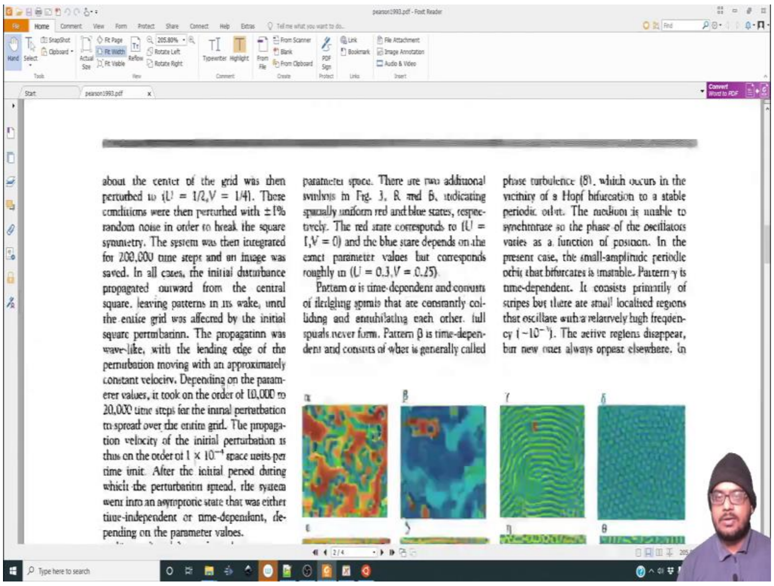
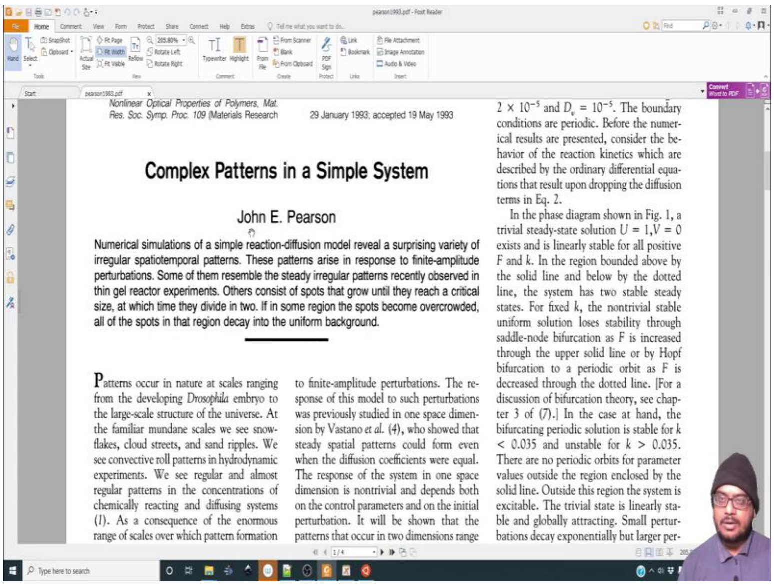
Tools in Scientific Computing
Prof. Aditya Bandopadhyay
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture - 38
PETSc – Turing patterns

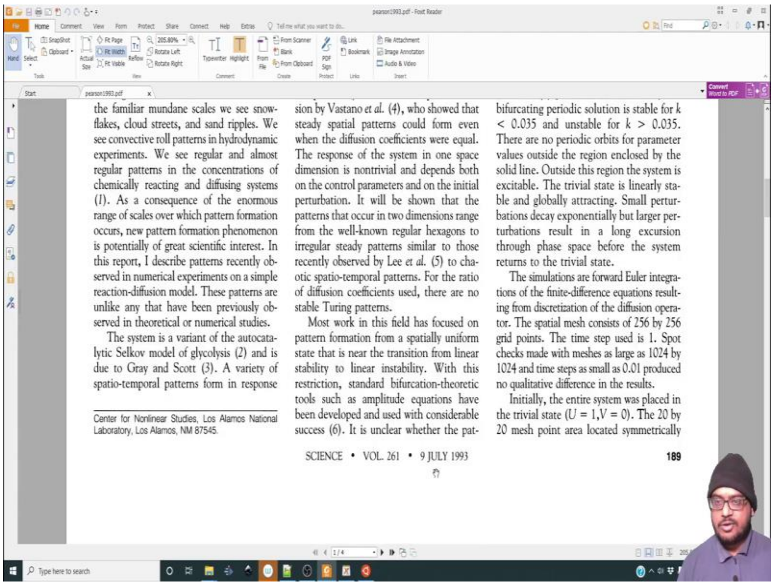
(Refer Slide Time: 00:26)



(Refer Slide Time: 00:37)

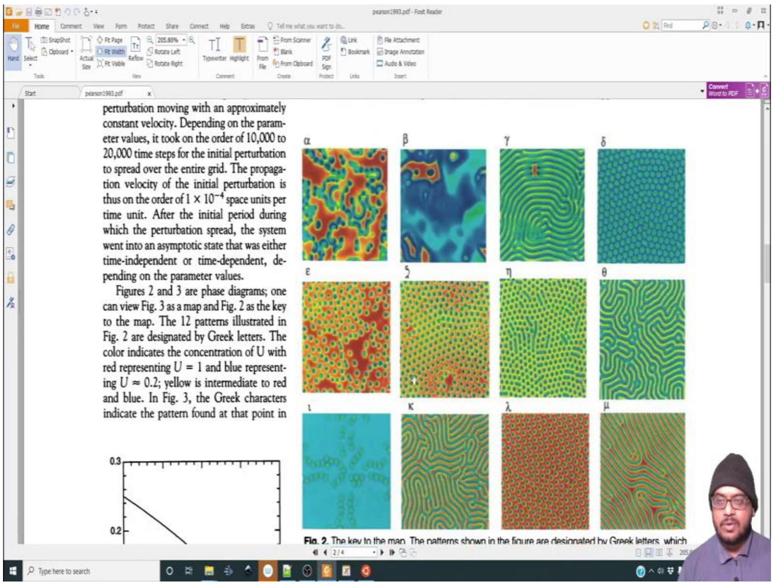


(Refer Slide Time: 00:45)



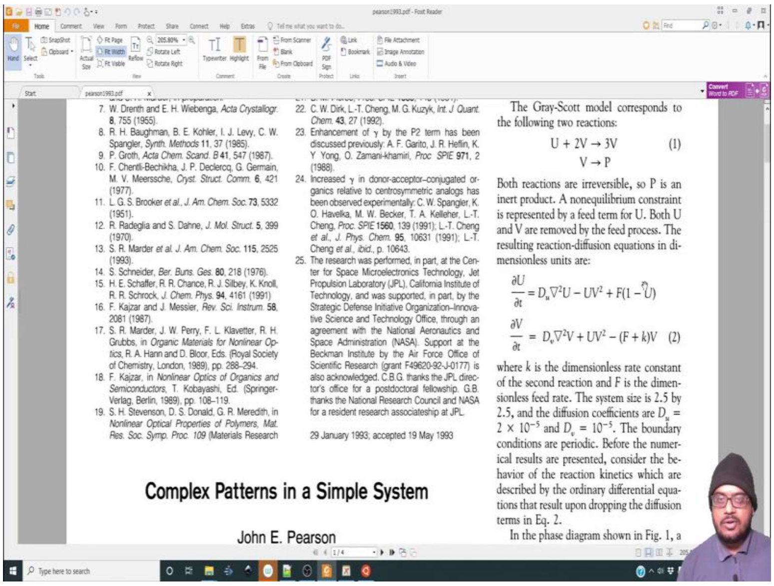
Hello everyone in this lecture we are going to have a look at some special patterns in very simple systems. So, in particular we will be looking at the article by John Pearson titled Complex Patterns in a Simple System and this was published in 1993 in the Journal Science. So, he was working at the center for non-linear studies and the question was whether complicated spatiotemporal behavior could be obtained through simple systems.

(Refer Slide Time: 01:06)



And patterns such as this you know stripes or islands, Labyrinth patterns, cells, spots even complicated Labyrinths.

(Refer Slide Time: 01:20)


7. W. Drenth and E. H. Wiebenga, *Acta Crystallogr.* **8**, 755 (1955).
8. R. H. Baughman, B. E. Kohler, I. J. Levy, C. W. Spangler, *Synth. Methods* **11**, 37 (1985).
9. P. Groth, *Acta Chem. Scand. B* **41**, 547 (1987).
10. F. Chentli-Bechikha, J. P. Declercq, G. Germain, M. V. Meerssche, *Cryst. Struct. Comm.* **6**, 421 (1977).
11. L. G. S. Brooker *et al.*, *J. Am. Chem. Soc.* **73**, 5332 (1951).
12. R. Radeglia and S. Dahne, *J. Mol. Struct.* **5**, 399 (1970).
13. S. R. Marder *et al. J. Am. Chem. Soc.* **115**, 2525 (1993).
14. S. Schneider, *Ber. Buns. Ges.* **80**, 218 (1976).
15. H. E. Schaffer, R. R. Chance, R. J. Silbey, K. Knoll, R. R. Schrock, *J. Chem. Phys.* **94**, 4161 (1991)
16. F. Kajzar and J. Messier, *Rev. Sci. Instrum.* **58**, 2081 (1987).
17. S. R. Marder, J. W. Perry, F. L. Klavetter, R. H. Grubbs, in *Organic Materials for Nonlinear Optics*, R. A. Hann and D. Bloor, Eds. (Royal Society of Chemistry, London, 1989), pp. 288–294.
18. F. Kajzar, in *Nonlinear Optics of Organics and Semiconductors*, T. Kobayashi, Ed. (Springer-Verlag, Berlin, 1989), pp. 108–119.
19. S. H. Stevenson, D. S. Donald, G. R. Meredith, in *Nonlinear Optical Properties of Polymers, Mat. Res. Soc. Symp. Proc. 109* (Materials Research
22. C. W. Dirk, L.-T. Cheng, M. G. Kuzyk, *Int. J Quant. Chem.* **43**, 27 (1992).
23. Enhancement of γ by the P2 term has been discussed previously: A. F. Garito, J. R. Heflin, K. Y. Yong, O. Zamani-khamiri, *Proc. SPIE* **971**, 2 (1988).
24. Increased γ in donor-acceptor–conjugated organics relative to centrosymmetric analogs has been observed experimentally: C. W. Spangler, K. O. Havelka, M. W. Becker, T. A. Kelleher, L.-T. Cheng, *Proc. SPIE* **1560**, 139 (1991); L.-T. Cheng *et al.*, *J. Phys. Chem.* **95**, 10631 (1991); L.-T. Cheng *et al.*, *ibid.*, p. 10643.
25. The research was performed, in part, at the Center for Space Microelectronics Technology, Jet Propulsion Laboratory (JPL), California Institute of Technology, and was supported, in part, by the Strategic Defense Initiative Organization–Innovative Science and Technology Office, through an agreement with the National Aeronautics and Space Administration (NASA). Support at the Beckman Institute by the Air Force Office of Scientific Research (grant F49620-92-J-0177) is also acknowledged. C.B.G. thanks the JPL director's office for a postdoctoral fellowship. G.B. thanks the National Research Council and NASA for a resident research associateship at JPL.

29 January 1993; accepted 19 May 1993

Complex Patterns in a Simple System

John E. Pearson

The Gray-Scott model corresponds to the following two reactions:

$$U + 2V \rightarrow 3V \quad (1)$$
$$V \rightarrow P$$

Both reactions are irreversible, so P is an inert product. A nonequilibrium constraint is represented by a feed term for U. Both U and V are removed by the feed process. The resulting reaction-diffusion equations in dimensionless units are:

$$\frac{\partial U}{\partial t} = D_u \nabla^2 U - UV^2 + F(1 - U)$$
$$\frac{\partial V}{\partial t} = D_v \nabla^2 V + UV^2 - (F + k)V \quad (2)$$

where k is the dimensionless rate constant of the second reaction and F is the dimensionless feed rate. The system size is 2.5 by 2.5, and the diffusion coefficients are $D_u = 2 \times 10^{-5}$ and $D_v = 10^{-5}$. The boundary conditions are periodic. Before the numerical results are presented, consider the behavior of the reaction kinetics which are described by the ordinary differential equations that result upon dropping the diffusion terms in Eq. 2.

In the phase diagram shown in Fig. 1, a

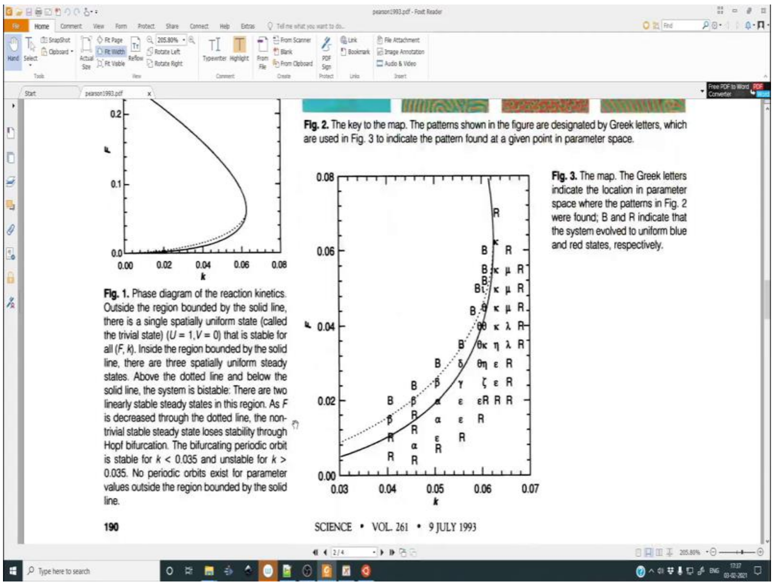
So, such kinds of patterns they were shown by Pearson that such a system is able to give rise to these complicated patterns. And in this lecture I am going to show an overview of the program and the reason I say overview is because this particular code would take a lot of time to write much more than an hour if I explain it everything.

So, I have already written down the C code and will be going through how you go about this. Well back in the day there is a; there is a big difference on in how things were done back in the day versus how one would do it nowadays. So, if you look at how he did it. So, the simulations are forward Euler integrations of the finite difference equations resulting from the discretization of the diffusion operator.

Spatial mesh is 256 by 256 and the time step is 1 and so, they refined the mesh they took a small time steps and this led to no qualitative difference in the results and the conclusion was whatever you see with this coarser relatively coarser mesh is ok.

And they had near the origin they gave a perturbation to the system ok with the random noise to break symmetry and once symmetry was broken after 200000 time steps they saw emergence of a pattern.

(Refer Slide Time: 03:04)

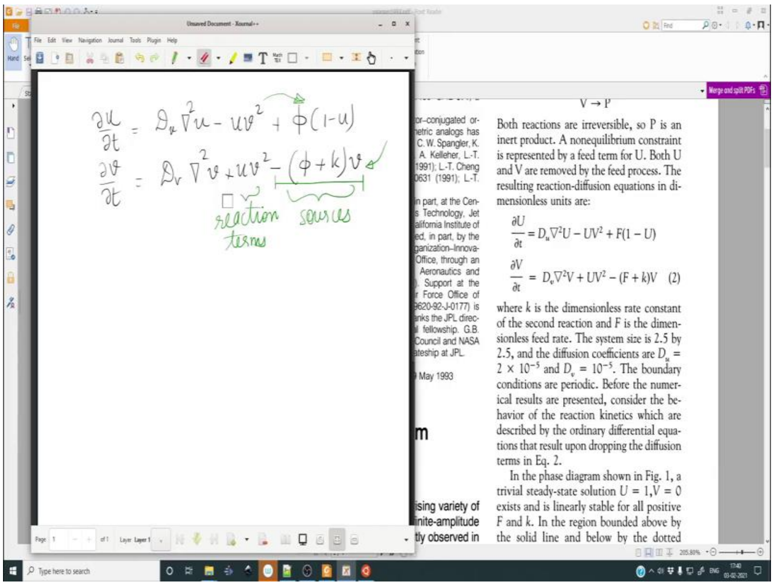


And the pattern is actually classified depending on the parameters at play. So, the parameters at play are F D u D v and k. So, in our system we will call this as phi and this has kappa ok. So, depending on that particular parametric space of F and in our case phi and kappa you will obtain one of these patterns ok.

So, B corresponds to uniform blue state where one product or one of the reactants dominates over the other. R corresponds to the red state where the other reactant dominates while epsilon, eta, kappa, lambda, mu, they are all different patterns as seen in this figure. And you can see that the band in which those kinds of patterns occur is not very large. If you go to a zone over here you will have completely blue.

If you go to a zone over here you would be completely red. So, as such you do not; I mean you have a very small parameter space to play with ok. So, you do get a host of interesting patterns.

(Refer Slide Time: 04:43)



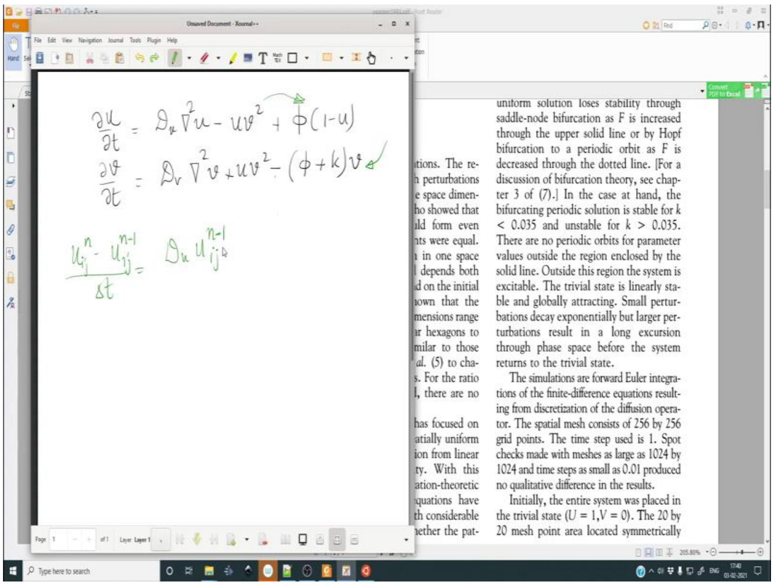
And so, let us see how we can go about solving this particular equation. So, the equation at hand is $\partial u / \partial t = D_u \nabla^2 u - uv^2 + \phi(1-u)$ and $\partial v / \partial t = D_v \nabla^2 v + uv^2 - (\phi + k)v$. So, these are the two systems.

So, you can clearly see that these two terms are the reaction terms and in this particular case u is being consumed due to the reaction and v is being produced because of the reaction and more v is produced per reaction because of the presence of v^2. So, what about these terms? So, these are the sources.

So, because of the consumption in u there is a source to make up for that consumed u and it is this sort of balance between consumption and production that is sort of driving this particular non-linear phenomenon. What about v?

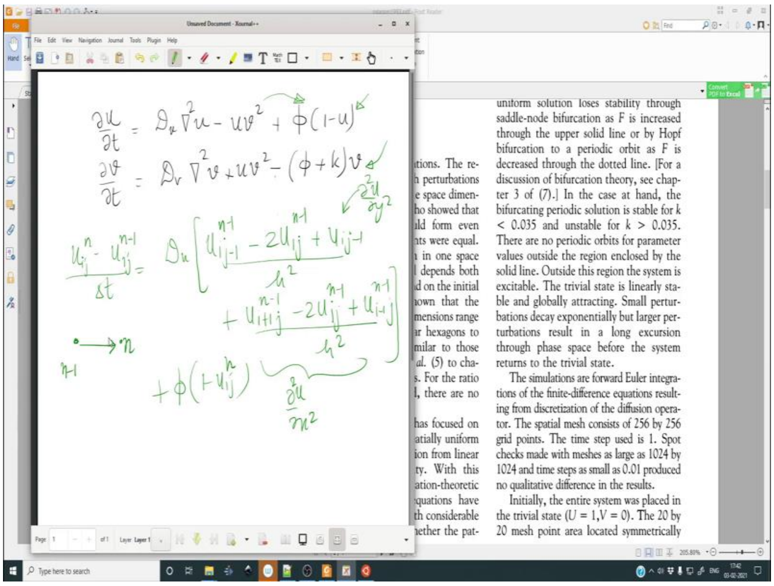
So, there is a steady decay you a bulk decay which is proportional to the local concentration and this gives rise to an evolution of the equation v. So, how can we solve such a system? Well, as has been told in the article itself you can simply perform a discretization.

(Refer Slide Time: 06:27)



So, you have $\frac{u_{ij}^{n+1} - u_{ij}^{n-1}}{\Delta t} = D_u u_{ij}^{n-1}$. So, this is an explicit method, but that is what exactly they did they did the finite the forward finite integration. So, its quite easy to do and so, I will. In fact, let me write down the entire discretization and you can take it as a challenge to program it using Python or Octave.

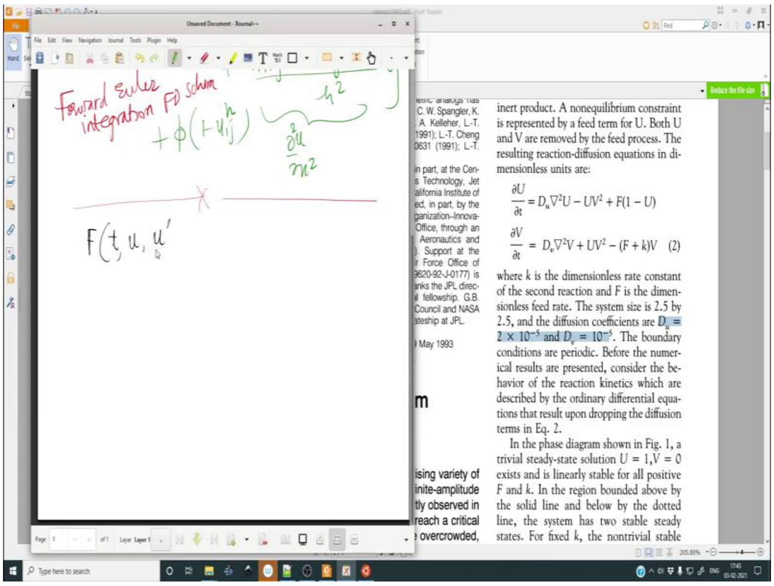
(Refer Slide Time: 07:01)



So, plus D u uij minus 1 plus minus 2 u ij plus minus 2 u ij j j. So, this these are all at n minus 1 times step plus u ij minus 1 upon h square plus u i plus 1 j and minus 1 minus 2 u ij n minus 1 plus u i minus 1 j at time n minus 1 upon h square. So, you can see that this is the del 2 u del y 2 term while this is the delta 2 u del x 2 term.

Well, now what about this? This is simply plus phi times 1 minus u ij at n minus 1. Because of the sort of linearity in this, this particular term you can actually take this to be at time n. And why is that? Because it does not break any explicitness of the problem, you can simply combine it with this particular term and obtain an expression for going from time n minus 1 to time n explicitly over each cell ok.

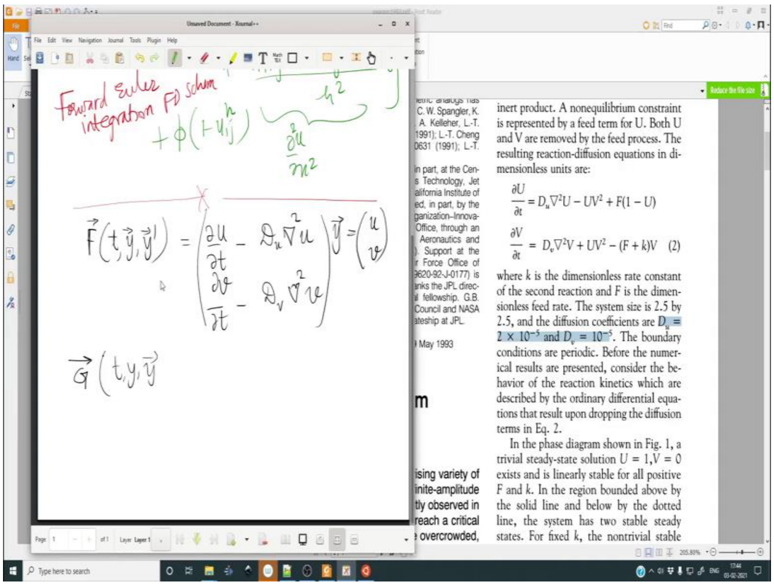
(Refer Slide Time: 08:42)



So, in similar fashion we can write the governing equation for v as well. Its everything is going to look the same and therefore, you have what is called as the forward Euler integration FD scheme as was done over here. And the delta t you can choose it to be 1 when the diffusion coefficients are appropriately chosen as mentioned in the paper over here ok.

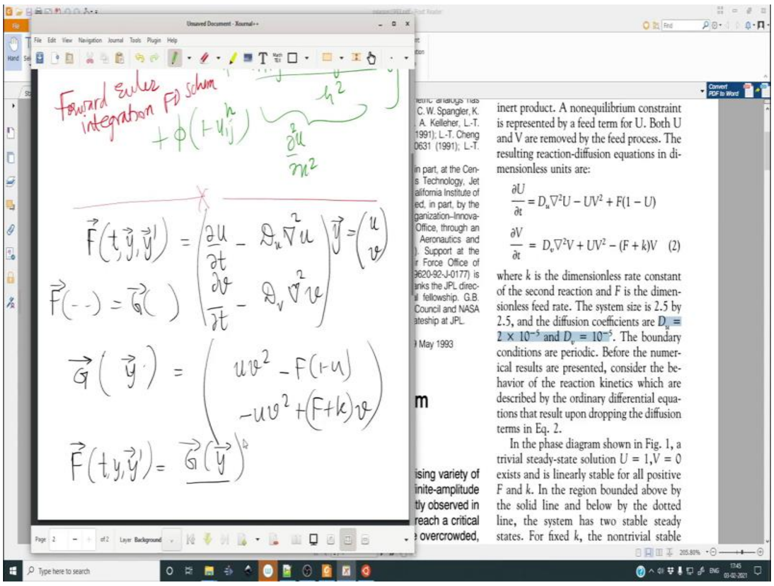
So, that can act as a nice exercise for you to do, but given that you have extensive solvers available nowadays. What we do is club the linear terms or non reaction and non source terms to one side and have the other terms on the other side. So, you can write let us say that F of t, u u prime or if I write it down in terms of the y vector.

(Refer Slide Time: 09:44)



So, y vector is actually comprised of u v, this is what y vector is comprised of. So, F let this be equal to del u del t minus D u Laplacian of u and dv dt minus D v Laplacian of v. So, these are the two linear terms whereas, let me define G as the source and the non-linear term; so, t y. So, its not a function of t. So, I can drop this particular t over here.

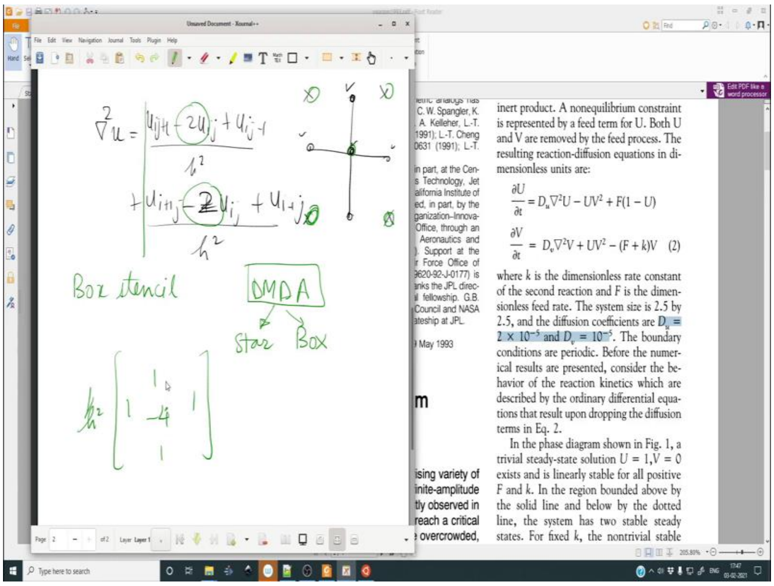
(Refer Slide Time: 10:28)



And simply write it as y comma y. It is in fact, not even function of y prime. So, I can simply write it as G of y. So, this would be equal to. So, we want to cast it in the form F of whatever it is equal to G of whatever it is. So, then these two terms have to go on the left hand side. So, it will be u v square minus F of 1 minus u and minus u v square minus F plus k or rather plus F plus k times v.

So, under these two when we write it like this we can write down the equation as F t, y, y prime is equal to G y ok. So, now, we have actually made a operator splitting into something which contains the time derivative and the spatial gradients and the other operator which contain simply the reaction term and the source term.

(Refer Slide Time: 11:50)



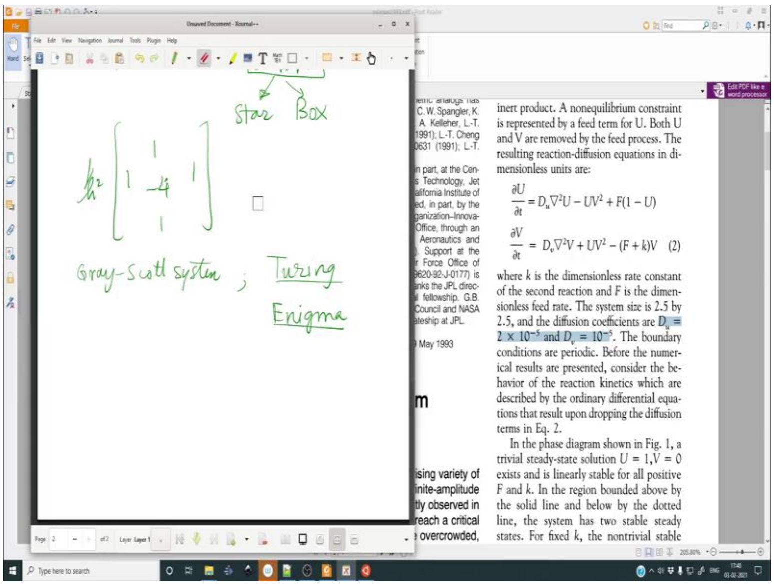
Well in the above situation for the Euler method I have written down the Laplacian I have written on the Laplacian in typical star format. So, by star format I mean if I am writing the Laplacian at this point I am writing in terms of this, this, this and that is it. So, its u ij plus 1 minus 2 u ij plus u ij minus 1 by h square plus u i plus 1 j plus rather minus 2 u ij plus u i minus 1 j upon h square.

So, when its like this you are writing it simply in terms of these terms while disregarding the contributions from this term and usually its fine nothing fancy happens, but in such cases where there is diffusion and all that you want to preserve the isotropy of the problem. There is another stencil which is appropriate and that is based on the box stencil.

So, if you recall that when we declared our DMDA there were two options one was to declare it as a star. So, DMDA for structured meshes there are two options. One is to declare as the star which is this case and one is to declare it as a box where you have access to these elements as well. So, when you account for this you essentially have 8 neighbours to a given point and you can write down the derivative in terms of them. In matrix form I can write down this as something like this.

So, it is 1 upon h square 1 1 1 1 minus 4 ok because these two terms they combine to minus 4. So, you are writing it at i comma j. So, there is 1 contribution from this, 1 contribution from this, 1 from this, 1 from this and minus 4 from the center point. What about the box?

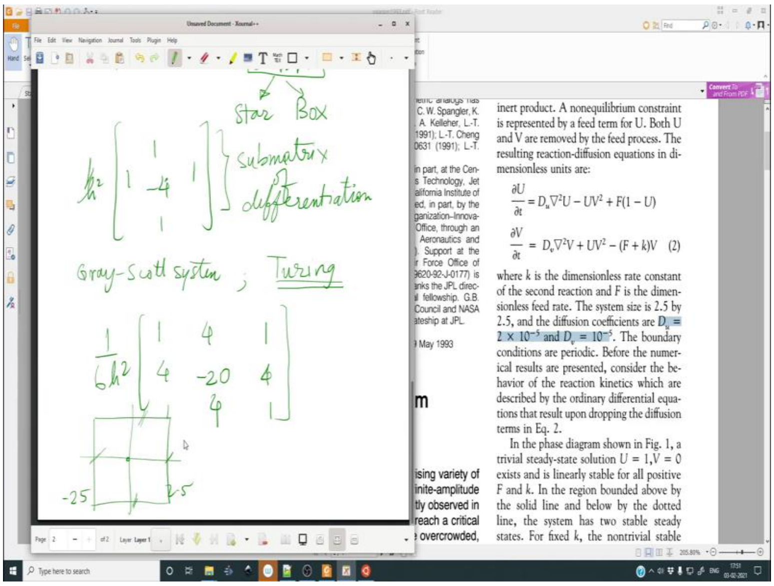
(Refer Slide Time: 13:43)



So, the discretization that many people have used in order to simulate what they call as the Gray Scott systems and actually this kind of things were analyzed in quite detailed details by Alan Turing. So, Alan Turing if you know he was the hero who was responsible for breaking of the Enigma cipher during the 2nd World War ok.

So, the Enigma cipher was a German encryption system with which they would send messages unimpeded across the I mean as radio waves and no one could even understand what their plans were, but Alan Turing was able to solve this is just a little bit of history just to break the monotonous routine.

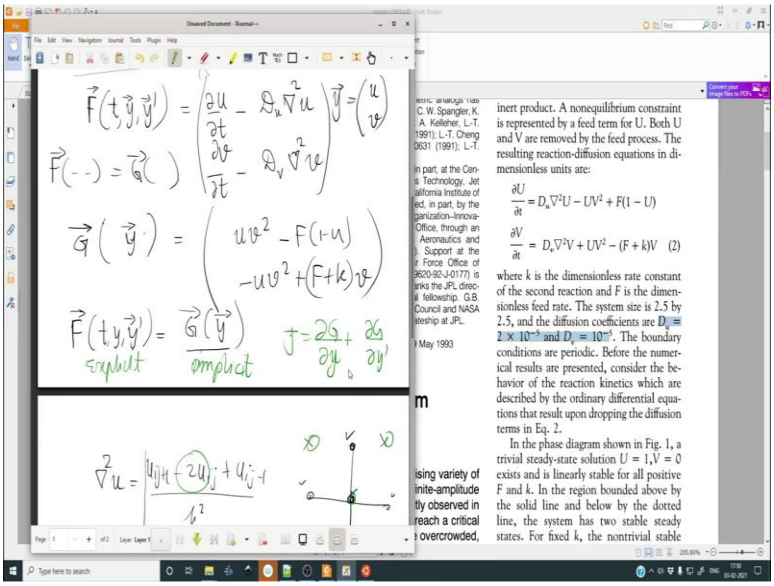
(Refer Slide Time: 14:41)



So, Alan Turing also started such kinds of systems and the general sub matrix. So, this is the differentiation sub matrix and the differentiation sub matrix for a better discretization would be this. So, this particular matrix gives us a better isotropic diffusion and helps us in removing some of the artifacts, but well you can try your hand at the previous discretion. And it does not make a difference its order x square anyway.

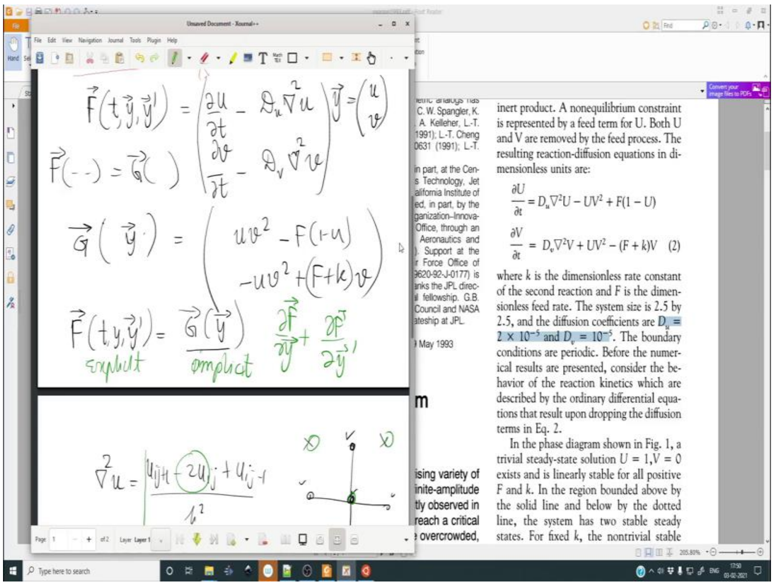
It is just accounting for more diagonal points which is always a good thing. The larger your basis set or rather the larger your sub matrix becomes the better you are at representing gradients ok.

(Refer Slide Time: 15:47)



So, what we have done is split the problem into two parts which is an explicit part and an implicit part. So, for the explicit part we must construct the RHS. For the implicit part we must construct both the implicit RHS and the sub matrix or rather the Jacobian in order to solve it using the SNES. And in this case the Jacobian will actually be a shift to Jacobian where it is not just del G del u or in this case y plus its del G del y prime.

(Refer Slide Time: 16:32)

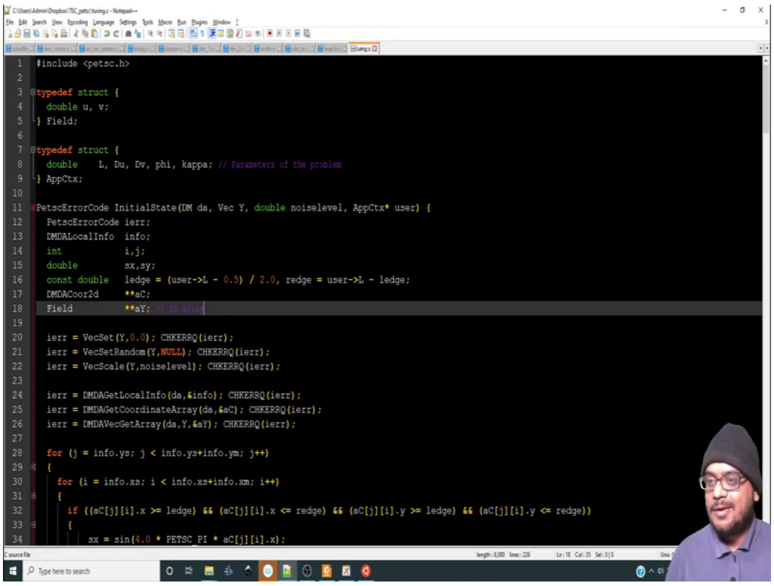


Sorry, we are going to create the Jacobian not for the non-linear part, but for the linear part. So, in this case it will be del F del y plus del F del y prime. So, you are going to construct this Jacobian, its called as a shifted Jacobian and once you start learning the theories of all this you will be; you will be more aware of what all these things are.

But my hope is once you do learn these things or if you have ever learned these things (Refer Time: 16:53) will enable you to implement them quite scalably on your home computer even ok. So, that is pretty much it and ok. So, the domain in this particular case is going to be periodic.

I have not drawn the domain the domain over which we are going to solve the problem spans from minus 2.5 to 2.5 if I am not mistaken in both sides and the origin is at the center and all the boundaries are periodic. So, its periodic in both x and y.

(Refer Slide Time: 17:33)



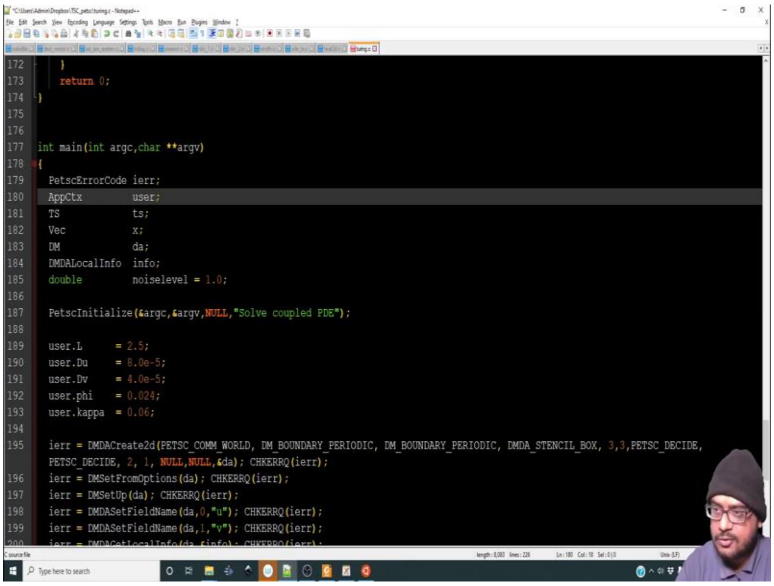
So, with this background let us go to the program I have written on the program already. So, we will do two things we will first create the Field structure which contains simply the u and v. We will create a structure which contains the parameters of the problem which is going to which are going to be the length of the domain, the u diffusivity, the v diffusivity, the phi and the kappa. So, these are the parameters of the problem.

And you will see the AppCtx that is the app context, it is being passed around to the functions so that you can construct everything in terms of the parameters that are being packaged under AppCtx. And the Field is required to declare the two degrees of freedom per node that we have ok.

So, recall that we have declared everything as one degree of freedom so far because we did not have a system of equations, but now we have at single node both values of u and values of v. So, we declare it as a struct and that struct is simply called is simply declared over here. And its star why? Because its a 2D matrix its a 2D array ok.

So, it helps in definition of the not the definition, but it helps to sort of reference to the data ok. So, let us see what the main looks like before going into the functions.

(Refer Slide Time: 19:03)



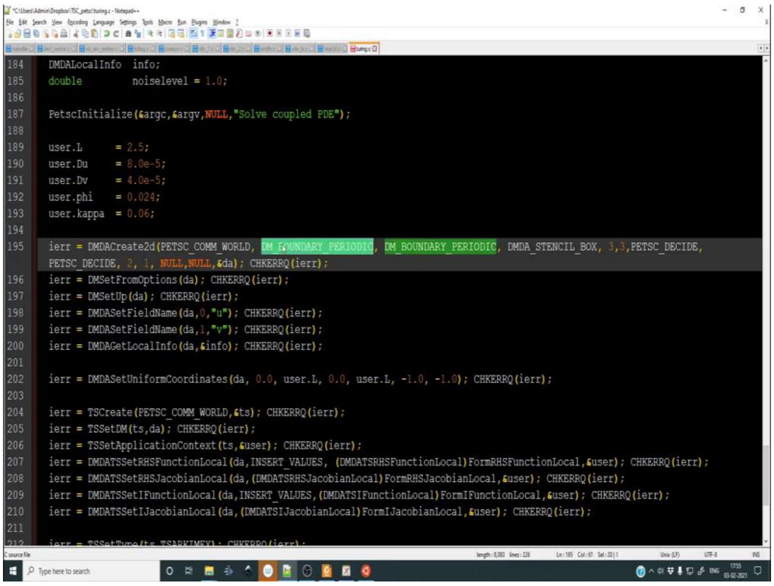
So, we have the declaration of the PetscErrorCode ierr. I mean you can make do without this as well. We have done a bunch of examples where we have not taken ierr as the output. But usually what you do is whenever you call PETSc function, you the return value is going to be an ierr that is a PetscErrorCode.

And then you check the error code with this function CHKERRQ, but you do not need to do that as you have seen in previous examples as well. Then you declare the AppCtx and assign it to a variable called as user then you declare the time stepping object, you declare the vector x, you declare the DM da, you declare the DMDALocalInfo and you declare a noiselevel.

So, this noiselevel you can change and recompile and run the program. Alternately, you can fetch the noiselevel from the command line arguments, but here we will do it through main as itself. Then you have PetscInitialize which you must have always then you declare the L, Du, Dv, phi and kappa. So, let us look into the paper.

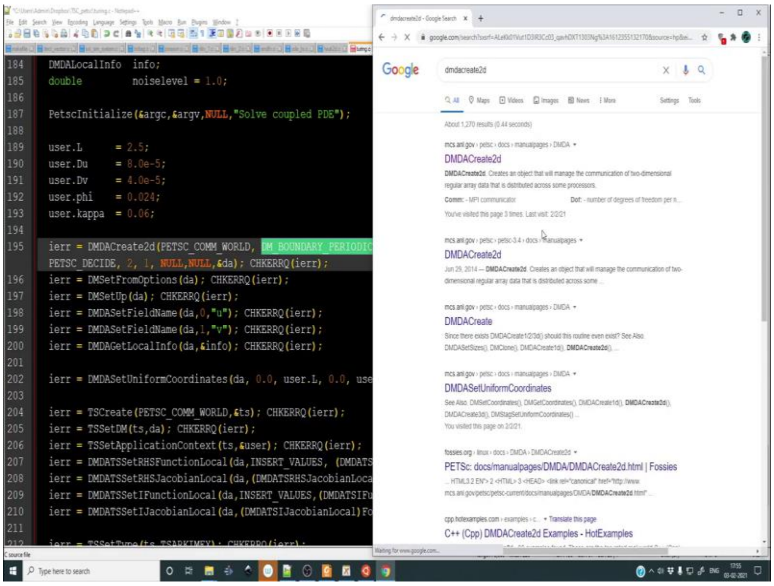
So, D u was 2 10 to the power minus 5 D v is 10 to the power minus 5. What I have chosen is 8 10 to the power minus 5, 4 10 to the power minus 5 and phi and kappa are kept to be 0.024 and 0.06. So, over here the various simulations are done, but where do we lie? 0.024, 0.6; 0.024, 0.6, somewhere over here ok, alright.

(Refer Slide Time: 20:53)

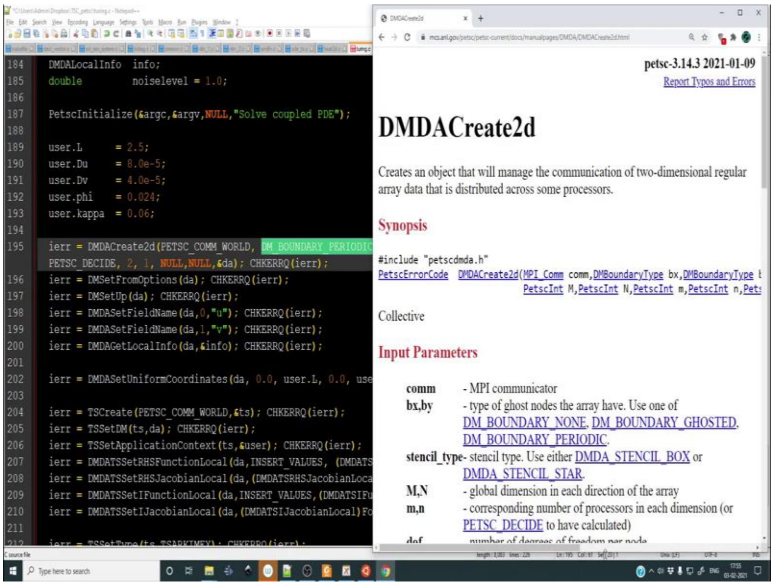


So, phi, kappa everything is defined then we create a DMDA and this is the entire creation of the DMDA. So, it contains a create 2D function and here the stencil box will be the stencil will be of the kind box, it is not going to be a star anymore ok. So, here the number of degrees of freedom is going to be 2 and that is it and here the boundaries are declared to be periodic.

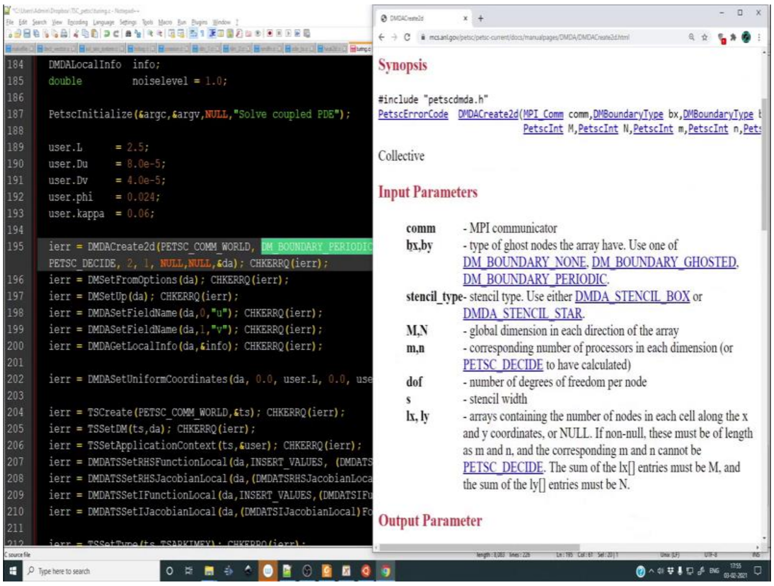
(Refer Slide Time: 21:27)



(Refer Slide Time: 21:42)



(Refer Slide Time: 21:44)



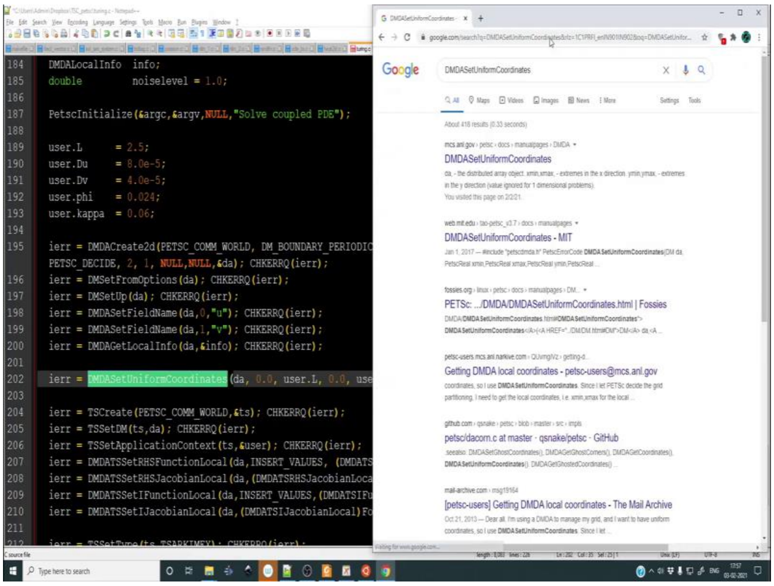
So, just for a reference just for reference, so, we have bx, by ok, let us zoom this out a bit. So, a stencil type which is stencil box then you have m comma n. So, in this case the default grid is 3 comma 3 and you have to decide let PETSC decide the number of the load balancing between the processes.

You have 2 degrees of freedom, stencil weight is 1, everything else is null null and you have to pass the address of the dm, da. This is we have done this plenty of times and there should be no doubt in this. Then you set from options in case you are passing commander arguments you need to allow this you set up the da.

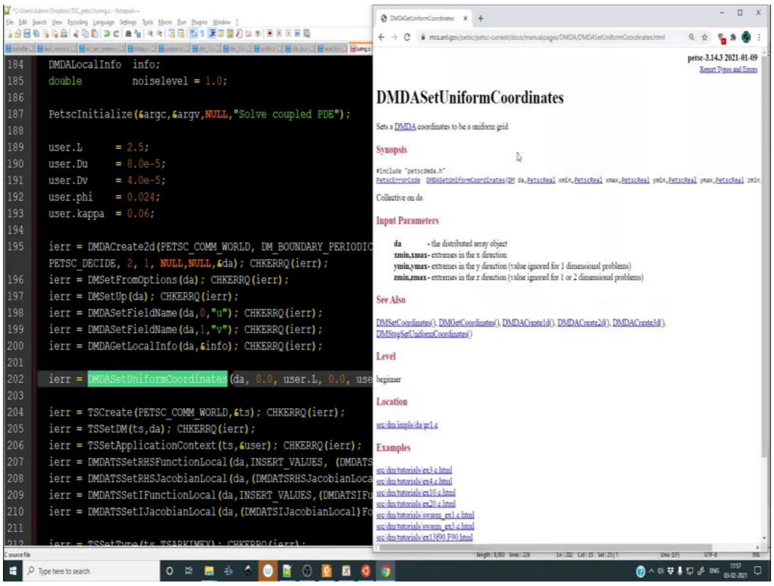
Now, you have to do DMDA set field name. So, because there is 2 degrees of freedom you are going to called the zero field as u and you are going to call the v field the second field as v. So, these two lines are used to define the alias for the 2 degrees of freedom over the entire grid.

So, the grid has 0 2 radial freedoms. So, the index of the 1st degree of freedom is 0 and it is it has an alias u. The index 1 has an alias equal to v and then you fetch the information about the grid because when you define the local functions for forming the RHS function on the Jacobian, you need to pass around the info object for the DMDA. So, that you can perform the loops and all that thing you can find out what h is going to be.

(Refer Slide Time: 23:24)



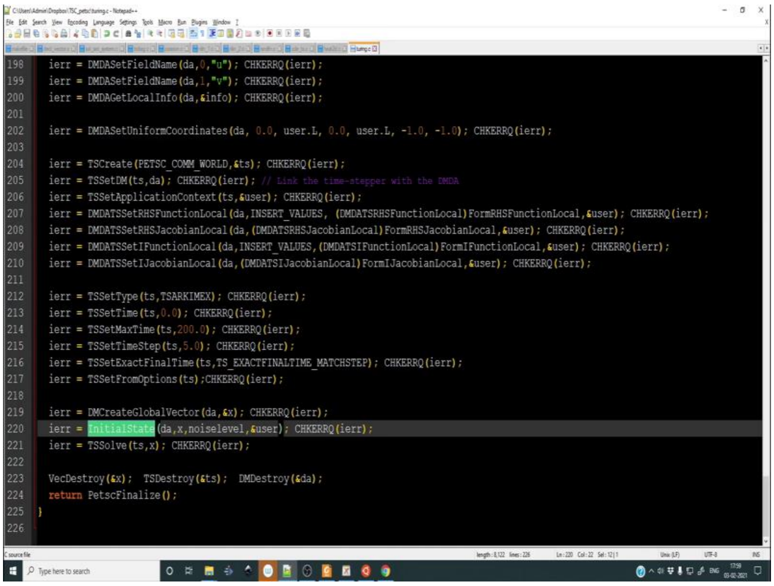
(Refer Slide Time: 23:27)



Now, here is something which we have not used so far, which DMDA I said in form coordinates. It is a very simple function which you can imagine what it means. It takes the default domain and it scales it between 0 and L. In this case L is 2.5. So, it is going to scale it from 0 to L.

So, you have to pass the da, you have to pass the xmin, you have to pass the length, you have to pass the ymin, you have to pass this and it does not matter what these two values are because it is a 2 dimensional problem because you are creating DMDA 2D. It does not matter what these values are. It is going to be ignored in the case of 2 dimensional problems. Then we are creating the time stepper. So, TSCreate PETSC COMM WORLD, SetDM.

(Refer Slide Time: 24:16)



So, this particular line is used to link the time stepper with the DMDA ok. So, ApplicationContext, so, you have to tell the time stepper that ok whatever parameters you are going to have they are stored under the variable called as users, you have to pass the address of user.

Then you have DMDA time stepping set function ok. So, here you are setting the RHS functions and the Jacobians ok. So, here you are doing the setting of the RHS function and the Jacobians ok and the time stepper type is ts RKIMEX which is which stands for Adaptive Runge Kutta Implicit Explicit.

So, its a advanced time stepping routine, but we can use the Crank Nicolson on anything, but in this case this gives us the best performance I have written it like this. So, you set the time parameters that is the initial time, the max time, the time step and so on, you can make this lower if you want and you have to match the final point with the time step. TSSet from options in case you are passing command line arguments. After this you have DMCreate vector global vector da.

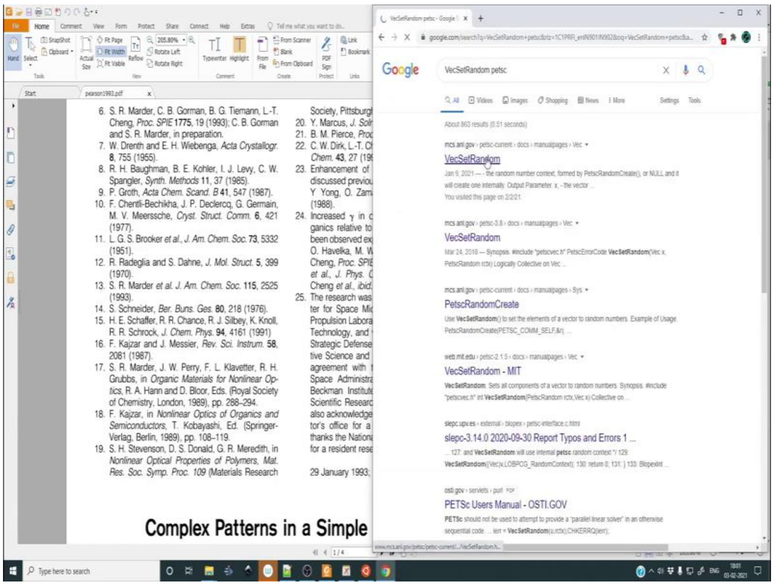
So, this is fine. So, you are creating a vector x based on the DMDA grid then you create an initial state. This is quite important because obviously, 1 comma 0 is going to be a solution. So, if you said u equal to 1 and v equal to 0, both these equations the LHS and RHS they match. It means that the system is having a trivial solution of u equal to 1 and v equal to 0. No or is it u equal to 1, u equal to 1 and v equal to 0 is a trivial solution.

So, in that case that is not of much interest to us and that is why we have to give a perturbation to the initial condition and that is what will be done through the function InitialState. So, in order to find the InitialState we are going to pass. So, these are all functions which we have to create ok. We will pass the DM da, we will pass the vector x which will be iterated in time and we will pass the noiselevel and we will pass also the set of parameters that we have stored in the structure AppCtx called user.

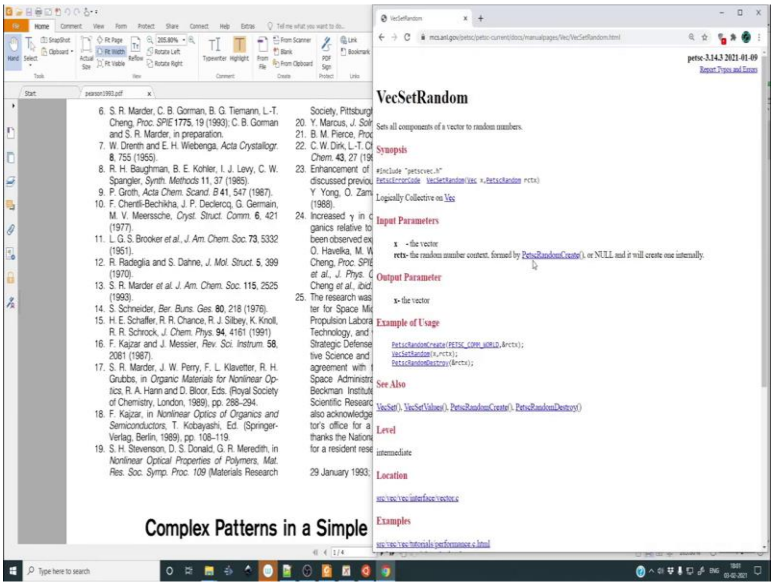
Then you solve it, then you destroy the variables, then you finalize that is it. It does not appear to be much more complicated than what we have done so far, but now let us look at the functions. So, first things first. Let us create the initial state. So, we have as an input to the function da, the vector y, the noiselevel and the AppCtx which contains all the different parameters of the problem.

So, I am going to just tell you the logic what is going on. You set the vector to 0 both the u and v are their initialized to 0 through this point. After that you set a random value of y or you sort of set it to y, but then you scale that to the appropriate noise level that is you first set the random value.

(Refer Slide Time: 27:55)

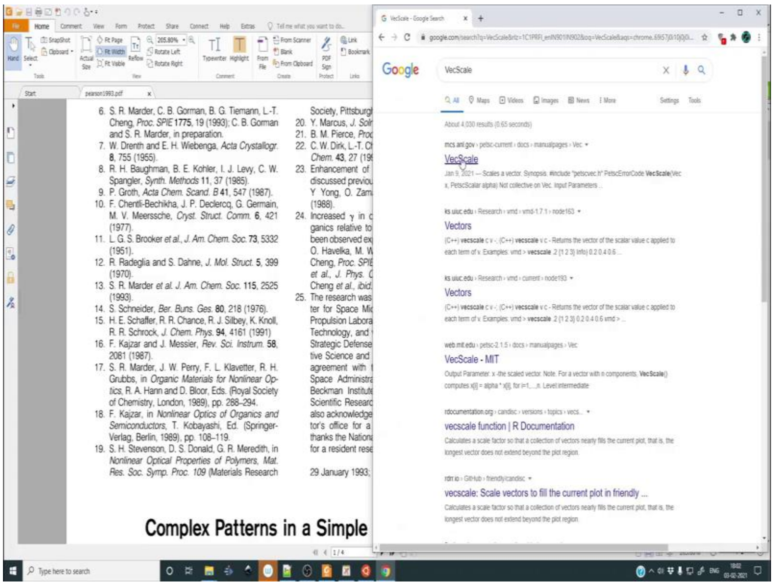


(Refer Slide Time: 27:57)

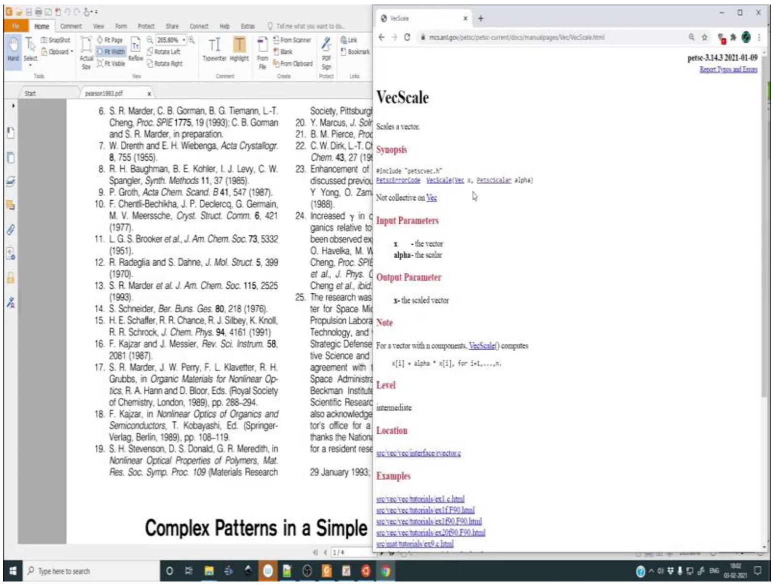


So, let me show that function reference. Set all the components of a vector to random values ok. After that you scale them to the appropriate noise level.

(Refer Slide Time: 28:14)



(Refer Slide Time: 28:17)

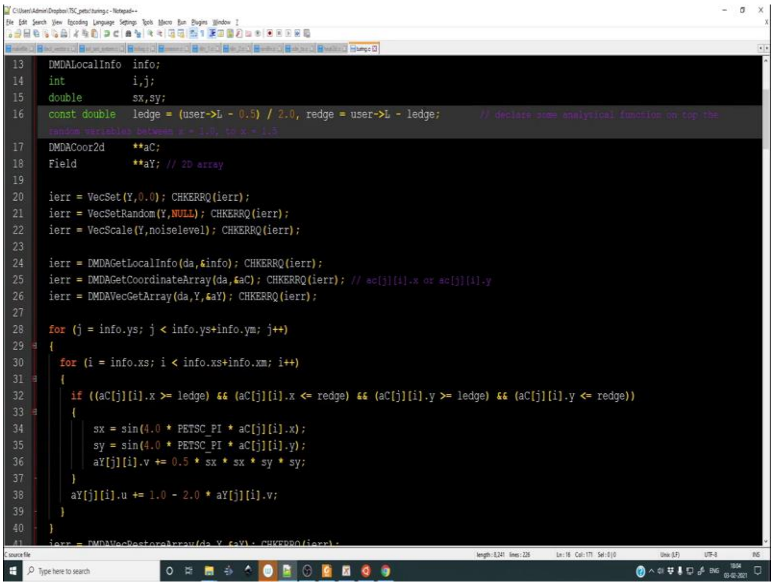


So, VecScale will scale it to the whatever the random values are. It will try to linearly scale it between 0 to the scaled vector. So, in this case we can make it 1 or you can make it even smaller ok. So, that is governed by the scalar noise level and in the main I have defined noise level to be 1, but you can keep changing it and recompiling it you will get new solutions.

Well, not really new solutions, but a difference in the time taken to evolve to the solution ok that is all going to change. So, then you get the Localinfo get the CoordinateArray.

So, you save the CoordinateArray in the from the DMDA into a variable called ac which is a second 2D array. So, that will contain all the xs and ys. And how do you call the xs and ys. So, inside this ac j comma i dot x.

(Refer Slide Time: 29:07)



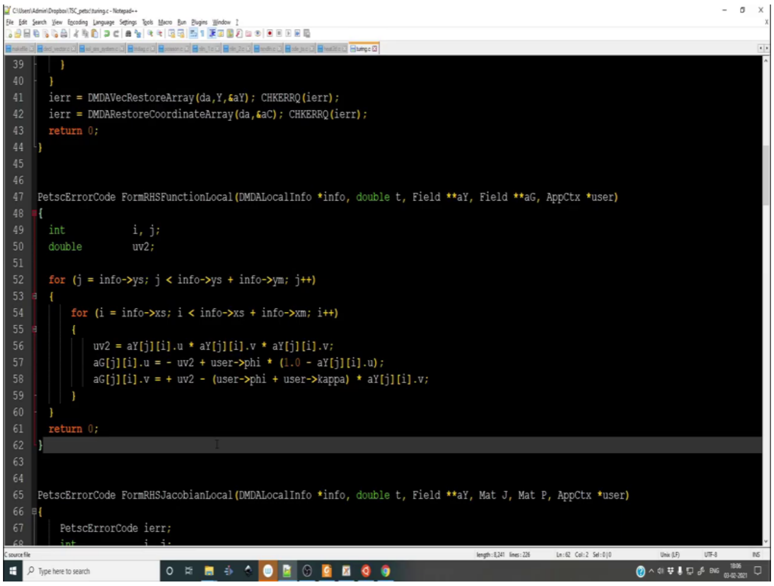
So, the calling sequence for the coordinates is going to be ac j i dot x or ac j i dot y. So, this is going to give you the local x and y coordinates something which can imagine to be of the kind mesh grid in python ok. Then you get the array. Then you assign it with a random number; not a random number you have already assigned everything to a random number, but in between L edge and R edge.

So, L edge and R edge are at the left edge and the right edge in the problem ok. So, it is going to be between 1 and 1.5 because user dot L is going to be 2.5. So, 2.5. So, here what we have is declare some analytical function on top of the random variables between x equal to 1 point rather x equal to 1.0 to x equal to 1.5.

So, you are going to set that to a random number. You are going to set that zone that strip with a defined function. In this case it is going to be sin 4 pi x square times sin 4 pi y square times half and you are going to sum it over all the domains that is like the not the domain, but the strip while for you are going to simply do 1 minus 2 v.

So, that is the initial condition for u. So, with the help of this you can create some perturbations in the initial condition. Lastly you must restore both the coordinate array and the on the unknown array that is u and v ok.

(Refer Slide Time: 31:12)

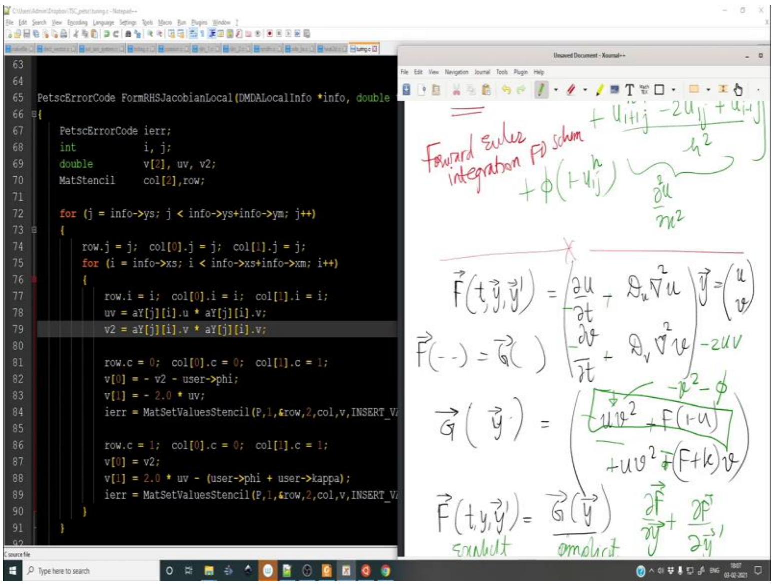


So, what about form RHS function local? Because its being done on the function G is going to be done on the function G, so, what is the function G ok? So, its going to be u v square minus F times 1 minus u. Well, over here I think I have taken the negative sign meaning. I have made this and this.

So, I have just it does not make any difference, it is nothing is going to change. So, it is minus uv square plus phi times 1 minus u and uv square times uv square minus phi times phi plus kappa times v. So, this is obviously, going to be a v times a v a y j comma a comma v. This is how you form the RHS function.

How do you find Jacobian? Its quite simple again. It is going to simply be Jacobian of, so, it is the its going to be the Jacobian of this guy which is going to be. So, del u; this is going to be what? Minus v square comma 0. So, let us see where I have written it. So, you are looping over the entire grid.

(Refer Slide Time: 33:15)

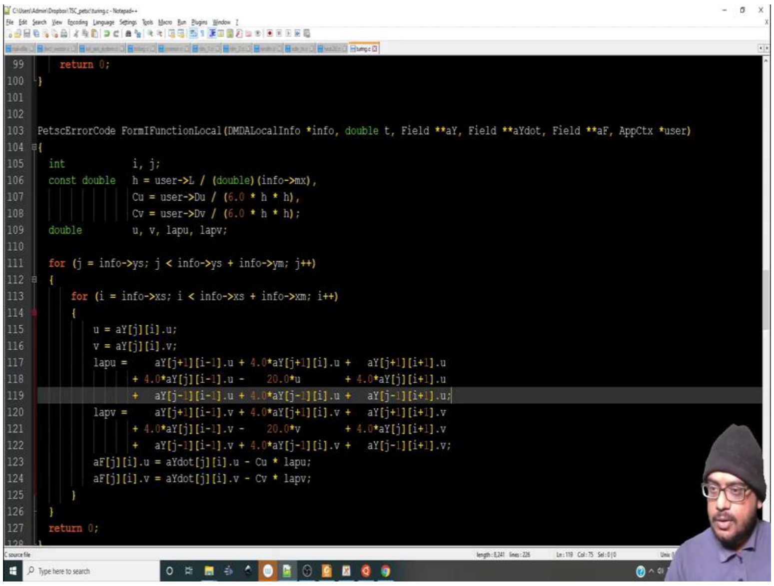


So, u v v 2. So, they are defined as u times v and v square. So, this is going to be minus v 2 minus F minus phi times u. So, when you take a derivative of this function with respect to u you get minus v square minus phi which appears over here then this is minus 2 u v. So, when you take a derivative of this with respect to v, you have minus 2 uv which is appearing over here.

So, uv is already declared as this double, it is a local variable inside a function. So, it does not matter then MatSetValuesStencil P comma 1 comma row comma 2 comma col. So, this is just to set the preconditioner. Similarly, you have this and you can easily verify it from the derivatives of that function this particular function and you set it and like always when you declare a Jacobian you are going to fill in the preconditioner.

Once you filled in the t preconditioner if J is not equal to P, you assemble the Jacobian anyway. If they are equal you essentially are assembling the preconditioner which is also equal to the Jacobian and we have discussed this in the previous lectures.

(Refer Slide Time: 34:22)

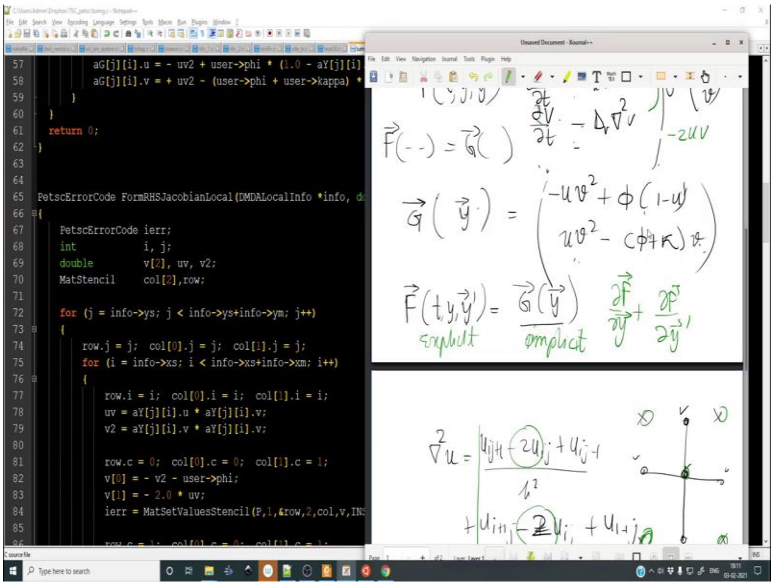


Now, we come to the implicit part that is we have to form the functional and not the functional the local function. So, let us see what we get ok. So, the derivative, so, let us see. So, you have this stencil, look at this stencil. This is exactly this particular stencil that I showed over here ok.

So, its Laplacian of u, Laplacian of v, but finally, aF u will be a dot u minus the diffusion coefficient times Laplacian of u. Essentially, its this term ok and the other function is going to be v dot minus Laplacian of v. Actually you are writing both these terms in terms of the in terms of u dot and u dot and v dot, not just the us and vs and hence because it is an implicit function you have to do all that in order to define it ok.

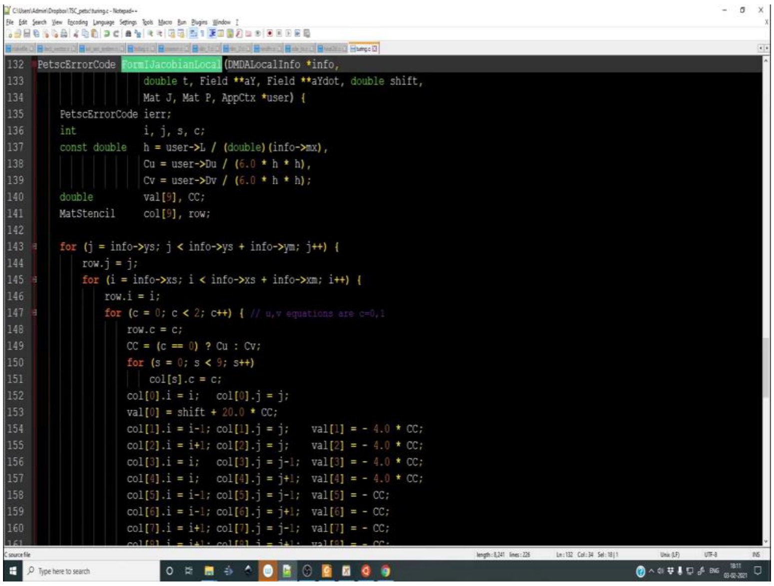
And what about the Jacobian for this? So, here comes the shifted Jacobian. Just a quick, let me just fix this real quick. If I want to write it in terms of F equal to G. So, let me just tidy this up a bit. I mean although the discussion is still valid, I just want to clearly write it.

(Refer Slide Time: 36:54)



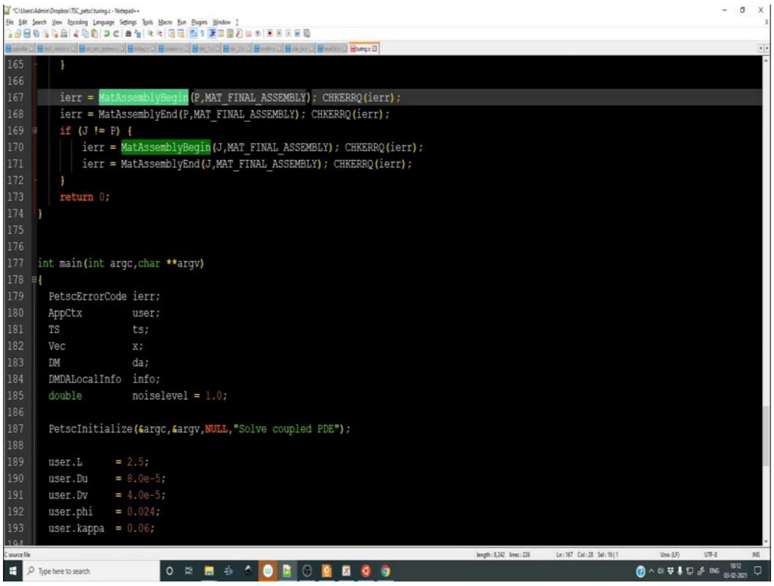
So, this will be del u del t minus the Laplacian of u. This will be del v del t minus Laplacian of v and this will be minus u v square that is this plus phi 1 minus u and this will be u v square minus phi plus kappa times v yeah ok. So, now we have to form the implicit Jacobian.

(Refer Slide Time: 37:29)



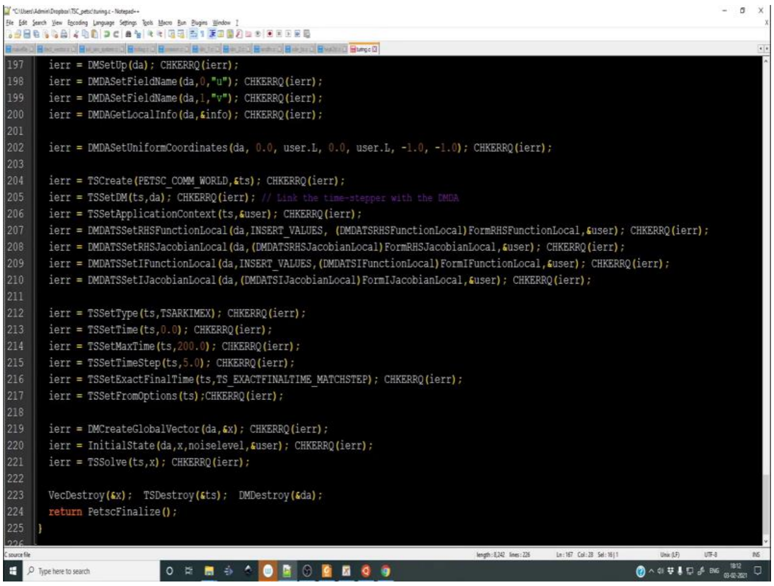
Now, the implicit Jacobian will actually be written in this form. It is going to be dF dY dot plus dF dY and its not at all difficult. So, these are all the sub matrix of differentiation. Now, remember whenever you want to loop over the two variables you need to do a final loop like this ok. So, each row value. So, when once you want to loop over a certain variable you must say row dot c equal to either 0 or 1 because you want to loop over u and v separately ok.

(Refer Slide Time: 38:15)



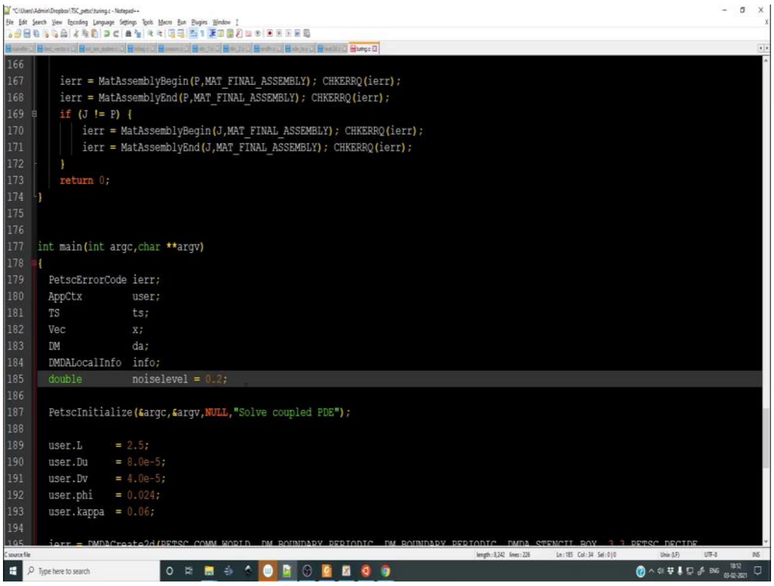
But, in internally it always sets the variable correctly and there is no issue in that. Then you do the same thing. SetStencil then AssemblyBegin AssemblyClose, if J is not equal to the preconditioner you assemble j anyway, but we have already done it.

(Refer Slide Time: 38:27)

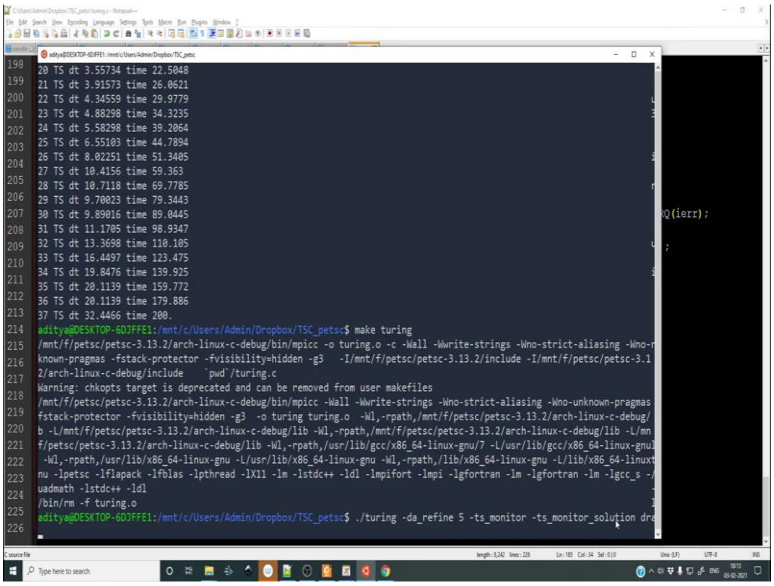


So, the functions are not that difficult and we have set an implicit, explicit problem. So, once everything is set I guess all that is left is to run the program.

(Refer Slide Time: 38:42)

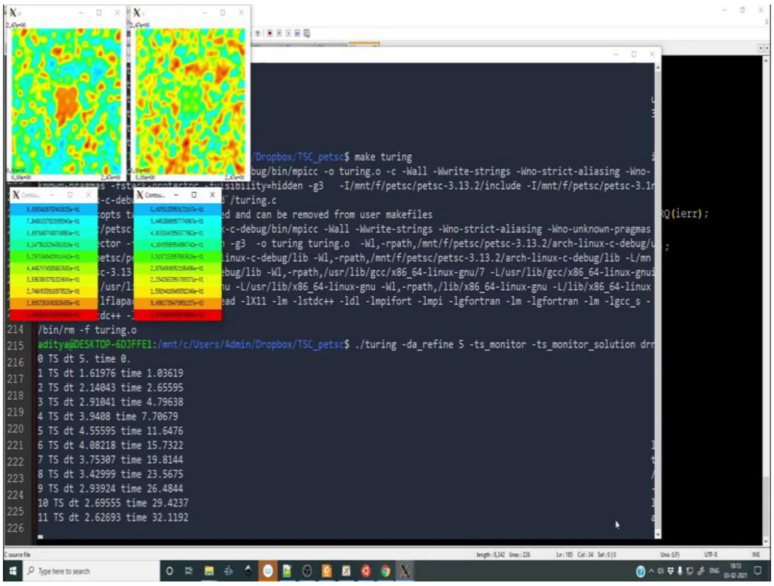


(Refer Slide Time: 38:50)

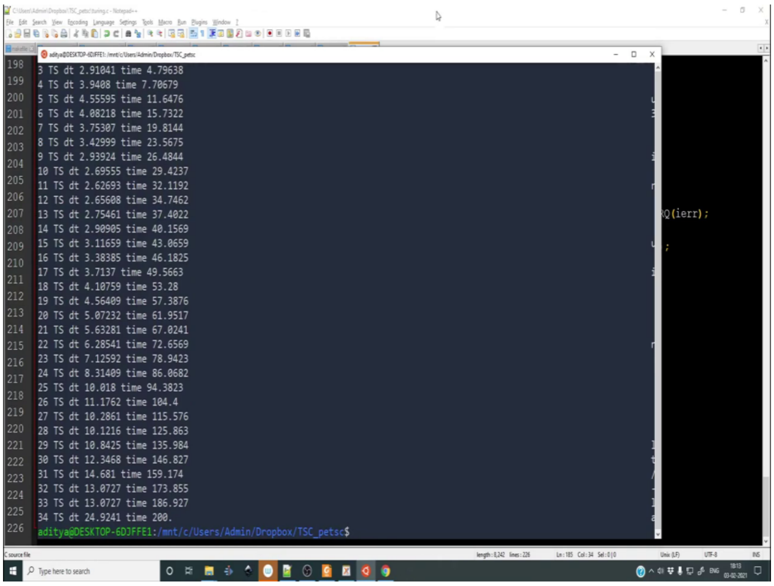


So, let me just make the noiselevel a bit low 0 point maybe 0 2, will increase the time later, but let me make turing. So, I have called the program as turing dot c and I also gone ahead and modified the make file then we do dot slash turing. So, da refine is because we have defined a 3 cross 3 grid, it is quite small. So, we refine it 5 orders of magnitude, ts monitor, ts monitor solution draw. So, whatever the solution we are going to get we are going to draw the solution as well.

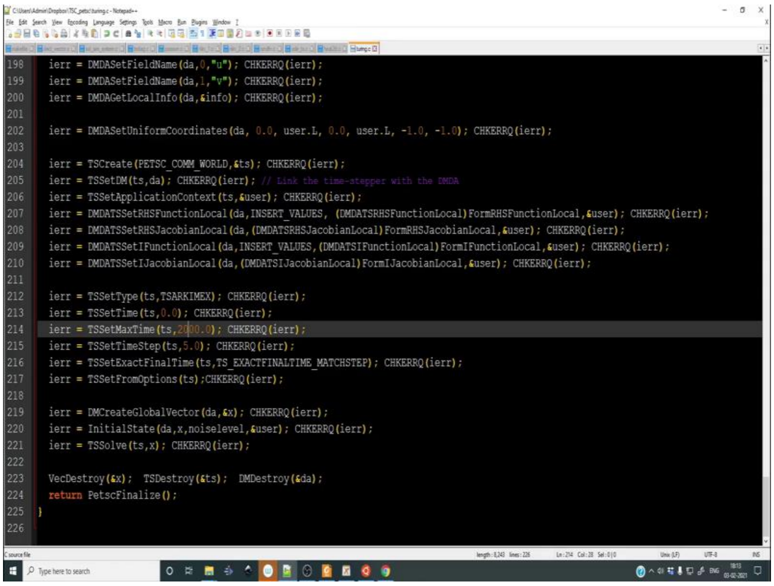
(Refer Slide Time: 39:14)



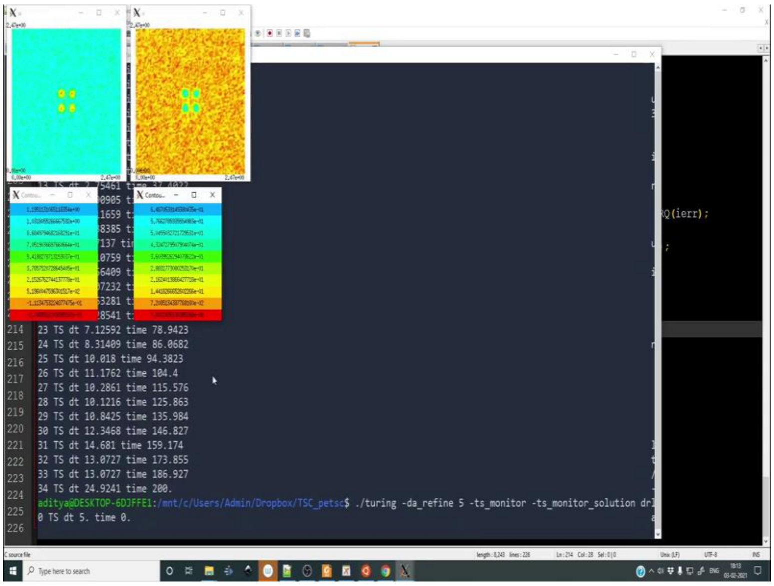
(Refer Slide Time: 39:20)



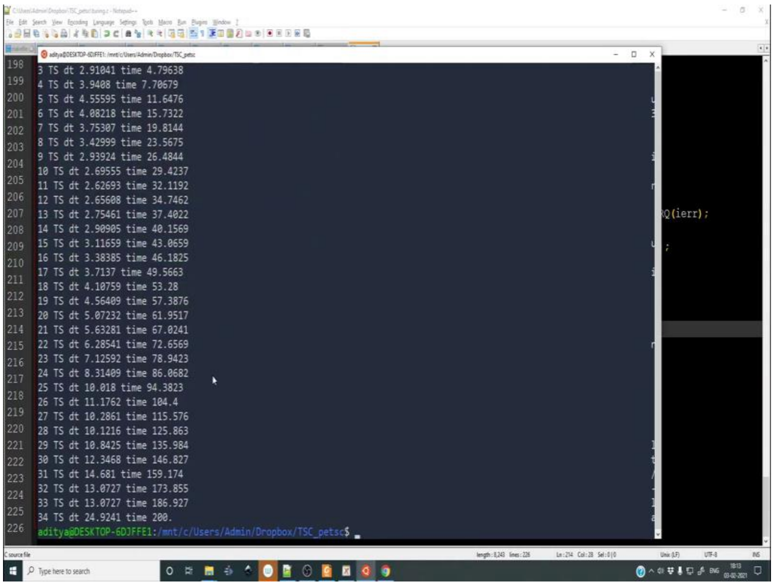
(Refer Slide Time: 39:30)



(Refer Slide Time: 39:38)

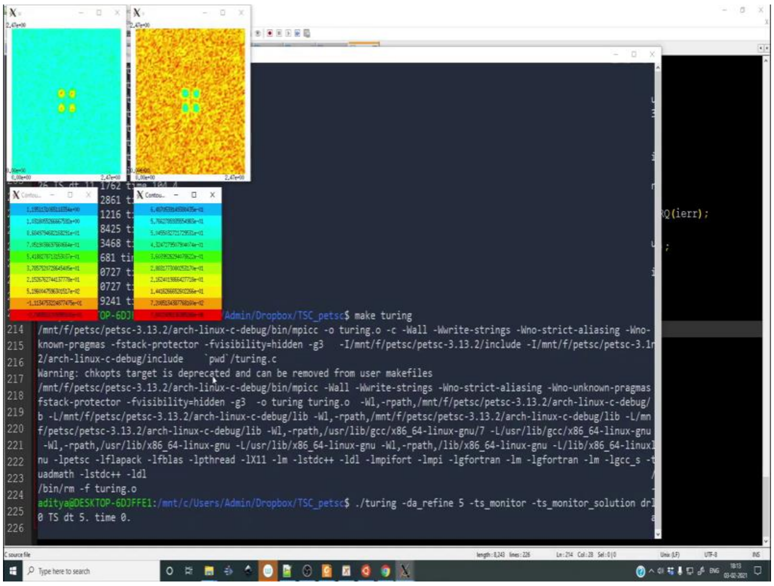


(Refer Slide Time: 39:41)



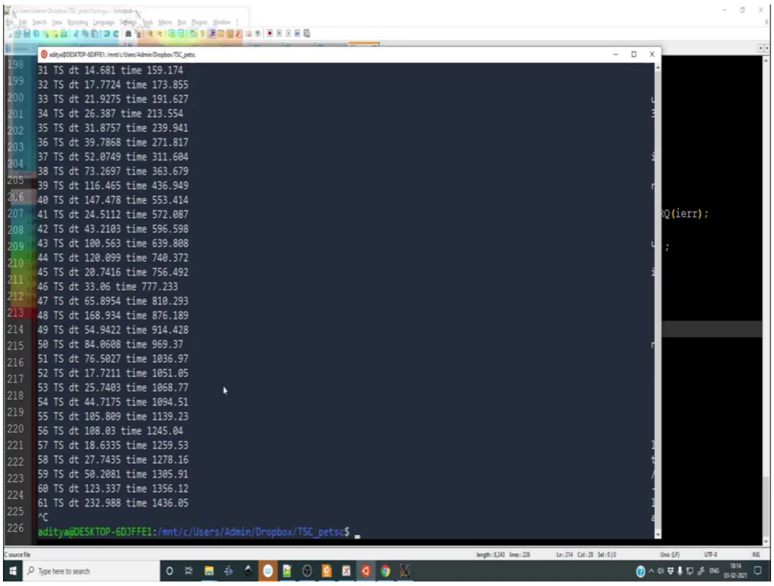
So, let us run this and let us see what happens ok. There you have it. We have weird looking oscillations ok. Maybe I need to increase the number of time steps to maybe say 2000. And what is the time step? The time step is fine.

(Refer Slide Time: 39:50)



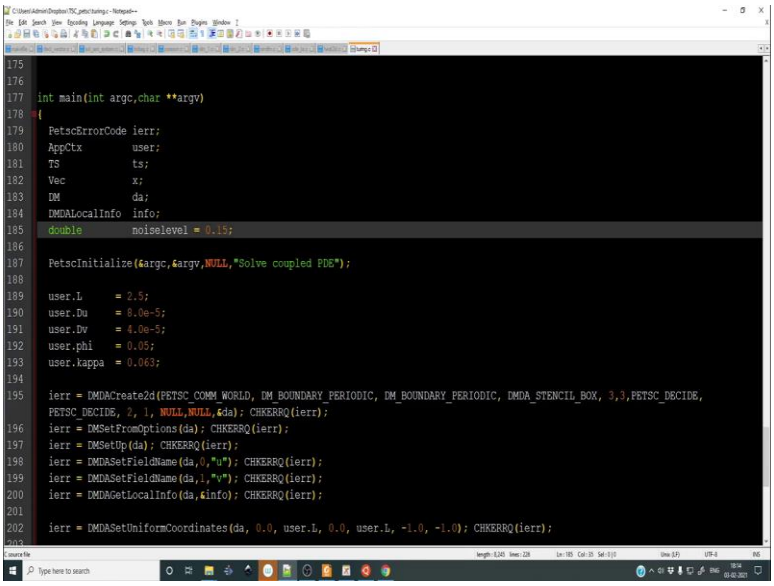
So, the initial condition was those 4 loops nothing else. I forgot to make it.

(Refer Slide Time: 40:09)

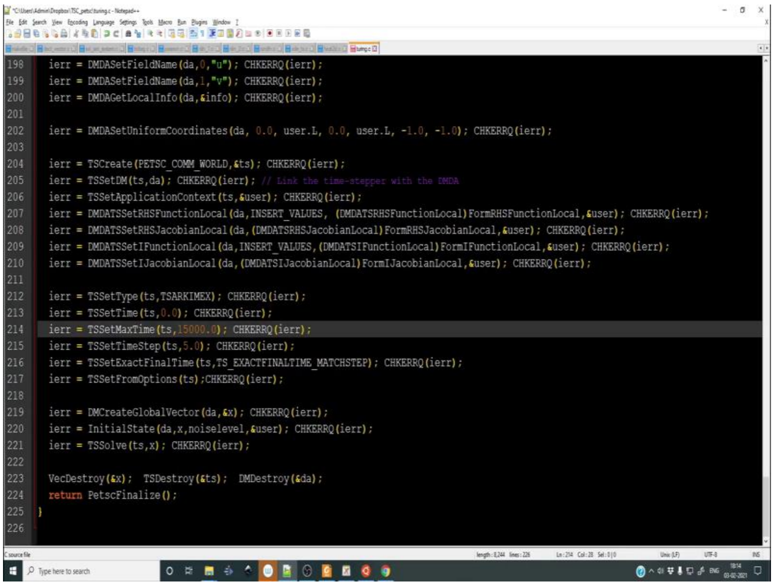


You have these oscillating solutions. It is not going to really die down. I guess we can stop. Let me just change some of the parameters.

(Refer Slide Time: 40:23)

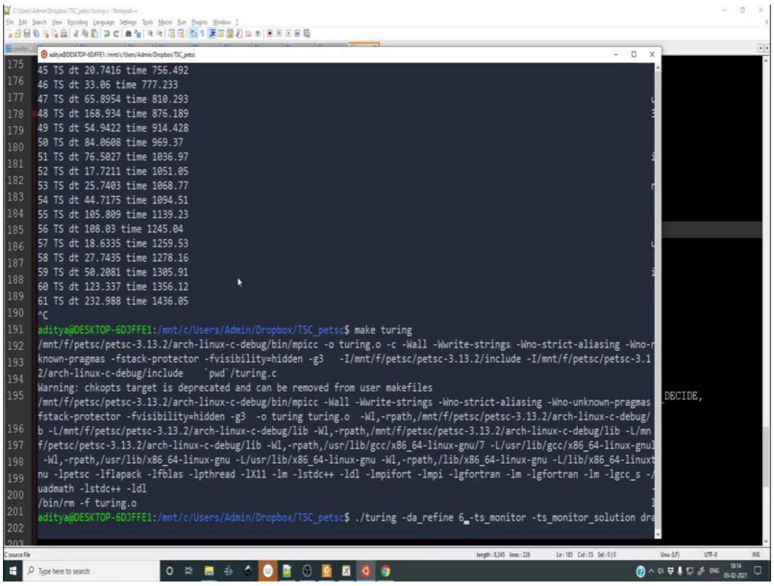


(Refer Slide Time: 40:35)

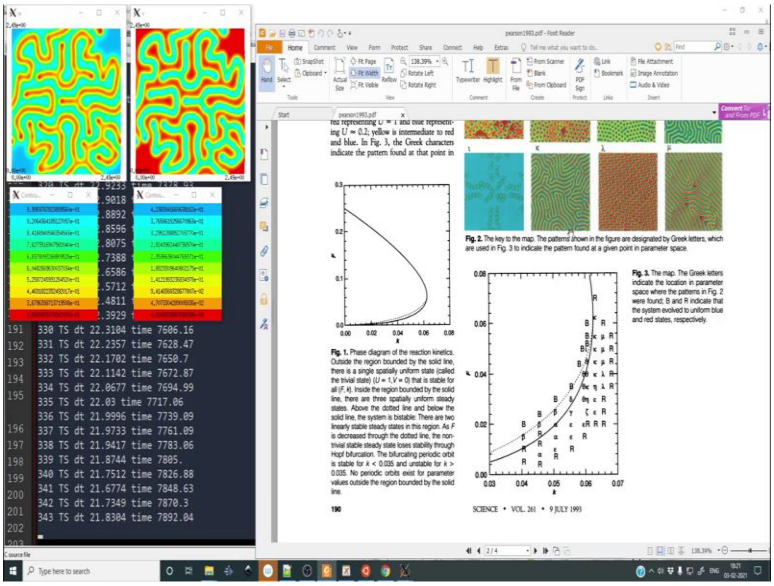


So, let me change 5 to 0.05 and let me change kappa to 0.063. The diffusion coefficients can remain the same. Let me change the MaxTime to something much larger maybe say 15000 and let me change the noise pattern to 0.15.

(Refer Slide Time: 40:48)



(Refer Slide Time: 40:57)



So, let us see let us make the file. Let me refine it. Maybe let me refine it a bit more. Let us see if we can get something fantastic out of this. We have chosen phi to be 0.05 and kappa to be 0.063 something in this region. So, we expect a kappa kind of pattern that is this Labyrinth pattern. It is almost like one of those Labyrinth mazes that you might have seen in the movie shining where Jack Nicholson is finally, stuck in a Labyrinth like this.

In fact, such patterns are also formed in ferrofluids when you subject it subject them to a uniform magnetic field. This is the kind of Labyrinth instability they form as well. Let us see whether it evolves to that. So, it does seem to evolve towards something everything. I think we need to let it run for a while.

So, its like the lobes are expanding outwards and at some point it should start folding onto itself after which the pattern will start forming that kind of a Labyrinth nature. Well, while this program runs I will take this opportunity to end this class over here. The video will contain the rest of the evolution. It will be (Refer Time: 43:29).