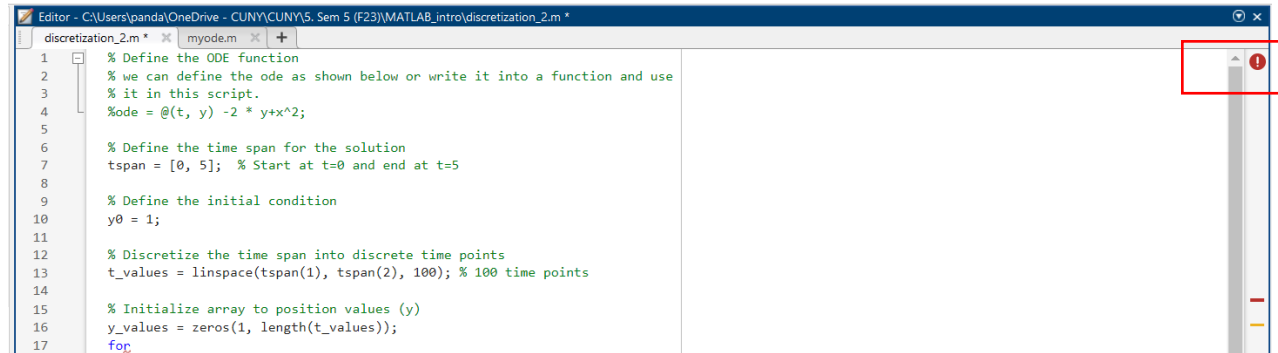
here we called in the variable x without defining it first. Also, the workspace has no variable stored



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



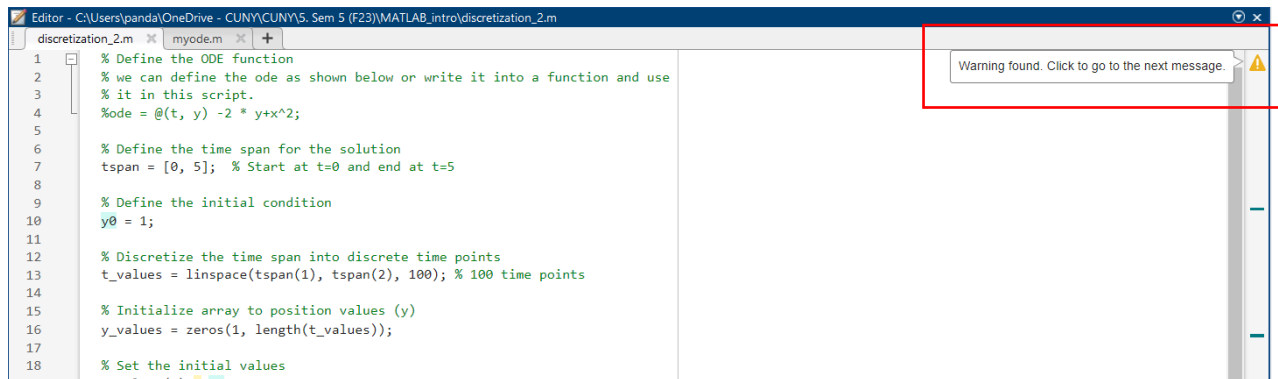
The image shows a MATLAB Editor window with a script named 'discretization_2.m'. The script contains the following code:

```
1 % Define the ODE function
2 % we can define the ode as shown below or write it into a function and use
3 % it in this script.
4 %ode = @(t, y) -2 * y*x^2;
5
6 % Define the time span for the solution
7 tspan = [0, 5]; % Start at t=0 and end at t=5
8
9 % Define the initial condition
10 y0 = 1;
11
12 % Discretize the time span into discrete time points
13 t_values = linspace(tspan(1), tspan(2), 100); % 100 time points
14
15 % Initialize array to position values (y)
16 y_values = zeros(1, length(t_values));
17 for
```

A red exclamation mark icon is visible in the top right corner of the editor window, indicating an error.

red exclamation – script won't run

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



The image shows the same MATLAB Editor window as above, but now with a yellow exclamation mark icon in the top right corner, indicating a warning. A tooltip is visible over the icon, stating: "Warning found. Click to go to the next message."

yellow exclamation – script might execute but with some issue



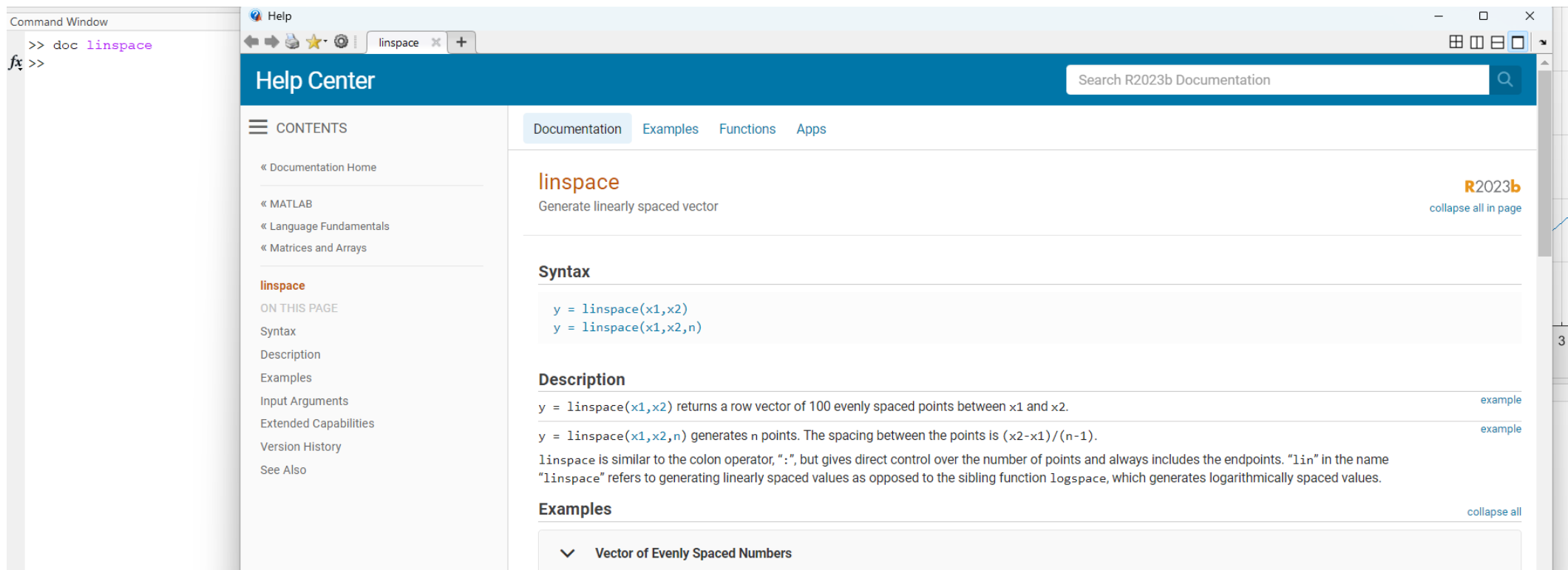
The image shows the same MATLAB Editor window as above, but now with a green tick icon in the top right corner, indicating that the script is error-free.

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.

Looking up info on **linspace**



The screenshot shows the MATLAB Help Center interface. On the left, the Command Window displays the command `>> doc linspace`. The Help Center window is open, showing the documentation for the **linspace** function. The page title is "linspace" with the subtitle "Generate linearly spaced vector". The version is R2023b. The page is organized into sections: Syntax, Description, and Examples. The Syntax section shows two ways to use the function: `y = linspace(x1,x2)` and `y = linspace(x1,x2,n)`. The Description section explains that `y = linspace(x1,x2)` returns a row vector of 100 evenly spaced points between `x1` and `x2`, and `y = linspace(x1,x2,n)` generates `n` points. The Examples section is partially visible, showing a section titled "Vector of Evenly Spaced Numbers".

Command Window

```
>> doc linspace  
fx >>
```

Help Center

Search R2023b Documentation

CONTENTS

- « Documentation Home
- « MATLAB
- « Language Fundamentals
- « Matrices and Arrays

linspace

ON THIS PAGE

- Syntax
- Description
- Examples
- Input Arguments
- Extended Capabilities
- Version History
- See Also

linspace

Generate linearly spaced vector

R2023b
[collapse all in page](#)

Syntax

```
y = linspace(x1,x2)  
y = linspace(x1,x2,n)
```

Description

`y = linspace(x1,x2)` returns a row vector of 100 evenly spaced points between `x1` and `x2`. [example](#)

`y = linspace(x1,x2,n)` generates `n` points. The spacing between the points is $(x2-x1)/(n-1)$. [example](#)

`linspace` is similar to the colon operator, `:`, but gives direct control over the number of points and always includes the endpoints. "lin" in the name "linspace" refers to generating linearly spaced values as opposed to the sibling function `logspace`, which generates logarithmically spaced values.

Examples [collapse all](#)

Vector of Evenly Spaced Numbers

Thank you!

Project due – Second week in November

Office hours: Tentatively one week before submission

For question: rpandar000@citymail.cuny.edu ; 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)

To access scripts used
here and additional study
material click [here](#)