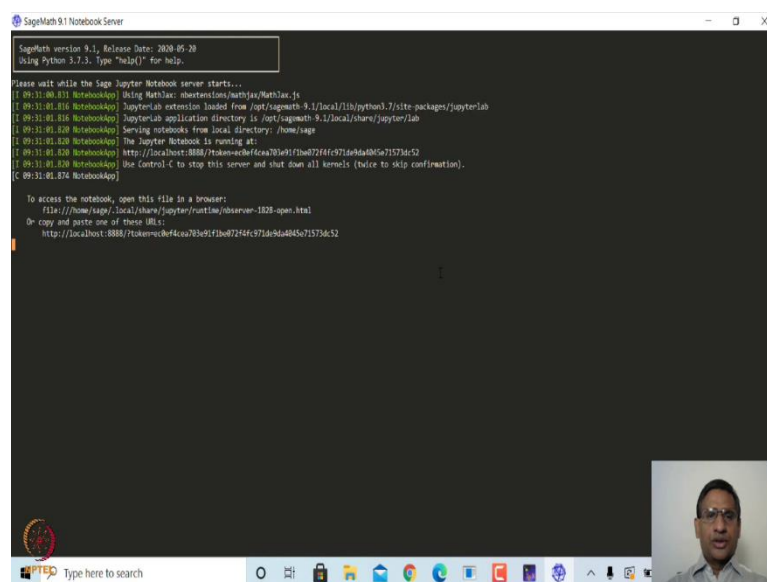


Computational Mathematics with SageMath
Prof. Ajit Kumar
Department of Mathematics
Institute of Chemical Technology, Mumbai

Lecture – 17
Solving Equations in SageMath

Welcome to the 17th lecture on Computational Mathematics with SageMath. In the last lecture we started learning SageMath and basically we explored integers we looked at various methods that can be applied to an integer. Now, let us explore more things in SageMath. So, let me start SageMath Notebook, let me double click this icon.

(Refer Slide Time: 00:46)

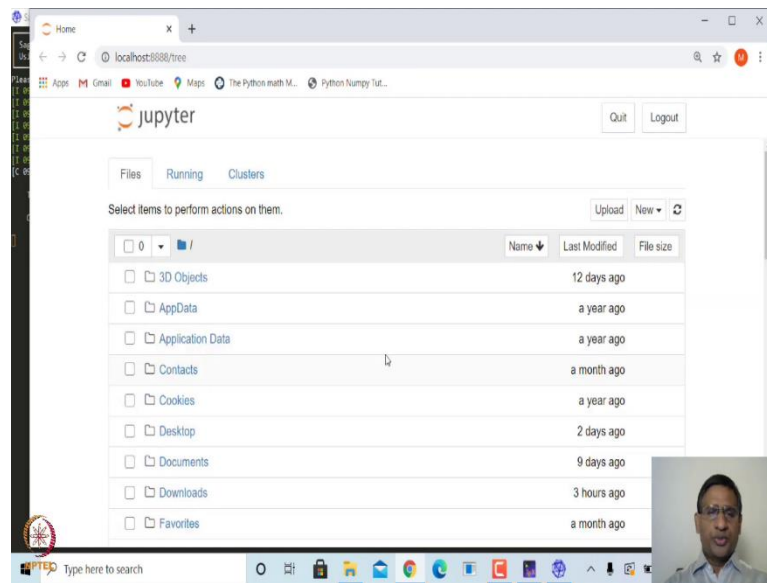


```
SageMath 9.1 Notebook Server
SageMath version 9.1, Release Date: 2020-05-20
Using Python 3.7.3. Type "help()" for help.

Please wait while the Sage Jupyter Notebook server starts...
[09:31:00.811 NotebookApp] Using MathJax: nbextensions/mathjax/MathJax.js
[09:31:01.816 NotebookApp] JupyterLab extension loaded from /opt/sagemath-9.1/local/lib/python3.7/site-packages/jupyterlab
[09:31:01.816 NotebookApp] JupyterLab application directory is /opt/sagemath-9.1/local/share/jupyter/lab
[09:31:01.820 NotebookApp] Serving notebooks from local directory: /home/sage
[09:31:01.820 NotebookApp] The Jupyter Notebook is running at:
[09:31:01.820 NotebookApp] http://localhost:8888/?token=ecbf6c5a703e91f1be07254fc971de9da8b5e71573de52
[09:31:01.820 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 09:31:01.876 NotebookApp]

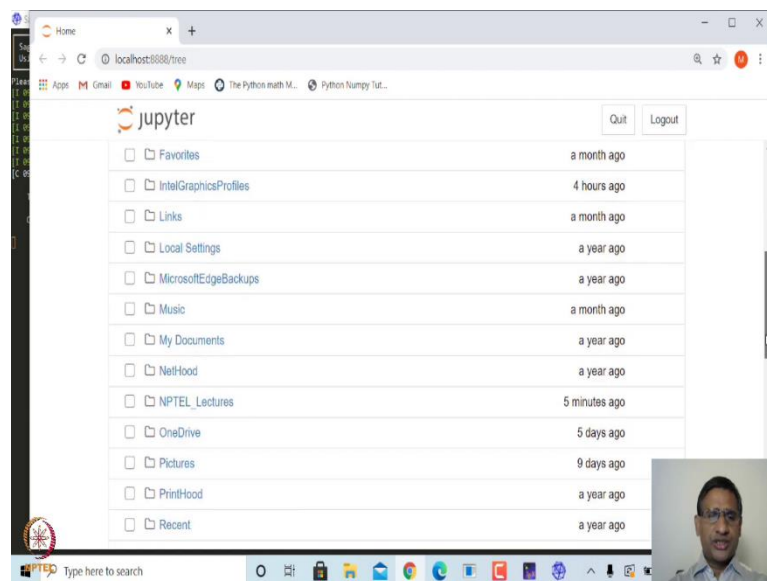
To access the notebook, open this file in a browser:
file:///home/sage/.local/share/jupyter/runtime/notebook-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=ecbf6c5a703e91f1be07254fc971de9da8b5e71573de52
```

(Refer Slide Time: 00:57)

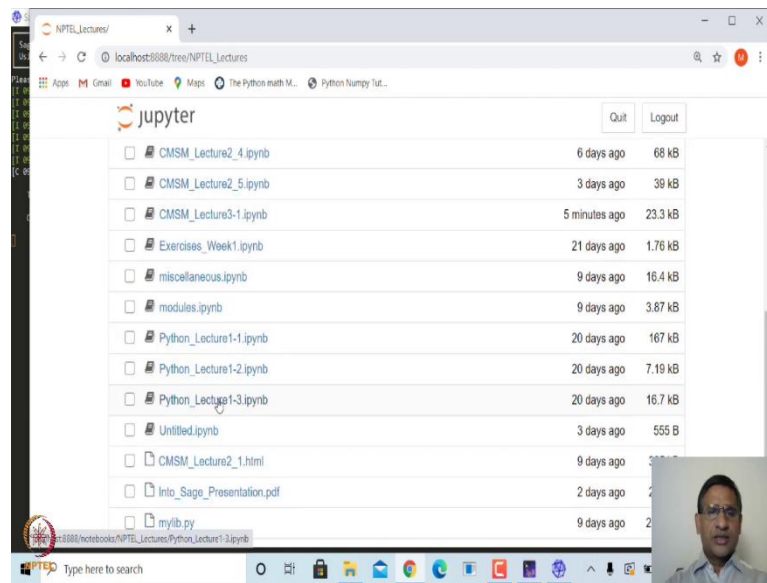


So, it will start SageMath server and it will open a Jupyter Notebook.

(Refer Slide Time: 01:00)

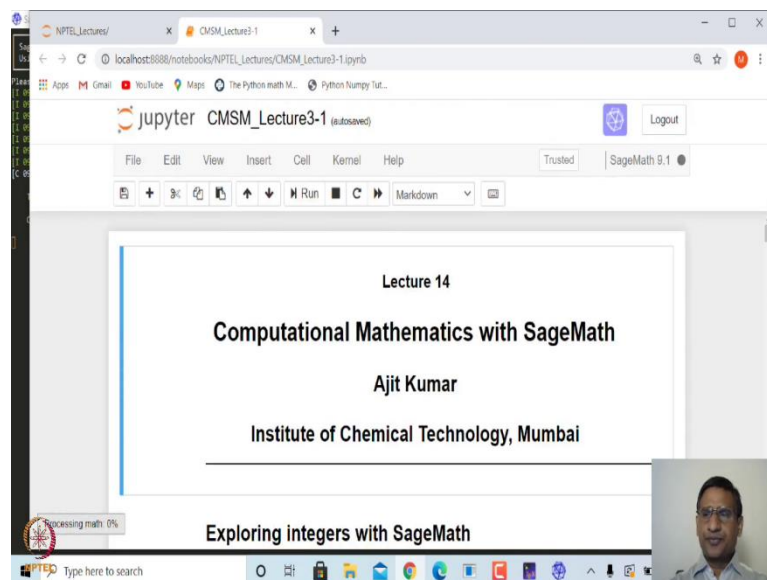


(Refer Slide Time: 01:02)

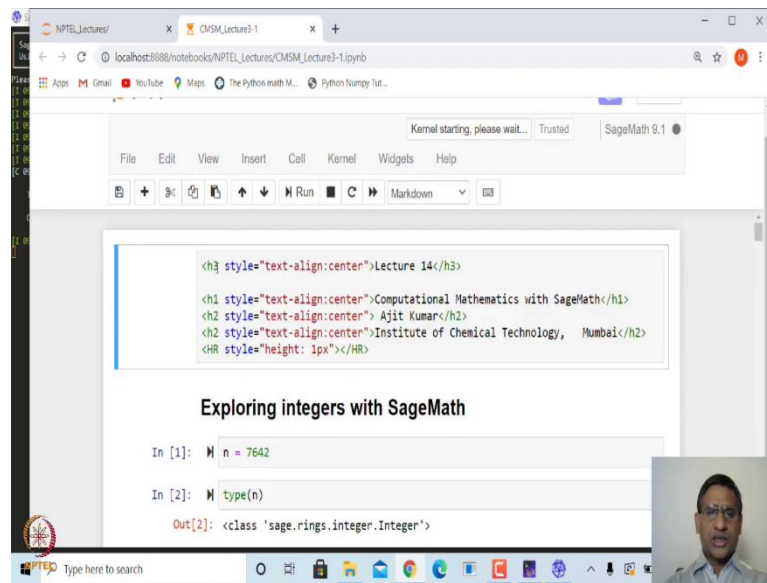


Let me go to the directory in which I have all the files. So, for example, this is the file we created last time computational mathematics with SageMath lecture 3.1.

(Refer Slide Time: 01:16)

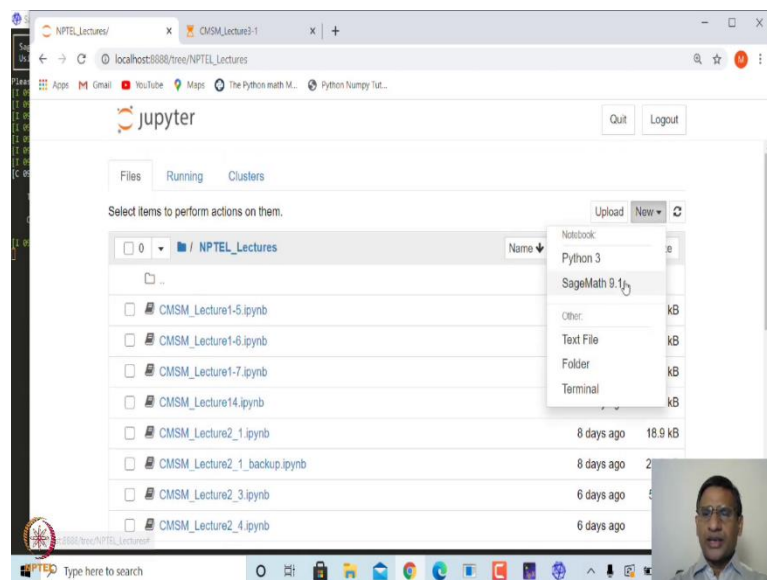


(Refer Slide Time: 01:24)

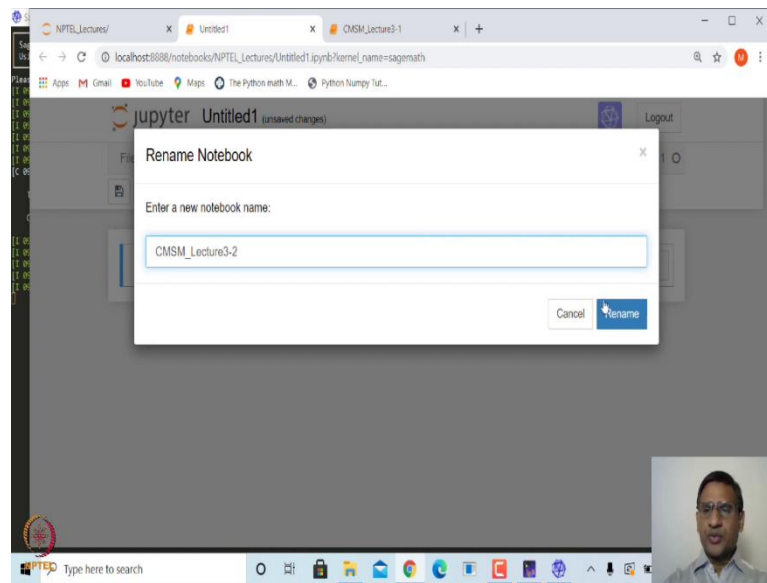


So, if I click on this it will open the file which we created in the last lecture. I just added the heading. So, let me just copy this and let me create another notebook.

(Refer Slide Time: 01:34)

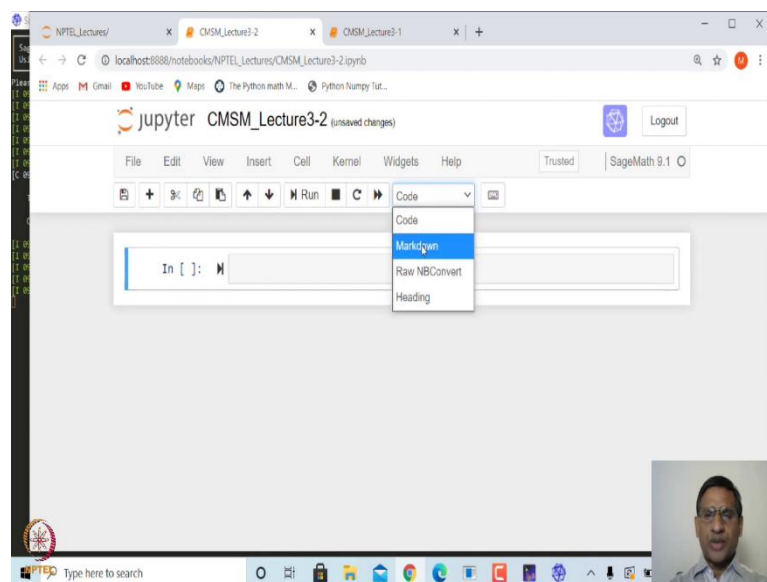


(Refer Slide Time: 01:42)



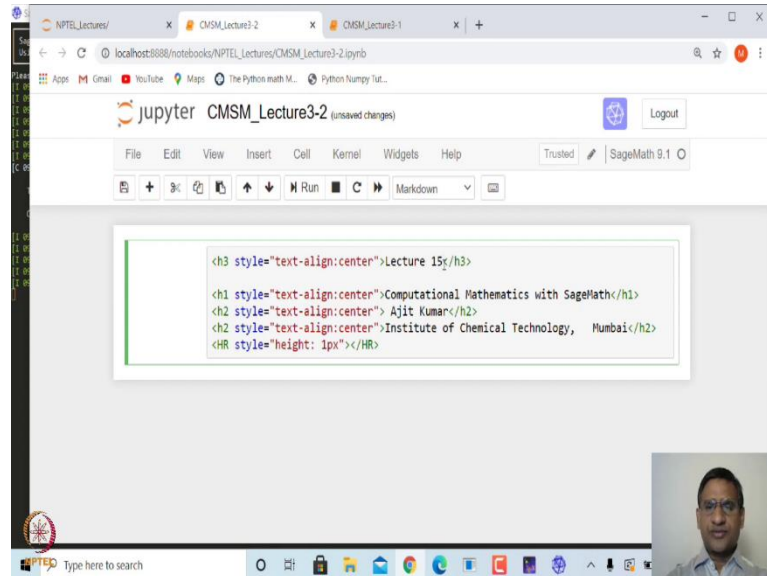
So, how do I do that? I will go to New and click on SageMath 9.1, this is the again you can see here untitled notebook. So, let me give the name computational mathematics which SageMath_lecture 3-2.

(Refer Slide Time: 01:56)



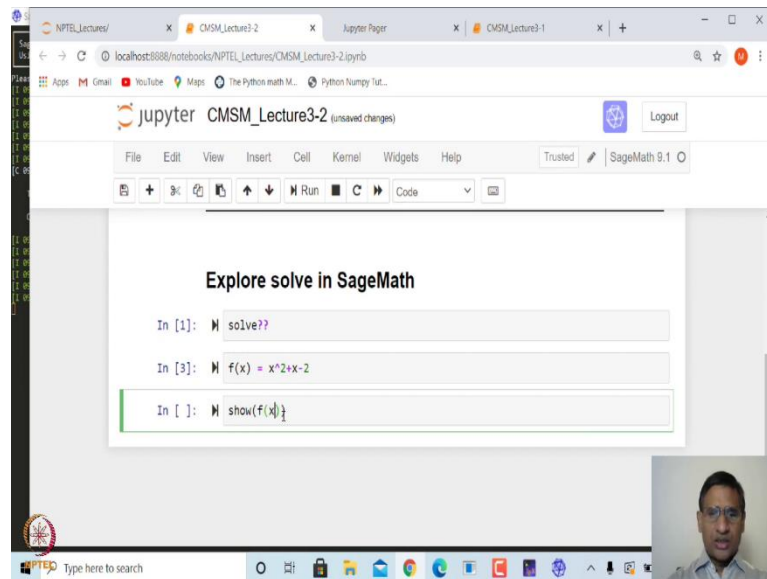
And then it will create a file name with default extension .ipynb. Now, in so, let me create a heading for this also.

(Refer Slide Time: 02:13)



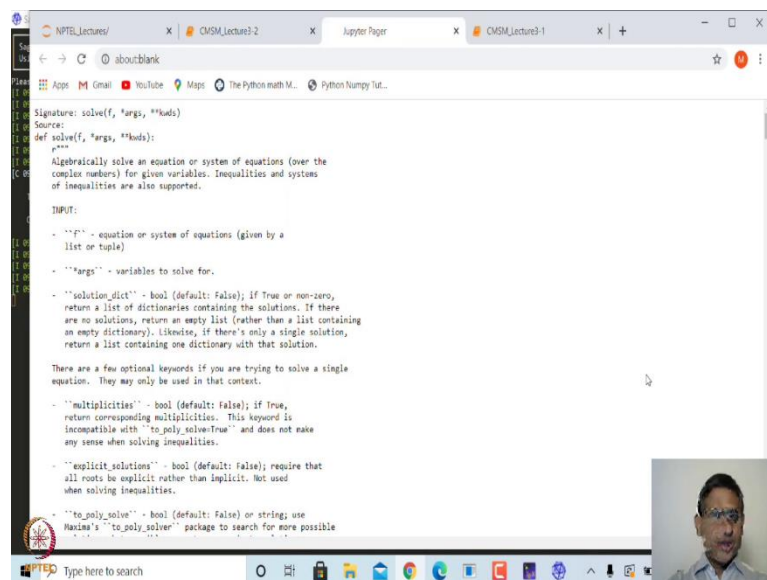
So, I can convert this into markdown and let me just paste what I copied from lecture previous lecture. This was lecture 14, now I will call lecture 15. So, and let us explore how to solve various objects in SageMath. So, let us explore solve function. So, solve function is very useful function in SageMath which can solve one equation, it can solve system of linear equation, it can solve inequalities, it can solve non-linear system of equations all these things can be done. So, first let us let me write a subtitle.

(Refer Slide Time: 02:56)



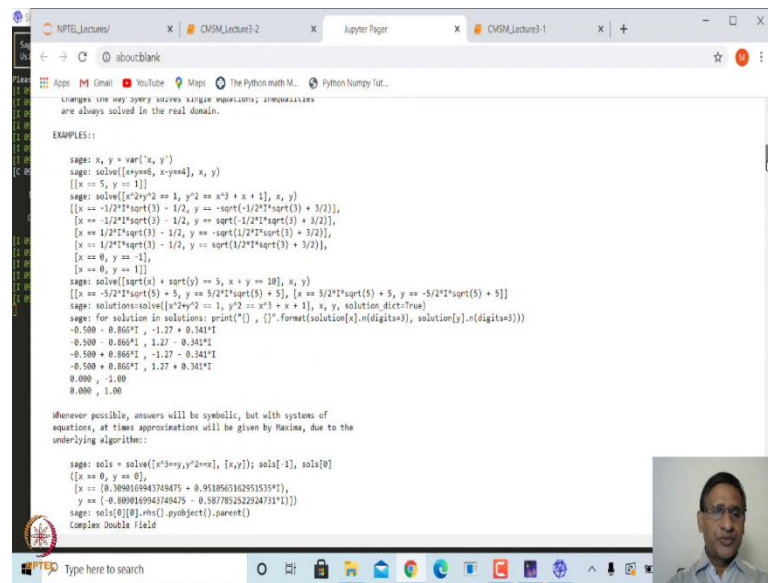
So, we want to let us say explore solve function in SageMath. So, that is the sub heading I am giving ok. So, first let us take a help on solve. So, I can say solve and let me put double question mark ('*solve??*'); I in the last lecture I explained what is meaning of double question mark and single question mark. So, this will open a help detailed help document along with source code. So, let us look at what it does.

(Refer Slide Time: 03:32)



So, this is what you see here. Let me make it slightly bigger let me also. So, this is the help document on solve and it tells you how what is it; it says algebraically solves an equation or system of equations over complex numbers and for detail for given variables it can also solve inequalities and system of inequalities all these things can be done.

(Refer Slide Time: 04:03)



```
sage: x, y = var('x, y')
sage: solve([x*y==5, x-y==4], x, y)
[[x == 5, y == 1]]
sage: solve([x^2+y^2 == 1, y^2 == x^3 + x + 1], x, y)
[[x == -1/2*I*sqrt(3) - 1/2, y == -sqrt(-1/2*I*sqrt(3) + 3/2)],
 [x == -1/2*I*sqrt(3) - 1/2, y == sqrt(-1/2*I*sqrt(3) + 3/2)],
 [x == 1/2*I*sqrt(3) - 1/2, y == -sqrt(1/2*I*sqrt(3) + 3/2)],
 [x == 1/2*I*sqrt(3) - 1/2, y == sqrt(1/2*I*sqrt(3) + 3/2)],
 [x == 0, y == 1],
 [x == 0, y == 1]]
sage: solve([sqrt(x) + sqrt(y) == 5, x + y == 10], x, y)
[[x == -5/2*I*sqrt(5) + 5, y == 5/2*I*sqrt(5) + 5], [x == 5/2*I*sqrt(5) + 5, y == -5/2*I*sqrt(5) + 5]]
sage: solutions=solve([x^2*y^2 == 1, y^2 == x^3 + x + 1], x, y, solution_dict=True)
sage: for solution in solutions: print("{} , {}".format(solution[x].n(digits=3), solution[y].n(digits=3)))
-0.500 - 0.866*I , -1.27 + 0.341*I
-0.500 - 0.866*I , 1.27 - 0.341*I
-0.500 + 0.866*I , -1.27 - 0.341*I
-0.500 + 0.866*I , 1.27 + 0.341*I
0.000 , -1.00
0.000 , 1.00

Whenever possible, answers will be symbolic, but with systems of
equations, at times approximations will be given by Maxima, due to the
underlying algorithm:

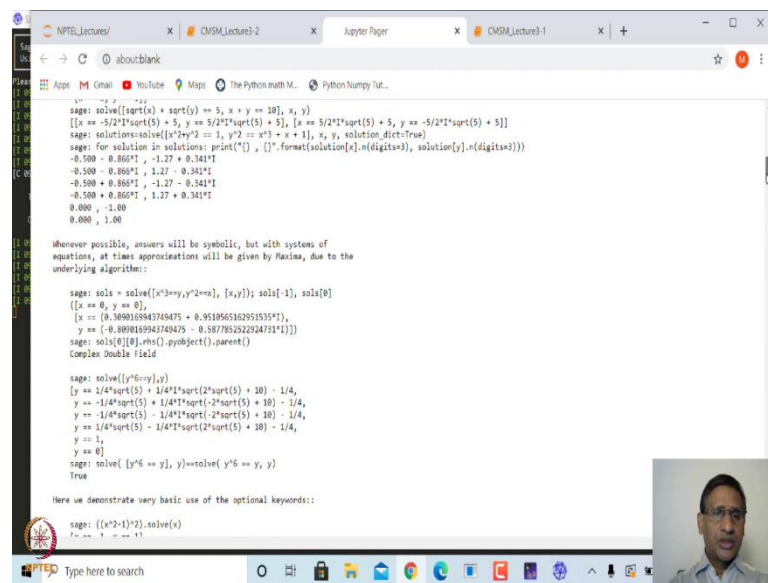
sage: sol = solve([x^3==y, y^2==x], [x,y]); sol[1], sol[0]
([x == 0, y == 0],
 [x == (0.3698169943740475 + 0.9519565162951535*I),
  y == (-0.0008169943740475 - 0.5877852522924731*I)])
sage: sol[0][0].rhs().pyobject().parent()
Complex Double field

sage: solve([y^6==x, y],
 [y == 1/4*sqrt(5) + 1/4*I*sqrt(2*sqrt(5) + 10) - 1/4,
  y == -1/4*sqrt(5) + 1/4*I*sqrt(-2*sqrt(5) + 10) - 1/4,
  y == -1/4*sqrt(5) - 1/4*I*sqrt(-2*sqrt(5) + 10) - 1/4,
  y == 1/4*sqrt(5) - 1/4*I*sqrt(2*sqrt(5) + 10) - 1/4,
  y == 1,
  y == 0])
sage: solve([y^6 == x], y)==solve([y^6 == x], y)
True

Here we demonstrate very basic use of the optional keywords:

sage: [(x^2-1)^2].solve(x)
[-1, 1, -1, 1]
```

(Refer Slide Time: 04:04)



```
sage: solve([sqrt(x) + sqrt(y) == 5, x + y == 10], x, y)
[[x == -5/2*I*sqrt(5) + 5, y == 5/2*I*sqrt(5) + 5], [x == 5/2*I*sqrt(5) + 5, y == -5/2*I*sqrt(5) + 5]]
sage: solutions=solve([x^2*y^2 == 1, y^2 == x^3 + x + 1], x, y, solution_dict=True)
sage: for solution in solutions: print("{} , {}".format(solution[x].n(digits=3), solution[y].n(digits=3)))
-0.500 - 0.866*I , -1.27 + 0.341*I
-0.500 - 0.866*I , 1.27 - 0.341*I
-0.500 + 0.866*I , -1.27 - 0.341*I
-0.500 + 0.866*I , 1.27 + 0.341*I
0.000 , -1.00
0.000 , 1.00

Whenever possible, answers will be symbolic, but with systems of
equations, at times approximations will be given by Maxima, due to the
underlying algorithm:

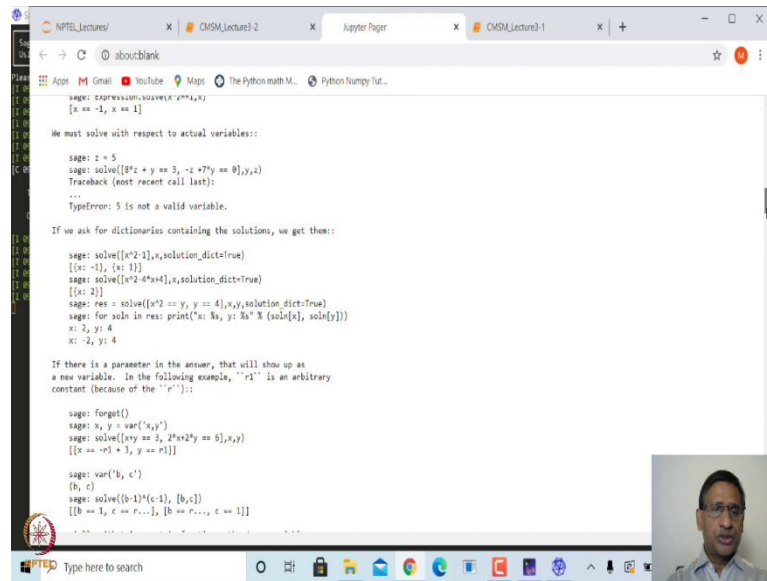
sage: sol = solve([x^3==y, y^2==x], [x,y]); sol[1], sol[0]
([x == 0, y == 0],
 [x == (0.3698169943740475 + 0.9519565162951535*I),
  y == (-0.0008169943740475 - 0.5877852522924731*I)])
sage: sol[0][0].rhs().pyobject().parent()
Complex Double field

sage: solve([y^6==x, y],
 [y == 1/4*sqrt(5) + 1/4*I*sqrt(2*sqrt(5) + 10) - 1/4,
  y == -1/4*sqrt(5) + 1/4*I*sqrt(-2*sqrt(5) + 10) - 1/4,
  y == -1/4*sqrt(5) - 1/4*I*sqrt(-2*sqrt(5) + 10) - 1/4,
  y == 1/4*sqrt(5) - 1/4*I*sqrt(2*sqrt(5) + 10) - 1/4,
  y == 1,
  y == 0])
sage: solve([y^6 == x], y)==solve([y^6 == x], y)
True

Here we demonstrate very basic use of the optional keywords:

sage: [(x^2-1)^2].solve(x)
[-1, 1, -1, 1]
```


(Refer Slide Time: 04:05)



```
sage: solve(x^2-1, x)
[[x == -1, y == 1]]

We must solve with respect to actual variables::

sage: z = 5
sage: solve([8^2 + y == 3, -z + 7^y == 0], y, z)
Traceback (most recent call last):
...
TypeError: 5 is not a valid variable.

If we ask for dictionaries containing the solutions, we get them::

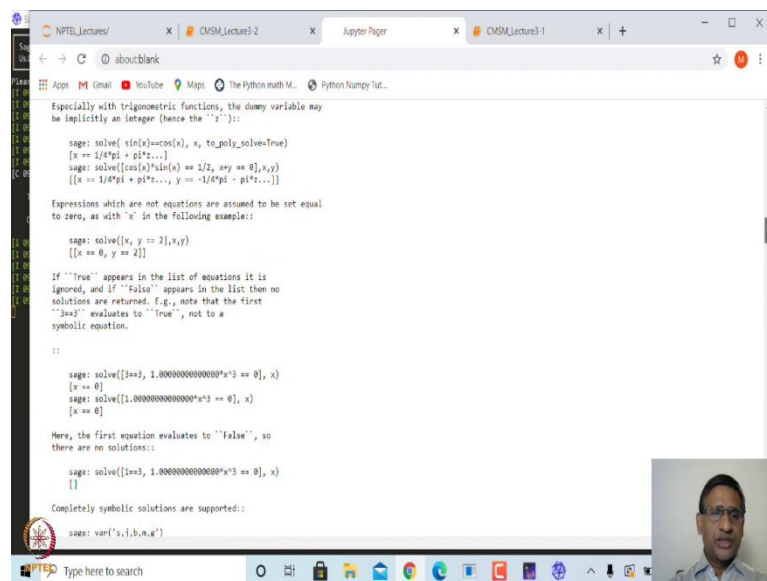
sage: solve([x^2-1], x, solution_dict=True)
[{x: -1}, {x: 1}]
sage: solve([x^2-4*x+4], x, solution_dict=True)
[{x: 2}]
sage: res = solve([x^2 == y, y == 4], x, y, solution_dict=True)
sage: for sol in res: print("x: %s, y: %s" % (sol[x], sol[y]))
x: 2, y: 4
x: -2, y: 4

If there is a parameter in the answer, that will show up as
a new variable. In the following example, "r1" is an arbitrary
constant (because of the "r"):

sage: forget()
sage: x, y = var('x,y')
sage: solve([x*y == 3, 2^x+2^y == 6], x, y)
[[x == -r1 + 3, y == r1]]

sage: var('b, c')
(b, c)
sage: solve((b-1)^(c-1), [b,c])
[[b == 1, c == r...], [b == r..., c == 1]]
```

(Refer Slide Time: 04:05)



```
Especially with trigonometric functions, the dummy variable may
be implicitly an integer (hence the "z")::

sage: solve(sin(x)==cos(x), x, to_poly_solve=True)
[x == 1/4*pi + pi^2*...]
sage: solve([cos(x)*sin(x) == 1/2, x*y == 0], x, y)
[[x == 1/4*pi + pi^2*..., y == -1/4*pi - pi^2*...]]

Expressions which are not equations are assumed to be set equal
to zero, as with 'x' in the following examples:

sage: solve([x, y == 2], x, y)
[[x == 0, y == 2]]

If "True" appears in the list of equations it is
ignored, and if "False" appears in the list then no
solutions are returned. E.g., note that the first
"3==3" evaluates to "True", not to a
symbolic equation.

::

sage: solve([3==3, 1.0000000000000000*x^3 == 0], x)
[x == 0]
sage: solve([1.0000000000000000*x^3 == 0], x)
[x == 0]

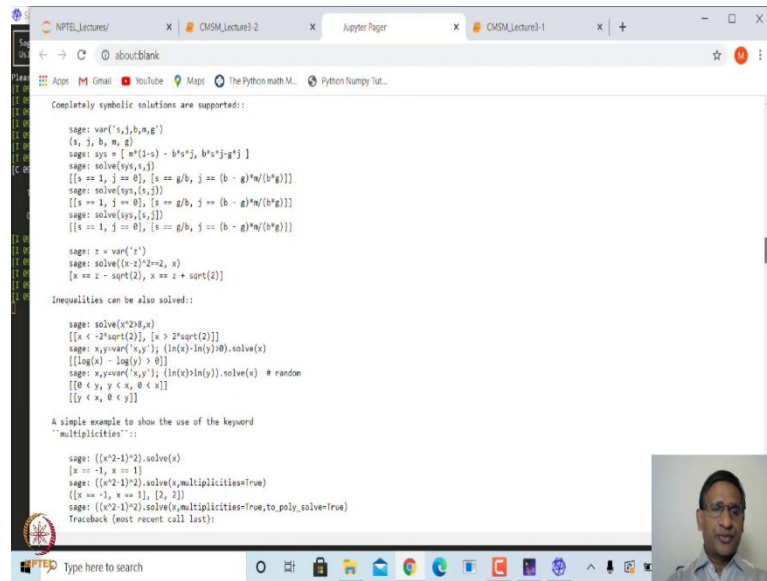
Here, the first equation evaluates to "False", so
there are no solutions:

sage: solve([1==3, 1.0000000000000000*x^3 == 0], x)
[]

Completely symbolic solutions are supported:

sage: var('a,i,b,m,g')
```

(Refer Slide Time: 04:06)



```
sage: var('s,j,b,w,g')
(s, j, b, w, g)
sage: sys = [ s*(1-s) - b*s*j, b*s*j-g*j ]
sage: solve(sys,s,j)
[[s == 1, j == 0], [s == g/b, j == (b - g)*w/(b*g)]]
sage: solve(sys,s,j)
[[s == 1, j == 0], [s == g/b, j == (b - g)*w/(b*g)]]
sage: solve(sys,s,j)
[[s == 1, j == 0], [s == g/b, j == (b - g)*w/(b*g)]]

sage: x = var('x')
sage: solve((x-2)^2==2, x)
[x == 2 - sqrt(2), x == 2 + sqrt(2)]

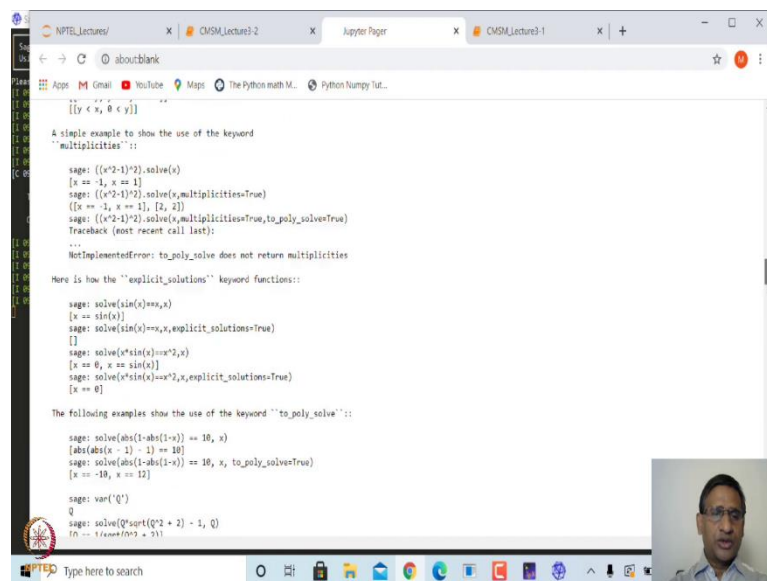
Inequalities can be also solved:

sage: solve(x^2>4,x)
[[x < -2*sqrt(2)], [x > 2*sqrt(2)]]
sage: x,y=var('x,y'); (ln(x)-ln(y)>0).solve(x)
[[log(x) - log(y) > 0]]
sage: x,y=var('x,y'); (ln(x)>ln(y)).solve(x) # random
[[0 < x, y < x, 0 < x]]
[[y < x, 0 < y]]

A simple example to show the use of the keyword
"multiplicities":

sage: ((x^2-1)^2).solve(x)
[x == -1, x == 1]
sage: ((x^2-1)^2).solve(x,multiplicities=True)
[[s == -1, s == 1], [2, 2]]
sage: ((x^2-1)^2).solve(x,multiplicities=True,to_poly_solve=True)
Traceback (most recent call last):
```

(Refer Slide Time: 04:06)



```
[[y < x, 0 < y]]

A simple example to show the use of the keyword
"multiplicities":

sage: ((x^2-1)^2).solve(x)
[x == -1, x == 1]
sage: ((x^2-1)^2).solve(x,multiplicities=True)
[[s == -1, s == 1], [2, 2]]
sage: ((x^2-1)^2).solve(x,multiplicities=True,to_poly_solve=True)
Traceback (most recent call last):
...
NotImplementedError: to_poly_solve does not return multiplicities

Here is how the "explicit_solutions" keyword functions:

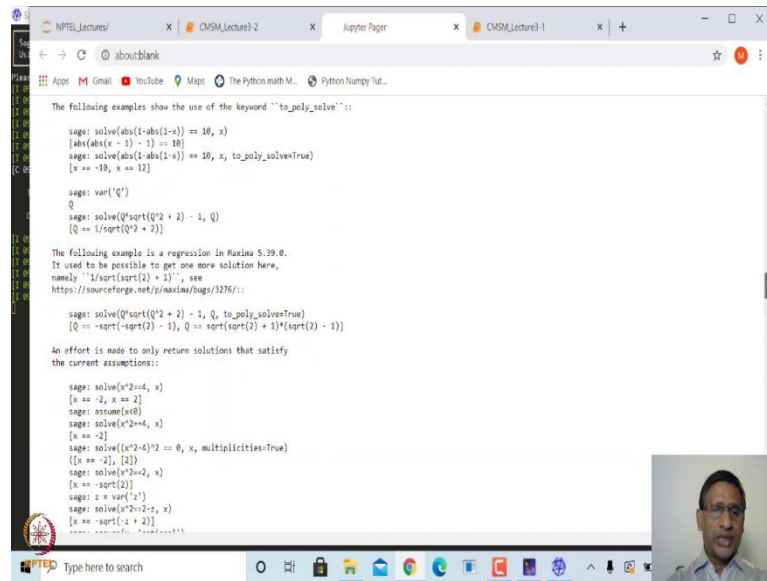
sage: solve(sin(x)==x,x)
[x == sin(x)]
sage: solve(sin(x)==x,x,explicit_solutions=True)
[]
sage: solve(x*sin(x)==x^2,x)
[x == 0, x == sin(x)]
sage: solve(x*sin(x)==x^2,x,explicit_solutions=True)
[x == 0]

The following examples show the use of the keyword "to_poly_solve":

sage: solve(abs(1-abs(1-x)) == 10, x)
[abs(abs(x - 1) - 1) == 10]
sage: solve(abs(1-abs(1-x)) == 10, x, to_poly_solve=True)
[x == -10, x == 12]

sage: var('Q')
Q
sage: solve(Q*sqrt(Q^2 + 2) - 1, Q)
[Q == 1/(sqrt(2)+1), Q == 1]
```

(Refer Slide Time: 04:07)



```
sage: solve(abs(1-abs(1-x)) == 10, x)
[soln(abs(x - 1) - 1) == 10]
sage: solve(abs(1-abs(1-x)) == 10, x, to_poly_solve=True)
[x == -10, x == 12]

sage: var('Q')
Q
sage: solve(Q*sqrt(Q^2 + 2) - 1, Q)
[Q == 1/sqrt(Q^2 + 2)]

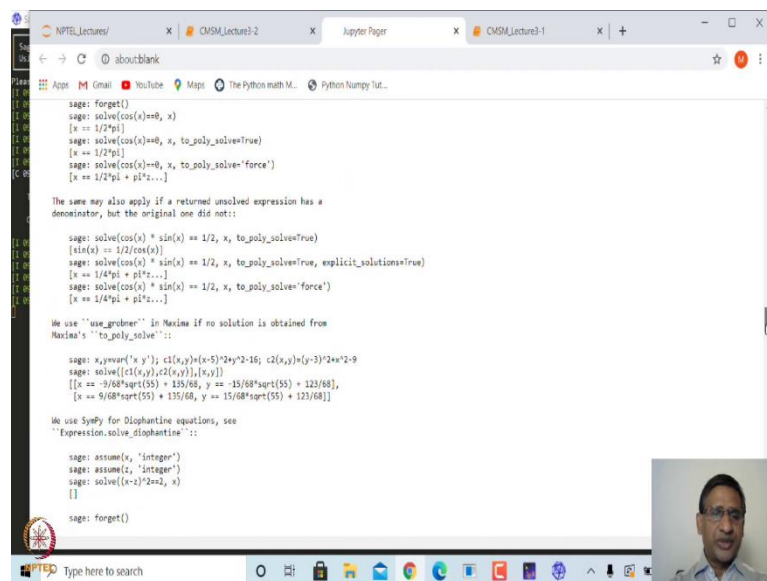
The following example is a regression in Maxima 5.39.0.
It used to be possible to get one more solution here,
namely "1/sqrt(sqrt(2) + 1)", see
https://sourceforge.net/p/maxima/bugs/3276/:

sage: solve(Q*sqrt(Q^2 + 2) - 1, Q, to_poly_solve=True)
[Q == -sqrt(-sqrt(2) - 1), Q == sqrt(sqrt(2) + 1)*(sqrt(2) - 1)]

An effort is made to only return solutions that satisfy
the current assumptions:

sage: solve(x^2==4, x)
[x == -2, x == 2]
sage: assume(x<0)
sage: solve(x^2==4, x)
[x == -2]
sage: solve((x^2-4)^2 == 0, x, multiplicities=True)
[(x == -2), [2]]
sage: solve(x^2==2, x)
[x == -sqrt(2)]
sage: z = var('z')
sage: solve(x^2==2-z, x)
[x == -sqrt(z + 2)]
.....
```

(Refer Slide Time: 04:08)



```
sage: forget()
sage: solve(cos(x)==0, x)
[x == 1/2*pi]
sage: solve(cos(x)==0, x, to_poly_solve=True)
[x == 1/2*pi]
sage: solve(cos(x)==0, x, to_poly_solve='force')
[x == 1/2*pi + pi*z...]

The same may also apply if a returned unsolved expression has a
denominator, but the original one did not:

sage: solve(cos(x) * sin(x) == 1/2, x, to_poly_solve=True)
[sin(x) == 1/2/cos(x)]
sage: solve(cos(x) * sin(x) == 1/2, x, to_poly_solve=True, explicit_solutions=True)
[x == 1/4*pi + pi*z...]
sage: solve(cos(x) * sin(x) == 1/2, x, to_poly_solve='force')
[x == 1/4*pi + pi*z...]

We use "use_grobner" in Maxima if no solution is obtained from
Maxima's "to_poly_solve":

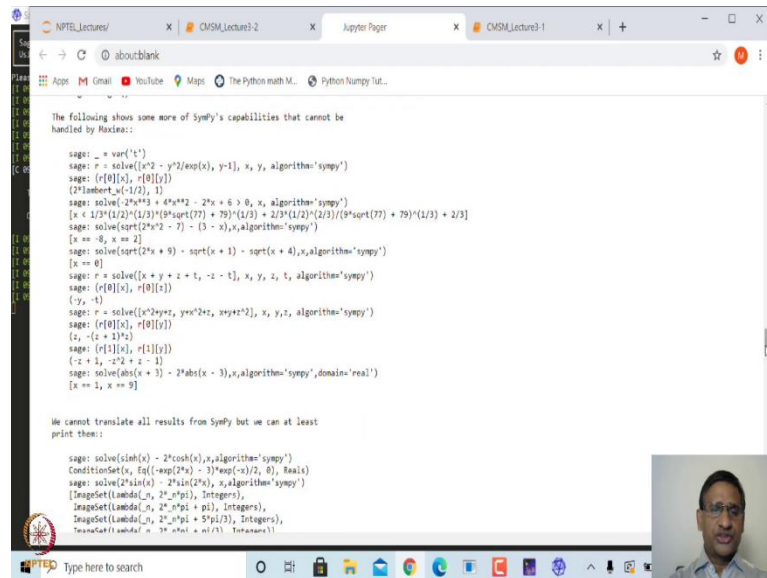
sage: x,y=var('x y'); c1(x,y)=(x-5)^2*y^2-16; c2(x,y)=(y-3)^2*x^2-9
sage: solve([c1(x,y),c2(x,y)],x,y)
[[x == -5/68*sqrt(55) + 135/68, y == -15/68*sqrt(55) + 123/68],
 [x == 5/68*sqrt(55) + 135/68, y == 15/68*sqrt(55) + 123/68]]

We use SymPy for Diophantine equations, see
"Expression.solve_diophantine":

sage: assume(x, 'integer')
sage: assume(z, 'integer')
sage: solve((x-z)^2==2, x)
[]

sage: forget()
```

(Refer Slide Time: 04:09)



```

sage: u = var('t')
sage: r = solve([x'' - y'^2/exp(x), y-1], x, y, algorithm='sympy')
sage: [r[0][x], r[0][y]]
(2^1/2*sqrt(2)*t^(1/2), 1)
sage: solve([2*x''+3 + 4*x'^2 - 2*x + 6 > 0, x, algorithm='sympy'])
[x < 1/3*(1/2)^(1/3)*(9*sqrt(77) + 79)^(1/3) + 2/3*(1/2)^(2/3)/(9*sqrt(77) + 79)^(1/3) + 2/3]
sage: solve(sqrt(2)*x'^2 - 7) - (3 - x), x, algorithm='sympy')
[x == 8, x == 2]
sage: solve(sqrt(2*x + 9) - sqrt(x + 1) - sqrt(x + 4), x, algorithm='sympy')
[x == 0]
sage: r = solve([x + y + z + t, -z - t], x, y, z, t, algorithm='sympy')
sage: [r[0][x], r[0][y]]
(-y, -t)
sage: r = solve([x^2*y+z, y*x^2*x, x*y*z^2], x, y, z, algorithm='sympy')
sage: [r[0][x], r[0][y]]
(x, -(-x + 3)^2)
sage: [r[1][x], r[1][y]]
(-z + 3, -z^2 + z - 1)
sage: solve(abs(x + 3) - 2*abs(x - 3), x, algorithm='sympy', domain='real')
[x == 2, x == 9]

We cannot translate all results from Sympy but we can at least
print them:

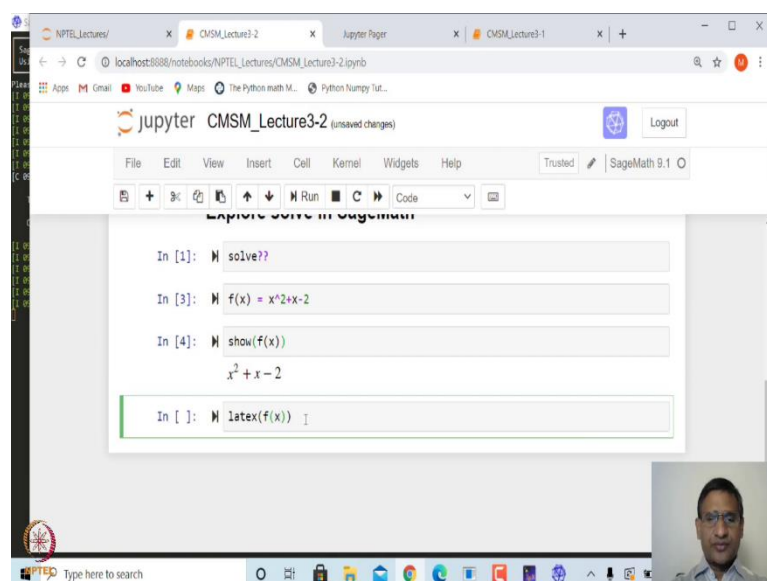
sage: solve(sinh(x) - 2*cosh(x), x, algorithm='sympy')
ConditionSet(x, Eq((-exp(2*x) - 3)*exp(-x)/2, 0), Real(x))
sage: solve(2*sin(x) - 2*sin(2*x), x, algorithm='sympy')
[ImageSet(Lambda_0, 2*pi*pi), Integers),
 ImageSet(Lambda_0, 2*pi*pi + pi), Integers),
 ImageSet(Lambda_0, 2*pi*pi + 5*pi/3), Integers),
 ImageSet(Lambda_0, 2*pi*pi + 5*pi/3 + 2*pi), Integers)]

```

And of course, it gives you gives you several examples. So, any of this example you can copy and paste it in sage worksheet. So, let me go to the worksheet. So, let us see. Suppose we want to first define a function.

So, the simplest way of defining a function let us say $y = f(x)$ in this is declare $f(x) = x^2 + x - 2$ that is the function. So, unlike Python here you can use for power you can use caret (^) symbol or you could have used double star (**) both are same.

(Refer Slide Time: 04:51)



```

In [1]: solve??

In [3]: f(x) = x^2+x-2

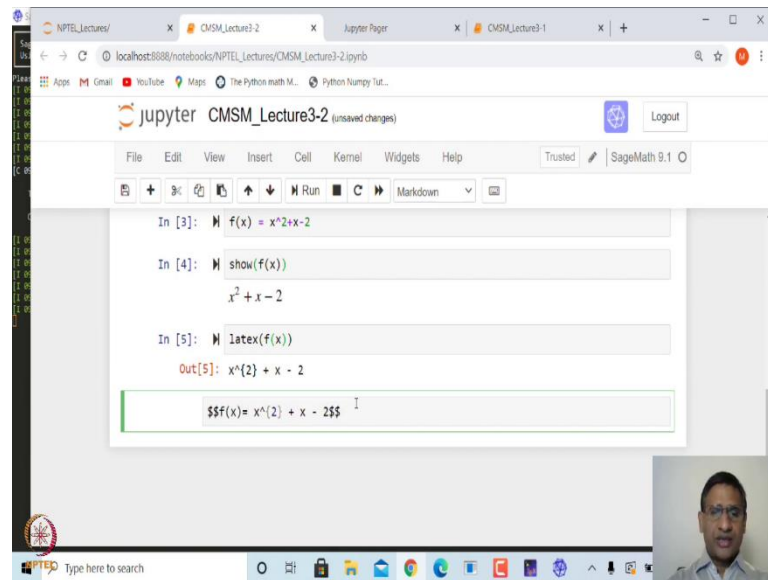
In [4]: show(f(x))
      x^2 + x - 2

In [ ]: latex(f(x))

```

So, let me just say $x^2 + x - 2$. Now, this is very simple way of defining a function. You could have also defined a function as we did in Python using `def` keyword. I can ask it to show what is $f(x)$, this will show you $f(x)$ in pretty print format this actually is rendered by LaTeX package.

(Refer Slide Time: 05:32)



The screenshot shows a Jupyter Notebook window titled 'CMSG_Lecture3-2 (unsaved changes)'. The notebook has three input cells and one output cell. The first cell contains the code `f(x) = x^2+x-2`. The second cell contains `show(f(x))`, which has rendered the output as $x^2 + x - 2$. The third cell contains `latex(f(x))`, which has rendered the output as $x^2 + x - 2$. The fourth cell is a text input field containing the LaTeX code `$$f(x)= x^2 + x - 2$$`. The Jupyter interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running cells, and markdown editing. A small video feed of a person is visible in the bottom right corner.

In fact, one can generate LaTeX code of this if I say `latex(f(x))` this will give LaTeX code as output. So, in case you are aware about LaTeX package, you would know what it does, you can copy paste this in this notebook. For example, I can just say a ctrl-C of this and let us say next cell I will convert into markdown and if I type `$$` and paste this LaTeX code and then again `$$` closed I can say $f(x)$ is equal to.

(Refer Slide Time: 06:02)

The screenshot shows a Jupyter Notebook titled 'CMSM_Lecture3-2' with the following content:

```

f(x) = x^2 + x - 2

In [6]: f(2)
Out[6]: 4

In [7]: f(-pi).n()
Out[7]: -pi + pi^2 - 2

In [ ]:

```

And then if I run this you will see that it has written this it has given you LaTeX output $f(x) = x^2 + x - 2$. So, this is what we are dealing with. Now, once we have created this function I can evaluate $f(x)$ at any point let us say at f at 2 or f at $-\pi$.

So, and so when I evaluate f at π it gives me the value in π itself if I want the numerical value by now you already know we can say `dot n()` and it will give me the numerical values. So, that is the numerical values. (Refer Slide Time: 06:42)

The screenshot shows the Jupyter Notebook with the following content:

```

Out[8]: 4.72881174749956

In [9]: solve(f(x)==0,x)
Out[9]: [x == 1, x == -2]

In [10]: solve(f(x)==0,x,solution_dict=True)
Out[10]: [{x: 1}, {x: -2}]

In [ ]: type(f)

```

So, now let us see how we can solve. So, we want to solve this $f(x) == 0$, we want to find x for which $f(x) = 0$. So, how do we make use of it? We can simply say `solve` and in the bracket $f(x)==0$ because it is an equation and with respect to variable x

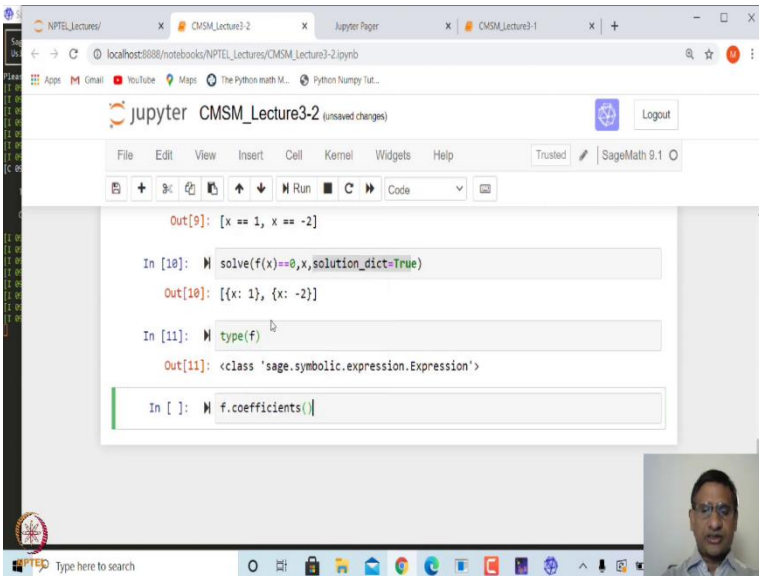
(`solve(f(x) == 0, x)`). So, it will tell me what are the roots. So, the roots are 1 and -2. So, it has given the roots as a list which has two elements; `x == 1` and `x == -2`.

You could also say an option; for example, if you want to print these roots as a dictionary there is an option called `solution_dict == True`, capital T r u e then the solution will be a list of dictionaries.

So, this is a list of 2 dictionaries. First dictionary says that x is 1 and second says that x is -2. So, this is very nice features in fact, one of the reason why I introduced dictionary when we were learning Python so that this kind of thing we can very easily explored.

And especially if you have lots of roots or solutions etc and you want to play with those solutions, you want to make use of those solutions it is much more convenient to get them as a dictionary so, that further calculations will become easier. You could also find. So, here again we have created f let me see what is the type of f. I do not know what it will give.

(Refer Slide Time: 08:44)

A screenshot of a Jupyter Notebook interface. The browser address bar shows 'localhost:8888/notebooks/NPTEL_Lectures/CMSM_Lecture3-2.ipynb'. The notebook title is 'CMSM_Lecture3-2 (unsaved changes)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The code area shows the following:

```
Out[9]: [x == 1, x == -2]

In [10]: solve(f(x)==0,x,solution_dict=True)
Out[10]: [{x: 1}, {x: -2}]

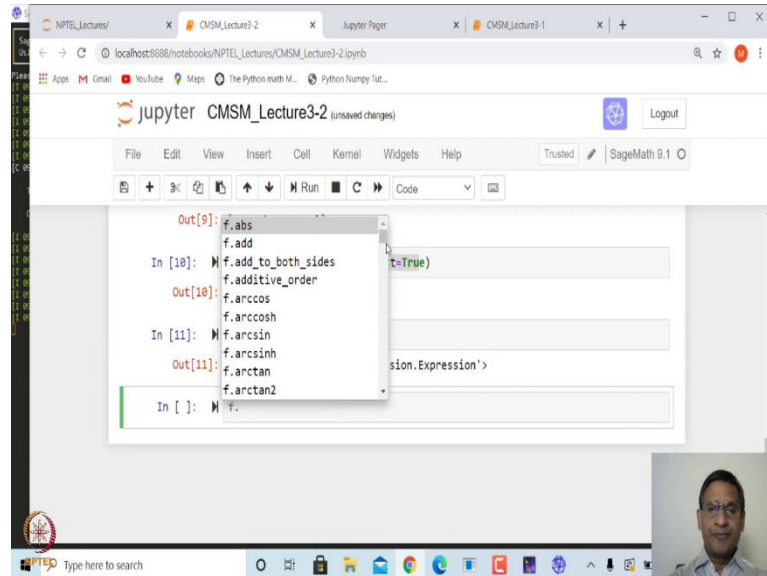
In [11]: type(f)
Out[11]: <class 'sage.symbolic.expression.Expression'>

In [ ]: f.coefficients()
```

The output of the last cell is partially visible. A small video feed of a man is visible in the bottom right corner of the notebook window.

It says that it is an expression. So, this is an expression class. And you can apply tab on this also a dot tab.

(Refer Slide Time: 08:50)

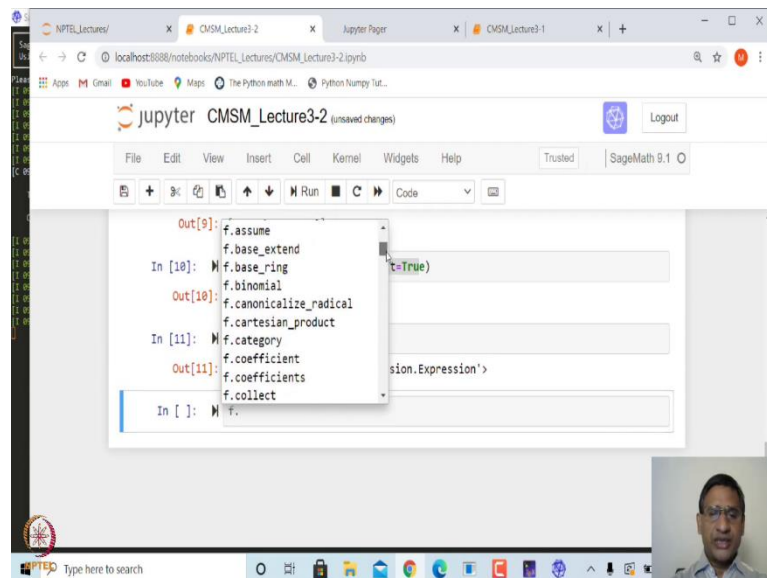


The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-2 (unsaved changes)' with the SageMath 9.1 kernel. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The code cell contains the following text:

```
Out[9]: f.
In [10]: f.add_to_both_sides
Out[10]: f.arccos
In [11]: f.arcsin
Out[11]: f.arcsinh
In [ ]: f.
```

A dropdown menu is open from the 'f.' input, displaying a list of available methods: f.abs, f.add, f.add_to_both_sides, f.additive_order, f.arccos, f.arccosh, f.arcsin, f.arcsinh, f.arctan, and f.arctan2. The background code cell also shows a partial definition of a function: `t=True)` and `sion.Expression'`.

(Refer Slide Time: 08:51)

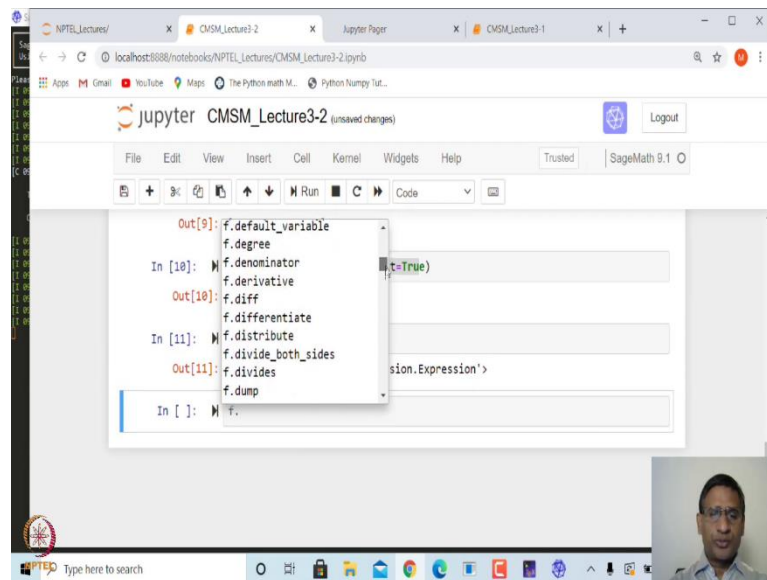


The screenshot shows the same Jupyter Notebook window. The code cell now contains:

```
Out[9]: f.assume
In [10]: f.base_extend
Out[10]: f.binomial
In [11]: f.canonicalize_radical
Out[11]: f.cartesian_product
In [ ]: f.
```

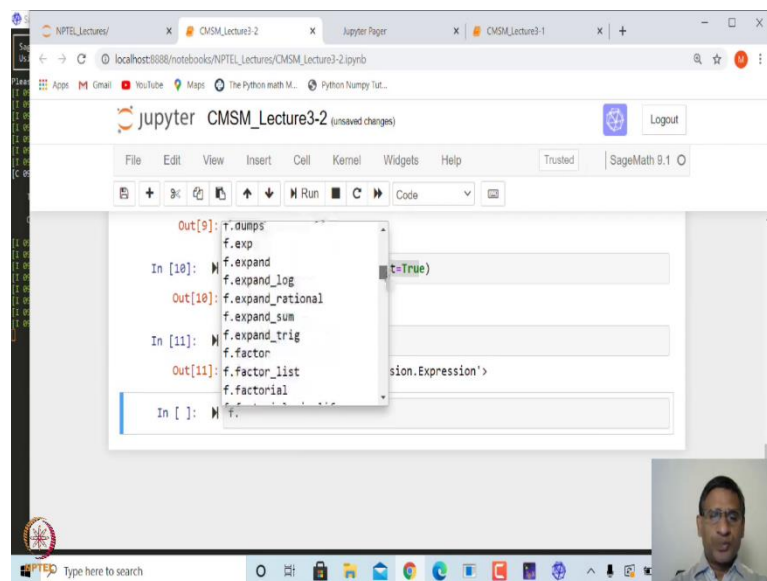
The dropdown menu is open, showing a different set of methods: f.assume, f.base_extend, f.base_ring, f.binomial, f.canonicalize_radical, f.cartesian_product, f.category, f.coefficient, f.coefficients, and f.collect. The background code cell shows `t=True)` and `sion.Expression'`.

(Refer Slide Time: 08:51)



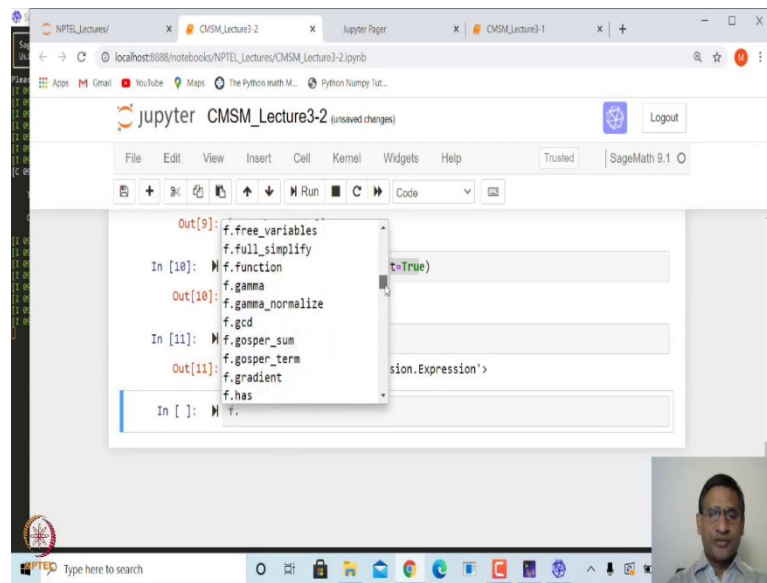
The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-2' with 'SageMath 9.1' in the top right. A dropdown menu is open for the variable 'f', displaying the following methods: `f.default_variable`, `f.degree`, `f.denominator`, `f.derivative`, `f.diff`, `f.differentiate`, `f.distribute`, `f.divide_both_sides`, `f.divides`, and `f.dump`. The notebook interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and saving. A small video feed of a person is visible in the bottom right corner.

(Refer Slide Time: 08:51)



This screenshot is similar to the one above, showing the same Jupyter Notebook interface. The dropdown menu for 'f' now lists different methods: `f.dumps`, `f.exp`, `f.expand`, `f.expand_log`, `f.expand_rational`, `f.expand_sum`, `f.expand_trig`, `f.factor`, `f.factor_list`, and `f.factorial`. The rest of the interface, including the menu bar, toolbar, and video feed, remains the same.

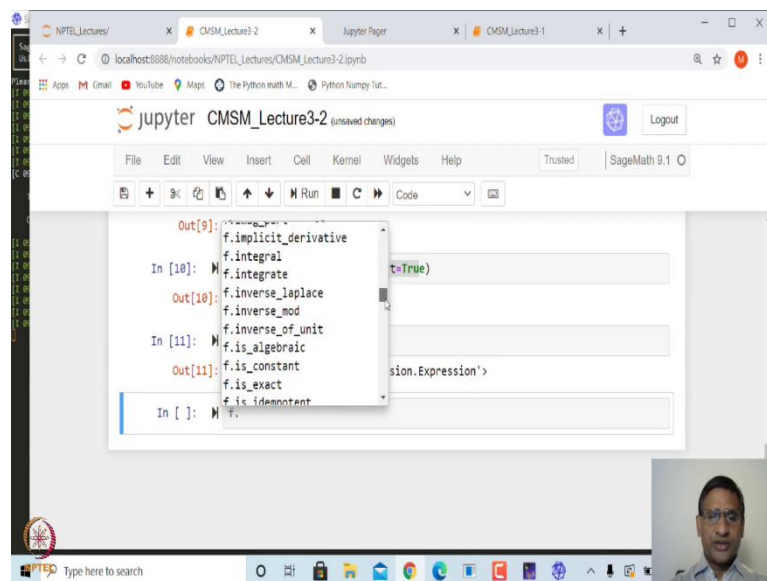
(Refer Slide Time: 08:51)



The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-2 (unsaved changes)' running on SageMath 9.1. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, code execution, and search. A dropdown menu is open, displaying a list of methods for the object 'f'. The methods listed are: `f.free_variables`, `f.full_simplify`, `f.function`, `f.gamma`, `f.gamma_normalize`, `f.gcd`, `f.gosper_sum`, `f.gosper_term`, `f.gradient`, and `f.has`. The input field below the dropdown shows `In []: f.`. In the background, code cells are visible, including one with `t=True)` and another with `sion.Expression'>`. A small video feed of a man is visible in the bottom right corner.

```
Out[9]: f.free_variables
         f.full_simplify
In [10]: f.function
         f.gamma
         f.gamma_normalize
         f.gcd
In [11]: f.gosper_sum
         f.gosper_term
         f.gradient
In [ ]: f.
```

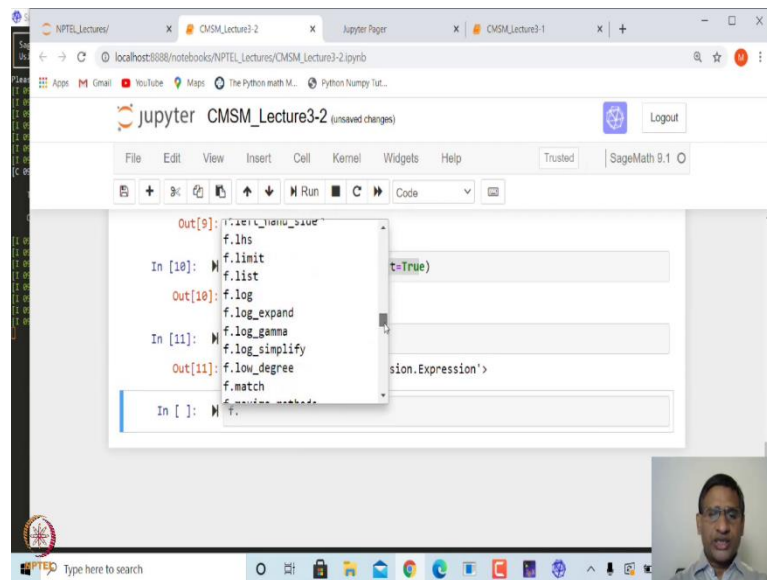
(Refer Slide Time: 08:52)



This screenshot is similar to the previous one, showing the same Jupyter Notebook interface. The dropdown menu for the object 'f' is open, displaying a different set of methods: `f.implicit_derivative`, `f.integral`, `f.integrate`, `f.inverse_laplace`, `f.inverse_mod`, `f.inverse_of_unit`, `f.is_algebraic`, `f.is_constant`, `f.is_exact`, and `f.is_idempotent`. The input field below the dropdown shows `In []: f.`. The background code cells and the video feed in the bottom right corner are the same as in the previous screenshot.

```
Out[9]: f.implicit_derivative
         f.integral
In [10]: f.integrate
         f.inverse_laplace
         f.inverse_mod
         f.inverse_of_unit
In [11]: f.is_algebraic
         f.is_constant
         f.is_exact
         f.is_idempotent
In [ ]: f.
```

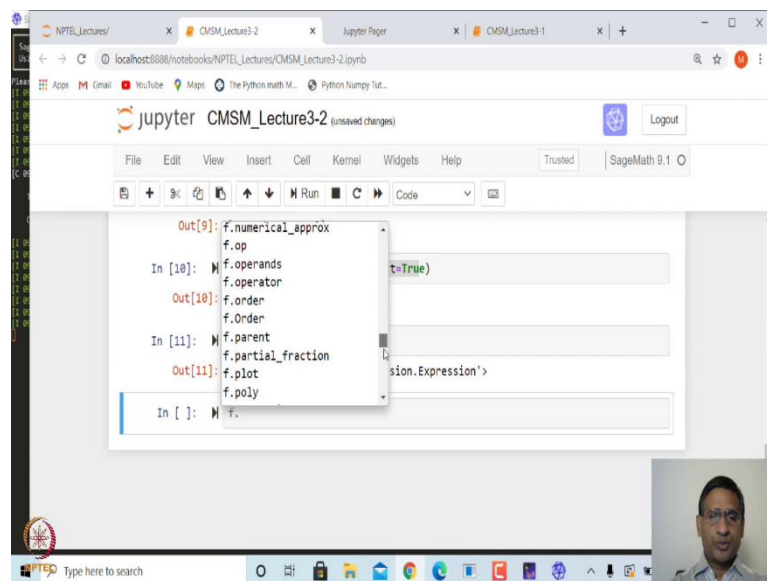
(Refer Slide Time: 08:52)



The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-2 (unsaved changes)' running on SageMath 9.1. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for saving, undo, redo, and running code. A dropdown menu is open, displaying a list of methods for the object 'f'. The methods listed are: f.lhs, f.limit, f.list, f.log, f.log_expand, f.log_gamma, f.log_simplify, f.low_degree, f.match, and f. The current input in the code cell is 'In []: f.'. A small video feed of a person is visible in the bottom right corner of the notebook window.

```
Out[9]: f.lhs
In [10]: f.limit
Out[10]: f.list
In [11]: f.log
Out[11]: f.log_expand
In [ ]: f.log_gamma
Out[ ]: f.log_simplify
In [ ]: f.low_degree
In [ ]: f.match
In [ ]: f.
```

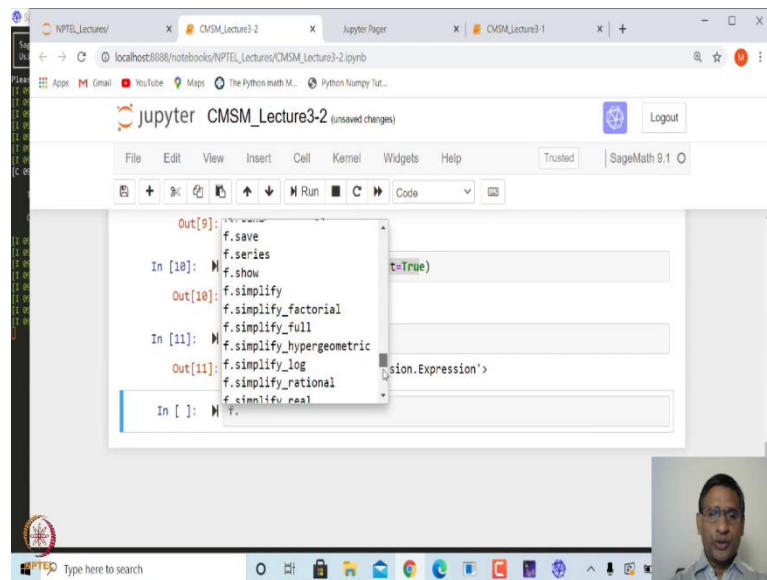
(Refer Slide Time: 08:53)



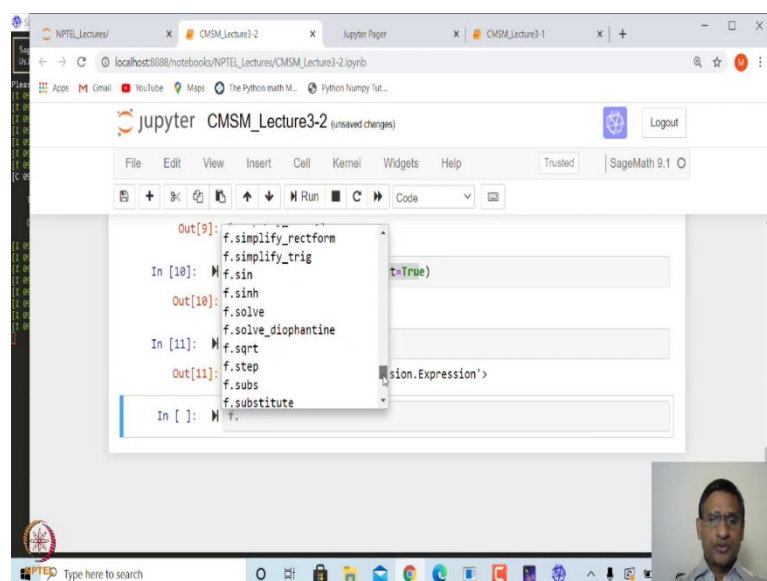
This screenshot is similar to the previous one, showing the same Jupyter Notebook interface. The dropdown menu for the object 'f' is open, displaying a different set of methods: f.numerical_approx, f.op, f.operands, f.operator, f.order, f.Order, f.parent, f.partial_fraction, f.plot, and f.poly. The current input in the code cell is 'In []: f.'. The video feed in the bottom right corner shows the same person.

```
Out[9]: f.numerical_approx
In [10]: f.op
Out[10]: f.operands
In [11]: f.operator
Out[11]: f.order
In [ ]: f.Order
Out[ ]: f.parent
In [ ]: f.partial_fraction
In [ ]: f.plot
In [ ]: f.poly
In [ ]: f.
```

(Refer Slide Time: 08:53)



(Refer Slide Time: 08:54)



f dot tab if you apply there will be again several options which you can explore. So, for example, one of the options just now I saw is for example, coefficients let us see. So, coefficients yes. So, if I click on coefficients it will tell me what are the coefficients of x , x square and things like that, right.

(Refer Slide Time: 09:08)

The screenshot shows a Jupyter Notebook titled 'CMSM_Lecture3-2' running on a local host. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running code, and viewing output. The notebook contains several input-output pairs:

```

Out[12]: [[x |--> -2, x |--> 0], [x |--> 1, x |--> 1], [x |--> 1, x |--> 2]]

In [13]: f.roots()
Out[13]: [(1, 1), (-2, 1)]

In [14]: (x^2+x+1).roots()
Out[14]: [(-1/2*I*sqrt(3) - 1/2, 1), (1/2*I*sqrt(3) - 1/2, 1)]

In [ ]: var('a,b,c')
         solve(a*x^2+b*x+c==0,x)

```

A small video inset in the bottom right corner shows a man speaking.

If I want to find roots I can say `f.roots()` and it will give me the roots. So, it has two roots; 1 and -2 and this gives me multiplicity. So, this `f.roots()` finds roots of a polynomial including complex roots. So, for example, if I say, let us say I have $x^2 + x + 1 = 0$ and let me just say `dot` and then say `roots`, it should find the complex roots these are complex roots right.

So, you can see here this `solve` function not only can solve in real over real fields, but also over complex fields. Now, suppose you want to for example, solve $ax^2 + bx + c = 0$ for a, b, c arbitrary. So, how do we do that? a, b, c we have to declare as a variable.

So, by default in Sage x is thought of as a variable. For example, we declare the function $f(x)$ we did not declare x as a variable by default it thinks of x as a variable, but any other thing when you want to declare or when you want to use as a variable you have to declare. So, how do I declare? I will say `variable` and the bracket a, b, c and then I want to say `solve` for example, $a * x^2 + b * x + c$ equal to 0 for x .

(Refer Slide Time: 11:08)

A screenshot of a Jupyter Notebook interface. The browser tabs at the top include 'NPTEL_Lectures/', 'CMSM_Lecture3-2', 'Jupyter Pager', and 'CMSM_Lecture3-1'. The address bar shows 'localhost:8888/notebooks/NPTEL_Lectures/CMSM_Lecture3-2.ipynb'. The notebook title is 'CMSM_Lecture3-2 (unsaved changes)'. The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with icons for saving, running, and other actions. The code area shows two input cells. The first cell contains `(x^2+x+1).roots()` and the output is `[(-1/2*I*sqrt(3) - 1/2, 1), (1/2*I*sqrt(3) - 1/2, 1)]`. The second cell contains `var('a,b,c')` followed by `sol = solve(a*x^2+b*x+c==0,x)`, and the output is `[x == -1/2*(b + sqrt(b^2 - 4*a*c))/a, x == -1/2*(b - sqrt(b^2 - 4*a*c))/a]`. A third empty input cell is at the bottom. A small video feed of a man is visible in the bottom right corner.

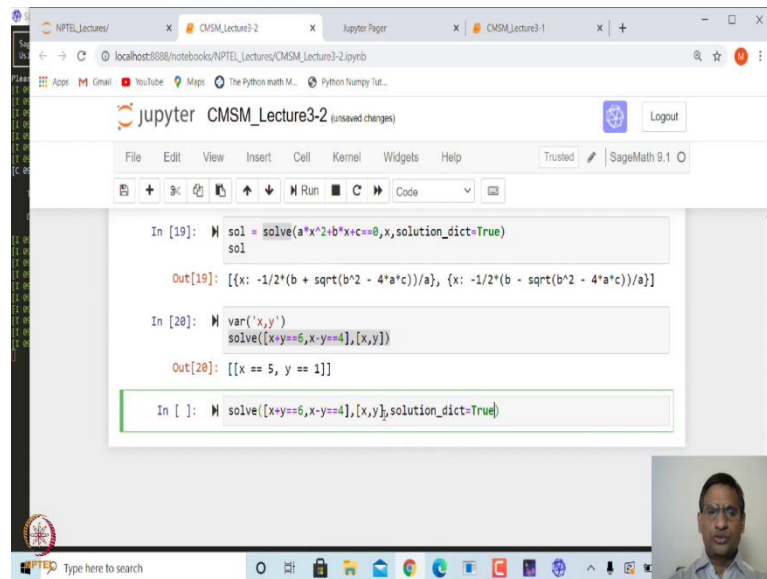
Then it will show you the first root is $-\frac{1}{2} * (b + \sqrt{b^2 - 4ac})/a$ and the second root is $-\frac{1}{2} * (b - \sqrt{b^2 - 4ac})/a$. So, let us store this in solution *sol*.

(Refer Slide Time: 11:28)

A screenshot of a Jupyter Notebook interface, similar to the previous one. The code area shows three input cells. The first cell contains `var('a,b,c')` followed by `sol = solve(a*x^2+b*x+c==0,x)`. The second cell contains `show(sol)` and the output is a pretty-printed quadratic formula:
$$x = -\frac{b + \sqrt{b^2 - 4ac}}{2a}, x = -\frac{b - \sqrt{b^2 - 4ac}}{2a}$$
. The third cell contains `sol = solve(a*x^2+b*x+c==0,x,sol)`. A dropdown menu is open below this cell, showing options: `solution_dict=`, `sol`, `solve`, `solve_diophantine`, `solve_ineq`, and `solve_mod`. A small video feed of a man is visible in the bottom right corner.

And if I ask it to how show solution *sol* then it will print the solution in nice pretty print format. You could have also said that display this as a dictionary. So, if I say solution. So, I am just typing keyword *sol* and pressing tab then already it gives me the options and then choose the appropriate option and this is true.

(Refer Slide Time: 11:53)



```
In [19]: sol = solve(a*x^2+b*x+c==0,x,solution_dict=True)
sol
Out[19]: [{x: -1/2*(b + sqrt(b^2 - 4*a*c))/a}, {x: -1/2*(b - sqrt(b^2 - 4*a*c))/a}]

In [20]: var('x,y')
solve([x+y==6,x-y==4],[x,y])
Out[20]: [{x == 5, y == 1}]

In [ ]: solve([x+y==6,x-y==4],[x,y],solution_dict=True)
```

So, now if I ask it to see what is solution, it gives you as a dictionary. So, this dictionary option is very useful later on also we will make use of this. You can also solve system of linear equations using solve function. So, for example, let us say I want to solve two equations let us say $x + y = 6$ and $x - y = 4$ with respect to x , and y , how do I do that?

So, first we need to declare y as a variable, x is already a variable, but y we need to declare. But it is also a good idea to declare both x and y as a variable because it is possible that you might have used x as some variable i.e. you might have stored some value in the variable x in that case it may create problem.

So, let me create x and y as variables and now we will say solve. So, you need to give the two equation as a list. So, first equation is $x + y == 6$ and second equation is $x - y == 4$ and you want to solve with respect to x and y . So, in the bracket you can write x and y , you can write this without bracket also. So, that is the solution $x = 5$ and $y = 1$.

(Refer Slide Time: 13:18)

```

Out[20]: [[x == 5, y == 1]]

In [21]: solve([x-y==6,x-y==4],[x,y],solution_dict=True)
Out[21]: [{x: 5, y: 1}]

In [22]: solve([x+y==5],[x,y])
Out[22]: [[x == -r1 + 5, y == r1]]

In [ ]: var('x,y')
         solve([x^2+y^2==4, x*y==2],[x,y])

```

Again here you could have said solution dictionary equal to True. So, I can say solution dictionary is equal to True then it will give you solution as a dictionary and you can extend this to two equations, in three equations, or in four equations in as many variables as you want and if you for example, if the system has multiple solutions it will print the solution in terms of variables.

So, if for example, if I have said solve let us say only one equation with respect to x and y this is let us say 5; then it will print the solution in terms of variable r1, r1 can take any value. So, it has infinitely many solutions, if I want to find a particular solution one can give value of r1 i.e. some particular value of r1 and then find it. You could also solve system of non-linear equations using the same solve function.

So, if let us say I want to find intersection of let us say circle $x^2 + y^2 = 4$ and hyperbola $x * y = 2$. So, if you imagine geometrically it should have 4 solutions. So, let us see what we get. So, if I say again let us declare x and y as a variable you do not need to declare again and again, but it is a good practice actually.

So, when I say solve and the first equation is $x^2 + y^2 == 4$ and the second equation is $x * y == 2$ and we want to solve with respect to variable x and y, then it gives you 2 solutions, only 2 solutions.

(Refer Slide Time: 15:10)

A screenshot of a Jupyter Notebook interface. The browser tabs at the top include 'NPTEL_Lectures/', 'CMSM_Lecture3-2', 'Jupyter Pager', and 'CMSM_Lecture3-1'. The address bar shows 'localhost:8888/notebooks/NPTEL_Lectures/CMSM_Lecture3-2.ipynb'. The notebook title is 'CMSM_Lecture3-2 (unsaved changes)'. The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with icons for saving, running, and other actions. The code area shows two input cells. The first cell contains the code `solve([x+y==5],[x,y])` and its output is `Out[22]: [[x == -r1 + 5, y == r1]]`. The second cell contains the code `var('x,y')` followed by `solve([x^2+y^2==4, x*y==2],[x,y])` and its output is `Out[23]: [[x == -sqrt(2), y == -sqrt(2)], [x == sqrt(2), y == sqrt(2)]]`. A third input cell is empty. A small video feed of a man is visible in the bottom right corner.

```

In [22]: solve([x+y==5],[x,y])
Out[22]: [[x == -r1 + 5, y == r1]]

In [23]: var('x,y')
         solve([x^2+y^2==4, x*y==2],[x,y])
Out[23]: [[x == -sqrt(2), y == -sqrt(2)], [x == sqrt(2), y == sqrt(2)]]

In [ ]:

```

(Refer Slide Time: 15:17)

A screenshot of a Jupyter Notebook interface, similar to the one above. The code area shows an input cell with `var('x,y')` followed by `solve([x^2+y^2==4, x*y==1],[x,y])`. The output is `Out[24]: [[x == -sqrt(sqrt(3) + 2), y == sqrt(sqrt(3) + 2)*(sqrt(3) - 2)], [x == sqrt(sqrt(3) + 2), y == -sqrt(sqrt(3) + 2)*(sqrt(3) - 2)], [x == -sqrt(-sqrt(3) + 2), y == -1/2*sqrt(3)*sqrt(2) - 1/2*sqrt(2)], [x == sqrt(-sqrt(3) + 2), y == 1/2*sqrt(3)*sqrt(2) + 1/2*sqrt(2)]]`. A small video feed of a man is visible in the bottom right corner.

```

In [24]: var('x,y')
         solve([x^2+y^2==4, x*y==1],[x,y])
Out[24]: [[x == -sqrt(sqrt(3) + 2), y == sqrt(sqrt(3) + 2)*(sqrt(3) - 2)], [x == sqrt(sqrt(3) + 2), y == -sqrt(sqrt(3) + 2)*(sqrt(3) - 2)], [x == -sqrt(-sqrt(3) + 2), y == -1/2*sqrt(3)*sqrt(2) - 1/2*sqrt(2)], [x == sqrt(-sqrt(3) + 2), y == 1/2*sqrt(3)*sqrt(2) + 1/2*sqrt(2)]]

In [ ]:

```

If I say $x * y = 1$ then it has 4 solutions.

(Refer Slide Time: 15:28)

The screenshot shows a Jupyter Notebook interface with the following code and output:

```

Out[22]: [[x == -r1 + 5, y == r1]]

In [25]: var('x,y')
          solve([x^2+y^2==1, x*y==1/4],[x,y])

Out[25]: [[x == -1/2*sqrt(sqrt(3) + 2), y == 1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)],
          [x == 1/2*sqrt(sqrt(3) + 2), y == -1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)],
          [x == -1/2*sqrt(-sqrt(3) + 2), y == -1/4*sqrt(3)*sqrt(2) - 1/4*sqrt(2)],
          [x == 1/2*sqrt(-sqrt(3) + 2), y == 1/4*sqrt(3)*sqrt(2) + 1/4*sqrt(2)]]

In [ ]: solve([x^2+y^2==1, x*y==1/4],[x,y],solution_dict=True)

```

So, or let me say $x^2 + y^2 = 1$ and $x * y =$ let us say 1 by 4, then also it has 4 solutions. And again we could have used we could have used this as a solution we can display in terms of dictionaries. So, let me say solution dictionary is equal to True.

(Refer Slide Time: 15:51)

The screenshot shows a Jupyter Notebook interface with the following code and output:

```

In [26]: solve([x^2+y^2==1, x*y==1/4],[x,y],solution_dict=True)

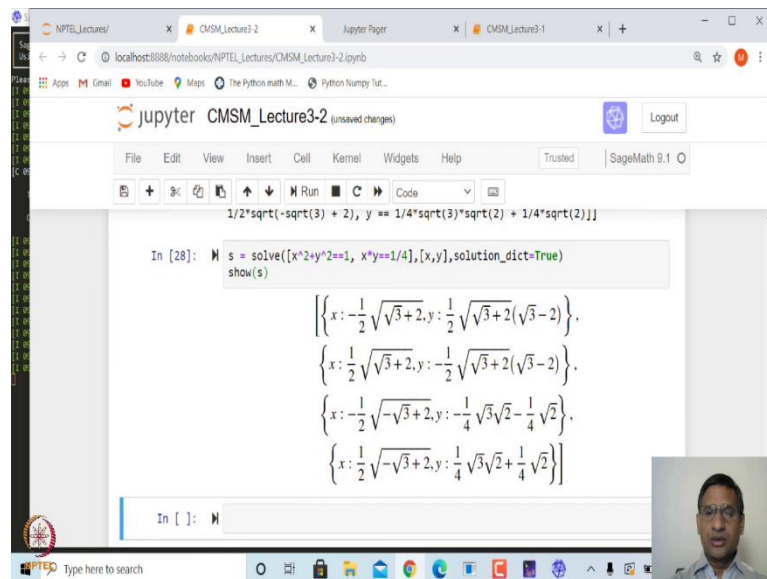
Out[26]: [{x: -1/2*sqrt(sqrt(3) + 2), y: 1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)},
          {x: 1/2*sqrt(sqrt(3) + 2), y: -1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)},
          {x: -1/2*sqrt(-sqrt(3) + 2), y: -1/4*sqrt(3)*sqrt(2) - 1/4*sqrt(2)},
          {x: 1/2*sqrt(-sqrt(3) + 2), y: 1/4*sqrt(3)*sqrt(2) + 1/4*sqrt(2)}]

In [ ]:

```

So these are the solutions.

(Refer Slide Time: 15:54)



The screenshot shows a Jupyter Notebook interface with a browser window. The notebook is titled 'CMSM_Lecture3-2 (unsaved changes)'. The code cell contains the following Python code:

```
1/2*sqrt(-sqrt(3) + 2), y == 1/4*sqrt(3)*sqrt(2) + 1/4*sqrt(2)]]

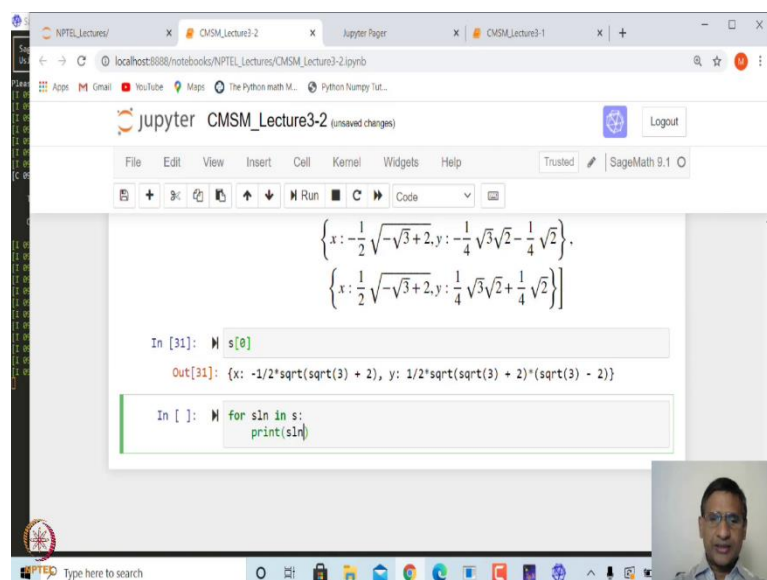
In [28]: M = solve([x^2+y^2==1, x*y==1/4],[x,y],solution_dict=True)
show(s)
```

The output of the code is a list of four solutions, each represented as a dictionary with 'x' and 'y' keys. The solutions are:

$$\left\{ x: -\frac{1}{2}\sqrt{\sqrt{3}+2}, y: \frac{1}{2}\sqrt{\sqrt{3}+2}(\sqrt{3}-2) \right\},$$
$$\left\{ x: \frac{1}{2}\sqrt{\sqrt{3}+2}, y: -\frac{1}{2}\sqrt{\sqrt{3}+2}(\sqrt{3}-2) \right\},$$
$$\left\{ x: -\frac{1}{2}\sqrt{-\sqrt{3}+2}, y: -\frac{1}{4}\sqrt{3}\sqrt{2} - \frac{1}{4}\sqrt{2} \right\},$$
$$\left\{ x: \frac{1}{2}\sqrt{-\sqrt{3}+2}, y: \frac{1}{4}\sqrt{3}\sqrt{2} + \frac{1}{4}\sqrt{2} \right\}$$

Now, suppose if I store this into s. So, if I ask it to show what is s; these are the values of s. And so, there are 4 solutions. Each of this solution if I want to kind of extract it is quite easy. So, you can see here it is a list of dictionary.

(Refer Slide Time: 16:17)



The screenshot shows a Jupyter Notebook interface with a browser window. The notebook is titled 'CMSM_Lecture3-2 (unsaved changes)'. The code cell contains the following Python code:

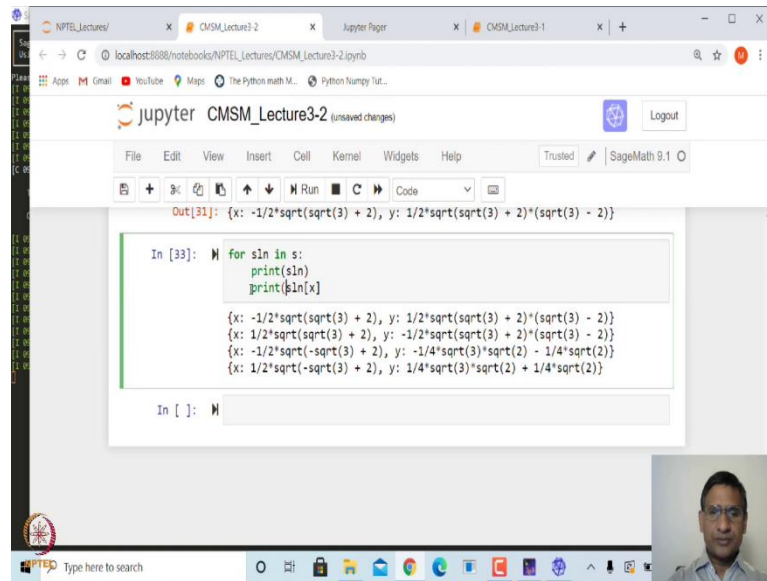
```
{x: -1/2*sqrt(-sqrt(3) + 2), y: -1/4*sqrt(3)*sqrt(2) - 1/4*sqrt(2)},
{x: 1/2*sqrt(-sqrt(3) + 2), y: 1/4*sqrt(3)*sqrt(2) + 1/4*sqrt(2)}]]

In [31]: M = s[0]
Out[31]: {x: -1/2*sqrt(sqrt(3) + 2), y: 1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)}

In [ ]: for sln in s:
print(sln)
```

So, this is s[0] this is the first solution. Now, you can run a loop over this solution set. So, I can say for let us say s or let us say it is solution so, 'for sln in s: print(sln)' and this it will print all the solutions.

(Refer Slide Time: 16:44)



A screenshot of a Jupyter Notebook interface. The browser tabs show 'NPTEL Lectures', 'CMSM_Lecture3-2', 'Jupyter Pager', and 'CMSM_Lecture3-1'. The address bar shows 'localhost:8888/notebooks/NPTEL_Lectures/CMSM_Lecture3-2.ipynb'. The notebook title is 'CMSM_Lecture3-2 (unsaved changes)'. The previous cell's output is visible: `Out[31]: {x: -1/2*sqrt(sqrt(3) + 2), y: 1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)}`. The current cell (In [33]) contains the following code:

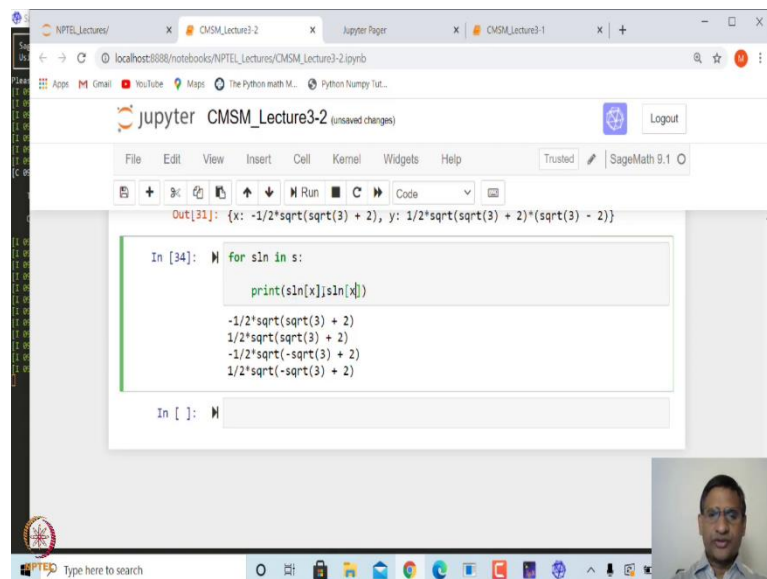
```
In [33]: for s in s:
         print(s)
         print(sln[x])
```

The output of this cell shows four dictionaries, each representing a solution s and its corresponding x value from sln :

```
{x: -1/2*sqrt(sqrt(3) + 2), y: 1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)}
{x: 1/2*sqrt(sqrt(3) + 2), y: -1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)}
{x: -1/2*sqrt(-sqrt(3) + 2), y: -1/4*sqrt(3)*sqrt(2) - 1/4*sqrt(2)}
{x: 1/2*sqrt(-sqrt(3) + 2), y: 1/4*sqrt(3)*sqrt(2) + 1/4*sqrt(2)}
```

Now, each one of these is a dictionary. So, I can even say *sln* solution the value of *x*.

(Refer Slide Time: 17:06)



A screenshot of the same Jupyter Notebook interface. The previous cell's output is still visible. The current cell (In [34]) contains the following code:

```
In [34]: for s in s:
         print(sln[x])
```

The output of this cell shows the x values extracted from each dictionary in the previous output:

```
-1/2*sqrt(sqrt(3) + 2)
1/2*sqrt(sqrt(3) + 2)
-1/2*sqrt(-sqrt(3) + 2)
1/2*sqrt(-sqrt(3) + 2)
```

So, let me ask it to print; so this is the first solution.

(Refer Slide Time: 17:12)

```

Out[31]: {x: -1/2*sqrt(sqrt(3) + 2), y: 1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)}

In [36]: for s in s:
          print(sln[x],sln[y])

-1/2*sqrt(sqrt(3) + 2) 1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)
1/2*sqrt(sqrt(3) + 2) -1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)
-1/2*sqrt(-sqrt(3) + 2) -1/4*sqrt(3)*sqrt(2) - 1/4*sqrt(2)
1/2*sqrt(-sqrt(3) + 2) 1/4*sqrt(3)*sqrt(2) + 1/4*sqrt(2)

In [ ]: solve(x^2-2*x-1>8,x)

```

So, for each one it is printing sln x and let us say this is the first solution this is the second solution so it gives you a list. So, I can make a list. So, this is how you can extract these solutions.

So, this is advantage of using this option called solution dictionary so that it becomes easier for you to extract. And you can also solve inequalities using solve functions so for example, let us say I say solve and let me write inequality say $x^2 - 2 * x - 1 > 8$ and you want to solve this for x.

(Refer Slide Time: 17:57)

```

In [36]: for s in s:
          print(sln[x],sln[y])

-1/2*sqrt(sqrt(3) + 2) 1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)
1/2*sqrt(sqrt(3) + 2) -1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)
-1/2*sqrt(-sqrt(3) + 2) -1/4*sqrt(3)*sqrt(2) - 1/4*sqrt(2)
1/2*sqrt(-sqrt(3) + 2) 1/4*sqrt(3)*sqrt(2) + 1/4*sqrt(2)

In [37]: solve(x^2-2*x-1>8,x)

Out[37]: [[x > 0, x < (1/9)]]

In [39]: var('x,y')
          g(x,y) = x^2+2*x*y-3*y+exp(-x^2-y^2)

In [ ]: g

In [ ]:

```

So, it gives you the solution of this inequality or you can have list of inequalities that will also work. So, you can see here this solve function is very useful which can solve one equation in one variable, it can solve system of linear equations, system of non-linear equations, it can also solve inequalities. So, it is quite useful.

So, if you want to for example, in between we give some say sub heading you can do that. So, how do I do that? For example, here we have to solve system of inequalities system of non-linear equations.

(Refer Slide Time: 18:54)

```

In [22]: solve([x+y==5],[x,y])
Out[22]: [[x == -r1 + 5, y == r1]]

In [25]: var('x,y')
         solve([x^2+y^2==1, x*y==1/4],[x,y])
Out[25]: [[x == -1/2*sqrt(sqrt(3) + 2), y == 1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)],
          [x == 1/2*sqrt(sqrt(3) + 2), y == -1/2*sqrt(sqrt(3) + 2)*(sqrt(3) - 2)],
          [x == -1/2*sqrt(-sqrt(3) + 2), y == -1/4*sqrt(3)*sqrt(2) - 1/4*sqrt(2)],
          [x == 1/2*sqrt(-sqrt(3) + 2), y == 1/4*sqrt(3)*sqrt(2) + 1/4*sqrt(2)]]

### Solving a system of non linear equations

In [30]: s = solve([x^2+y^2==1, x*y==1/4],[x,y],solution_dict=True)
         show(s)

```

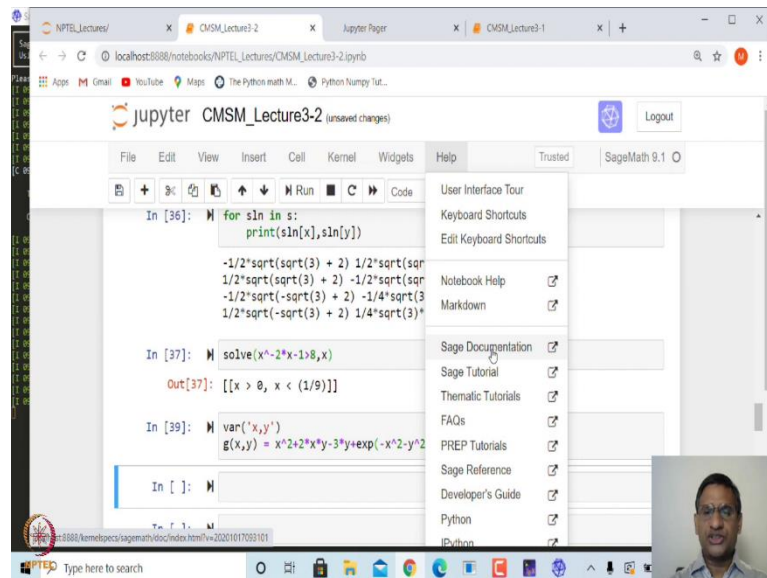
So, I if I want after this one I can insert one cell and convert this into markdown and then we can say; solving a system of non-linear equations. Similarly, you can do it with other things. So, it is quite convenient to use this solve function. Now, let us look at suppose you want to declare say function of two variables.

So, the way to do is first you need to declare the x and y as a variable and then just let us just write $g(x,y) = x^2 + 2*x*y - 3*y + \text{exponential of } (-x^2 - y^2)$. And again whatever methods that you want to apply on g all these things will be available using g dot and tab. So, you can go through this list and then explore these functions.

So, this of course, we will come to this kind of functions when we do when we explore calculus. Now, another thing I just wanted to tell you, last time I told you, if you go to SageMath website it provides you lot of documentations, but when you install sage this

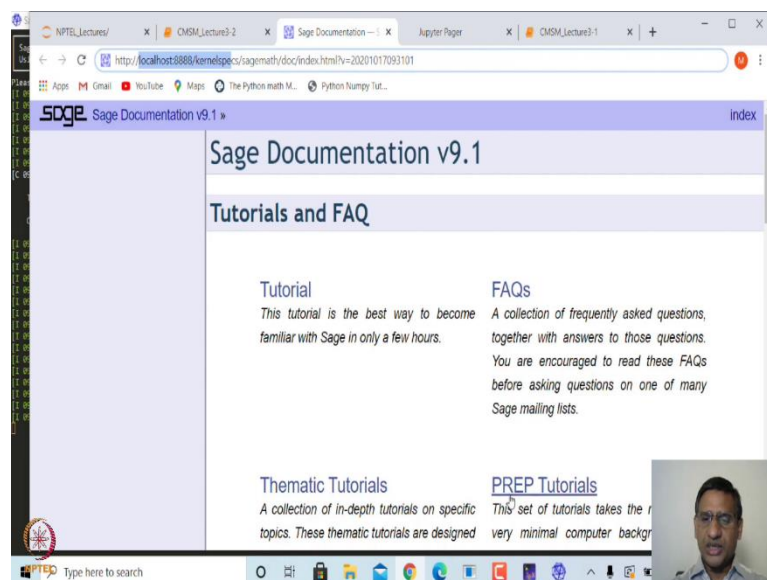
also has some inbuilt document. One help document is when we take help on a particular function or particular method using `help()` or using question mark '?' or double question mark '??'.

(Refer Slide Time: 20:36)



But, if you click on this help you can look at sage documentation.

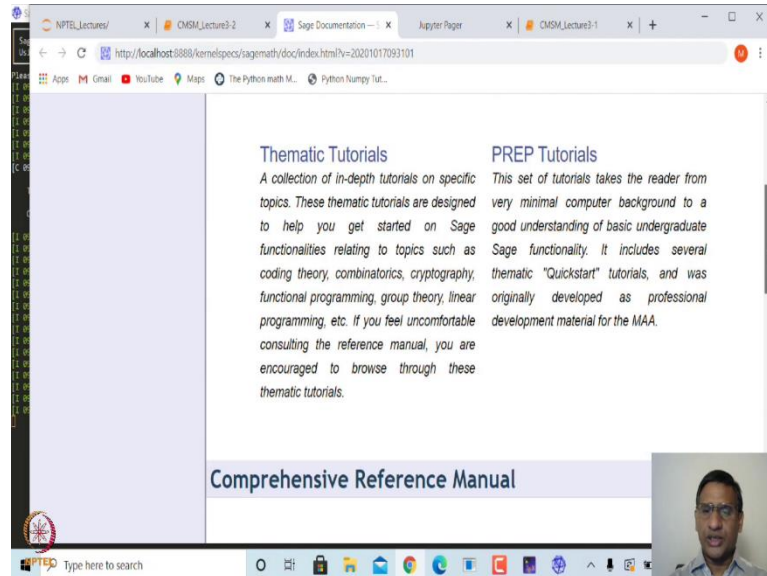
(Refer Slide Time: 20:39)



If you click on sage documentation. So, this is actually if you look at it and notice this is not going to the internet it is from the local host itself. So, you can go through this tutorial

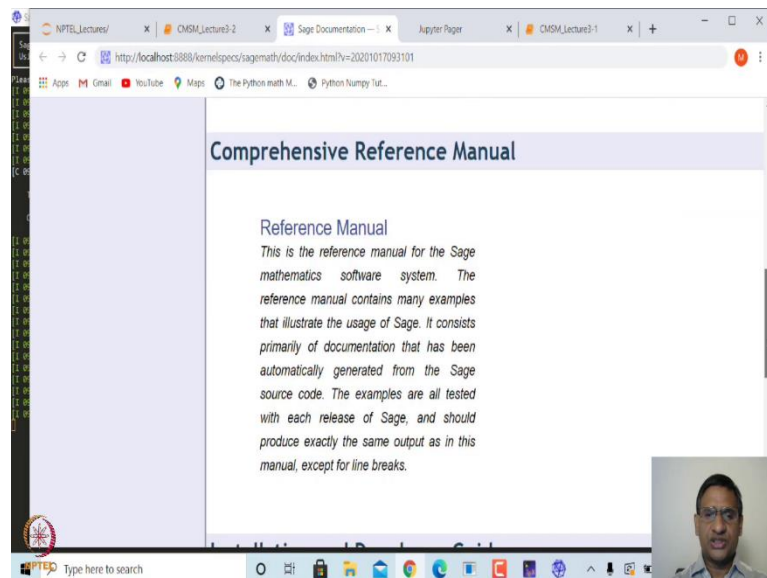
the thematic tutorial there is a PREP tutorial you can click on any of these things and then explore.

(Refer Slide Time: 20:56)



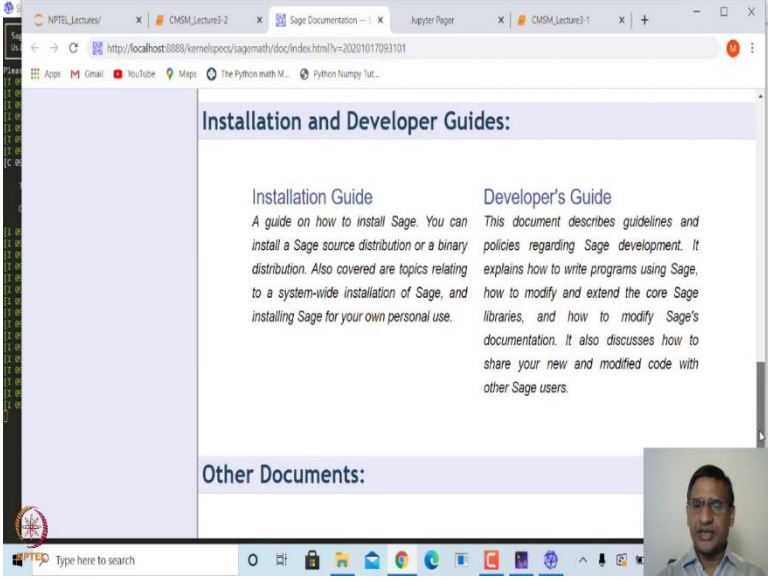
The screenshot shows a web browser window with multiple tabs. The active tab is titled "Sage Documentation" and displays the URL <http://localhost:8888/kernelspecs/sagemath/doc/index.html?v=20201017093101>. The page content is divided into two columns. The left column is titled "Thematic Tutorials" and contains the text: "A collection of in-depth tutorials on specific topics. These thematic tutorials are designed to help you get started on Sage functionalities relating to topics such as coding theory, combinatorics, cryptography, functional programming, group theory, linear programming, etc. If you feel uncomfortable consulting the reference manual, you are encouraged to browse through these thematic tutorials." The right column is titled "PREP Tutorials" and contains the text: "This set of tutorials takes the reader from very minimal computer background to a good understanding of basic undergraduate Sage functionality. It includes several thematic 'Quickstart' tutorials, and was originally developed as professional development material for the MAA." Below these columns is a purple header bar with the text "Comprehensive Reference Manual". A small video feed of a man is visible in the bottom right corner of the browser window.

(Refer Slide Time: 20:57)



The screenshot shows the same web browser window as the previous one, but the content has changed. The purple header bar now reads "Comprehensive Reference Manual". Below it, the text reads: "Reference Manual" followed by "This is the reference manual for the Sage mathematics software system. The reference manual contains many examples that illustrate the usage of Sage. It consists primarily of documentation that has been automatically generated from the Sage source code. The examples are all tested with each release of Sage, and should produce exactly the same output as in this manual, except for line breaks." The video feed of the man remains in the bottom right corner.

(Refer Slide Time: 21:00)



The screenshot shows a web browser window displaying the Sage Documentation website. The browser's address bar shows the URL `http://localhost:8888/kernelspecs/sagemath/doc/index.html?v=20201017093101`. The page has a light blue header with the title "Installation and Developer Guides:". Below this, there are two columns of text. The left column is titled "Installation Guide" and describes a guide on how to install Sage, covering topics like installing a Sage source distribution or a binary distribution, and installing Sage for personal use. The right column is titled "Developer's Guide" and describes guidelines and policies regarding Sage development, explaining how to write programs using Sage, how to modify and extend the core Sage libraries, and how to modify Sage's documentation. Below these columns is a section titled "Other Documents:". In the bottom right corner, there is a small video feed of a man with glasses and a light blue shirt. The browser's taskbar at the bottom shows various application icons, including the Windows Start button, File Explorer, Edge, and several other applications.

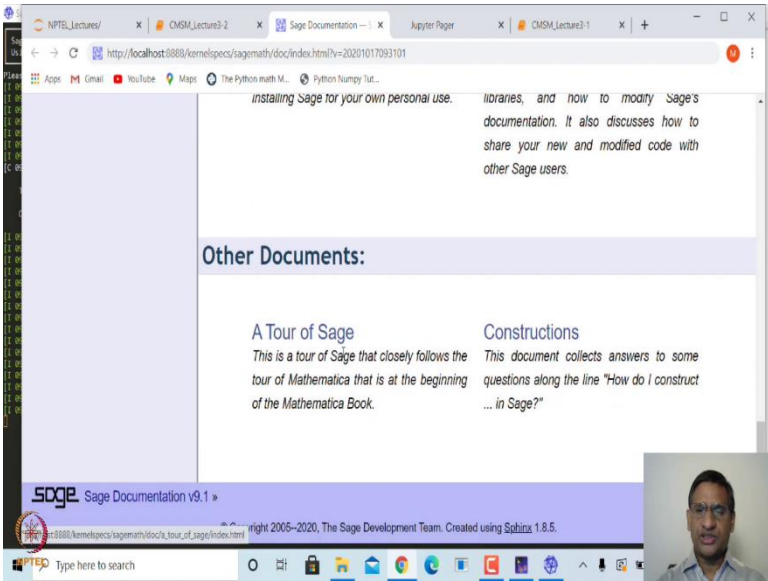
Installation and Developer Guides:

Installation Guide
A guide on how to install Sage. You can install a Sage source distribution or a binary distribution. Also covered are topics relating to a system-wide installation of Sage, and installing Sage for your own personal use.

Developer's Guide
This document describes guidelines and policies regarding Sage development. It explains how to write programs using Sage, how to modify and extend the core Sage libraries, and how to modify Sage's documentation. It also discusses how to share your new and modified code with other Sage users.

Other Documents:

(Refer Slide Time: 21:05)



The screenshot shows the same Sage Documentation website, but now the "Other Documents:" section is expanded. It contains two columns of text. The left column is titled "A Tour of Sage" and describes a tour of Sage that closely follows the tour of Mathematica that is at the beginning of the Mathematica Book. The right column is titled "Constructions" and describes a document that collects answers to some questions along the line "How do I construct ... in Sage?". At the bottom of the page, there is a footer that reads "Sage Documentation v9.1" and "© 2005–2020, The Sage Development Team. Created using Sphinx 1.8.5." In the bottom right corner, there is a small video feed of the same man as in the previous slide. The browser's taskbar at the bottom shows various application icons, including the Windows Start button, File Explorer, Edge, and several other applications.

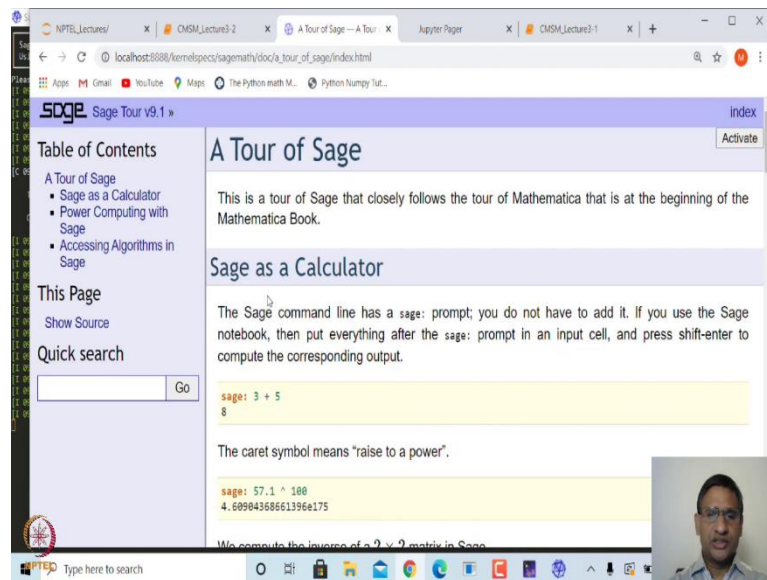
Other Documents:

A Tour of Sage
This is a tour of Sage that closely follows the tour of Mathematica that is at the beginning of the Mathematica Book.

Constructions
This document collects answers to some questions along the line "How do I construct ... in Sage?"

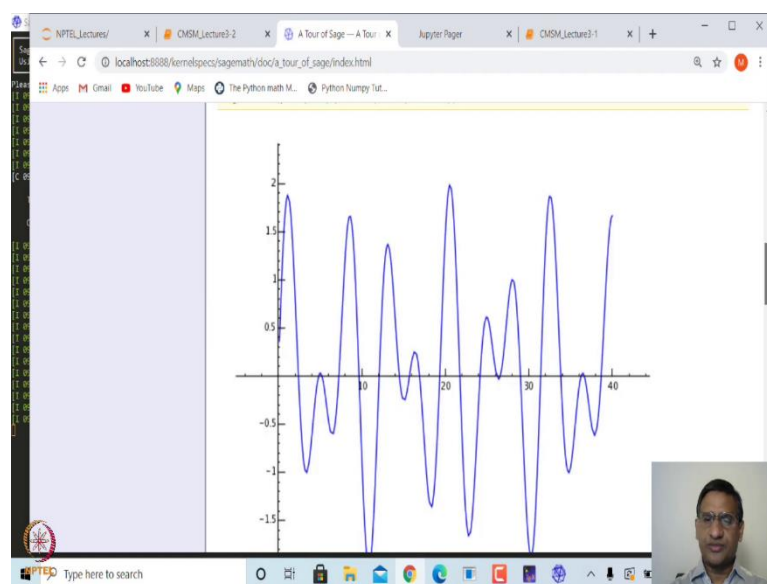
Sage Documentation v9.1
© 2005–2020, The Sage Development Team. Created using Sphinx 1.8.5.

(Refer Slide Time: 21:07)

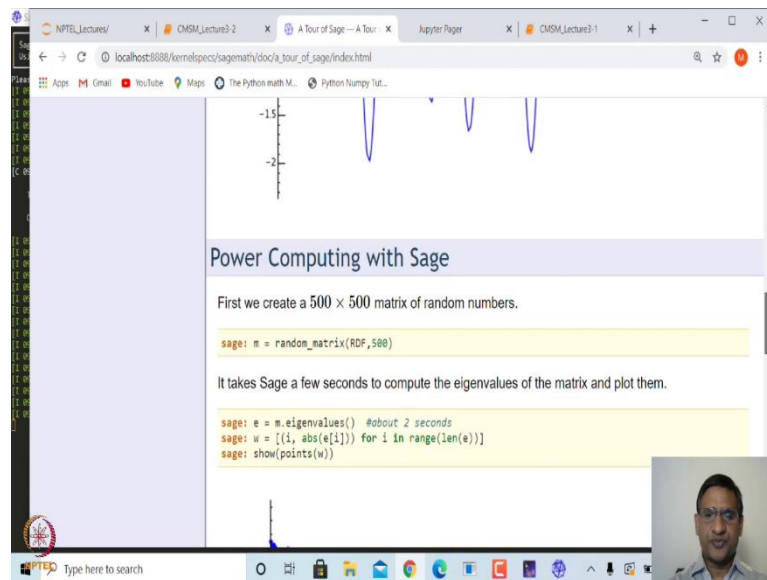


The screenshot shows the Sage Tour V9.1 web interface. The left sidebar contains a 'Table of Contents' with links to 'A Tour of Sage', 'Sage as a Calculator', 'Power Computing with Sage', and 'Accessing Algorithms in Sage'. Below this is a 'This Page' section with a 'Show Source' link and a 'Quick search' box. The main content area is titled 'A Tour of Sage' and includes a paragraph: 'This is a tour of Sage that closely follows the tour of Mathematica that is at the beginning of the Mathematica Book.' Below this is a section titled 'Sage as a Calculator' with a paragraph: 'The Sage command line has a sage: prompt; you do not have to add it. If you use the Sage notebook, then put everything after the sage: prompt in an input cell, and press shift-enter to compute the corresponding output.' Two code blocks are shown: the first contains 'sage: 3 + 5' followed by the output '8', and the second contains 'sage: 57.1 ^ 100' followed by the output '4.60904368661396e175'. A small video feed of a man is visible in the bottom right corner.

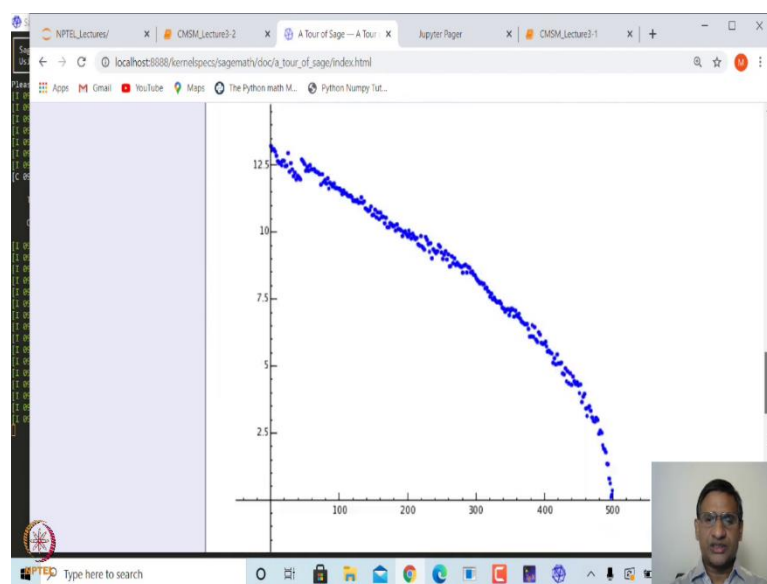
(Refer Slide Time: 21:11)



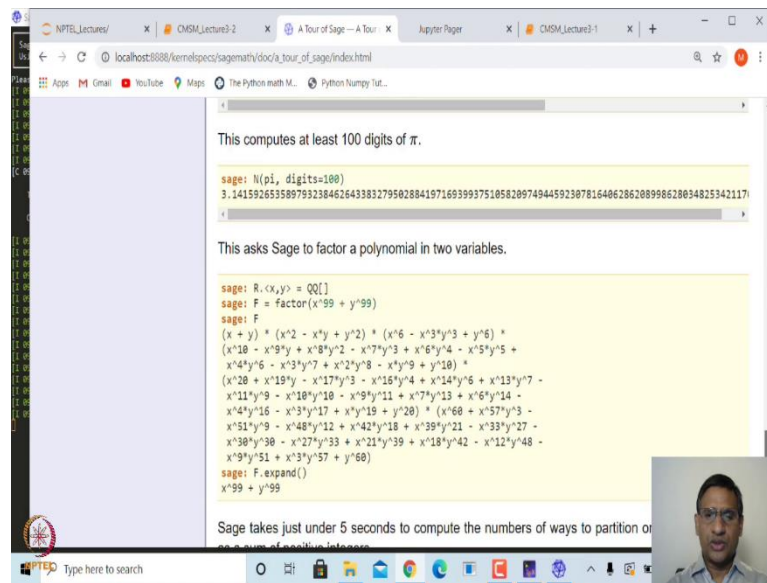
(Refer Slide Time: 21:12)



(Refer Slide Time: 21:12)



(Refer Slide Time: 21:13)



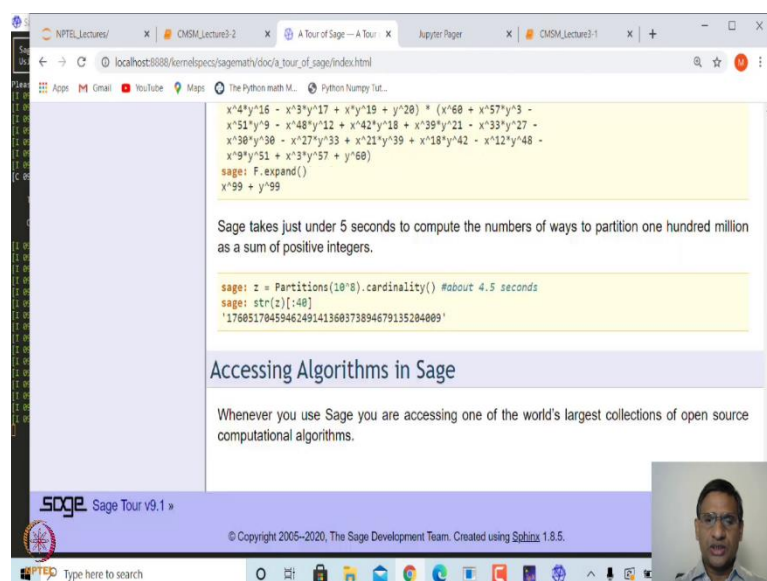
```
sage: N(pi, digits=100)
3.141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117
```

```
sage: R.<x,y> = QQ[]
sage: F = factor(x^99 + y^99)
sage: F
(x + y) * (x^2 - x*y + y^2) * (x^6 - x^3*y^3 + y^6) *
(x^10 - x^9*y + x^8*y^2 - x^7*y^3 + x^6*y^4 - x^5*y^5 +
x^4*y^6 - x^3*y^7 + x^2*y^8 - x*y^9 + y^10) *
(x^20 + x^19*y - x^17*y^3 - x^16*y^4 + x^14*y^6 + x^13*y^7 -
x^11*y^9 - x^10*y^10 - x^9*y^11 + x^7*y^13 + x^6*y^14 -
x^4*y^16 - x^3*y^17 + x^2*y^19 + y^20) * (x^60 + x^57*y^3 -
x^51*y^9 - x^48*y^12 + x^42*y^18 + x^39*y^21 - x^33*y^27 -
x^30*y^30 - x^27*y^33 + x^21*y^39 + x^18*y^42 - x^12*y^48 -
x^9*y^51 + x^3*y^57 + y^60)
sage: F.expand()
x^99 + y^99
```

Sage takes just under 5 seconds to compute the numbers of ways to partition one hundred million as a sum of positive integers.

```
sage: z = Partitions(10^8).cardinality() #about 4.5 seconds
sage: str(z)[-40]
'1760517045946249141360373894679135284009'
```

(Refer Slide Time: 21:13)



```
x^4*y^16 - x^3*y^17 + x^2*y^19 + y^20) * (x^60 + x^57*y^3 -
x^51*y^9 - x^48*y^12 + x^42*y^18 + x^39*y^21 - x^33*y^27 -
x^30*y^30 - x^27*y^33 + x^21*y^39 + x^18*y^42 - x^12*y^48 -
x^9*y^51 + x^3*y^57 + y^60)
sage: F.expand()
x^99 + y^99
```

Sage takes just under 5 seconds to compute the numbers of ways to partition one hundred million as a sum of positive integers.

```
sage: z = Partitions(10^8).cardinality() #about 4.5 seconds
sage: str(z)[-40]
'1760517045946249141360373894679135284009'
```

Accessing Algorithms in Sage

Whenever you use Sage you are accessing one of the world's largest collections of open source computational algorithms.

Sage Sage Tour V9.1 »

© Copyright 2005–2020, The Sage Development Team. Created using Sphinx 1.8.5.

So, for example, there is an installation guide or developers guide and a tour of sage if I click on tour of sage this will give you how to get started with sage with various things including the plotting and other things etc. Or you can go to for example, again help and look at sage thematic tutorial or let us say sage tutorial.

(Refer Slide Time: 21:32)

Sage Sage Tutorial v9.1 [next](#) [index](#)

Table of Contents

- Welcome to the Sage Tutorial! Indices and tables
- Next topic: [Introduction](#)
- This Page: [Show Source](#)
- Quick search: [Go](#)

Welcome to the Sage Tutorial! ¶

Sage is free, open-source math software that supports research and teaching in algebra, geometry, number theory, cryptography, numerical computation, and related areas. Both the Sage development model and the technology in Sage itself are distinguished by an extremely strong emphasis on openness, community, cooperation, and collaboration: we are building the car, not reinventing the wheel. The overall goal of Sage is to create a viable, free, open-source alternative to Maple, Mathematica, Magma, and MATLAB.

This tutorial is the best way to become familiar with Sage in only a few hours. You can read it in HTML or PDF versions, or from the Sage notebook (click [help](#), then click [Tutorial](#) to interactively work through the tutorial from within Sage).

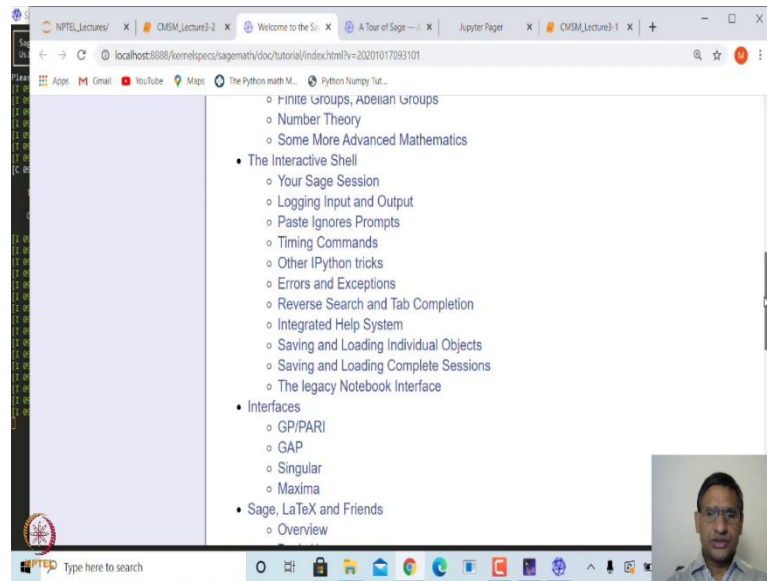
This work is licensed under a [Creative Commons Attribution-Share Alike 3.0 License](#).

- Introduction
 - Installation
 - Ways to Use Sage
 - Longterm Goals for Sage
- A Guided Tour

(Refer Slide Time: 21:34)

- Introduction
 - Installation
 - Ways to Use Sage
 - Longterm Goals for Sage
- A Guided Tour
 - Assignment, Equality, and Arithmetic
 - Getting Help
 - Functions, Indentation, and Counting
 - Basic Algebra and Calculus
 - Plotting
 - Some Common Issues with Functions
 - Basic Rings
 - Linear Algebra
 - Polynomials
 - Parents, Conversion and Coercion
 - Finite Groups, Abelian Groups
 - Number Theory
 - Some More Advanced Mathematics
- The Interactive Shell
 - Your Sage Session
 - Logging Input and Output

(Refer Slide Time: 21:35)

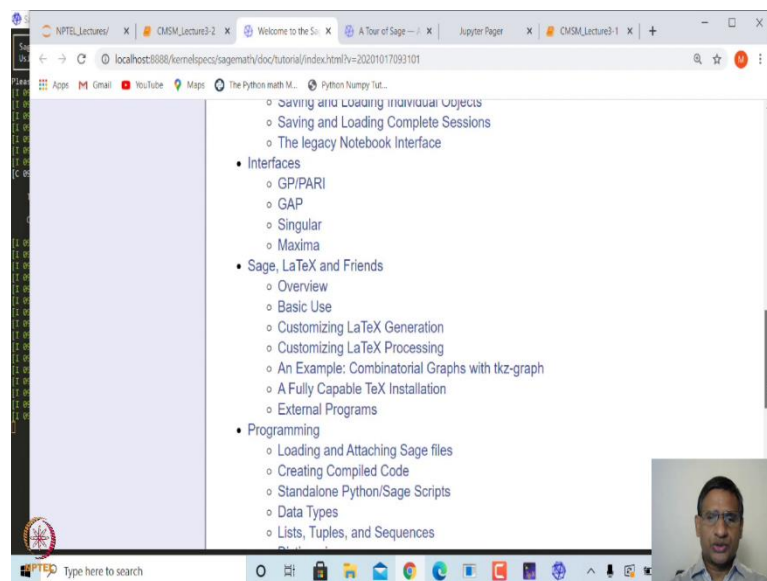


The screenshot shows a web browser window with the URL `localhost:8888/kernel/sagemath/doc/tutorial/index.html?v=20201017093101`. The page content is as follows:

- Finite Groups, Abelian Groups
- Number Theory
- Some More Advanced Mathematics
- The Interactive Shell
 - Your Sage Session
 - Logging Input and Output
 - Paste Ignores Prompts
 - Timing Commands
 - Other IPython tricks
 - Errors and Exceptions
 - Reverse Search and Tab Completion
 - Integrated Help System
 - Saving and Loading Individual Objects
 - Saving and Loading Complete Sessions
 - The legacy Notebook Interface
- Interfaces
 - GP/PARI
 - GAP
 - Singular
 - Maxima
- Sage, LaTeX and Friends
 - Overview

A small video feed of a man is visible in the bottom right corner of the browser window.

(Refer Slide Time: 21:35)

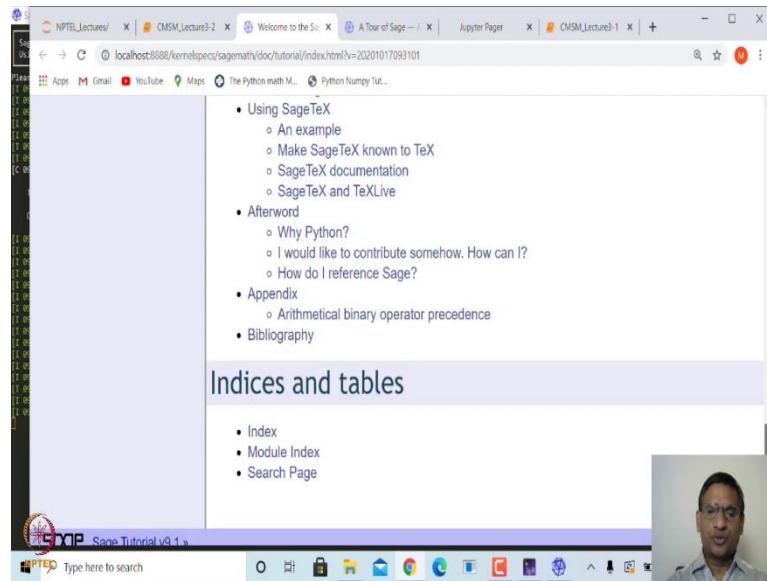


The screenshot shows a web browser window with the URL `localhost:8888/kernel/sagemath/doc/tutorial/index.html?v=20201017093101`. The page content is as follows:

- Saving and Loading Individual Objects
- Saving and Loading Complete Sessions
- The legacy Notebook Interface
- Interfaces
 - GP/PARI
 - GAP
 - Singular
 - Maxima
- Sage, LaTeX and Friends
 - Overview
 - Basic Use
 - Customizing LaTeX Generation
 - Customizing LaTeX Processing
 - An Example: Combinatorial Graphs with tkz-graph
 - A Fully Capable TeX Installation
 - External Programs
- Programming
 - Loading and Attaching Sage files
 - Creating Compiled Code
 - Standalone Python/Sage Scripts
 - Data Types
 - Lists, Tuples, and Sequences

A small video feed of a man is visible in the bottom right corner of the browser window.

(Refer Slide Time: 21:36)



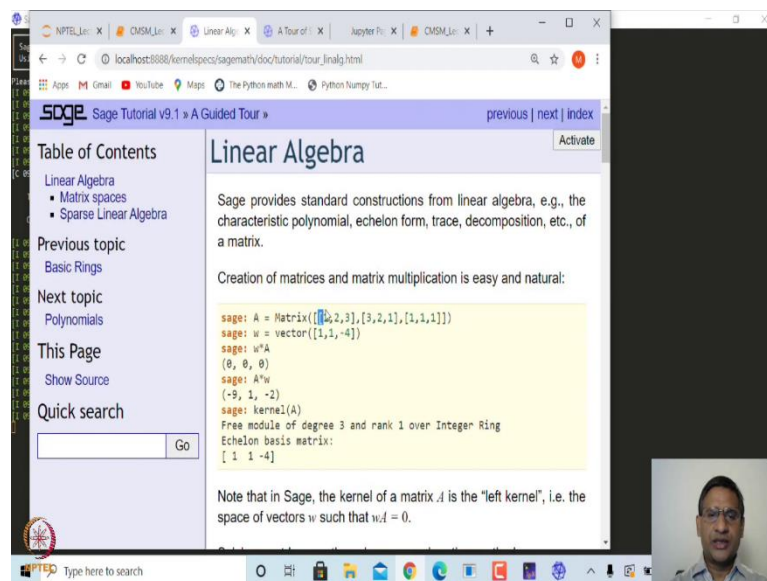
The screenshot shows a web browser displaying the Sage Tutorial V9.1 index page. The page has a light blue header with the Sage logo and the text "Sage Tutorial V9.1". The main content area is white and contains a list of links: "Using SageTeX", "Afterword", "Appendix", and "Bibliography". Below these links is a section titled "Indices and tables" which includes links for "Index", "Module Index", and "Search Page". A small video inset of a man is visible in the bottom right corner.

- Using SageTeX
 - An example
 - Make SageTeX known to TeX
 - SageTeX documentation
 - SageTeX and TeXLive
- Afterword
 - Why Python?
 - I would like to contribute somehow. How can I?
 - How do I reference Sage?
- Appendix
 - Arithmetical binary operator precedence
- Bibliography

Indices and tables

- Index
- Module Index
- Search Page

(Refer Slide Time: 21:51)



The screenshot shows a web browser displaying the Sage Tutorial V9.1 Linear Algebra page. The page has a light blue header with the Sage logo and the text "Sage Tutorial V9.1". The main content area is white and contains the title "Linear Algebra" and a paragraph explaining that Sage provides standard constructions from linear algebra. Below this is a code block showing Sage commands for creating a matrix and a vector, and calculating the kernel of a matrix. A small video inset of a man is visible in the bottom right corner.

Linear Algebra

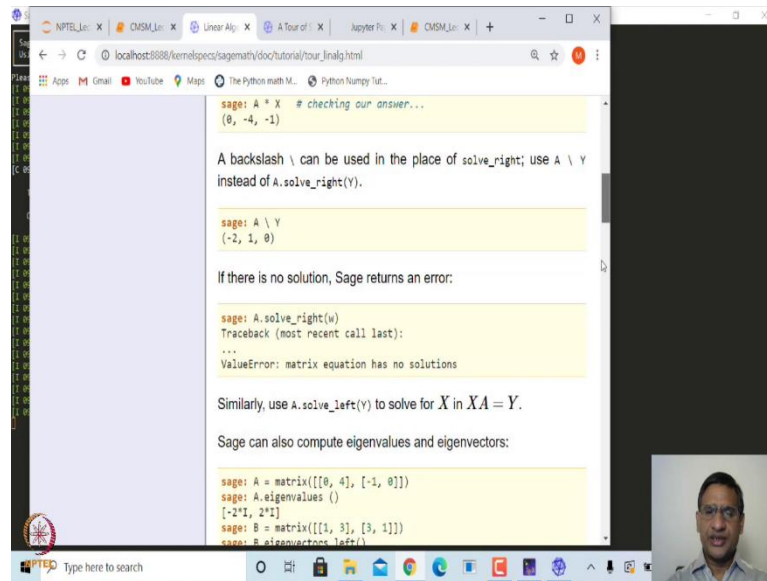
Sage provides standard constructions from linear algebra, e.g., the characteristic polynomial, echelon form, trace, decomposition, etc., of a matrix.

Creation of matrices and matrix multiplication is easy and natural:

```
sage: A = Matrix([[4,2,3],[3,2,1],[1,1,1]])
sage: w = vector([1,1,-4])
sage: w*A
(0, 0, 0)
sage: A*w
(-9, 1, -2)
sage: kernel(A)
Free module of degree 3 and rank 1 over Integer Ring
Echelon basis matrix:
[ 1 1 -4]
```

Note that in Sage, the kernel of a matrix A is the "left kernel", i.e. the space of vectors w such that $wA = 0$.

(Refer Slide Time: 21:55)



The screenshot shows a SageMath Jupyter Notebook interface. The browser address bar indicates the URL is `localhost:8888/kernelspecs/sagemath/doc/tutorial/hour_1inalg.html`. The notebook contains the following code and text:

```
sage: A * X # checking our answer...
(0, -4, -1)
```

A backslash `\` can be used in the place of `solve_right`; use `A \ Y` instead of `A.solve_right(Y)`.

```
sage: A \ Y
(-2, 1, 0)
```

If there is no solution, Sage returns an error:

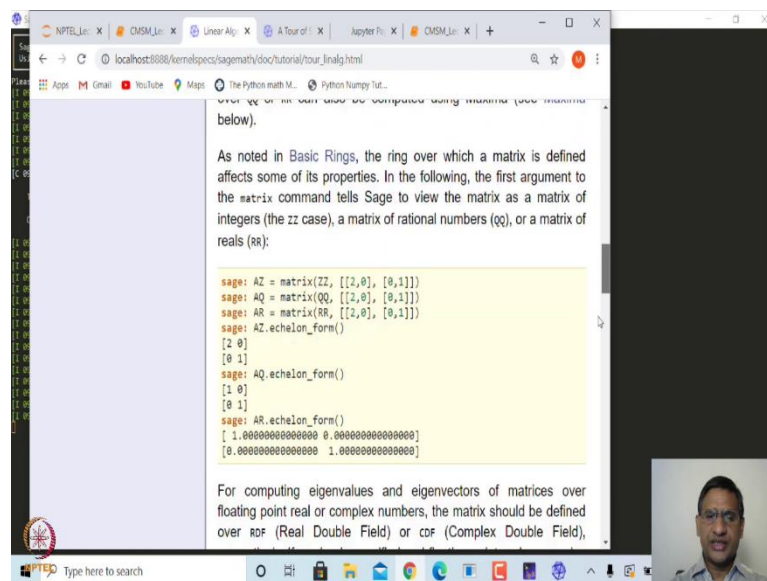
```
sage: A.solve_right(w)
Traceback (most recent call last):
...
ValueError: matrix equation has no solutions
```

Similarly, use `A.solve_left(Y)` to solve for X in $XA = Y$.

Sage can also compute eigenvalues and eigenvectors:

```
sage: A = matrix([[0, 4], [-1, 0]])
sage: A.eigenvalues()
[-2*I, 2*I]
sage: B = matrix([[1, 3], [3, 1]])
sage: B.eigenvectors_left()
```

(Refer Slide Time: 21:55)



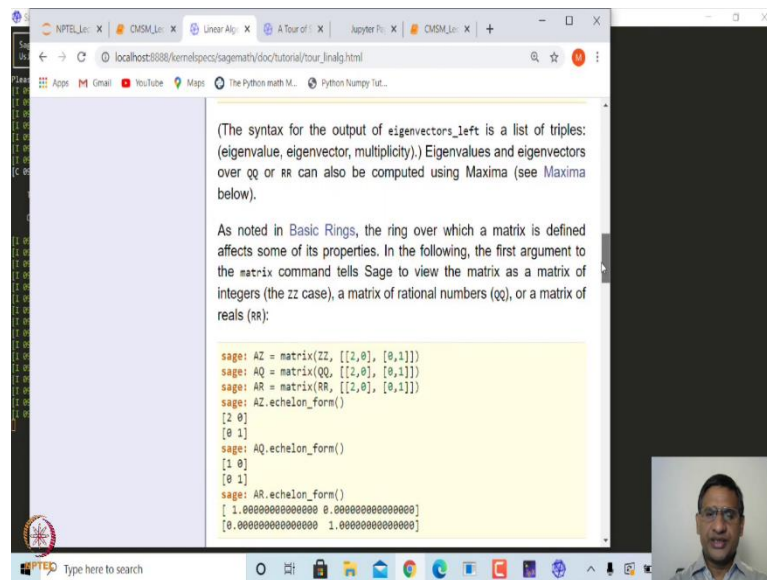
The screenshot shows a SageMath Jupyter Notebook interface. The browser address bar indicates the URL is `localhost:8888/kernelspecs/sagemath/doc/tutorial/hour_1inalg.html`. The notebook contains the following code and text:

As noted in Basic Rings, the ring over which a matrix is defined affects some of its properties. In the following, the first argument to the `matrix` command tells Sage to view the matrix as a matrix of integers (the `ZZ` case), a matrix of rational numbers (`QQ`), or a matrix of reals (`RR`):

```
sage: AZ = matrix(ZZ, [[2,0], [0,1]])
sage: AQ = matrix(QQ, [[2,0], [0,1]])
sage: AR = matrix(RR, [[2,0], [0,1]])
sage: AZ.echelon_form()
[2 0]
[0 1]
sage: AQ.echelon_form()
[1 0]
[0 1]
sage: AR.echelon_form()
[ 1.0000000000000000 0.0000000000000000]
[0.0000000000000000 1.0000000000000000]
```

For computing eigenvalues and eigenvectors of matrices over floating point real or complex numbers, the matrix should be defined over `ROF` (Real Double Field) or `COF` (Complex Double Field).

(Refer Slide Time: 21:55)



The screenshot shows a SageMath Jupyter Notebook interface. The browser address bar indicates the URL is `localhost:8080/kernelspecs/sagemath/doc/tutorial/hour_1inalg.html`. The notebook contains the following text and code:

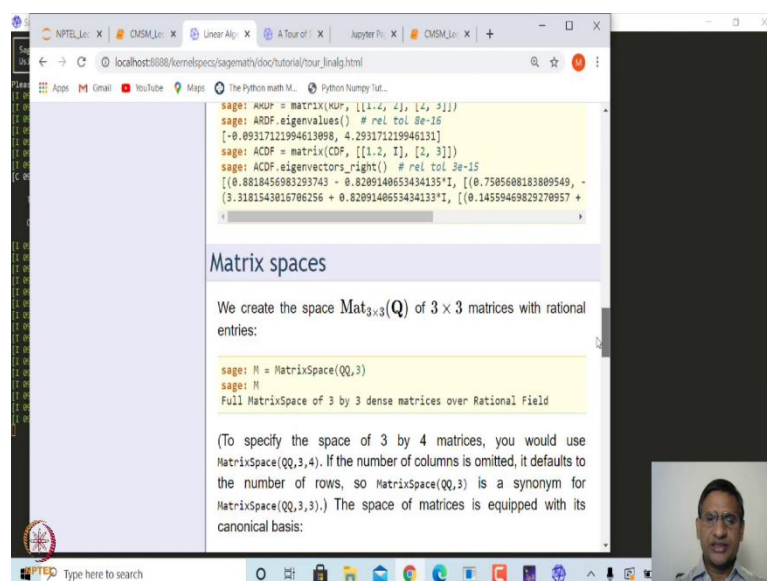
(The syntax for the output of `eigenvectors_left` is a list of triples: (eigenvalue, eigenvector, multiplicity).) Eigenvalues and eigenvectors over `QQ` or `RR` can also be computed using Maxima (see Maxima below).

As noted in Basic Rings, the ring over which a matrix is defined affects some of its properties. In the following, the first argument to the `matrix` command tells Sage to view the matrix as a matrix of integers (the `ZZ` case), a matrix of rational numbers (`QQ`), or a matrix of reals (`RR`):

```
sage: AZ = matrix(ZZ, [[2,0], [0,1]])
sage: AQ = matrix(QQ, [[2,0], [0,1]])
sage: AR = matrix(RR, [[2,0], [0,1]])
sage: AZ.echelon_form()
[2 0]
[0 1]
sage: AQ.echelon_form()
[1 0]
[0 1]
sage: AR.echelon_form()
[ 1.0000000000000000 0.0000000000000000]
[ 0.0000000000000000 1.0000000000000000]
```

A small video inset in the bottom right corner shows a man speaking.

(Refer Slide Time: 21:55)



The screenshot shows a SageMath Jupyter Notebook interface. The browser address bar indicates the URL is `localhost:8080/kernelspecs/sagemath/doc/tutorial/hour_1inalg.html`. The notebook contains the following text and code:

```
sage: AKUR = matrix(KUR, [[1,2,4], [4,3]])
sage: ARDF.eigenvalues() # rel tol 8e-16
[-0.09317121994613098, 4.293171219946131]
sage: ACD = matrix(CDF, [[1,2,1], [2,3]])
sage: ACD.eigenvectors_right() # rel tol 3e-15
[(0.8818456983293743 - 0.8209140653434135*I, [(0.7505608183809549, -
(3.3181543816706256 + 0.8209140653434133*I, [(0.14559469829270957 +
```

Matrix spaces

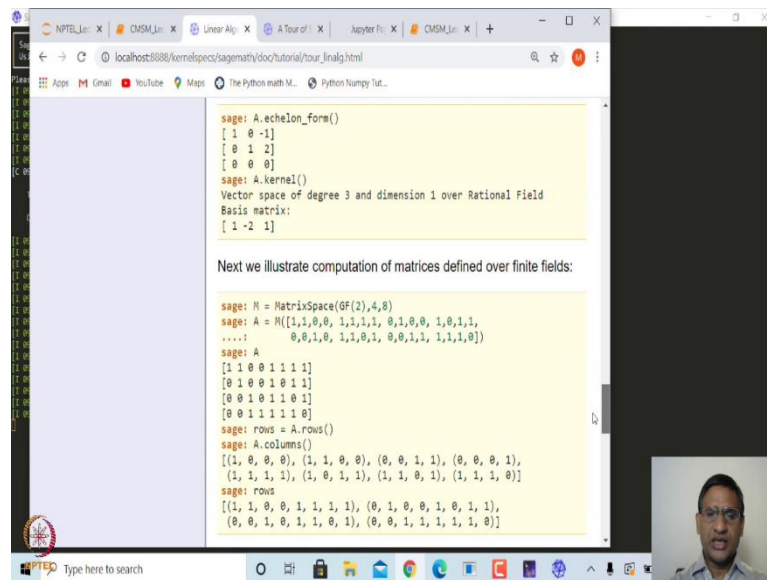
We create the space $\text{Mat}_{3 \times 3}(\mathbb{Q})$ of 3×3 matrices with rational entries:

```
sage: M = MatrixSpace(QQ,3)
sage: M
Full MatrixSpace of 3 by 3 dense matrices over Rational Field
```

(To specify the space of 3 by 4 matrices, you would use `MatrixSpace(QQ,3,4)`. If the number of columns is omitted, it defaults to the number of rows, so `MatrixSpace(QQ,3)` is a synonym for `MatrixSpace(QQ,3,3)`.) The space of matrices is equipped with its canonical basis:

A small video inset in the bottom right corner shows a man speaking.

(Refer Slide Time: 21:55)



The screenshot shows a SageMath Jupyter Notebook with the following content:

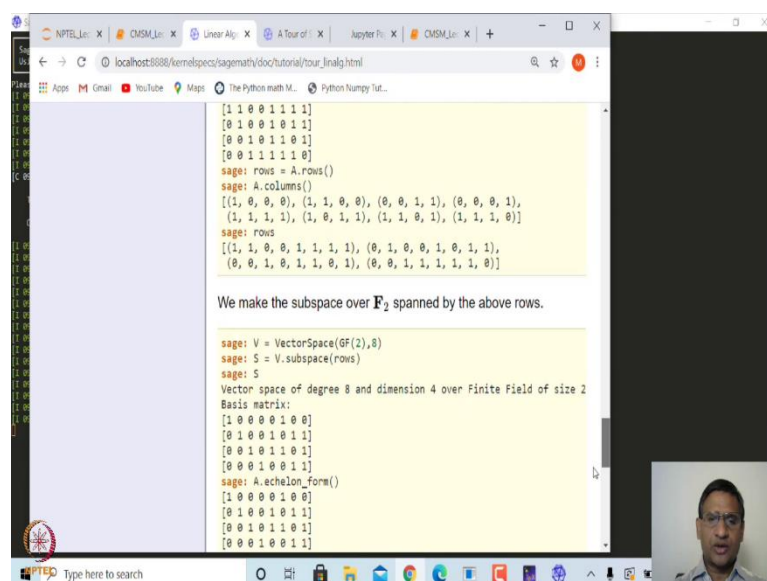
```
sage: A.echelon_form()
[1 0 -1]
[0 1 2]
[0 0 0]
sage: A.kernel()
Vector space of degree 3 and dimension 1 over Rational Field
Basis matrix:
[1 -2 1]
```

Next we illustrate computation of matrices defined over finite fields:

```
sage: M = MatrixSpace(GF(2),4,8)
sage: A = M([[1,1,0,0, 1,1,1,1, 0,1,0,0, 1,0,1,1,
.....:      0,0,1,0, 1,1,0,1, 0,0,1,1, 1,1,1,0]])
sage: A
[1 1 0 0 1 1 1 1]
[0 1 0 0 1 0 1 1]
[0 0 1 0 1 1 0 1]
[0 0 1 1 1 1 1 0]
sage: rows = A.rows()
sage: A.columns()
[(1, 0, 0, 0), (1, 1, 0, 0), (0, 0, 1, 1), (0, 0, 0, 1),
(1, 1, 1, 1), (1, 0, 1, 1), (1, 1, 0, 1), (1, 1, 1, 0)]
sage: rows
[(1, 1, 0, 0, 1, 1, 1, 1), (0, 1, 0, 0, 1, 0, 1, 1),
(0, 0, 1, 0, 1, 1, 0, 1), (0, 0, 1, 1, 1, 1, 1, 0)]
```

A video feed of a presenter is visible in the bottom right corner.

(Refer Slide Time: 21:56)



The screenshot shows a SageMath Jupyter Notebook with the following content:

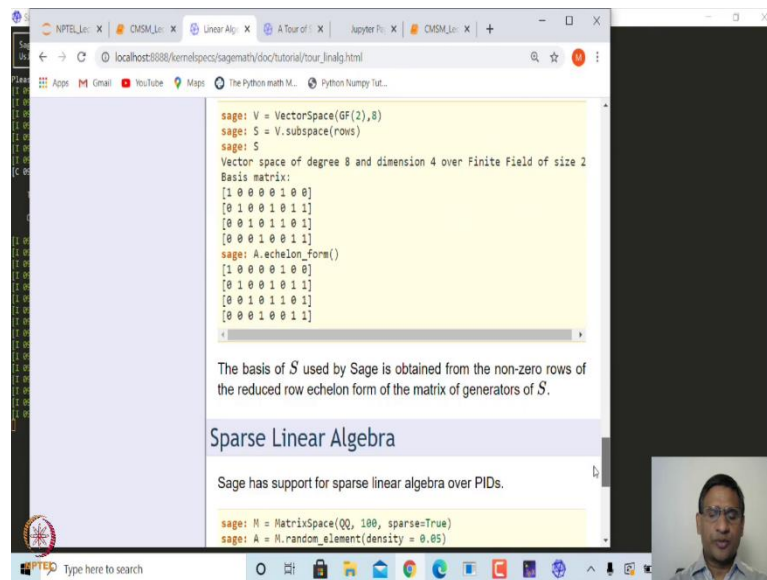
```
[1 1 0 0 1 1 1 1]
[0 1 0 0 1 0 1 1]
[0 0 1 0 1 1 0 1]
[0 0 1 1 1 1 1 0]
sage: rows = A.rows()
sage: A.columns()
[(1, 0, 0, 0), (1, 1, 0, 0), (0, 0, 1, 1), (0, 0, 0, 1),
(1, 1, 1, 1), (1, 0, 1, 1), (1, 1, 0, 1), (1, 1, 1, 0)]
sage: rows
[(1, 1, 0, 0, 1, 1, 1, 1), (0, 1, 0, 0, 1, 0, 1, 1),
(0, 0, 1, 0, 1, 1, 0, 1), (0, 0, 1, 1, 1, 1, 1, 0)]
```

We make the subspace over \mathbb{F}_2 spanned by the above rows.

```
sage: V = VectorSpace(GF(2),8)
sage: S = V.subspace(rows)
sage: S
Vector space of degree 8 and dimension 4 over Finite Field of size 2
Basis matrix:
[1 0 0 0 1 0 0 0]
[0 1 0 0 1 0 1 1]
[0 0 1 0 1 1 0 1]
[0 0 1 0 0 1 1]
sage: A.echelon_form()
[1 0 0 0 1 0 0 0]
[0 1 0 0 1 0 1 1]
[0 0 1 0 1 1 0 1]
[0 0 0 0 0 0 0 0]
```

A video feed of a presenter is visible in the bottom right corner.

(Refer Slide Time: 21:56)



The screenshot shows the SageMath documentation page for Linear Algebra. The page is titled "Linear Algebra" and contains several sections. The first section, "Vector Space", shows the creation of a vector space V over the finite field $\text{GF}(2)$ of degree 8 and dimension 4. It then shows the creation of a subspace S and the calculation of its basis matrix. The basis matrix is displayed as a 4x8 matrix of 0s and 1s. The second section, "Sparse Linear Algebra", states that Sage has support for sparse linear algebra over PIDs. It shows the creation of a matrix space M and a random element A with a density of 0.05.

```
sage: V = VectorSpace(GF(2),8)
sage: S = V.subspace(rows)
sage: S
Vector space of degree 8 and dimension 4 over Finite Field of size 2
Basis matrix:
[1 0 0 0 1 0 0]
[0 1 0 0 1 0 1]
[0 0 1 0 1 1 0]
[0 0 0 1 0 0 1]
sage: A.echelon_form()
[1 0 0 0 1 0 0]
[0 1 0 0 1 0 1]
[0 0 1 0 1 1 0]
[0 0 0 1 0 0 1]
```

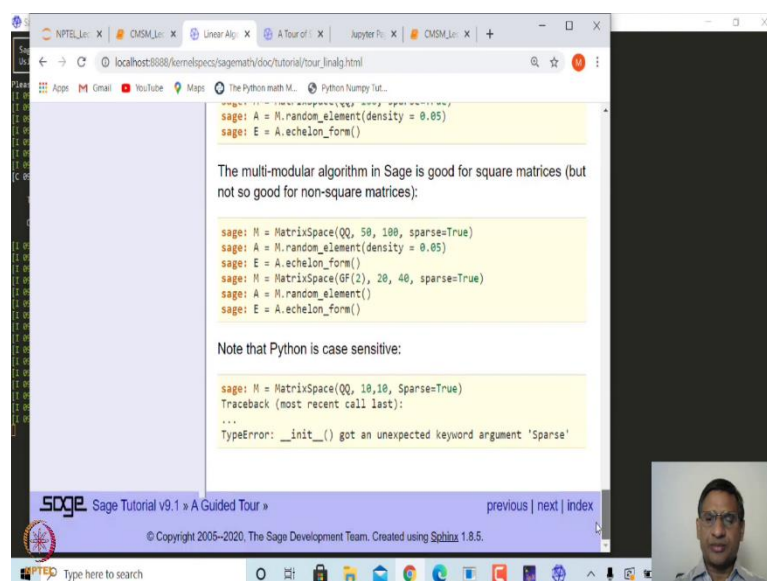
The basis of S used by Sage is obtained from the non-zero rows of the reduced row echelon form of the matrix of generators of S .

Sparse Linear Algebra

Sage has support for sparse linear algebra over PIDs.

```
sage: M = MatrixSpace(QQ, 100, sparse=True)
sage: A = M.random_element(density = 0.05)
```

(Refer Slide Time: 21:57)



The screenshot shows the SageMath documentation page for Sparse Linear Algebra. The page is titled "Sparse Linear Algebra" and contains several sections. The first section, "Multi-modular algorithm", states that the multi-modular algorithm in Sage is good for square matrices (but not so good for non-square matrices). It shows the creation of a matrix space M and a random element A with a density of 0.05. The second section, "Note that Python is case sensitive:", shows a traceback error for a typo in the `sparse=True` argument.

```
sage: M = MatrixSpace(QQ, 50, 100, sparse=True)
sage: A = M.random_element(density = 0.05)
sage: E = A.echelon_form()
sage: M = MatrixSpace(GF(2), 20, 40, sparse=True)
sage: A = M.random_element()
sage: E = A.echelon_form()
```

The multi-modular algorithm in Sage is good for square matrices (but not so good for non-square matrices):

```
sage: M = MatrixSpace(QQ, 10, 10, Sparse=True)
Traceback (most recent call last):
...
TypeError: __init__() got an unexpected keyword argument 'Sparse'
```

Note that Python is case sensitive:

```
sage: M = MatrixSpace(QQ, 10, 10, Sparse=True)
Traceback (most recent call last):
...
TypeError: __init__() got an unexpected keyword argument 'Sparse'
```

[previous](#) | [next](#) | [index](#)

© Copyright 2005–2020, The Sage Development Team. Created using Sphinx 1.8.5.

And then you can just look at each one of these for example, if I say click on something to do with let us say click on linear algebra it will tell you some of the basic things in linear algebra and you can copy paste any of this command.

(Refer Slide Time: 22:08)

NPTEL_Lecture: X CMSM_Lecture: X Welcome to th: X Linear Algebra: X A Tour of Sage: X Jupyter Pager: X CMSM_Lecture: X

localhost:8888/kernelspecs/sagemath/doc/reference/index.html?v=20201017093101

Apps Gmail YouTube Maps The Python math ML... Python Numpy Tut...

Sage Reference v9.1 modules | index

Table of Contents

- Welcome to the Sage Reference Manual
- User Interface
- Graphics
- Mathematics
- Parents and Categories
- Basic Rings and Fields
- Linear Algebra
- Calculus and Analysis
- Probability and Statistics
- Mathematical Structures
- Discrete Mathematics
- Geometry and Topology
- Number Fields, Function Fields, and Valuations

Welcome to the Sage Reference Manual

This manual contains documentation for (almost) all of Sage's features, each illustrated with examples that are systematically tested with each release. A thematic index is available below.

User Interface

- Command Line Interface (REPL)
- For the Jupyter notebook interface, visit its documentation.
- For the legacy notebook interface, which is no longer actively maintained, visit the source repository.

Graphics

- 2D Graphics
- 3D Graphics

Mathematics

(Refer Slide Time: 22:11)

NPTEL_Lecture: X CMSM_Lecture: X Welcome to th: X Linear Algebra: X A Tour of Sage: X Jupyter Pager: X CMSM_Lecture: X

localhost:8888/kernelspecs/sagemath/doc/reference/index.html?v=20201017093101

Apps Gmail YouTube Maps The Python math ML... Python Numpy Tut...

Sage Reference v9.1 modules | index

Mathematics

Parents and Categories

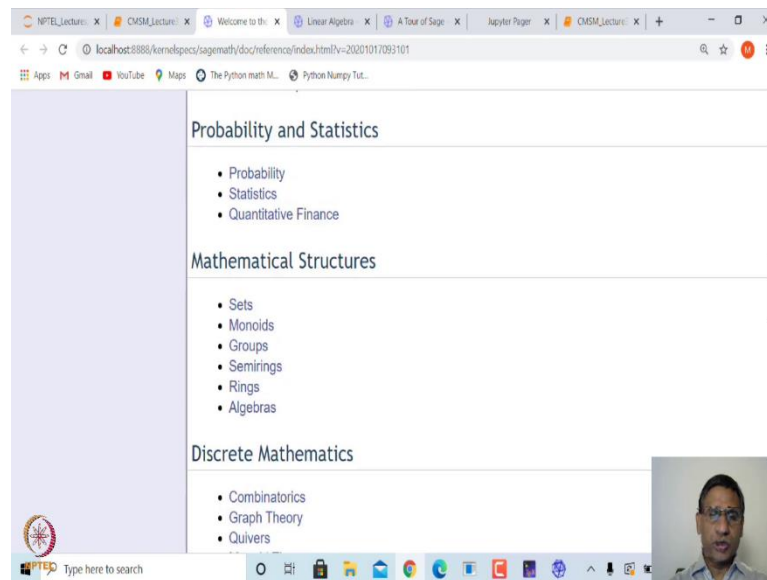
- Parents and Elements
- Coercion
- Categories

Basic Rings and Fields

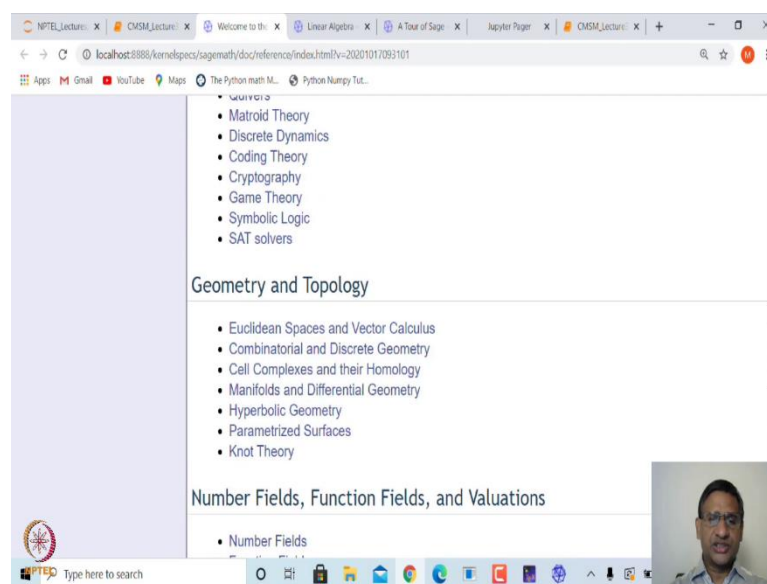
- Integers and Rational Numbers
- Real and Complex Numbers
- Polynomials
- Power Series and Laurent Series
- Finite Rings and Fields
- p-Adic Numbers
- Quaternion Algebras

Linear Algebra

(Refer Slide Time: 22:12)



(Refer Slide Time: 22:12)



If you go to help and then go to sage reference then it will tell you the reference manual for various things.

(Refer Slide Time: 22:18)

NPTEL Lectures: X CMSM_Lecture: X Coding Theory X Linear Algebra X A Tour of Sage X Jupyter Pager X CMSM_Lecture: X

localhost:8888/kernelspecs/sagemath/doc/reference/coding/index.html

Sage Reference Manual » Sage 9.1 Reference Manual: Coding Theory » next | modules | index

Coding Theory

Coding theory is the mathematical theory for algebraic and combinatorial codes used for forward error correction in communications theory. Sage provides an extensive library of objects and algorithms in coding theory.

Basic objects in coding theory are channels, codes, linear codes, encoders, and decoders. The following modules provide the base classes defining them.

- Base class for Codes
- Generic structures for linear codes of any metric
- Linear codes
- Channels
- Encoders
- Decoders

Catalogs for available constructions of the basic objects and for bounds on the parameters of codes are provided.

Index of code constructions

Table of Contents

- Coding Theory
 - Families of Codes
 - Derived Code Constructions
 - Decoding
 - Automorphism Groups of Linear Codes
 - Bounds for Parameters of Linear Codes
 - Databases for Coding Theory
 - Miscellaneous Modules
- Indices and Tables

Next topic


Base class for Codes

This Page

Show Source

Search

Type here to search



(Refer Slide Time: 22:21)

NPTEL Lectures: X CMSM_Lecture: X Coding Theory X Linear Algebra X A Tour of Sage X Jupyter Pager X CMSM_Lecture: X

localhost:8888/kernelspecs/sagemath/doc/reference/coding/index.html

Index of code constructions

- Index of decoders
- Index of encoders
- Index of bounds on the parameters of codes

Families of Codes


Famous families of codes, listed below, are represented in Sage by their own classes. For some of them, implementations of special decoding algorithms or computations for structural invariants are available.

- Parity-check code
- Hamming codes
- Cyclic code
- BCH code
- Golay code
- Reed-Muller code
- Reed-Solomon codes and Generalized Reed-Solomon codes
- Goppa code

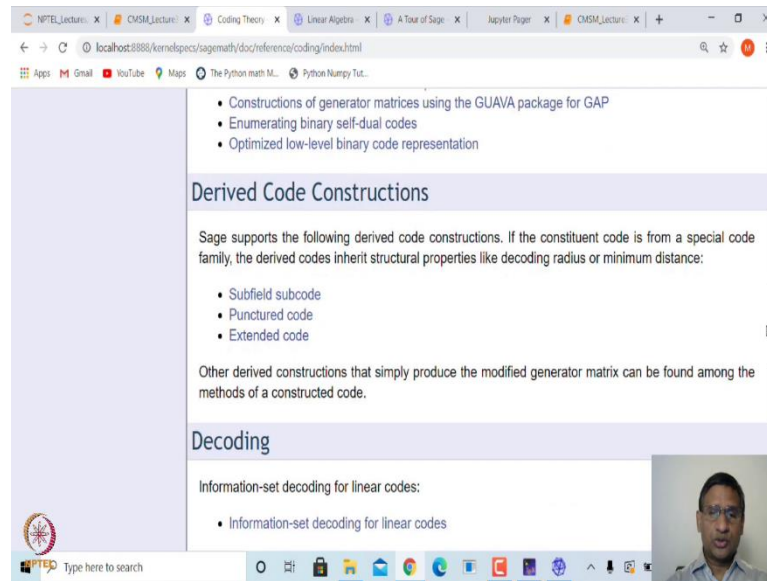
In contrast, for some code families Sage can only construct their generator matrix if you have prior knowledge on them:

Search

Type here to search



(Refer Slide Time: 22:21)



• Constructions of generator matrices using the GUAVA package for GAP

- Enumerating binary self-dual codes
- Optimized low-level binary code representation

Derived Code Constructions

Sage supports the following derived code constructions. If the constituent code is from a special code family, the derived codes inherit structural properties like decoding radius or minimum distance:

- Subfield subcode
- Punctured code
- Extended code

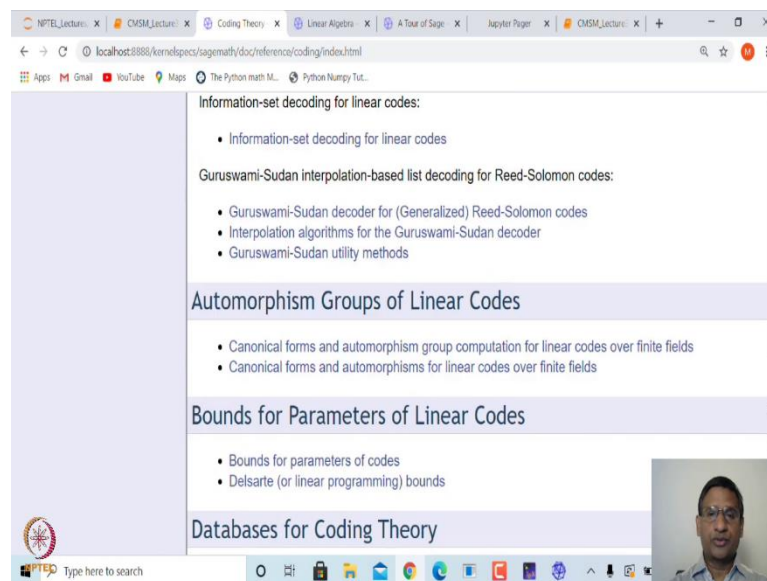
Other derived constructions that simply produce the modified generator matrix can be found among the methods of a constructed code.

Decoding

Information-set decoding for linear codes:

- Information-set decoding for linear codes

(Refer Slide Time: 22:22)



• Information-set decoding for linear codes

Guruswami-Sudan interpolation-based list decoding for Reed-Solomon codes:

- Guruswami-Sudan decoder for (Generalized) Reed-Solomon codes
- Interpolation algorithms for the Guruswami-Sudan decoder
- Guruswami-Sudan utility methods

Automorphism Groups of Linear Codes

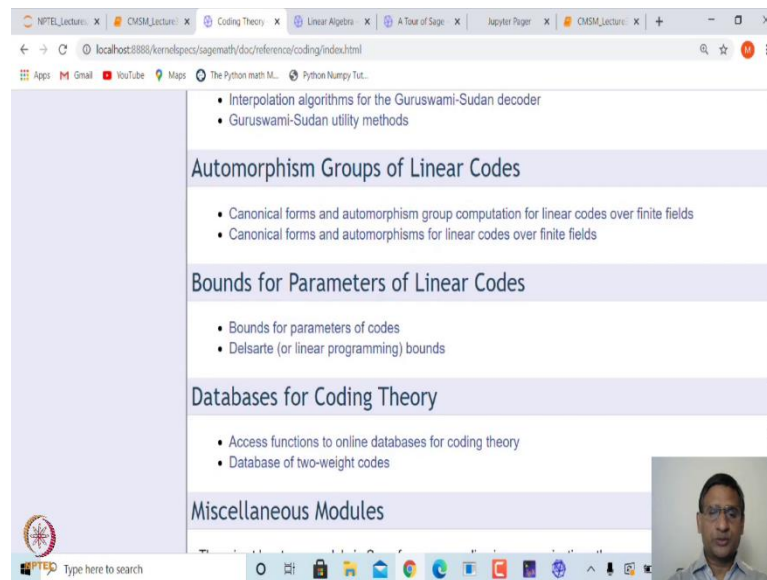
- Canonical forms and automorphism group computation for linear codes over finite fields
- Canonical forms and automorphisms for linear codes over finite fields

Bounds for Parameters of Linear Codes

- Bounds for parameters of codes
- Delsarte (or linear programming) bounds

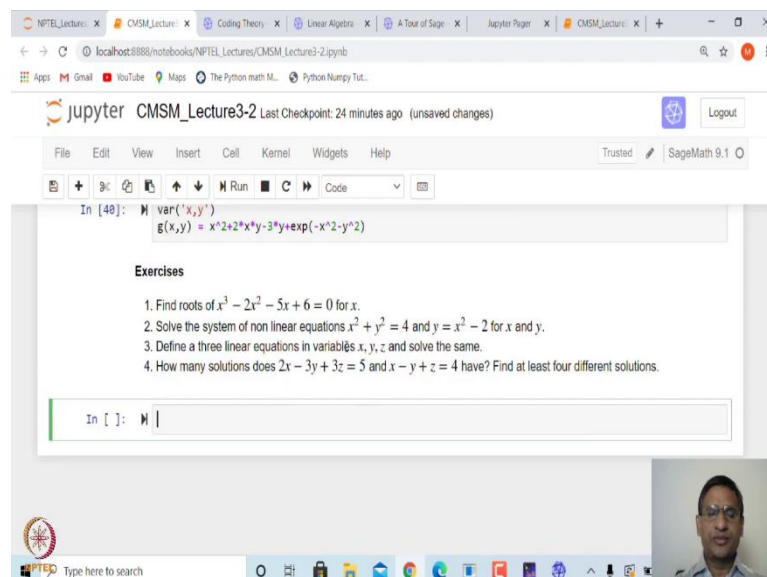
Databases for Coding Theory

(Refer Slide Time: 22:22)



So, for example, even if I say reference manual for coding theory it will tell you many things about coding theory. So, you can explore whatever is of your interest and definitely when we start learning each of these concepts we can explore this further. So, at the end let me also leave you with some exercises. So, let me delete all these cells I think the previous one also got deleted.

(Refer Slide Time: 22:55)



So, let me go to this and undo the deleted cell this is what we explored. So, let me convert this into markdown. So, they are very simple exercises like for example, find the roots of

this cubic with respect to x , solve system of non-linear equations this, define three linear equations in three variables x, y, z and then solve them and find out how many solutions does these two equations in three variables have and find at least four different solutions. So, these are few very simple exercises.

So, we will look at more on SageMath in the next lecture; especially on how to plot graphs of various 2 dimensional and 3 dimensional objects that we will look at next.

Thank you very much.