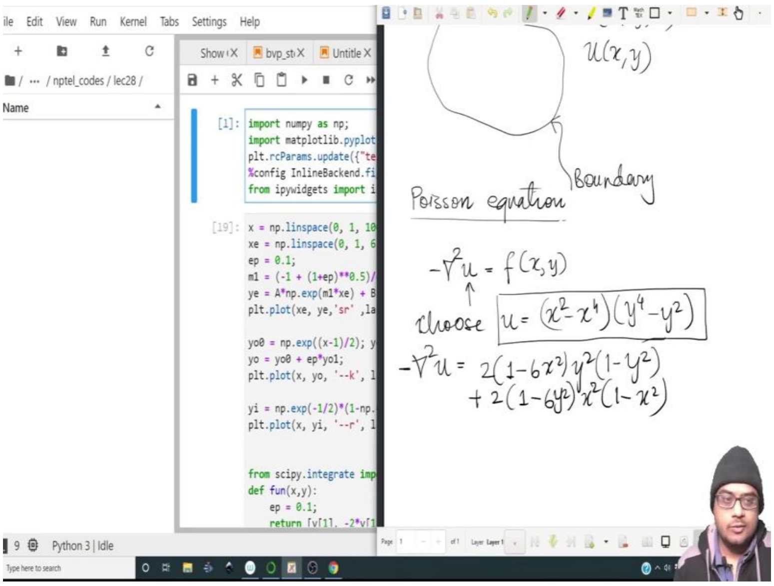
Tools in Scientific Computing
Prof. Aditya Bandopadhyay
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture - 28
2D Boundary Value Problems

(Refer Slide Time: 00:26)

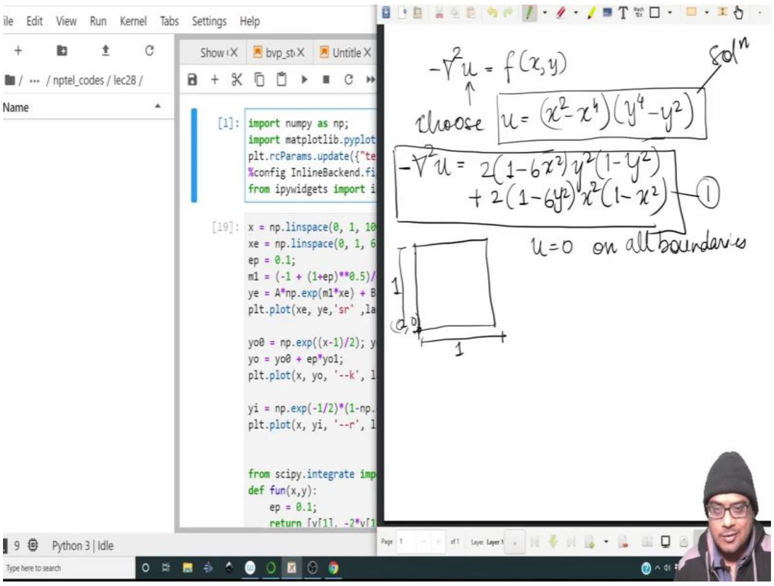


Hi everyone, welcome to this lecture in which we are going to analyze Boundary Value Problem, but in 2 Dimensions; meaning we will have some kind of a domain. There will be some governing equation which will govern how variable say u, which will in general be a function of x, y and t; but if everything is steady, it will be a function of x and y with some specified condition at the boundary.

And in this particular lecture, we are going to be solving a Poisson equation; in particular we will be solving - Laplacian of $-\nabla^2 u = f(x, y)$. But in this particular lecture, since we are going to do it numerically; why do not we select what f is going to be, so that rather why do not we select what u is going to be, based on that we will find out the Laplacian and figure out what the function f has to be.

So, what we will do is, let us choose $u = (x^2 - x^4)(y^4 - y^2)$. So, essentially this particular function is the solution of which equation, where we are doing a back calculation. If u is this, then the Laplacian of u and I have the expression in front of me, it is $-\nabla^2 u = 2(1-6x^2)\,y^2(1-y^2) + 2(1-6y^2)\,x^2(1-x^2)$, alright.

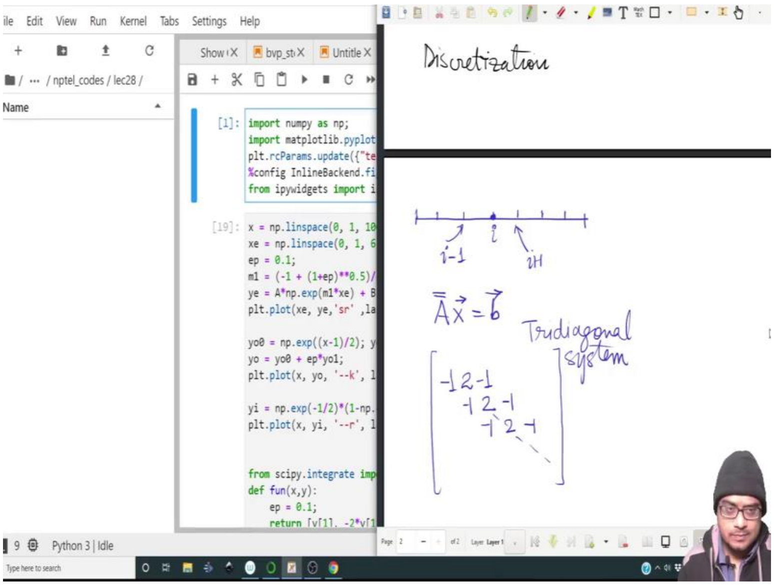
(Refer Slide Time: 02:56)



So, we can imagine that this is the problem, ok. So, this is the governing equation subjected to. So, we have the domain given as a square, this is the origin and the side is 1, alright. So, this equation 1 is subjected to u = 0 on all boundaries and if this is the case, then the solution is given by this.

Well, we have taken some form of view and figure out what the governing equation should be. Well, this is just to eventually match it with the numerical solution. So, yeah that is the problem we have, solving this Poisson equation. So, how do we go about solving this Poisson equation, alright?

(Refer Slide Time: 04:05)

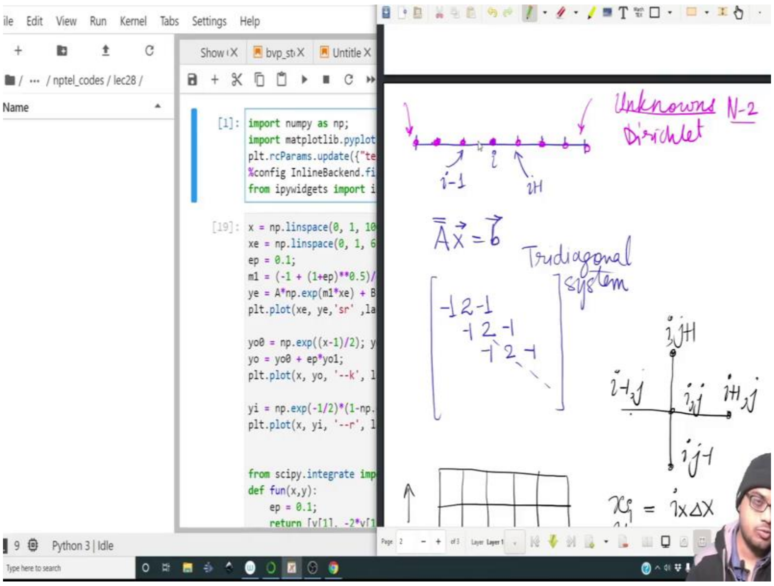


So, first things first, Discretization: So, we have already covered a 1 dimensional problem. So, if you have a 1 dimensional problem, the boundary value problem consists of conditions at two boundaries and you discretize the domain like such; like so. And depending on which point you are focusing on; you can write the form of a derivative at i in terms of i, i + 1 and i - 1, alright.

So, then what we had was a system of equations which resembled something like this and the solution was found by doing an inversion of A and we used this pass solver of scipy to get our job done, alright. And how did the matrix A look like? Well, we recall that it was a tri-diagonal matrix except for the boundaries.

So, if there was a second derivative, there was some diagonal value like 2 and then the off diagonal value was - 1, - 1, then it is 2, - 1, - 1 and so on, ok. So, it was a tri-diagonal system; but what about systems in 2 dimensions? Well, things start becoming slightly more complicated and we will have a penta diagonal matrix. But why is that? Let us see.

(Refer Slide Time: 06:04)

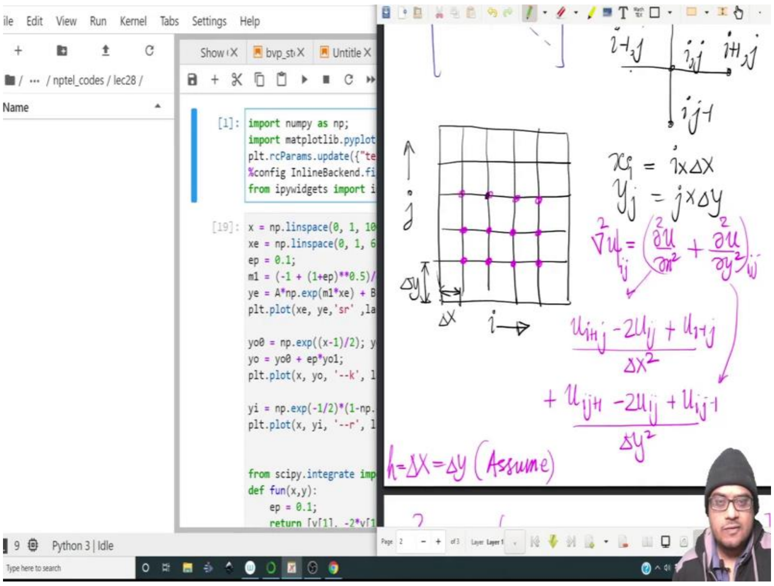


We have a domain like this which is divided into many parts, essentially we have a mesh grid, alright. So, when we do have a mesh grid, we have to number this. So, let us number the columns by i and the rows by j and this is customary, because; because of I will show you why it is customary. So, if we focus our attention at this point, let me zoom in.

So, this is what we have; this is i, j and the location i , j corresponds to some location x i, which will be i $\times\Delta$ X and y j which will be j $\times\Delta$ Y; but Δ X and Δ Y are these grid spacing's, alright. So, now, if this is i, j, this particular thing will be i + 1, j; this will be i - 1, j, this will be i, j - 1, this will be i, j + 1. So, that is how the grid points are. So, let us write down the Laplacian in the discrete form.

So, the $\nabla^2 u_{i,j} = \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)_{i,j}$. This particular term, will be well we can directly write this down; because in one of the previous classes, we have written down what the expression for this particular term should be in the case of an ordinary differential equation. Well we will have $\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta X^2}$.

(Refer Slide Time: 08:42)



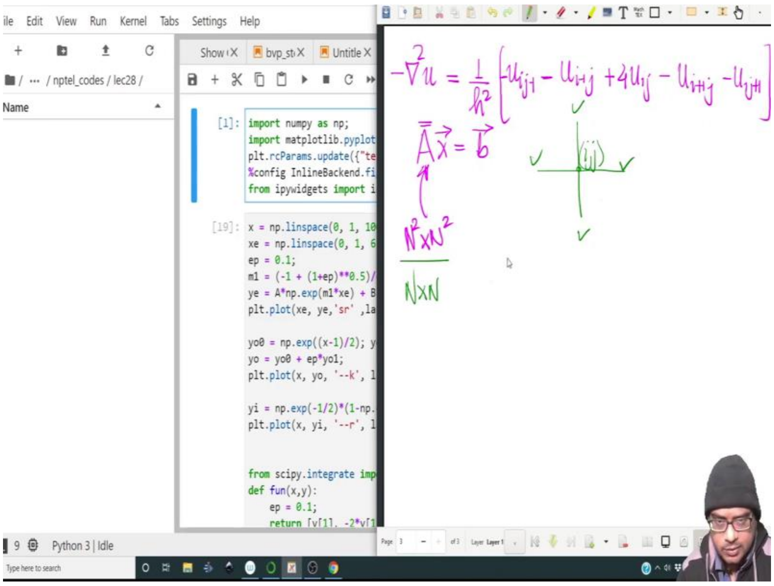
And this particular term will give us $\frac{u_{ij+1} - 2u_{i,j} + u_{ij-1}}{\Delta Y^2}$. Let us recall what the equation is. So, it is $-\nabla^2 u$ equal to this function whatever it is, alright. So, $-\nabla^2 u_{i,j}$ will therefore be 1 upon.

So, let us assume that ΔX and ΔY are equal and it is not a very bad assumption. But anyway if you do have the case where they will be different, you can easily modify whatever I am gonna show without much difficulty. But for now it serves our purpose to do this particular assumption. So, let us assume that it is equal to h. So, 1/ h^2. So, there is a - sign over here, so you multiply everything by - 1.

So, we will have $-\nabla^2 u = \frac{1}{h^2}\left[-u_{ij-1} - u_{i+j} + 4u_{ij} - u_{i+1j} - u_{ij+1}\right]$; this is what we have. But, now in the case of 1 dimension, in the case of a 1 dimensional problem; if we have N nodes, ok. So, the number of nodes are N; then the number of unknowns depending on. So, suppose we have Dirichlet boundary condition at the boundaries; then the number of unknowns will be N - 2, if there are N nodes. But, what about in the case of a 2 dimensional problem?

So, if all the boundaries have a Dirichlet boundary condition; then the number of unknowns will be all the inside grid points, it will be $(N - 2)^2$, alright. So, those will be the unknowns; but rather than just focusing on the interior grids, we will let there be N^2 number of grid points.

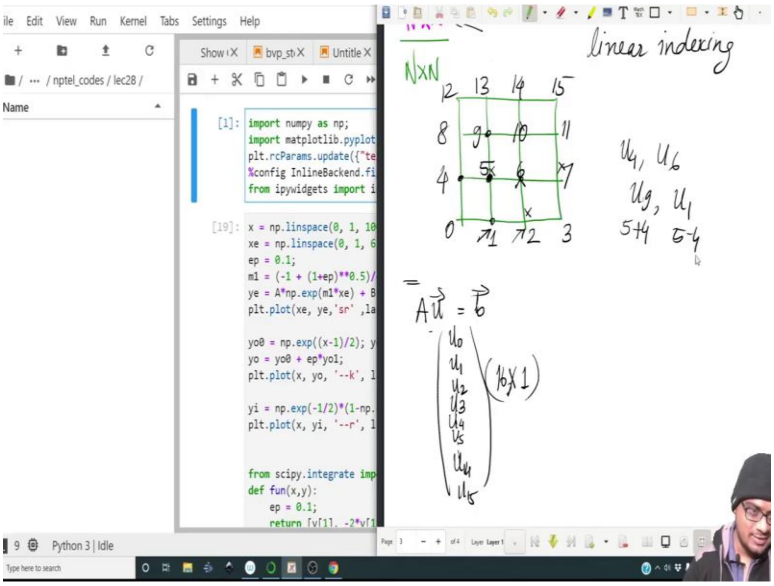
(Refer Slide Time: 11:30)



So, if there are N^2 number of unknowns; if we were to arrive at an equation which would look something like this, the size of the matrix A has to be $N^2 \times N^2$, which is in contrast to the case of a 1 dimensional problem, where the size of the matrix A was only $N \times N$.

And this $N^2 \times N^2$ square arises solely because; we have N^2 number of unknowns that is all the grid points. So, if I were to focus on this, let me just draw the grid again; I have at i, j point contribution from this point, this point, this point and this point, that is from all the nearest neighbors, alright. So, we have contributions from all the nearest neighbors. So, suppose I am focusing my attention on, well ok.

(Refer Slide Time: 12:42)



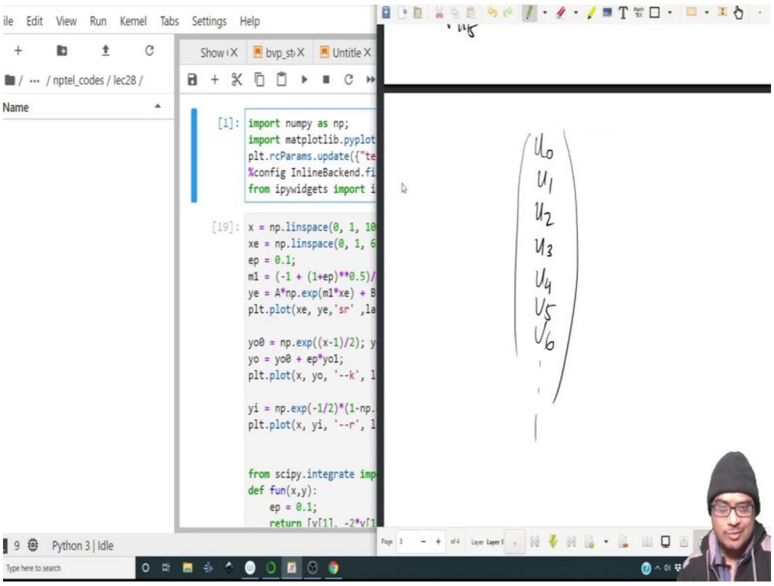
Let me write this, let me take only a few grid points, something like this. So, if I write down the equation for this point, I will have contribution of these four points; if I write an equation for this point, I will have contribution from these four points. I am not discussing about writing an equation for this point or this point, because they are at the boundary.

If it is a Dirichlet boundary condition the conditions over there will be automatically defined, I do not have to worry about them anymore. So, what about this point? We will make use of what is known as linear indexing and linear indexing is useful for us to represent a pair i j as a simple linear index.

So, this point becomes 0, this point is 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15. So, these are the linear indices of all the variables. So, in the end, we will have A $\times$ X = b, where X is the value of the. So, X is essentially u alright X; I mean I am calling it X, essentially it is u. So, then what will be u? u will be this thing u_0, u_1, u_2, u all the way to u_{14}, u_{15} alright.

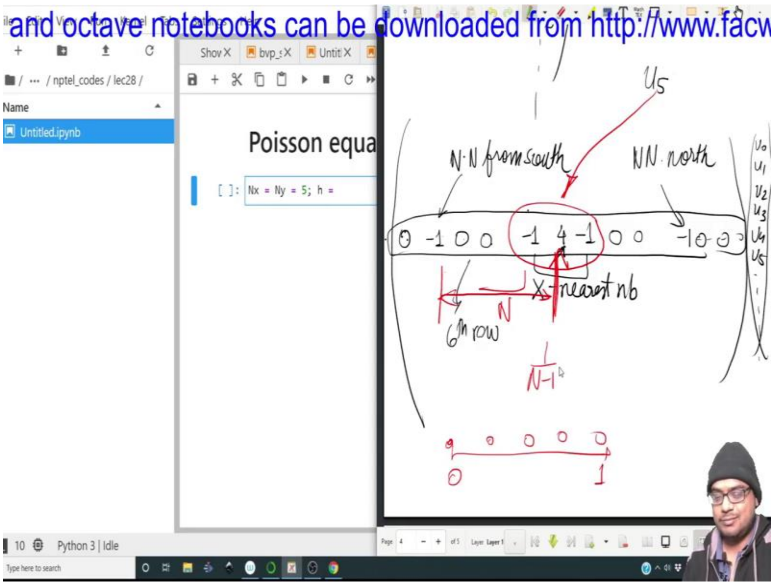
So, there are 16 elements over here. So, u is 16 $\times$ 1; A 16 $\times$ 16 as I have explained it is going to be $N^2 \times N^2$ and b is going to be 16 $\times$ 1. Now, let us focus on 5, u_5. So, when I am writing on the equation for u_5, alright. So, there will be u_3, u_4, u_5 and so on. So, what about the equation for u_5? u_0, u_1, u_2, u_3, u_4, u_5, u_6 and so on.

(Refer Slide Time: 15:01)



So, u $_5$ will have contributions from u $_4$, u $_6$, u $_9$ and u $_1$; because these are the nearest neighbors of 5, alright. So, in terms of the linear index; what are the contributions? It is the linear index - 1 + 1 + N, where N is the size number of grid points in one direction. So, this is what 5 + 4, this is 5 - 4. So, it is the linear index + N and linear index - N, right. So, the matrix A will resemble something like this.

(Refer Slide Time: 16:00)



So, it will be what do we have? We have one; so 0, - 1, 0, 0. So, there will be a - 1, 4, - 1, 0, 0, - 1. So, this is what? This is going to multiply sorry and so on and it is going to multiply u_0, u_1, u_2, u_3, u_4, u_5 and so on. So, this is going to be the row, which will of which this particular element will multiply u_5; because we are focusing on u_5.

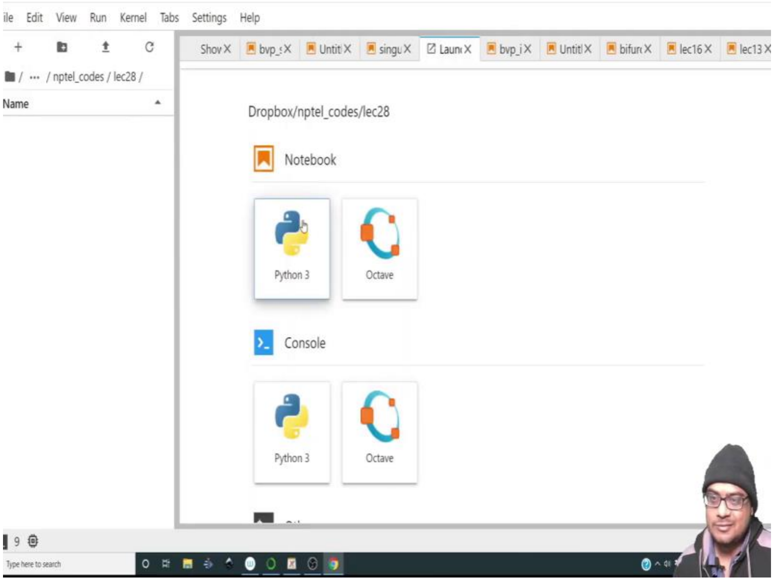
So, 0, 1, 2, 3, 4, 5 excellent; this is going to be, this is, if this multiplies u_5, this is going to multiply u_4, this is going to multiply u_6. So, these are the X nearest neighbors, this is the nearest neighbor from the south and this is the nearest neighbor from the north, alright. So, all of this is this is which number of row is this is going to be the 6th row, all this everything else is 0. So, all this is the 6th row; it is going to multiply such that this 4 multiplies with u_5, think about it, right.

Well, like I have said, many times this is not a course where we are going to discuss all this; but hey in case you know all this, I am just giving a quick refresher, in case you are not aware, I will post some links you can have a look. But it should be clear that there are N square number of elements; apart from the nearest neighbors in the X direction, you also have the nearest neighbor contributions from the north and the south.

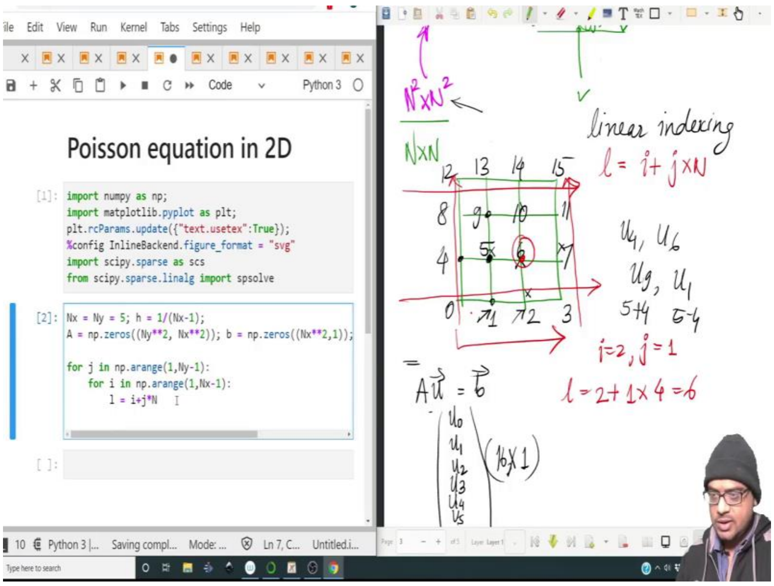
And because of the shift in the linear index by N, so there will be from this position; from the position of the prime importance that is i j. So, this is the linear index corresponding to. So, this corresponds to the linear index 5, alright.

So, the nearest neighbor from the north and south will have a shift of N, where N is the number of grid points, alright. So, with this in mind, let us start programming and as we go in the program, we will discuss various aspects of it, alright.

(Refer Slide Time: 18:52)



(Refer Slide Time: 18:59)



So, let me go over here. So, let me create a new file. So, the purpose of this function is to solve a Poisson equation in 2D. And the Poisson equation under consideration has already been mentioned in the notebook, but. So, first things first we are going to define the number of grid points in the x and y direction. So, let us say Nx = Ny = 5, only 5 grid points and the corresponding h.

So, h is going to be what, alright? So, if you have five grid points from 0 to 1; then the grid size is going to be 1/(N – 1), alright. So, let us do that; h is going to be 1/(Nx – 1), alright so far so good. So, let us predefine or before all this, I have forgotten to add the different packages that we will need in fact; we do not need ipy widgets, in fact we going to import scipy. sparse as scs. And from scipy. Sparse.linagl; we going to import sp solve, right. So, far we have defined what Nx, Ny are and what h is and now based on this; let us predefine what A and B are going to be.

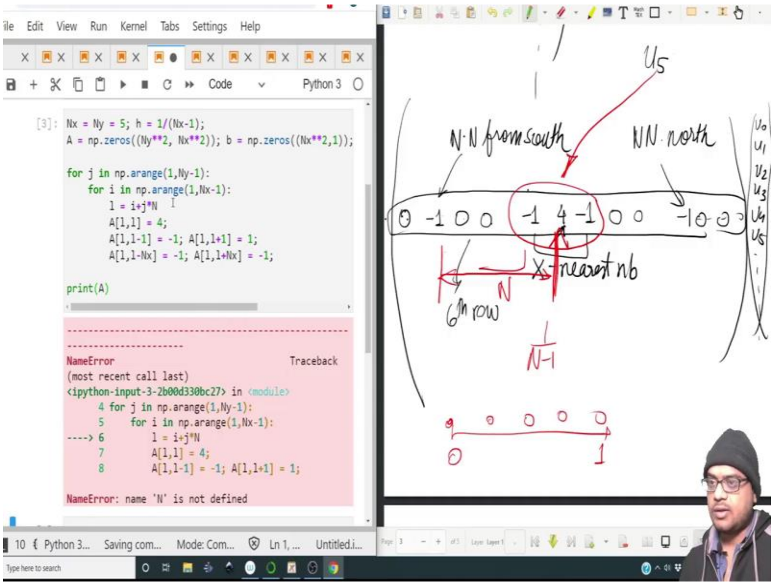
So, A = np.zeros((Ny**2, Nx**2)); well actually it has to be Nx, $Ny^2 \times Nx^2$, but Nx, Ny is they are equal, but still for completeness I am going to make this $Ny^2 \times$ Nx, alright. b is going to be np.zeros and this is going to be Nx^2, 1 or it is going to be actually Nx $\times$Ny , 1; but anyway there equal, so it does not make a difference.

And similarly x is also going to be r, let me call it u. So, when np dot zeros Nx^2, 1, alright so far so good, nothing wrong. So, what we have done so far is, we have simply defined all the different matrices, arrays that we are going to use, nothing fancy. So, let us do a loop for j in np., ok. So, let us only focus on arriving at the inner points. So, what we are going to do is, we are going to disregard the i = 0.

So, i = 0 corresponds to this, i = N - 1 corresponds to this, j = 0 corresponds to this, j = N - 1 corresponds to this. So, what we can do initially, so we can disregard all those boundary points. So, we can simply do a loop over the inner point. So, i for j = np. arange 1 to Nx - 1 for i in np. arange 1 to. So, this has to be Ny - 1 and this has to be Nx - 1.

So, now what are we going to do? So, first we will define l as the linear index. So, what is l? So, if you look at this thing. So, the linear index l will be = i + j $\times$N right, where N is the number of points in this direction; they are equal, so it does not make a difference. So, let us see. So, for this point i is = 2, j = 1. So, the linear index is 2 + 1 times 4, which is 6 and this, the linear index is actually 6 so good.

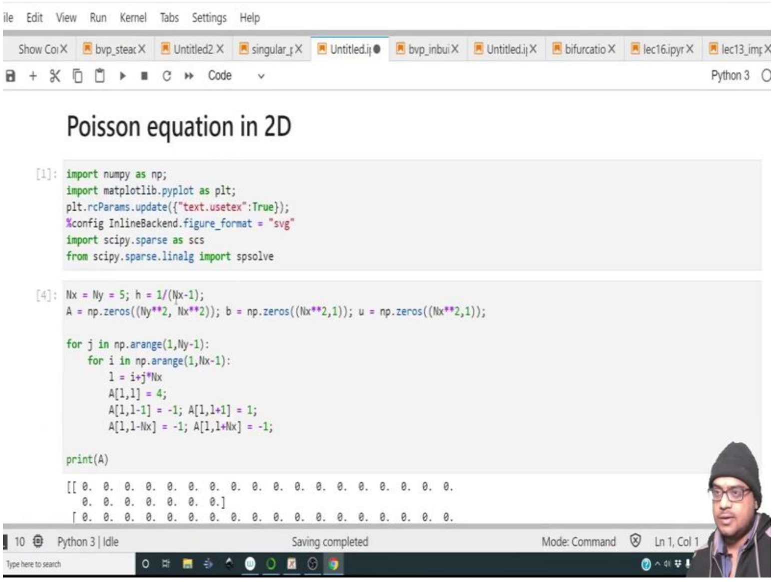
(Refer Slide Time: 23:48)



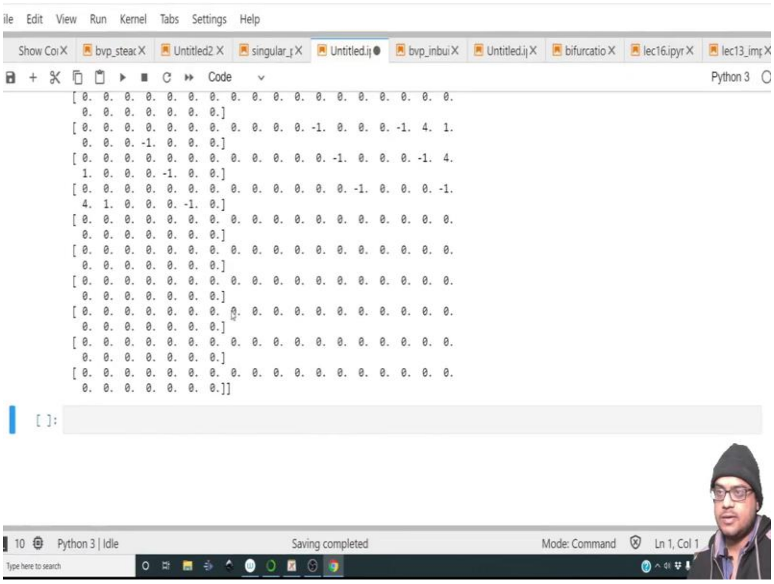
So, the linear index is i + j -. So, this is what? Linear index l is i + j × N, great. So, once we have the linear index, we have to then construct the matrix this, particular matrix. So, the diagonal element is going to be 4. So, A l , l that is the diagonal element, it is going to be 4; A l , l - 1, it is going to be the neighbor on the west, it is going to be - 1; A l , l + 1 is going to be 1, that is the neighbor from the east.

A l , l - Nx, it is going to be - 1 that is the neighbor from the south and this is going to be l , l + Nx and that is going to be the neighbor from the north, alright. So, so far we have just constructed the matrix A, let us print it out, alright.

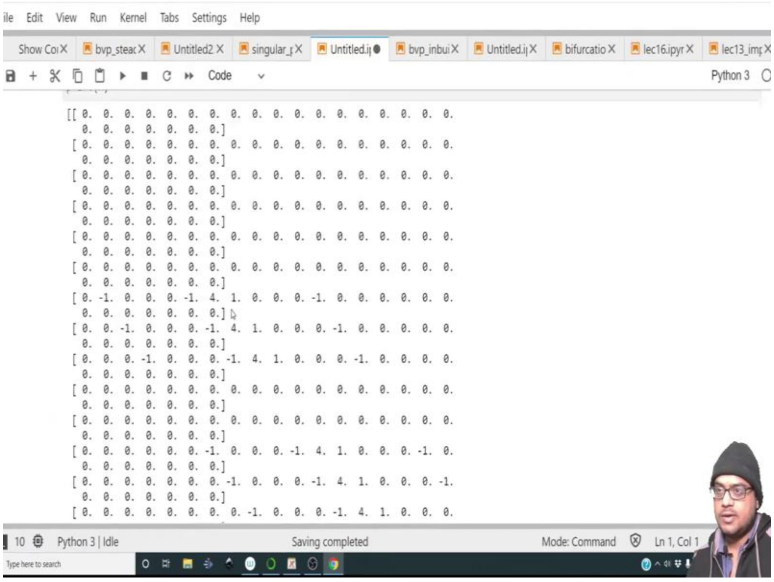
(Refer Slide Time: 25:02)



(Refer Slide Time: 25:04)

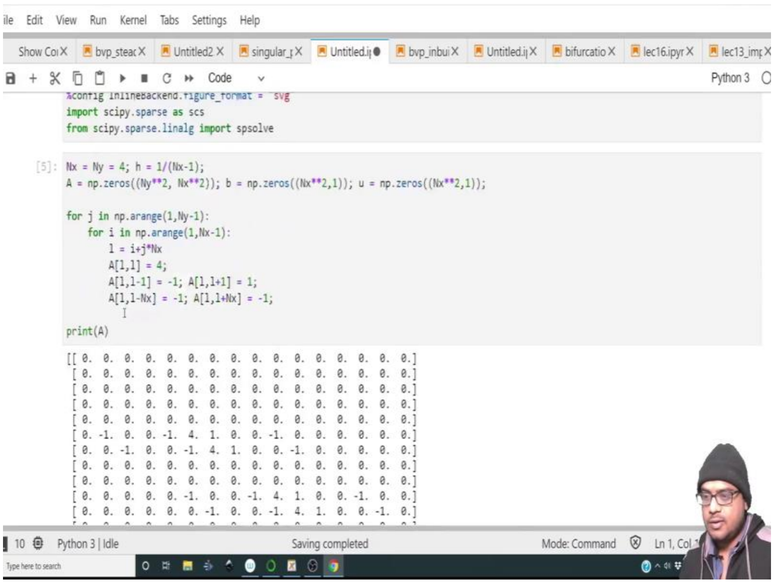


(Refer Slide Time: 25:06).

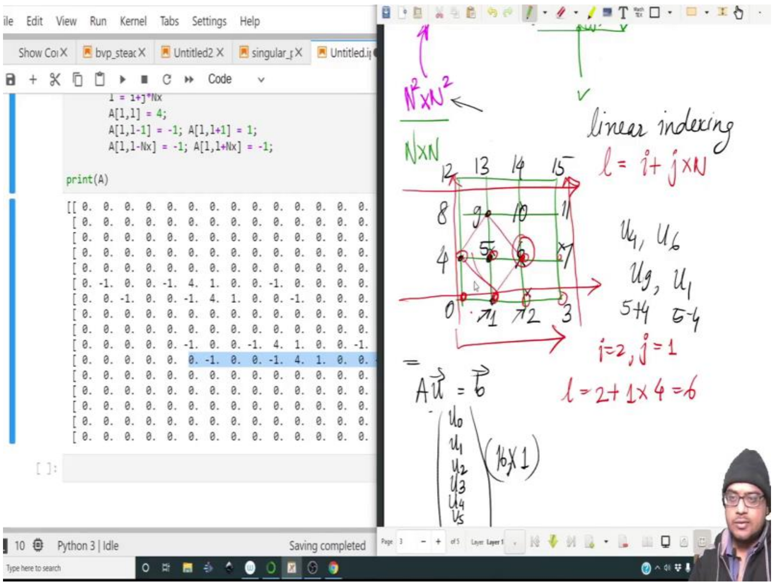


So, N is not defined ok, this has to be Nx, alright ok.

(Refer Slide Time: 25:11)



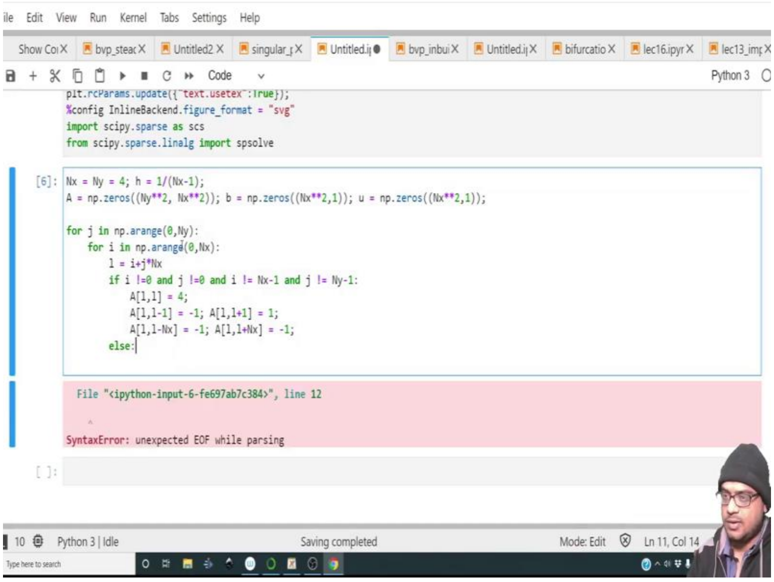
(Refer Slide Time: 25:11)



So, we have a lot of things; let me make it 4, right. So, look at this. So, we do have the penta diagonal matrix system. So, this, this, this and this. So, let us see, this is done for 4. So, what we have over here is 4. So, this is what 1, 2, 3, 4, 5, 6, 7, 8. So, for the element with the linear index 5, we do have four internal points. So, let us see 0, 1, 2, 3, 4, 5 alright; so 0, 1, 2, 3, 4, 5, ok.

So, the diagonal element checks out, then for 6 also we do have the nearest neighbors. So, this is for 6. Then for 9, we do have a nearest neighbor. So, if this is 6; 7, 8, 9. So, nine also checks out; you can verify this column number is also going to be 9. So, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, alright. So, everything checks out. So, we know that the code is correct as far as internal node generation is concerned, right.

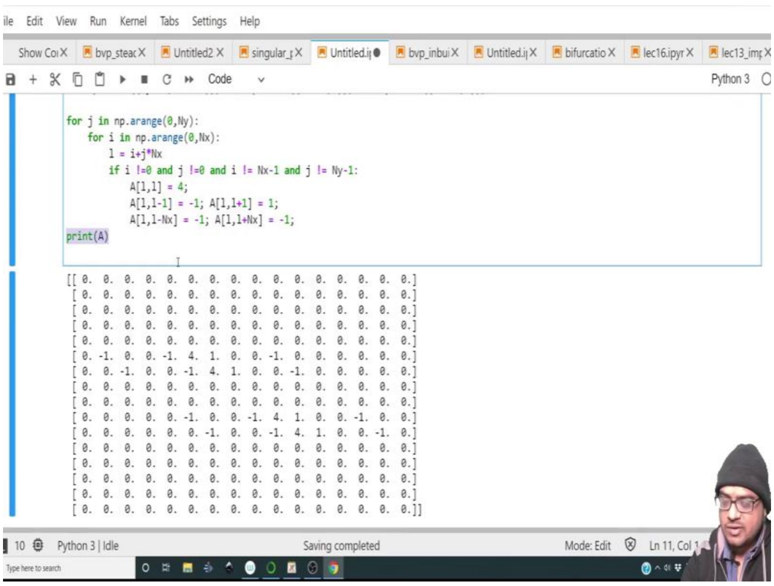
(Refer Slide Time: 26:38)



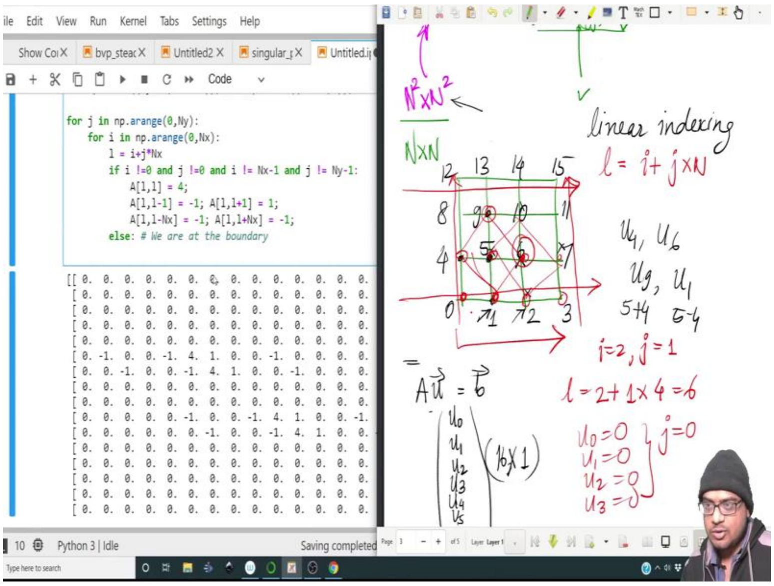
So, what we can do is now extend our code, so that we are not restricted to only the inner nodes; we want to account for all the nodes, but what we are going to do is make a check. So, if i $\neq$ 0 and j $\neq$ 0 and i $\neq$ Nx - 1 and j $\neq$ Ny - 1.

So, what this means is; if you are not at the outer boundaries, then you do this particular assignment, right. So, then you do this particular assignment, else you do something else, right. So, this should would do nothing, I mean else you do nothing ok; I need to, let me just removed this.

(Refer Slide Time: 27:42)

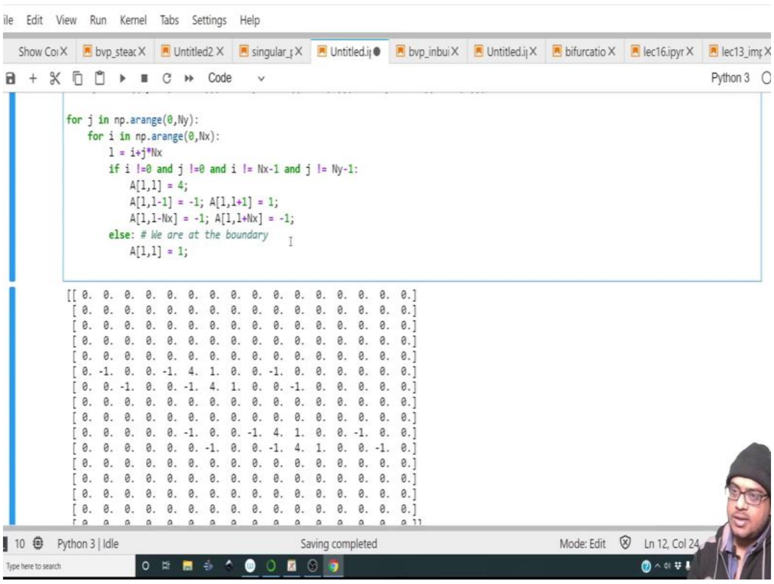


(Refer Slide Time: 27:50)



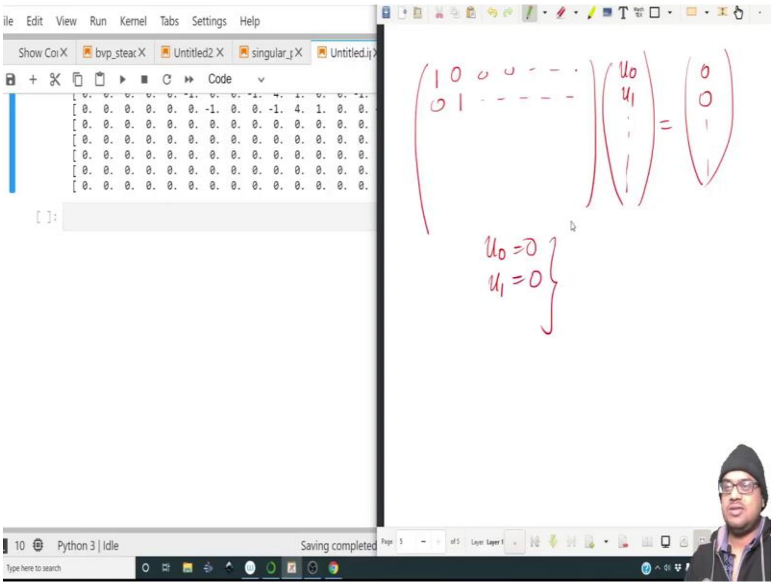
So, it still gets the same output. So, now, what we need to do is inside this, we need to put in the boundary conditions. So, if this is not satisfied else. So, this else statement means, we are at the boundary and at the boundary u = 0; meaning $u_0 = 0$, $u_1 = 0$, $u_2 = 0$, $u_3 = 0$. So, all these linear indices correspond to j = 0, alright. So, what I mean to say is, the linear index corresponding to these boundaries have the Dirichlet condition that $u_0 = 0$, $u_1 = 0$.

(Refer Slide Time: 28:34)



So, simply this will imply A l , l is = 1 and the RHS will be = 0; meaning how will the matrix look?

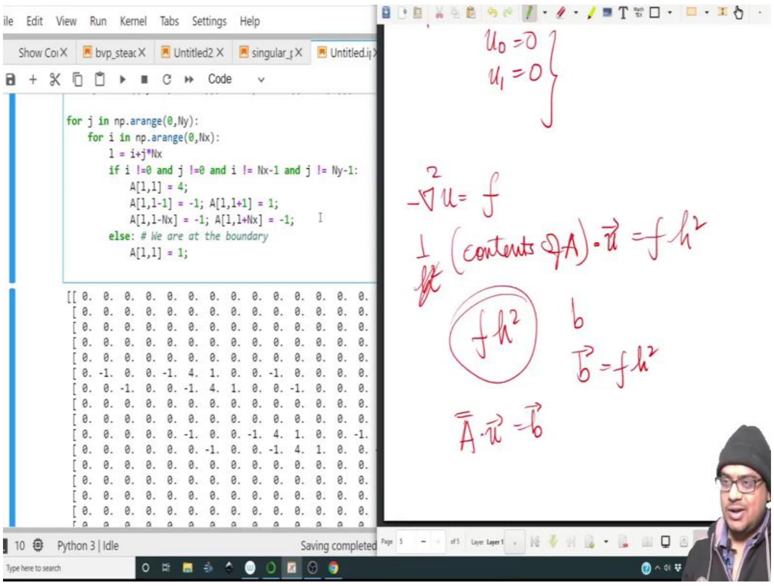
(Refer Slide Time: 28:50)



So, u_0 is 0. So, 1, 0, 0, 0, 0 all zeros; 0, 1, 0, 0, 0, 0 all zeros 0, 0. So, one times u_0 = 0. So, the first linear equation that we get from this is $u_0 = 0$; from this $0 \times u_0 + 1 \times u_1$, so you will have $u_1 = 0$. So, it is as simple as that, you need to just assign u linear index , linear index is equal to 1 that is it.

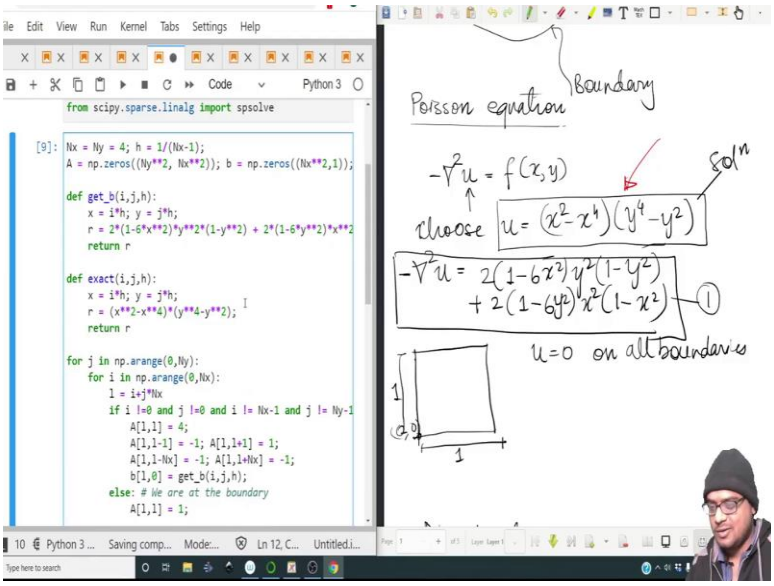
So, now we are at the boundary, alright. So, now, we have put in the boundary conditions as well or they have forgotten to create the right hand side matrix; by right hand side matrix, I mean this matrix b. So, what is b? So, we have just gone ahead with the discretization of, we have just written this inside A.

(Refer Slide Time: 29:52)



So, what we have is . So, then $-\frac{1}{h^2}\cdot$(contents of A) $\cdot$ u = whatever we have. So, - sorry this - sign we had absorbed into the Laplacian. So, h^2 goes over here. So, fh^2 is what we have to put in b. So, A $\cdot$ u = b, where b is going to be fh^2, alright. So, h we have already defined; what is f, what is f ? f is this.

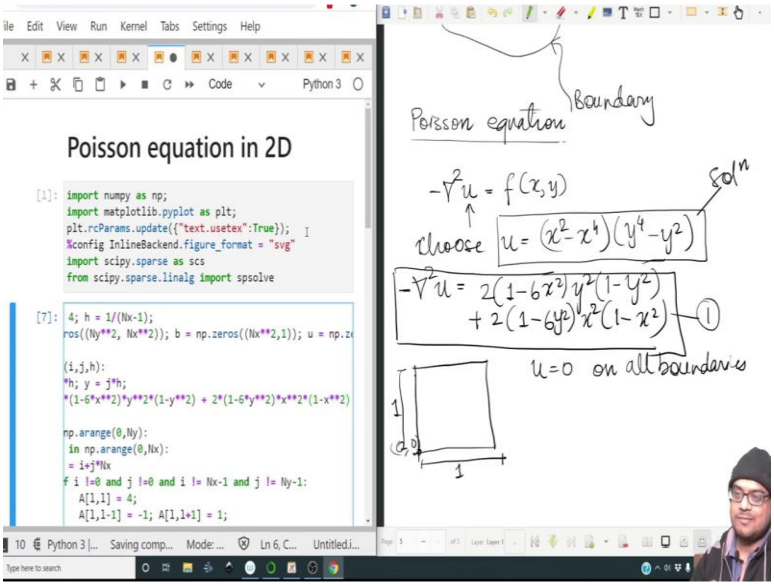
(Refer Slide Time: 30:49)



So, what we will do is create a function. So, b l , 1 0; lth row, 0th column, because it is a column vector is going to be get b and we are going to pass i , j , h. The reason why I am passing only i j and h; because from i, j and h, we can find out what x and y are.

And given what x and y are, we can find out that complicated looking function, it is quite easy. So, let me go over here and let me define get b and the inputs will be i , j , h. So, x will be i $\cdot$ h, y will be j $\cdot$ h and then $r = 2(1-6x^2)y^2(1-y^2)+2(1-6y^2)x^2(1-x^2)$, right.

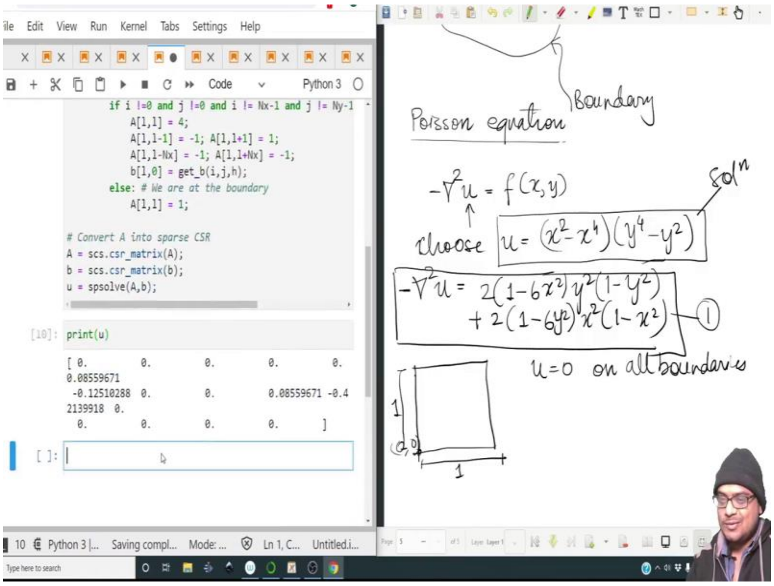
(Refer Slide Time: 32:01)



So, we must return r, that is it, that is pretty much it. So, we have to assign b this value of the RHS, so far so good. Over at the Dirichlet boundary conditions, we do not have to do anything; because as such b has been initialized to 0. So, unless it is something non zero, we do not really have to worry about it, alright. So, let me run this and see if something is broken; no, everything runs just fine.

So, what do we have? So, we have constructed the matrix A and matrix b. So, let me convert A to a sparse matrix and the reason we want to convert A to a sparse matrix is because, we want to use this sparse solver. A sparse solver is usually much faster than a dense solver; it requires much less energy right and not energy, it requires much less RAM, you do not have to store all the elements, alright.

(Refer Slide Time: 33:13)

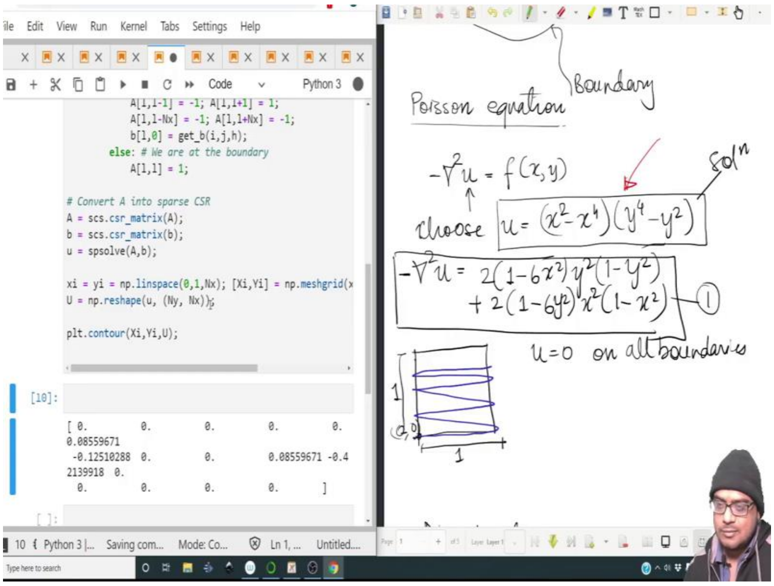


So, convert A into sparse, so CSR format, so compressed storage row format. And so, A will be scs. csr matrix of A; b will be scs. csr matrix of b and finally, u will be sp solve A , b, boom everything runs great. So, now, so now what, let us print u and see what we have. So, you have some values, we do not know whether it is correct or not; but how do we know whether it is correct?

If you know whether it is correct or not, because we had assumed the solution in the first place, right. So, we do have a way of certainly whether whatever we have written it is correct or not.

So, let us define this, let me define exact at i , j , h. So, it will have this, the r $= (x^2 - x^4)(y^4 - y^2)$, alright ok. You must return r and the reason why I am returning it is; I mean what I can do is, I mean this is multiple ways of doing it anyway.

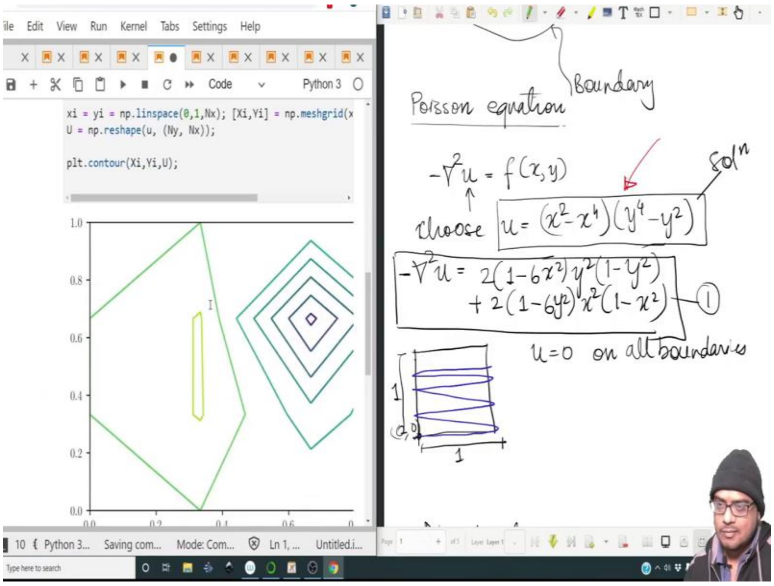
(Refer Slide Time: 35:17)



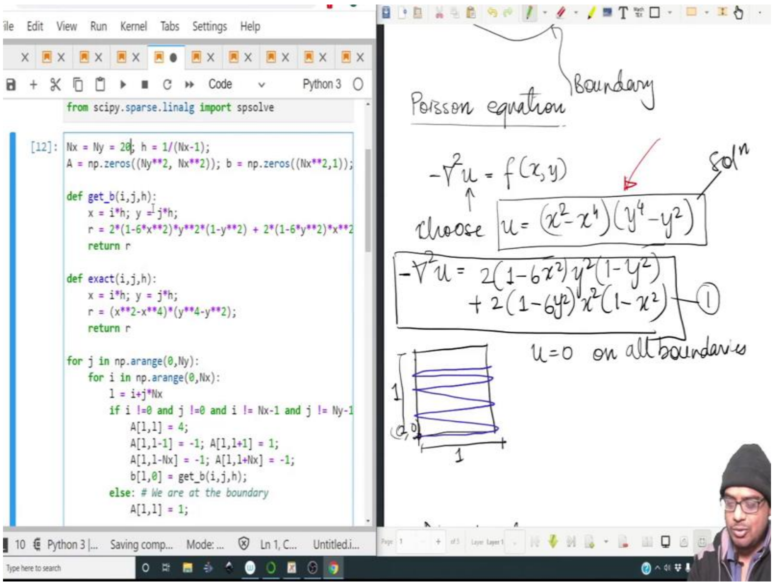
So, let me first reshape ok, let me define xi and yi as linspace 0 to 1 having Nx i's, right. So, now, X i , Y i will be np. mesh grid xi , yi. Now, what I will do is, I will reshape the solution u. So, remember A is now a linear index. So, if I were to traverse that particular linear index, I would traverse something like this, ok.

So, I am traversing it like this, but I want to reshape it to a square matrix. So, what I will do is, U = np. reshape u , Nx , Ny or rather it should be Ny , Nx; I mean it does not matter, because they are the same. So, we have reshaped, now let us do a plot. So, plt. dot plot, not plot; contour Xi , Yi , U; let us see how it looks.

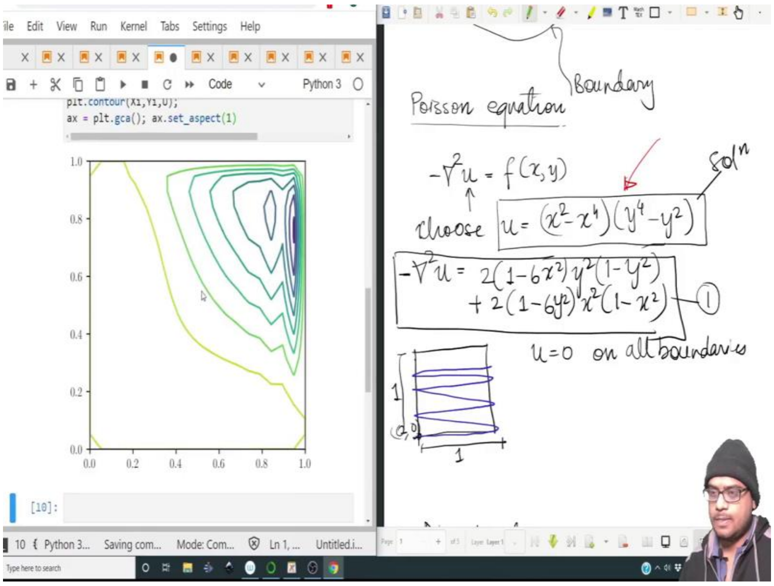
(Refer Slide Time: 36:43)



(Refer Slide Time: 36:51)

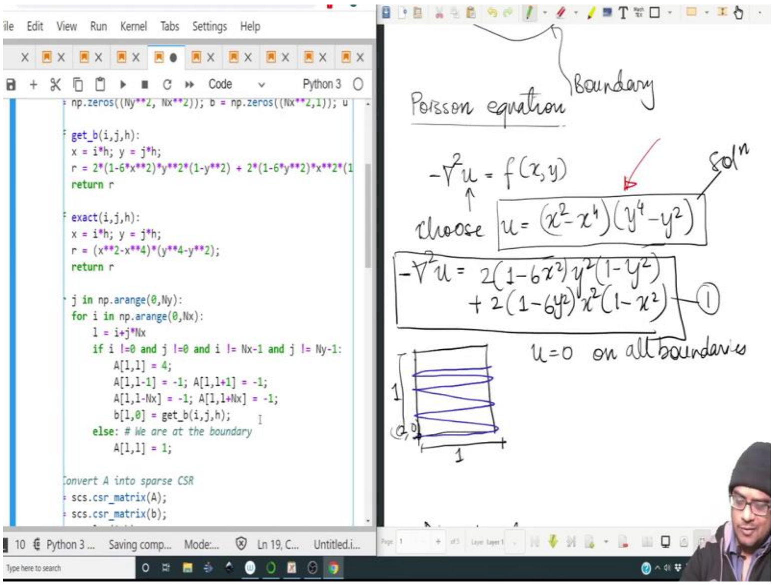


(Refer Slide Time: 36:54)

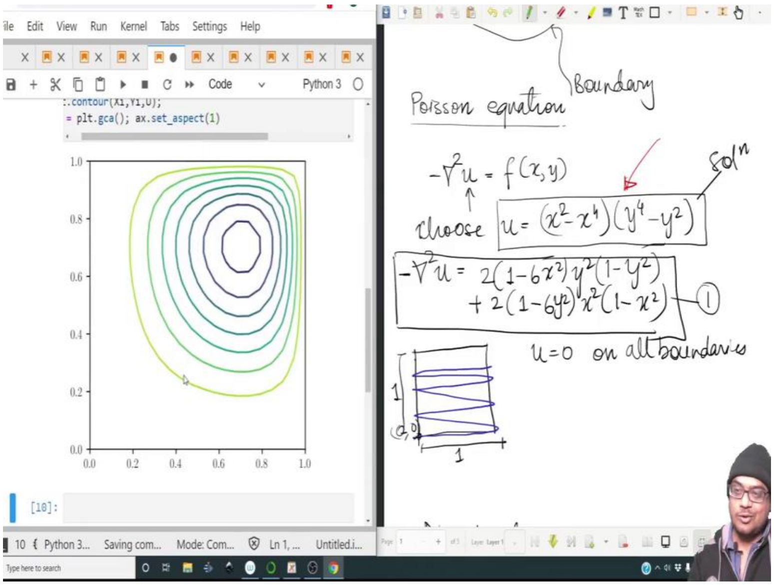


Looks weird, maybe because we have very less points; let me take 20 points. So, we must reshape, we had made one fatal mistake, ok.

(Refer Slide Time: 37:41)

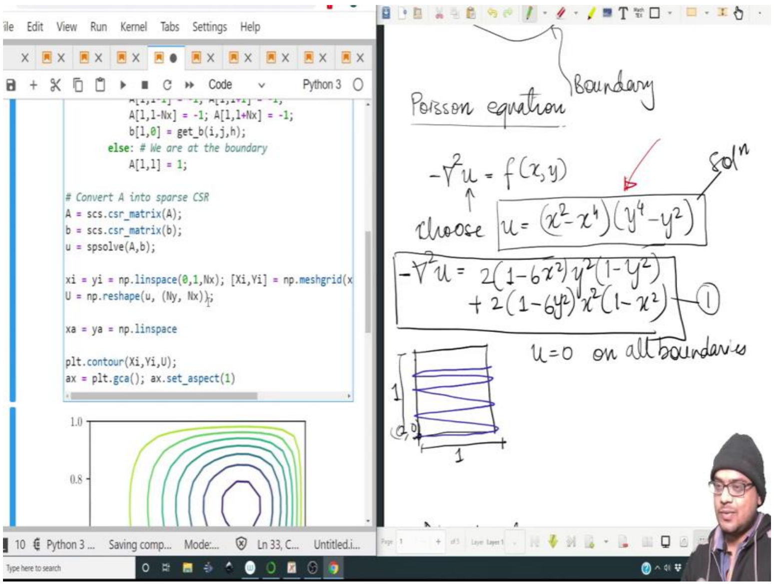


(Refer Slide Time: 37:42)



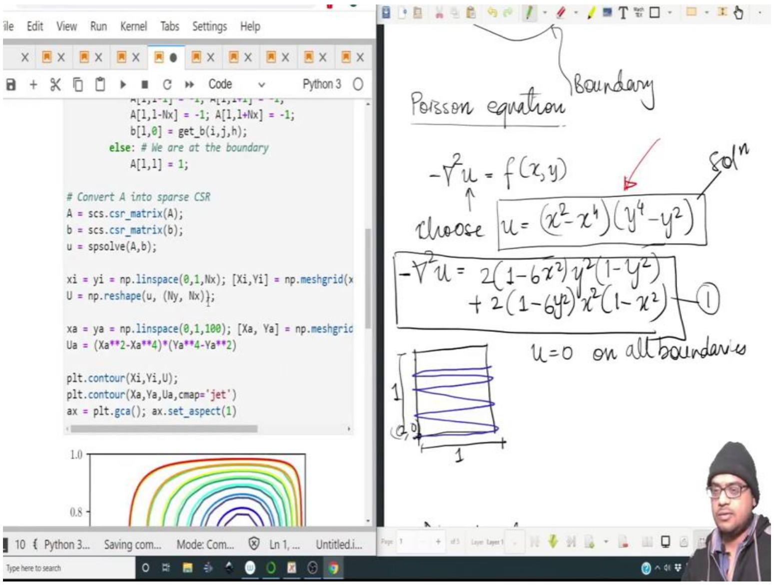
This looks much more like it. I was wondering the solution does not look right ok; anyway there was a small mistake over here, I had written + 1 instead of - 1, such things happen when you are doing it all out, but anyway. But how do we know this is correct? I mean you need to also plot the exact solution, right.

(Refer Slide Time: 38:07)

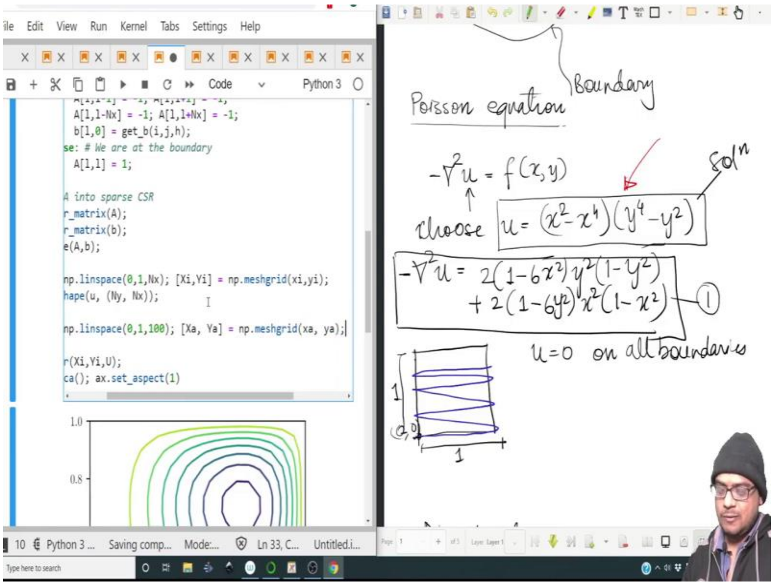


So, let us do that xa = ya = np.linspace.

(Refer Slide Time: 38:17)

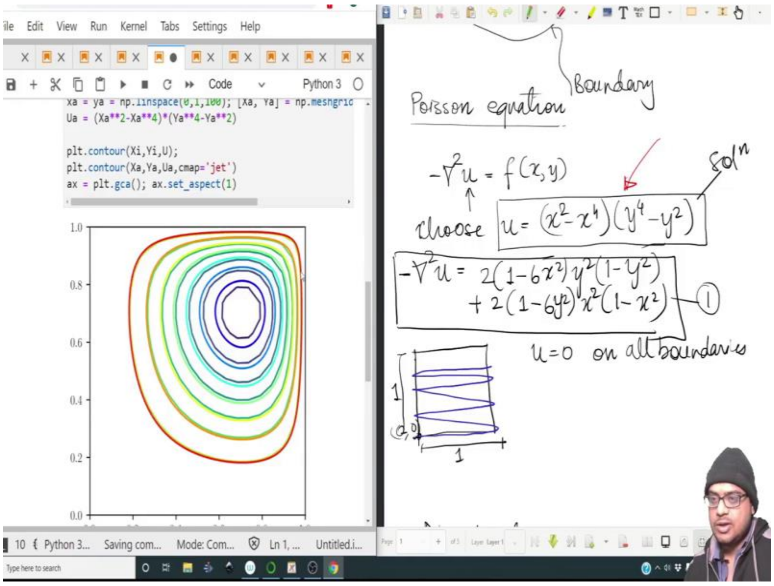


(Refer Slide Time: 38:30)



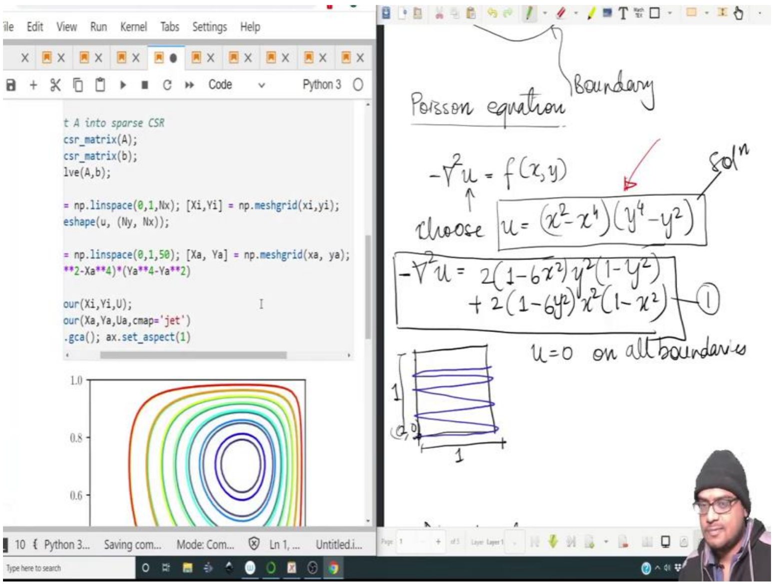
Or in fact, xi and yi, let me just define it xa = ya = np. linspace 0 to 1, 100 Xa, Ya = np .mesh grid xa, ya; then U analytical will be simply (Xa^2 - Xa^4)(Ya^4 – Ya^2). So, then we will also do plt. contour Xa, Ya, Ua, color map or c map = jet alright, let us.

(Refer Slide Time: 39:13)

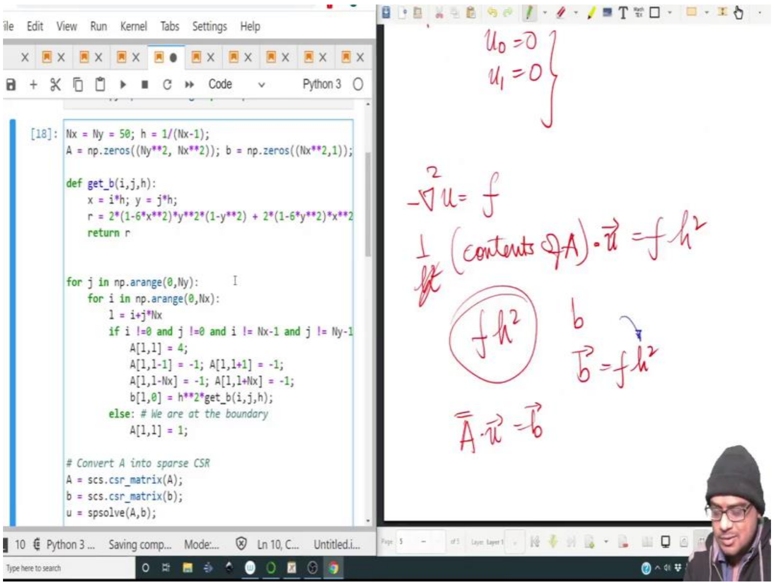


So, the there are two color maps; one is the jet and this and they do appear to be quite close. If I take a larger number of point's maybe 50, they seem close; but not exact, they should be exactly the same, maybe I have gotten some term wrong somewhere, let us see we do not need this. Well, maybe they are correct I just.

(Refer Slide Time: 41:07)

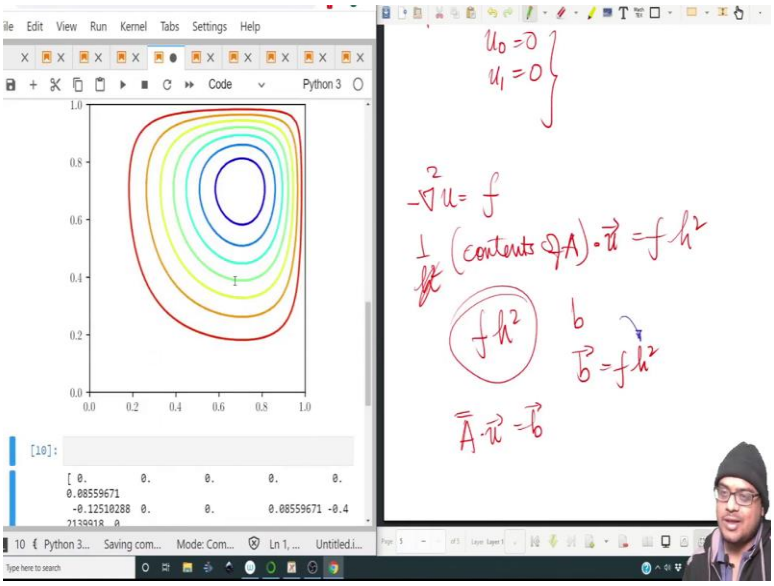


(Refer Slide Time: 41:38)



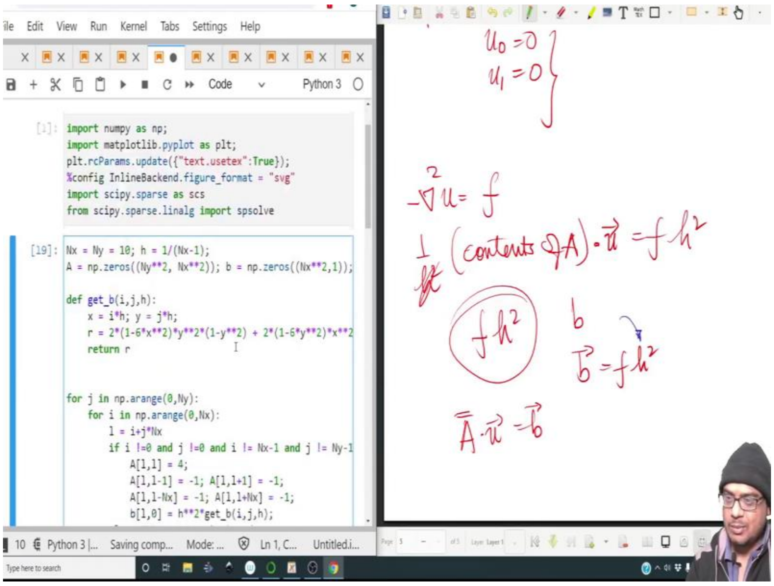
Let me just resample this to 50; I know, what is wrong I am, I forgot to multiply this by h^2. This is very small mistake; in fact spoke about that, but forgot to implement it. This particular h square that comes from the discretization of the Laplacian, I forgot to write it down.

(Refer Slide Time: 41:59)

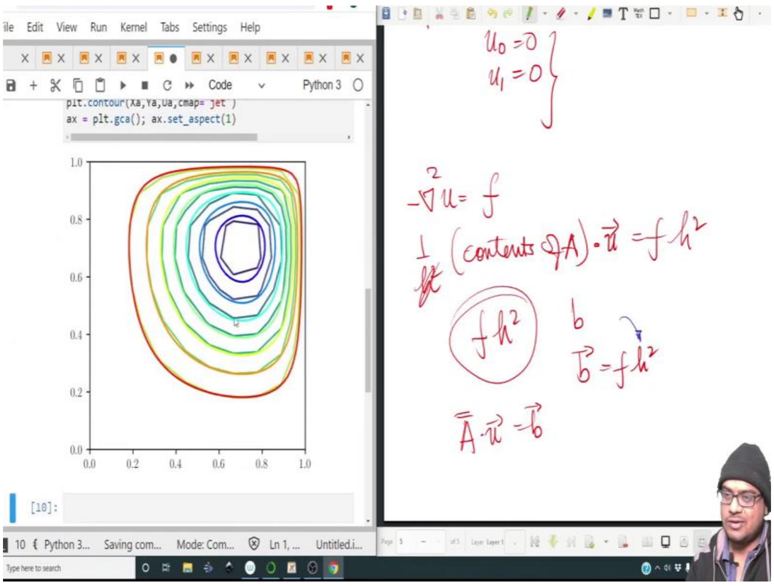


Well, now everything should match quite well and there you go; everything matches quite well and you can go ahead and take a loop over all the points and find out the maximum error, I am not going to do it over here.

(Refer Slide Time: 42:10)



(Refer Slide Time: 42:13)



Let me just show you what happens when you reduce the resolution; it is it looks qualitatively the same. Once you start taking higher number of points, maybe just 20; it resembles it quite well, with just 30 points, it converges quite nicely. So, what I request you to do is to find out how the maximum error changes as the number of grid points increases.

So, to conclude in this particular lecture, we have looked at how we can discretize a 2 dimensional system; we have seen that we end up with a penta diagonal matrix, we can do a sparse solution of that matrix, we can find out the solution. And in this particular synthetic problem, we have sort of compared the solution against the analytical solution and they do turn out to be quite well once you have a larger number of grid points.

So, with this strategy, you can pretty much implement any boundary condition you have; you can use the ghost node strategy to implement the Neumann boundary conditions as well, I have not discussed it over here. But it is a straightforward extension of what we had done for the 1 dimensional case a few lectures ago. So, with this I am going to end this particular lecture in this week. Next week we are going to start with pet c and I will see you then, until then, good bye.