

**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)

about the center of the grid was then perturbed to  $(U = 1/2, V = 1/4)$ . These conditions were then perturbed with  $\pm 1\%$  random noise in order to break the square symmetry. The system was then integrated for 200,000 time steps and an image was saved. In all cases, the initial disturbance propagated outward from the central square, leaving patterns in its wake, until the entire grid was affected by the initial square perturbation. The propagation was wave-like, with the leading edge of the perturbation moving with an approximately constant velocity. Depending on the parameter values, it took on the order of 10,000 to 30,000 time steps for the initial perturbation to spread over the entire grid. The propagation velocity of the initial perturbation is thus on the order of  $1 \times 10^{-3}$  space units per time unit. After the initial period during which the perturbation spread, the system went into an asymptotic state that was either time-independent or time-dependent, depending on the parameter values.

parameters space. There are two additional solutions in Fig. 3, R and B, indicating spatially uniform red and blue states, respectively. The red state corresponds to  $(U = 1, V = 0)$  and the blue state depends on the exact parameter values but corresponds roughly to  $(U = 0.3, V = 0.25)$ .

Pattern  $\alpha$  is time-dependent and consists of filigree spirals that are constantly colliding and annihilating each other. Full spirals never form. Pattern  $\beta$  is time-dependent and consists of what is generally called phase turbulence [8], which occurs in the vicinity of a Hopf bifurcation to a stable periodic orbit. The mechanism is unable to synchronize so the phase of the oscillations varies as a function of position. In the present case, the small-amplitude periodic orbit that bifurcates is unstable. Pattern  $\gamma$  is time-dependent. It consists primarily of stripes but there are small localized regions that oscillate with a relatively high frequency ( $\sim 10^{-3}$ ). The active regions disappear, but new ones always appear elsewhere. In

(Refer Slide Time: 00:37)

**Complex Patterns in a Simple System**

John E. Pearson

Numerical simulations of a simple reaction-diffusion model reveal a surprising variety of irregular spatiotemporal patterns. These patterns arise in response to finite-amplitude perturbations. Some of them resemble the steady irregular patterns recently observed in thin gel reactor experiments. Others consist of spots that grow until they reach a critical size, at which time they divide in two. If in some region the spots become overcrowded, all of the spots in that region decay into the uniform background.

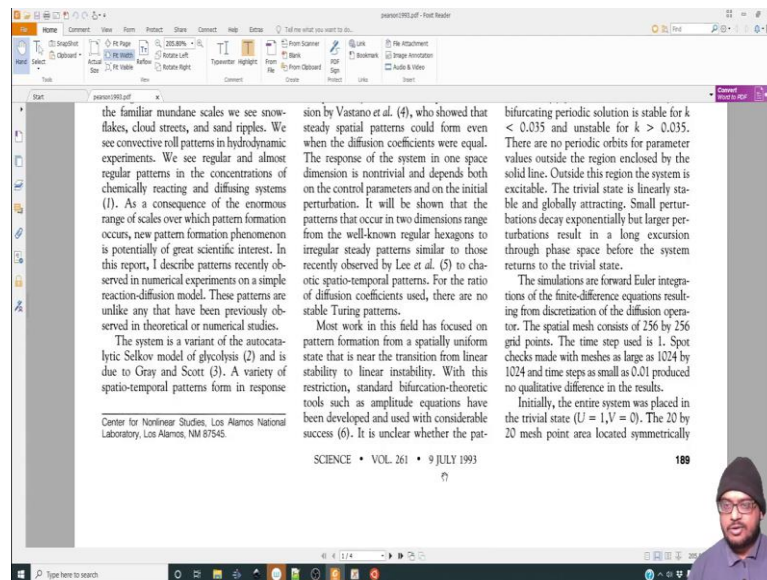
Patterns occur in nature at scales ranging from the developing *Drosophila* embryo to the large-scale structure of the universe. At the familiar mundane scales we see snowflakes, cloud streets, and sand ripples. We see convective roll patterns in hydrodynamic experiments. We see regular and almost regular patterns in the concentrations of chemically reacting and diffusing systems (1). As a consequence of the enormous range of scales over which pattern formation

to finite-amplitude perturbations. The response of this model to such perturbations was previously studied in one space dimension by Vastano *et al.* (4), who showed that steady spatial patterns could form even when the diffusion coefficients were equal. The response of the system in one space dimension is nontrivial and depends both on the control parameters and on the initial perturbation. It will be shown that the patterns that occur in two dimensions range

$2 \times 10^{-3}$  and  $D_2 = 10^{-5}$ . 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.

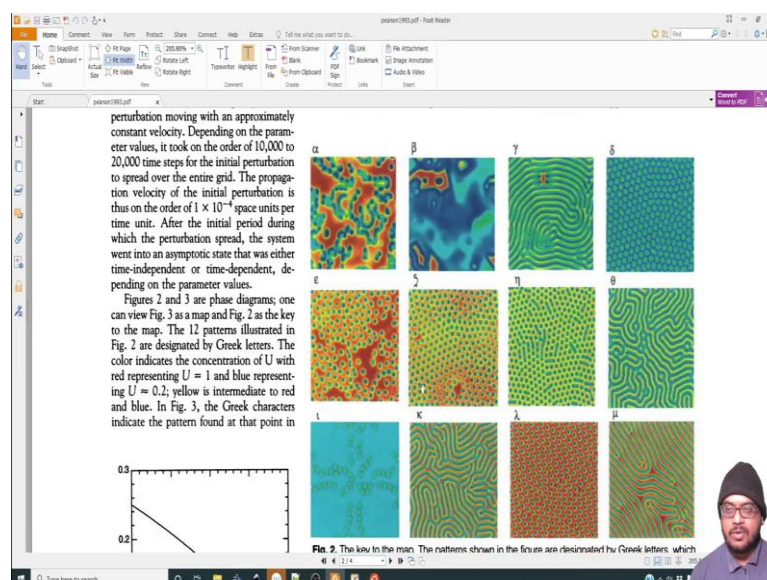
In the phase diagram shown in Fig. 1, a trivial steady-state solution  $U = 1, V = 0$  exists and is linearly stable for all positive  $F$  and  $k$ . In the region bounded above by the solid line and below by the dotted line, the system has two stable steady states. For fixed  $k$ , the nontrivial stable uniform solution loses stability through saddle-node bifurcation as  $F$  is increased through the upper solid line or by Hopf bifurcation to a periodic orbit as  $F$  is decreased through the dotted line. For a discussion of bifurcation theory, see chapter 3 of (7). In the case at hand, the bifurcating periodic solution is stable for  $k < 0.035$  and unstable for  $k > 0.035$ . There are no periodic orbits for parameter values outside the region enclosed by the solid line. Outside this region the system is excitable. The trivial state is linearly stable and globally attracting. Small perturbations decay exponentially but larger per-

(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)

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  $D_U = 2 \times 10^{-3}$  and  $D_V = 10^{-3}$ . 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

**Complex Patterns in a Simple System**

John E. Pearson

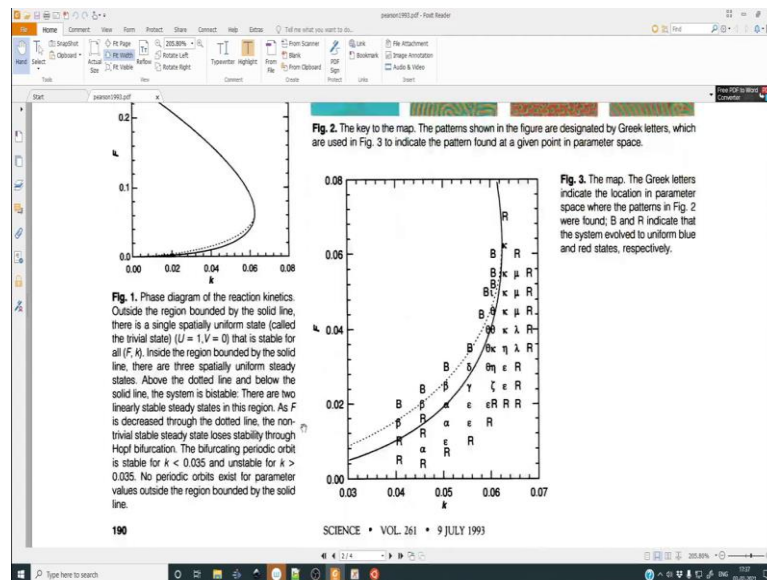
So, such kinds of patterns they were shown by Pearson that such a system is able to give rise to these complicated patterns. And in this lecture I am going to show an overview of the program and the reason I say overview is because this particular code would take a lot of time to write much more than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

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u - uv^2 + \phi(1-u)$$

$$\frac{\partial v}{\partial t} = D_v \nabla^2 v + uv^2 - (\phi+k)v$$

reaction sources terms

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

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

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  $D_u = 2 \times 10^{-3}$  and  $D_v = 10^{-3}$ . 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 trivial steady-state solution  $U = 1, V = 0$  exists and is linearly stable for all positive  $F$  and  $k$ . In the region bounded above by the solid line and below by the dotted

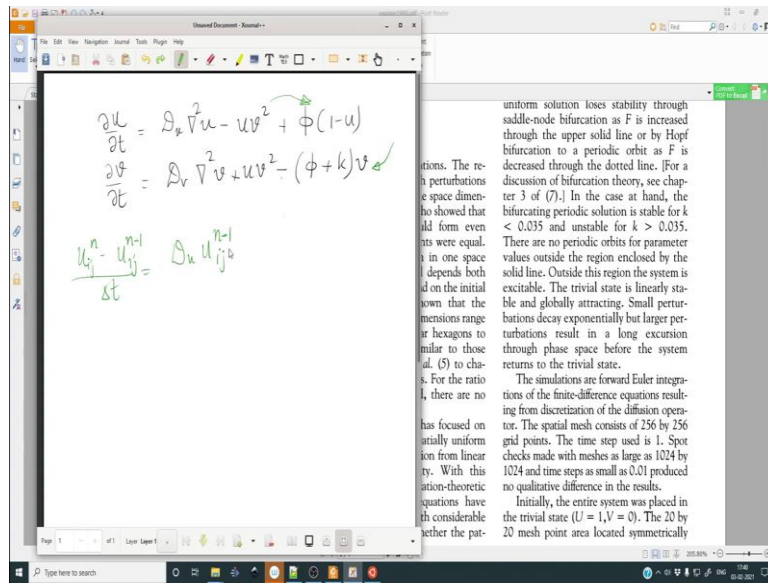
And so, let us see how we can go about solving this particular equation. So, the equation at hand is  $\partial u / \partial t = D_u \nabla^2 u - uv^2 + \phi(1-u)$  and  $\partial v / \partial t = D_v \nabla^2 v + uv^2 - (\phi+k)v$ . So, these are the two systems.

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  $v$ ?

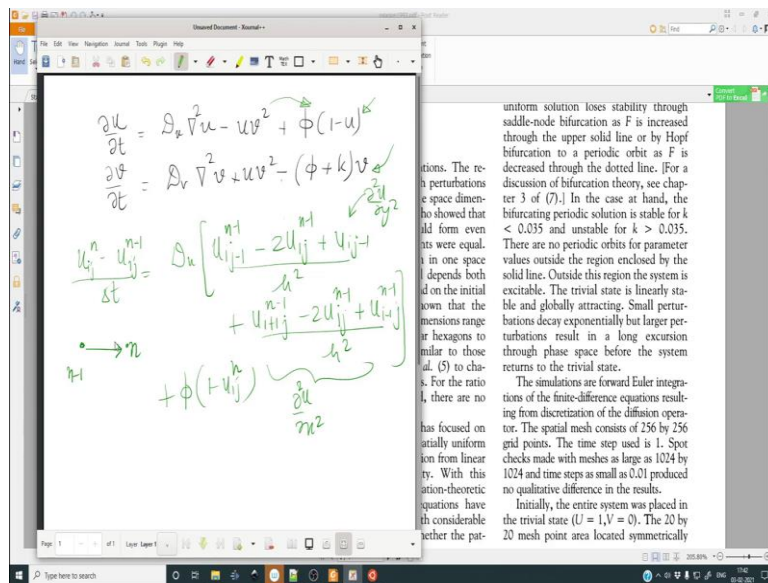
So, there is a steady decay you a bulk decay which is proportional to the local concentration and this gives rise to an evolution of the equation  $v$ . So, how can we solve such a system? Well, as has been told in the article itself you can simply perform a discretization.

(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)

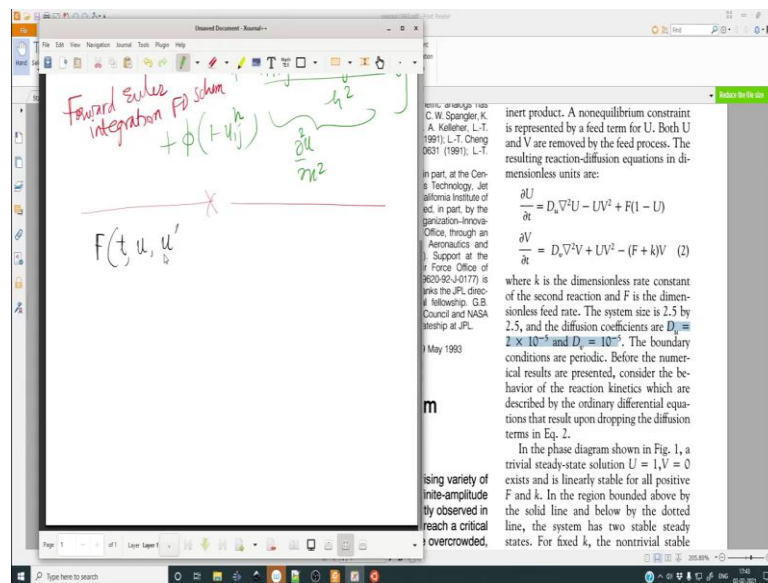


So, plus  $D u_{ij} - 1 + u_{ij} - 2 u_{ij} + u_{i,j+1} + u_{i,j-1}$ . So, this these are all at  $n$  minus 1 times step plus  $u_{ij} - 1$  upon  $h^2$  plus  $u_{i+1,j}$  and minus 1 minus 2

$u_{ij}^{n-1} + u_{i-1,j}^{n-1}$  at time  $n-1$  upon  $h^2$ . So, you can see that this is the  $\Delta^2 u / \Delta x^2$  term while this is the  $\Delta^2 u / \Delta t^2$  term.

Well, now what about this? This is simply  $\phi$  times  $1 - u_{ij}^{n-1}$ . 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  $n-1$  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  $t, u, u'$  or if I write it down in terms of the  $y$  vector.

(Refer Slide Time: 09:44)

The slide contains handwritten notes in red and green ink on a white background. At the top, it says "Forward Euler integration" and "FD scheme". Below this, there are several equations:

$$\vec{F}(t, \vec{y}, \vec{y}') = \begin{pmatrix} \frac{\partial u}{\partial t} - D_u \nabla^2 u \\ \frac{\partial v}{\partial t} - D_v \nabla^2 v \end{pmatrix} \vec{y}' = \begin{pmatrix} u \\ v \end{pmatrix}$$

$$\vec{G}(t, \vec{y}) = \begin{pmatrix} u^2 - F(1-u) \\ -uv^2 + (F+k)v \end{pmatrix}$$

There are also some smaller equations and terms like  $\phi(t, y, y')$  and  $\frac{\partial u}{\partial t}$ .

On the right side of the slide, there is a printed text block from a research paper, which includes the following equations:

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

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

The text discusses reaction-diffusion equations and boundary conditions.

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)

This slide is similar to the previous one, but with more handwritten equations. It shows the simplification of the vector G:

$$\vec{G}(\vec{y}) = \begin{pmatrix} u^2 - F(1-u) \\ -uv^2 + (F+k)v \end{pmatrix}$$

$$\vec{F}(t, \vec{y}, \vec{y}') = \vec{G}(\vec{y})$$

The printed text on the right is the same as in the previous slide, discussing reaction-diffusion equations and boundary conditions.

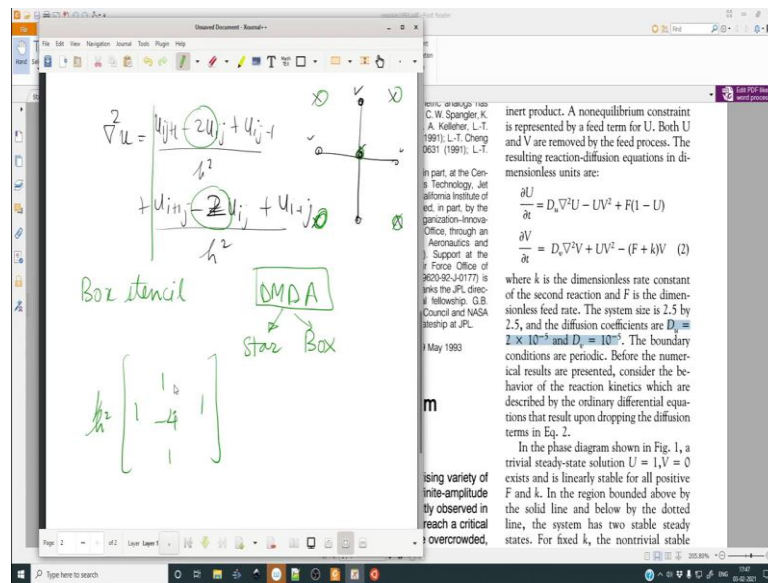
And simply write it as y comma y. It is in fact, not even function of y prime. So, I can simply write it as G of y. So, this would be equal to. So, we want to cast it in the form F of whatever it is equal to G of whatever it is. So, then these two terms have to go on the



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  $u_{ij+1}$  minus  $2u_{ij}$  plus  $u_{ij-1}$  by  $h$  square plus  $u_{i+1j}$  minus  $2u_{ij}$  plus  $u_{i-1j}$  upon  $h$  square.

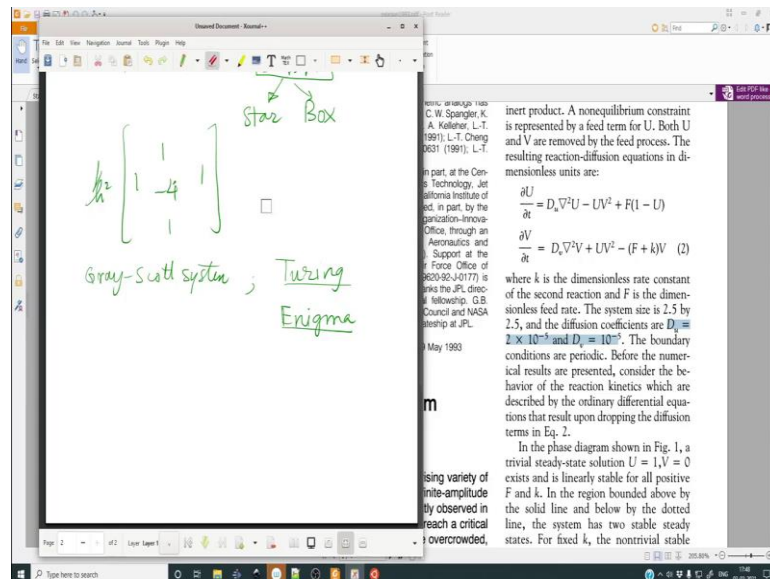
So, when its like this you are writing it simply in terms of these terms while disregarding the contributions from this term and usually its fine nothing fancy happens, but in such cases where there is diffusion and all that you want to preserve the isotropy of the problem. There is another stencil which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

The image shows a presentation slide with handwritten notes and a printed text snippet. The handwritten notes include:

- A diagram of a "Star Box" with arrows pointing to a matrix.
- A matrix: 
$$\frac{1}{h^2} \begin{bmatrix} 1 & & \\ & -4 & \\ & & 1 \end{bmatrix}$$
- The text "submatrix differentiation".
- The text "Gray-Scott system ; Turing".
- A larger matrix: 
$$\frac{1}{6h^2} \begin{bmatrix} 1 & & 4 & & \\ & 4 & & -20 & 4 \\ & & & & \\ & & & 4 & \\ -25 & & & & 1 \end{bmatrix}$$
- A diagram of a 2D grid with a central point and arrows pointing to it, with values -25 and 2.5 indicated.

The printed text snippet on the right side of the slide includes:

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  $D_u = 2 \times 10^{-3}$  and  $D_v = 10^{-2}$ . 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 trivial steady-state solution  $U = 1, V = 0$  exists and is linearly stable for all positive F and k. In the region bounded above by the solid line and below by the dotted line, the system has two stable steady states. For fixed k, the nontrivial stable

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)

$$\vec{F}(t, \vec{y}, \vec{y}') = \begin{pmatrix} \frac{\partial u}{\partial t} - D_u \nabla^2 u \\ \frac{\partial v}{\partial t} - D_v \nabla^2 v \end{pmatrix} \vec{y} = \begin{pmatrix} u \\ v \end{pmatrix}$$

$$\vec{F}(\cdot) = \vec{G}(\cdot)$$

$$\vec{G}(\vec{y}) = \begin{pmatrix} uv^2 - F(1-u) \\ -uv^2 + (F+k)v \end{pmatrix}$$

$$\vec{F}(t, \vec{y}, \vec{y}') = \vec{G}(\vec{y}) \quad J = \frac{\partial \vec{F}}{\partial \vec{y}} + \frac{\partial \vec{F}}{\partial \vec{y}'}$$

$$\nabla^2 u = \frac{1}{b^2} (4u_{jj} + 2u_{j-1j} + u_{j+1j})$$

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  $D_u = 2 \times 10^{-3}$  and  $D_v = 10^{-2}$ . 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 trivial steady-state solution  $U = 1, V = 0$  exists and is linearly stable for all positive F and k. In the region bounded above by the solid line and below by the dotted line, the system has two stable steady states. For fixed k, the nontrivial stable

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)

$$\vec{F}(t, \vec{y}, \vec{y}') = \vec{G}(\vec{y}) \quad J = \frac{\partial \vec{F}}{\partial \vec{y}} + \frac{\partial \vec{F}}{\partial \vec{y}'}$$

$$\nabla^2 u = \frac{1}{b^2} (4u_{jj} + 2u_{j-1j} + u_{j+1j})$$

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  $D_u = 2 \times 10^{-3}$  and  $D_v = 10^{-2}$ . 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 trivial steady-state solution  $U = 1, V = 0$  exists and is linearly stable for all positive F and k. In the region bounded above by the solid line and below by the dotted line, the system has two stable steady states. For fixed k, the nontrivial stable

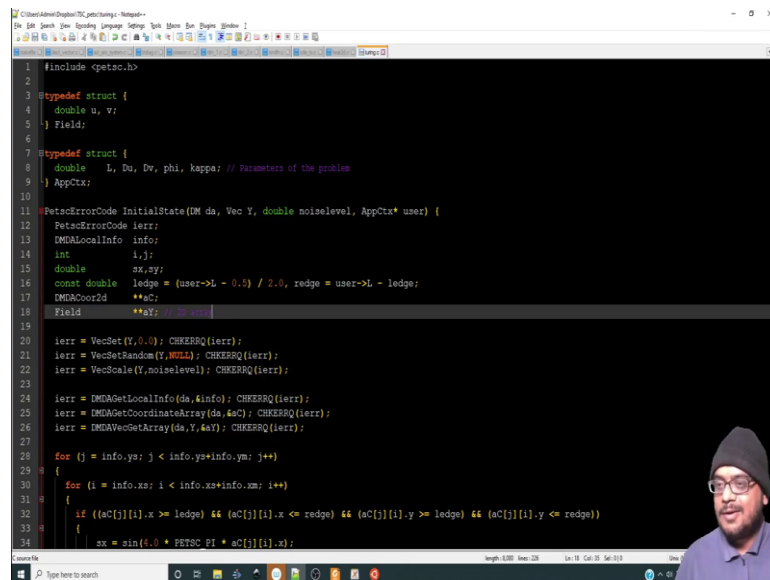
Sorry, we are going to create the Jacobian not for the non-linear part, but for the linear part. So, in this case it will be del F del y plus del F del y prime. So, you are going to

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)



```
1 #include <petsc.h>
2
3 #typedef struct {
4     double u, v;
5 } Field;
6
7 #typedef struct {
8     double L, Dg, Dv, phi, kappa; // Parameters of the problem
9 } AppCtx;
10
11 PetscErrorCode InitialState(DM da, Vec Y, double noislevel, AppCtx* user) {
12     PetscErrorCode ierr;
13     DMDataLocalInfo info;
14     int i, j;
15     double x, y;
16     const double ledge = (user->L - 0.5) / 2.0, redge = user->L - ledge;
17     DMDataCoord2d **aC;
18     Field **aY; // to store
19
20     ierr = VecSet(Y, 0.0);CHKERRQ(ierr);
21     ierr = VecSetRandom(Y, 0.01);CHKERRQ(ierr);
22     ierr = VecScale(Y, noislevel);CHKERRQ(ierr);
23
24     ierr = DMDataGetLocalInfo(da, &info);CHKERRQ(ierr);
25     ierr = DMDataGetCoordinateArray(da, &aC);CHKERRQ(ierr);
26     ierr = DMDataVecGetArray(da, Y, &aY);CHKERRQ(ierr);
27
28     for (j = info.ys; j < info.ys+info.ym; j++)
29     {
30         for (i = info.xs; i < info.xs+info.xm; i++)
31         {
32             if ((aC[j][i].x >= ledge) && (aC[j][i].x <= redge) && (aC[j][i].y >= ledge) && (aC[j][i].y <= redge))
33             {
34                 aY[j][i].x = sin(4.0 * PETSC_PI * aC[j][i].x);
```

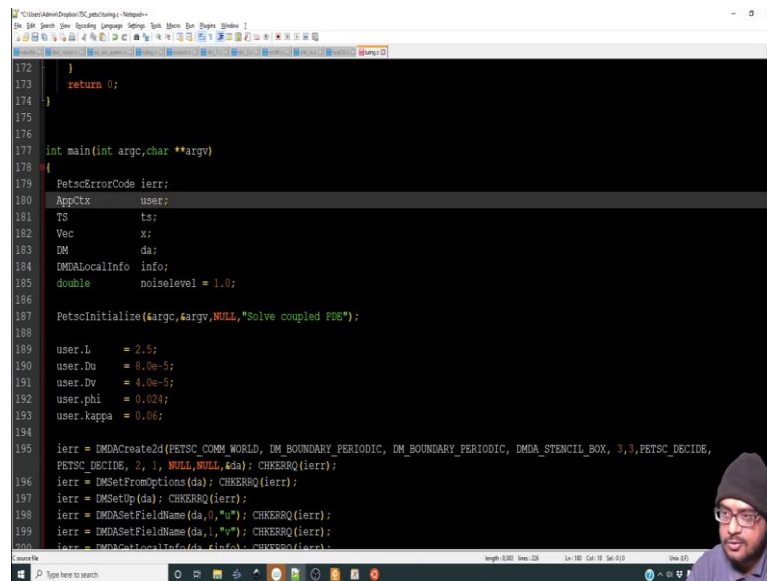
So, with this background let us go to the program I have written on the program already. So, we will do two things we will first create the Field structure which contains simply the u and v. We will create a structure which contains the parameters of the problem which is going to which are going to be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



```
172 }
173     return 0;
174 }
175
176
177 int main(int argc, char **argv)
178 {
179     PetscErrorCode ierr;
180     AppCtx      user;
181     TS          ts;
182     Vec        x;
183     DM         da;
184     DMDALocalInfo info;
185     double     noiselevel = 1.0;
186
187     PetscInitialize(&argc, &argv, NULL, "Solve coupled PDE");
188
189     user.L      = 2.5;
190     user.Du     = 8.0e-5;
191     user.Dv     = 4.0e-5;
192     user.phi    = 0.024;
193     user.kappa  = 0.06;
194
195     ierr = DMACreate2d(PETSC_COMM_WORLD, DM_BOUNDARY_PERIODIC, DM_BOUNDARY_PERIODIC, DMDA_STENCIL_BOX, 3, 3, PETSC_DECIDE,
196     PETSC_DECIDE, 2, 1, NULL, NULL, &da); CHKERRQ(ierr);
197     ierr = DMSetFromOptions(da); CHKERRQ(ierr);
198     ierr = DMSetUp(da); CHKERRQ(ierr);
199     ierr = DMDASetFieldName(da, 0, "u"); CHKERRQ(ierr);
200     ierr = DMDASetFieldName(da, 1, "v"); CHKERRQ(ierr);
201     ierr = DMDALocalInfoGetInfo(&info); CHKERRQ(ierr);
202 }
```

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.

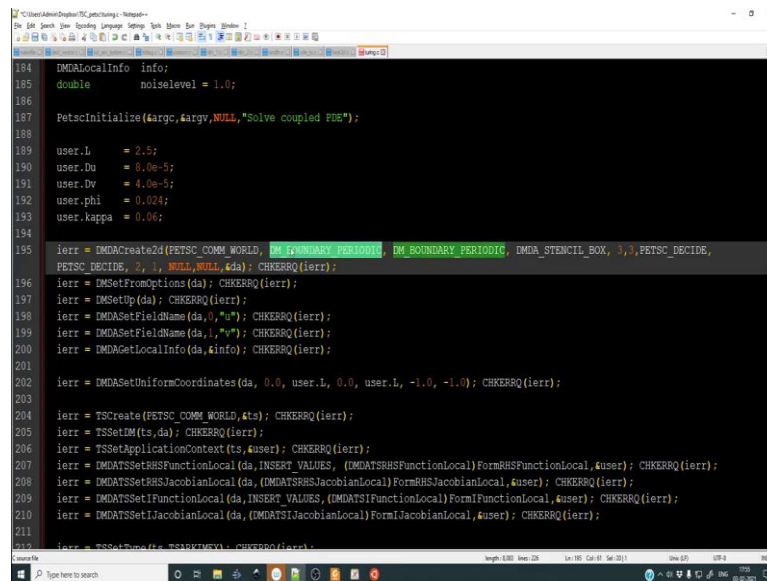
And then you check the error code with this function CHKERRQ, but you do not need to do that as you have seen in our previous examples as well. Then you declare the AppCtx and assign it to a variable called as user then you declare the time stepping object, you declare the vector x, you declare the DM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

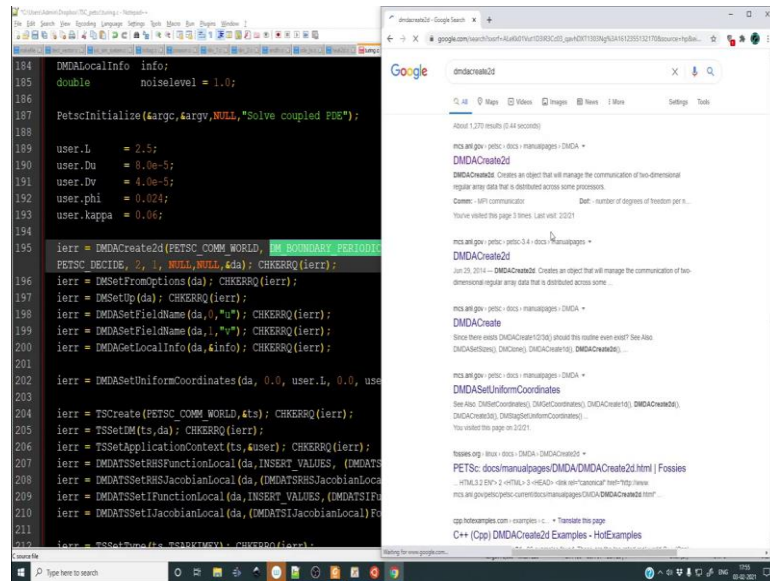
(Refer Slide Time: 20:53)



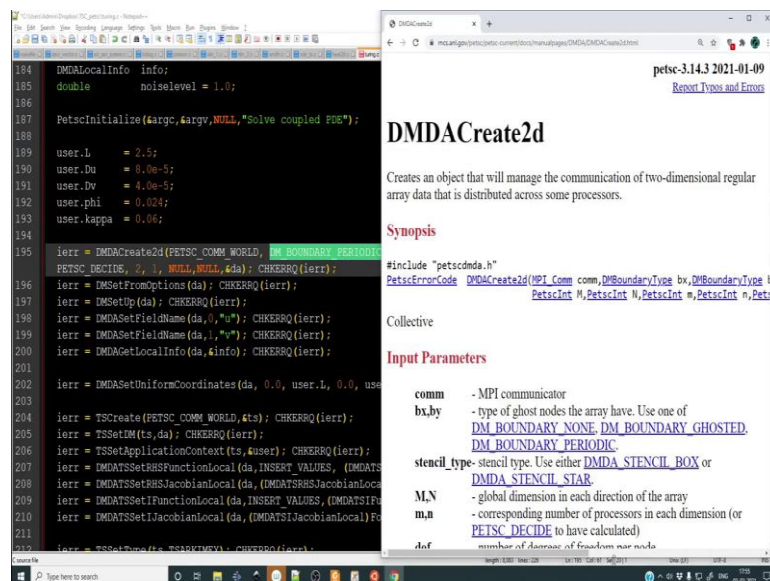
```
184 DMDALocalInfo info;
185 double noiselevel = 1.0;
186
187 PetscInitialize(&argc,&argv,NULL,"Solve coupled PDE");
188
189 user.L = 2.5;
190 user.Du = 8.0e-5;
191 user.Dv = 4.0e-5;
192 user.phi = 0.024;
193 user.kappa = 0.06;
194
195 ierr = DMDACreate2d(PETSC_COMM_WORLD, DM_BOUNDARY_PERIODIC, DM_BOUNDARY_PERIODIC, DMDA_STENCIL_BOX, 3,3,PETSC_DECIDE,
196 PETSC_DECIDE, 2, 1, NULL,NULL,&da); CHKERRQ(ierr);
197 ierr = DMSetFromOptions(da); CHKERRQ(ierr);
198 ierr = DMSetUp(da); CHKERRQ(ierr);
199 ierr = DMDASetFieldName(da,0,"u"); CHKERRQ(ierr);
200 ierr = DMDASetFieldName(da,1,"v"); CHKERRQ(ierr);
201 ierr = DMDAGetLocalInfo(da,&info); CHKERRQ(ierr);
202
203 ierr = DMDASetUniformCoordinates(da, 0.0, user.L, 0.0, user.L, -1.0, -1.0); CHKERRQ(ierr);
204
205 ierr = TSCreate(PETSC_COMM_WORLD,&ts); CHKERRQ(ierr);
206 ierr = TSSetDM(ts,da); CHKERRQ(ierr);
207 ierr = TSSetApplicationContext(ts,&user); CHKERRQ(ierr);
208 ierr = DMDATSRHSFunctionLocal(da,INSERT_VALUES, (DMDATSRHSFunctionLocal)FormRHSFunctionLocal,&user); CHKERRQ(ierr);
209 ierr = DMDATSSetIJacobianLocal(da,INSERT_VALUES, (DMDATSIJacobianLocal)FormIJacobianLocal,&user); CHKERRQ(ierr);
210 ierr = DMDATSSetIJKacobianLocal(da, (DMDATSIJacobianLocal)FormIJKacobianLocal,&user); CHKERRQ(ierr);
211
212 ierr = TSSolve(ts,TSARKIMEX1); CHKERRQ(ierr);
```

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)

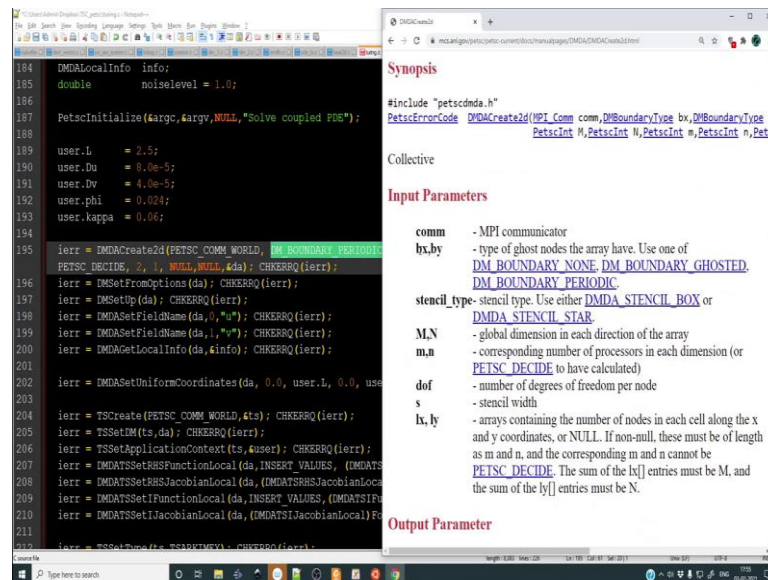


(Refer Slide Time: 21:42)





(Refer Slide Time: 21:44)



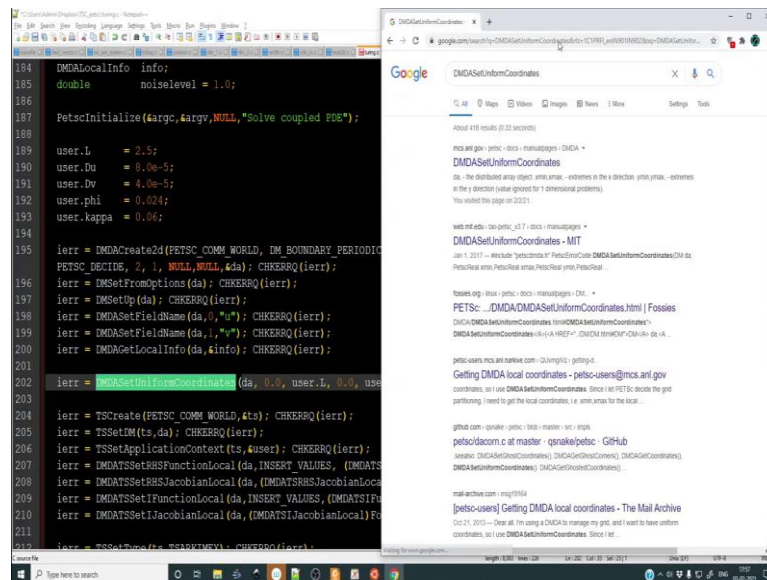
So, just for a reference just for reference, so, we have `bx, by` ok, let us zoom this out a bit. So, a stencil type which is stencil box then you have `m` comma `n`. So, in this case the default grid is 3 comma 3 and you have to decide let PETSC decide the number of the load balancing between the processes.

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`.

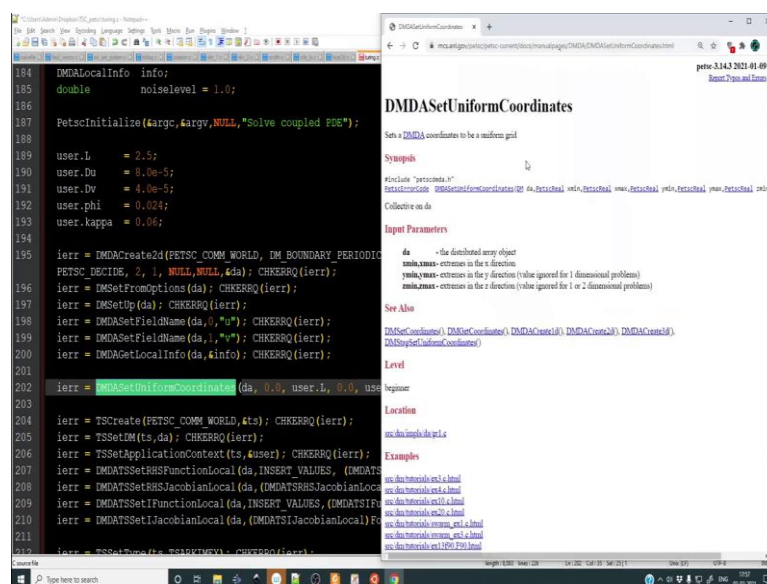
Now, you have to do `DMDA set field name`. So, because there is 2 degrees of freedom you are going to called the zero field as `u` and you are going to call the `v` field the second field as `v`. So, these two lines are used to define the alias for the 2 degrees of freedom over the entire grid.

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)

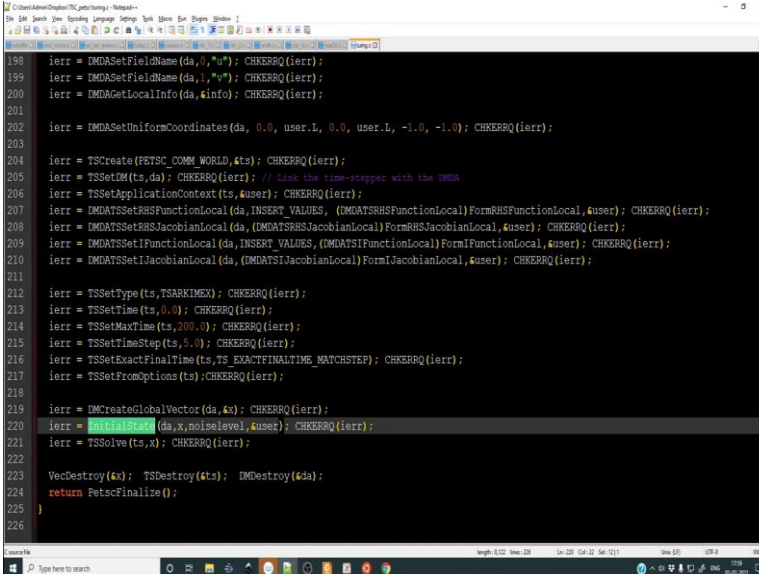


Now, here is something which we have not used so far, which DMDA I said in form coordinates. It is a very simple function which you can imagine what it means. It takes the default domain and it scales it between 0 and L. In this case L is 2.5. So, it is going to scale it from 0 to L.

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)



```
198 ierr = DMDASetFieldName(da,0,"u"); CHKERRQ(ierr);
199 ierr = DMDASetFieldName(da,1,"v"); CHKERRQ(ierr);
200 ierr = DMDAGetLocalInfo(da,&info); CHKERRQ(ierr);
201
202 ierr = DMDASetUniformCoordinates(da, 0.0, user.L, 0.0, user.L, -1.0, -1.0); CHKERRQ(ierr);
203
204 ierr = TSCreate(PETSC_COMM_WORLD,&ts); CHKERRQ(ierr);
205 ierr = TSSetDM(ts,da); CHKERRQ(ierr); // link the time-stepper with the DMDA
206 ierr = TSSetApplicationContext(ts,&user); CHKERRQ(ierr);
207 ierr = DMDATSSetRHSFunctionLocal(da,INSERT_VALUES, (DMDATSSetRHSFunctionLocal)FormRHSFunctionLocal,&user); CHKERRQ(ierr);
208 ierr = DMDATSSetRHSJacobianLocal(da, (DMDATSSetRHSJacobianLocal)FormRHSJacobianLocal,&user); CHKERRQ(ierr);
209 ierr = DMDATSSetIFunctionLocal(da,INSERT_VALUES, (DMDATSSetIFunctionLocal)FormIFunctionLocal,&user); CHKERRQ(ierr);
210 ierr = DMDATSSetIJacobianLocal(da, (DMDATSSetIJacobianLocal)FormIJacobianLocal,&user); CHKERRQ(ierr);
211
212 ierr = TSSetType(ts,TSARKIMEX); CHKERRQ(ierr);
213 ierr = TSSetTime(ts,0.0); CHKERRQ(ierr);
214 ierr = TSSetMaxTime(ts,200.0); CHKERRQ(ierr);
215 ierr = TSSetTimeStep(ts,5.0); CHKERRQ(ierr);
216 ierr = TSSetExactFinalTime(ts,TS_EXACTFINALTIME_MATCHSTEP); CHKERRQ(ierr);
217 ierr = TSSetFromOptions(ts);CHKERRQ(ierr);
218
219 ierr = DMCreateGlobalVector(da,&x); CHKERRQ(ierr);
220 ierr = TSSetStart(da,x,noiseLevel,&user); CHKERRQ(ierr);
221 ierr = TSSolve(ts,x); CHKERRQ(ierr);
222
223 VecDestroy(&x); TSDestroy(&ts); DMDestroy(&da);
224 return PetscFinalize();
225 }
226
```

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)

The slide displays a list of 25 references related to nonlinear optics and materials research. A search window on the right shows results for 'VecSetRandom', including a link to the MIT website and a PDF of the 'VecSetRandom' manual.

**Complex Patterns in a Simple**

6. S. R. Marder, C. B. Gorman, B. G. Tiemann, L.-T. Cheng, *Proc. SPIE* 1775, 19 (1993); C. B. Gorman and S. R. Marder, in preparation.

7. W. Drieth 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. Cherni-Bechikha, J. P. Deckercq, G. Germain, M. V. Meersche, *Cyst. Struct. Comm.* 6, 421 (1977).

11. L. G. S. Brooker et al., *J. Am. Chem. Soc.* 73, 5332 (1951).

12. R. Radejgla 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. Kroil, 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 Society, Pittsburgh), 29 January 1993.

20. Y. Marcus, *J. Soft*

21. B. M. Pierce, *Proc*

22. C. W. Dirk, L.-T. Ch

23. Chem. 43, 27 (198

24. Enhancement of discussed previous Y. Yong, O. Zam (1988).

25. Increased  $\gamma$  in c ganics relative to been observed ex O. Havelka, M. W Cheng, *Proc. SPIE et al., J. Phys. C Cheng et al., ibid*

26. The research was ter for Space Mi Propulsion Labora Technology, and Strategic Defense ive Science and agreement with Space Administr Beckman Institut Scientific Research also acknowledge tor's office for a thanks the Nations for a resident rese 29 January 1993.

(Refer Slide Time: 27:57)

The slide displays the same list of 25 references as the previous slide. A detailed documentation window on the right provides information about the 'VecSetRandom' function, including its syntax, input parameters, output parameters, and examples of usage.

**Complex Patterns in a Simple**

6. S. R. Marder, C. B. Gorman, B. G. Tiemann, L.-T. Cheng, *Proc. SPIE* 1775, 19 (1993); C. B. Gorman and S. R. Marder, in preparation.

7. W. Drieth 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. Cherni-Bechikha, J. P. Deckercq, G. Germain, M. V. Meersche, *Cyst. Struct. Comm.* 6, 421 (1977).

11. L. G. S. Brooker et al., *J. Am. Chem. Soc.* 73, 5332 (1951).

12. R. Radejgla 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. Kroil, 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 Society, Pittsburgh), 29 January 1993.

20. Y. Marcus, *J. Soft*

21. B. M. Pierce, *Proc*

22. C. W. Dirk, L.-T. Ch

23. Chem. 43, 27 (198

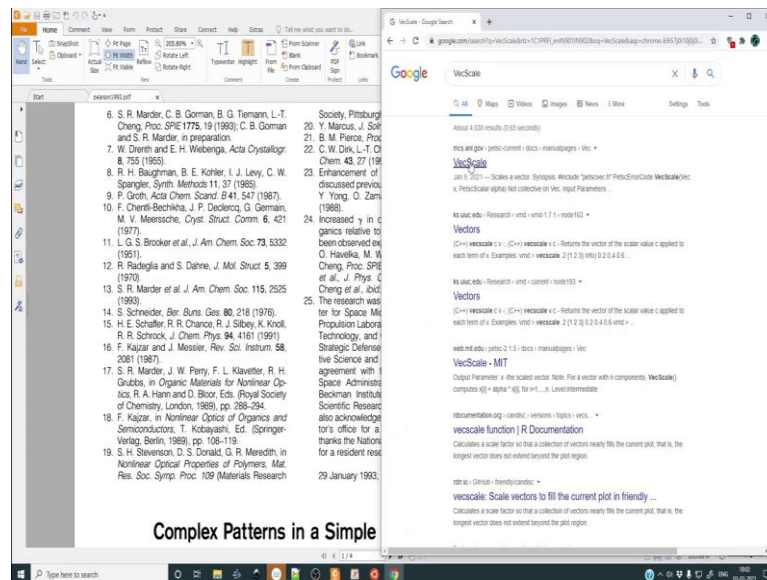
24. Enhancement of discussed previous Y. Yong, O. Zam (1988).

25. Increased  $\gamma$  in c ganics relative to been observed ex O. Havelka, M. W Cheng, *Proc. SPIE et al., J. Phys. C Cheng et al., ibid*

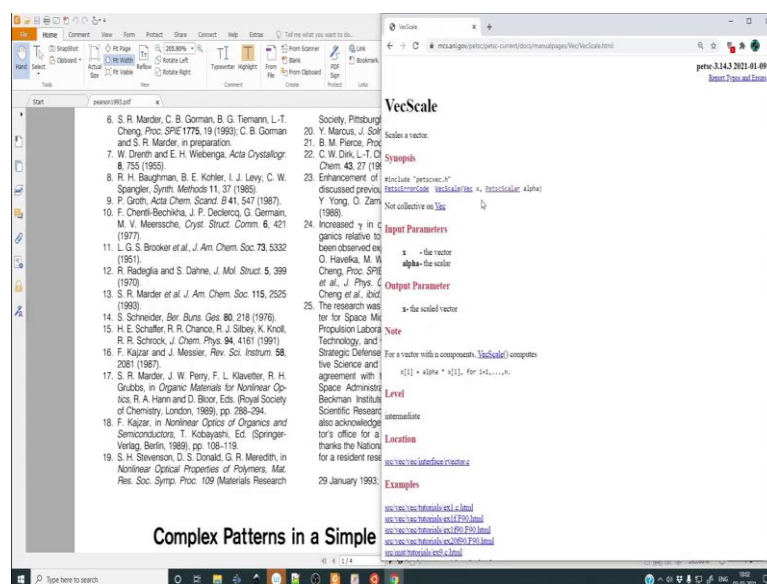
26. The research was ter for Space Mi Propulsion Labora Technology, and Strategic Defense ive Science and agreement with Space Administr Beckman Institut Scientific Research also acknowledge tor's office for a thanks the Nations for a resident rese 29 January 1993.

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)

```

13 DMDALocalInfo info;
14 int i, j;
15 double sx, sy;
16 const double ledge = (user->L - 0.5) / 2.0, redge = user->L - ledge; // declare some analytical function on top the
// random variables between x = 1.0 to x = 1.5
17 DMDACoord2d **aC;
18 Field **aY; // 2D array
19
20 ierr = VecSet(Y, 0.0); CHKERRQ(ierr);
21 ierr = VecSetRandom(Y, NULL); CHKERRQ(ierr);
22 ierr = VecScale(Y, noiselevel); CHKERRQ(ierr);
23
24 ierr = DMDAGetLocalInfo(da, &info); CHKERRQ(ierr);
25 ierr = DMDAGetCoordinateArray(da, &aC); CHKERRQ(ierr); // ac[j][i].x or ac[j][i].y
26 ierr = DMDAVecGetArray(da, Y, &aY); CHKERRQ(ierr);
27
28 for (j = info.ys; j < info.ys+info.yb; j++)
29 {
30     for (i = info.xs; i < info.xs+info.xb; i++)
31     {
32         if ((ac[j][i].x >= ledge) && (ac[j][i].x <= redge) && (ac[j][i].y >= ledge) && (ac[j][i].y <= redge))
33         {
34             sx = sin(4.0 * PETSFC_PI * ac[j][i].x);
35             sy = sin(4.0 * PETSFC_PI * ac[j][i].y);
36             aY[j][i].v += 0.5 * sx * sx * sy * sy;
37         }
38         aY[j][i].u += 1.0 - 2.0 * aY[j][i].v;
39     }
40 }
41 ierr = DMDAVecRestoreArray(da, Y, &aY); CHKERRQ(ierr);

```

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)

```

39 }
40 }
41 ierr = IMDAvecRestoreArray(da,Y,aaY); CHKERRQ(ierr);
42 ierr = IMDARestoreCoordinateArray(da,aaC); CHKERRQ(ierr);
43 return 0;
44 }
45
46 PetscErrorCode FormBHSFunctionLocal(IMDALocalInfo *info, double t, Field **aY, Field **aG, AppCtx *user)
47 {
48     int i, j;
49     double uv2;
50
51     for (j = info->ys; j < info->ys + info->ym; j++)
52     {
53         for (i = info->xs; i < info->xs + info->xm; i++)
54         {
55             uv2 = aY[j][i].u * aY[j][i].v * aY[j][i].v;
56             aG[j][i].u = - uv2 + user->phi * (1.0 - aY[j][i].u);
57             aG[j][i].v = + uv2 - (user->phi + user->kappa) * aY[j][i].v;
58         }
59     }
60     return 0;
61 }
62
63 PetscErrorCode FormBHSJacobianLocal(IMDALocalInfo *info, double t, Field **aY, Mat J, Mat P, AppCtx *user)
64 {
65     PetscErrorCode ierr;
66     int i, j;

```

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  $u^2 - uv^2 + \phi(1 - u)$ . 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  $uv^2 + \phi(1 - u) - uv^2 + \phi(1 - u) + \kappa v$ . So, this is obviously, going to be  $a_y[j][i].v$  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,  $\frac{\partial}{\partial u}$ ; this is going to be what?  $-uv^2$  comma  $0$ . So, let us see where I have written it. So, you are looping over the entire grid.



(Refer Slide Time: 33:15)

The image shows a code editor window with C++ code for a Jacobian matrix calculation. The code is as follows:

```

63
64
65 PetscErrorCode FormBHSJacobianLocal(MDALocalInfo *info, double
66 #if
67 PetscErrorCode ierr;
68 int i, j;
69 double v[2], uv, v2;
70 MatStencil col[3], row;
71
72 for (j = info->ys; j < info->ys+info->ym; j++)
73 {
74 row.j = j; col[0].j = j; col[1].j = j;
75 for (i = info->xs; i < info->xs+info->xm; i++)
76 {
77 row.i = i; col[0].i = i; col[1].i = i;
78 uv = av[j][i].u * av[j][i].v;
79 v2 = av[j][i].v * av[j][i].v;
80
81 row.c = 0; col[0].c = 0; col[1].c = 1;
82 v[0] = -v2 - user->xphi;
83 v[1] = -2.0 * uv;
84 ierr = MatSetValuesStencil(P, 1, &row, 2, col, v, INSERT_VALUES);
85
86 row.c = 1; col[0].c = 0; col[1].c = 1;
87 v[0] = v2;
88 v[1] = 2.0 * uv - (user->xphi + user->kappa);
89 ierr = MatSetValuesStencil(P, 1, &row, 2, col, v, INSERT_VALUES);
90 }
91 }

```

Handwritten notes on the right side of the code editor include:

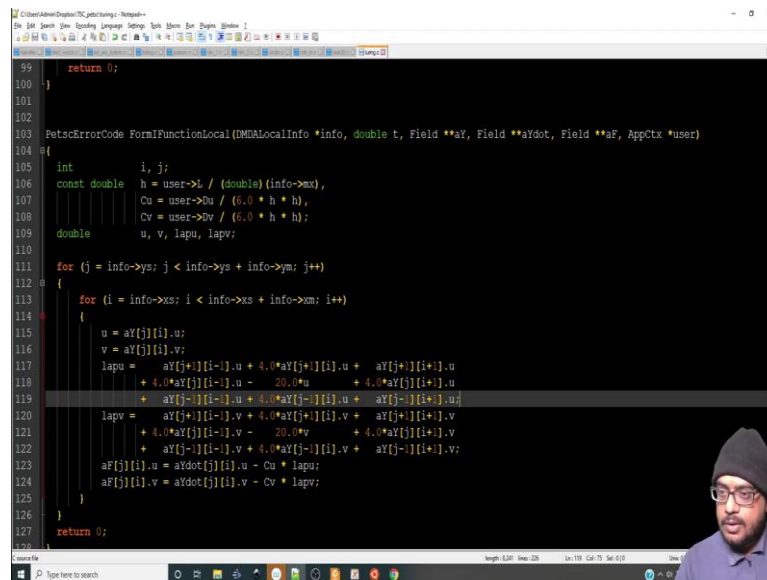
- Forward Euler FD scheme
- $$+ u_{i+1,j} - 2u_{i,j} + u_{i-1,j}$$
- $$+ \phi(t, u_{i,j})$$
- $$\frac{\partial u}{\partial t}$$
- $$\frac{\partial u}{\partial t} + \Delta_x \nabla_x^2 u$$
- $$\vec{F}(t, \vec{y}, \vec{y}') = \begin{pmatrix} \frac{\partial u}{\partial t} + \Delta_x \nabla_x^2 u \\ \frac{\partial \phi}{\partial t} + \Delta_x \nabla_x^2 \phi - 2uv \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix}$$
- $$\vec{F}(\cdot) = \vec{G}(\cdot)$$
- $$\vec{G}(\vec{y}, \vec{y}') = \begin{pmatrix} -u^2 - \phi \\ -2uv \end{pmatrix}$$
- $$\vec{F}(t, \vec{y}, \vec{y}') = \vec{G}(\vec{y}, \vec{y}')$$
- submit
- omolucit
- $$\frac{\partial \vec{F}}{\partial \vec{y}} + \frac{\partial \vec{F}}{\partial \vec{y}'}$$

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  $2u$   $v$ . So, when you take a derivative of this with respect to  $v$ , you have minus  $2uv$  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, 1, row, 2, col, v, INSERT_VALUES)`. 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)



```
99     return 0;
100 }
101
102
103 PetscErrorCode FormFunctionLocal(DMDALocalInfo *info, double t, Field **aY, Field **aYdot, Field **aF, AppCtx *user)
104 {
105     int i, j;
106     const double h = user->h / (double)(info->mx);
107     Cu = user->Cu / (6.0 * h * h);
108     Cv = user->Cv / (6.0 * h * h);
109     double u, v, lapu, lapv;
110
111     for (j = info->ys; j < info->ys + info->ym; j++)
112     {
113         for (i = info->xs; i < info->xs + info->xm; i++)
114         {
115             u = aY[j][i].u;
116             v = aY[j][i].v;
117             lapu = aY[j+1][i-1].u + 4.0*aY[j+1][i].u + aY[j+1][i+1].u
118                 + 4.0*aY[j][i-1].u - 20.0*u + 4.0*aY[j][i+1].u
119                 + aY[j-1][i-1].u + 4.0*aY[j-1][i].u + aY[j-1][i+1].u;
120             lapv = aY[j+1][i-1].v + 4.0*aY[j+1][i].v + aY[j+1][i+1].v
121                 + 4.0*aY[j][i-1].v - 20.0*v + 4.0*aY[j][i+1].v
122                 + aY[j-1][i-1].v + 4.0*aY[j-1][i].v + aY[j-1][i+1].v;
123             aF[j][i].u = aYdot[j][i].u - Cu * lapu;
124             aF[j][i].v = aYdot[j][i].v - Cv * lapv;
125         }
126     }
127     return 0;
128 }
```

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)

The code editor shows the following C++ code:

```

57   aG[j][i].u = - uv2 + user->phi * (1.0 - aF[j][i])
58   aG[j][i].v = + uv2 - (user->phi + user->kappa) *
59   }
60   }
61   return 0;
62   }
63   }
64   }
65   PetscErrorCode FormBHSJacobianLocal(DMDALocalInfo *info, d
66   {
67   PetscErrorCode ierr;
68   int i, j;
69   double v[2], uv, v2;
70   MatStencil col[2], row;
71   for (j = info->ys; j < info->ys+info->ym; j++)
72   {
73   row.j = j; col[0].j = j; col[1].j = j;
74   for (i = info->xs; i < info->xs+info->xm; i++)
75   {
76   row.i = i; col[0].i = i; col[1].i = i;
77   uv = aF[j][i].u * aF[j][i].v;
78   v2 = aF[j][i].v * aF[j][i].v;
79   }
80   row.c = 0; col[0].c = 0; col[1].c = 1;
81   v[0] = - v2 - user->phi;
82   v[1] = - 2.0 * uv;
83   ierr = MatSetValuesStencil(P, 1, &row, 2, col, v, IRR
84   }
85   }
86   }

```

The presentation slide contains the following handwritten notes:

$$\vec{F}(\vec{y}) = \vec{G}(\vec{y})$$

$$\vec{G}(\vec{y}) = \begin{pmatrix} -uv^2 + \phi(1-u) \\ uv^2 - (\phi + \kappa)v \end{pmatrix}$$

$$\vec{F}(t, y, \vec{y}) = \vec{G}(\vec{y}) \frac{\partial \vec{F}}{\partial y} + \frac{\partial \vec{F}}{\partial y}$$

$$\nabla^2 u = \frac{4u_{jj} - 2u_{jj} + 4u_{j-1}}{h^2} + u_{i+1,j} - 2u_{i,j} + u_{i-1,j}$$

A diagram shows a 2D grid with a central point and its neighbors, illustrating the stencil used in the code.

So, this will be  $\frac{\partial u}{\partial t}$  minus the Laplacian of  $u$ . This will be  $\frac{\partial v}{\partial t}$  minus Laplacian of  $v$  and this will be  $-uv^2$  that is this plus  $\phi(1-u)$  and this will be  $uv^2$  minus  $\phi$  plus  $\kappa v$  yeah ok. So, now we have to form the implicit Jacobian.

(Refer Slide Time: 37:29)

The code editor shows the following C++ code:

```

132 PetscErrorCode FormJacobianLocal(DMDALocalInfo *info,
133 double t, Field **aF, Field **aDot, double shift,
134 Mat J, Mat P, AppCtx *user) {
135 PetscErrorCode ierr;
136 int i, j, s, c;
137 const double h = user->L / (double)(info->mx);
138 Cu = user->Du / (6.0 * h * h);
139 Cv = user->Dv / (6.0 * h * h);
140 double val[2], CC;
141 MatStencil col[2], row;
142 for (j = info->ys; j < info->ys + info->ym; j++) {
143 row.j = j;
144 for (i = info->xs; i < info->xs + info->xm; i++) {
145 row.i = i;
146 for (c = 0; c < 2; c++) { // u,v equations are c=0,1
147 row.c = c;
148 CC = (c == 0) ? Cu : Cv;
149 for (s = 0; s < 2; s++)
150 col[s].c = c;
151 col[0].i = i; col[0].j = j;
152 val[0] = shift + 25.0 * CC;
153 col[1].i = i-1; col[1].j = j; val[1] = - 4.0 * CC;
154 col[1].i = i+1; col[1].j = j; val[2] = - 4.0 * CC;
155 col[1].i = i; col[1].j = j-1; val[3] = - 4.0 * CC;
156 col[1].i = i; col[1].j = j+1; val[4] = - 4.0 * CC;
157 col[1].i = i-1; col[1].j = j-1; val[5] = - CC;
158 col[1].i = i-1; col[1].j = j+1; val[6] = - CC;
159 col[1].i = i+1; col[1].j = j-1; val[7] = - CC;
160 col[1].i = i+1; col[1].j = j+1; val[8] = - CC;
161 }

```

Now, the implicit Jacobian will actually be written in this form. It is going to be  $dF \cdot dY$  plus  $dF \cdot dY$  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)

```
C:\Users\shah\Desktop\TC_projects> Notepad++
File Edit Search View Encoding Language Settings Spell Check Run Plugins Window Help
165 }
166 }
167 ierr = MatAssemblyBegin(P_MAT_FINAL_ASSEMBLY); CHKERRQ(ierr);
168 ierr = MatAssemblyEnd(P_MAT_FINAL_ASSEMBLY); CHKERRQ(ierr);
169 if (J != P) {
170     ierr = MatAssemblyBegin(J_MAT_FINAL_ASSEMBLY); CHKERRQ(ierr);
171     ierr = MatAssemblyEnd(J_MAT_FINAL_ASSEMBLY); CHKERRQ(ierr);
172 }
173 return 0;
174 }
175
176
177 int main(int argc, char **argv)
178 {
179     PetscErrorCode ierr;
180     AppCtx      user;
181     TS          ts;
182     Vec         x;
183     DM          da;
184     DMALocalInfo info;
185     double      noiselevel = 1.0;
186
187     PetscInitialize(&argc, &argv, NULL, "Solve coupled PDE");
188
189     user.L = 2.5;
190     user.Du = 8.0e-5;
191     user.Dv = 4.0e-5;
192     user.phi = 0.024;
193     user.kappa = 0.06;
194
195 }
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

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)

```
C:\Users\shah\Desktop\TC_projects> Notepad++
File Edit Search View Encoding Language Settings Spell Check Run Plugins Window Help
197 ierr = DMSetup(da); CHKERRQ(ierr);
198 ierr = DMDataSetName(da, 0, "u"); CHKERRQ(ierr);
199 ierr = DMDataSetName(da, 1, "v"); CHKERRQ(ierr);
200 ierr = DMGetLocalInfo(da, &info); CHKERRQ(ierr);
201
202 ierr = DMSetUniformCoordinates(da, 0.0, user.L, 0.0, user.L, -1.0, -1.0); CHKERRQ(ierr);
203
204 ierr = TSCreate(PETSC_COMM_WORLD, &ts); CHKERRQ(ierr);
205 ierr = TSSetDM(ts, da); CHKERRQ(ierr); // Link the time-stepper with the DMGA
206 ierr = TSSetApplicationContext(ts, &user); CHKERRQ(ierr);
207 ierr = DMATSRHSFunctionLocal(da, INSERT_VALUES, (DMATSRHSFunctionLocal)FormRHSFunctionLocal, &user); CHKERRQ(ierr);
208 ierr = DMATSRHSJacobianLocal(da, (DMATSRHSJacobianLocal)FormRHSJacobianLocal, &user); CHKERRQ(ierr);
209 ierr = DMATSIFunctionLocal(da, INSERT_VALUES, (DMATSIFunctionLocal)FormFunctionLocal, &user); CHKERRQ(ierr);
210 ierr = DMATSIJacobianLocal(da, (DMATSIJacobianLocal)FormIJacobianLocal, &user); CHKERRQ(ierr);
211
212 ierr = TSSetType(ts, TSARKIMEX); CHKERRQ(ierr);
213 ierr = TSSetTime(ts, 0.0); CHKERRQ(ierr);
214 ierr = TSSetMaxTime(ts, 200.0); CHKERRQ(ierr);
215 ierr = TSSetTimeStep(ts, 5.0); CHKERRQ(ierr);
216 ierr = TSSetExactFinalTime(ts, TS_EXACTFINALTIME_MATCHSTEP); CHKERRQ(ierr);
217 ierr = TSSetFromOptions(ts); CHKERRQ(ierr);
218
219 ierr = DMCreateGlobalVector(da, &x); CHKERRQ(ierr);
220 ierr = InitialState(da, x, noiselevel, &user); CHKERRQ(ierr);
221 ierr = TSSolve(ts, x); CHKERRQ(ierr);
222
223 VecDestroy(&x); TSDestroy(&ts); DMDestroy(&da);
224 return PetscFinalize();
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

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)

```
166
167 ierr = MatAssemblyBegin(P_MAT_FINAL_ASSEMBLY); CHKERRQ(ierr);
168 ierr = MatAssemblyEnd(P_MAT_FINAL_ASSEMBLY); CHKERRQ(ierr);
169 if (J != F) {
170     ierr = MatAssemblyBegin(J_MAT_FINAL_ASSEMBLY); CHKERRQ(ierr);
171     ierr = MatAssemblyEnd(J_MAT_FINAL_ASSEMBLY); CHKERRQ(ierr);
172 }
173 return 0;
174 }
175
176
177 int main(int argc, char **argv)
178 {
179     PetscErrorCode ierr;
180     AppCtx      user;
181     TS          ts;
182     Vec         x;
183     DM          da;
184     DMALocalInfo info;
185     double      noiselevel = 0.2;
186
187     PetscInitialize(&argc, &argv, NULL, "Solve coupled PDE");
188
189     user.L     = 2.5;
190     user.Du    = 0.0e-5;
191     user.Dv    = 4.0e-5;
192     user.phi   = 0.024;
193     user.kappa = 0.06;
194
195     ierr = DMCreate3D(PETSC_COMM_WORLD, DM_BOUNDARY_PERIODIC, DM_BOUNDARY_STATIC, DMV3_CROSS3I, DMV3_CROSS3I, DMV3_CROSS3I);
196 }
```

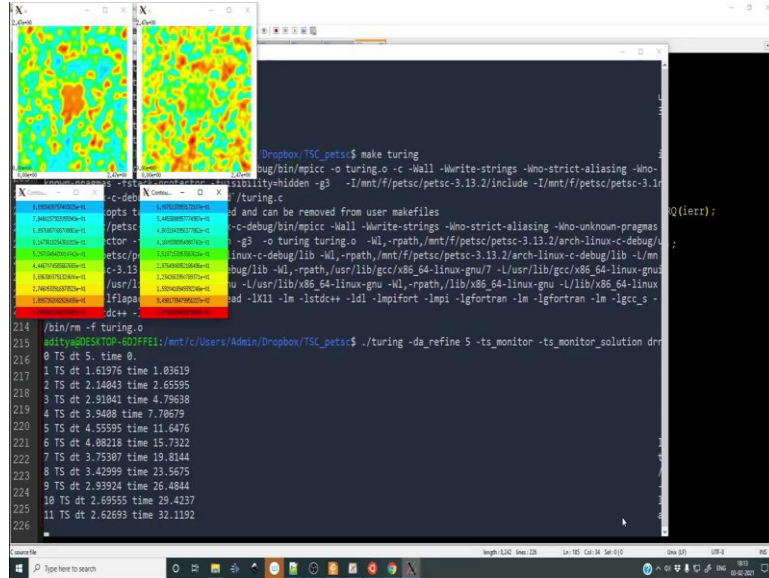
(Refer Slide Time: 38:50)

```
198 20 TS dt 3.55734 time 22.5048
199 21 TS dt 3.91573 time 26.0621
200 22 TS dt 4.34559 time 29.9779
201 23 TS dt 4.88298 time 34.3235
202 24 TS dt 5.58298 time 39.2064
203 25 TS dt 6.55103 time 44.7894
204 26 TS dt 8.02251 time 51.3465
205 27 TS dt 10.4156 time 59.363
206 28 TS dt 10.7118 time 69.9785
207 29 TS dt 9.76012 time 79.3443
208 30 TS dt 9.89015 time 89.0445
209 31 TS dt 11.1705 time 98.9347
210 32 TS dt 13.3698 time 110.105
211 33 TS dt 16.4497 time 123.475
212 34 TS dt 19.8476 time 139.925
213 35 TS dt 20.1139 time 159.772
214 36 TS dt 20.1139 time 179.886
215 37 TS dt 32.4456 time 200.
216
217 #! /usr/bin/env bash
218 cd /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -o turing.o -c -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility=hidden -g3 -I/mnt/f/petsc/petsc-3.13.2/include -I/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/include -I/pwd/turing.c
219 Warning: chkopts target is deprecated and can be removed from user makefiles
220 /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility=hidden -g3 -o turing.o -M, -rpath, /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L, /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -M, -rpath, /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L, /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -M, -rpath, /usr/lib/gcc/x86_64-linux-gnu/7 -L/usr/lib/gcc/x86_64-linux-gnu -M, -rpath, /usr/lib/x86_64-linux-gnu -L/usr/lib/x86_64-linux-gnu -M, -rpath, /lib/x86_64-linux-gnu -L/lib/x86_64-linux-gnu -lpetsc -lflask -lftlas -lphread -lX11 -lm -lstdc++ -ldl -lmpifort -lmpi -lfortran -lm -lgfortran -lm -lgcc_s -ldmath -lstdc++ -ldl
221 /bin/rm -f turing.o
222 #! /usr/bin/env bash
223 cd /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -o turing.o -c -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility=hidden -g3 -I/mnt/f/petsc/petsc-3.13.2/include -I/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/include -I/pwd/turing.c
224 Warning: chkopts target is deprecated and can be removed from user makefiles
225 /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility=hidden -g3 -o turing.o -M, -rpath, /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L, /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -M, -rpath, /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L, /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -M, -rpath, /usr/lib/gcc/x86_64-linux-gnu/7 -L/usr/lib/gcc/x86_64-linux-gnu -M, -rpath, /usr/lib/x86_64-linux-gnu -L/usr/lib/x86_64-linux-gnu -M, -rpath, /lib/x86_64-linux-gnu -L/lib/x86_64-linux-gnu -lpetsc -lflask -lftlas -lphread -lX11 -lm -lstdc++ -ldl -lmpifort -lmpi -lfortran -lm -lgfortran -lm -lgcc_s -ldmath -lstdc++ -ldl
226 /bin/rm -f turing.o
```

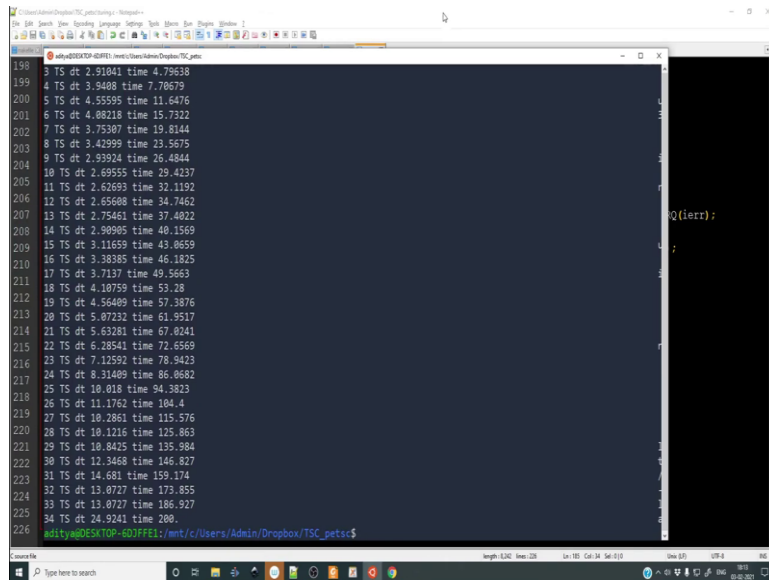
So, let me just make the noiselevel a bit low 0 point maybe 0 2, will increase the time later, but let me make turing. So, I have called the program as turing dot c and I also gone ahead and modified the make file then we do dot slash turing. So, da refine is because we have defined a 3 cross 3 grid, 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)



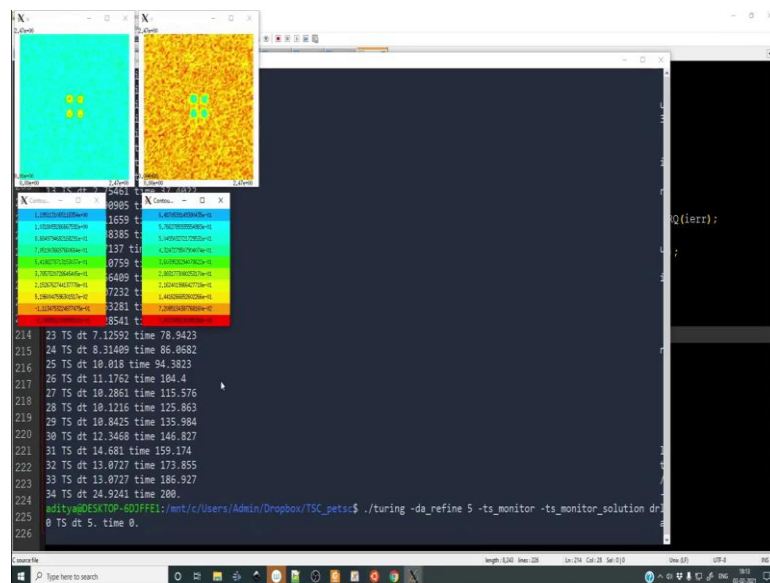
(Refer Slide Time: 39:20)



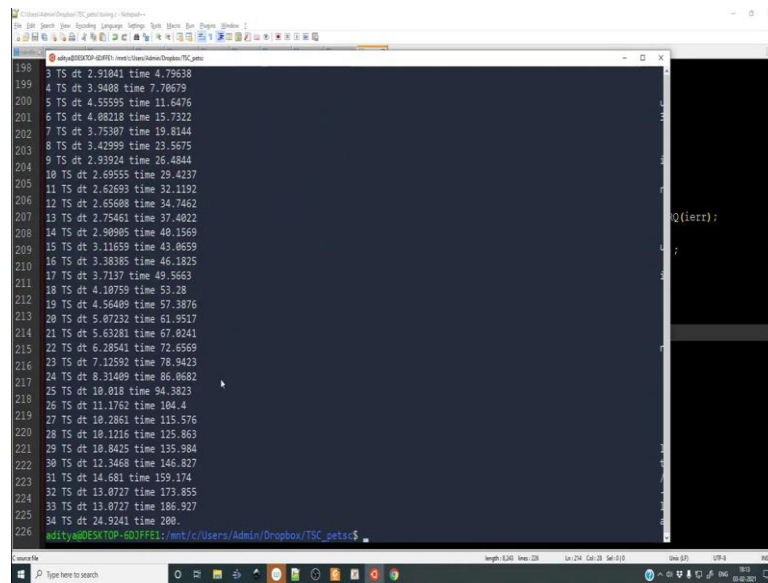
(Refer Slide Time: 39:30)

```
198 ierr = DMDSaveFieldData(da,0,"u"); CHKERRQ(ierr);
199 ierr = DMDSaveFieldData(da,1,"v"); CHKERRQ(ierr);
200 ierr = DMDSaveLocalInfo(da,4,info); CHKERRQ(ierr);
201
202 ierr = DMDSaveUniformCoordinates(da,0.0,user.L,0.0,user.L,-1.0,-1.0); CHKERRQ(ierr);
203
204 ierr = TScreate(PETSC_COMM_WORLD,&ts); CHKERRQ(ierr);
205 ierr = TSsetDM(ts,da); CHKERRQ(ierr); // Link the time-stepper with the DM
206 ierr = TSsetApplicationContext(ts,&user); CHKERRQ(ierr);
207 ierr = DMDSetsRHSFunctionLocal(da,INSERT_VALUES,(DMDSetsRHSFunctionLocal)FormRHSFunctionLocal,&user); CHKERRQ(ierr);
208 ierr = DMDSetsRHSJacobianLocal(da,(DMDSetsRHSJacobianLocal)FormRHSJacobianLocal,&user); CHKERRQ(ierr);
209 ierr = DMDSetsIFunctionLocal(da,INSERT_VALUES,(DMDSetsIFunctionLocal)FormIFunctionLocal,&user); CHKERRQ(ierr);
210 ierr = DMDSetsIJacobianLocal(da,(DMDSetsIJacobianLocal)FormIJacobianLocal,&user); CHKERRQ(ierr);
211
212 ierr = TSsetType(ts,TSARKIMEX); CHKERRQ(ierr);
213 ierr = TSsetTime(ts,0.0); CHKERRQ(ierr);
214 ierr = TSsetMaxTime(ts,20.0); CHKERRQ(ierr);
215 ierr = TSsetTimeStep(ts,5.0); CHKERRQ(ierr);
216 ierr = TSsetExactFinalTime(ts,TS_EXACTFINALTIME_MATCHSTEP); CHKERRQ(ierr);
217 ierr = TSsetFromOptions(ts); CHKERRQ(ierr);
218
219 ierr = DMcreateGlobalVector(da,&x); CHKERRQ(ierr);
220 ierr = InitialState(da,x,&user); CHKERRQ(ierr);
221 ierr = TSSolve(ts,x); CHKERRQ(ierr);
222
223 VecDestroy(&x); TSDestroy(&ts); DMDestroy(&da);
224 return PetscFinalize();
225
226
```

(Refer Slide Time: 39:38)



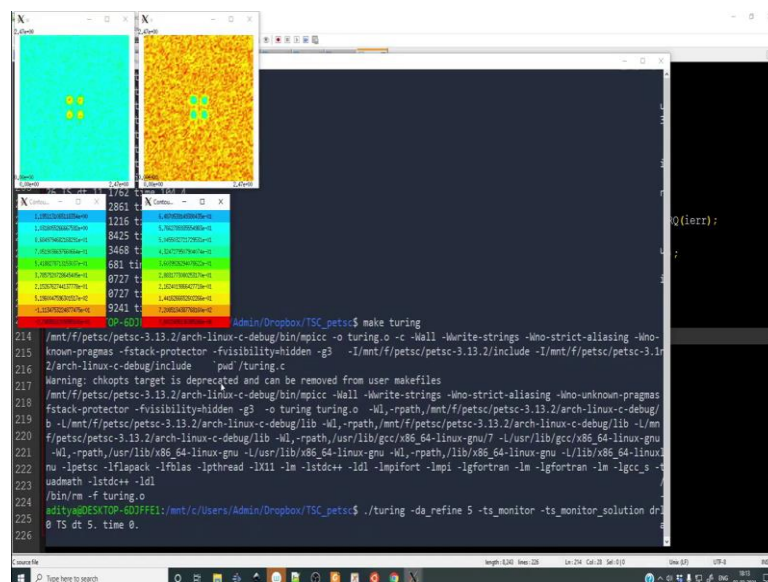
(Refer Slide Time: 39:41)



```
1998 3 TS dt 2.91841 time 4.79638
1999 4 TS dt 3.9488 time 7.78679
2000 5 TS dt 4.55595 time 11.6476
2001 6 TS dt 4.88218 time 15.7922
2002 7 TS dt 3.75307 time 19.8144
2003 8 TS dt 3.42899 time 23.5675
2004 9 TS dt 2.93924 time 26.4844
2005 10 TS dt 2.69555 time 29.4237
2006 11 TS dt 2.62693 time 32.1192
2007 12 TS dt 2.65688 time 34.7462
2008 13 TS dt 2.75461 time 37.4022
2009 14 TS dt 2.98985 time 40.1569
2010 15 TS dt 3.11659 time 43.0659
2011 16 TS dt 3.38385 time 46.1825
2012 17 TS dt 3.7187 time 49.5663
2013 18 TS dt 4.18759 time 53.28
2014 19 TS dt 4.56409 time 57.3876
2015 20 TS dt 5.07232 time 61.9517
2016 21 TS dt 5.63281 time 67.0241
2017 22 TS dt 6.28541 time 72.6569
2018 23 TS dt 7.12592 time 78.9423
2019 24 TS dt 8.31489 time 86.0682
2020 25 TS dt 10.018 time 94.3823
2021 26 TS dt 11.1782 time 104.4
2022 27 TS dt 10.2861 time 115.576
2023 28 TS dt 10.1216 time 125.863
2024 29 TS dt 10.8425 time 135.984
2025 30 TS dt 12.3468 time 146.827
2026 31 TS dt 14.681 time 159.174
2027 32 TS dt 13.8727 time 173.855
2028 33 TS dt 13.8727 time 186.927
2029 34 TS dt 24.9241 time 200.
aditya@DESKTOP-6D0FFFE1:~/mnt/c:/Users/Admin/Dropbox/TSC_petsc$
```

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)

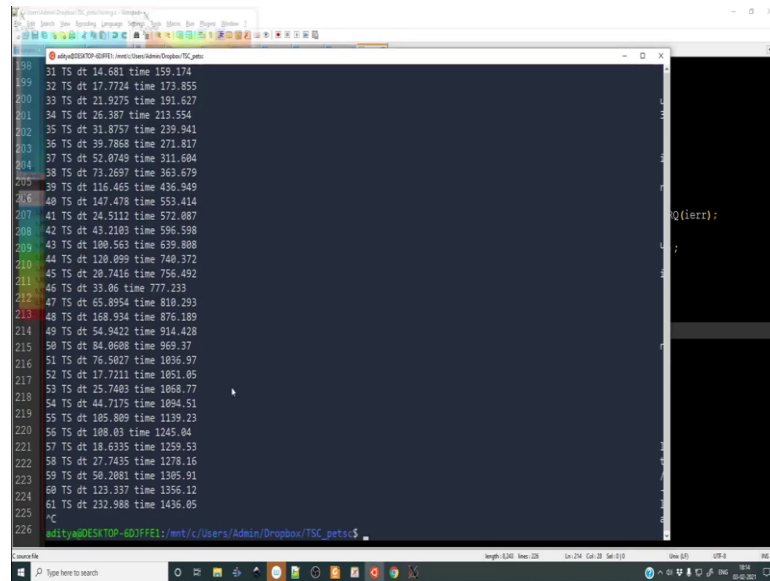


```
214 /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -o turing.o -c -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-
215 known-pragmas -fstack-protector -fvvisibility-hidden -g3 -I/mnt/f/petsc/petsc-3.13.2/include -I/mnt/f/petsc/petsc-3.1
216 2/arch-linux-c-debug/include -I/mnt/f/petsc/petsc-3.13.2/include -I/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/incl
Warning: chxopts target is deprecated and can be removed from user makefiles
218 /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas
219 fstack-protector -fvvisibility-hidden -g3 -o turing turing.o -Wl,-rpath,/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/b
220 b -L/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl,-rpath,/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/m
221 f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl,-rpath,/usr/lib/gcc/x86_64-linux-gnu/7 -L/usr/lib/gcc/x86_64-linux-gnu
222 -Wl,-rpath,/usr/lib/x86_64-linux-gnu -L/usr/lib/x86_64-linux-gnu -Wl,-rpath,/lib/x86_64-linux-gnu -L/lib/x86_64-linux
223 nu -lpetsc -lflapack -lflblas -lpthread -lX11 -lm -lstdc++ -ldl -lmpifort -lmpi -lgfortran -lm -lgcc_s -l
224 uadmath -lstdc++ -ldl
225 /bin/rm -f turing.o
aditya@DESKTOP-6D0FFFE1:~/mnt/c:/Users/Admin/Dropbox/TSC_petsc$ ./turing -da_refine 5 -ts_monitor -ts_monitor_solution dr
226 0 TS dt 5. time 0.
```

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



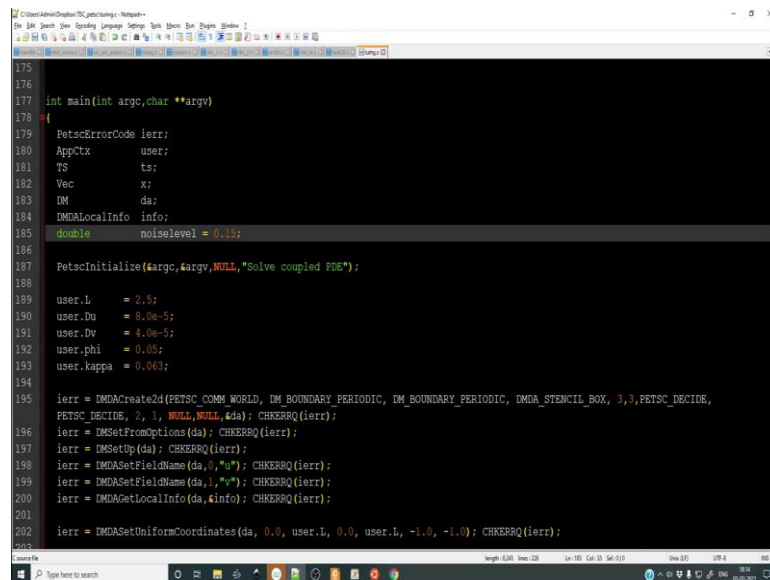
(Refer Slide Time: 40:09)



```
edit@DESKTOP-6D3FF6E1:~/mnt/c:/users/Admin/Dropbox/TSC_petsc$
31 TS dt 14.681 time 159.174
32 TS dt 17.7724 time 173.855
33 TS dt 21.9275 time 191.627
34 TS dt 26.387 time 213.554
35 TS dt 31.8757 time 239.941
36 TS dt 39.7868 time 271.817
37 TS dt 52.8749 time 311.694
38 TS dt 73.2697 time 363.679
39 TS dt 116.465 time 436.949
40 TS dt 147.478 time 553.414
41 TS dt 24.5112 time 572.087
42 TS dt 43.2183 time 596.598
43 TS dt 100.563 time 639.808
44 TS dt 120.899 time 748.372
45 TS dt 20.7416 time 756.492
46 TS dt 33.86 time 777.233
47 TS dt 65.8954 time 818.293
48 TS dt 168.934 time 876.189
49 TS dt 54.9422 time 914.428
50 TS dt 84.8608 time 969.37
51 TS dt 76.5027 time 1036.97
52 TS dt 17.7211 time 1051.05
53 TS dt 25.7483 time 1068.77
54 TS dt 44.7175 time 1094.51
55 TS dt 185.809 time 1139.23
56 TS dt 188.83 time 1245.84
57 TS dt 18.6335 time 1259.53
58 TS dt 27.7435 time 1278.16
59 TS dt 50.2081 time 1385.91
60 TS dt 123.337 time 1356.12
61 TS dt 232.988 time 1436.05
^C
edit@DESKTOP-6D3FF6E1:~/mnt/c:/users/Admin/Dropbox/TSC_petsc$
```

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)



```
175
176
177 int main(int argc, char **argv)
178 {
179     PetscErrorCode ierr;
180     AppCtx user;
181     TS ts;
182     Vec x;
183     DM da;
184     DMALocalInfo info;
185     double noiselevel = 0.15;
186
187     PetscInitialize(&argc, &argv, NULL, "Solve coupled PDE");
188
189     user.L = 2.5;
190     user.Du = 0.0e-5;
191     user.Dv = 4.0e-5;
192     user.phi = 0.05;
193     user.kappa = 0.063;
194
195     ierr = DMACreate2d(PETSC_COMM_WORLD, DM_BOUNDARY_PERIODIC, DM_BOUNDARY_PERIODIC, DMA_STENCIL_BOX, 3, 3, PETSC_DECIDE,
196     PETSC_DECIDE, 2, 1, NULL, NULL, &da); CHKERRQ(ierr);
197     ierr = DMSetFromOptions(da); CHKERRQ(ierr);
198     ierr = DMSetup(da); CHKERRQ(ierr);
199     ierr = DMSetFieldName(da, 0, "u"); CHKERRQ(ierr);
200     ierr = DMSetFieldName(da, 1, "v"); CHKERRQ(ierr);
201     ierr = DMGetLocalInfo(da, &info); CHKERRQ(ierr);
202
203     ierr = DMSetUniformCoordinates(da, 0.0, user.L, 0.0, user.L, -1.0, -1.0); CHKERRQ(ierr);
204
205 }
```

(Refer Slide Time: 40:35)

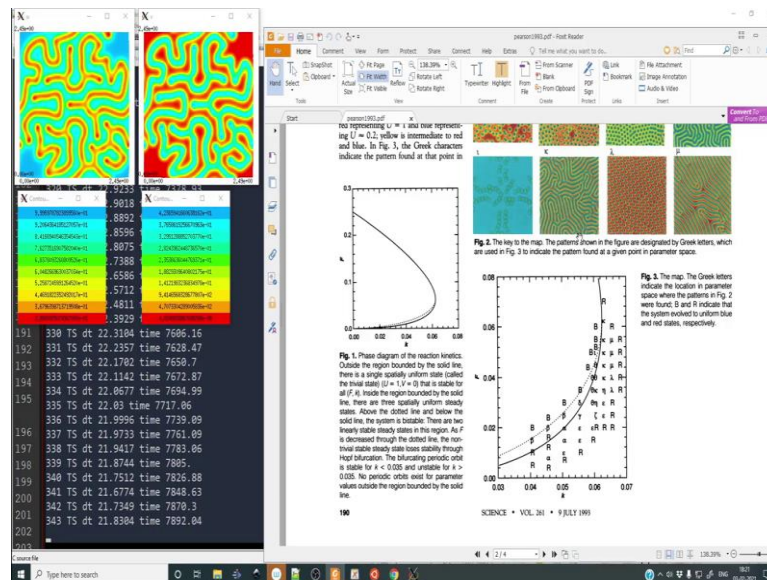
```
C:\Users\Admin\Desktop\TSC_petsc\mingw - Notepad++
File Edit Search View Encoding Language Settings Spell Check Run Debug Window Help
198 ierr = DMDASetFieldName(da,0,"u"); CHKERRQ(ierr);
199 ierr = DMDASetFieldName(da,1,"v"); CHKERRQ(ierr);
200 ierr = DMDASetLocalInfo(da,4,info); CHKERRQ(ierr);
201
202 ierr = DMDASetUniformCoordinates(da,0.0,user.L,0.0,user.L,-1.0,-1.0); CHKERRQ(ierr);
203
204 ierr = TSCreate(PETSC_COMM_WORLD,&ts); CHKERRQ(ierr);
205 ierr = TSSetDM(ts,da); CHKERRQ(ierr); // Link the time-stepper with the DMDA
206 ierr = TSSetApplicationContext(ts,&user); CHKERRQ(ierr);
207 ierr = DMDATSRHSFunctionLocal(da,INSERT_VALUES,(DMDATSRHSFunctionLocal)FormRHSFunctionLocal,&user); CHKERRQ(ierr);
208 ierr = DMDATSRHSJacobianLocal(da,(DMDATSRHSJacobianLocal)FormRHSJacobianLocal,&user); CHKERRQ(ierr);
209 ierr = DMDATSIFunctionLocal(da,INSERT_VALUES,(DMDATSIFunctionLocal)FormFunctionLocal,&user); CHKERRQ(ierr);
210 ierr = DMDATSIJacobianLocal(da,(DMDATSIJacobianLocal)FormJacobianLocal,&user); CHKERRQ(ierr);
211
212 ierr = TSSetType(ts,TSARKIMEX); CHKERRQ(ierr);
213 ierr = TSSetTime(ts,0.0); CHKERRQ(ierr);
214 ierr = TSSetMaxTime(ts,15000.0); CHKERRQ(ierr);
215 ierr = TSSetTimeStep(ts,5.0); CHKERRQ(ierr);
216 ierr = TSSetExactFinalTime(ts,TS_EXACTFINALTIME_MATCHSTEP); CHKERRQ(ierr);
217 ierr = TSSetFromOptions(ts); CHKERRQ(ierr);
218
219 ierr = DMCreateGlobalVector(da,&x); CHKERRQ(ierr);
220 ierr = InitialState(da,x,noiselevel,&user); CHKERRQ(ierr);
221 ierr = TSSolve(ts,x); CHKERRQ(ierr);
222
223 VecDestroy(&x); TSDestroy(&ts); DMDestroy(&da);
224 return PetscFinalize();
225
226
```

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)

```
C:\Users\Admin\Desktop\TSC_petsc\mingw - Notepad++
175 45 TS at 20.7416 time 756.492
176 46 TS at 33.86 time 777.233
177 47 TS at 65.8954 time 819.293
178 48 TS at 168.934 time 876.189
179 49 TS at 54.5422 time 914.428
180 50 TS at 84.8608 time 969.37
181 51 TS at 76.5027 time 1036.97
182 52 TS at 17.7211 time 1051.05
183 53 TS at 25.7403 time 1068.77
184 54 TS at 44.7175 time 1094.51
185 55 TS at 105.809 time 1139.23
186 56 TS at 108.03 time 1245.04
187 57 TS at 18.6335 time 1259.53
188 58 TS at 27.7435 time 1278.16
189 59 TS at 50.2081 time 1385.91
190 60 TS at 123.337 time 1356.12
191 61 TS at 232.988 time 1436.05
192
193 gcc@DESKTOP-8D3FFFE1: /mnt/c:/Users/Admin/Desktop/TSC_petsc$ make turing
194 Warning: chokopts target is deprecated and can be removed from user makefiles
195 /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -o turing.o -c -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvvisibility-hidden -g3 -I/mnt/f/petsc/petsc-3.13.2/include -I/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/include -pwd /turing.c
196 /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvvisibility-hidden -g3 -o turing turing.o -Wl,-rpath,/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl,-rpath,/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl,-rpath,/usr/lib/gcc/x86_64-linux-gnu/7 -L/usr/lib/gcc/x86_64-linux-gnu -Wl,-rpath,/usr/lib/x86_64-linux-gnu -L/usr/lib/x86_64-linux-gnu -Wl,-rpath,/lib/x86_64-linux-gnu -L/lib/x86_64-linux-gnu -lpetsc -lflapack -lflblas -lthreads -lX11 -lm -lstltdc++ -ldl -lmpifort -lmpi -lgfortran -lm -lgfortran -lm -lgcc_s -ludmath -lstltdc++ -ldl
197 /bin/rm -f turing.o
198 gcc@DESKTOP-8D3FFFE1: /mnt/c:/Users/Admin/Desktop/TSC_petsc$ ./turing -da_refine 6 -ts_monitor -ts_monitor_solution dr
199
200
201
202
```

(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).