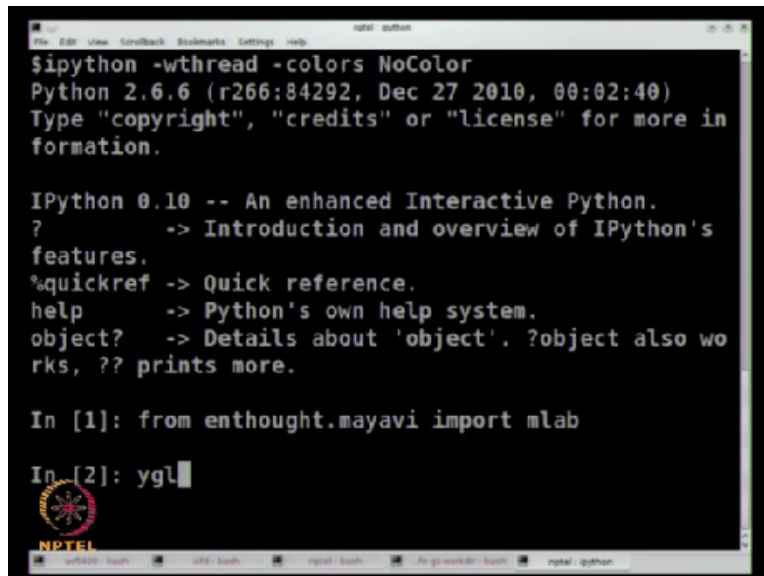


Introduction to Computational Fluid Dynamics
Prof. M. Ramakrishna
Department of Aerospace Engineering
Indian Institute of Technology - Madras

Lecture – 14
Demo - Laplace equation, SOR

Okay, good morning.

(Refer Slide Time: 00:13)



```
$ipython -wthread -colors NoColor
Python 2.6.6 (r266:84292, Dec 27 2010, 00:02:40)
Type "copyright", "credits" or "license" for more information.

IPython 0.10 -- An enhanced Interactive Python.
?          -> 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]: ygl
```

I am going to get back to the demo on Laplace's equation. What we will do today is I will start with the same 5*5 grid that is we will have 3 interior grid points. Last time I do not know whether you realized it or not but we were doing Jacobi iterations. Just for that you realized that I was doing Jacobi iterations, right, it was simultaneous relaxation. All the points were changed in one shot, okay.

That is basically what was happening. We will also try to say whether we can do Gauss-Seidel, okay. So we will do that on a 5*5 system and 5*5 grid, basically remember gives you a 3*3 interior grid points. So the actual system of equations that you are solving is a 9*9 system of equations. So we will do that with, say with that 5*5 system and then maybe we will try something larger, right and for the larger one, I will basically use a package, not package but a program that I have already written so that I do not sit down.

And we can do it manually, mean it does not matter. I can write the program here. It is not big deal. Maybe even for the larger one I will just, I will just do it by, do it by hand. It is not, it is just averaging, right. Then if we have time today, we will also do SOR. I do not know whether you have tried looking at SOR, right but I already have program written for SOR for finding the optimal omega value.

It is not, to aid in finding the optimal omega value, right. It does not actually hunt for the, it does not do an optimisation, does not solve an optimisation problem, okay. So that is as far as that goes. The other thing is if you have tried SOR, right, so that is remember it was over-relaxation but it was still successive over-relaxation. It was for the successive iteration, that is for the Gauss-Seidel, okay.

My suggestion is try to find out what happens if you use a relaxation parameter with Jacobi iteration just for the fun of it, just see what happens if you try, right. If you try using a relaxation parameter with the original Jacobi iteration algorithm that we came up. Is that fine? It is okay? So just try to find out what happens and we will revisit that question what I just suggested now, we will revisit it a few weeks from now. I will come back to that point a few from now, okay.

So in starting of IPython. When you use it, of course you do not have to do this nocolor stuff mean, I am doing nocolor simply because the red that comes here does not come out along the video, okay. That is the only reason why I am shutting off color. I just wanted to show you that as a difference between last class and this class so that we do not run into any confusion, okay. So we will set it up quickly.

So let me, just to, so that is the, that is the plotting utility and this package Mayavi, just so that you know, is built on top of another package called visualisation toolkit which is a C++ packaged. So if you want to access any of these 3D plotting, utilities, functions, right either in C or C++, you can easily write a program and try to visualisation toolkit. It is called VTK. The library is called libvtk, okay.

I will just type it out so that you know what it is. I am not going to, the library is called libvtk.

What Mayavi does is it try places a layer of an interface between Python and the VTK toolkit, the visualisation toolkit. That is basically what is happening. So all the visualization because I had a question in the last class about visualizing what tools do we use if you are writing programs in C and C++ and so on, right.

So fundamentally all the visualisation programs are basically written in C, C++, Fortran, right and what you see here in Python is normally a Python interface to an existing programme in Fortran. So there are lot of, lot of packages out there that you can use, okay. So grace plot for instance is the front-end to something called the grace which has a history of its own, either xmgrace or xmgr or whatever it is.

So you can, you can, you can try to interface directly to grace, okay. You can try a package that I like to use, it is called ygl, I will just type it again because, this package basically tries to emulate the silicon graphics, graphics library called gl, right, so which is very easy to use. So you can try. I mean as I said but there are lots of packages out there that you can do, use and finally of course is a suggestion, if you can put a dot on the screen and this is relevant to what we are doing in this course, right.

So the screen that you are seeing right upfront is what I am using on my laptop is 1024*768 dots. If you can put a dot on the screen, you can draw a line on the screen. You understand that if you can draw a line on the screen, you can plot, okay. Is that fine? Okay. So there is always, always ways by which you can plot, right. There are always ways by which you can plot. Maybe at a, maybe in the later class, I will show you, do a few demos of various plotting packages that you can use and so on, okay.

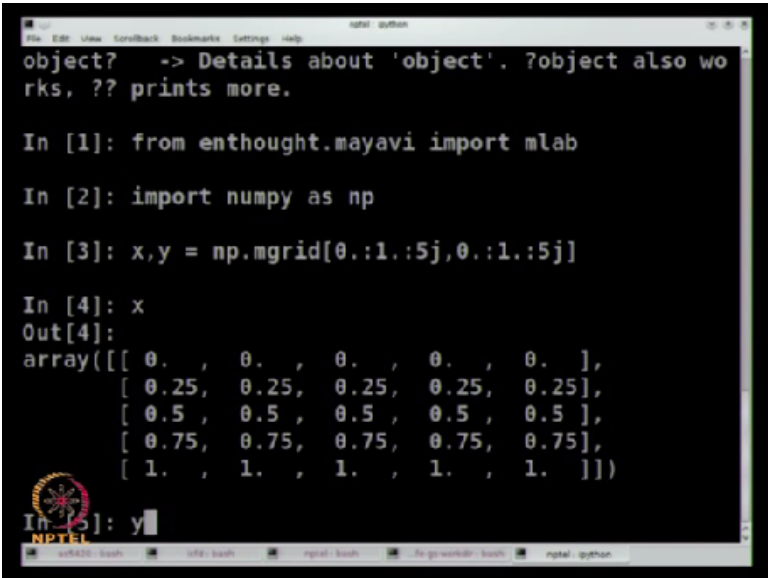
The other part, as far as programming goes, before I get into the demo itself, because of the nature of this course, you are writing all your programs, right. I encourage you to write your programmes in C, C++, or Fortran and so on but when you eventually get to code development, right, so the way I do my code development is very often I start my initial code development in python.

Is that fine? Right and then as and when, so if I have my program functionally correct, program is functionally correct, you understand what I am saying it functions properly, then I start through a process of writing either C routines, Fortran routines, or C++ routines to replace elements of that python program with something that runs faster as required. So as a consequence, I am able to sit when I am in this process.

I am always able to interact through python, interact with my program, always interact and if it is batch, then I use my python again to write a script to run the batch program. That is the idea. So in essence using python as a programming language to secure large programs that is the, that is what you are looking for, that is the ability that you want to get to right at some point in the future.

But right now, my suggestion is if you want to try to do that, do that but my suggestion is keep writing programs in C, C++, Fortran till you, make sure that you are totally comfortable doing it, okay. Let us get on with it.

(Refer Slide Time: 07:22)



```
object? -> Details about 'object'. ?object also works, ?? prints more.

In [1]: from enthought.mayavi import mlab

In [2]: import numpy as np

In [3]: x,y = np.mgrid[0.:1.:5j,0.:1.:5j]

In [4]: x
Out[4]:
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. ]])

In [5]: y
```

So I am going to import numpy because I need, I need an array, right. So I will, just like last time, I will create a 5*5 grid going from 0.0 to 1.0, I think by now the syntax is most probably I hope familiar to you, 0.0, 1.0 and that should create for me a 5*5 mesh, okay. That should create for me a 5*5 mesh.

(Refer Slide Time: 08:02)

```
npml - python
File Edit View Console Back Bookmarks Settings Help

In [4]: x
Out[4]:
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. ]])

In [5]: y
Out[5]:
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 [6]:
```

Those are the 2 things that we looked at in the last class.

(Refer Slide Time: 08:06)

```
npml - python
File Edit View Console Back Bookmarks Settings Help

In [6]: phi = x*x - y*y

In [7]: phi[-1:1,-1:1] = 0.

In [8]: phi
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. ]])

In [9]: phi[1:-1,1:-1] = 0.
```

And of course as a consequence just like I did last time, I will say phi is $x*x-y*y$, okay, right and we will, we will set to 0, okay and of course at this font size, it looks a bit funny but you can see that. I do the right thing? Phi of -1,1:1-1:1, oh, sometimes since that does not make sense, we will do the right thing, right. That is what I was telling you. This is what I was telling you. The reason why I do this is, yes I make mistakes. We all make mistakes, right.

(Refer Slide Time: 09:19)

```

In [9]: phi[1:-1,1:-1] = 0.

In [10]: phi
Out[10]:
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 [11]: 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:

```

I am not trying to make it a, I am not trying to make it perfect. Now you can see that all the interior points are 0. It is fine? It is okay? Let me just type up, because this, this helps me at a later point. So phi of 1:-1,1:-1=what is it? Today I will get a little lazy, 0.25* what we have? 0:-2, that shifted to the left, + shifted to the right, + shifted up or shifted down, + that shifted up, fine.

(Refer Slide Time: 10:42)

```

In [12]: phi = x*x - y*y

In [13]: phi[1:-1,1:-1] = 0.

In [14]: ErrJ = []

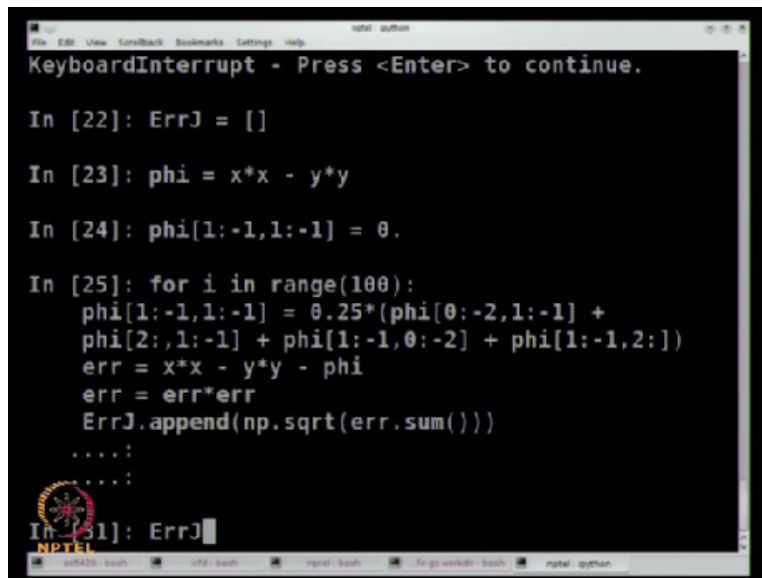
In [15]: for i in range(100):
...:     phi[1:-1,1:-1] = 0.25*(phi[0:-2,1:-1] +
...:     phi[2:,1:-1] + phi[1:-1,0:-2] + phi[1:-1,2:
...:     err = x*x - y*y - phi
...:     err = err*err
...:     Err.append(np.sqrt(err.sum()))
...:
...:

```

So that in fact will give me, that in fact will give me the first iteration, okay. Is that fine? Everyone? So we can, and we can go through this iteration. What if I wanted to do Gauss-Seidel instead? So you can go through this iteration. What I will do is I will quickly, I will just quickly redo this. So I reset my phi, I set it = 0, okay because I, I need to store, I need to store my error, error, shall I say Jacobi= that, okay, fine. How many iterations shall I do?

“Professor - student conversation starts” 10. 10 iterations, 50 iterations? 50 iterations. Or let us do a 100 iterations just to be. **“Professor - student conversation ends”** I in range 100, okay. I hope I have got everything right. So I will just copy, copy that out, see if that works. Whatever has to be deleted, delete, okay. Error is $x^2 - y^2 - \phi$, ϕ i, is that fine? I will just square it. I will take the sum. So `err.append`, fine. If possibly even take the square root of that, `np.square root`, sum of the squares. Is that fine? This should give me what I want.

(Refer Slide Time: 13:09)



```
KeyboardInterrupt - Press <Enter> to continue.

In [22]: ErrJ = []

In [23]: phi = x*x - y*y

In [24]: phi[1:-1,1:-1] = 0.

In [25]: 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:])
    err = x*x - y*y - phi
    err = err*err
    ErrJ.append(np.sqrt(err.sum()))
    ....:
    ....:
In [31]: ErrJ
```

And I made a mistake. Err J, yes, so when you get clever that is what happens. Err J, let me set up my phi again, please bear with me. Err J, okay, that is fine, okay.

(Refer Slide Time: 13:49)

```
ErrJ.append(np.sqrt(err.sum()))
....:
....:

In [31]: ErrJ
Out[31]:
[0.30618621784789724,
 0.10825317547305482,
 0.038273277230987154,
 0.013531646934131853,
 0.0047841596538733943,
 0.0016914558667664816,
 0.00059801995673417429,
 0.0002114319833458102,
 7.4752494591771786e-05,
 2.6428997918226276e-05,
 3.40618239714732e-06,
 3.3036247397782844e-06,
```

So err J seems to have lot of 0s. They are way up there somewhere, so obviously the suggestion for 10 or 15 iteration was pretty good, pretty good suggestion and the interesting thing is that the error afterwards is 0, is identically 0 okay. So it is actually possible that 2 iterates get the same, it converges to the same value. You need to think about what that means in this context. So what about Gauss-Seidel? I want to do Gauss-Seidel. What would Gauss-Seidel give me.

(Refer Slide Time: 14:18)

```
In [32]:

In [33]: ErrG = []

In [34]: phi = x*x - y*y

In [35]: phi[1:-1,1:-1] = 0.

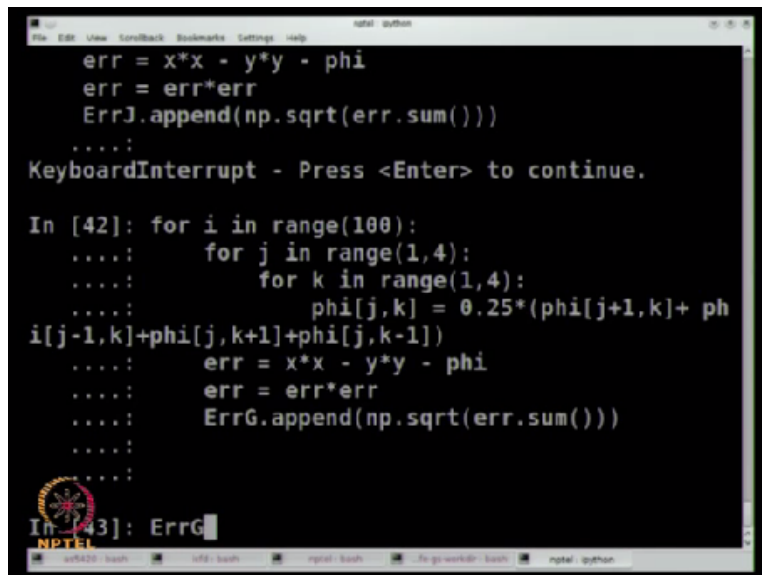
In [36]: for i in range(100):
    phi[1:-1,1:-1] = 0.25*(phi[0:-2,1:-1] +
    phi[2:,1:-1] + phi[1:-1,0:-2] + phi[1:-1,2:])
    err = x*x - y*y - phi
    err = err*err
    ErrJ.append(np.sqrt(err.sum()))
....:
KeyboardInterrupt - Press <Enter> to continue.

In [42]:
```

I set phi x phi as x squared-y squared, so I, let me create err g for Gauss-Seidel. I set phi as x squared-y squared. I set the, make sure that the initial guess is right, okay and then Gauss-Seidel, what changes? Something here changes. So in Gauss-Seidel, I will have, I cannot do that phi i, you know I cannot do that 1 blah blah blah. So I have to, let me, let me, I have to actually iterate

through the whole thing, right. I have to actually iterate through the whole thing.

(Refer Slide Time: 15:13)



```
err = x*x - y*y - phi
err = err*err
ErrJ.append(np.sqrt(err.sum()))
....:
KeyboardInterrupt - Press <Enter> to continue.

In [42]: for i in range(100):
....:     for j in range(1,4):
....:         for k in range(1,4):
....:             phi[j,k] = 0.25*(phi[j+1,k]+ phi
i[j-1,k]+phi[j,k+1]+phi[j,k-1])
....:             err = x*x - y*y - phi
....:             err = err*err
....:             ErrG.append(np.sqrt(err.sum()))
....:
....:
In [43]: ErrG
```

So what I will do is, let me just type it up. For i in range 100, yes, what we want to do now? For j in range 1,4, for k in range 1, 4. If you have already done this, this is most probably a hassle but anyway it is okay. Phi of i, j=0.25*phi of, okay. As I said this is most probably the last time I will bother you by writing this j, k. I would say wakeup Ramakrishna, j+1,k+phi of j-1, k. I will have to +phi of j,k+1+phi of j,k-1. Is that okay? Everyone? And it is updated.

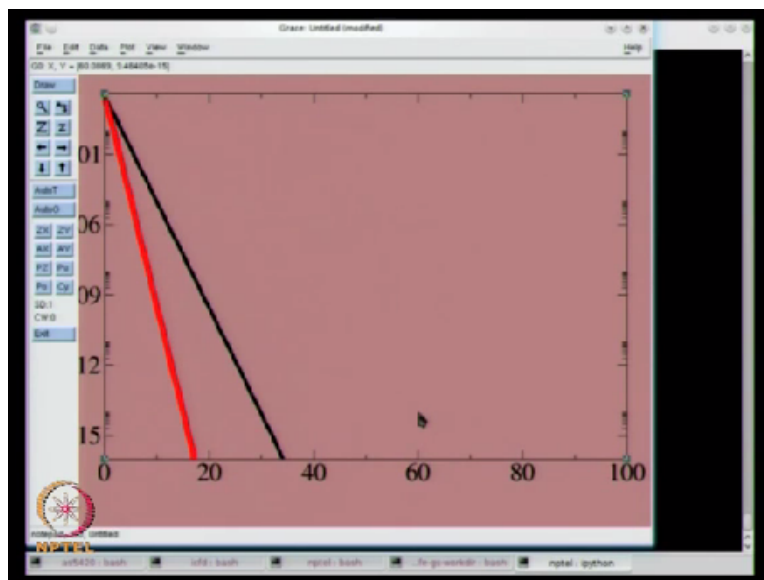
I am using the same array, so it is updated immediately, okay. So that is as far as that loop was concerned. Do I need to do anything else there? I do not need to do anything else there. So that is for the first forward loop. That is for the second forward loop. So err=x*x-y*y-phi. I am just copying from above. As I suggest bear with me. I do not want to make a mistake here, np.square root of what? Is that fine? Okay. So now we can plot these 2 and see what they look like.

(Refer Slide Time: 18:05)

```
0.0,  
0.0,  
0.0,  
0.0,  
0.0,  
0.0,  
0.0,  
0.0,  
0.0,  
0.0,  
0.0,  
0.0]  
  
In [44]: from gracePlot import gracePlot  
  
In [45]: g = gracePlot()  
  
In [46]:
```

If you want to just err G, lot of 0s, okay. We can plot these 2. So from grace plot, import my grace plot. I create the plotting utility. That is all.

(Refer Slide Time: 18:25)

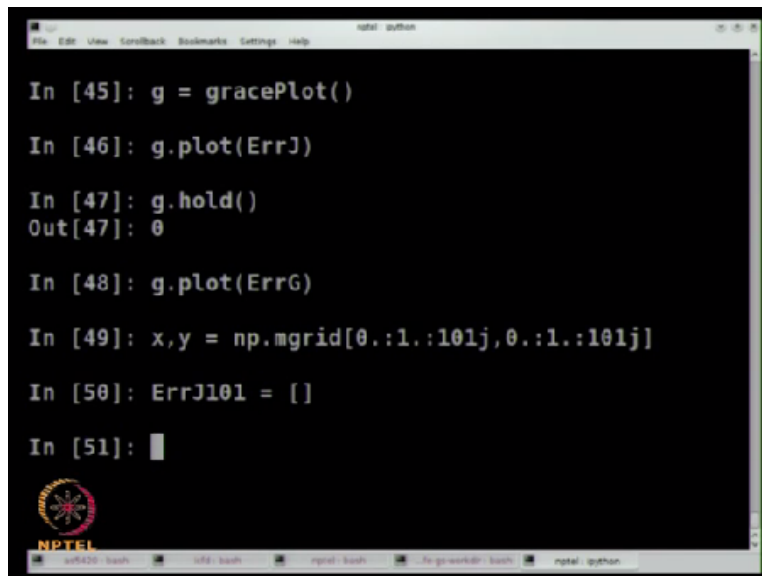


What do I want to plot? Plot..., okay. 2 things very similar as I have always said, use the log scale. So go to a log scale, apply, I do not know why it went into 10 power -4. $1E-16$. Even 10 power -16, it seems to just okay, fine and as I have indicated earlier, as I have indicated earlier, Gauss-Seidel converges faster than, right. Gauss-Seidel converges faster than Jacobi iteration and it is sort of if you squint at it, it does look as though it is twice as fast.

On a log scale, the slope does look as though it is twice as fast. Is that fine? Is that okay? Right.

So I am not going to say how far down it goes. It is going to suddenly drop to 0, right, that was very clear. So one other things of course what I normally do is, I do not just allow the program to run in the background. I usually plot this graph while the program is running so that you only look at the graphical output rather than. What I will do is, why do not we run a 101×101 or something of that sort. Run a big, a big one.

(Refer Slide Time: 20:14)



```
In [45]: g = gracePlot()
In [46]: g.plot(ErrJ)
In [47]: g.hold()
Out[47]: 0
In [48]: g.plot(ErrG)
In [49]: x,y = np.mgrid[0.:1.:101j,0.:1.:101j]
In [50]: ErrJ101 = []
In [51]:
```

So I will recreate this. So I will just basically say x, y is... 101. What do you want? 101 is fine? 101. 101. So that gives me a 101×101 grid, okay and everything else should actually work as it is because there is nothing in there that was specific to the number of grid points. Except for the Gauss-Seidel, we have to be a bit careful. So what I will do is I will make this err J=, I will just overwrite, it does not matter or maybe you can make it 101, either way.

(Refer Slide Time: 21:17)

```

In [49]: x,y = np.mgrid[0.:1.:101j,0.:1.:101j]

In [50]: ErrJ101 = []

In [51]: phi = x*x - y*y

In [52]: phi[1:-1,1:-1] = 0.

In [53]: 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:])
    err = x*x - y*y - phi
    err = err*err
    ErrJ101.append(np.sqrt(err.sum()))
    ....:
    ....:
In [59]:

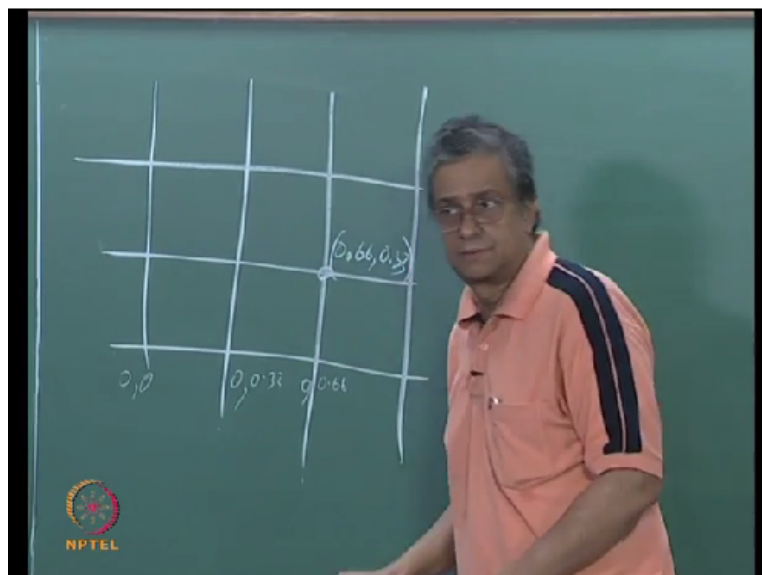
```

Go back to the, go back to the Gauss-Seidel, I am sorry, Jacobi iteration. So that should do Jacobi iteration first. Is that fine? J should be 101. **“Professor - student conversation starts”** Is that fine? Thank you. Clear, okay. Any questions? Something? I mean x and y is mgrid. Why do you get such a matrix one where in column 0 to 1, other one rows. Well that basically if you think about the xy coordinate system, at any given point, I have an x, y, right.

So that mgrid is essentially generating for me the array of x's and y's that represent the mesh, underlying grid. Is that make sense? I can go to the board. It is fine? We will see if it shows, yes.

“Professor - student conversation ends”

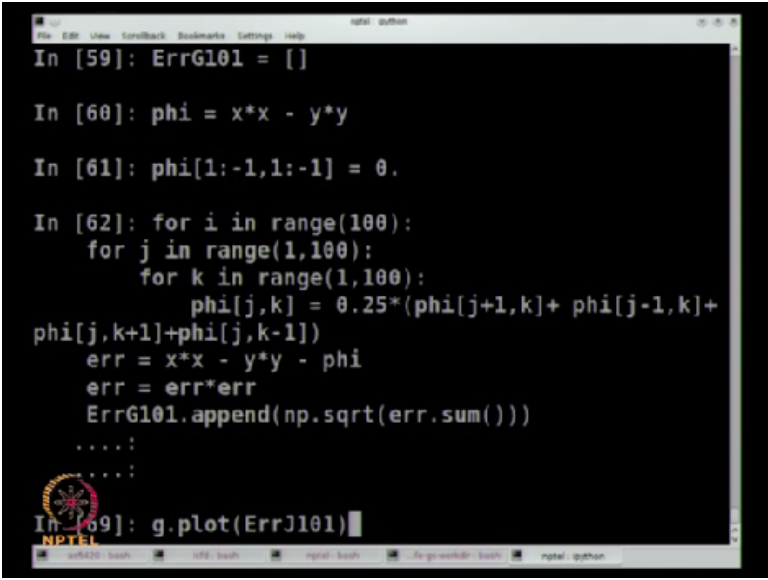
(Refer Slide Time: 22:45)



So essentially what we are doing is, no, no it is fine. It is okay. Essentially what we are saying, essentially what we are saying is, so this is 0,0. This is what I am generating, right. So this is 0, in this case 0.3, 33, one-third, 0,0.66 approximately, right. So the x matrix consists of all the x's and the y matrix consists of all the y's, right. So if you are, if you are somewhere in between, so this would be 0.66,0.33, so on.

So the x matrix consists of all the x entries and the y matrix consists of all the y entries, okay. That is basically what it does do. Is that fine? Right, yes. Sorry about that, anyway we will. So let us get back to this. I think this code is fine as it stands. So if I run this now, it should give me, okay and we did this a 100 times. Is there any chance, anything would have happened, 100 times? You want to run it longer? Okay. Why do not we, why do not we repeat the process.

(Refer Slide Time: 24:12)



```

In [59]: ErrG101 = []

In [60]: phi = x*x - y*y

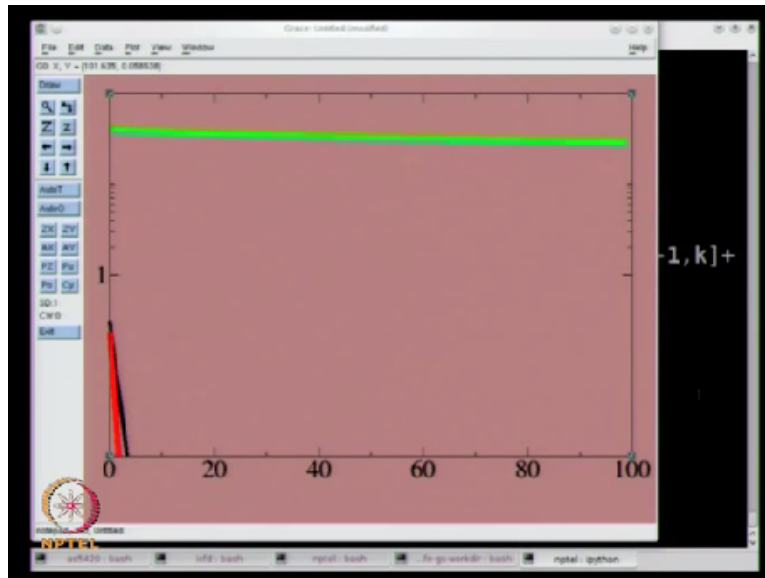
In [61]: phi[1:-1,1:-1] = 0.

In [62]: for i in range(100):
    for j in range(1,100):
        for k in range(1,100):
            phi[j,k] = 0.25*(phi[j+1,k]+ phi[j-1,k]+
phi[j,k+1]+phi[j,k-1])
        err = x*x - y*y - phi
        err = err*err
        ErrG101.append(np.sqrt(err.sum()))
    ....:
    ....:
In [69]: g.plot(ErrJ101)

```

I will just make this, I will just make this G, repeat the process for Gauss-Seidel and Gauss-Seidel you remember I have to change quite a few things. I have to make this errG101 and I have to change the limits to 100. Is that fine? Okay. And that takes a little longer because now we are sort of looping through. I am not using the inherent iterations capabilities of numpy or the matrix abilities of numpy but doing it myself manually in Python, okay. So g.plot errJ101, it has non-positive values.

(Refer Slide Time: 25:23)



What happened?

(Refer Slide Time: 25:30)

```

In [60]: phi = x*x - y*y

In [61]: phi[1:-1,1:-1] = 0.

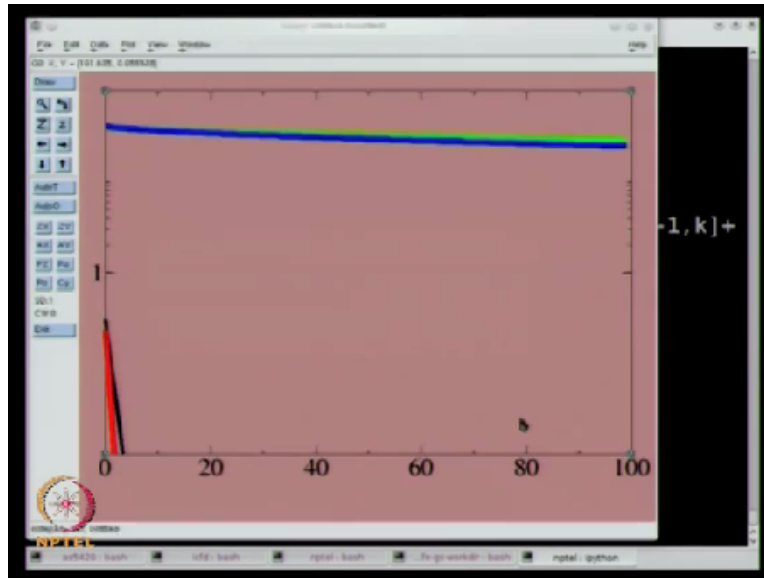
In [62]: for i in range(100):
          for j in range(1,100):
              for k in range(1,100):
                  phi[j,k] = 0.25*(phi[j+1,k]+ phi[j-1,k]+
phi[j,k+1]+phi[j,k-1])
                  err = x*x - y*y - phi
                  err = err*err
                  ErrG101.append(np.sqrt(err.sum()))
          ....:
          ....:

In [69]: g.plot(ErrJ101)
In [70]: g.plot(ErrG101)

```

Let us try errG, maybe I made a mistake somewhere.

(Refer Slide Time: 25:42)



Yes. Both have non-positive values.

(Refer Slide Time: 25:52)

```

In [71]: x.shape
Out[71]: (101, 101)

In [72]: x[50,50]
Out[72]: 0.5

In [73]: phi = x*x - y*y

In [74]: ErrJ101 = []

In [75]: phi[1:-1,1:-1] = 0.

In [76]:

```

What is x? X is 101*101 matrix. I just wanted to make sure that was right. How was that possible? How is it possible for the square root to get a non-positive value? It does not make sense. Let us try it that one more time. Make sure I have not made a mistake anywhere. I have initialised phi, I set it = 0.

(Refer Slide Time: 27:13)

```

Out[72]: 0.5

In [73]: phi = x*x - y*y

In [74]: ErrJ101 = []

In [75]: phi[1:-1,1:-1] = 0.

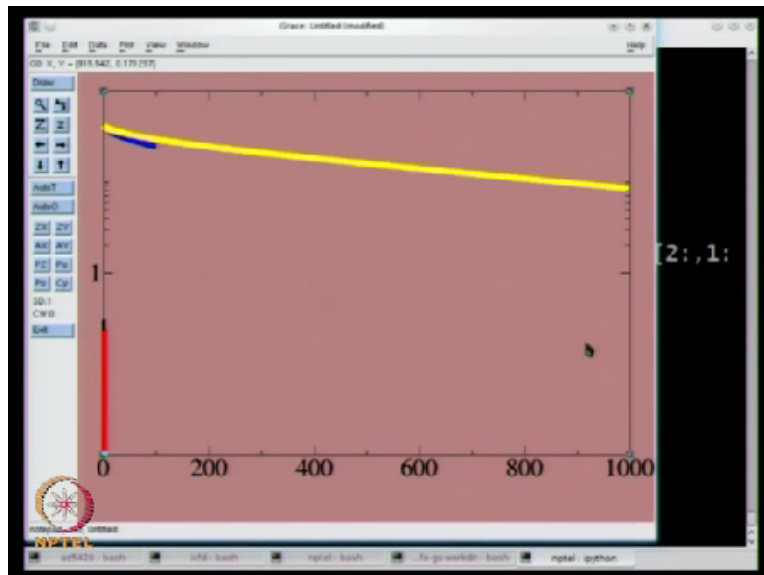
In [76]: for i in range(1000):
    phi[1:-1,1:-1] = 0.25*(phi[0:-2,1:-1] + phi[2:,1:-1]
    + phi[1:-1,0:-2] + phi[1:-1,2:])
    err = x*x - y*y - phi
    err = err*err
    ErrJ101.append(np.sqrt(err.sum()))
    ....:
    ....:

In [77]: g.plot(ErrJ101)

```

It will do a 1000 iterations but of course that should make a difference. Just in case there is some issue there. I will just do that. As you can see even a 1000 iterations if you use, numpy, it is quite fast and let me plot that, just to make sure that I am not trying to, I am going to get the same thing, then we will just have to go on.

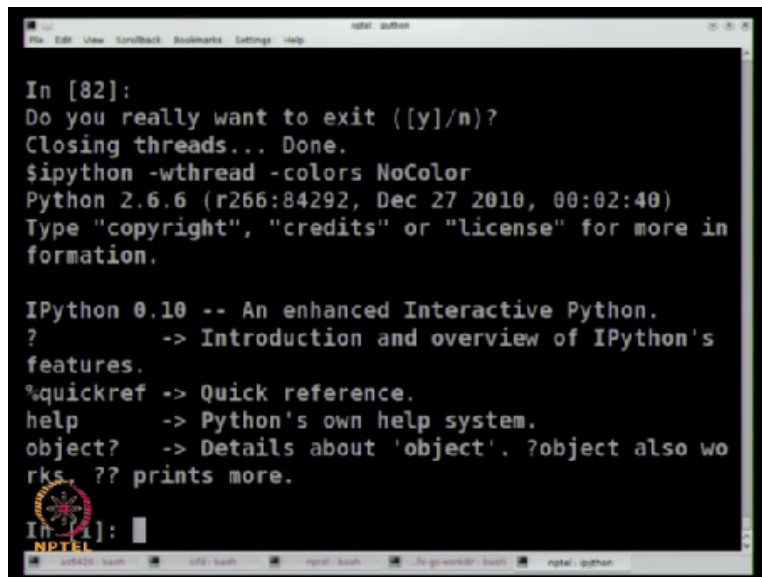
(Refer Slide Time: 28:13)



Data has non-positive value. I still do not get it but anyway it is okay. It does not matter. The point that I want to, the real, really the critical point that I want to make out here, right, this first thing, the blue line which is Gauss-Seidel, does seem to converge faster still than Jacobi iteration. One thing good. But the bad thing is which we have, which is what we expect that the rate of convergence is quite slow, right.

Look at what 5×5 is doing down here and look at what my Gauss-Seidel is doing above, Jacobi is doing up there. It is going to take forever for it to get down to a small enough value that is 10^{-6} or 10^{-16} is what we did in the other one. I am making sense? Is that okay? Right. So convergence is actually extremely slow.

(Refer Slide Time: 29:12)

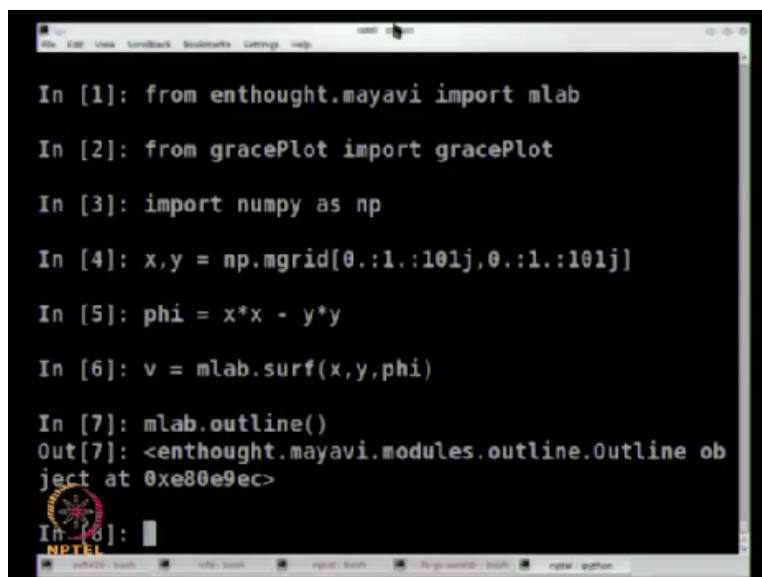


```
File Edit View Scrollback Bookmarks Settings Help
In [82]:
Do you really want to exit ([y]/n)?
Closing threads... Done.
$python -wthread -colors NoColor
Python 2.6.6 (r266:84292, Dec 27 2010, 00:02:40)
Type "copyright", "credits" or "license" for more in
formation.

IPython 0.10 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's
features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object'. ?object also wo
rks. ?? prints more.
In [1]:
```

So just as, let me, let me do than a, I will just restart this as to be clear that there are no threads running and so on.

(Refer Slide Time: 29:18)



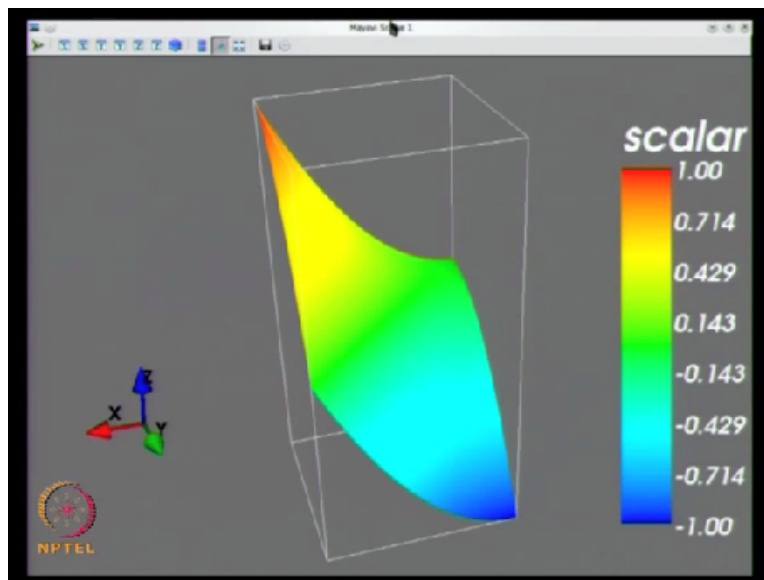
```
File Edit View Scrollback Bookmarks Settings Help
In [1]: from enthought.mayavi import mlab
In [2]: from gracePlot import gracePlot
In [3]: import numpy as np
In [4]: x,y = np.mgrid[0.:1.:101j,0.:1.:101j]
In [5]: phi = x*x - y*y
In [6]: v = mlab.surf(x,y,phi)
In [7]: mlab.outline()
Out[7]: <enthought.mayavi.modules.outline.Outline ob
ject at 0xe80e9ec>
In [8]:
```

Let us just now do SOR or something of that sort, something little different, okay, right and we

will try to see whether we can animate the code while it runs. So from, maybe I could have done it with the earlier thing but it does not matter. It is okay, just to be, fine, right, okay. We will, we will do, maybe we will do one other thing. What we will do is we will see how these iterations go.

I just wanted to show you that in the last class I used Mayavi to draw contour plots. I just wanted to show you that you can actually do surface plots, okay. So why do not I, why do not I just do that. So I am going to do, redo the 101×101 grid. I created the 101×101 grid, right. Φ is $x^2 - y^2$. I do not know how many of you have tried using Mayavi so far but it is relatively straightforward to use. So what I do is I say `v=mlab.surface x,y,phi`. This is to show you what the initial solution looks like.

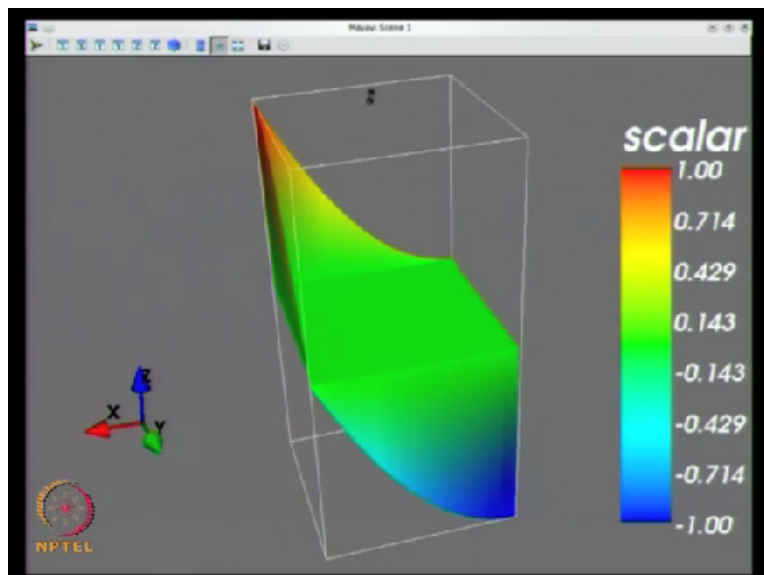
(Refer Slide Time: 31:04)



Yes, that should generate something that looks like that, like what is that? One of the things that I like about, one of the things that I like about Mayavi is that you can interact with it, okay. So let me maximize it or maximise it, okay. I am going to add; I am going to add legend to it. Show legend, so that we can see what it is. So it goes from -1 to +1 and the coordinate system is at the bottom left-hand corner. If you cannot see the coordinate system, I will make that larger, okay.

The coordinate systems is at the bottom left-hand corner, right, fine. So this is a solution that we are expecting, right. So if you are saying, wait a minute. What does this, what is this, okay let me

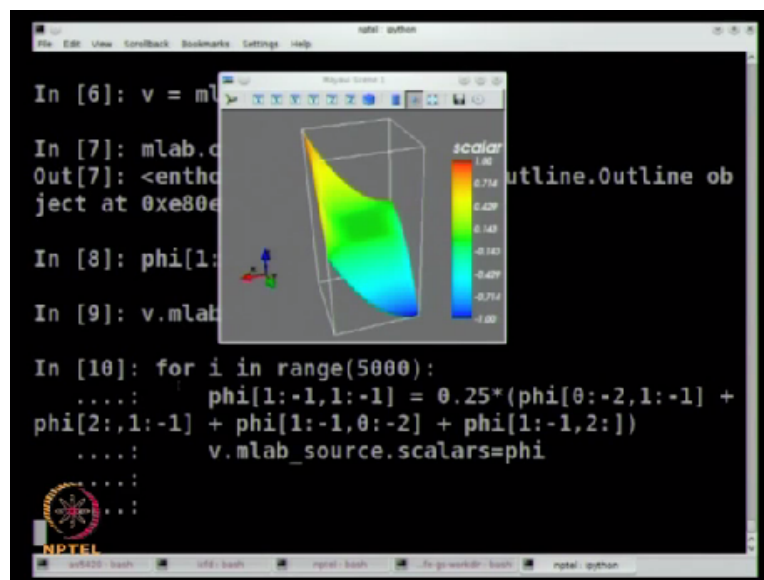
(Refer Slide Time: 32:40)



That is our initial condition, okay or initial condition except at the boundary where it is $x^2 + y^2$ squared, everywhere else it is 0. That is our initial condition. Is that fine? Everyone? Okay? So now, now that we know Gauss-Seidel and Jacobi, we can actually see the solution evolve. So what we will do is I will say, let us do Gauss-Seidel. Let us do Jacobi, okay. For i in range, Jacobi is faster, right. Gauss-Seidel is slower which for the sake of a demo is better but anyway.

How many, how many iterations do you want me to do? At the rate at which it was going, most probably 5000 iterations or something is most volume. 5000 iterations, okay, fine. Let us see if you can find the Jacobi control or phi, I will just switch back, okay. That is Jacobi iteration. Is that fine? Everyone? Okay. So and I am going to plot only the phi. So if you have computed the phi v.mlab scalars at phi. Let us run it and you can see solution evolving and my computer is having problem.

(Refer Slide Time: 34:37)



It is not going to rescale till I get to the last version, okay, that is fine. So all you can make out, so the 5000 is quite a lot. All you can make out is that flat portion seems to be diminishing at a rather slow rate and it is really going to take for almost ever, in fact on the, on the video itself, I do not know whether this will be visible. This is going to take forever to, it is going to take forever to converge.

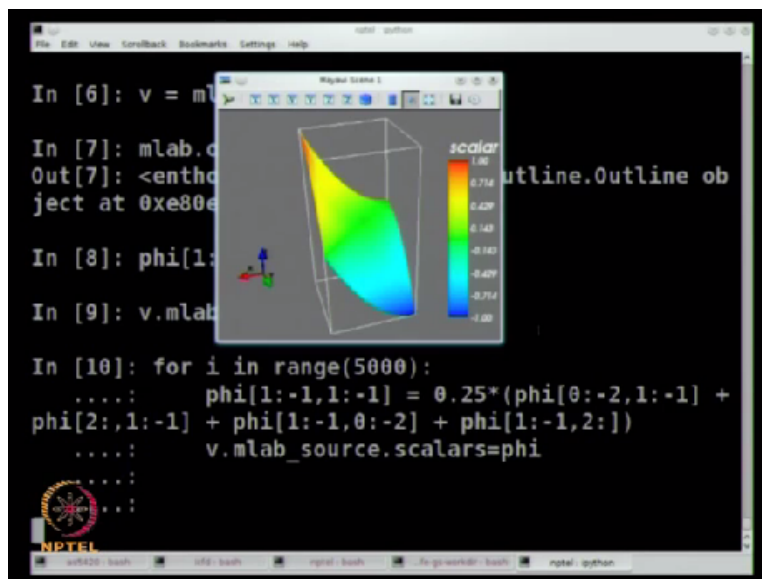
So it looks like even 5000 may not quite make it, right. You can see that little grey spot in the

middle, you can see that little grey spot in the middle diminishing and is going towards the solution. Remember this is Jacobi, right. Gauss-Seidel will basically run twice as fast, okay and that is basically it. So essentially the thing that you can do is that you can visualise, you can visualise, as I said maybe I should not have minimized it but you can visualise this code running.

As the code evolves, it is actually possible for you to look at the solution, right and remember that just because on the screen, it looks close to the solution does not mean that it is close to the solution. Screen resolution is very small. Is that fine? Okay. So this is, these are some skills that possibly I am not, I am only showing you the tools being used. These are some skills that I think you should try to acquire, that you are able to visualise, a that you are able to visualise your data.

B that you are able to do it while your code is actually running, right. There are sometimes especially if you are going to do it with a scripting language like Python or something of that sort, there is a certain advantage to being able to interact with the code, that you allow it to finish executing certain number of time steps, certain number of iterations, or whatever it does and then come back and interact with the code, okay, fine.

(Refer Slide Time: 36:59)



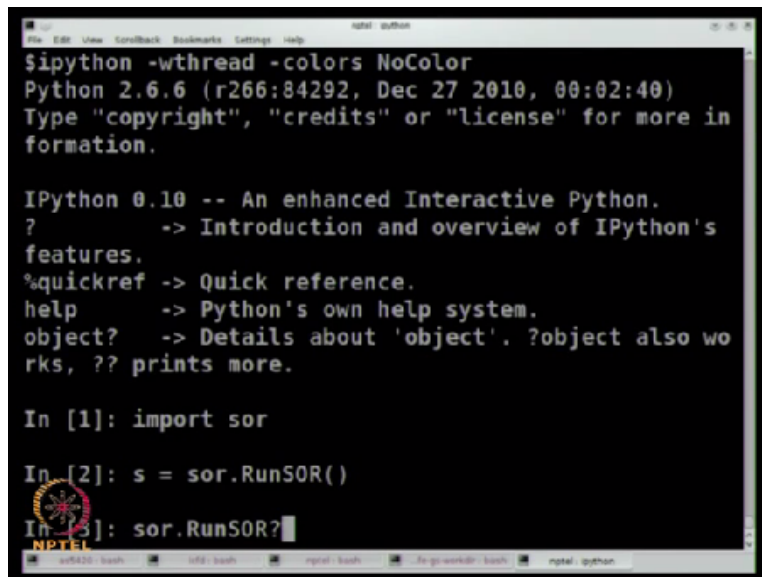
So that patch is almost gone. So actually at 5000, it seems to be quite close. Maybe 5000 was on the large side. Let us see, fine, okay. Meanwhile are there any questions? Let me see if I maximise this whether it will. I give it time, it will. So even if I try to interact with it right now,

because my, because it is completely tight. If you think about it, 101×101 , $10,000^*$, is it there. It is 10,000 unknowns.

You are solving a $10,000 \times 10,000$ system. If you are doing 5000 iterations, that is a lot of calculations. So it is also good idea to do the computation as to how many operations we are performing beforehand before getting in to it like I just did right now. Maybe I should just say 1000 instead of 5000. Anyway it is okay. It does not matter. It should be close to completion.

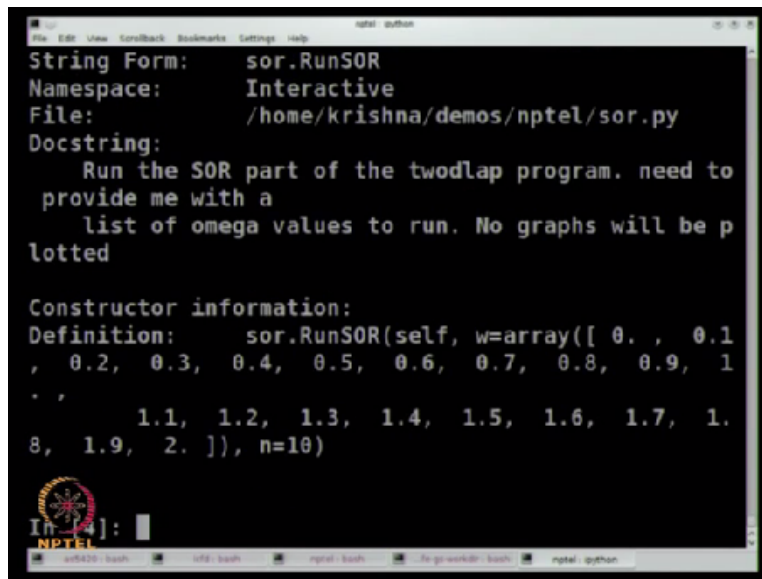
so the other point of course is that if you try to do these 5000 iterations directly in C, if you have already done this, you would most probably seem a lot faster than this, okay. So that is a big advantage of doing it in C or C++ or Fortran. So you pay a price for the ability to interact with the code. I have kept it; you just kill it. Maybe I will just kill it, okay, fine. Let us get back, we will start Python again.

(Refer Slide Time: 38:52)

A screenshot of a terminal window titled 'ipynb - python'. The terminal shows the command '\$ipython -wthread -colors NoColor' and the output 'Python 2.6.6 (r266:84292, Dec 27 2010, 00:02:40)'. It then displays the IPython 0.10 startup message and a list of commands: '? -> Introduction and overview of IPython's features.', '%quickref -> Quick reference.', 'help -> Python's own help system.', and 'object? -> Details about 'object'. Below this, the user enters 'In [1]: import sor', 'In [2]: s = sor.RunSOR()', and 'In [3]: sor.RunSOR?'. The terminal window has a standard menu bar with 'File', 'Edit', 'View', 'Shellback', 'Bookmarks', 'Settings', and 'Help'. At the bottom, there is a status bar with icons for various file types and a small 'NPTEL' logo.

Now we will do, we have just enough time to do SOR. So I have a Laplace equation solver that I have already written and SOR basically what it does is, it is going to run SOR for me for that Laplace equation solver. Right now it is hardwired for a 41×41 grid, okay. It is hard-wire for a 41×41 grid. So I will create something that will run SOR for me, okay and it is very, it has very simple, it has very simple way to do it. So what does this do. Let me, let me, what is runSOR do. What am I doing?

(Refer Slide Time: 39:44)

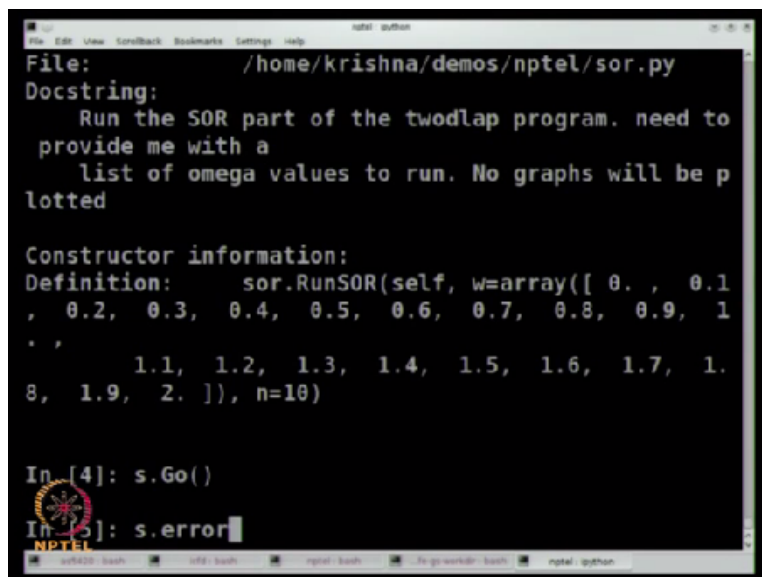


```
String Form: sor.RunSOR
Namespace: Interactive
File: /home/krishna/demos/nptel/sor.py
Docstring:
    Run the SOR part of the twodlap program. need to
    provide me with a
    list of omega values to run. No graphs will be p
    lotted

Constructor information:
Definition: sor.RunSOR(self, w=array([ 0. , 0.1
, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
,
, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.
8, 1.9, 2. ]), n=10)
```

What runSOR does is, it is going to run SOR for the values going from 0.0 to 2., okay and I am taking 20 values in between. It is going to run for 20 values. It is going to take 10 iterations. Is that fine? I think I have told you what you guys out with this. Anyways okay, let us see. Let me just run it and see what we get.

(Refer Slide Time: 40:10)



```
File: /home/krishna/demos/nptel/sor.py
Docstring:
    Run the SOR part of the twodlap program. need to
    provide me with a
    list of omega values to run. No graphs will be p
    lotted

Constructor information:
Definition: sor.RunSOR(self, w=array([ 0. , 0.1
, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
,
, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.
8, 1.9, 2. ]), n=10)

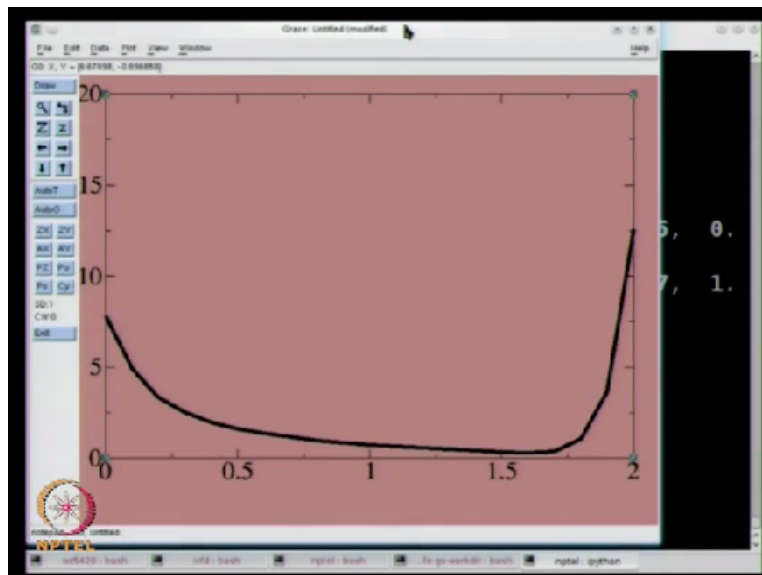
In [4]: s.Go()
In [5]: s.error
```

(Refer Slide Time: 40:24)


```
1.0551776697508599,  
3.6227348122691576,  
12.585077125326182]  
  
In [6]: s.omega  
Out[6]:  
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.  
        7,  0.8,  0.9,  1. ,  
        1.1,  1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.  
        8,  1.9,  2. ])  
  
In [7]: from gracePlot import gracePlot  
  
In [8]: g = gracePlot()  
  
In [9]: g.plot(s.omega,s.error)  
  
In [10]:
```

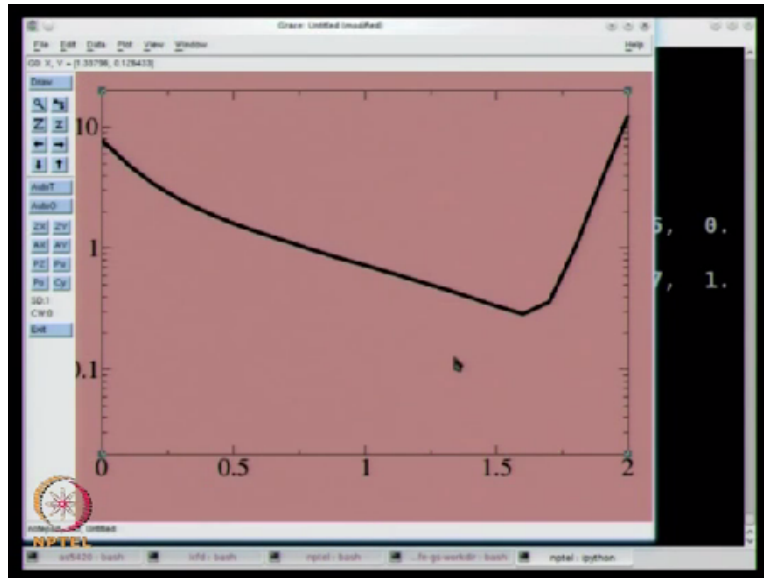
And this s has an error which is the error versus omega, right. I have, s has an omega. This is the omega values that we have and it has a set of errors. Is that fine? Okay? Right. So I will import just to plot it.

(Refer Slide Time: 40:57)



We will just quickly do this, okay and plot, there we go, right and anytime we are plotting error, we really need, this looks relatively flat, okay. This does not look that encouraging but then we go to the power of the log scale. Anytime we plot error, we always look at it at the log scale.

(Refer Slide Time: 41:28)



The error looks that flat now, right. So it looks like whatever we are looking for, this is important by the way. So we have to look at it in the log scale. Whatever we are looking for, the optimal omega value that we are looking for is somewhere there. Is everybody with me? Optimal omega value that we are looking for is somewhere there.

(Refer Slide Time: 41:49)

```

In [10]: g.hold()
Out[10]: 0

In [11]: import numpy as np
In [12]: w = np.r_[1.5:1.9:10j]
In [13]: s1 = sor.RunSOR(w)
In [14]: s1.Go()
In [15]: g.plot(s1.omega,s1.error)
In [16]: s1.GoMore()
In [17]: g.plot(s1.omega,s1.error)
In [18]: 

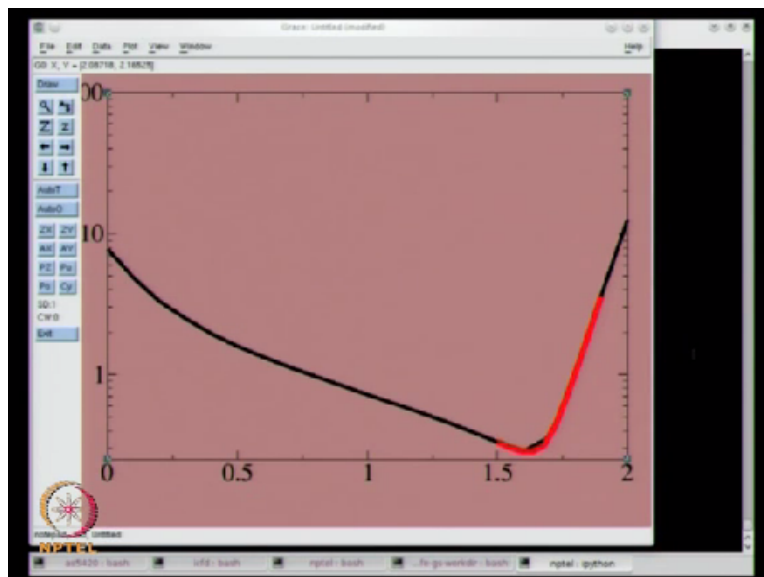
```

So let me hold that graph because I want to plot on top of it, okay and I will create another SOR solver for me, okay but okay, let us go back to that graph. Now I want to hunt. I want to try out another set, okay. So what value shall I take. What values of omega, where shall I hunt? Look at this graph, where shall I hunt? 1.5 to 2. Well 1.2 you know is not going to, is not going to do it. 1.5 to 1.9 maybe you should, right. Is conservative.

Bear in mind when you are doing this that the convergence that you have is non-uniform convergence, that is what we want, okay. So this limit is going to drift, right. Is that make sense? Does it seem, or it just going to continue down. So 1.9 to, so I say omega, I will say ω_{np} . I will create a vector, syntax is very similar to mgrid, from 1.5 to 1.9 and maybe we will take 10 samples instead of 20, okay. 10 samples, okay.

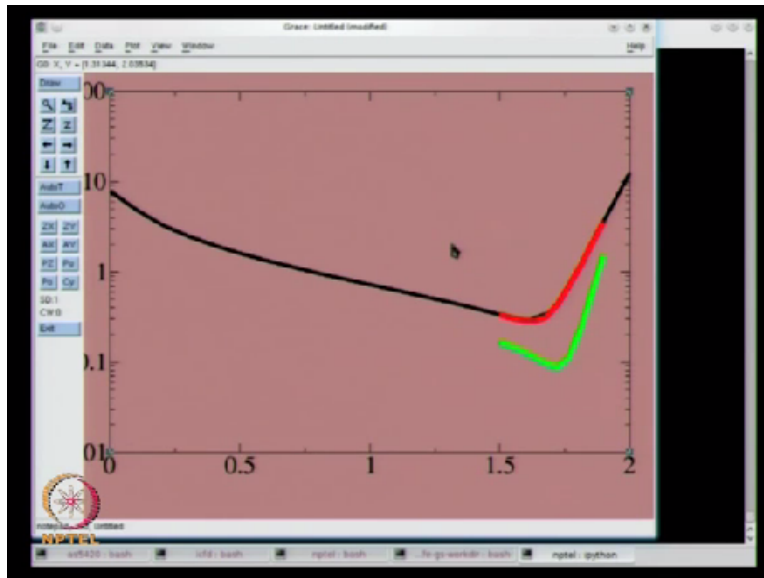
And I will create another SOR solver and pass it this omega. So s1 should run 10 iterations. S1 should run 10 iterations for, s1 also has an error. So I will plot. There is my plot. S1.omega versus s1.error.

(Refer Slide Time: 43:53)



Well 10 iterations what did you expected. When it is going to be the, it is going to be the same thing. The only point is that the curve is smoother because we have more points, that is all that has happened, okay. So I fortunately, I can go more, okay. So I have, I will go for a little more time. I will little go more which will go for a little more time. For 10 more iterations and if I do that. Is it okay?

(Refer Slide Time: 44:27)



Then I can plot that again and what do I get? Ah ha, so it is, it is better and it is a little sharper. So it seems to be somewhere there, okay. It seems to be somewhere there. Is that okay? In fact, what I will do, I will get rid of, I will get rid of some of this, some of this part or let, it is okay. I get rid of some of this part. Will not need, okay, does not look that sharp because I have removed that part but right. So it seems to be around, what do you say? Well, yes, I will be conservative still, 1.65, okay, that is 1.7, 1.65 to 1.85, why risk it, okay.

(Refer Slide Time: 45:15)

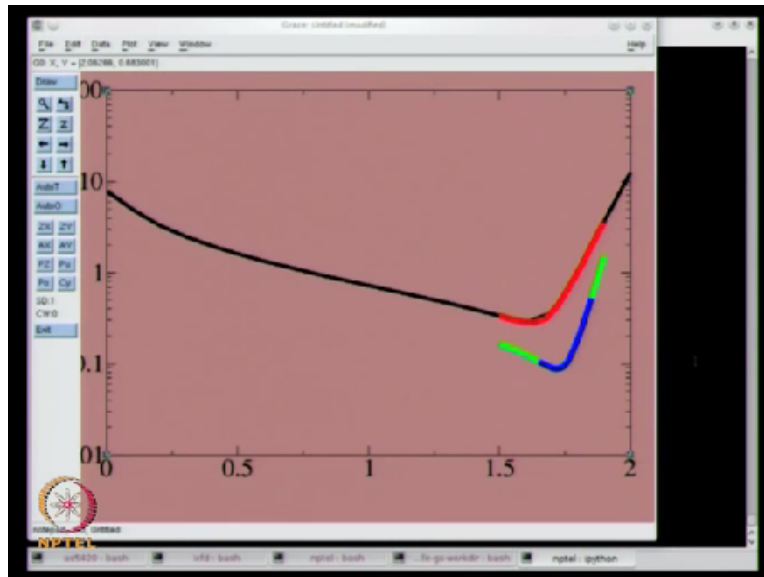
```

In [17]: g.plot(s1.omega,s1.error)
In [18]: w = np.r_[1.65:1.85:10j]
In [19]: s2 = sor.RunSOR(w)
In [20]: s2.Go()
In [21]: s2.GoMore()
In [22]: g.plot(s2.omega,s2.error)
In [23]: s2.GoMore()
In [24]: g.plot(s2.omega,s2.error)
In [25]: 

```

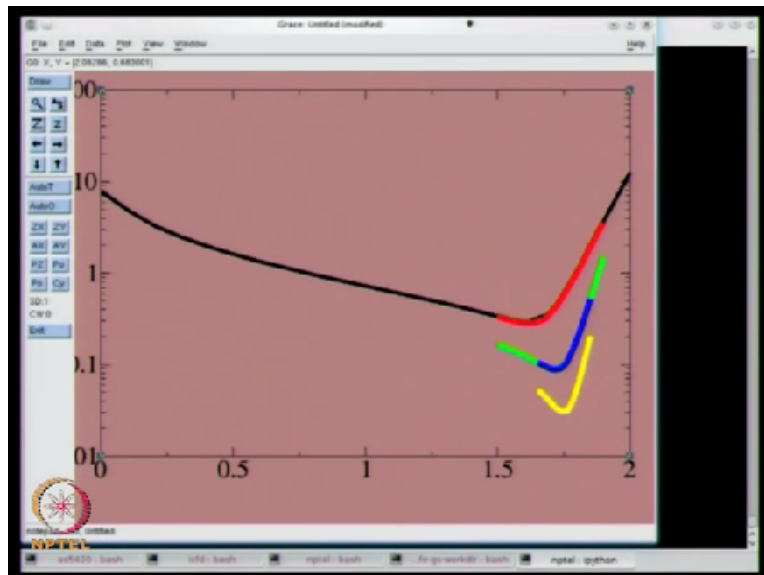
So now I go back and I set this from 1.65 to 1.85. I am doing this because this is a demo. Lot you would do is, you would hunt for it maybe use by section or something, you will hunt for it in a more systematic fashion. The other way is by which we can do this. So now I will create s2, so

that does only 10, that is no, that is no good. So we will do an s2, GoMore and we plot s2.
(Refer Slide Time: 46:07)



As you would expect, it is the same. You have done it twice; it is the same. It is a smoother curve. We will do it some more.

(Refer Slide Time: 46:19)



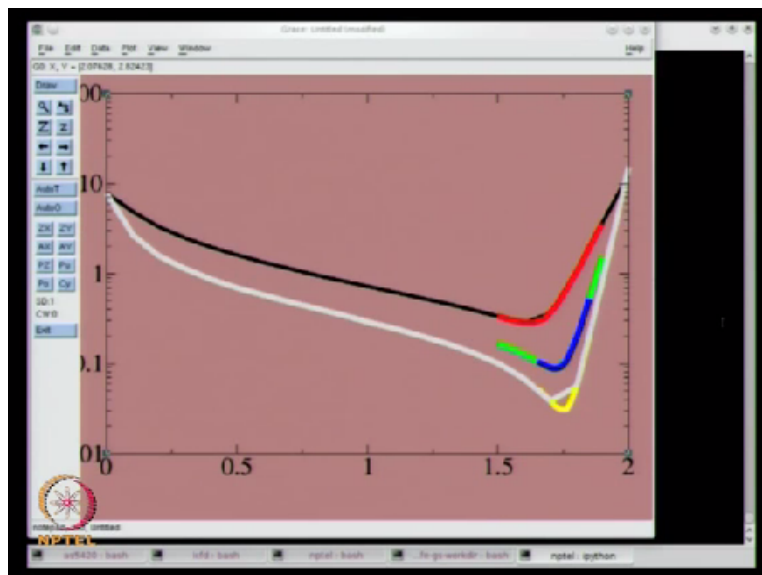
Fine, okay. So you can see it is getting tighter and it seems to be around, between 1.75 and 1.8. So it is possible for you, it is possible for you to actually hunt for an optimal omega, right and trust me, this curve here is not going to move that much, okay.

(Refer Slide Time: 46:46)

```
File Edit View Scrollback Bookmarks Settings Help
In [20]: s2.Go()
In [21]: s2.GoMore()
In [22]: g.plot(s2.omega,s2.error)
In [23]: s2.GoMore()
In [24]: g.plot(s2.omega,s2.error)
In [25]: s.GoMore()
In [26]: s.GoMore()
In [27]: g.plot(s.omega,s.error)
In [28]:
```

If I were to plot, if I were to do just for the fun of it, if I were to run s, remember our friend s, there are 20 of them, so it will take a little more time. Maybe I will do it twice because I did the other one twice, okay.

(Refer Slide Time: 47:16)



If I were to plot this, oh oh, but you do not know what happened, okay. That is what you get. So it is very clear that when I say optimal omega, it is truly optimal omega and look at this. Look what is happening here. You have picked the value because you took 20 samples, you picked the value there, you picked a value here. The optimal is actually somewhere in between. The optimal is actually somewhere in between, okay.

So you can look at this and be under the notion that it is at 1.6. Actually it is not at 1.6. It is because of the way that you are taking the samples, okay. That is an idea. See you are looking at it and you are actually doing some kind of finite difference, that you are looking at it, is increasing here, decreasing there, you are looking at slopes, you are doing derivatives in your mind and you get the impression that the minima by setting the derivative to 0 to somewhere here.

But actually it is somewhere to the right and because of non-uniform convergence, it looks as though it is drifting to that. It is drifting to one side, okay. So the hunt for the optimal, the hunt for the optimal value ω , has to be done a little careful, right but if you are going to make production. I am going to make lots of runs, I am going to solve this Laplace equation for different boundary conditions, you know, I am going to do 10,000 different boundary conditions.

Then spending time finding that optimal ω makes a big difference because where here you are still at 1. something, this is at 0.01 something, right. In fact, it is 0.001 something. Am I making sense? This is just minimum here. Just to give you an idea that 0.003. At $\omega=1$, it is at 0.3. Please remember it is a log scale. At $\omega=1$, the residue is at 0.3. At $\omega=1.8$ or 1.75, the residue is at 0.003.

So running 10 times faster. This is converging. Right now it looks like 10 times faster. Is that fine? Okay and I casually made a suggestion that if you are changing boundary conditions, that you can find the optimal ω and find out, and solve for the various boundary conditions, right. The question that you should pop to me immediately is how do you know that ω does not depend on the boundary conditions.

Do you think that the ω will depend on the right-hand side, the optimal ω will depend on the right-hand side or does it just depend on the matrix a . If you are solving $ax=b$ using SOR, question is does the optimal ω depend on the right-hand side or does it just depend on the matrix a . Think back to that graph that I drew, right in the steepest descent idea that we had which I mistakenly said Newton method at that time, steepest descent idea that I had, okay, fine, okay.

So I guess we will meet in tomorrow's class. I would suggest that you try out some of these things for yourself, okay. We will meet in tomorrow's class. Thank you.