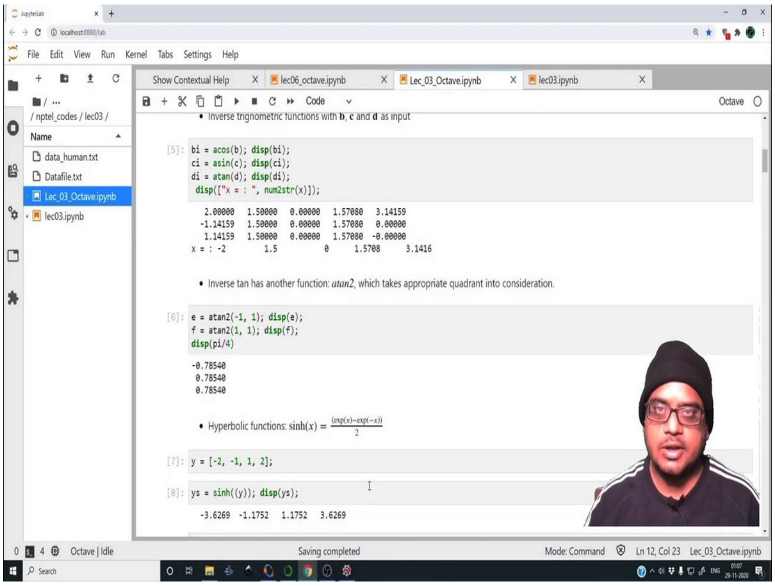
Tools in Scientific Computing

Prof. Aditya Bandopadhyay
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture – 07
Overview of Jupyter Lab, Octave GUI, Spyder GUI

(Refer Slide Time: 00:27)



- Inverse trignometric functions with **b**, **c** and **d** as input

```
[5]: bi = acos(b); disp(bi);
     ci = asin(c); disp(ci);
     di = atan(d); disp(di);
      disp(["x = : ", num2str(x)]);

       2.00000   1.50000   0.00000   1.57080   3.14159
      -1.14159   1.50000   0.00000   1.57080   0.00000
       1.14159   1.50000   0.00000   1.57080  -0.00000
     x = : -2         1.5           0       1.5708      3.1416
```

- Inverse tan has another function: *atan2*, which takes appropriate quadrant into consideration.

```
[6]: e = atan2(-1, 1); disp(e);
     f = atan2(1, 1); disp(f);
     disp(pi/4)

     -0.78540
      0.78540
      0.78540
```

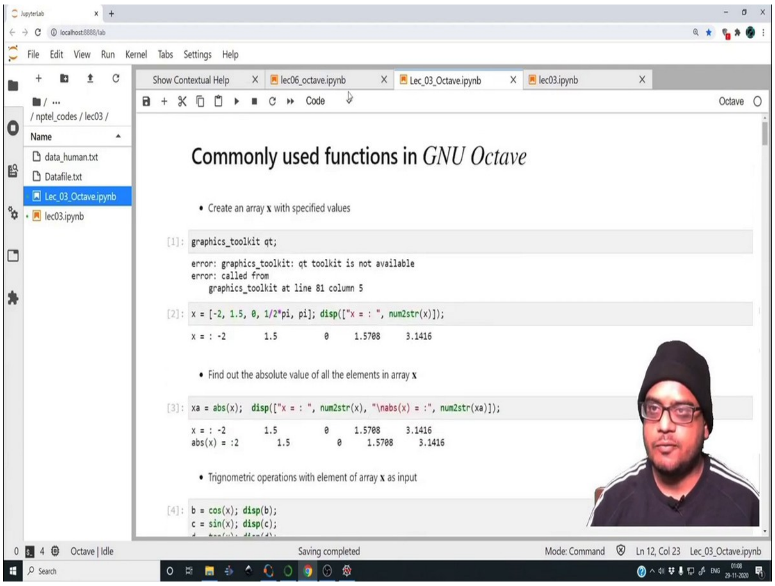
- Hyperbolic functions: $\sinh(x) = \frac{(\exp(x)-\exp(-x))}{2}$

```
[7]: y = [-2, -1, 1, 2];

[8]: ys = sinh((y)); disp(ys);

       -3.6269  -1.1752   1.1752   3.6269
```

Hello everyone to the 7th lecture. This lecture is not going to be about something new, but rather we are going to discuss some aspects of GNU Octave. Throughout this course so far we have used Python as the primary language of programming, but this course is not about just one particular tool in programming.

While, we have used Python extensively one can achieve the same things using GNU Octave. GNU Octave is sort of a clone of MATLAB; whatever you would write in MATLAB would run exactly the same or almost exactly the same in GNU Octave.

(Refer Slide Time: 01:18)



Commonly used functions in *GNU Octave*

- Create an array **x** with specified values

```
[1]: graphics_toolkit qt;

error: graphics_toolkit: qt toolkit is not available
error: called from
    graphics_toolkit at line 81 column 5

[2]: x = [-2, 1.5, 0, 1/2*pi, pi]; disp(["x = : ", num2str(x)]);

x = : -2        1.5          0      1.5708      3.1416
```

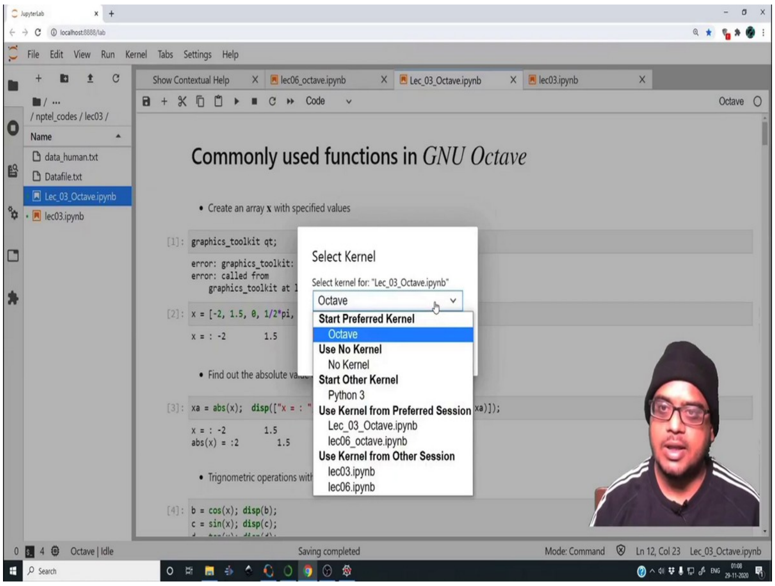
- Find out the absolute value of all the elements in array **x**

```
[3]: xa = abs(x);  disp(["x = : ", num2str(x), "\nabs(x) = :", num2str(xa)]);

x = : -2        1.5          0      1.5708      3.1416
abs(x) = :2        1.5          0      1.5708      3.1416
```

- Trignometric operations with element of array **x** as input

```
[4]: b = cos(x); disp(b);
     c = sin(x); disp(c);
```

(Refer Slide Time: 01:26)



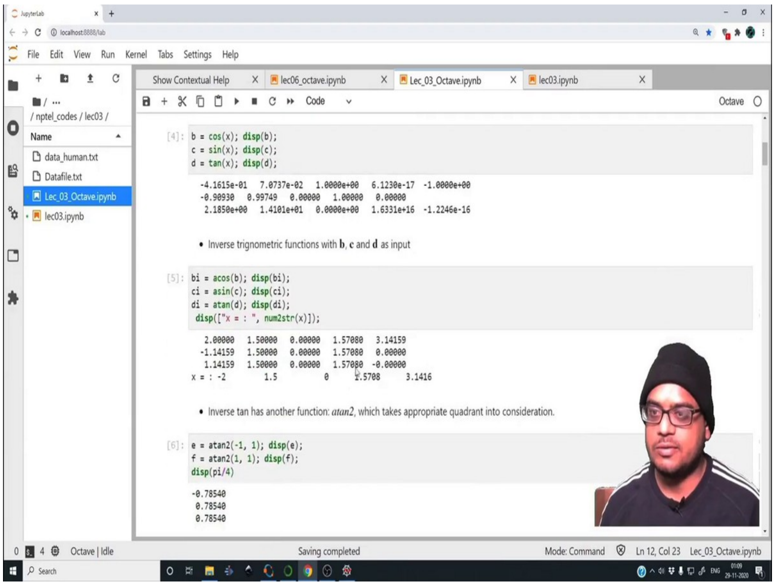
Select Kernel

Select kernel for: "Lec_03_Octave.ipynb"

Octave

In this particular case, we have used the environment of Jupyter Lab and used the Kernel Octave, Jupyter Lab allows you to use multiple Kernels. You can program in Python, you can program in Octave, you can program in Julia. You can in fact program in MATLAB also using Jupyter Lab. Jupyter Lab provides you a nice way of writing down documentation as you progress in the program; things like bullet points, things like equations ok.

(Refer Slide Time: 01:53)



```
[4]: b = cos(x); disp(b);
     c = sin(x); disp(c);
     d = tan(x); disp(d);

   -4.1615e-01   7.0737e-02   1.0000e+00   6.1230e-17  -1.0000e+00
   -0.90930   0.99749   0.00000   1.00000   0.00000
    2.1850e+00   1.4101e+01   0.0000e+00   1.6331e+16  -1.2246e-16
```

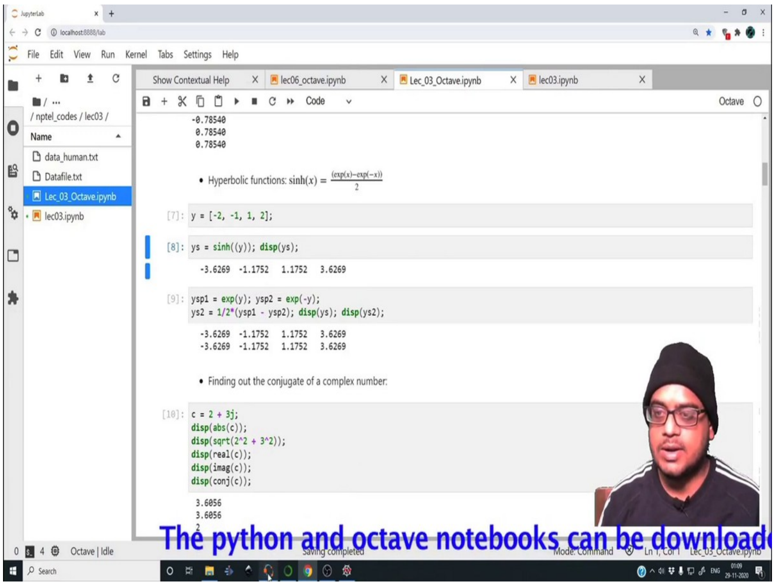
- Inverse trignometric functions with **b**, **c** and **d** as input

```
[5]: bi = acos(b); disp(bi);
     ci = asin(c); disp(ci);
     di = atan(d); disp(di);
      disp(["x = : ", num2str(x)]);

    2.00000   1.50000   0.00000   1.57080   3.14159
   -1.14159   1.50000   0.00000   1.57080   0.00000
    1.14159   1.50000   0.00000   1.57080  -0.00000
x = : -2         1.5           0      1.5708      3.1416
```

- Inverse tan has another function: *atan2*, which takes appropriate quadrant into consideration.

```
[6]: e = atan2(-1, 1); disp(e);
     f = atan2(1, 1); disp(f);
     disp(pi/4)

-0.78540
 0.78540
 0.78540
```

(Refer Slide Time: 01:54)



```
-0.78540
 0.78540
 0.78540
```

- Hyperbolic functions: $\sinh(x) = \frac{(\exp(x)-\exp(-x))}{2}$

```
[7]: y = [-2, -1, 1, 2];

[8]: ys = sinh((y)); disp(ys);

   -3.6269  -1.1752   1.1752   3.6269

[9]: ysp1 = exp(y); ysp2 = exp(-y);
     ys2 = 1/2*(ysp1 - ysp2); disp(ys); disp(ys2);

   -3.6269  -1.1752   1.1752   3.6269
   -3.6269  -1.1752   1.1752   3.6269
```

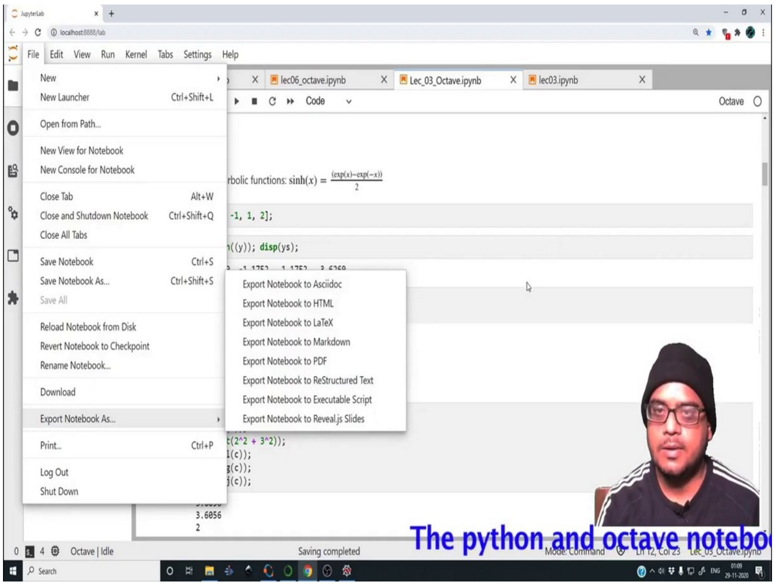
- Finding out the conjugate of a complex number:

```
[10]: c = 2 + 3j;
      disp(abs(c));
      disp(sqrt(2^2 + 3^2));
      disp(real(c));
      disp(imag(c));
      disp(conj(c));

 3.6056
 3.6056
 2
```

The python and octave notebooks can be download

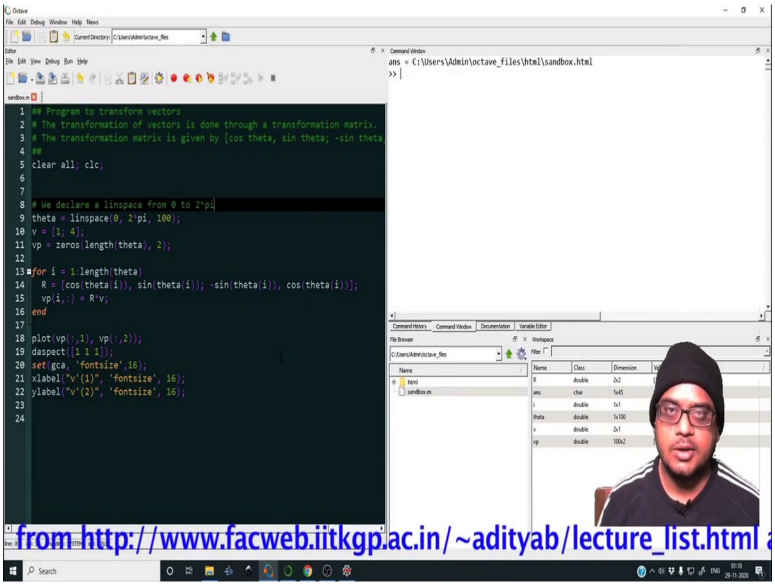
So, it has latex input for typing out equations, you can export this file to whatever you want ok.

(Refer Slide Time: 02:03)



You can export the notebook as PDF, Markdown, LaTeX, HTML, Ascii whatever you want. So, Jupyter Lab provides you this kind of an environment, where you can embed code, you can embed text.

(Refer Slide Time: 02:24)



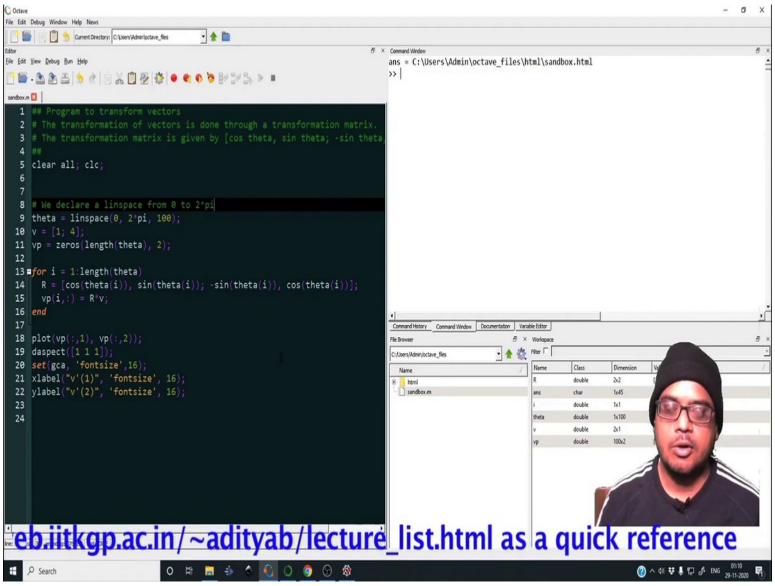
```
## Program to transform vectors
# The transformation of vectors is done through a transformation matrix.
# The transformation matrix is given by [cos theta, sin theta; -sin theta
##
clear all; clc;


# We declare a linspace from 0 to 2*pi
theta = linspace(0, 2*pi, 100);
v = [1; 4];
vp = zeros(length(theta), 2);

for i = 1:length(theta)
  R = [cos(theta(i)), sin(theta(i)); -sin(theta(i)), cos(theta(i))];
  vp(i,:) = R*v;
end

plot(vp(:,1), vp(:,2));
daspect([1 1 1]);
set(gca, 'fontsize',16);
xlabel("v'(1)", 'fontsize', 16);
ylabel("v'(2)", 'fontsize', 16);
```

```
ans = C:\Users\Admin\octave_files\html\sandbox.html
>>
```

Octave
File Edit Debug Window Help News
Current Directory: C:\Users\Admin\octave_files
Editor
sandbox.m
Command Window
File Browser
Workspace
html
sandbox.m
from http://www.facweb.iitkgp.ac.in/~adityab/lecture_list.html

Apart from this, we can program directly in the environment Octave as shown. So, I have written a small file and this is the GUI for Octave. Octave also has a command line utility a cli, but we will make use of this GUI. Mind you that whatever we have done so far in Python, I have also uploaded the same, the corresponding octave file as well on the same website. For anyone who is interested you can browse through the html which contains the same code when implemented through Octave.

(Refer Slide Time: 03:09)



```
## Program to transform vectors
# The transformation of vectors is done through a transformation matrix.
# The transformation matrix is given by [cos theta, sin theta; -sin theta
##
clear all; clc;


# We declare a linspace from 0 to 2*pi
theta = linspace(0, 2*pi, 100);
v = [1; 4];
vp = zeros(length(theta), 2);

for i = 1:length(theta)
  R = [cos(theta(i)), sin(theta(i)); -sin(theta(i)), cos(theta(i))];
  vp(i,:) = R*v;
end

plot(vp(:,1), vp(:,2));
daspect([1 1 1]);
set(gca, 'fontsize',16);
xlabel("v'(1)", 'fontsize', 16);
ylabel("v'(2)", 'fontsize', 16);
```

```
ans = C:\Users\Admin\octave_files\html\sandbox.html
>>
```

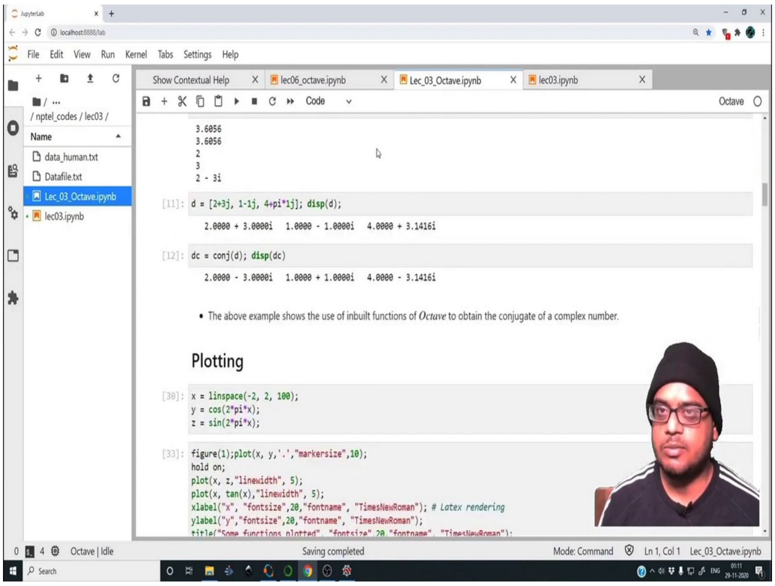
Name	Class	Dimension
R	double	2x2
ans	char	1x45
i	double	1x1
theta	double	1x100
v	double	2x1
vp	double	100x2

Octave
File Edit Debug Window Help News
Current Directory: C:\Users\Admin\octave_files
Editor
Command Window
sandbox.m
Command History
Command Window
Documentation
Variable Editor
File Browser
Workspace
html
sandbox.m
eb.iitkgp.ac.in/~adityab/lecture_list.html as a quick reference

All the lectures starting from arrays, matrices, transformation of vectors, transformation of matrix; then fixed point iterations, finding out the roots, Newton's basins. All these things what we have done in Python are also available through Octave.

So, those of you are interested in implementing things in Octave do have a look. This lecture is going to be a quick breezer for someone who is not aware of things in Octave. I will show you how to work in this particular GUI, but you could very well make use of Jupyter Lab to write your codes. This is one example of lecture 3.

(Refer Slide Time: 03:54)



```
3.6056
3.6056
2
3
2 - 3i
```

```
[11]: d = [2+3j, 1-1j, 4+pi*1j]; disp(d);
```

```
   2.0000 + 3.0000i   1.0000 - 1.0000i   4.0000 + 3.1416i
```

```
[12]: dc = conj(d); disp(dc)
```

```
   2.0000 - 3.0000i   1.0000 + 1.0000i   4.0000 - 3.1416i
```

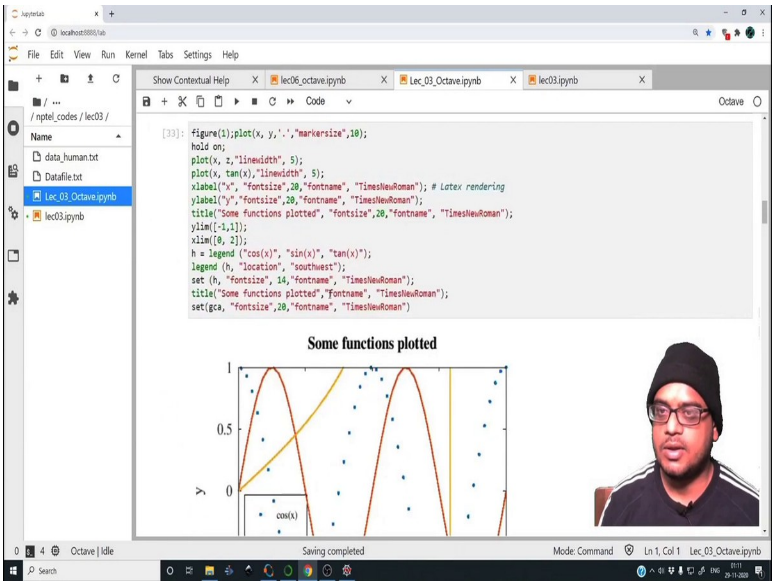
- The above example shows the use of inbuilt functions of *Octave* to obtain the conjugate of a complex number.

Plotting

```
[30]: x = linspace(-2, 2, 100);
      y = cos(2*pi*x);
      z = sin(2*pi*x);
```

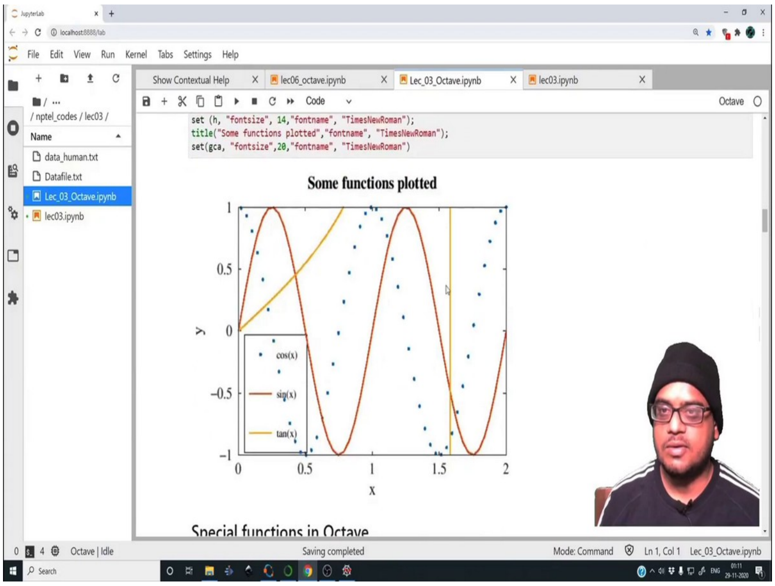
```
[33]: figure(1);plot(x, y,'.',"markersize",10);
      hold on;
      plot(x, z,"linewidth", 5);
      plot(x, tan(x),"linewidth", 5);
      xlabel("x", "fontsize",20,"fontname", "TimesNewRoman"); # Latex rendering
      ylabel("y","fontsize",20,"fontname", "TimesNewRoman");
      title("Some functions plotted", "fontsize",20,"fontname", "TimesNewRoman");
```

(Refer Slide Time: 03:35)



```
[33]: figure(1);plot(x, y,'.',"markersize",10);
      hold on;
      plot(x, z,"linewidth", 5);
      plot(x, tan(x),"linewidth", 5);
      xlabel("x", "fontsize",20,"fontname", "TimesNewRoman"); # Latex rendering
      ylabel("y","fontsize",20,"fontname", "TimesNewRoman");
      title("Some functions plotted", "fontsize",20,"fontname", "TimesNewRoman");
      ylim([-1,1]);
      xlim([0, 2]);
      h = legend ("cos(x)", "sin(x)", "tan(x)");
      legend (h, "location", "southwest");
      set (h, "fontsize", 14,"fontname", "TimesNewRoman");
      title("Some functions plotted","fontname", "TimesNewRoman");
      set(gca, "fontsize",20,"fontname", "TimesNewRoman")
```

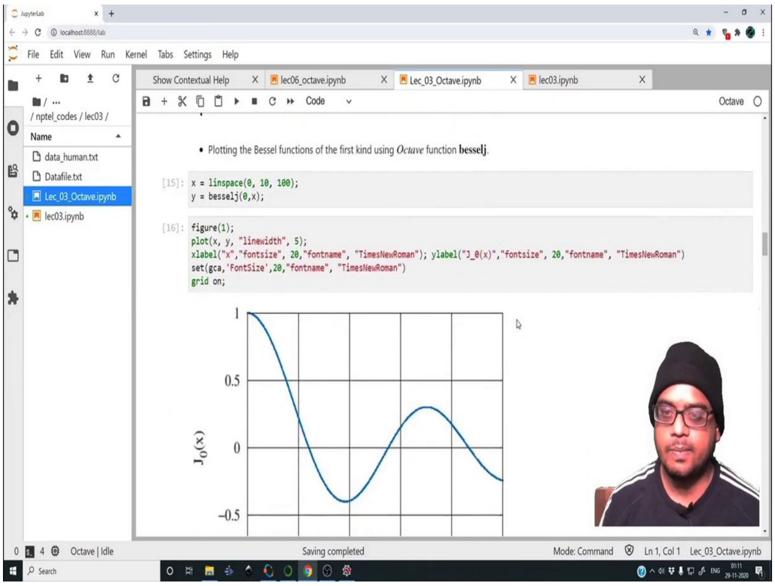
(Refer Slide Time: 04:00)



```
set (h, "fontsize", 14,"fontname", "TimesNewRoman");
title("Some functions plotted","fontname", "TimesNewRoman");
set(gca, "fontsize",20,"fontname", "TimesNewRoman")
```

Special functions in Octave

(Refer Slide Time: 04:02)



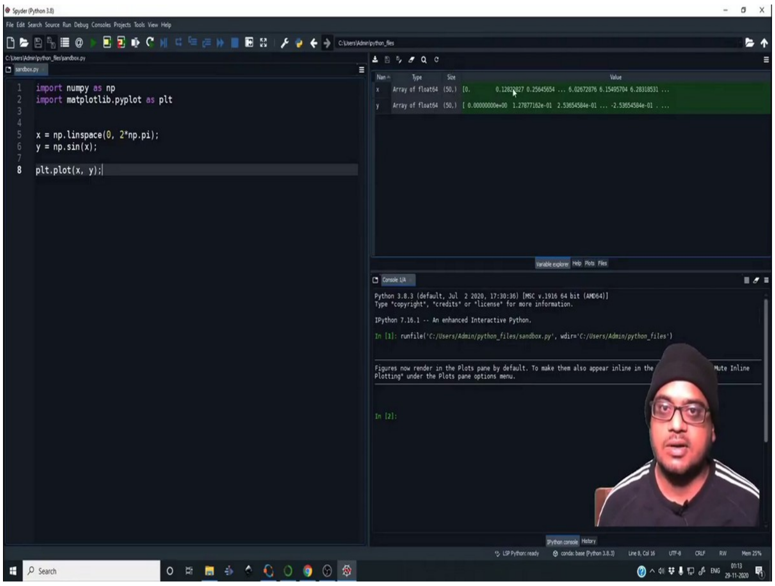
- Plotting the Bessel functions of the first kind using *Octave* function **besselj**.

```
[15]: x = linspace(0, 10, 100);
      y = besselj(0,x);
```

```
[16]: figure(1);
      plot(x, y, "linewidth", 5);
      xlabel("x","fontsize", 20,"fontname", "TimesNewRoman"); ylabel("J_0(x)","fontsize", 20,"fontname", "TimesNewRoman")
      set(gca,'FontSize',20,"fontname", "TimesNewRoman")
      grid on;
```

It is how to make a notebook file, you even embedded everything in a single file. But, you can very well use this GUI, actually the same kind of GUI also exists for Python as well.

(Refer Slide Time: 04:14)



Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Admin\python_files
C:\Users\Admin\python_files\sandbox.py
sandbox.py
Variable explorer Help Plots Files
Console 1/A
IPython console History

```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(0, 2*np.pi);
y = np.sin(x);

plt.plot(x, y);
```

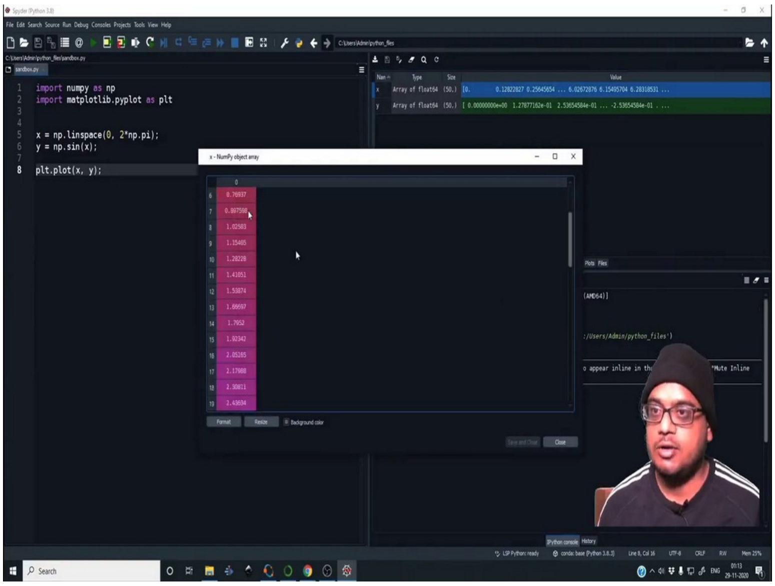
```
Nan  Type             Size   Value
x    Array of float64 (50,)  [0.         0.12822827 0.25645654 ... 6.02672876 6.15495704 6.28318531 ...
y    Array of float64 (50,)  [ 0.00000000e+00  1.27877162e-01  2.53654584e-01 ... -2.53654584e-01 . ...
```

```
Python 3.8.3 (default, Jul  2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Admin/python_files/sandbox.py', wdir='C:/Users/Admin/python_files')

Figures now render in the Plots pane by default. To make them also appear inline in the       Mute Inline
Plotting" under the Plots pane options menu.

In [2]:
```

And that GUI is called as Spyder. This particular version is Spyder 4 and it is running a Python kernel 3.8.3 ok. So, let me show you whatever we have written down in the notebook file, it can execute the same thing over here. So, import numpy as np, import matplotlib.pyplot as plt x = np.linspace(0,2*np.pi) ,y=np.sin (x) alright plt.plot(x, y).

So, when we press F5 on this, we would have run this entire script ok. So, upon pressing F5 let us see what the GUI of Spyder tells us. So, there is a tab called as Variable explorer and it shows all the variables that are available to us. So, it says we have 2 variables x and y and they are of type float64. The size of these things are 50, these are the contents.

(Refer Slide Time: 05:35)



Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Admin\python_files
C:\Users\Admin\python_files\sandbox.py
sandbox.py
x - NumPy object array
Format
Resize
Background color
Save and Close
Close
Plots Files
IPython console History

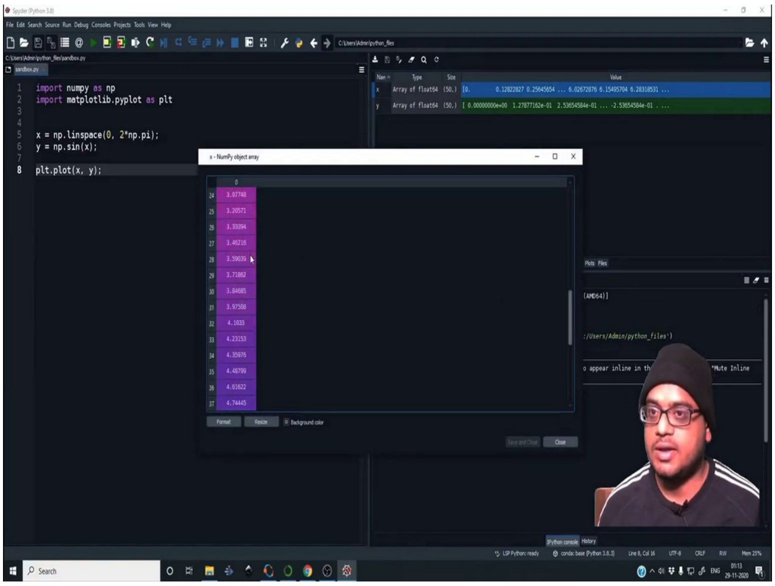
```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(0, 2*np.pi);
y = np.sin(x);

plt.plot(x, y);
```

Name	Type	Size	Value
x	Array of float64	(50,)	[0. 0.12822827 0.25645654 ... 6.02672876 6.15495704 6.28318531 ...
y	Array of float64	(50,)	[0.00000000e+00 1.27877162e-01 2.53654584e-01 ... -2.53654584e-01 . ...

	0
6	0.76937
7	0.897598
8	1.02583
9	1.15405
10	1.28228
11	1.41051
12	1.53874
13	1.66697
14	1.7952
15	1.92342
16	2.05165
17	2.17988
18	2.30811
19	2.43634

```
(AMD64)]
:/Users/Admin/python_files')
o appear inline in th        "Mute Inline
```

(Refer Slide Time: 05:35)



Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Admin\python_files
C:\Users\Admin\python_files\sandbox.py
sandbox.py
x - NumPy object array
Format
Resize
Background color
Save and Close
Close
Plots Files
IPython console History

```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(0, 2*np.pi);
y = np.sin(x);

plt.plot(x, y);
```

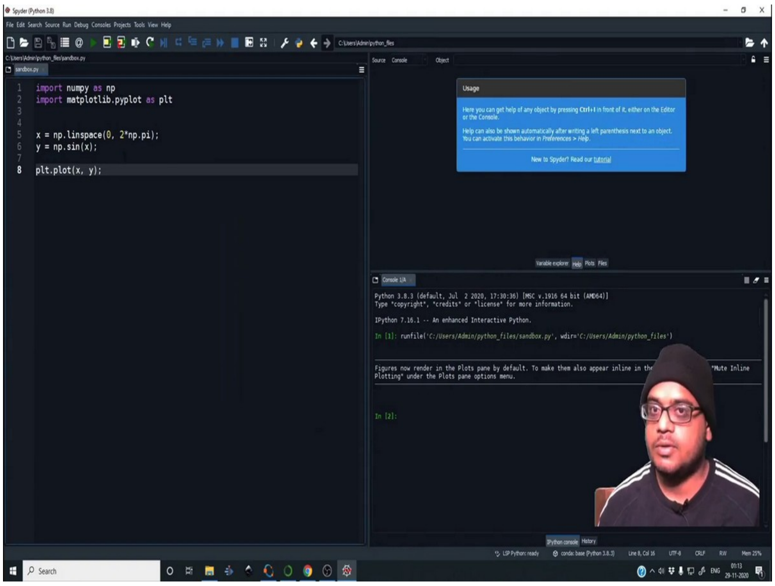
Name	Type	Size	Value
x	Array of float64	(50,)	[0. 0.12822827 0.25645654 ... 6.02672876 6.15495704 6.28318531 ...
y	Array of float64	(50,)	[0.00000000e+00 1.27877162e-01 2.53654584e-01 ... -2.53654584e-01 . ...

	0
24	3.07748
25	3.20571
26	3.33394
27	3.46216
28	3.59039
29	3.71862
30	3.84685
31	3.97508
32	4.1033
33	4.23153
34	4.35976
35	4.48799
36	4.61622
37	4.74445

```
(AMD64)]
:/Users/Admin/python_files')
o appear inline in th        "Mute Inline
```

If we double click on it, we can browse through the content.

(Refer Slide Time: 05:41)



Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Admin\python_files
C:\Users\Admin\python_files\sandbox.py
sandbox.py
Source Console Object
Variable explorer Help Plots Files
Console 1/A
IPython console History

```python
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(0, 2*np.pi);
y = np.sin(x);

plt.plot(x, y);
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

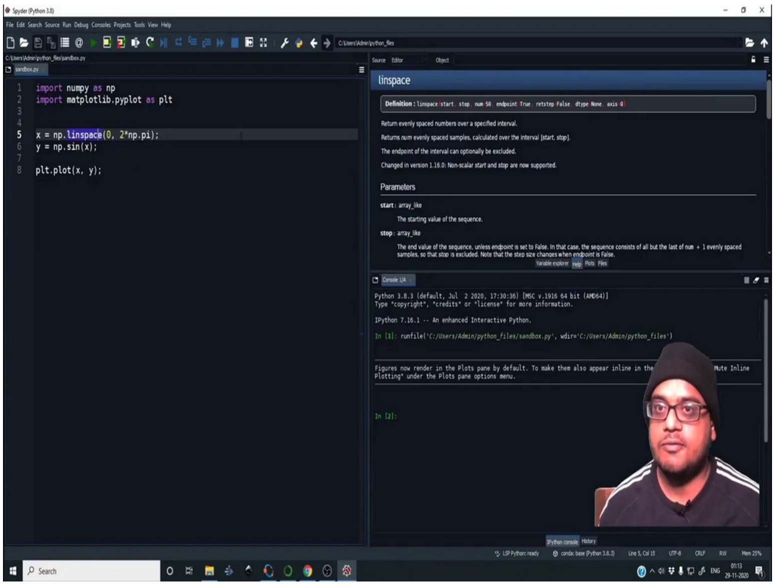
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
Python 3.8.3 (default, Jul  2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Admin/python_files/sandbox.py', wdir='C:/Users/Admin/python_files')


Figures now render in the Plots pane by default. To make them also appear inline in the       "Mute Inline
Plotting" under the Plots pane options menu.


In [2]:
```

There is another tab called as Help. So, let us press control I in place of this.

(Refer Slide Time: 05:49)



Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Admin\python_files
C:\Users\Admin\python_files\sandbox.py
sandbox.py
Source Editor Object
Variable explorer Help Plots Files
Console 1/A
IPython console History

```python
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(0, 2*np.pi);
y = np.sin(x);

plt.plot(x, y);
```

linspace

Definition : linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)

Return evenly spaced numbers over a specified interval.

Returns *num* evenly spaced samples, calculated over the interval [*start*, *stop*].

The endpoint of the interval can optionally be excluded.

Changed in version 1.16.0: Non-scalar *start* and *stop* are now supported.

Parameters

start : array_like

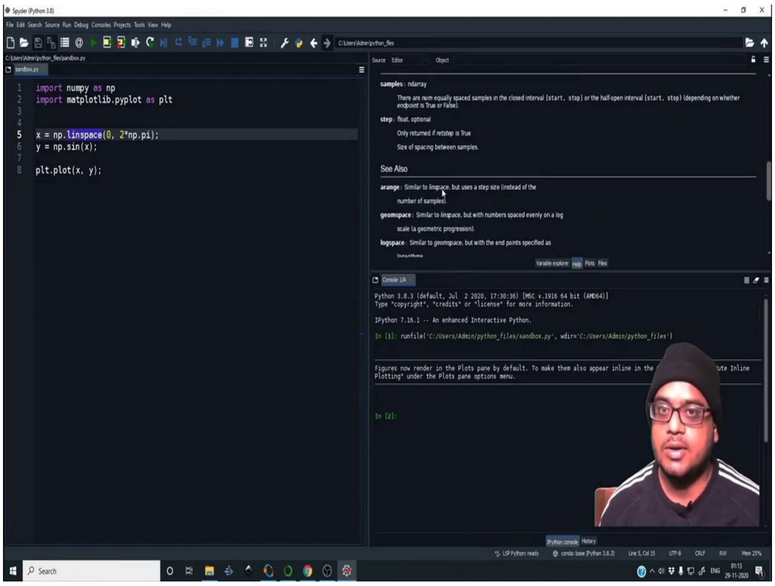
The starting value of the sequence.

stop : array_like

The end value of the sequence, unless *endpoint* is set to False. In that case, the sequence consists of all but the last of num + 1 evenly spaced samples, so that *stop* is excluded. Note that the step size changes when *endpoint* is False.

```
Python 3.8.3 (default, Jul  2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Admin/python_files/sandbox.py', wdir='C:/Users/Admin/python_files')


Figures now render in the Plots pane by default. To make them also appear inline in the       Mute Inline
Plotting" under the Plots pane options menu.


In [2]:
```

(Refer Slide Time: 05:53)



```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(0, 2*np.pi);
y = np.sin(x);

plt.plot(x, y);
```

samples : ndarray

There are *num* equally spaced samples in the closed interval [start, stop] or the half-open interval [start, stop) (depending on whether endpoint is True or False).

step : float, optional

Only returned if *retstep* is True

Size of spacing between samples.

See Also

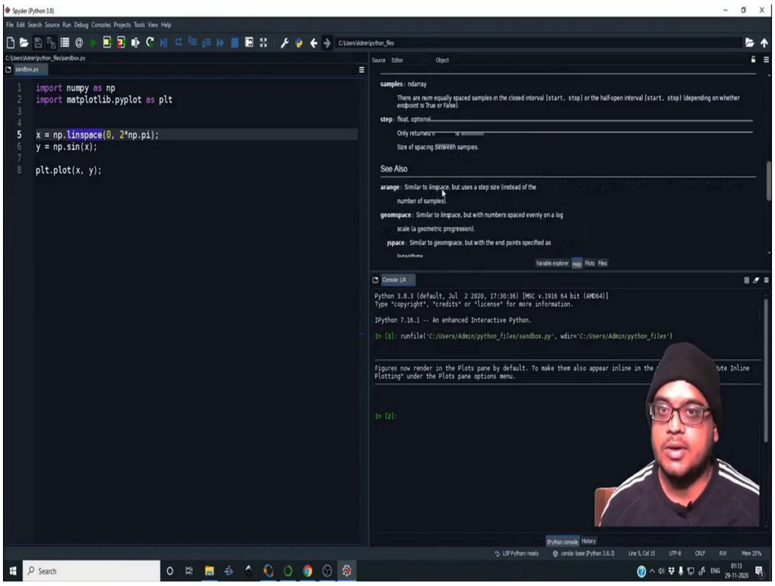
arange : Similar to *linspace*, but uses a step size (instead of the number of samples).

geomspace : Similar to *linspace*, but with numbers spaced evenly on a log scale (a geometric progression).

logspace : Similar to *geomspace*, but with the end points specified as logarithms

```
Python 3.8.3 (default, Jul  2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Admin/python_files/sandbox.py', wdir='C:/Users/Admin/python_files')

Figures now render in the Plots pane by default. To make them also appear inline in the ute Inline
Plotting" under the Plots pane options menu.

In [2]:
```

(Refer Slide Time: 05:55)



```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(0, 2*np.pi);
y = np.sin(x);

plt.plot(x, y);
```

samples : ndarray

There are *num* equally spaced samples in the closed interval [start, stop] or the half-open interval [start, stop) (depending on whether endpoint is True or False).

step : float, optional

Only returned

Size of spacing between samples.

See Also

arange : Similar to *linspace*, but uses a step size (instead of the number of samples).

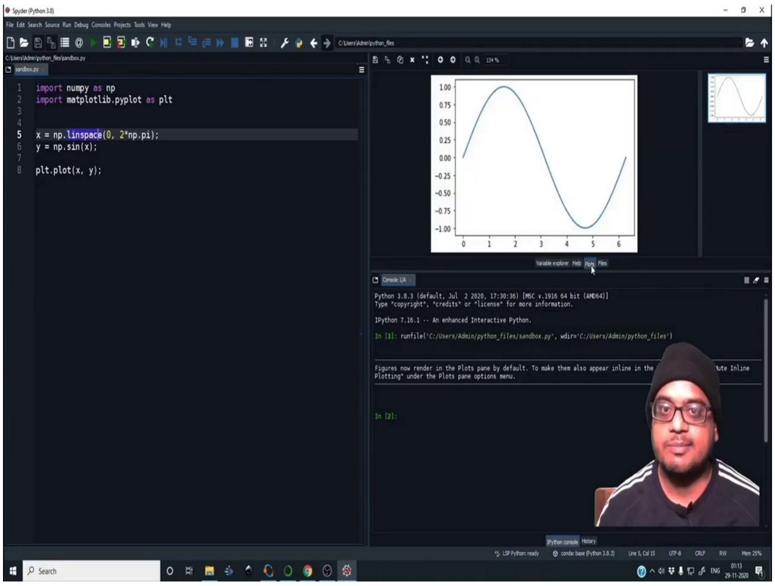
geomspace : Similar to *linspace*, but with numbers spaced evenly on a log scale (a geometric progression).

space : Similar to *geomspace*, but with the end points specified as logarithms

```
Python 3.8.3 (default, Jul  2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Admin/python_files/sandbox.py', wdir='C:/Users/Admin/python_files')

Figures now render in the Plots pane by default. To make them also appear inline in the ute Inline
Plotting" under the Plots pane options menu.

In [2]:
```

When we press Ctrl I, it will fetch the documentation for that particular function. It is the same thing as contextual help that we were using in Jupyter Lab alright.

(Refer Slide Time: 06:02)



Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Admin\python_files
C:\Users\Admin\python_files\sandbox.py
sandbox.py
134 %
Variable explorer Help Plots Files
Console 1/A
IPython console History

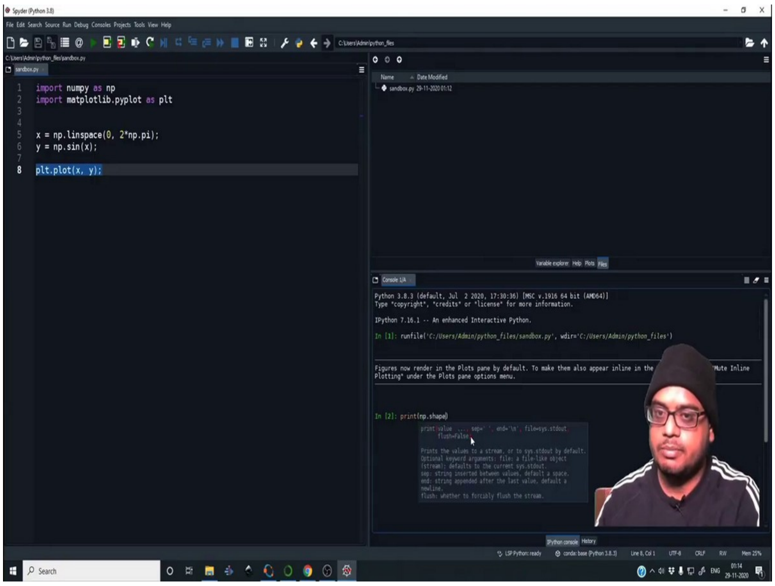
```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(0, 2*np.pi);
y = np.sin(x);

plt.plot(x, y);
```

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5
6

```
Python 3.8.3 (default, Jul  2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Admin/python_files/sandbox.py', wdir='C:/Users/Admin/python_files')

Figures now render in the Plots pane by default. To make them also appear inline in the            ute Inline
Plotting" under the Plots pane options menu.

In [2]:
```

There is another tab called as Plots and whatever we have plotted shows up in this particular tab and the last tab is Files. So, in this particular folder C Users Admin and python files, we have a file called sandbox.py. So, this is the overall structure, this is called as the command window or I mean in MATLAB it is called as command window. In Python it is called as the IPython console. So, IPython is like an interactive Python console. So, we can run commands over here as well.

(Refer Slide Time: 06:44)

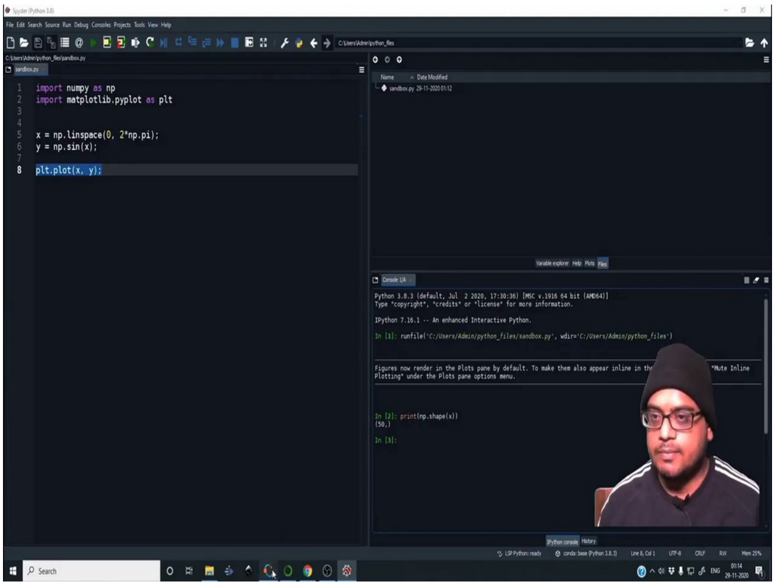


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Admin\python_files
C:\Users\Admin\python_files\sandbox.py
sandbox.py
Name
Date Modified
sandbox.py 29-11-2020 01:12
Variable explorer Help Plots Files
Console 1/A
IPython console History
LSP Python: ready
conda: base (Python 3.8.3)
Line 8, Col 1
UTF-8
CRLF
RW
Mem 25%
ENG
01:14
29-11-2020

```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(0, 2*np.pi);
y = np.sin(x);

plt.plot(x, y);
```

```
Python 3.8.3 (default, Jul  2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Admin/python_files/sandbox.py', wdir='C:/Users/Admin/python_files')

Figures now render in the Plots pane by default. To make them also appear inline in the            Mute Inline
Plotting" under the Plots pane options menu.

In [2]: print(np.shape)
            print(value, ..., sep=' ', end='\n', file=sys.stdout,
                  flush=False)

            Prints the values to a stream, or to sys.stdout by default.
            Optional keyword arguments: file: a file-like object
            (stream); defaults to the current sys.stdout.
            sep: string inserted between values, default a space.
            end: string appended after the last value, default a
            newline.
            flush: whether to forcibly flush the stream.
```

(Refer Slide Time: 06:51)

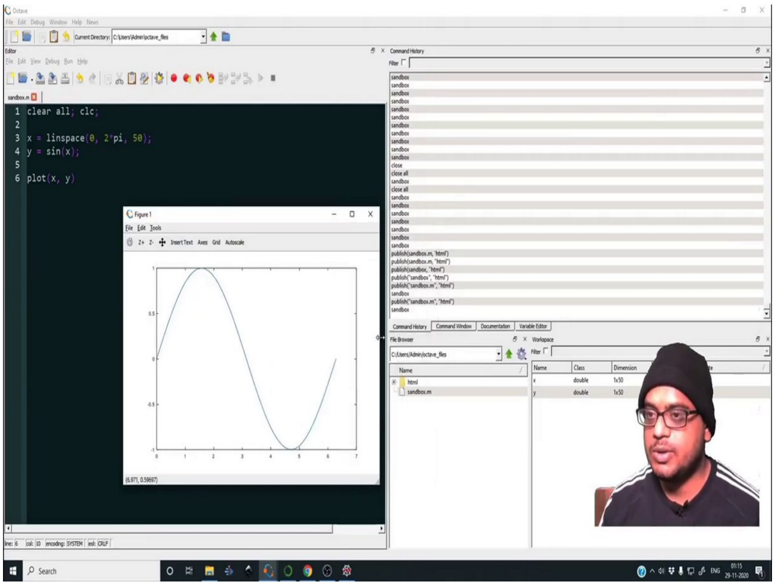


```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(0, 2*np.pi);
y = np.sin(x);

plt.plot(x, y);
```

```
Python 3.8.3 (default, Jul  2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Admin/python_files/sandbox.py', wdir='C:/Users/Admin/python_files')

In [2]: print(np.shape(x))
(50,)

In [3]:
```

So, let us say we want to fetch the shape of x. So, we can simply say print np.shape(x). So, it says 50 comma 0. So, this window is like you can execute single commands, but if you want to execute a string of commands you have to make a script.

(Refer Slide Time: 07:10)



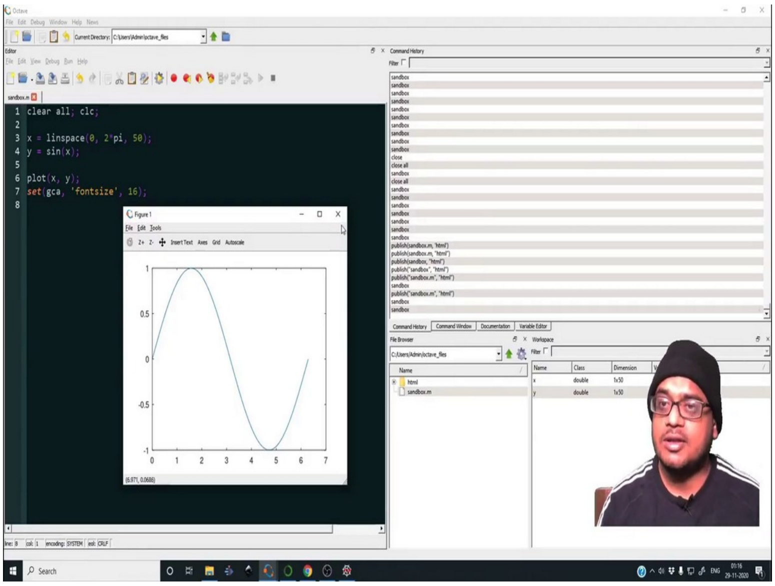
```
clear all; clc;

x = linspace(0, 2*pi, 50);
y = sin(x);

plot(x, y)
```

Name	Class	Dimension
x	double	1x50
y	double	1x50

Similarly, in Octave; so, let me clear all this. So, clear all is to clear all the variables; clc is to clear the command window. So, we have the command window over here, we have the variable explorer over here. It is the same structure and more or less all the GUIs try to keep the same structure. We have the files also, additionally we have a command history, a documentation, a variable editor.

So, let us do this. So, x equal to similar to the numpy linspace, there is an equivalent in Octave as well. In fact, there is an equivalent in MATLAB as well. So, linspace(0,2*pi, 50), let us say we have 50 points, y=sin(x); just for completeness; so, sin (x) plot(x,y) . So, when we press F5, the script will be executed and we will have the plot ok.

(Refer Slide Time: 08:16)


Octave
File Edit Debug Window Help News
Current Directory: C:\Users\Admin\octave_files
Editor
sandbox.m
Figure 1
File Edit Tools
Z+ Z- Insert Text Axes Grid Autoscale
1
0.5
0
-0.5
-1
0 1 2 3 4 5 6 7
Command History
Filter
Command History Command Window Documentation Variable Editor
File Browser
C:/Users/Admin/octave_files
Name
html
sandbox.m
Workspace

```
clear all; clc;

x = linspace(0, 2*pi, 50);
y = sin(x);

plot(x, y);
set(gca, 'fontsize', 16);
```

```
sandbox
sandbox
sandbox
sandbox
sandbox
sandbox
sandbox
sandbox
sandbox
sandbox
close
close all
sandbox
close all
sandbox
sandbox
sandbox
sandbox
sandbox
sandbox
sandbox
publish(sandbox.m, 'html')
publish(sandbox.m, "html")
publish(sandbox, "html")
publish("sandbox", "html")
publish("sandbox.m", "html")
sandbox
publish("sandbox.m", "html")
sandbox
sandbox
```

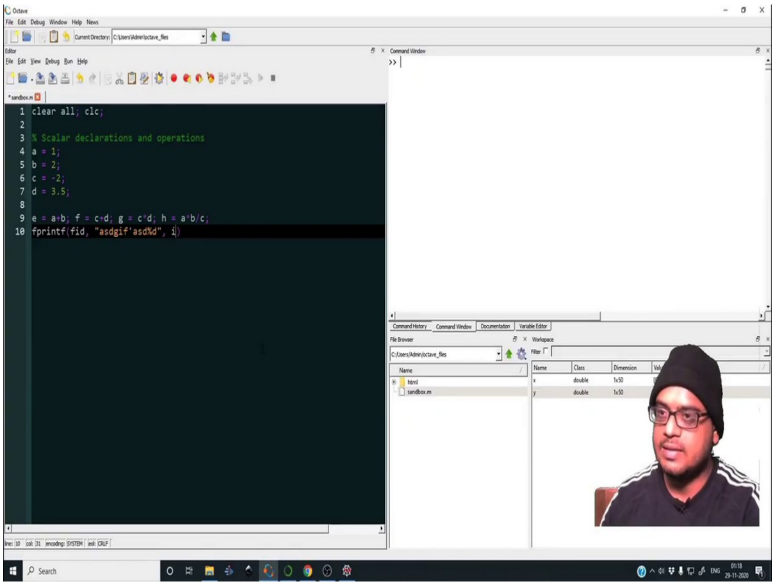
Name	Class	Dimension
x	double	1x50
y	double	1x50

So, the plot may seem a bit anemic at the moment. We can increase the font size as by set(gca, ‘fontsize’, 16). Once we execute this, we see that the font size has increased. So, whatever we have done in a Jupyter Lab, we can execute it equivalently on a script as well.

Do not think that you have to do everything in Jupyter Lab, Jupyter Lab helps me to organize whatever I am trying to teach or at least give you a background on. And, whatever codes I am writing down to supplement that information in a single file; it helps me export an html which I can upload to my website.

Apart from that actually you can export these scripts also as an html, it takes no extra effort; there are various functions to do that. So, without further ado let us get into some of the basics of GNU Octave, some loops, some conditionals. And, hopefully with the help of this and the uploaded files on the website you will be in a very good position to start making programs in GNU Octave as well as Python.

(Refer Slide Time: 09:34)



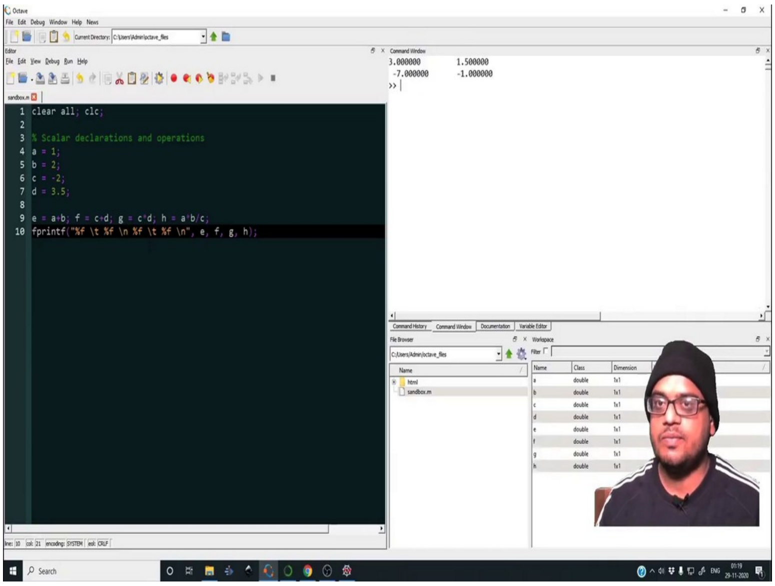
```
clear all; clc;

% Scalar declarations and operations
a = 1;
b = 2;
c = -2;
d = 3.5;

e = a+b; f = c+d; g = c*d; h = a*b/c;
fprintf(fid, "asdgif'asd%d", i)
```

So, clear all clc. So, let us define scalars. So, comments are written using a hash(#) sign. In fact, they can be written with the percentage(%) sign as well, does not make a difference. So, Scalar declaration and operations; so, let us say a=1, b = 2, c = -2, d =3.5.

So, then we can define e =a + b; f = c + d; g =c *d; h =a * b/ c and we can print all of them. So, in Python we use print commands directly, but in GNU Octave or in fact, even in MATLAB we must use the fprintf command.

So, the fprintf command if you know a bit of C, it is a printing command which prints to a file; so, instead of printing to a file. So, typically in C what would you do? You would given a file id, then you would have whatever you want to print and if you have some literals, that you want to print some placeholders then you would give something like this.

(Refer Slide Time: 10:57)



```
clear all; clc;

% Scalar declarations and operations
a = 1;
b = 2;
c = -2;
d = 3.5;

e = a+b; f = c+d; g = c*d; h = a*b/c;
fprintf("%f \t %f \n %f \t %f \n", e, f, g, h);
```

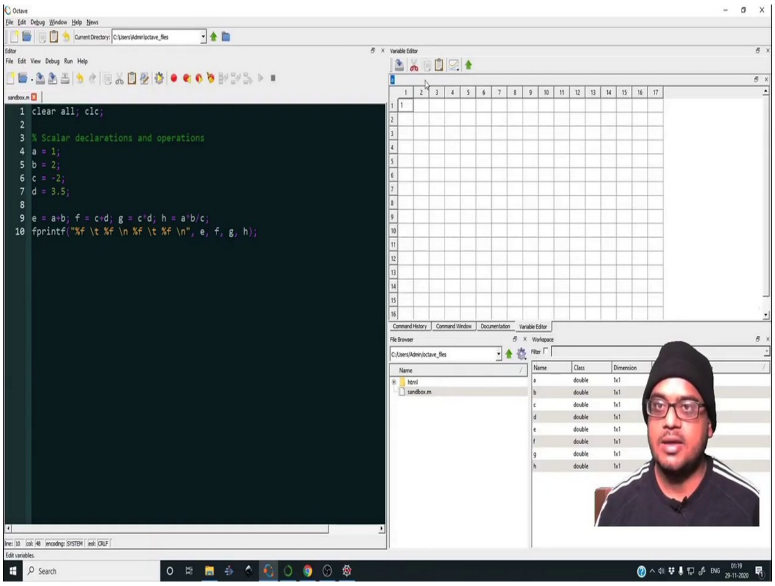
```
3.000000         1.500000
 -7.000000       -1.000000
>>
```

Name	Class	Dimension
a	double	1x1
b	double	1x1
c	double	1x1
d	double	1x1
e	double	1x1
f	double	1x1
g	double	1x1
h	double	1x1

But over here if you omit the file id, it will by default print to the command window ok. So, fprintf, let us print out "%f \t %f \t %f \t %f \n" and here we will print out e, f, g, h.

So, \t means you give a tab, \n means you go to the new line ok. In fact, let us put a \n over here as well for good measure. So, once we execute this, we would have all the different values printed as a floating point number in the command window. And, these scalar operations are as shown over here.

(Refer Slide Time: 12:02)

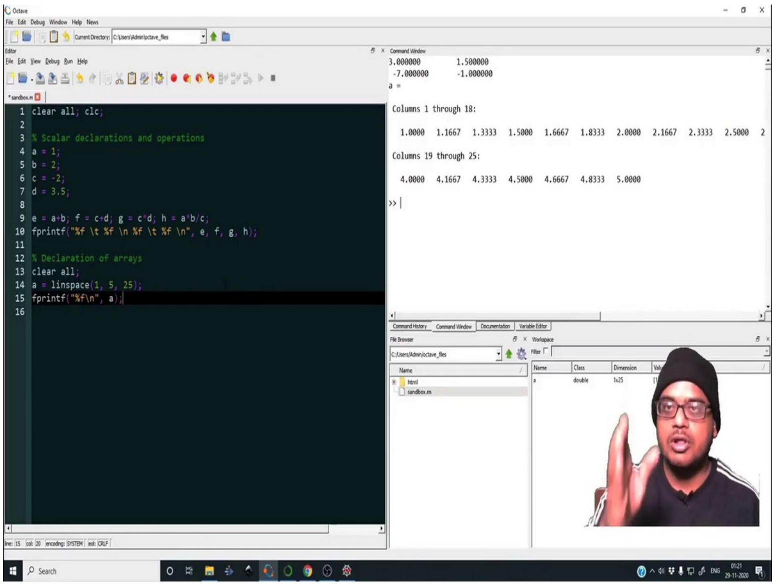


```
clear all; clc;

% Scalar declarations and operations
a = 1;
b = 2;
c = -2;
d = 3.5;

e = a+b; f = c+d; g = c*d; h = a*b/c;
fprintf("%f \t %f \n %f \t %f \n", e, f, g, h);
```

Name	Class	Dimension
a	double	1x1
b	double	1x1
c	double	1x1
d	double	1x1
e	double	1x1
f	double	1x1
g	double	1x1
h	double	1x1

The variables are available in the workspace editor as well. If we go into the Variable Editor and if you double click on a, it will show a matrix where you can manipulate the variable as per your liking; I do not suggest you use all these, but if need be you can use it. So, let us go to the command window ok. So, this is how you can do a manipulation of scalars and you can print them out.

(Refer Slide Time: 12:26)



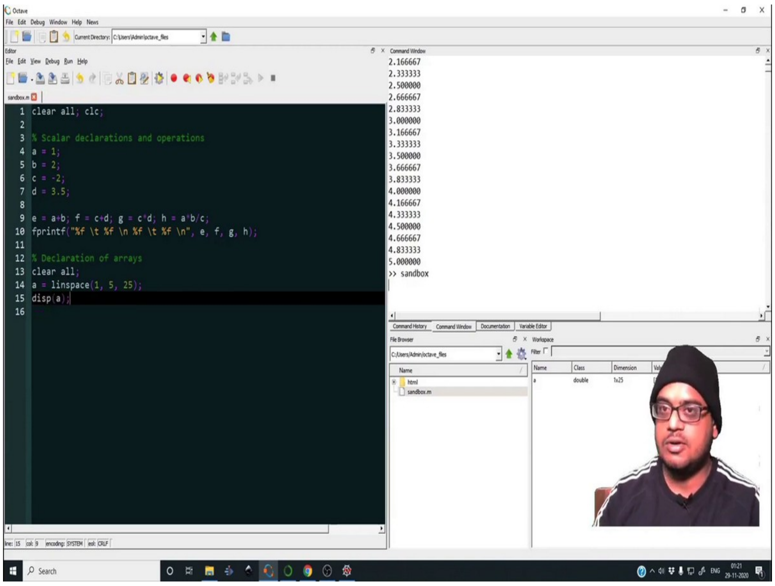
```
clear all; clc;

% Scalar declarations and operations
a = 1;
b = 2;
c = -2;
d = 3.5;

e = a+b; f = c+d; g = c*d; h = a*b/c;
fprintf("%f \t %f \n %f \t %f \n", e, f, g, h);

% Declaration of arrays
clear all;
a = linspace(1, 5, 25);
fprintf("%f\n", a);

```

```
3.000000        1.500000
 -7.000000       -1.000000
a =

 Columns 1 through 18:

   1.0000   1.1667   1.3333   1.5000   1.6667   1.8333   2.0000   2.1667   2.3333   2.5000   2

 Columns 19 through 25:

   4.0000   4.1667   4.3333   4.5000   4.6667   4.8333   5.0000

>> 
```

Now, let us declare some arrays alright. So, a =linspace(1, 5, 25) and let us say we have 25, points b equal to ok. So, let me just declare this as of now. So, before doing this we can clear the variable space. So, it sort of clears all the variables that exist up to that point, let me run this. So, so far because we have cleared all the variables only a should exist.

So, if I go to the command window and type on a, it will give me all the values of a. So, here is the thing in GNU Octave and in MATLAB as well, if you omit the semicolon in the end; you end up printing out that value ok, here is how it looks like. You could alternately choose to print. So, fprintf (“%f \n”,a). So, this will print out all the elements of a with a new line at the end of each element, let me execute this.

(Refer Slide Time: 13:46)

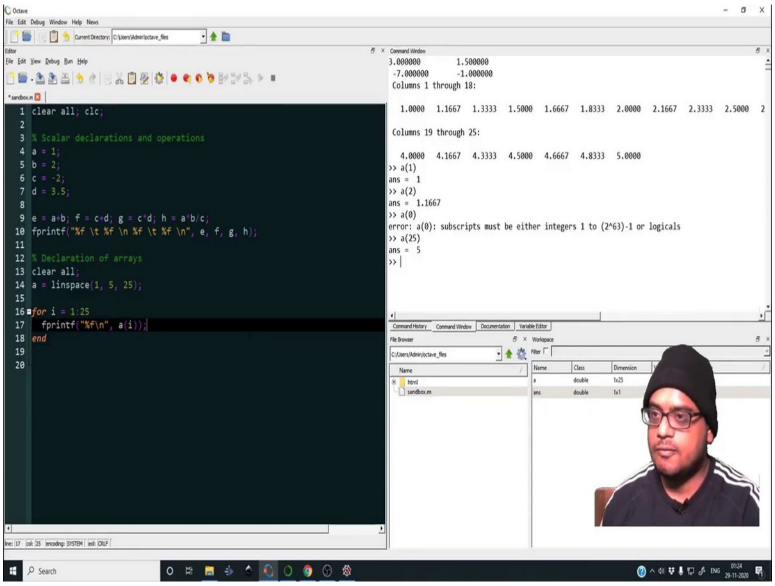


```
clear all; clc;

% Scalar declarations and operations
a = 1;
b = 2;
c = -2;
d = 3.5;

e = a+b; f = c+d; g = c*d; h = a*b/c;
fprintf("%f \t %f \n %f \t %f \n", e, f, g, h);

% Declaration of arrays
clear all;
a = linspace(1, 5, 25);
disp(a);
```

```
2.166667
2.333333
2.500000
2.666667
2.833333
3.000000
3.166667
3.333333
3.500000
3.666667
3.833333
4.000000
4.166667
4.333333
4.500000
4.666667
4.833333
5.000000
>> sandbox
```

So, this is how all the elements of a can be printed out ok. Apart from this you can simply write down a over here or you can set this a; this also does the same thing. So, this is how you can display the elements of a alright. So, with this information, let us do a loop over all the elements of a.

(Refer Slide Time: 14:19)



```
clear all; clc;

% Scalar declarations and operations
a = 1;
b = 2;
c = -2;
d = 3.5;

e = a+b; f = c+d; g = c*d; h = a*b/c;
fprintf("%f \t %f \n %f \t %f \n", e, f, g, h);

% Declaration of arrays
clear all;
a = linspace(1, 5, 25);

for i = 1:25
  fprintf("%f\n", a(i));
end
```

```
3.000000        1.500000
 -7.000000       -1.000000
 Columns 1 through 18:

   1.0000   1.1667   1.3333   1.5000   1.6667   1.8333   2.0000   2.1667   2.3333   2.5000   2

 Columns 19 through 25:

   4.0000   4.1667   4.3333   4.5000   4.6667   4.8333   5.0000
>> a(1)
ans =  1
>> a(2)
ans =  1.1667
>> a(0)
error: a(0): subscripts must be either integers 1 to (2^63)-1 or logicals
>> a(25)
ans =  5
>>
```

So, we know that a contains 25 elements. So, we can do the following for i = 1:25 end; so, it is a good idea to always match a for and an end. So, in Python you do not need to really end a for loop, you just remove the indentation and the for loop gets restored.

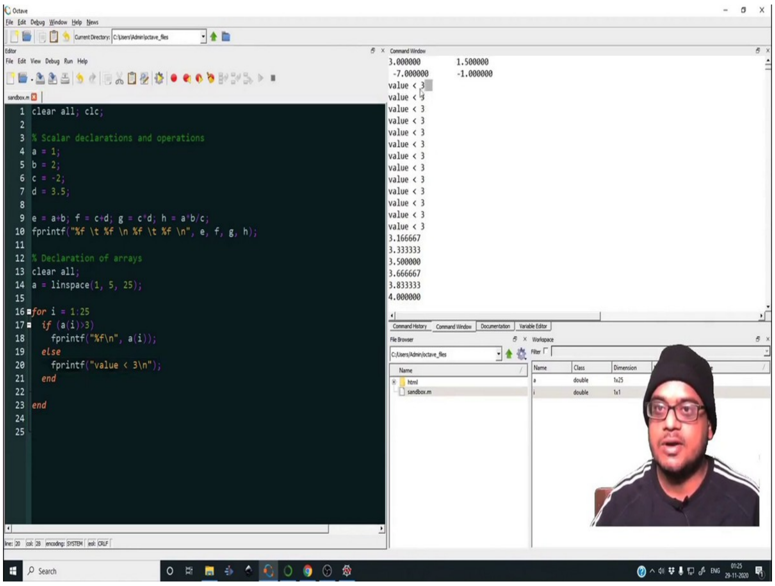
While, the for loop gets terminated, while in the case of GNU Octave, you have to explicitly declare that this line is the ending of the for loop. Indentation plays no role in GNU Octave, this is same in MATLAB ok. In the case of Python, you just remove the indentation to terminate a loop, it is as simple as that.

So, let us go over here and let us print all the values of a. So, we will do fprintf(“%f \n”, a(i)). So, in GNU Octave and MATLAB the indexing starts from 1. So, a 1 is 1, a 2 is 1.667 and so on. While, a 0 if you try to print this out, it will throw an error because subscripts cannot go, the array subscript cannot go below 1.

Moreover, the last element is a 25, it is equal to 5. In Python it is equal to n minus 1. So, in Python it would have been something like this, this would have been the last element alright. So, that is the small difference in indexing. So, Python follows C indexing, while GNU Octave and MATLAB follow an indexing which starts from 1.

Each has their own minute and limit, but truly speaking if you have an idea in mind, you should face no difficulty to implement it in either language; that is the; that is the way your mind thinks about it. It should be clear what you are doing, it should be clear in an abstract way about what you really want to achieve. And, then it all becomes a matter of semantics ok. So, we have written down the code, let me execute the code. So, it prints out all the elements of a; now we can put a conditional on a.

(Refer Slide Time: 16:47)



```
clear all; clc;

% Scalar declarations and operations
a = 1;
b = 2;
c = -2;
d = 3.5;

e = a+b; f = c+d; g = c*d; h = a*b/c;
fprintf("%f \t %f \n %f \t %f \n", e, f, g, h);

% Declaration of arrays
clear all;
a = linspace(1, 5, 25);

for i = 1:25
  if (a(i)>3)
    fprintf("%f\n", a(i));
  else
    fprintf("value < 3\n");
  end

end
```

```
3.000000        1.500000
 -7.000000       -1.000000
value < 3
value < 3
value < 3
value < 3
value < 3
value < 3
value < 3
value < 3
value < 3
value < 3
value < 3
value < 3
value < 3
3.166667
3.333333
3.500000
3.666667
3.833333
4.000000
```

Current Directory: C:\Users\Admin\octave_files

File Browser: C:/Users/Admin/octave_files

Name	Class	Dimension
a	double	1x25
i	double	1x1

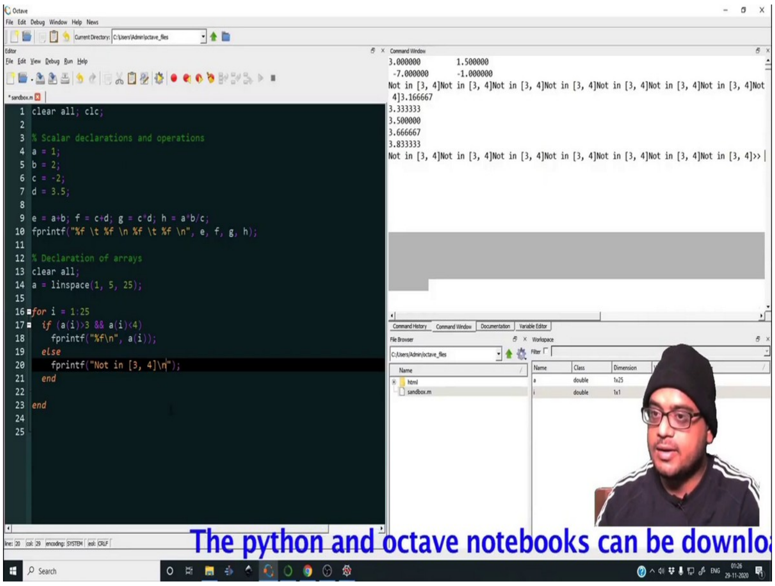
So, we can say that only if a > 3 ok, if a >3 we want to print out the elements or say we want let us let us do that first. So, if a(i) >3, then you do this else you say that value less than 3 and then we end the if else block as well.

So, the if else block has to be also explicitly ended with the end command. Unlike, Python where if else statements are also dependent on indentations, if you indent a code inside an if statement, you end up sort of embedding those lines inside the conditional. In the case of MATLAB and GNU Octave it is just declaring explicitly where you want to end the if else statement ok.

So, let us run this ok. So, value less than 3 ok, we forgot to put a new carriage \n and because of not putting a new line, it is printing everything in a single line. We need to explicitly say that you go to the next line, let me run this again. I press F5 ok. So, we get value less than 3, then the moment it becomes greater than 3; we get values printed out.

So, let us do this, let us print out the values when we satisfy two conditionals and a(i)<4 ok. So, essentially we are telling if a(i)> 3 and a(i) <4, you print out these values. Let me run this, excellent. So, it has only printed me values between 3 and 4.

(Refer Slide Time: 19:00)

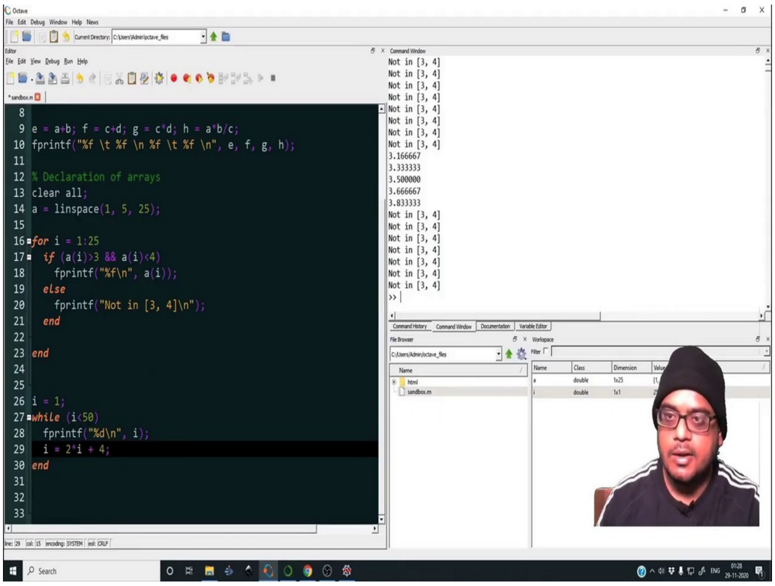


Octave
File Edit Debug Window Help News
Current Directory: C:\Users\Admin\octave_files
Editor
File Edit View Debug Run Help
*sandbox.m
Command Window
line: 20 col: 29 encoding: SYSTEM eol: CRLF
The python and octave notebooks can be downlo

```
clear all; clc;

% Scalar declarations and operations
a = 1;
b = 2;
c = -2;
d = 3.5;

e = a+b; f = c+d; g = c*d; h = a*b/c;
fprintf("%f \t %f \n %f \t %f \n", e, f, g, h);

% Declaration of arrays
clear all;
a = linspace(1, 5, 25);

for i = 1:25
  if (a(i)>3 && a(i)<4)
    fprintf("%f\n", a(i));
  else
    fprintf("Not in [3, 4]\n");
  end

end
```

```
3.000000        1.500000
 -7.000000       -1.000000
Not in [3, 4]Not in [3, 4]Not in [3, 4]Not in [3, 4]Not in [3, 4]Not in [3, 4]Not in [3, 4]Not
 4]3.166667
3.333333
3.500000
3.666667
3.833333
Not in [3, 4]Not in [3, 4]Not in [3, 4]Not in [3, 4]Not in [3, 4]Not in [3, 4]Not in [3, 4]>>
```

(Refer Slide Time: 19:13)



```

e = a+b; f = c+d; g = c*d; h = a*b/c;
fprintf("%f \t %f \n %f \t %f \n", e, f, g, h);

% Declaration of arrays
clear all;
a = linspace(1, 5, 25);

for i = 1:25
  if (a(i)>3 && a(i)<4)
    fprintf("%f\n", a(i));
  else
    fprintf("Not in [3, 4]\n");
  end

end


i = 1;
while (i<50)
  fprintf("%d\n", i);
  i = 2*i + 4;
end
```

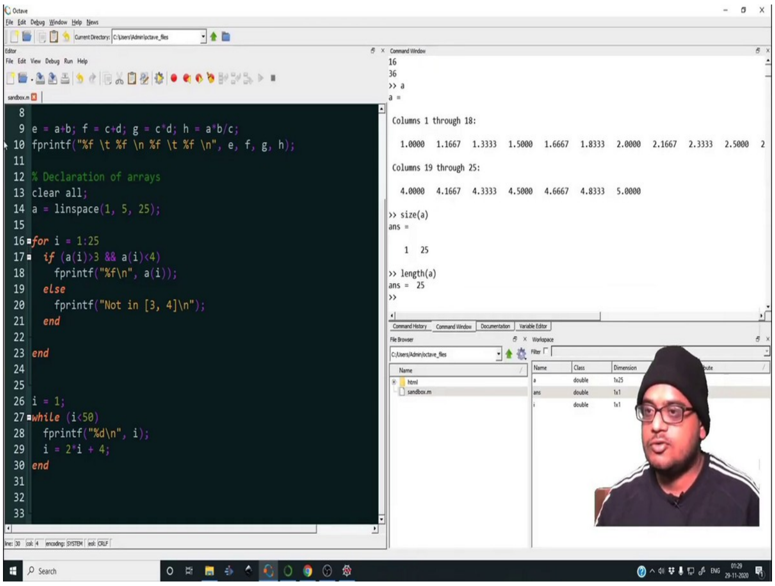
```
Not in [3, 4]
Not in [3, 4]
Not in [3, 4]
Not in [3, 4]
Not in [3, 4]
Not in [3, 4]
Not in [3, 4]
Not in [3, 4]
3.166667
3.333333
3.500000
3.666667
3.833333
Not in [3, 4]
Not in [3, 4]
Not in [3, 4]
Not in [3, 4]
Not in [3, 4]
Not in [3, 4]
Not in [3, 4]
>>
```

So, let us make a meaningful statement, Not in 3 comma 4 ok. So, let me run this again, I forgot the new line character ok. So, this is how you would print out values which lie in 3 to 4. So, in one shot we have covered how to write a loop or at least a for loop and how to do a conditional ok.

So, let us do a while loop as well, this is something which will be quite useful in various contexts. So, while loop is something which executes until a certain criterion is reached. So, let us do this. So, i = 1. So, while i < 50 end so, again you have to explicitly end a while loop. In the case of Python you do not need to do that, you just need to remove the indentation to end the loop.

So, we will do print not print fprintf ("%d \n,i) and let us update i = 2*i+4; just as an example. So, then what do we expect? As i grows there will be a certain point in the loop where i will exceed the value of 50 and once it does that, this loop should terminate.

(Refer Slide Time: 20:49)



```
e = a+b; f = c+d; g = c*d; h = a*b/c;
fprintf("%f \t %f \n %f \t %f \n", e, f, g, h);

% Declaration of arrays
clear all;
a = linspace(1, 5, 25);

for i = 1:25
  if (a(i)>3 && a(i)<4)
    fprintf("%f\n", a(i));
  else
    fprintf("Not in [3, 4]\n");
  end

end


i = 1;
while (i<50)
  fprintf("%d\n", i);
  i = 2*i + 4;
end
```

```
16
36
>> a
a =

 Columns 1 through 18:

   1.0000   1.1667   1.3333   1.5000   1.6667   1.8333   2.0000   2.1667   2.3333   2.5000   2

 Columns 19 through 25:

   4.0000   4.1667   4.3333   4.5000   4.6667   4.8333   5.0000

>> size(a)
ans =

   1   25

>> length(a)
ans =  25
>>
```

Name	Class	Dimension
a	double	1x25
ans	double	1x1
i	double	1x1

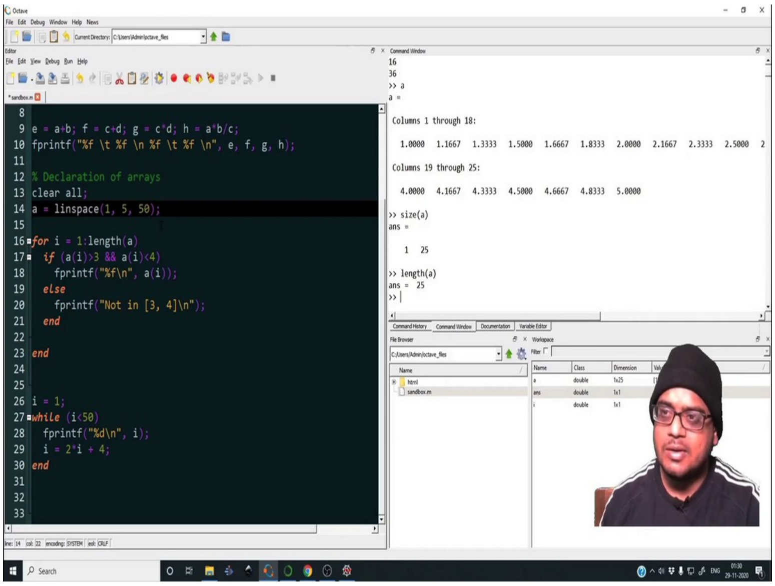
Octave
File Edit Debug Window Help News
Current Directory: C:\Users\Admin\octave_files
Editor
File Edit View Debug Run Help
sandbox.m
line: 30 col: 4 encoding: SYSTEM eol: CRLF
Command Window
Command History
Command Window
Documentation
Variable Editor
File Browser
C:/Users/Admin/octave_files
Name
html
sandbox.m
Workspace
Filter
Search

So, let me run F5. So, i is 1 16 1 6 16 36, the next value would have been larger than 50 and hence the loop terminates. So, inside a while loop we can also put various conditionals; this is particularly useful if you want to run error loops.

So, you want to check for errors in your iterations for example, you will see examples of a while loop in the html file of Octave uploaded on my website for lecture 4 in which were dealing with fixed point iterations. So, there you will see that when the error becomes less than a certain threshold, the loop has to end ok. So, this is particularly useful for all that.

Apart from this we can query the size of a. So, let me go to the command window, I do not want to write this in the main script because there is something which may not be useful at this particular moment. So, let me do this. So, we have we already have a, this is a. So, let me say size (a), it is 1 row and 25 columns. It means ask for length (a), it is 25.

(Refer Slide Time: 22:11)



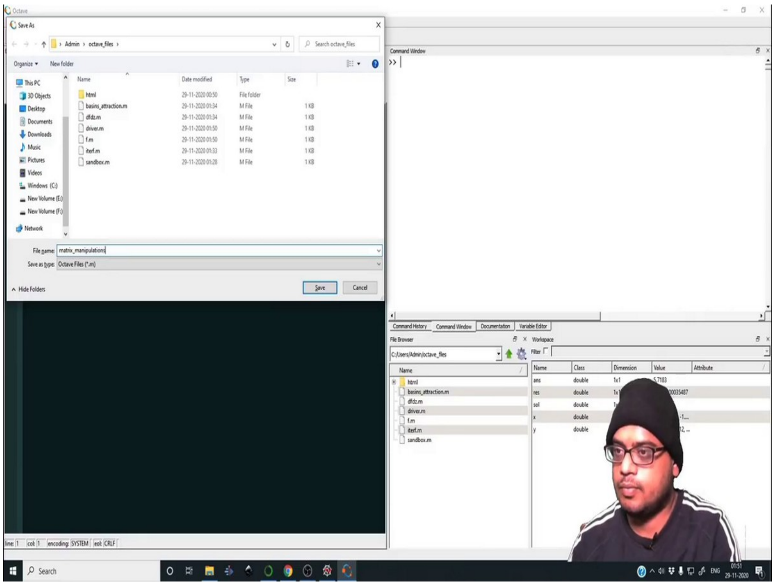
```
e = a+b; f = c+d; g = c*d; h = a*b/c;
fprintf("%f \t %f \n %f \t %f \n", e, f, g, h);

% Declaration of arrays
clear all;
a = linspace(1, 5, 50);

for i = 1:length(a)
  if (a(i)>3 && a(i)<4)
    fprintf("%f\n", a(i));
  else
    fprintf("Not in [3, 4]\n");
  end

end


i = 1;
while (i<50)
  fprintf("%d\n", i);
  i = 2*i + 4;
end
```

```
16
36
>> a
a =

 Columns 1 through 18:

   1.0000   1.1667   1.3333   1.5000   1.6667   1.8333   2.0000   2.1667   2.3333   2.5000   2

 Columns 19 through 25:

   4.0000   4.1667   4.3333   4.5000   4.6667   4.8333   5.0000

>> size(a)
ans =

    1   25

>> length(a)
ans =  25
>> |
```

So, instead of hard coding this loop, we could have alternately done this for i=1:length(a). Meaning, I would naturally go from 1 to 25, we do not need to worry about what value we define over here. I could very well define it to be 50 and the loop would have automatically adjusted its length to go from 1 to 50. I do not need to hardcode the value of the for loop ok.

So, this is quite useful in order to execute loops in a very structured fashion. Let us continue with some more relevant things that you should know about arrays and arrays multiplication, element wise multiplication in matrices and all these things.

(Refer Slide Time: 22:59)



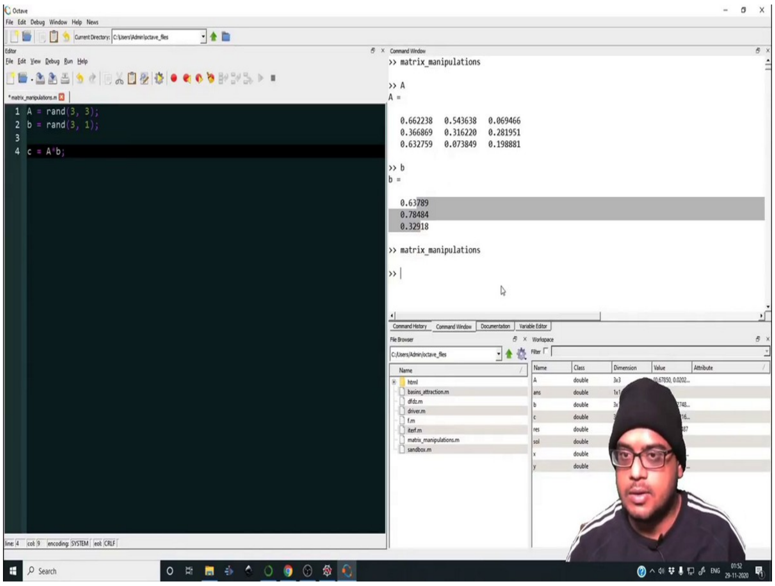
Name	Date modified	Type	Size
html	29-11-2020 00:50	File folder	
basins_attraction.m	29-11-2020 01:34	M File	1 KB
dfdz.m	29-11-2020 01:34	M File	1 KB
driver.m	29-11-2020 01:50	M File	1 KB
f.m	29-11-2020 01:50	M File	1 KB
iterf.m	29-11-2020 01:33	M File	1 KB
sandbox.m	29-11-2020 01:28	M File	1 KB

File name: matrix_manipulations

Save as type: Octave Files (*.m)

So, let me create a new file. Let us save this and let me call it matrix manipulations alright.

(Refer Slide Time: 23:12)



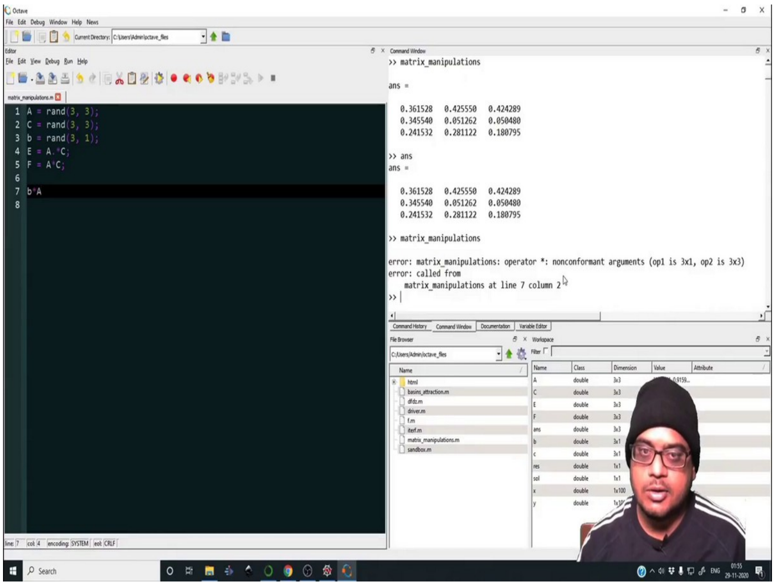
```
A = rand(3, 3);
b = rand(3, 1);

c = A*b;
```

```
>> matrix_manipulations

>> A
A =

   0.662238   0.543638   0.069466
   0.366869   0.316220   0.281951
   0.632759   0.073849   0.198881

>> b
b =

   0.63789
   0.78484
   0.32918

>> matrix_manipulations

>> |
```

So, suppose you create a random array A of size 3 x 3 and you create a random vector b of size 3 x 1, let me run this. So, let us see what A and b are alright. So, now, we can perform c =A *b. So, essentially this is a multiplication of a matrix with an array.

So, the first element of c will be this row multiplying this column. So, this multiplied by this plus this multiplied by this plus this multiplied by this, that will be the first element of c. The second element of c will be this row rather this row multiplying this column and so on. So, let me run this, let me see what c is as well. So, c is valid ok.

(Refer Slide Time: 24:16)



```
A = rand(3, 3);
C = rand(3, 3);
b = rand(3, 1);
E = A.*C;
F = A*C;

b*A

```

```
>> matrix_manipulations

ans =

   0.361528   0.425550   0.424289
   0.345540   0.051262   0.050480
   0.241532   0.281122   0.180795

>> ans
ans =

   0.361528   0.425550   0.424289
   0.345540   0.051262   0.050480
   0.241532   0.281122   0.180795

>> matrix_manipulations

error: matrix_manipulations: operator *: nonconformant arguments (op1 is 3x1, op2 is 3x3)
error: called from
    matrix_manipulations at line 7 column 2
>> |
```

But, if I were to make another random array C=rand(3,3) and if I wanted to multiply each element of A with each element of b; meaning I do not really want a matrix multiplication in the classical sense. But, I want each element to multiply with the corresponding element of C.

Let me show you what exactly I mean. So, let me print out A and C. So, suppose I write E=A .* C versus F =A *C. So, E would contain the first element of E would be; so, (1,1) would be this multiplied by this that is it. Then the second, the first row second column of E would be this multiplied by this, then the first row third column will be this multiplied by this and so on.

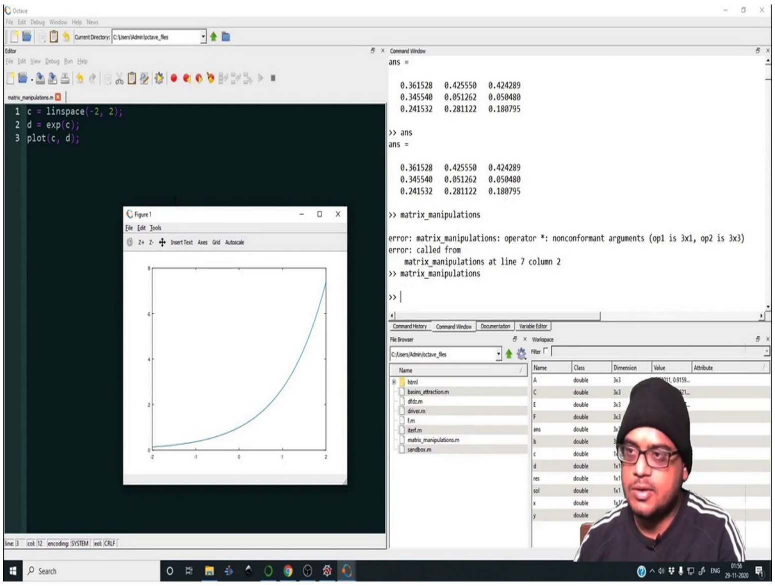
But, F which is A times C is going to do the entire matrix multiplication. So, the first element of F, now the first row first column will be this multiplied by this. Then the first row second column will be this multiplied by this and so on.

So, this is how you can achieve element wise operations. This particularly important because, you do not want to end up doing a bunch of incorrect calculations; because you thought that it would do element wise multiplication, but instead it did matrix multiplication. In fact, element wise operation of A and b would be forbidden.

So, A .* b; what is A .* b? Well, it does do something, but ideally you want to avoid all these because b and A are not of the same size, b *A would be forbidden because the yeah. So, it is non conformant arguments, because the size of b. So, the size(b) is 3x1 and you cannot multiply 3x1 array with a 3x3 matrix.

You can multiply a 3x3 matrix with a 3x1 vector that is perfectly fine alright. So, this is how you would go around doing it and all the same rules apply for division and typically operations such as exponentiation, they are broadcast over all the elements.

(Refer Slide Time: 27:13)



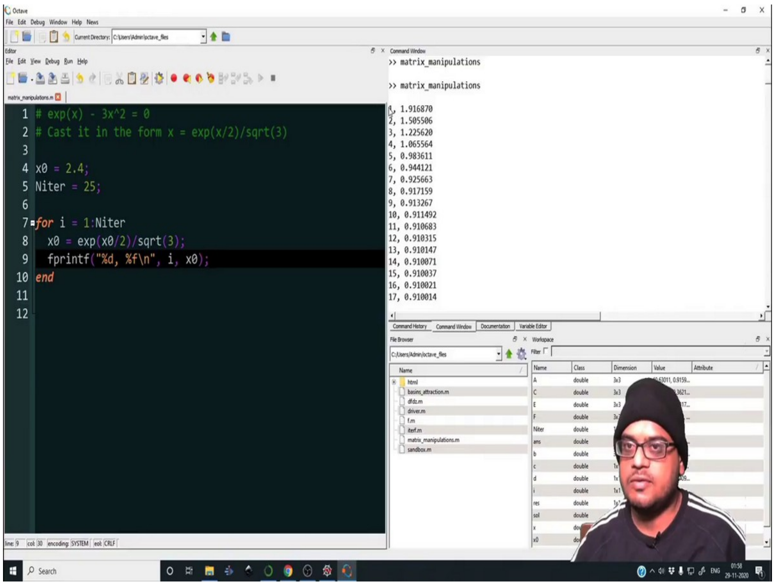
```
c = linspace(-2, 2);
d = exp(c);
plot(c, d);
```

```
ans =

   0.361528   0.425550   0.424289
   0.345540   0.051262   0.050480
   0.241532   0.281122   0.180795

>> ans
ans =

   0.361528   0.425550   0.424289
   0.345540   0.051262   0.050480
   0.241532   0.281122   0.180795

>> matrix_manipulations

error: matrix_manipulations: operator *: nonconformant arguments (op1 is 3x1, op2 is 3x3)
error: called from
    matrix_manipulations at line 7 column 2
>> matrix_manipulations

>> |
```

So, let me say A =linspace, let me call c=linspace(-2, 2), then d =exp (c). So, all the elements of c are exponentiated and it is assigned to d ok, we can plot(c,d). Let me run this ok. There you go this is the plot of exponential of x ok.

So, now let me wrap up this particular lecture with a quick demonstration of fixed point iterations. I mean all the codes are available on the website as such you can have a look over there. We have discussed in detail about what we are trying to achieve in Python, but remember you can apply the same logic in Octave as well.

(Refer Slide Time: 28:05)



```
# exp(x) - 3x^2 = 0
# Cast it in the form x = exp(x/2)/sqrt(3)

x0 = 2.4;
Niter = 25;

for i = 1:Niter
  x0 = exp(x0/2)/sqrt(3);
  fprintf("%d, %f\n", i, x0);
end
```

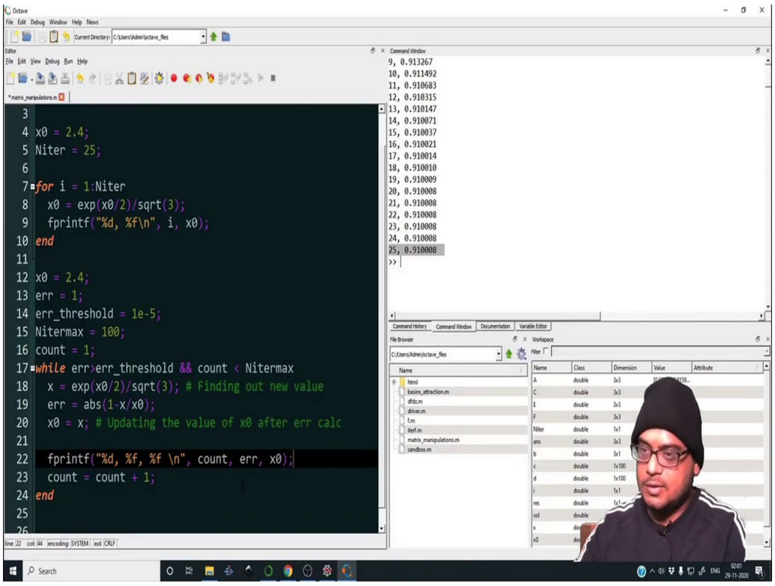
```
>> matrix_manipulations

>> matrix_manipulations

1, 1.916870
2, 1.505506
3, 1.225620
4, 1.065564
5, 0.983611
6, 0.944121
7, 0.925663
8, 0.917159
9, 0.913267
10, 0.911492
11, 0.910683
12, 0.910315
13, 0.910147
14, 0.910071
15, 0.910037
16, 0.910021
17, 0.910014
```

So, what do we have? We have suppose we want to we want to solve exp (x)- 3*x^2 =0; this is the equation that we want to solve. So, let me Cast it in the form x=exp (x/2) /square root of 3. So, you want to cast it in this form.

So, let us say x_guess or x0=2.5 or 2.4 for i=1:Niter. So, Niter would be equal to say 25, let us end the for loop. Then over here we will do x=exp (x0/2) / square root of 3.

In fact, we can just do x0 over here, it makes no difference. Then we can print a fprintf x0. Let us in fact, print the iteration number and the value of x0 as it is updated from the initial value of 2.4 ok. So, let us see what this output gives us ok.

(Refer Slide Time: 30:01)



```

x0 = 2.4;
Niter = 25;

for i = 1:Niter
  x0 = exp(x0/2)/sqrt(3);
  fprintf("%d, %f\n", i, x0);
end

x0 = 2.4;
err = 1;
err_threshold = 1e-5;
Nitermax = 100;
count = 1;
while err>err_threshold && count < Nitermax
  x = exp(x0/2)/sqrt(3); # Finding out new value
  err = abs(1-x/x0);
  x0 = x; # Updating the value of x0 after err calc

  fprintf("%d, %f, %f \n", count, err, x0);
  count = count + 1;
end

```

```
9, 0.913267
10, 0.911492
11, 0.910683
12, 0.910315
13, 0.910147
14, 0.910071
15, 0.910037
16, 0.910021
17, 0.910014
18, 0.910010
19, 0.910009
20, 0.910008
21, 0.910008
22, 0.910008
23, 0.910008
24, 0.910008
25, 0.910008
>>
```

Octave
File Edit Debug Window Help News
Current Directory: C:\Users\Admin\octave_files
Editor
File Edit View Debug Run Help
*matrix_manipulations.m
line: 22 col: 44 encoding: SYSTEM eol: CRLF
Command Window
Command History
Command Window
Documentation
Variable Editor
File Browser
C:/Users/Admin/octave_files
Workspace
Filter
Name
Class
Dimension
Value
Attribute
Search

So, the first after the first iteration we have this and slowly we converse towards this root ok. So, this is how you would achieve fixed point iterations, if we were to implement this as a while loop. So, we will do error =1, err_threshold=1e-5 and we will say while error>err_threshold and count<Nitermax.

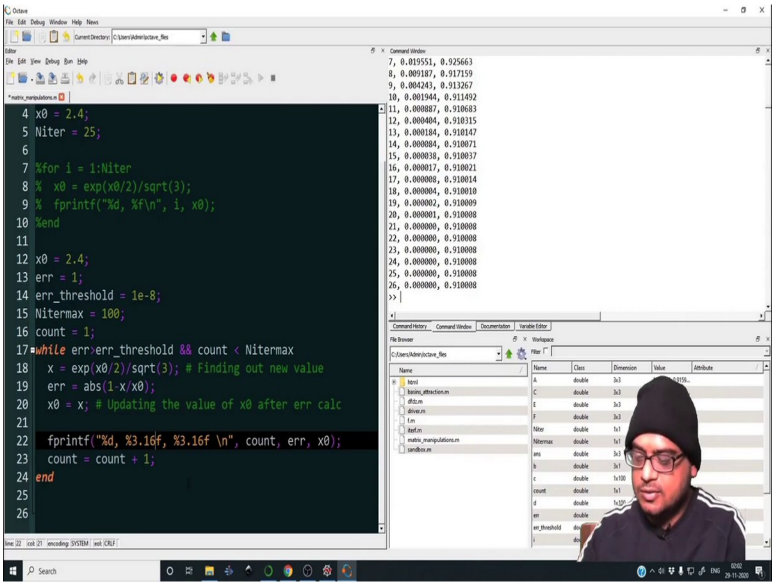
So, like we have discussed in the Python lecture that while loop can run infinitely long. So, we want to terminate the while loop if we have reached a certain number of thresholds, if you are never going to reach the threshold; the loop will keep on going infinitely long. So, you say that I have done the calculation so many times, that is Nitermax, after that I do not want to calculate this anymore. I will say iterations are failed, end of story.

So, we will say Nitermax = 100, count = 1 ok, we will end the loop; then we will put in whatever we want to do. So, we will do x = exp (x0/2) /square root of 3. We will define x0 again because, we are doing a different method over here.

So, this updates the value of x, let us find out error; relative error will be absolute 1 – x0/x or it can be x /x0, it depends on how you define the relative error. So, this is the value of error, then we will update the value of x0. Then, so this is Finding out the new value, this is updating the value of x0 after calculating the error ok.

We will increment count alright. So, let us print out as well print again it is just switching between different programming languages, sometimes its gets you get used to some language ok. So, % d, we will also require %f for the error and %f for the root. So, we will print count, actually this has to be afterwards ok. So, we will print count, then we will be we will print the error, then we will print x0 ok.

(Refer Slide Time: 33:10)



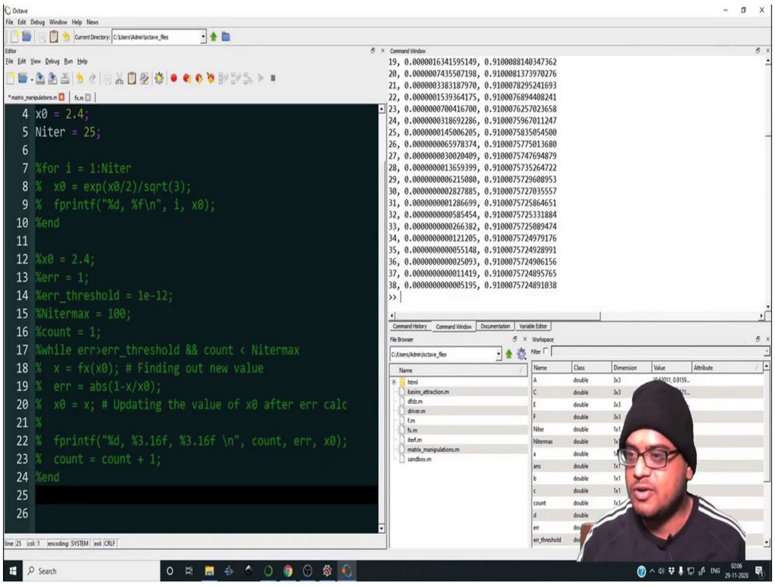
```
x0 = 2.4;
Niter = 25;

%for i = 1:Niter
%  x0 = exp(x0/2)/sqrt(3);
%  fprintf("%d, %f\n", i, x0);
%end

x0 = 2.4;
err = 1;
err_threshold = 1e-8;
Nitermax = 100;
count = 1;
while err>err_threshold && count < Nitermax
  x = exp(x0/2)/sqrt(3); # Finding out new value
  err = abs(1-x/x0);
  x0 = x; # Updating the value of x0 after err calc

  fprintf("%d, %3.16f, %3.16f \n", count, err, x0);
  count = count + 1;
end
```

```
7, 0.019551, 0.925663
8, 0.009187, 0.917159
9, 0.004243, 0.913267
10, 0.001944, 0.911492
11, 0.000887, 0.910683
12, 0.000404, 0.910315
13, 0.000184, 0.910147
14, 0.000084, 0.910071
15, 0.000038, 0.910037
16, 0.000017, 0.910021
17, 0.000008, 0.910014
18, 0.000004, 0.910010
19, 0.000002, 0.910009
20, 0.000001, 0.910008
21, 0.000000, 0.910008
22, 0.000000, 0.910008
23, 0.000000, 0.910008
24, 0.000000, 0.910008
25, 0.000000, 0.910008
26, 0.000000, 0.910008
>>
```

So, let us run this ok. So, let me comment out this part, we do not need this. So, I just select a bunch of lines and press the shortcut ctrl+r, that comments everything ok. So, let me run only this part. So, these are the iteration numbers. So, after 17 iterations, we have reached the threshold of 10^(-5). The relative error is less than 10^(-5) after 17 iterations

In fact, if we make this 10^(-8), it took 26 iterations ok. Let me improve the number of digits in printing. So, 3.16f means 3 digits on the left of the decimal, 16 digits on the right of the decimal.

(Refer Slide Time: 34:03)



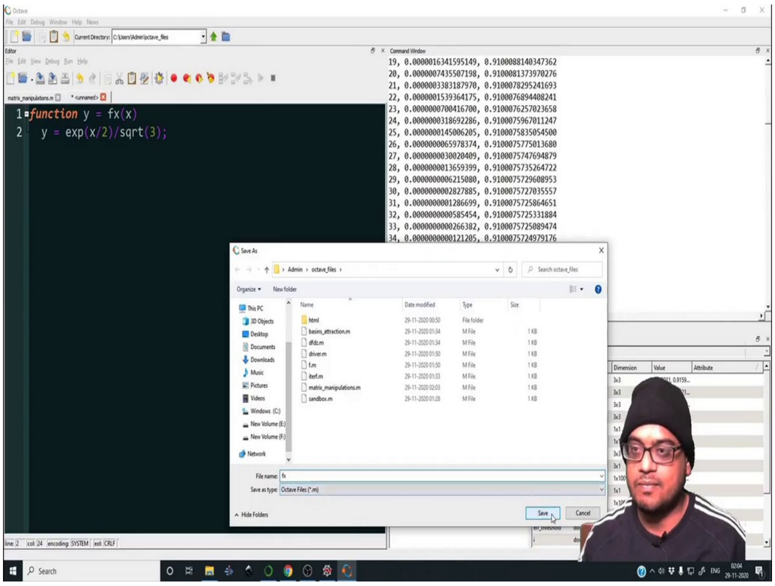
```
x0 = 2.4;
Niter = 25;

%for i = 1:Niter
%  x0 = exp(x0/2)/sqrt(3);
%  fprintf("%d, %f\n", i, x0);
%end

%x0 = 2.4;
%err = 1;
%err_threshold = 1e-12;
%Nitermax = 100;
%count = 1;
%while err>err_threshold && count < Nitermax
%  x = fx(x0); # Finding out new value
%  err = abs(1-x/x0);
%  x0 = x; # Updating the value of x0 after err calc
%
%  fprintf("%d, %3.16f, %3.16f \n", count, err, x0);
%  count = count + 1;
%end
```

```
19, 0.0000016341595149, 0.9100088140347362
20, 0.0000007435507198, 0.9100081373970276
21, 0.0000003383187970, 0.9100078295241693
22, 0.0000001539364175, 0.9100076894408241
23, 0.0000000700416700, 0.9100076257023658
24, 0.0000000318692286, 0.9100075967011247
25, 0.0000000145006205, 0.9100075835054500
26, 0.0000000065978374, 0.9100075775013680
27, 0.0000000030020409, 0.9100075747694879
28, 0.0000000013659399, 0.9100075735264722
29, 0.0000000006215080, 0.9100075729608953
30, 0.0000000002827885, 0.9100075727035557
31, 0.0000000001286699, 0.9100075725864651
32, 0.0000000000585454, 0.9100075725331884
33, 0.0000000000266382, 0.9100075725089474
34, 0.0000000000121205, 0.9100075724979176
35, 0.0000000000055148, 0.9100075724928991
36, 0.0000000000025093, 0.9100075724906156
37, 0.0000000000011419, 0.9100075724895765
38, 0.0000000000005195, 0.9100075724891038
>> 
```

So, let me execute this ok. So, this gives a better picture, you see that the error is really small. Let us see what happens when we push it further, it took 38 iterations to reach a relative error 10^(-12) ok.

So, we could have defined this as a function as well ok. So, we had done this in Python already. So, ideally let me cut this, you just want f fx(x0) ok, but what is fx(f0)?

(Refer Slide Time: 34:44)



```
function y = fx(x)
  y = exp(x/2)/sqrt(3);
```

```
19, 0.0000016341595149, 0.9100088140347362
20, 0.0000007435507198, 0.9100081373970276
21, 0.0000003383187970, 0.9100078295241693
22, 0.0000001539364175, 0.9100076894408241
23, 0.0000000700416700, 0.9100076257023658
24, 0.0000000318692286, 0.9100075967011247
25, 0.0000000145006205, 0.9100075835054500
26, 0.0000000065978374, 0.9100075775013680
27, 0.0000000030020409, 0.9100075747694879
28, 0.0000000013659399, 0.9100075735264722
29, 0.0000000006215080, 0.9100075729608953
30, 0.0000000002827885, 0.9100075727035557
31, 0.0000000001286699, 0.9100075725864651
32, 0.0000000000585454, 0.9100075725331884
33, 0.0000000000266382, 0.9100075725089474
34, 0.0000000000121205, 0.9100075724979176
```

Save As

Name	Date modified	Type	Size
html	29-11-2020 00:50	File folder	
basins_attraction.m	29-11-2020 01:34	M File	1 KB
dfdz.m	29-11-2020 01:34	M File	1 KB
driver.m	29-11-2020 01:50	M File	1 KB
f.m	29-11-2020 01:50	M File	1 KB
iterf.m	29-11-2020 01:33	M File	1 KB
matrix_manipulations.m	29-11-2020 02:03	M File	1 KB
sandbox.m	29-11-2020 01:28	M File	1 KB

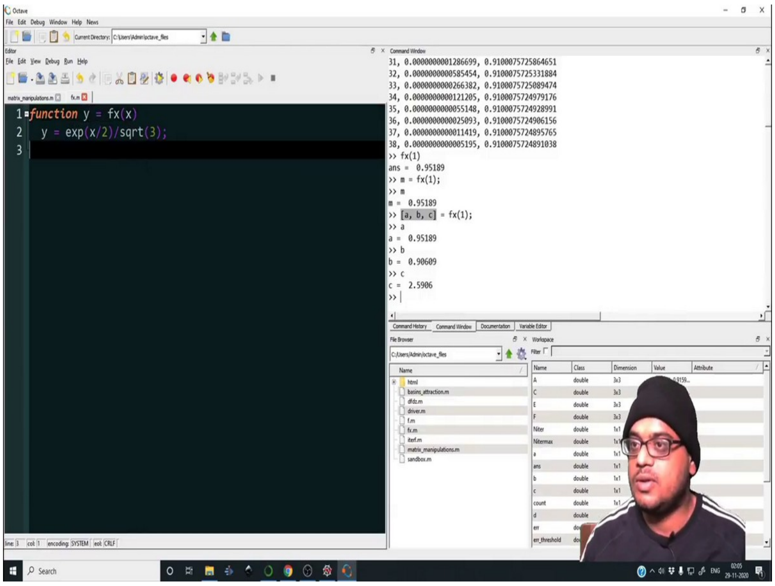
File name: fx

Save as type: Octave Files (*.m)

So, for this we have to create a new file. In this new file we will create the function, function return value will be y=fx and the input will be x. So, fx is the name of the function, input is x, the output is y and we have to return this. So, y is this. We do not have to explicitly mention return y because, the return argument is already mentioned in the function itself.

So, function if fx and y is the return variable; remember that x or y and whatever we are defining in this particular file is strictly local to this file. It does not have a global sense, it does not have a global scope. So, these are created locally and terminated once y has been returned to the main file. So, the function file has to be named exactly equal to this function name. So, we have to create a file fx.

(Refer Slide Time: 35:48)



```
function y = fx(x)
  y = exp(x/2)/sqrt(3);
```

```
31, 0.0000000001286699, 0.9100075725864651
32, 0.0000000000585454, 0.9100075725331884
33, 0.0000000000266382, 0.9100075725089474
34, 0.0000000000121205, 0.9100075724979176
35, 0.0000000000055148, 0.9100075724928991
36, 0.0000000000025093, 0.9100075724906156
37, 0.0000000000011419, 0.9100075724895765
38, 0.0000000000005195, 0.9100075724891038
>> fx(1)
ans =  0.95189
>> m = fx(1);
>> m
m =  0.95189
>> [a, b, c] = fx(1);
>> a
a =  0.95189
>> b
b =  0.90609
>> c
c =  2.5906
>>
```

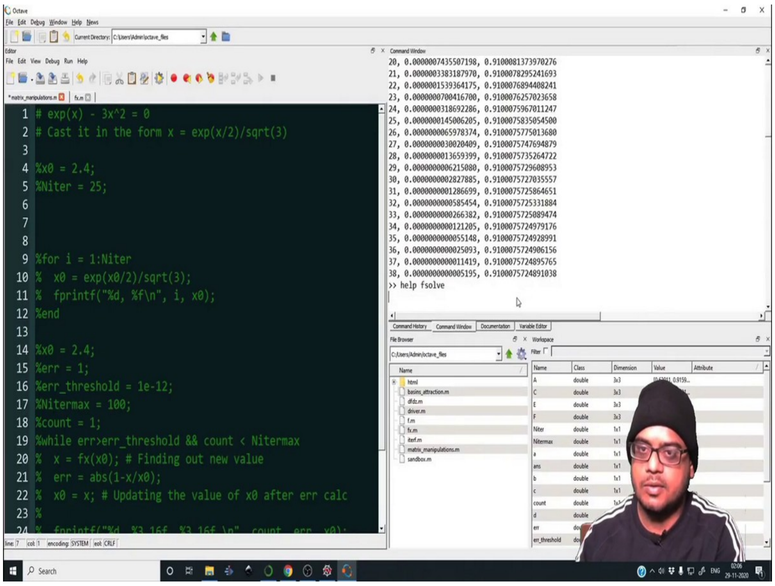
So, let me do this, let fx(1). So, it passes 1 to this function, it evaluates the function and returns me value of y. If I had multiple outputs, if I had multiple outputs; I would simply do this. So, this is how multiple outputs can be obtained.

So, if I let me save this file. So, if I make m = fx(1). So, m will actually; so, I need to yeah. So, I have to supply this a, b, c. So, now a is this, b is this, c is this. So, these are the 3 return values that fx has return to me ok. I must appropriately in the main wherever I am going to use this function, I must have these 3 variables waiting for accepting the output of fx.

So, let me remove this, we do not need this from now; we just need a single scalar output of this function. So, I am going to remove this, I am going to go to this. So, I am going to run this again, we get the same output. So, this is how you can abstract your tasks to a different file, you can make a function, you can create a different file. In Python you could make the whole thing in a single script that is more convenient.

But, sometimes it is more efficient to keep files separately and you would see this in bunch of programs you keep separate files separately and then you compile everything in one go. So, we have done this apart from doing fixed point iterations, there are a bunch of inbuilt functions as well. The most common function to solve is called as fsolve.

(Refer Slide Time: 38:10)

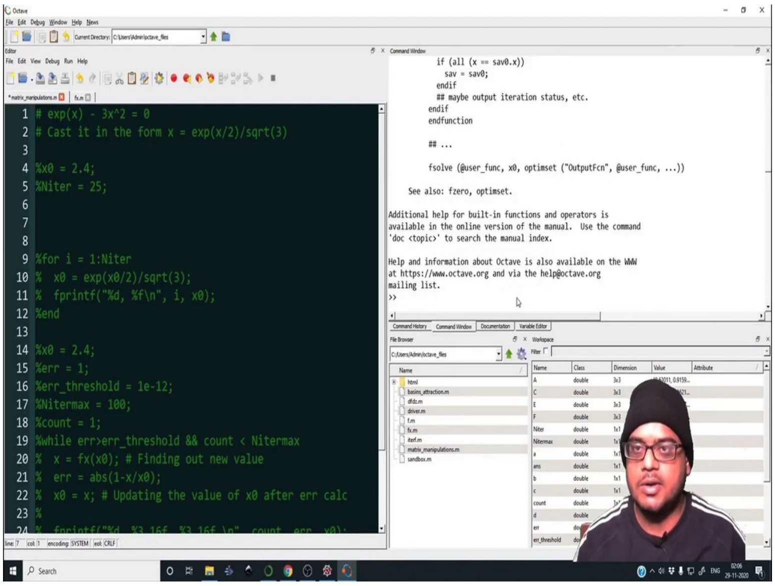


```
# exp(x) - 3x^2 = 0
# Cast it in the form x = exp(x/2)/sqrt(3)

%x0 = 2.4;
%Niter = 25;



%for i = 1:Niter
%  x0 = exp(x0/2)/sqrt(3);
%  fprintf("%d, %f\n", i, x0);
%end

%x0 = 2.4;
%err = 1;
%err_threshold = 1e-12;
%Nitermax = 100;
%count = 1;
%while err>err_threshold && count < Nitermax
%  x = fx(x0); # Finding out new value
%  err = abs(1-x/x0);
%  x0 = x; # Updating the value of x0 after err calc
%
%  fprintf("%d, %3.16f, %3.16f \n", count, err, x0);
```

```
20, 0.0000007435507198, 0.9100081373970276
21, 0.0000003383187970, 0.9100078295241693
22, 0.0000001539364175, 0.9100076894408241
23, 0.0000000700416700, 0.9100076257023658
24, 0.0000000318692286, 0.9100075967011247
25, 0.0000000145006205, 0.9100075835054500
26, 0.0000000065978374, 0.9100075775013680
27, 0.0000000030020409, 0.9100075747694879
28, 0.0000000013659399, 0.9100075735264722
29, 0.0000000006215080, 0.9100075729608953
30, 0.0000000002827885, 0.9100075727035557
31, 0.0000000001286699, 0.9100075725864651
32, 0.0000000000585454, 0.9100075725331884
33, 0.0000000000266382, 0.9100075725089474
34, 0.0000000000121205, 0.9100075724979176
35, 0.0000000000055148, 0.9100075724928991
36, 0.0000000000025093, 0.9100075724906156
37, 0.0000000000011419, 0.9100075724895765
38, 0.0000000000005195, 0.9100075724891038
>> help fsolve
```

So, if you are ever confused. So, we will just go to help fsolve.

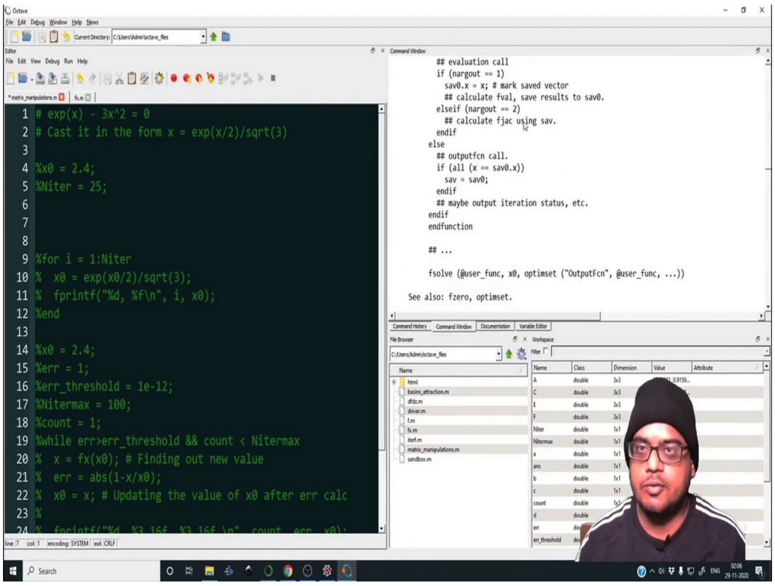
(Refer Slide Time: 38:16)



```
# exp(x) - 3x^2 = 0
# Cast it in the form x = exp(x/2)/sqrt(3)

%x0 = 2.4;
%Niter = 25;



%for i = 1:Niter
%  x0 = exp(x0/2)/sqrt(3);
%  fprintf("%d, %f\n", i, x0);
%end

%x0 = 2.4;
%err = 1;
%err_threshold = 1e-12;
%Nitermax = 100;
%count = 1;
%while err>err_threshold && count < Nitermax
%  x = fx(x0); # Finding out new value
%  err = abs(1-x/x0);
%  x0 = x; # Updating the value of x0 after err calc
%
%  fprintf("%d, %3.16f, %3.16f \n", count, err, x0);
```

```
            if (all (x == sav0.x))
              sav = sav0;
            endif
            ## maybe output iteration status, etc.
          endif
          endfunction

          ## ...

          fsolve (@user_func, x0, optimset ("OutputFcn", @user_func, ...))

     See also: fzero, optimset.

Additional help for built-in functions and operators is
available in the online version of the manual.  Use the command
'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at https://www.octave.org and via the help@octave.org
mailing list.
>>
```

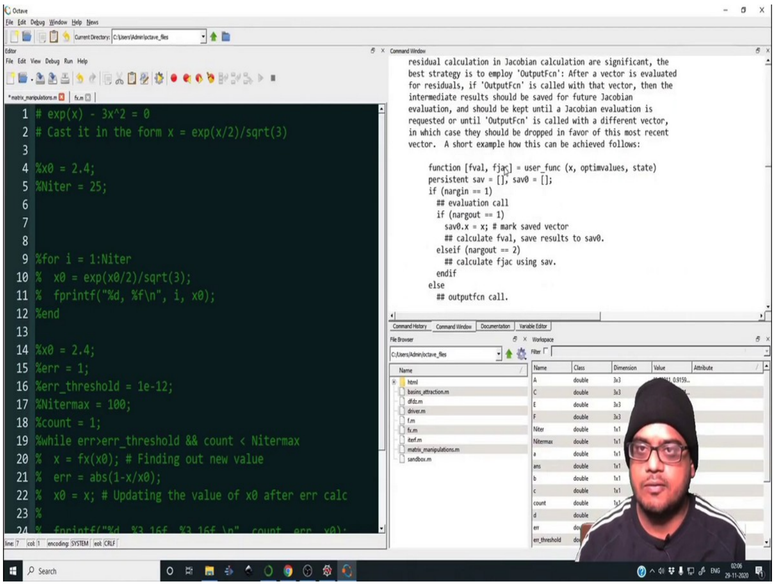
(Refer Slide Time: 38:17)



```
# exp(x) - 3x^2 = 0
# Cast it in the form x = exp(x/2)/sqrt(3)

%x0 = 2.4;
%Niter = 25;




%for i = 1:Niter
%  x0 = exp(x0/2)/sqrt(3);
%  fprintf("%d, %f\n", i, x0);
%end

%x0 = 2.4;
%err = 1;
%err_threshold = 1e-12;
%Nitermax = 100;
%count = 1;
%while err>err_threshold && count < Nitermax
%  x = fx(x0); # Finding out new value
%  err = abs(1-x/x0);
%  x0 = x; # Updating the value of x0 after err calc
%
%  fprintf("%d  %3.16f  %3.16f \n", count, err, x0);
```

```
          ## evaluation call
          if (nargout == 1)
            sav0.x = x; # mark saved vector
            ## calculate fval, save results to sav0.
          elseif (nargout == 2)
            ## calculate fjac using sav.
          endif
        else
          ## outputfcn call.
          if (all (x == sav0.x))
            sav = sav0;
          endif
          ## maybe output iteration status, etc.
        endif
        endfunction

        ## ...

        fsolve (@user_func, x0, optimset ("OutputFcn", @user_func, ...))

    See also: fzero, optimset.
```

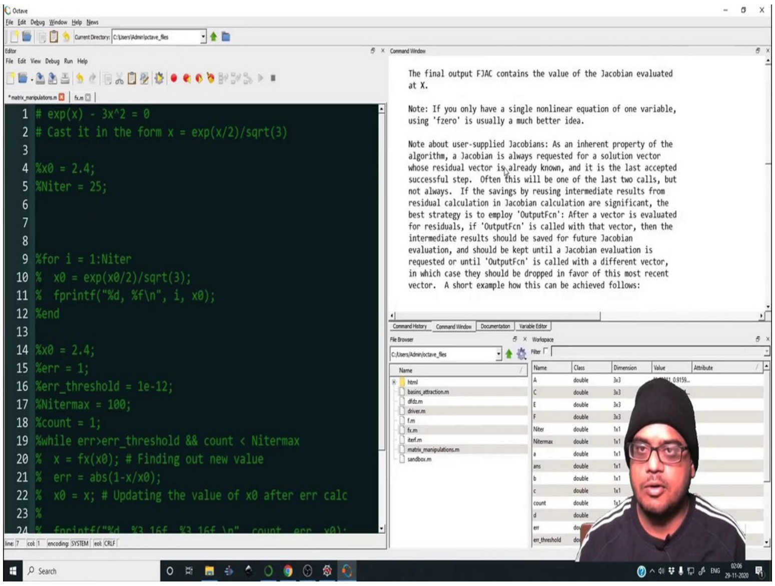
(Refer Slide Time: 38:18)



```
# exp(x) - 3x^2 = 0
# Cast it in the form x = exp(x/2)/sqrt(3)

%x0 = 2.4;
%Niter = 25;




%for i = 1:Niter
%  x0 = exp(x0/2)/sqrt(3);
%  fprintf("%d, %f\n", i, x0);
%end

%x0 = 2.4;
%err = 1;
%err_threshold = 1e-12;
%Nitermax = 100;
%count = 1;
%while err>err_threshold && count < Nitermax
%  x = fx(x0); # Finding out new value
%  err = abs(1-x/x0);
%  x0 = x; # Updating the value of x0 after err calc
%
%  fprintf("%d  %3.16f  %3.16f \n", count, err, x0);
```

```
    residual calculation in Jacobian calculation are significant, the
    best strategy is to employ 'OutputFcn': After a vector is evaluated
    for residuals, if 'OutputFcn' is called with that vector, then the
    intermediate results should be saved for future Jacobian
    evaluation, and should be kept until a Jacobian evaluation is
    requested or until 'OutputFcn' is called with a different vector,
    in which case they should be dropped in favor of this most recent
    vector.  A short example how this can be achieved follows:

        function [fval, fjac] = user_func (x, optimvalues, state)
        persistent sav = [], sav0 = [];
        if (nargin == 1)
          ## evaluation call
          if (nargout == 1)
            sav0.x = x; # mark saved vector
            ## calculate fval, save results to sav0.
          elseif (nargout == 2)
            ## calculate fjac using sav.
          endif
        else
          ## outputfcn call.
```

(Refer Slide Time: 38:18)

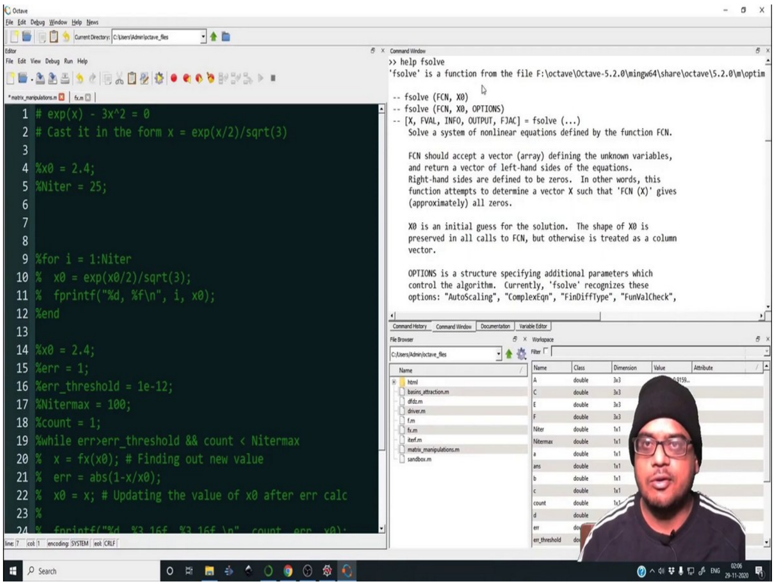


```
# exp(x) - 3x^2 = 0
# Cast it in the form x = exp(x/2)/sqrt(3)

%x0 = 2.4;
%Niter = 25;



%for i = 1:Niter
%  x0 = exp(x0/2)/sqrt(3);
%  fprintf("%d, %f\n", i, x0);
%end

%x0 = 2.4;
%err = 1;
%err_threshold = 1e-12;
%Nitermax = 100;
%count = 1;
%while err>err_threshold && count < Nitermax
%  x = fx(x0); # Finding out new value
%  err = abs(1-x/x0);
%  x0 = x; # Updating the value of x0 after err calc
%
%  fprintf("%d  %3.16f  %3.16f \n", count, err, x0);
```

```
    The final output FJAC contains the value of the Jacobian evaluated
    at X.

    Note: If you only have a single nonlinear equation of one variable,
    using 'fzero' is usually a much better idea.

    Note about user-supplied Jacobians: As an inherent property of the
    algorithm, a Jacobian is always requested for a solution vector
    whose residual vector is already known, and it is the last accepted
    successful step.  Often this will be one of the last two calls, but
    not always.  If the savings by reusing intermediate results from
    residual calculation in Jacobian calculation are significant, the
    best strategy is to employ 'OutputFcn': After a vector is evaluated
    for residuals, if 'OutputFcn' is called with that vector, then the
    intermediate results should be saved for future Jacobian
    evaluation, and should be kept until a Jacobian evaluation is
    requested or until 'OutputFcn' is called with a different vector,
    in which case they should be dropped in favor of this most recent
    vector.  A short example how this can be achieved follows:
```

It will give you what all you have to do ok.

(Refer Slide Time: 38:24)



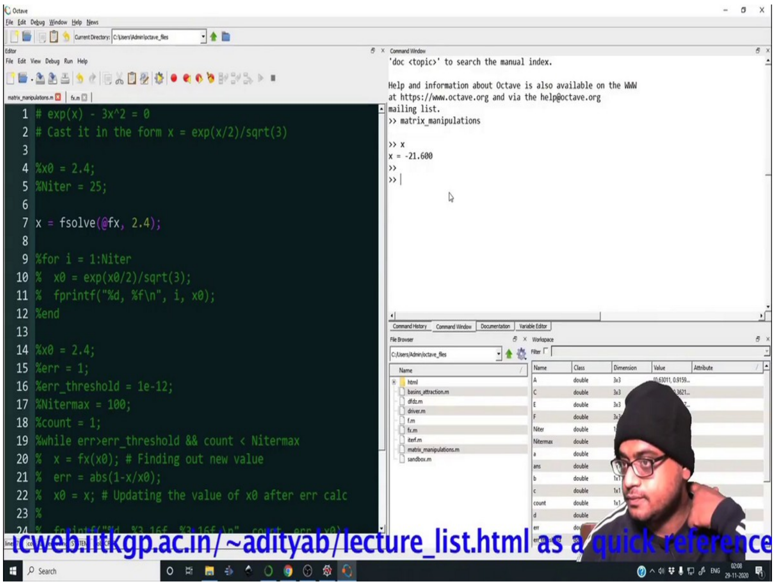
```
# exp(x) - 3x^2 = 0
# Cast it in the form x = exp(x/2)/sqrt(3)

%x0 = 2.4;
%Niter = 25;



%for i = 1:Niter
%  x0 = exp(x0/2)/sqrt(3);
%  fprintf("%d, %f\n", i, x0);
%end

%x0 = 2.4;
%err = 1;
%err_threshold = 1e-12;
%Nitermax = 100;
%count = 1;
%while err>err_threshold && count < Nitermax
%  x = fx(x0); # Finding out new value
%  err = abs(1-x/x0);
%  x0 = x; # Updating the value of x0 after err calc
%
%  fprintf("%d  %3.16f  %3.16f \n", count, err, x0);
```

```
>> help fsolve
'fsolve' is a function from the file F:\octave\Octave-5.2.0\mingw64\share\octave\5.2.0\m\optim

 -- fsolve (FCN, X0)
 -- fsolve (FCN, X0, OPTIONS)
 -- [X, FVAL, INFO, OUTPUT, FJAC] = fsolve (...)
     Solve a system of nonlinear equations defined by the function FCN.

     FCN should accept a vector (array) defining the unknown variables,
     and return a vector of left-hand sides of the equations.
     Right-hand sides are defined to be zeros.  In other words, this
     function attempts to determine a vector X such that 'FCN (X)' gives
     (approximately) all zeros.

     X0 is an initial guess for the solution.  The shape of X0 is
     preserved in all calls to FCN, but otherwise is treated as a column
     vector.

     OPTIONS is a structure specifying additional parameters which
     control the algorithm.  Currently, 'fsolve' recognizes these
     options: "AutoScaling", "ComplexEqn", "FinDiffType", "FunValCheck",
```

So, fsolve is a non-linear solver, it takes as an input the function and the initial guess and the output can be X, FVAL, INFO, OUTPUT. So, let us just take one output. So, if there are multiple outputs coming in and if you only define one variable call.

So, if I do X= fsolve, it will X will only store this first value, if I give X ,y =fsolve then y will contain FVAL, where FVAL its defined; the documentation is a bit wonky anyway. So, we all we care about is X at this moment.

(Refer Slide Time: 39:24)



```
# exp(x) - 3x^2 = 0
# Cast it in the form x = exp(x/2)/sqrt(3)

%x0 = 2.4;
%Niter = 25;

x = fsolve(@fx, 2.4);

%for i = 1:Niter
%  x0 = exp(x0/2)/sqrt(3);
%  fprintf("%d, %f\n", i, x0);
%end

%x0 = 2.4;
%err = 1;
%err_threshold = 1e-12;
%Nitermax = 100;
%count = 1;
%while err>err_threshold && count < Nitermax
%  x = fx(x0); # Finding out new value
%  err = abs(1-x/x0);
%  x0 = x; # Updating the value of x0 after err calc
%
%  fprintf("%d, %3.16f, %3.16f \n", count, err, x0)
```

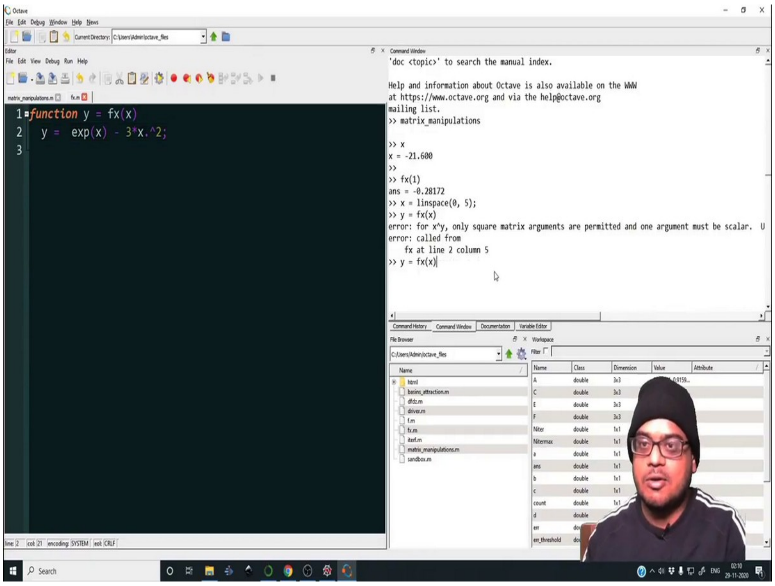
```
'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at https://www.octave.org and via the help@octave.org
mailing list.
>> matrix_manipulations

>> x
x = -21.600
>>
>> |
```

ıcweb.iitkgp.ac.in/~adityab/lecture_list.html as a quick reference

So, x equal to fsolve, now in Python we had to give the function handle, the same thing we have to do over here. So, passing a function handle is done with the help of the at the rate(@) sign. So now, if we do this and give an initial guess; so 2.4, let us see what happens.

Let me execute this, let me press F5 ok. So, x is so, this is not the function that we want, we actually have to give the actual function that we are trying to solve for; that is this function, this function ok.

(Refer Slide Time: 40:19)



```
function y = fx(x)
  y =  exp(x) - 3*x.^2;
```

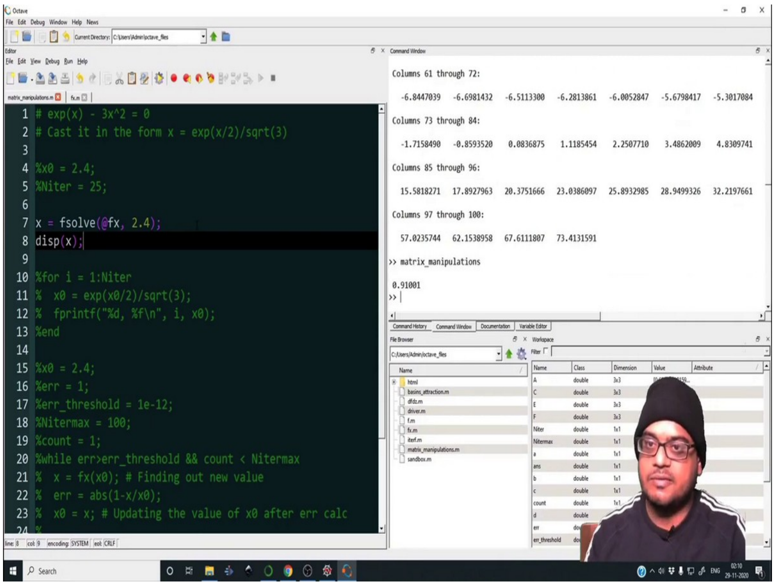
```
'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at https://www.octave.org and via the help@octave.org
mailing list.
>> matrix_manipulations

>> x
x = -21.600
>>
>> fx(1)
ans = -0.28172
>> x = linspace(0, 5);
>> y = fx(x)
error: for x^y, only square matrix arguments are permitted and one argument must be scalar.  U
error: called from
    fx at line 2 column 5
>> y = fx(x)
```

So, let me replace this alright. So, this has to be 3 *x.^2. So, I am giving x.^2 with an impending hope, that in case I pass an array of x to this function; it should be able to do the proper array raising to the power 2.

So, what I mean is had I not written this, if I just do fx of 1, it will give me the correct answer, no problem. But, if I define x=linspace(0,5) and y=fx(x), it would throw an error. Because, I am passing an array into the function and that array is to be squared, each element of array has to be squared.

But over here I am I have not written it in that way, this raised 2 means it is a matrix squaring process and you cannot square a vector; you can square a matrix, but you cannot. So, you can square a square matrix, you cannot square any other matrix. So, this has to be an element wise raise to 2. Now, once we save this, once I call this again; now it can give me the function ok.

(Refer Slide Time: 41:39)

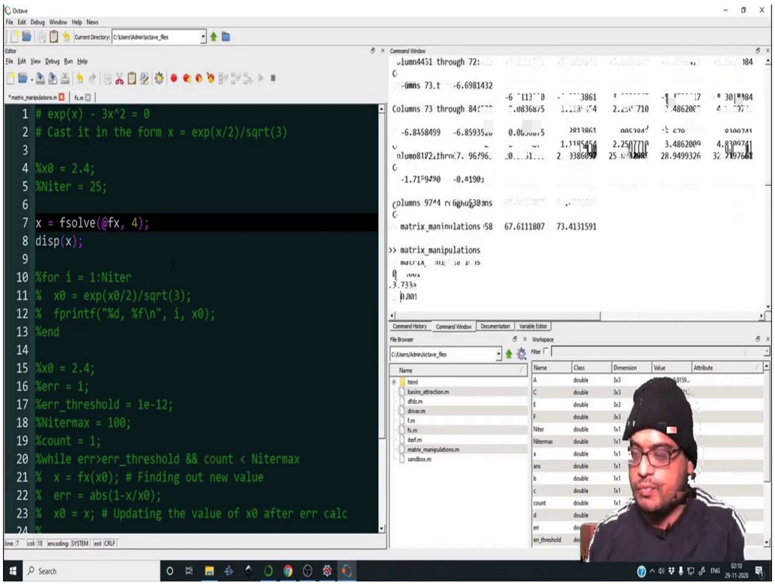


```
# exp(x) - 3x^2 = 0
# Cast it in the form x = exp(x/2)/sqrt(3)

%x0 = 2.4;
%Niter = 25;

x = fsolve(@fx, 2.4);
disp(x);

%for i = 1:Niter
%  x0 = exp(x0/2)/sqrt(3);
%  fprintf("%d, %f\n", i, x0);
%end

%x0 = 2.4;
%err = 1;
%err_threshold = 1e-12;
%Nitermax = 100;
%count = 1;
%while err>err_threshold && count < Nitermax
%  x = fx(x0); # Finding out new value
%  err = abs(1-x/x0);
%  x0 = x; # Updating the value of x0 after err calc
%
```

```
Columns 61 through 72:

  -6.8447039   -6.6981432   -6.5113300   -6.2813861   -6.0052847   -5.6798417   -5.3017084

Columns 73 through 84:

  -1.7158490   -0.8593520    0.0836875    1.1185454    2.2507710    3.4862009    4.8309741

Columns 85 through 96:

  15.5818271   17.8927963   20.3751666   23.0386097   25.8932985   28.9499326   32.2197661

Columns 97 through 100:

  57.0235744   62.1538958   67.6111807   73.4131591

>> matrix_manipulations

0.91001
>> 
```

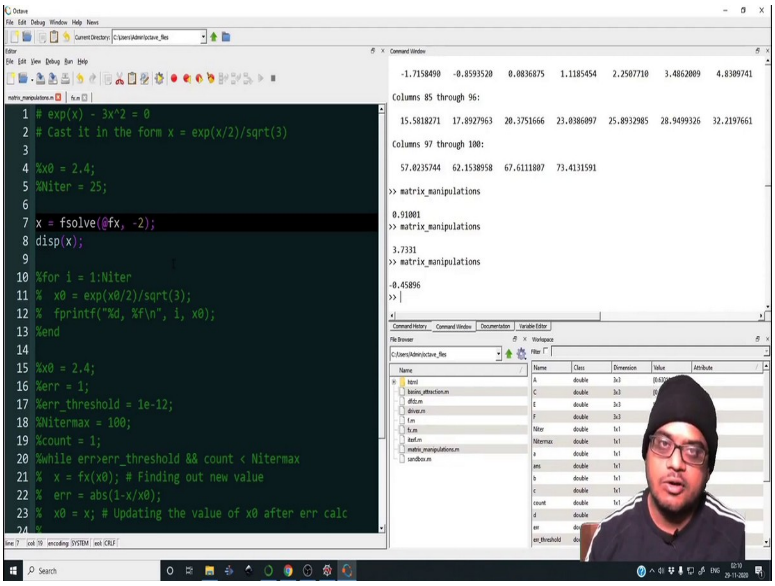
So, that is why with the impending notion that you can pass arrays to this function, you need to appropriately put the element wise operator in this. So now, over here I have passed the function handle, I have given an initial guess. Let me display x as well; once I press F5 it gives me 0.91.

(Refer Slide Time: 42:06)



```
# exp(x) - 3x^2 = 0
# Cast it in the form x = exp(x/2)/sqrt(3)

%x0 = 2.4;
%Niter = 25;

x = fsolve(@fx, 4);
disp(x);

%for i = 1:Niter
%  x0 = exp(x0/2)/sqrt(3);
%  fprintf("%d, %f\n", i, x0);
%end

%x0 = 2.4;
%err = 1;
%err_threshold = 1e-12;
%Nitermax = 100;
%count = 1;
%while err>err_threshold && count < Nitermax
%  x = fx(x0); # Finding out new value
%  err = abs(1-x/x0);
%  x0 = x; # Updating the value of x0 after err calc
```

(Refer Slide Time: 42:11)



```
# exp(x) - 3x^2 = 0
# Cast it in the form x = exp(x/2)/sqrt(3)

%x0 = 2.4;
%Niter = 25;

x = fsolve(@fx, -2);
disp(x);

%for i = 1:Niter
%  x0 = exp(x0/2)/sqrt(3);
%  fprintf("%d, %f\n", i, x0);
%end

%x0 = 2.4;
%err = 1;
%err_threshold = 1e-12;
%Nitermax = 100;
%count = 1;
%while err>err_threshold && count < Nitermax
%  x = fx(x0); # Finding out new value
%  err = abs(1-x/x0);
%  x0 = x; # Updating the value of x0 after err calc
```

```
  -1.7158490   -0.8593520   0.0836875   1.1185454   2.2507710   3.4862009   4.8309741

 Columns 85 through 96:

  15.5818271   17.8927963   20.3751666   23.0386097   25.8932985   28.9499326   32.2197661

 Columns 97 through 100:

  57.0235744   62.1538958   67.6111807   73.4131591

>> matrix_manipulations

 0.91001
>> matrix_manipulations

 3.7331
>> matrix_manipulations

-0.45896
>>
```

Let me give a different guess value, it gives us the other root. Let me give a different guess value, it gives us the third root, excellent. So, this is how you can quickly get things done with the help of GNU Octave.

I hope I have given you a very quick intro to some of the functions; I will link some more videos that I have done as a part of my SWAYAM course. And, it will help you get started in case you are coming at it from a fluid mechanics view point. I had done all this related to fluid mechanics. So, with this I am closing this lecture over here. We are done with week 1; I will see you again next time with stuff on non-linear dynamics.

Its goodbye from me, have a good day. Bye.