

Introduction to Computational Fluid Dynamics
Prof. M. Ramakrishna
Department of Aerospace Engineering
Indian Institute of Technology - Madras

Lecture – 13
Demo - representation error, Laplace equation

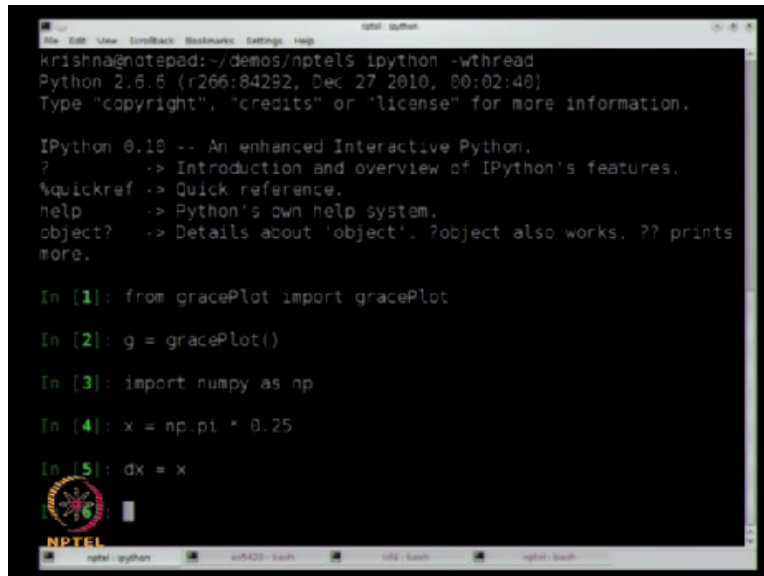
Okay, so we will look at a few demos that I have been planning to do for some time, right. So I think quite a few of you sent me the error and evaluation of the derivative of sine x , okay. So, I will repeat part of that process, most of the codes that you write will be either in C, C++, or Fortran or whatever but because this is a demo like last time, I am going to use the scripting language, Python, right and again to do the plots, I will use Grace plot.

And we will also look at the second demo which is solution to Laplace's equation, right. So I am not going to do anything fancy. I will just do the initial simple, right just to show you how the thing goes and again as I said most of you may have already tried to write these programs. So the objective is 2 fold. So I am not going to show you a canned, precanned package. So I will actually write it so that you can see that I make the same kind of mistakes that you make, right.

There is nothing, nothing unusual about it. All of us are error prone. The second thing of course is maybe you will get an idea so how those process actually works, right and along the way, we look at the results and possibly discuss some of the results. If you have any questions, right, you feel free to stop me, right. So we will just explore what it is that we have. So the first demo was going to be basically how well we are able to evaluate the derivative.

I will write for the first order, right, one-sided difference, say a forward difference. We will do a forward difference and look at what happens and maybe we can also do a central difference just for fun. After that I do have a canned package, right because I do not want to do the third derivative and fourth derivative. It is that you do not get anything out of doing that, right. Is that okay, right. So I am going to use my particular favourite version of Python chordal IPython since I am also going to do Laplace's equation.

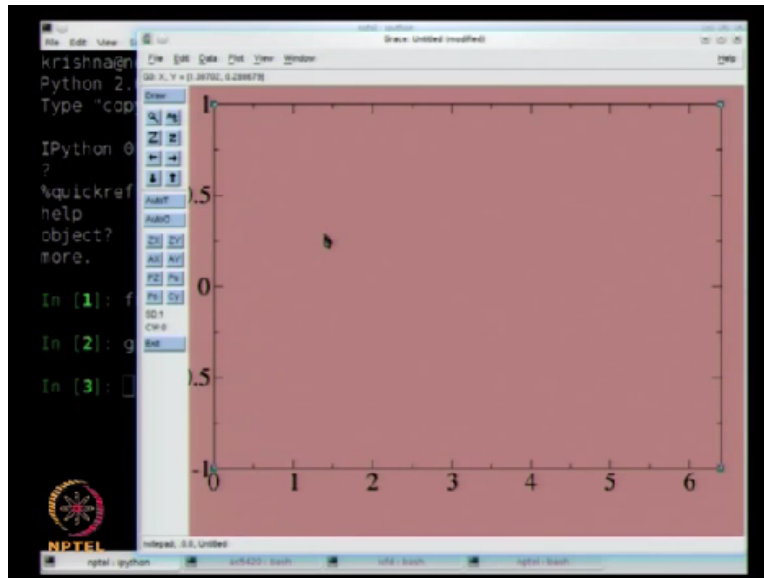
(Refer Slide Time: 02:21)

A screenshot of a terminal window titled 'nptel: ipython'. The window shows the command 'krishna@notepad: ~/demos/nptel\$ ipython -wthread' and the output 'Python 2.6.6 (r266:84292, Dec 27 2010, 00:02:40)'. Below this, the IPython 0.10 startup message is displayed, including a list of shortcuts: '?', '%quickref', 'help', and 'object?'. The user then enters five lines of code in the IPython prompt: 'In [1]: from gracePlot import gracePlot', 'In [2]: g = gracePlot()', 'In [3]: import numpy as np', 'In [4]: x = np.pi * 0.25', and 'In [5]: dx = x'. The terminal window has a standard Linux-style title bar and a taskbar at the bottom showing several open windows.

And I am going to do the visualisation using a package called Mayavi which was originally developed in this institute. So obviously I am going to use that package. It is part of enthought sweep of programs right now. So let me just get started, right and along the way, yes, I mean I know that some of you may not be familiar with Python and so on. Where its important I will try to show you what happening, right, what this language is all about.

But if you want to find out, I would suggest there are tutorials out there, you can go learn the language by yourself. It is not that difficult, okay. So once you know of one programming language, you should be able to handle it, right, okay. So the first thing is to just set up Grace plot so that we are ready to plot, okay. Since I know I want to plot, so I get my plotting package ready and of course it is going to be an intense white color that may not come out that well on the screen.

(Refer Slide Time: 03:26)

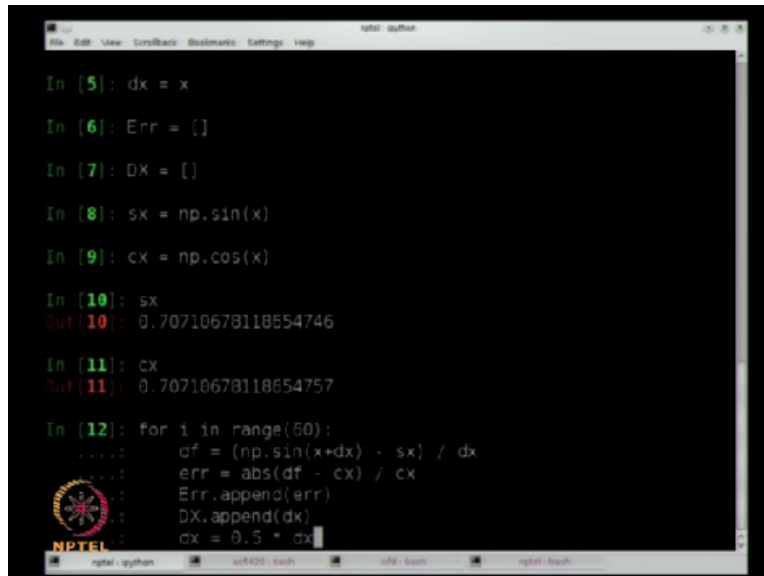


So I will quickly change it to, quickly change it to a colour that will, quickly change it to a color that is more amenable to the videotaping and so on. So I have my plot screen ready. All I need to do now is to get something to plot, right, okay. So I am going to use again the package Numpy which is part of enthought scientific sweep. So from, maybe I will just import, I will do it, this is compared to what I did last time.

This is really the proper way to do. So I import Numpy as np. So it will be called np instead of Numpy, right, okay. Okay? The point at which I am going to take the derivative I think in the assignment I suggested $\pi/4$. So I will say $x=$, so in np because having called it np, I may make a mistake, so keep your, keep safe, stay alert so that I do not make a mistake, okay. I think $\pi/4$ is what I started, $1/4$ of $\pi/4$.

I say $\pi/4$ but I always multiply by 0.25, right, even if I am using the scripting language and the initial dx, we can choose the same as, we can choose the same as x and so it is very large. Initial dx is quite large, okay, right. $\pi/4$, 4 is almost 3 quarters, initial dx is quite large. Now I want to be plotting the stuff. So what we will do is, we will have an error and we will have the dx that we are going to generate.

(Refer Slide Time: 05:11)



```
In [5]: dx = x
In [6]: Err = []
In [7]: DX = []
In [8]: sx = np.sin(x)
In [9]: cx = np.cos(x)
In [10]: sx
Out[10]: 0.70710678118554746
In [11]: cx
Out[11]: 0.70710678118554757
In [12]: for i in range(60):
...:     df = (np.sin(x+dx) - sx) / dx
...:     err = abs(df - cx) / cx
...:     Err.append(err)
...:     DX.append(dx)
...:     cx = 0.5 * dx
```

So I am going to have a list that I will create, an empty list that I will create which is called error and I will create DX, capital DX which will be the list of DX's that we are going to use, right. The error is set now. Because I am a bit lazy and it is a good programming practice, I will predefine sine x because I know that I am going to constantly be using sine of x. I predefine sine x as, right, so sx is sine x and its derivative which I will just call cx.

It is np.cos of x, fine. So we have now defined sine x and cosine of x, same value because it is $\pi/4$, right. So we are now set. So what we will do is, how many, how many different dx value should we take? About 60, 60, you have taken about 60, okay, I will do 60 or i in range 60, okay. So first we will define the derivative, df. So that is going to be, I know I am going to divide by dx.

So I open bracket close bracket. Every time you open bracket, should get into the practice of closing the bracket. Sine of, np. sine of $x+dx$, having said that I open and then close the bracket, sx/dx , that is the derivative, that is our forward difference and there is an error in this derivative. What is this error? So I am going to take the absolute value of the error. So that is going to be $df - \cos x$ which is the error and I want the relative value, okay, so the relative error.

So that is the error. So now I have got for that dx, I have the error. So all I need to do is I need to save this somewhere. I create a Err just for this reason. So to the list of errors, I append this

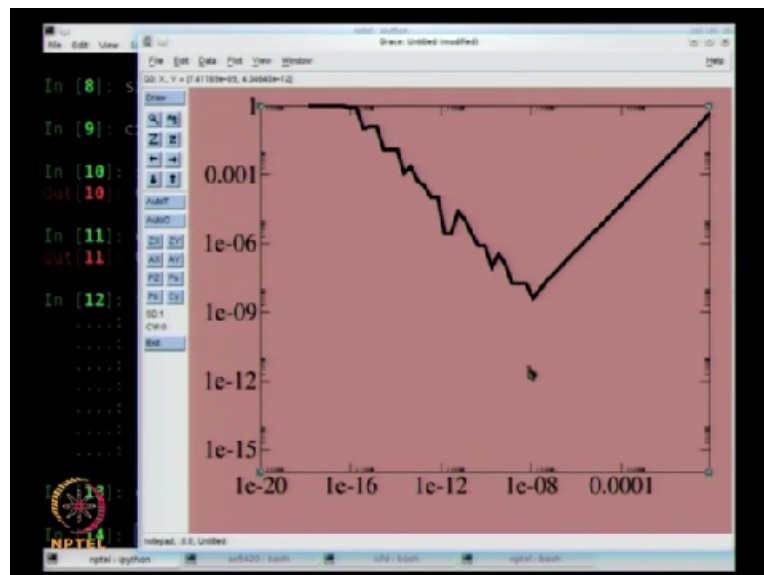
current error and to the list of DX's, I append my dx, okay. All I have to do is create the next dx. So DX is 0.5, I will not use any weird, right you can say star equals but I just said $dx = 0.5 * dx$. So it will half the dx value, fine. I think I have got everything there.

(Refer Slide Time: 08:03)

```
File Edit View Shell/IO Bookmarks Settings Help
In [8]: sx = np.sin(x)
In [9]: cx = np.cos(x)
In [10]: sx
Out[10]: 0.70710678118654746
In [11]: cx
Out[11]: 0.70710678118654757
In [12]: for i in range(50):
...:     dx = (np.sin(x+dx) - sx) / dx
...:     err = abs(dx - cx) / cx
...:     Err.append(err)
...:     DX.append(dx)
...:     cx = 0.5 * dx
...:
In [13]: g.plot( DX, Err )
```

So it should be done. So what we do is, we plot this. So on the graph that I created, I plot dx versus error, error versus dx, error versus dx and what do I get?

(Refer Slide Time: 08:20)



Well that does not make sense. So the deal is what is happening. What is happening here, that does not make sense. So we will have to fiddle around a little with the scale. So it seems that the error started off at some value, goes linear, this is what you will predict that it linearly goes to 0

but there seems to be something happening on the, on the spot. There seems to be something happening here.

So I will change the scale. The key thing to do is I want to change the scale to a logarithmic scale, so I go to $1\text{E}-16$ which is about a small δ I can get here. I go to a logarithmic scale. I apply that, oo, it is and change the major spacing values, okay. So this is what I would... as I said this is what I would normally go through and let me just except this. Let us see. So we get something that looks like that but clearly it goes down to 10^{-8} , 10^{-9} .

Let me change the x axis also because I know that too goes through. So $1\text{E}-20$, right, okay and go to a log scale is to I need and I apply that and that is what I have got, right. I think a lot of you have seen this, I see a lot of people nodding. This is what I have got. So if it were, so in calculus what you do is, remember, it is called finite differences because we just took a difference and left it like that.

We did not go through the infinite process. In the infinite processes, you take the limit δx going to 0. So in calculus what you would expect is as δx goes to 0, the error should go to 0 and you should get the derivative, right but we have a machine that has finite precision and therefore, we have round off error and the slope of this line is, what does this or what do you expect the slope of this line to be? The slope of this line, the slope of that line should be what?

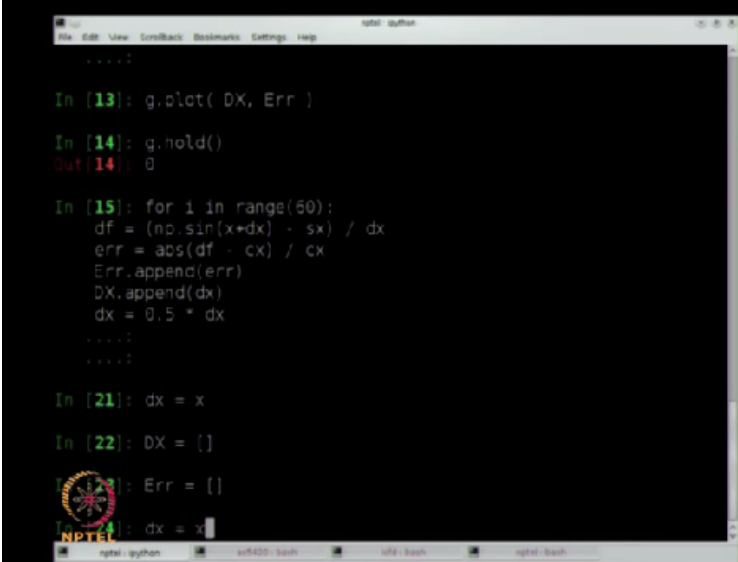
Right, because the truncation error, the convergence, remember this is the order at which, the speed at which it is going down. It is converging to the actual answer was that δx , right. So the exponent is 1. So the slope of this line would be 1 and plotting $\log \delta x$, log error versus $\log \delta x$ now, okay. So the slope of this line is 1 but surprisingly, there seems to be, it seems to stop at something that is not such smooth line.

But if you squint at it, it looks like it has slope -1, right. You start squint at it, it looks like it has slope -1, right, okay. So we will leave this graph and this value here, is of the order of 10^{-8} , 2×10^{-8} , okay. You may not be able to actually read it out but it is of the order of 10^{-8} and that error occurs close to 10^{-8} . So looking at this, the immediate conclusion

that we can come to is, if you are doing forward differences, at least with this particular function, we will try out different functions a little later.

At least with this particular function, it is not worth taking a Δx smaller than 10^{-8} , okay. It is just not worth taking Δx below 10^{-8} because you are going to just be getting, your round off error is going to dominate and it is not getting any more accurate, fine.

(Refer Slide Time: 11:52)



```

In [13]: g.plot( DX, Err )

In [14]: g.hold()
Out[14]: 0

In [15]: for i in range(60):
    df = (np.sin(x+dx) - sx) / dx
    err = abs(df - cx) / cx
    Err.append(err)
    DX.append(dx)
    dx = 0.5 * dx
    ....:
    ....:

In [21]: dx = x

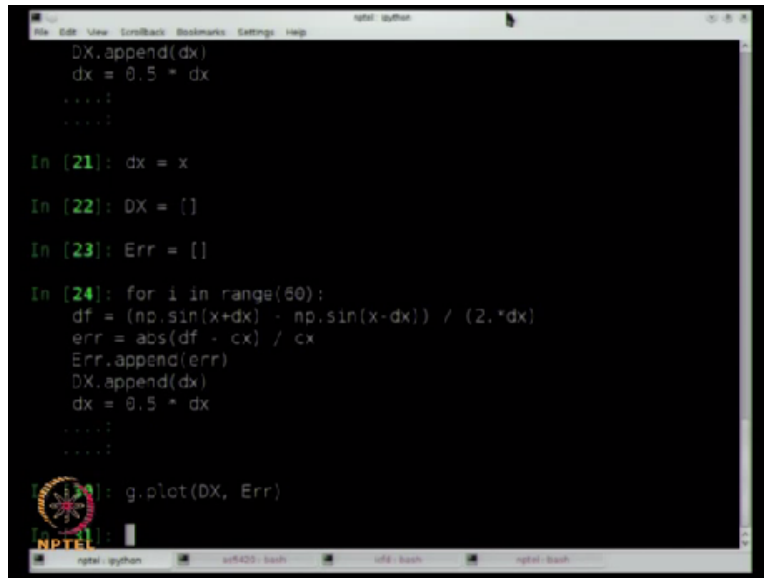
In [22]: DX = []

In [23]: Err = []

In [24]: dx = x
```

So I will just hold this graph just so that I can plot another graph on top of it and what we will do is we will go through the same process. We will now do central differences, okay. We will go through the same process and we will do central differences. So let me redefine my Δx because I sort of set may be this is not what I want to, oh oh oh oh. I made a mistake. Here we go. Let me define my Δx as x again. I redefine, I clear up my capital Δx . I clear up my error, I reset them to 0, okay.

(Refer Slide Time: 12:44)



```
DX.append(dx)
dx = 0.5 * dx
....:
....:

In [21]: dx = x

In [22]: DX = []

In [23]: Err = []

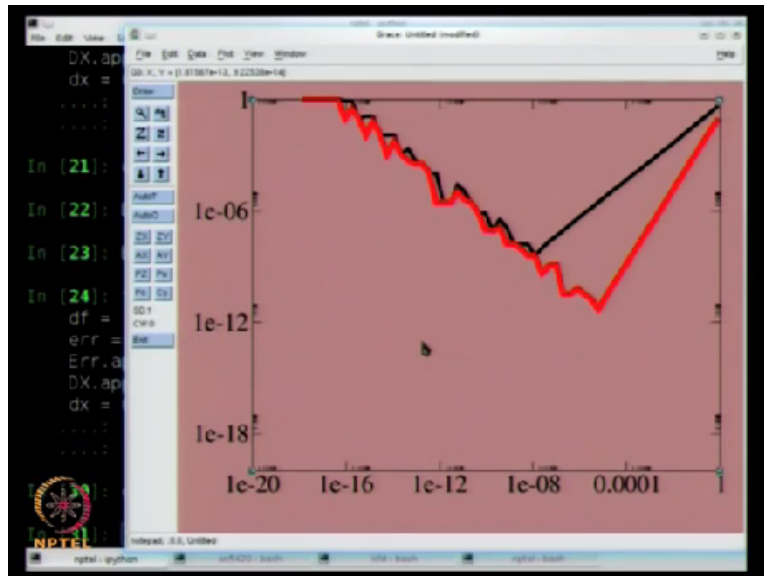
In [24]: for i in range(60):
df = (np.sin(x+dx) - np.sin(x-dx)) / (2.*dx)
err = abs(df - cx) / cx
Err.append(err)
DX.append(dx)
dx = 0.5 * dx
....:
....:

In [25]: g.plot(DX, Err)
```

And in order to get, in order to get the first difference, so I am just reusing, I am just reusing the same code, the only difference is for central subtle differences, this is sine of $x - \Delta x$. Now you have understood why I am sort of using this and I have to divide by $1/2$, $2\Delta x$. In this case, I will make it $2\Delta x$, I will divide by $2\Delta x$. I think everything else is fine, everything else is the same, everything else remains the same, okay.

That is fine? So if I plot this now, so this is second-order actually. If I plot this now, you already have seen, so for this kind of exploratory work and interactive interface like this is quite, quite useful, right. So if you are doing production runs and for the kind of thing that you are learning, I will suggest that you stay with C or C++ or Fortran or whatever and this is what we have, okay.

(Refer Slide Time: 13:54)



So it is the, it is Δx squared, the truncation error is of the order of Δx squared, that means that it is converging at Δx squared. So the slope of this line here, is 2 because I have taken log, okay and I am plotting versus Δx . This slope has not changed. Round off error is not changing. So what has happened is I have gone to a more accurate scheme and there is a line here because this is log, log scale.

I would suggest that if you have tried it out already, figure it out, just write out the relative error, take log on both sides and see what you get, right and you will see that you can actually explain why this is a 45 degree line, the round off error line is a 45 degree line. It is a 45 degree line and essentially what it says is that in the numerator, right, every time the Δx becomes closer to 0, okay, in a binary system by 1 width.

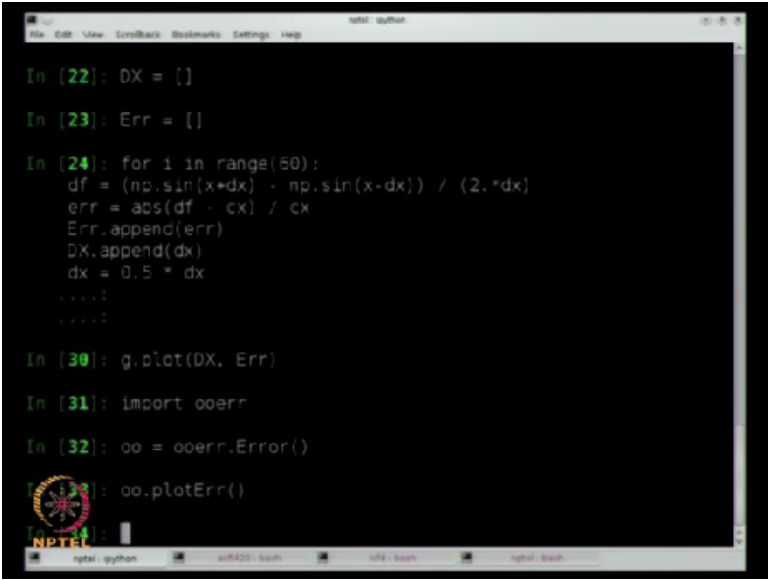
You lose 1 bit in the numerator in round off error and effectively you have, so you are not getting as you hit up, you hit this limit. So this value is of the order of 10^{-11} , okay. So the best that you can get is of the order of 10^{-11} but the Δx that you can take in order to get that 10^{-11} is 10^{-6} . So if you decide yes I want a more, I want something that has better truncation error, I go to a Δx , so I go to something that is converging as Δx squared.

I cannot take as small a Δx . Yes, I have gained something. Instead of 10^{-8} , it has

become 10^{-11} but I just cannot make it, you know I cannot go back to 10^{-8} at Δx as 10^{-8} . That is not possible. That does not work. Because if you do that, you are going to end up back here. So you have gained nothing. The second-order scheme is giving you the same as the first-order scheme, that is the key, okay, that is the key.

It is very important because sometimes we have tendency to say, "oh I am going for very high accurate, I want to get this really accurate scheme, go for as higher order as possible, make the Δx as small as possible". So there are, you have to be careful. You have to be careful. We have the resources but you have to be careful, okay. So, now instead of, so I can do a backward difference, forward difference. You know there are variety of ways by which I can, I can go through this. As I have said I have got a prewritten, prewritten code for...

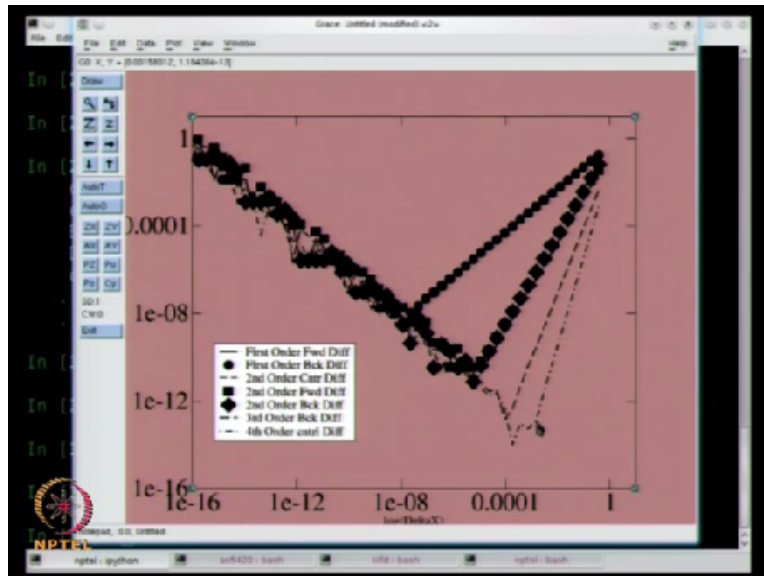
(Refer Slide Time: 16:21)



```
In [22]: DX = []
In [23]: Err = []
In [24]: for i in range(50):
    df = (np.sin(x+dx) - np.sin(x-dx)) / (2.*dx)
    err = abs(df - cx) / cx
    Err.append(err)
    DX.append(dx)
    dx = 0.5 * dx
    ....:
    ....:
In [30]: g.plot(DX, Err)
In [31]: import ooerr
In [32]: oo = ooerr.Error()
In [33]: oo.plotErr()
```

So I will just import prewritten code that I have. So all of this stuff as I said you can just go through a Python tutorial quickly and figure out what we have been doing here. So in this I have something called error.

(Refer Slide Time: 16:42)



And what this will do for me is this will open up this nice window again. I will make sure that I set it up so that the demo colours are on, right, so that I do not blind you guys. Let me go back here and I have a simple function plot error which will actually plot the error. It will compute this for first-order, forward difference, backward difference, second-order, right, second-order, central differences, one-sided differences.

So there are whole host of these kinds of things, third order and fourth order. That is what it is going to do. So let me show you, so I have an error there but it does not matter. Let me show you what I have got. So that is what we have, okay. So this needs a little explanation, it is really messy. So what I have got here. I hope this is clear. What I have got here is first-order forward difference.

The symbol, this symbol is backward, is the dot is backward difference, second-order center, second-order forward, the rhombus, the second-order backward difference, third-order and fourth-order. We will zoom in on that so that it is clear as to what we are, what we are up to. In this scale, what I want to show you is yes that line, round off line, you cannot get, you cannot avoid it. It is going to be there.

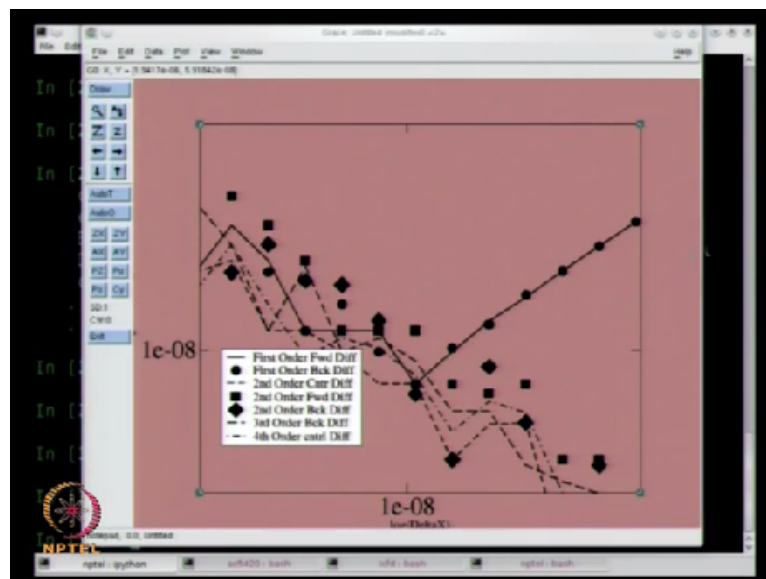
You have to live with it, okay. So whether you do first-order, second-order, third-order, fourth-order, I know you know the slopes are 1, 2, 3, 4, okay and in this region, it does not really look

like. So I would not trust anything beyond this. It is lower here but this looks like the same noisy thing going on. So fourth-order, that is of the order of 10^{-13} . So going from second-order to fourth-order, I went from 10^{-11} to 10^{-14} , 10^{-13} and the Δx is of the order of 0.001, okay.

And in the similar fashion here, that is still of the order of 10^{-12} and the Δx is of the order of 0.002. So it does it, okay. So I want you to bear this in mind. So if you are going to use combination, you are going to do these derivatives, you are going to use combinations of Δx , you know second derivatives, third derivatives which we will see as we go along.

At that time, when you are doing it, I am going to add the second derivative term, I am going to add a fourth derivative term. Please remember that curtails and limits the size of the Δx that you can choose. Is that fine everyone? Okay. Let's zoom in to see what is happening here. So what I can do is I can pick one segment somewhere there and we have lots of graphs but this is the one that I mean to stress.

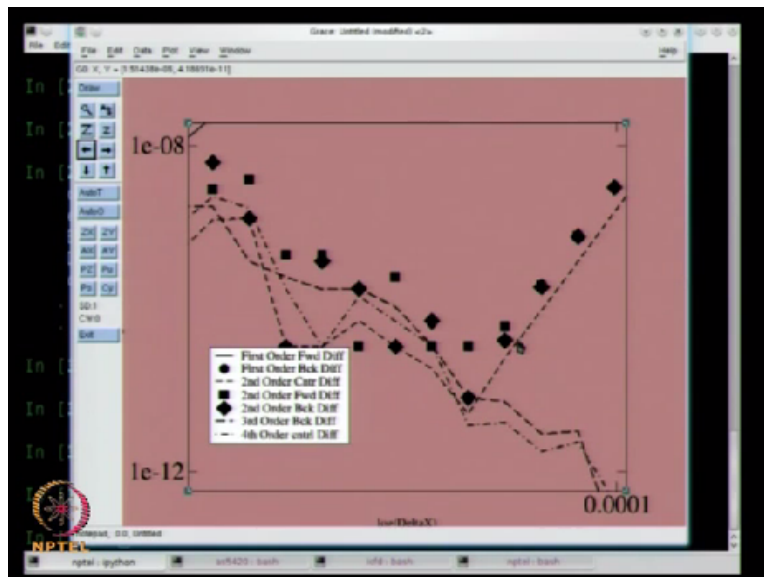
(Refer Slide Time: 19:32)



This was my first 45 degree line, right, forward difference and the dots on it are the backward difference and this part, they are reasonably good, one on top of the other. Remember I am taking absolute value. The sign was different but they are reasonably good. They are essentially one on top of the other. The derivative is the same you would expect to be the same. In the other case

here, they do vary. They are close to each other. They do vary, fine.

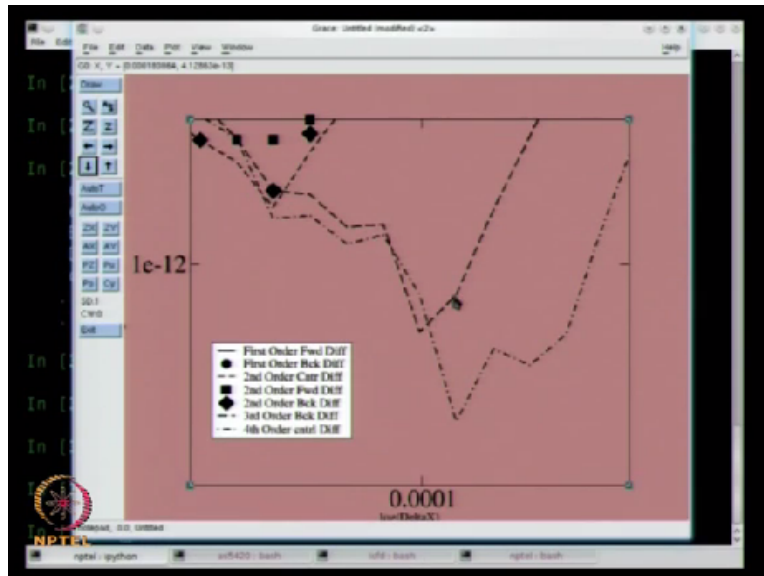
(Refer Slide Time: 20:03)



So if I hunt for the, if I just hunt for the other trough, here is the second derivative term, second-order term for the first derivative, second-order term for the first derivative. There are 3 of them. So you can see the box, the black box, the black rhombus, right, the square, the rhombus and the dash line. The minimum is not quite at the same point. So I most probably but they are, they are reasonably, reasonably fine but they stick to each other, right.

They are quite close. So in that sense if you are forced, so central differences have certain advantages. One-sided differences have certain advantages. We will see why we would use each one at different points. We will see that as we go along but the truncation error behaviour seems to be the same, okay. Is that fine? And finally, just to keep, just for completeness, so you can see around here as I said I do not quite trust this. So I would most probably take a value around there, right which is about 0.02 and possibly a value around there.

(Refer Slide Time: 21:04)



That would be the smallest delta x that I will choose, right and that is about 0.015. Is that fine? Please bear in mind that this is for sine of x at $\pi/4$, okay. So we are not doing mathematical analysis or something of that sort. This is purely empirical; this is purely empirical. Is that fine? What we have? What is the next thing that we can do? May be we can change the function. Should we try changing the function. So what I will do is I will create another...

(Refer Slide Time: 21:43)

```

In [30]: g.plot(DX, Err)

In [31]: import ooerr

In [32]: oo = ooerr.Error()

In [33]: oo.plotErr()

In [34]: ool = ooerr.Error()

In [35]: def f(x):
...:     return x*x
...:

In [36]: def fp(x):
...:     return 2.*x
...:

In [37]: ool.plotErr(f,fp)

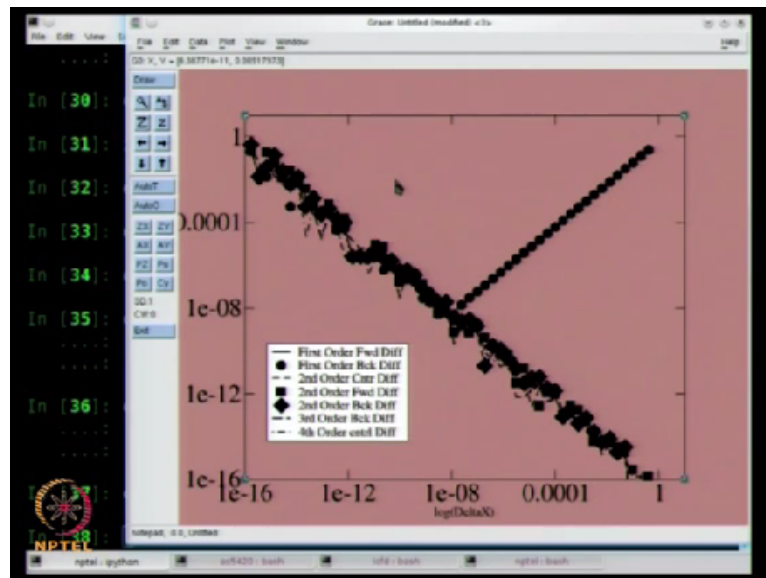
```

I will create another plotting. I will create something else to evaluate it. So ooerr., I will just set this up first, okay. So you are going to get a flash of white again. Let me quickly load my demo parameters so that I get that out of your eyes, okay. So we have that ready. We need to define 2 functions, okay. So maybe I would not do it and you can, you can create lambda functions in

Python but I will just create, I will say define $f(x)$, I will define $f(x)$.

What do you want to, what, which function do you want? Linear function. X squared, okay, fine, x squared. So I return, return x squared. Define f prime. F prime is the derivative, right. So I have to be careful. I could have it do it analytically but let us do it ourselves, okay. Return, I do not want to do any fancy programming here, $2 * x$, $2x$, the derivative, okay. Is that fine? So what I will do is I will now plot and if everything works well, that should just work.

(Refer Slide Time: 23:09)



Ooop, I seem to have some error as I said but it is fine. What is this? What is the strange thing that has happened here? What is the strange thing that has happened here, right? You just get what, what is this line? This line is going all the way down. This line goes all the way down. That is a 45 degree line, right. Can we explain this line? and then you have this which is the first order. So can you explain this line.

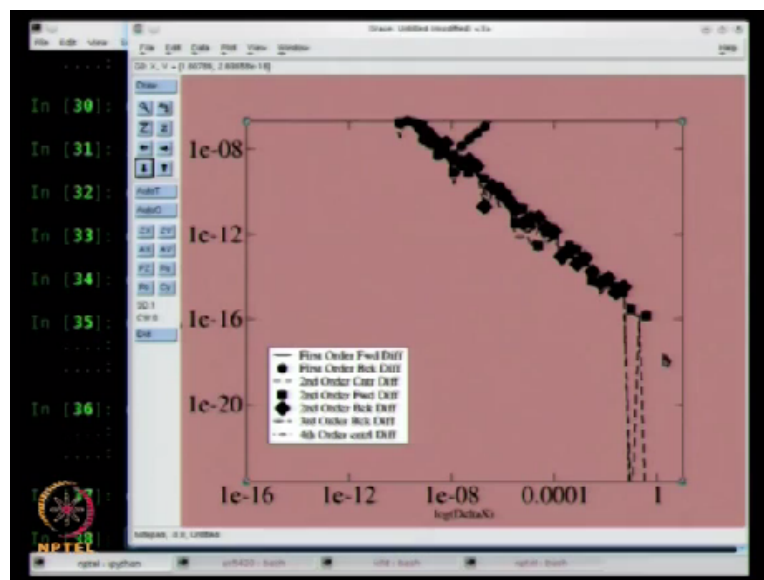
So normally if you were to do this, what you have to do is in your mind, it is like playing a detective game. In your mind you have to predict. From what you know, you have to tell this is what I expect. I try this function; this is what I expect. You have to try it out, right, that is how you, that is how you exercise your understanding. You will predict what is going to, and then plot, okay.

That is what you are doing. Then you have to explain the difference. If you manage to get what you, what you have predicted here, it is fine but otherwise, you have to explain the difference. What has happened here. Can you tell me what is the deal? You have to remember we are plotting x squared and we have a , so this truncation error, if you go back and look at it, this truncation error was like Δx^2 , right.

It was the second, in this case, it is doing squared f do x squared. So Δx^2 is 2. You see what I am saying. Δx^2 is 2. But any higher order, is supposed to be a 0. It is supposed to be 0. It is supposed to be 0 but actually what you have got is only round off error. The truncation error can go, the round off error is always with you unless something peculiar happens.

We will see something peculiar can happen. We will try out one more and see something, right. So the truncation error is 0 because you have picked something so that the third derivative, the truncation error has a third derivative. So legally you should have got 0, right but you are struck with round off error. Is that fine? You are struck with round off error and that round off error is with you. So even if I were to shift the graph down and there is something funny happening here.

(Refer Slide Time: 25:36)



So there may be because of the representation, you may have chosen Δx values where indeed the truncation error by chance and the round off error because of the values are exact in binary

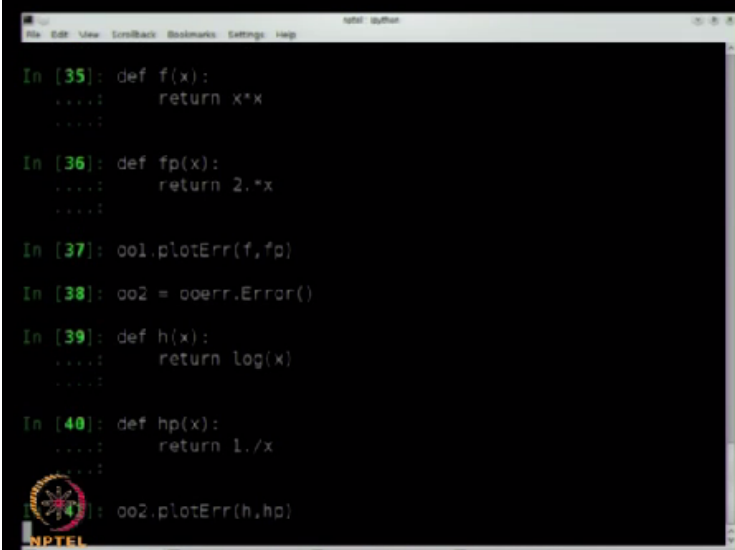
system, happened to be 0. Sometimes you are lucky and that is what these lines indicate. There are values, there are values at which it is actually gone to log 0 which is -infinity. There are few values where it happens.

Is that fine? Okay? But otherwise, otherwise, what we have, I am sorry, maybe I should not have done that. I will close it. Otherwise, what we have is, you just have round off error, right, and that is a, you have to live with it. Even though you do not have, even though you have managed to choose a scheme so that your truncation error is 0. Is that fine? Okay? Are there any questions? You want to try a third function? Suggestions for a function?

Yes, I think well that is going to behave very close to sine. Unless you pick a dirty value or something of that sort, right. You want to try log x? That is okay. Let us say log x, right. Unless you could try, you could try the sinc function or something, yes, we have to pick a value, maybe you could try a value which is close to 0 but then I have to be careful how can I divide. I cannot take relative errors.

You have to take absolute errors, okay, fine. We will, we are human. We will try log x or whatever it is, okay, fine. So I will create, I will create one more...

(Refer Slide Time: 27:24)



```
In [35]: def f(x):
...:     return x*x
...:

In [36]: def fp(x):
...:     return 2.*x
...:

In [37]: oo1.plotErr(f,fp)

In [38]: oo2 = ooerr.Error()

In [39]: def h(x):
...:     return log(x)
...:

In [40]: def hp(x):
...:     return 1./x
...:

In [41]: oo2.plotErr(h,hp)
```

This time I will go through this quickly and yes, I really should have this so that I reuse the

plotting program but anyway it does not matter. So each one of those sort of inputs, its own grace plot, it does not matter. So I now define g of x , a different, oops, I do not want to call it g of x , h of x . See this is how you make programming errors. I already have a g which is the plotting, right. So I want to return, you want log of x , is that what you said, log of x and I have defined h prime, I call it h_p of x and return, the derivative is? $1/x$, okay. So let us see what we get now. 002 h , h_p , I will popup maybe.

(Refer Slide Time: 28:37)

```

-----
NameError                                Traceback (most recent call last)

/home/krishna/demos/nptel/<ipython console> in <module>()

/home/krishna/demos/nptel/coerr.pyc in plotErr(self, f, fp, x)
    37
    38     for i in range(55):
--> 39         sx = f(x)
    40         cx = fp(x)
    41         smx = f(x - dx[-1])

/home/krishna/demos/nptel/<ipython console> in h(x)

NameError: global name 'log' is not defined

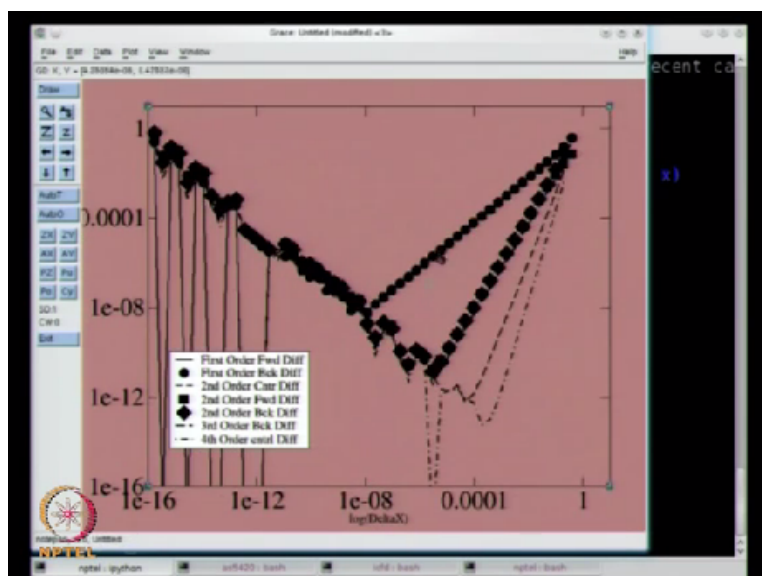
In [42]: def h(x):
        return np.log(x)
        ...

002.plotErr(h,hp)

```

Oh, what happened? I made a mistake and sometimes you make, you can play, you can pay a price for this. See whether the price, how bad the price is? Oh my god.

(Refer Slide Time: 28:56)



Look at all those. So log is, log is a little interesting. So you get something that is very similar to, well it is a transcendental function. So maybe even sync would have been the same, right. But there are, there are places where, there are places where it does seem, they are right. The round off error seems to go away. There are places where, when you are doing computations, you cannot hunt for, mean you do not know what values of x you are going to get, right.

So these are not, it is interesting to know that this can happen and by chance, by chance, right, if it happens for you, it is nice. In this case, because I am getting -infinity and so when the program is not doing anything bad, right, you pop up a window and I kill that window, program is not doing anything bad but you have to remember that when you are taking log, you have to check the argument.

That is one of the things, by chance it may turn out that it is 0, okay. By chance it may turn out that it is but otherwise, I think it looks the behaviour is essentially the same, right. So that is about 10^{-19} , 10^{-13} . This value though is $0.004 \cdot 10^{-4}$, 10^{-5} , the x value. So you have to, I think, right. So the slopes and where they intersect, what values that you take are essentially the same, okay. Okay, fine?

So we are done with this if there are no questions? Are there any questions? If there are no questions, I will leave this demo B. Let me just show you, right, we will try to see how much we can do by way of Laplace's equation today, fine.

(Refer Slide Time: 30:46)

```
ll last)

/home/krishna/demos/nptel/<ipython console> in <module>()

/home/krishna/demos/nptel/ooerr.pyc in plotErr(self, f, fp, x)
    37
    38     for i in range(55):
--> 39         sx = f(x)
    40         cx = fp( x )
    41         smx = f( x - dx[-1] )

/home/krishna/demos/nptel/<ipython console> in h(x)
NameError: global name 'log' is not defined

In [42]: def h(x):
        return np.log(x)
        ....:

In [44]: oo2.plotErr(h, hp)

In [45]: from enthought.mayavi import mlab
```

So I will choose the small, small grid to start with. What I am going to do is I am going to use as I said a package called Mayavi. So from, I will just set it up right. So from this Mayavi, I will import, write a particular module called mlab. Now initially what I will do is I will use a 5*5, I will choose, I am sorry.

(Refer Slide Time: 31:30)

```
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object'. ?object also works, ?? prints more.

In [1]: from enthought.mayavi import mlab

In [2]: import numpy as np

In [3]: from gracePlot import gracePlot

In [4]: x,y = np.mgrid[0.:1.:5j,0.:1.:5j]

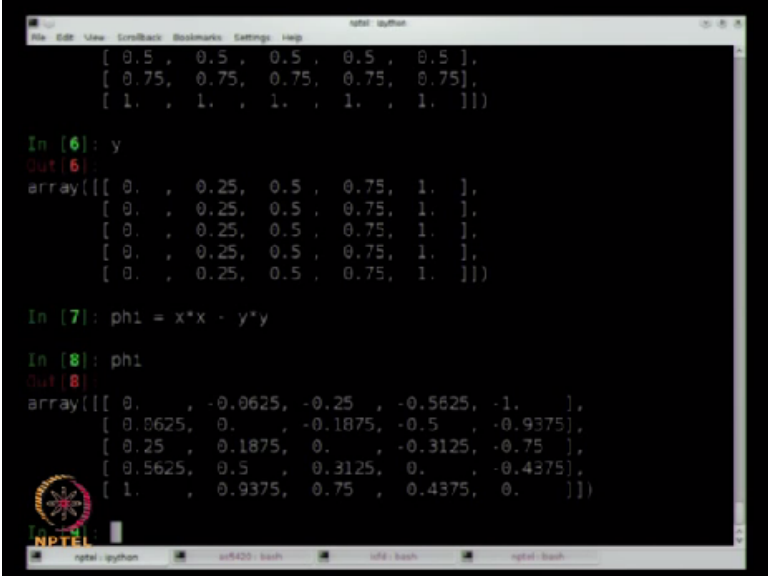
In [5]: x
Out[5]:
array([[ 0. ,  0. ,  0. ,  0. ,  0. ],
       [ 0.25,  0.25,  0.25,  0.25,  0.25],
       [ 0.5 ,  0.5 ,  0.5 ,  0.5 ,  0.5 ],
       [ 0.75,  0.75,  0.75,  0.75,  0.75],
       [ 1. ,  1. ,  1. ,  1. ,  1. ]])
```

This is what is called, so if you have a live demo, these are the things that can go wrong. You see if I get out of that and we restart it, okay. So what I will do is I will, I will create a 5*5 system first or a, right, 6*6 system or whatever. We will create a small system and then just so that you understand how the Numpy Python part works, right. I just want to make sure that you get that part first, right and then we will see.

So then we make sure since I said dumpy, let me make sure that I import Numpy as np and just in case, I need, I will import grace plot also, okay. So all that stuff is my basic homework is done. Now in this case I am going to use a package called mgrid, I am going to use a function called mgrid and what it does is, so between 0 and when we have been solving problems between 0 and 1, I am going to have 5 grid points.

You have seen this notation before and in the x direction and in the y direction also we will have the same thing. So this 5j, it is actually a complex number. It is a way by which you tell, you tell Numpy that you want 5 grid points and that creates essentially a mesh, okay. So let me show you the mesh. That is x. So you can see the x values are, right, 0 and then they are increasing by 0.25, 0.5, 0.75, 1, okay.

(Refer Slide Time: 33:33)



```

[ 0.5 , 0.5 , 0.5 , 0.5 , 0.5 ],
[ 0.75, 0.75, 0.75, 0.75, 0.75],
[ 1. , 1. , 1. , 1. , 1. ]]

In [6]: y
Out[6]:
array([[ 0. , 0.25, 0.5 , 0.75, 1. ],
       [ 0. , 0.25, 0.5 , 0.75, 1. ],
       [ 0. , 0.25, 0.5 , 0.75, 1. ],
       [ 0. , 0.25, 0.5 , 0.75, 1. ],
       [ 0. , 0.25, 0.5 , 0.75, 1. ]])

In [7]: ph1 = x*x - y*y

In [8]: ph1
Out[8]:
array([[ 0. , -0.0625, -0.25 , -0.5625, -1.   ],
       [ 0.0625, 0. , -0.1875, -0.5 , -0.9375],
       [ 0.25 , 0.1875, 0. , -0.3125, -0.75 ],
       [ 0.5625, 0.5 , 0.3125, 0. , -0.4375],
       [ 1. , 0.9375, 0.75 , 0.4375, 0.   ]])

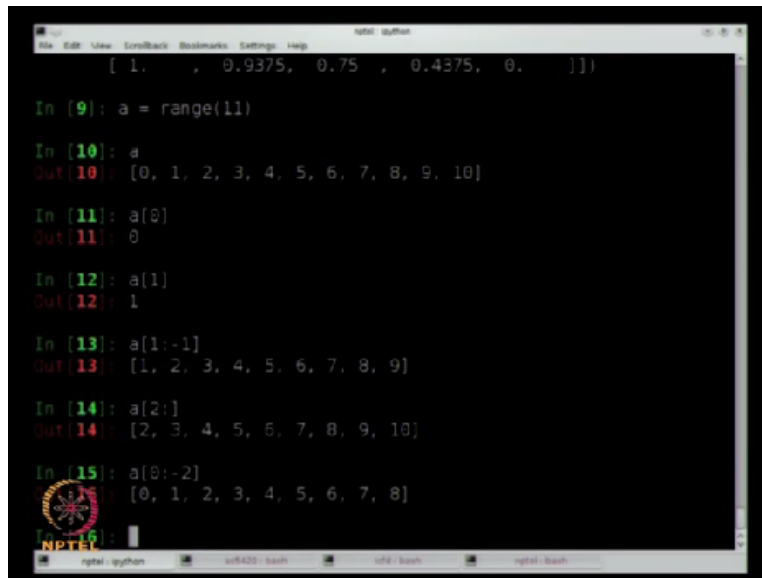
```

So those are my x values and my y values are in a similar fashion, 0, 0.25, 0.5, 0.75, 1. So xy combinations will give me coordinates. A combination of an entry from x and an entry from y will give me coordinates, okay and my solution that I use, what I use for, right, what I suggested for you guys to try out was x squared-y squared. So I can see what that looks like, okay. That is x squared-y squared.

So that is p, x squared-y squared and yes, you can check it out but you can see that the diagonals

are 0, $x=y$ is 0, right. So that is reasonable. That looks okay. Is that fine? Okay? Now I want to show you and this is something that is done in most, lot of these programming languages, scripting languages. Let me just show you to see how I am going to index. I am going to choose ranges of indices so that I do not have to do a lot of loops.

(Refer Slide Time: 34:44)

A screenshot of a Jupyter Notebook interface with a dark background. The notebook shows a series of input-output pairs for creating and indexing a list. At the top, there is a line of code: `[1., 0.9375, 0.75, 0.4375, 0.]]`. Below it, the following interactions are shown:

- `In [9]: a = range(11)`
- `In [10]: a`
`Out[10]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
- `In [11]: a[0]`
`Out[11]: 0`
- `In [12]: a[1]`
`Out[12]: 1`
- `In [13]: a[1:-1]`
`Out[13]: [1, 2, 3, 4, 5, 6, 7, 8, 9]`
- `In [14]: a[2:]`
`Out[14]: [2, 3, 4, 5, 6, 7, 8, 9, 10]`
- `In [15]: a[0:-2]`
`Out[15]: [0, 1, 2, 3, 4, 5, 6, 7, 8]`

The Jupyter logo and the text "NPTEL" are visible in the bottom left corner of the notebook window.

So I just created a list, an array, which has 11 elements, 0 through 10, okay. So if I say `a[0]`, that gives me 0. `a[1]`, gives me 1, fine, no big deal. You can also do this. You can say `a[1:-1]` and that will basically drop the first element, drop the last element. Is that fine? Everybody is with me? So instead of 0 through 10, I have got 1 through 9, okay.

I just want you to understand this. So if I make this 2 instead of 1, it drops the first 2 elements, retains all the others, okay. If I make this -2, it drops the last 2 elements. Am I making sense? So what this allows me to do is when I am saying `i+1` `i-1`, all I will do is I will shift that, okay. So for instance `a[1:-1]` gives me all the interior points essentially. Is it okay?

(Refer Slide Time: 35:47)

```

In [15]: [0, 1, 2, 3, 4, 5, 6, 7, 8]

Out[15]: [0, 1, 2, 3, 4, 5, 6, 7, 8]

In [16]: phi[1:-1,1:-1]
Out[16]:
array([[ 0.    , -0.1875, -0.5   ],
       [ 0.1875,  0.    , -0.3125],
       [ 0.5   ,  0.3125,  0.    ]])

In [17]: phi[1:-1,1:-1] = 0.0

In [18]: phi
Out[18]:
array([[ 0.    , -0.0625, -0.25  , -0.5625, -1.    ],
       [ 0.0625,  0.    ,  0.    ,  0.    , -0.9375],
       [ 0.25  ,  0.    ,  0.    ,  0.    , -0.75  ],
       [ 0.5625,  0.    ,  0.    ,  0.    , -0.4375],
       [ 1.    ,  0.9375,  0.75  ,  0.4375,  0.    ]])

In [19]: phi[1:-1,1:-1] = 0.25*(phi[0:-2,1:-1]+phi[2:,1:-1]+phi[1:-1,0:-2])

In [20]: phi[1:-1,1:-1] = 0.0

```

So in a similar fashion, phi of i, this has 2 coordinates 1:-1, 1:-1 will give me the interior points, okay. So that gives me the interior points. Is that fine? Because we are going to solve for this. See we are not supposed to know the, know the solution, it has been solved for this, I will set it = 0. I will set the interior points to be 0. So that is my phi0. This is my solution vector, okay.

This is my candidate solution. My initial condition is 0 and the boundary conditions are set. Is that fine? Everybody? Okay. So how would I do Laplace's equation, right. So normally in C, right, you would use a forward loop. In Fortran, you would use a do loop. Here we are just basically going to write it out. So what you would have done so far was, so 1:-1, so the interior points, I am going to update the interior points, $=0.25 \times$ one-fourth of what?

Phi of, we want to shift to the left, 0:-2 that shifted to the left, 1:-1, nothing done in the y direction, +phi of, I want to shift to the right, 2:,1:-1, I do not want to do anything in the y direction. Now we repeat the same process but only in the y direction, okay. So phi of, what is it? 1:-1. So I do not do anything in the x direction now, 0:-2. So I have shifted it down, no, sorry, sorry, sorry, sorry, sorry.

(Refer Slide Time: 37:50)

```
npTEL Python
File Edit View Toolbar Bookmarks Settings Help
In [17]: phi[1:-1,1:-1] = 0.0

In [18]: phi
Out[18]:
array([[ 0.        , -0.0625, -0.25      , -0.5625, -1.        ],
       [ 0.0625,  0.        ,  0.        ,  0.        , -0.9375],
       [ 0.25     ,  0.        ,  0.        ,  0.        , -0.75     ],
       [ 0.5625,  0.        ,  0.        ,  0.        , -0.4375],
       [ 1.        ,  0.9375,  0.75     ,  0.4375,  0.        ]])

In [19]: phi[1:-1,1:-1] = 0.25*(phi[0:-2,1:-1]+phi[2:,1:-1]+phi[1:-1,0:-2])

In [20]: phi[1:-1,1:-1] = 0.0

In [21]: phi[1:-1,1:-1] = 0.25*(phi[0:-2,1:-1]+phi[2:,1:-1]+phi[1:-1,0:-2]+phi[1:-1,2:])

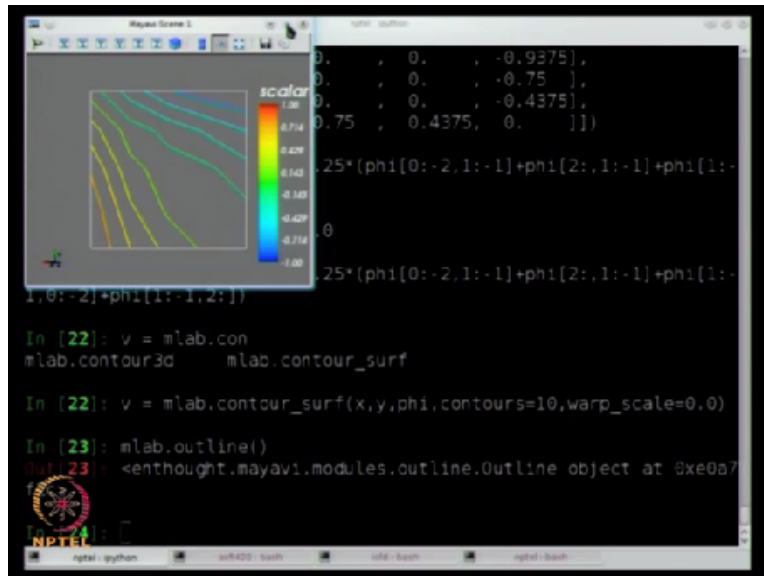
In [22]: v = mlab.con
mlab.contour3d      mlab.contour_surf
npTEL
v = mlab.contour_surf(x,y,phi,contours=10,warp_scale=0.0)
```

Let us go through this again. I hit enter too quickly. Set the center value to 0 and I have one more, phi of 1-1, tell me? 2:, okay. That should be right. So we just have this, right and it is always important that we able to see what we have, fine. So what I will do is I will create a visualization now. I am going to use Mayavi to do that. So I will say v, there are various ways to do it.

As we go along, maybe I will show you different ways by which we can do it. Today, I will just use mlab and I will use something called a contour surface, contour surface. To contour surface, I need to pass it the xy values and the phi value, okay. So I have xy phi and I would like to tell it how many contours. So let us say we add about 10 contours, okay. Now because this is a actually a 3D plotting utility, it will plot the contours on the actual solution surface, right.

Today, right now I do not want to do that. We will do that; we will do that at the later date. Right now, I want to keep it simple because, so what I will do is, I will say a warp scale. So how is, right. So all of these things as I said you can, the documentation is available on line. So I will just set that to 0 and that should think for a short while and just basically give me that. That does not make sense.

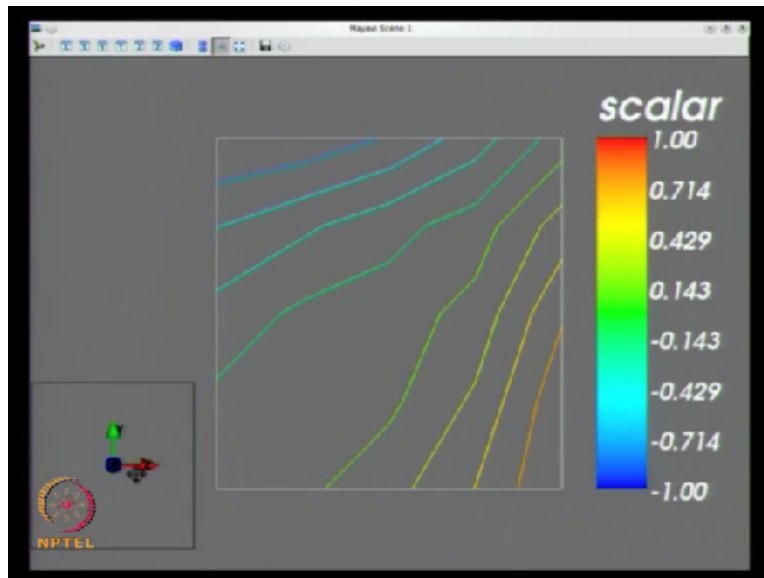
(Refer Slide Time: 39:23)



So you can ask the question, what is that? Okay, so we will try to see whether we are able to make any sense or do it give it a coordinate system. I will, I will scale it up. So that still does not make sense. So let us see if we can, I use mlab to add an outline or something of that sort so that we get the box okay. That is our unit square, okay. That is our unit square with the contours and let me maximize it for you, make it large, okay.

I think I will also add a scale so that you have an idea as to what is happening. Okay we have added number of contours. It should be in texturing, okay. That is not what I want to do. Or do I want to do. Show legend, that is what I want to do. I want to show a legend and I close this. Yes, you have got the legend there. Now I will maximize this so that you can look at what we have.

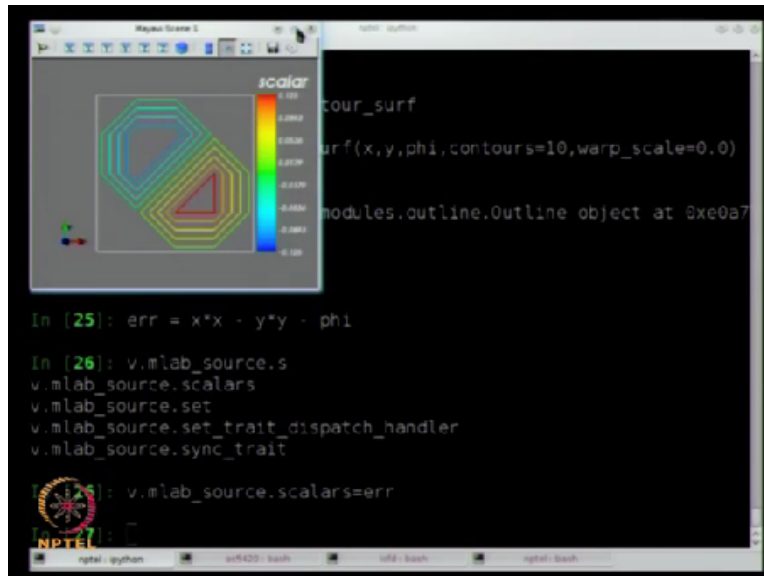
(Refer Slide Time: 41:03)



Here we go. So I have a legend, fine. I have a, I have the legend. It goes from +1 to -1 as you would expect and that is my initial guess. Fine, everyone? So this, in case you cannot make that scale out, yes. So that the xy coordinates. So this is, this is the x direction, that is y direction, fine. I hope you can make out the scale. Are there any questions? Okay. Let me just reduce this to something that is smaller and do this one more time, okay.

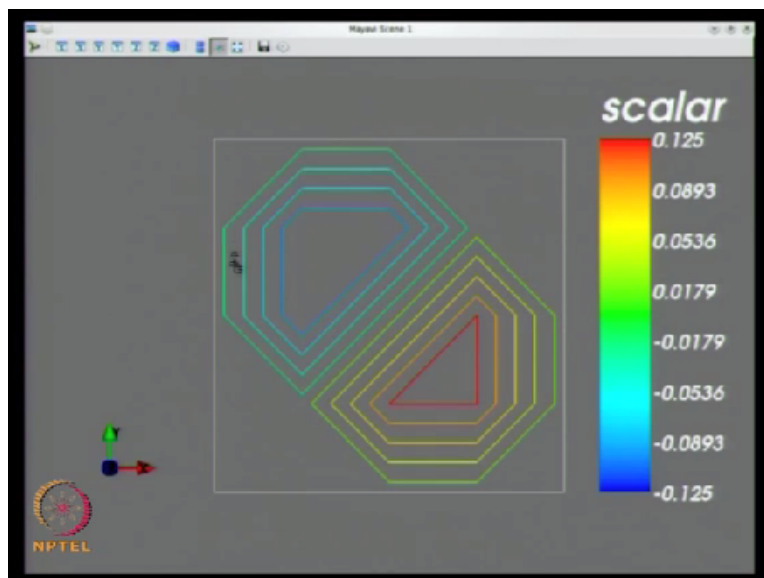
So and we may want to keep track of the error, is it possible for us to look at the error. We can actually look at the error, okay. It is possible for us to look at the error. So what I will do is I want to keep track of the error. So I will like last time, I will create, I will create a list called Err but I will calculate the error. Error is, what is the error? What was the solution? $x^2 - y^2$ and from this I want to subtract $(())$ (42:11). Everyone, that is fine? Okay. So and what we will do is, we can visualize the error also. We can see what the error is.

(Refer Slide Time: 42:24)



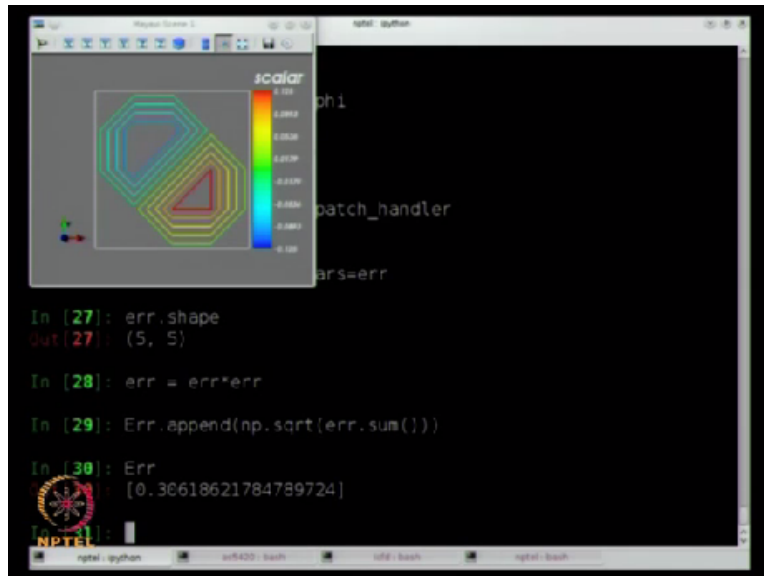
So `v.mlab scalars=err` and that is what the error looks like, okay. So the error is symmetric. The error is symmetric. If it is too small for you to see, I will maximize it again.

(Refer Slide Time: 42:44)



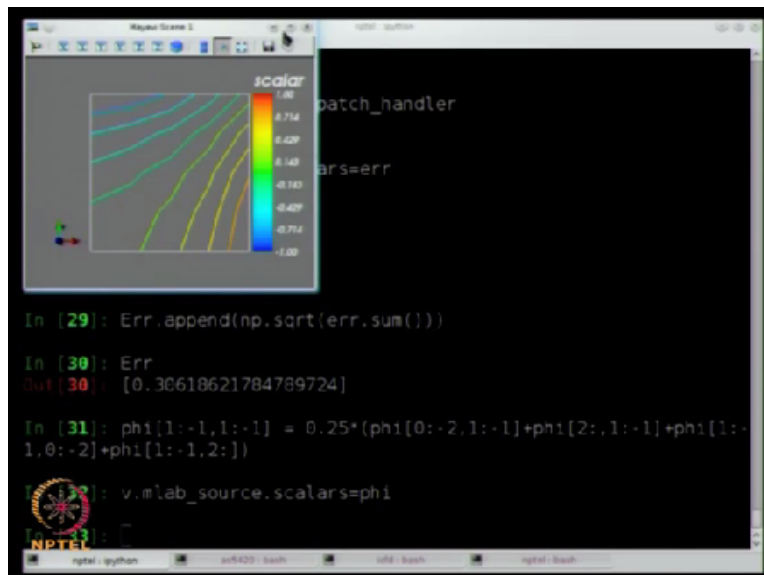
The error is symmetric. The error goes from $+0.125$ to -0.125 . The error is symmetric, okay. So this demo is 2 fold. One is how do you explore. When you are doing numerical solutions and so on, how do you explore your schemes. How do you go about that process? This is interactive. You can try things out, okay. So and the other is that we try to figure out how, how fast. So I have got this error but this is a, this error is a matrix, right.

(Refer Slide Time: 43:16)



If you say, if you ask the question what is this error? This error in fact is a matrix. So 5*5 matrix. It does not help. So let me take the square of that matrix so that those negative numbers go away, okay. Let me take the and what I will do is, I will take the square root, I will take the square root and put it in error, err, my list, okay, square root of the sum of all those terms. That is why you should open and close brackets every time you do it, right and make mistakes otherwise, okay. So what is err? Err has one value.

(Refer Slide Time: 44:04)

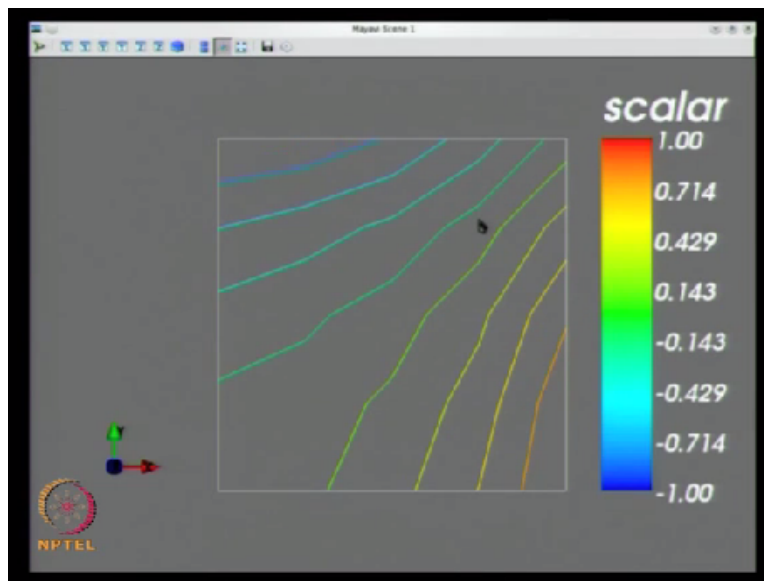


So the error now, the square root or the sum of the squares, this point 3 after one iteration. Is that fine. I can do one more iteration. Let me go back, find my iteration, I will do one more iteration, okay, one more iteration. What does that do to the solution? What does that do to the solution?

So that is what you get.

It looks about the same, right. What we will do is at a later date without doing all those errors and all that, we will run it once and you can see the solution of all that, right. I will make one fancy demo where I have got everything right, so that you can see the solution evolving as it goes along, okay which is all so far but as I have indicated, you are, let me just maximize it.

(Refer Slide Time: 44:56)



As I have indicated, it looks symmetric. Everything is nice and I can also plot.

(Refer Slide Time: 45:06)

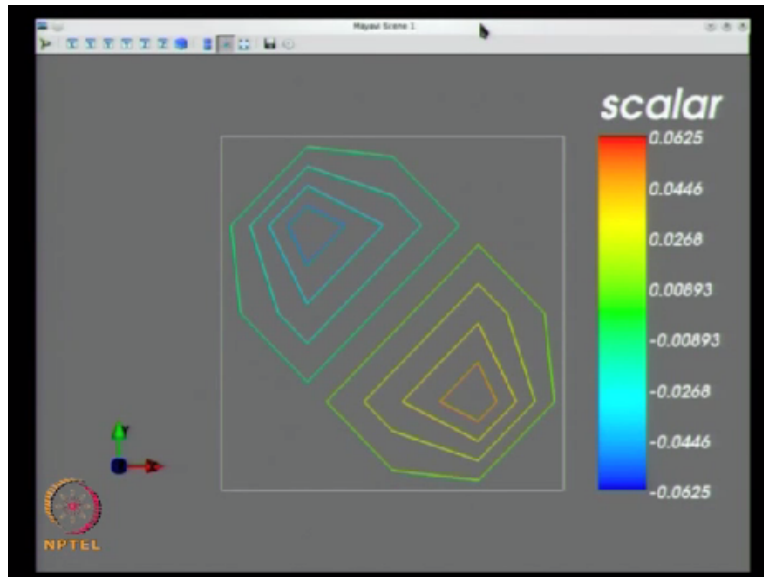
```
npip: ipython
File Edit View Shell/Back Bookmarks Settings Help

In [26]: v.mlab_source.scalars=err
In [27]: err.shape
Out[27]: (5, 5)
In [28]: err = err*err
In [29]: Err.append(np.sqrt(err.sum()))
In [30]: Err
Out[30]: [0.30618621784789724]
In [31]: phi[1:-1,1:-1] = 0.25*(phi[0:-2,1:-1]+phi[2:,1:-1]+phi[1:-1,0:-2]+phi[1:-1,2:])
In [32]: v.mlab_source.scalars=phi
In [33]: err = x*x - y*y - phi
In [34]: v.mlab_source.scalars=err
```

I can also plot the error just like we did last time. So I do that error is $x^2 - y^2 - 5$. I can

plot that.

(Refer Slide Time: 45:22)



See what that looks like. So from 0.125, it has become 0.0625. In fact, it seems to have halved. Range of the scale seems to have halved, okay. So that is basically what we have. The range of the scale seems to have halved. So we can do, we can do this business now. So we say where did we have that. I had errors. I will square the error and I will append that. I will append that, fine. Okay, everyone? Okay, so we can keep doing this manually instead of doing this, it is nice when you are doing it once or twice but what we will now do is, we will iterate.

(Refer Slide Time: 46:12)

```
npml - author
File Edit View Graphics Windows Settings Help
In [33]: err = x*x - y*y - phi
In [34]: v.mlab_source.scalars=err
In [35]: err = err*err
In [36]: Err.append(np.sqrt(err.sum()))
In [37]: for i in range(100):
...:     phi[1:-1,1:-1] = 0.25*(phi[0:-2,1:-1]+phi[2:,1:-1]+phi[1:-1,0:-2]+phi[1:-1,2:])
...:     v.mlab_source.scalars=phi
...:     err = x*x - y*y - phi
...:     err = err*err
...:     Err.append(np.sqrt(err.sum()))
...:
In [38]: g = gracePlot()
In [39]: g.plot(Err)
NPTEL
```

How many times you want to do this? 100 times? You want to do it 100 times, let us do it 100

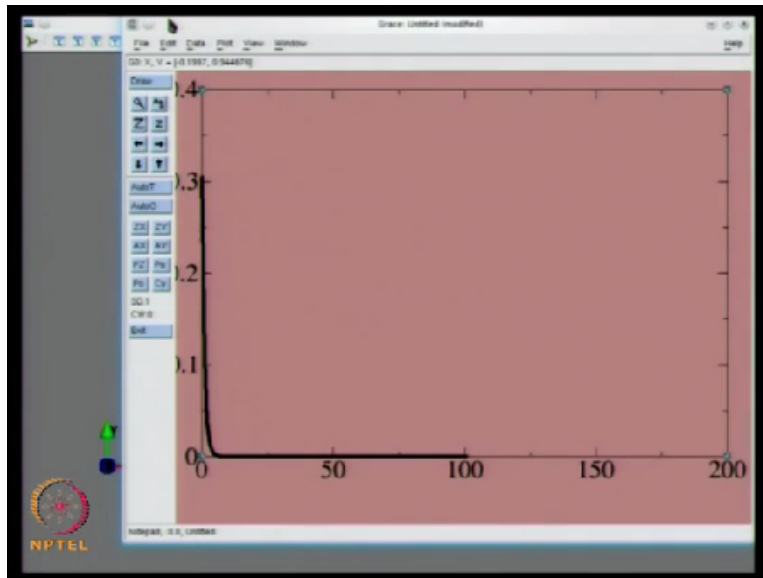
times. See what we get? Okay, 100 times. I am a lazy guy and scared I may make a mistake. So right, so we have to be, we have to be careful. So that is ϕ , right and again we will stop looking at plotting the error. You understand what I am saying. So we will plot only the, we will plot only the ϕ as it evolves.

We can always plot the error part later, okay. So we reset that. We have to re-compute the error each time. So where do I have that. I have to re-compute the error each time, okay. What else? We need to square that and I need to append it. That will make sure. I think I have got everything. I have done all the steps that I need to do, okay. So programming, this is one thing I have to repeat, we are all error prone, right. So you have to be constantly paranoid.

You have to check to make sure that everything is fine and what is happening here, well the solution apparently is evolving. I cannot make out anything. That is the tragedy of I have gone through. It is finished, right. Because the errors are small. So this is the other thing that I want you to understand. So if you look at 2 graphs on the screen, if you look at 2 graphs on the screen.

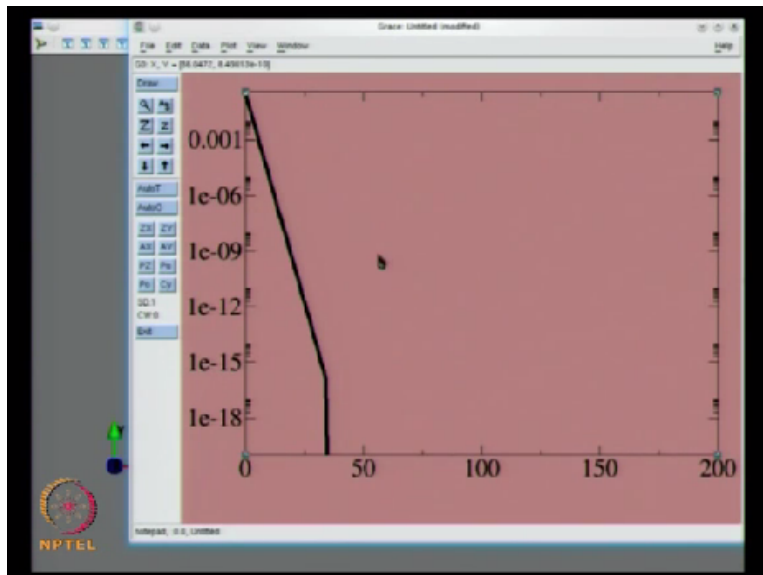
And the graphs look very close, that does not say much, right because the screen, typical screen resolution, this is like 70 dots per inch, right or 100 dots per inch. Just because 2 lines are close, does not mean that they are close. Am I making sense. So it does not mean. So you have to, that is the reason why I stored that error, the other error, okay. So now I will create my grace plot where we are going to look at it, right. Remember right at the beginning I had this.

(Refer Slide Time: 48:39)



I will quickly, I am sorry, quickly get rid of that, right. I have created my grace plot and I will plot err, okay and as usual, it gives me, see this is the other thing. So lot of us do this plotting. You have to always remember, always plot when you are plotting residues and so on, the range is very large. So you need to go to a log scale, a log scale. So we will change this to a log scale.

(Refer Slide Time: 49:18)



So I will make this very ambitiously, 1E-16, put it now on the log scale, apply that, make, oh I should have made that say 1000 or 10,000 or something of that sort so that it does not look cluttered up. So it seems to have done, it seems to have done. This is only 50. It seems to have done better than that. Maybe 1E-20. Oh, it is not bad, okay. So it is possible that, it is possible that you get 2 identical numbers.

This is actually possible. So what has happened is after about, after about 30-40 iterations, your answers became the exact answer on a log scale, that looks linear, looks like a straight line. So you could try to find out what is the convergence rate. Is that fine, okay. So next time in the next class, what we will do is we will do a demo but we will do a larger, larger, right, larger one and see what the scale works out, okay. Is that fine? Thank you.