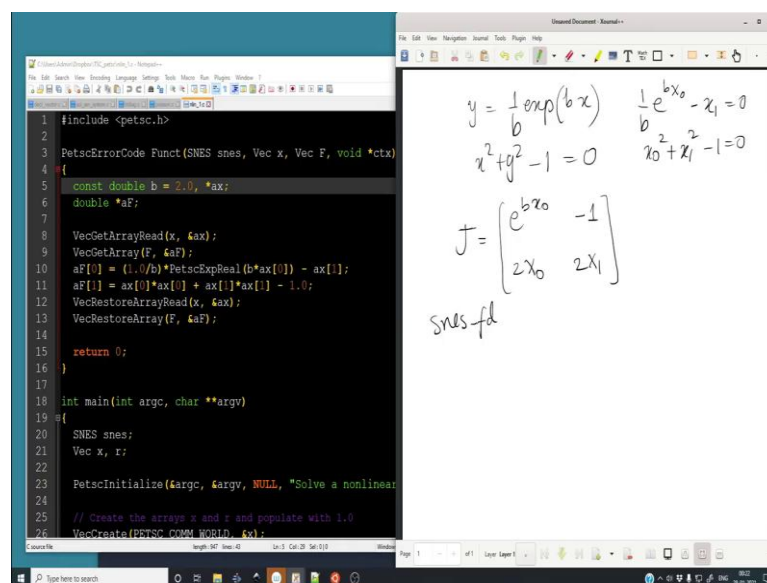


Tools in Scientific Computing
Prof. Aditya Bandopadhyay
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture - 34
Nonlinear Solver with Jacobian in PETSc

Hello, everyone. In this particular lecture, we are going to advance the program that we had written last time to also account for the analytical Jacobian.

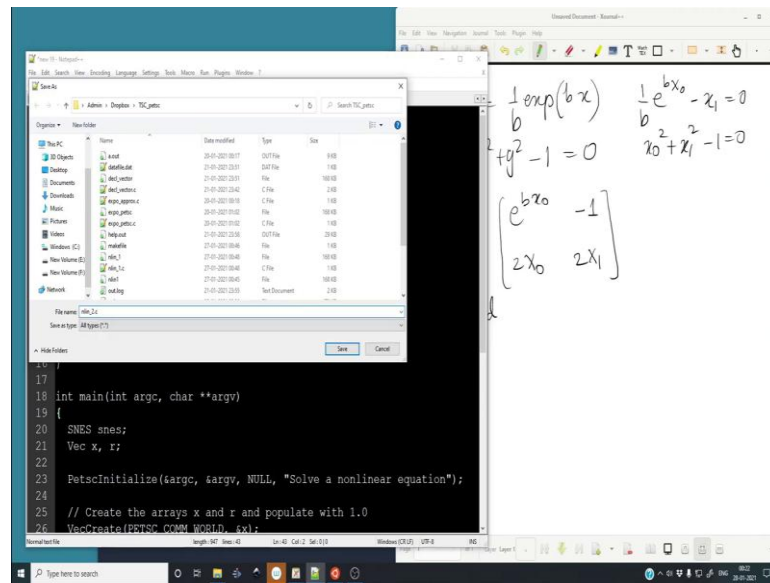
(Refer Slide Time: 00:51)



So, just to recap the program that we had written last time was to solve $y = \frac{1}{b} \exp(bx)$ and $x^2 + y^2 - 1 = 0$ and so, the Jacobian is essentially so, we arrange these equations like this $\frac{1}{b} e^{bx_0} - X_1 = 0$ and $X_0^2 + X_1^2 - 1 = 0$.

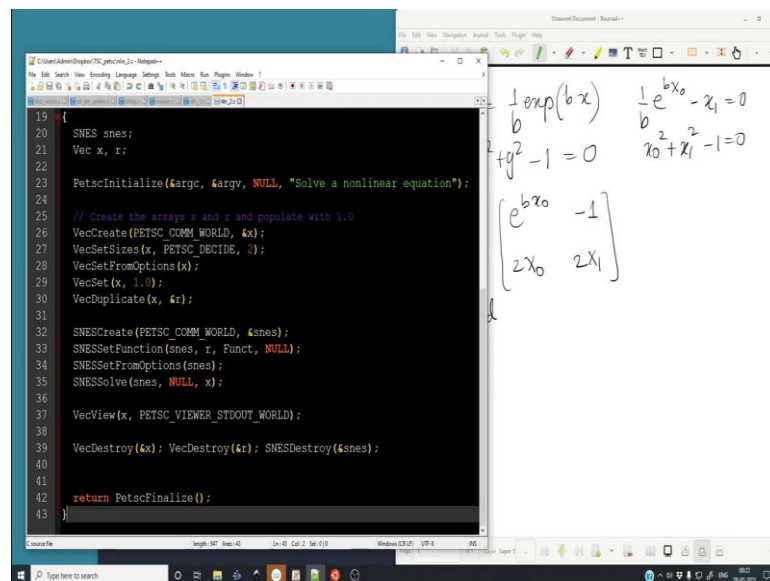
So, the analytical Jacobian was $J = \begin{pmatrix} e^{bx_0} & -1 \\ 2X_0 & 2X_1 \end{pmatrix}$. So, instead of passing the analytical Jacobian, we had asked PETSC to find out the Jacobian for us and that was done using the snes fd option, but now we can also pass the Jacobian.

(Refer Slide Time: 02:00)



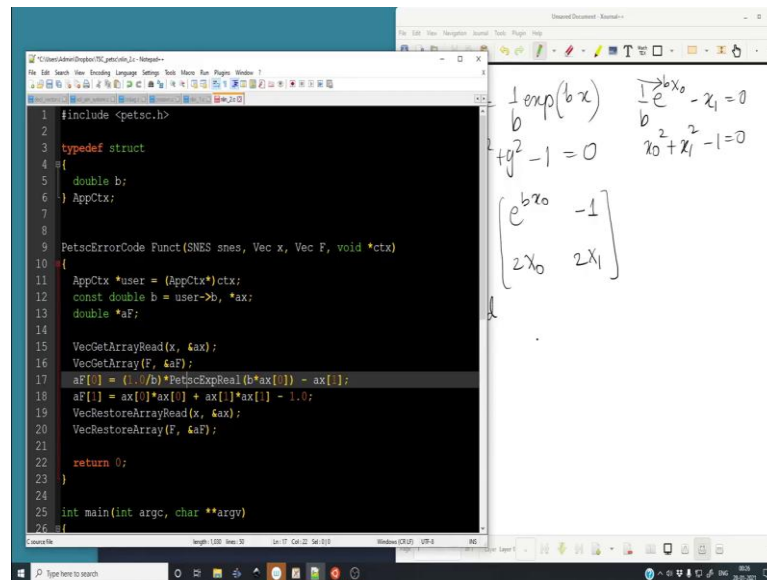
So, let me copy this program and let me first save this. So, it is nlin2.c, alright.

(Refer Slide Time: 02:10)



So, the core of the code remains the same, but in this particular version we are going to make some slight changes. So, we are going to make use of this context; so, this application context. So, we will go ahead and define a structure.

(Refer Slide Time: 02:29)



```
1 #include <petsc.h>
2
3 typedef struct
4 {
5     double b;
6 } AppCtx;
7
8
9 PetscErrorCode Funct(SNES snes, Vec x, Vec F, void *ctx)
10 {
11     AppCtx *user = (AppCtx*)ctx;
12     const double b = user->b, *ax;
13     double *aF;
14
15     VecGetArrayRead(x, &ax);
16     VecGetArray(F, &aF);
17     aF[0] = (1.0/b)*PetscExpReal(b*ax[0]) - ax[0];
18     aF[1] = ax[0]*ax[0] + ax[1]*ax[1] - 1.0;
19     VecRestoreArrayRead(x, &ax);
20     VecRestoreArray(F, &aF);
21
22     return 0;
23 }
24
25 int main(int argc, char **argv)
26 {
```

Handwritten notes on the right side of the editor:

$$\frac{1}{b} \exp(bx)$$
$$+y^2 - 1 = 0$$
$$\begin{bmatrix} e^{bx_0} & -1 \\ 2x_0 & 2x_1 \end{bmatrix}$$
$$\frac{1}{b} e^{bx_0} - x_1 = 0$$
$$b^2 x_0^2 + x_1^2 - 1 = 0$$

So, typedef struct or typedef struct and so, this defines a structure which we will call Ctx AppCtx. So, it stands for application context. So, this it is a structure which will contain for example, in this case the parameter b, alright. So, we will go and define inside the structure double b that is it. So, it is going to just contain the value of b.

So, now, when we enter into the function what we must do is, first extract what this pointer will contain ok. So, since it is of type void we must first cast this Ctx into the form AppCtx, ok. So, we have to do AppCtx *user. So, we are casting it into this it is equal to AppCtx. So, where this is the type casting step Ctx with the help of this we are taking whatever is there in Ctx and we are addressing it to user.

So, now what that means, is instead of hard coding b as 2.0 we can say b = user->b. So, whatever the user contains in its container as b is going to be assigned to b. This is a local copy which resides inside the Ctx address ok. So, in case you do not know about structs just have a look at any C programming book. This is how you can do it.

Alternately, user->b = *user.b that is whatever the user points to dot b, the b content of that point b ok. So, let us not go into that at this moment this is how you can address it and in various codes that you will write in PETSC, this is how you can declare the constant. So, now, we do not need it to be a const, but we can keep it to be a const, we do not need to rewrite a lot of code.

So, everything else in this function remains the same ok, apart from this user defined business nothing else changes, right. So, now what do we have? So, we have this the function nothing changes over here as well. So, over here we need to make the Ctx variable, ok.

(Refer Slide Time: 05:53)

```

17  aF[0] = (1.0/b)*PetscExpReal(b*ax[0]) - ax[1];
18  aF[1] = ax[0]*ax[0] + ax[1]*ax[1] - 1.0;
19  VecRestoreArrayRead(x, &ax);
20  VecRestoreArray(F, &aF);
21
22  return 0;
23  }
24
25  int main(int argc, char **argv)
26  {
27  SNES snes;
28  Vec x, r;
29  Mat J;
30  AppCtx user;
31  PetscInitialize(&argc, &argv, NULL, "Solve a nonlinear equation");
32
33  user.b = 2.0;
34
35  // Create the arrays x and r and populate with 1.0
36  VecCreate(PETSC_COMM_WORLD, &x);
37  VecSetSizes(x, PETSC_DECIDE, 2);
38  VecSetFromOptions(x);
39  VecSet(x, 1.0);
40  VecDuplicate(x, &r);
41
42

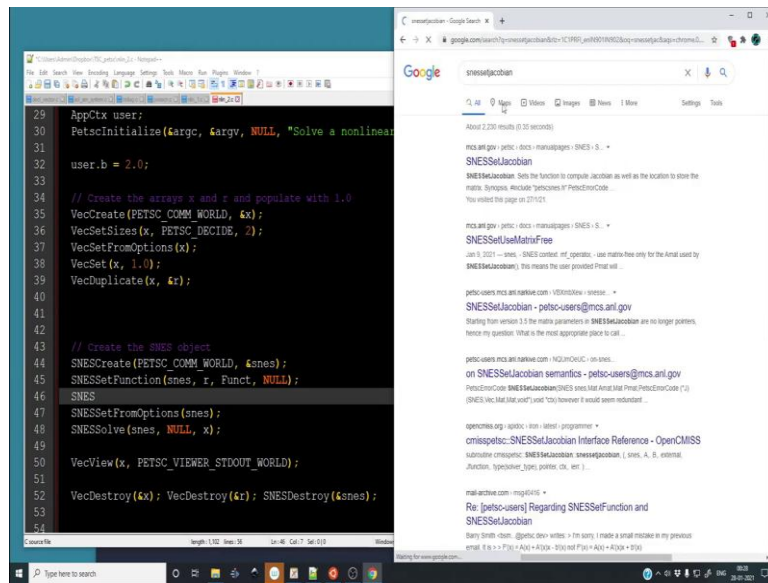
```

So, inside the main we must declare it. So, we will write app Ctx user and now, we must populate what the content of user is. So, user.b it will be equal to 2.0 and we have defined it over here. Alternately, we could pass this from a command line argument and simply use the command line interpreter a function like a to i, but for double you cast it from a string to a float, right.

So, it is as simple as that. We do not need to worry about this, but for now let us declare it in the main. The reason why we would like to declare it in the main is because once we have declared it in the main; we can simply pass them to the different functions. We do not need to keep changing the value in all the functions.

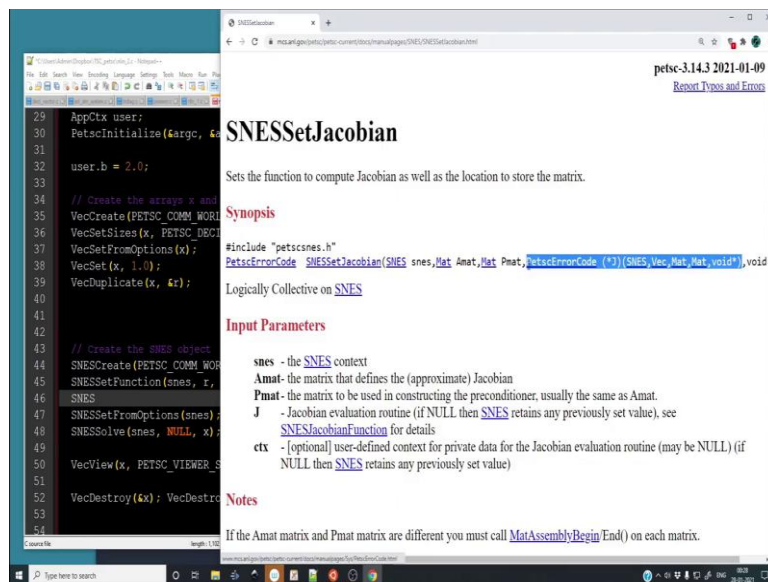
Well, in this particular easy case it is only 1 function which is making use of b, but in case you have many callbacks you need to define b carefully inside all the different functions if you are not using this Ctx object, alright.

(Refer Slide Time: 07:12)



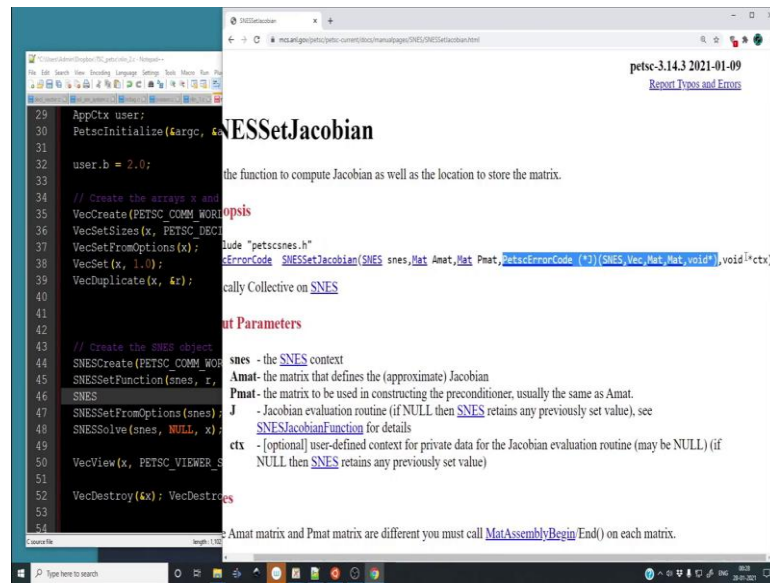
So, user.b is 2.0, it is fine. So, we have created the vectors and this is where we create this SNES object alright. So, now, before solving, so, once the function is set we must also set the Jacobian.

(Refer Slide Time: 07:47)



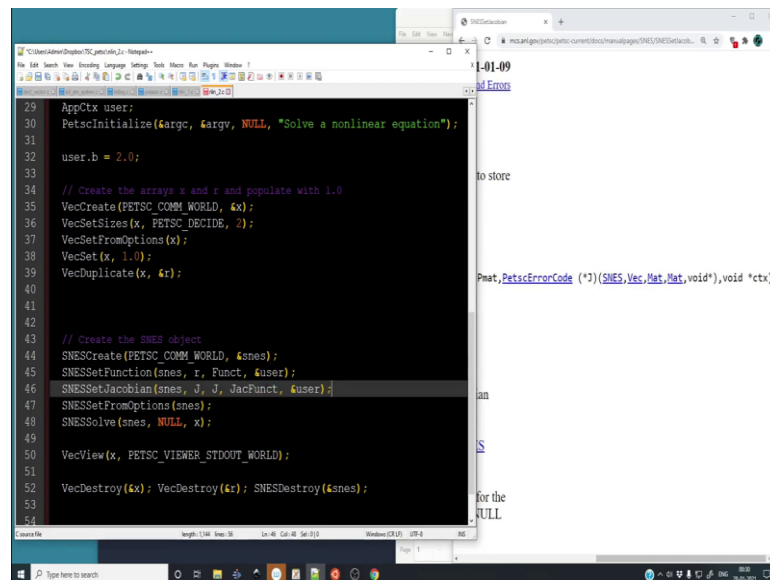
So, let me show you the function reference. So, it contains a very specific syntax. So, PetscErrorCode SNESSetJacobian SNES snes, Mat Amat, Mat Pmat and PetscErrorCode and that last PetscErrorCode points to the function mat we must define which will construct the matrices and I will explain that you know while.

(Refer Slide Time: 08:08)



And, lastly the user context ok. So, we must maintain the same call callback. So, let me keep it over here, right.

(Refer Slide Time: 08:25)



So, SNESSetJacobian snes. So, Mat Amat is the Amat is the matrix which defines the Jacobian, we will call it J. Then, we have Pmat which is the pre conditioner for the Jacobian. In this case it is going to be J again because usually it is the same as Amat ok. In case you wanted something else, we will do it in the function itself. Then we must provide the routine.

So, JacFunc and finally, you must provide the Ctx. So, in this case it will be the address of. So, it will be the address of user. Similarly, in the form function we will also pass the address of the CtxApp Ctx which is user, alright. So, we are passing it, we are casting it, we are making use of it, alright. So, now we must define what this JacFunc will be. So, JacFunc is a function which will help us create the matrix for the Jacobian, but before that we must create what J is.

(Refer Slide Time: 09:57)

```

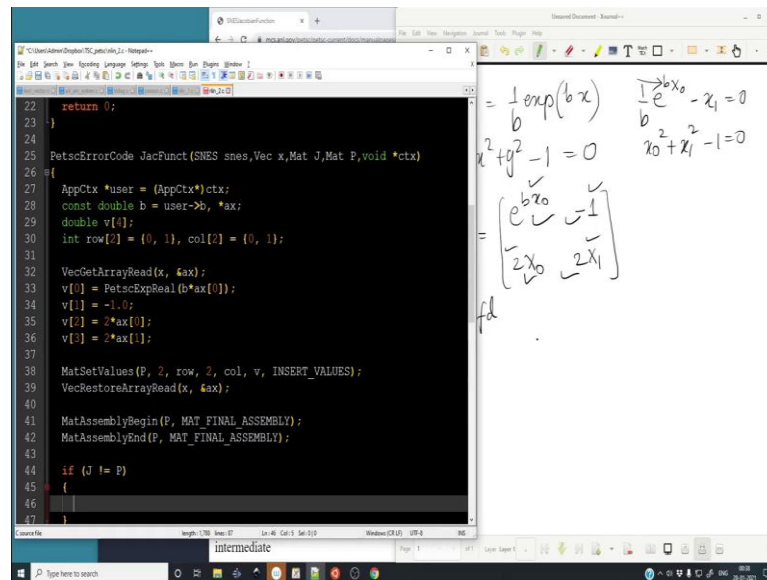
26 {
27     SNES snes;
28     Vec x, r;
29     Mat J;
30     AppCtx user;
31     PetscInitialize(&argc, &argv, NULL, "Solve a nonlinear equation");
32
33     user.b = 2.0;
34
35     // Create the arrays x and r and populate with 1.0
36     VecCreate(PETSC_COMM_WORLD, &x);
37     VecSetSizes(x, PETSC_DECIDE, 2);
38     VecSetFromOptions(x);
39     VecSet(x, 1.0);
40     VecDuplicate(x, &r);
41
42     MatCreate(PETSC_COMM_WORLD, &J);
43     MatSetSizes(J, PETSC_DECIDE, PETSC_DECIDE, 2, 2);
44     MatSetFromOptions(J);
45     MatSetup(J);
46
47     // Create the SNES object
48     SNESCreate(PETSC_COMM_WORLD, &snes);
49     SNESSetFunction(snes, r, Funct, &user);
50     SNESSetJacobian(snes, J, J, JacFunc, &user);
51     SNESSetFromOptions(snes);

```

So, after that we will create Mat J; after this we will create the matrix. So, MatCreate; so, it will have the usual calling sequence. After MatCreate there is MatSetSizes, then there is MatSetFromOptions, then there is MatSetup and this will be J itself. And, SetFromOptions will also be set to J SetSizes ok. So, it will create PETSC COMM WORLD and we will pass the address of J. SetSizes will be to J PETSC DECIDE comma PETSC DECIDE , 2 , 2, alright.

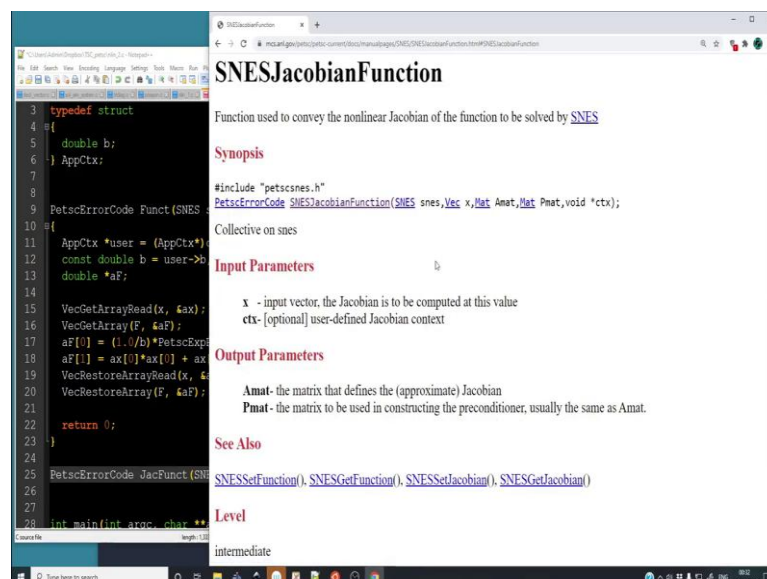
So, we have to set the size to be 2 , 2 because in our particular problem we have a 2×2 Jacobian and we can hard code it, no problem, alright. So, we have created the Jacobian over here. So, now, all that is left is to construct this JacFunc function. So, it has to be a function which has this particular callback, right.

(Refer Slide Time: 11:37)



Has to be a function which has this particular callback. It is the same as this actually, I mean almost the same as this. So, PetscErrorCode JacFunct, alright and it will take SNES snes, then it will accept it needs a vector, ok.

(Refer Slide Time: 11:59)



Let us go over here. So, it needs the input x vector like over here. So, like x it will need the Jacobian matrix, it will need the pre-conditioner matrix and it will need a void star ctx, alright. So, inside this function we must first unpack the Ctx like we did over here.

So, we can copy this and then we can assign `const double b = user->b`, and we can define the auxiliary array to store `x`, alright.

Then, we must also declare this has to be tabbed and this does not have to be tab, but its good to keep it indented. So, then we must create a value vector with which we will assign the values of the matrix ok. This value vector is just to sort of plug in values at the desired locations and we have done something like this in the previous programs. So, we have `double v[4]`. So, its a array of size 4 which will contain the values.

Lastly, we must have an integer array for rows. So, row 2 is equal to it is simply going to be row 0, 1 and col 2 will be again 0, 1, alright. So, we have declared the insertion rows and columns and now we can simply go ahead and create the matrix. So, first we must unpack what the `x` vector will be.

So, `VecGetArrayRead` it will be going from `x` to the address of auxiliary `ax` and over here `v[0]` will be the first entry. So, what is the first entry? It is e^{bx} . So, that will be `PetscExpReal(b*ax[0])`, then `v[1] = -1.0; v[2] = 2*ax[0]; v[3] = 2*ax[1]`. So, we have declared these four entries, alright.

So, now we must insert these into the array. So, how do we insert them? So, it is `MatSetValues`. So, now, we will put these values into `p` because `J` and `P` are one and the same. So, `P`, so, we are going to insert two rows row and 2 columns. So, here row and col correspond to the indices where we are going to insert and its packed as row fixed column, then column, the next row, then column.

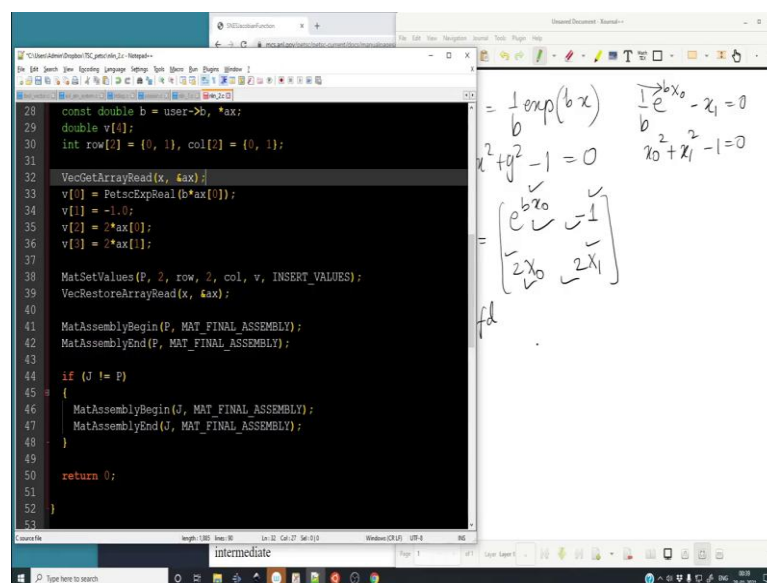
So, what it means is we are going to insert two rows at a time and two columns at a time and with the sequence is going to be 1, 2, 3, 4, right. Next we have to insert which values you have to specify the values to be inserted. So, it is simply the elements of the vector `b` and we will `INSERT VALUES`. So, if in case you do not remember these, this is the way you insert values into a matrix in `Petsc`. Once this is done we can restore the vector so, `VecRestoreArrayRead(x, &ax)`, alright.

Now, we can go to the matrix assembly. So, `MatAssemblyBegin`, its going to simply be `P, MAT_FINAL_ASSEMBLY` and we are going to copy this paste it over here and this is simply going to be `End`. So, with the help of this we assemble the matrix. Now, if `J != P`, right then we must also assemble the matrix `J`.

Now, remember we are passing from the main J, J which means through this modification to P whatever we have done J is actually modified. In case J and P are distinct, we must assemble the matrix J as it is; if in case they are equal an assembly of P is turned around to assembly of J itself, alright.

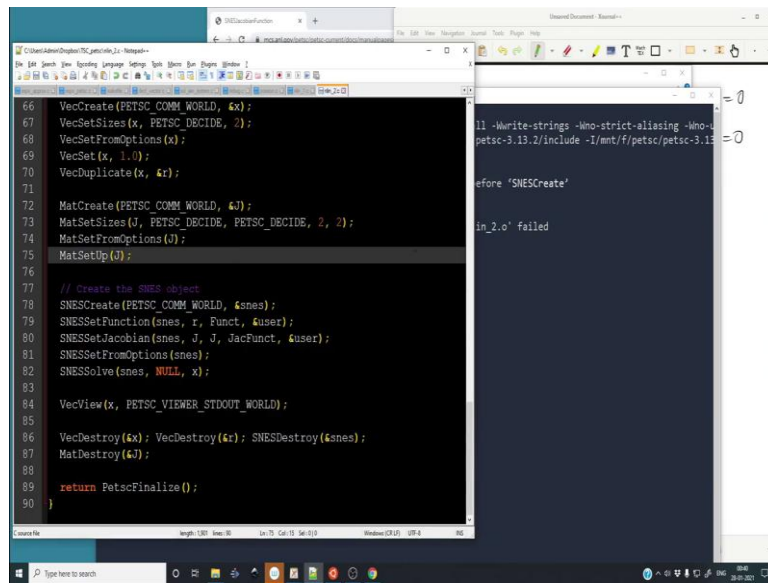
So, if $J \neq P$ this is the case for the pre conditioner and the Jacobian are different. In most of the cases, it will be the same. So, this line is sort of redundant anyway we must have. So, we will just copy these lines because it is the same lines, but done for J, right.

(Refer Slide Time: 17:51)



So, over here we will have J and over here we will have J and at the end of successful function call we will return 0 because the output is a PetscErrorCode. So, this particular set of lines declares how the Jacobian is analytically built and this is the code in particular, alright.

(Refer Slide Time: 18:34)

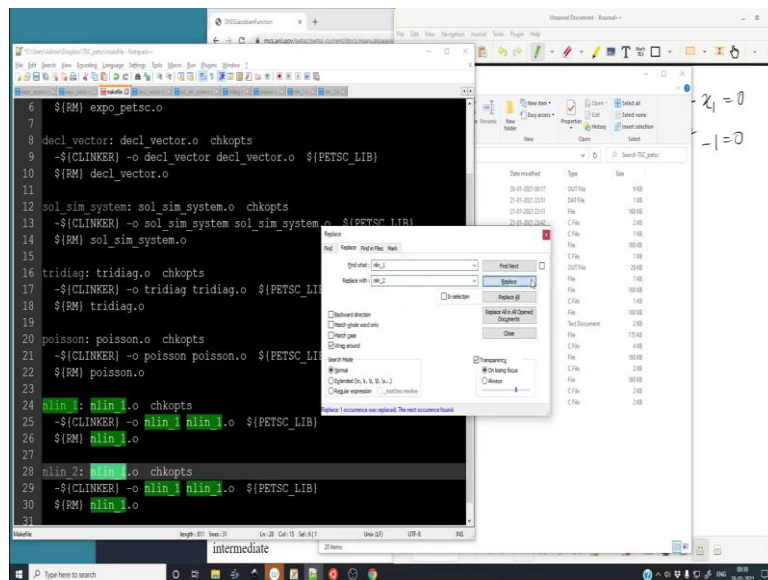


```
66 VecCreate(PETSC_COMM_WORLD, &x);
67 VecSetSizes(x, PETSC_DECIDE, 2);
68 VecSetFromOptions(x);
69 VecSet(x, 1.0);
70 VecDuplicate(x, &r);
71
72 MatCreate(PETSC_COMM_WORLD, &J);
73 MatSetSizes(J, PETSC_DECIDE, PETSC_DECIDE, 2, 2);
74 MatSetFromOptions(J);
75 MatSetUp(J);
76
77 // Create the SNES object
78 SNESCreate(PETSC_COMM_WORLD, &snes);
79 SNESSetFunction(snes, r, Funct, &user);
80 SNESSetJacobian(snes, J, J, JacFunct, &user);
81 SNESSetFromOptions(snes);
82 SNESolve(snes, NULL, x);
83
84 VecView(x, PETSC_VIEWER_STDOUT_WORLD);
85
86 VecDestroy(&x); VecDestroy(&r); SNESDestroy(&snes);
87 MatDestroy(&J);
88
89 return PetscFinalize();
90 }
```

```
11 -Write-strings -Wno-strict-aliasing -Wno-
petsc-3.13.2/include -I/nmt/f/petsc/petsc-3.13
before 'SNEScreate'
in_2.o' failed
```

So, now that we have called it through the function we have created the matrix and we must finally, destroy the matrix. So, MatDestroy, ampersand of J, alright. So, set from options is there anything left? No. So, that is pretty much it. So, now, we can go ahead and first edit the make file. So, the make file this somewhere here no, it is not. Yes, it is.

(Refer Slide Time: 19:11)



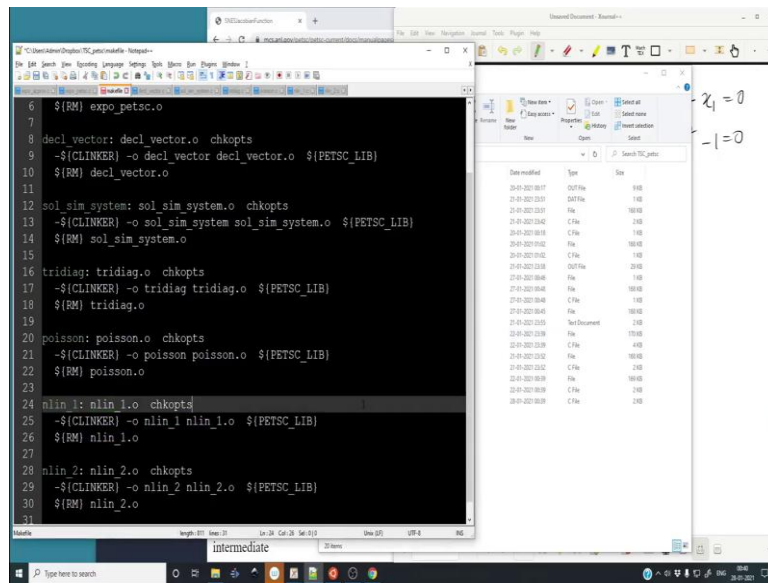
```
6 $(RM) expo_petsc.o
7
8 decl_vector: decl_vector.o chkopts
9 	-$(LINKER) -o decl_vector decl_vector.o $(PETSC_LIB)
10 $(RM) decl_vector.o
11
12 sol_sim_system: sol_sim_system.o chkopts
13 	-$(LINKER) -o sol_sim_system sol_sim_system.o $(PETSC_LIB)
14 $(RM) sol_sim_system.o
15
16 tridiag: tridiag.o chkopts
17 	-$(LINKER) -o tridiag tridiag.o $(PETSC_LIB)
18 $(RM) tridiag.o
19
20 poisson: poisson.o chkopts
21 	-$(LINKER) -o poisson poisson.o $(PETSC_LIB)
22 $(RM) poisson.o
23
24 nlin 1: nlin_1.o chkopts
25 	-$(LINKER) -o nlin_1 nlin_1.o $(PETSC_LIB)
26 $(RM) nlin_1.o
27
28 nlin 2: nlin_2.o chkopts
29 	-$(LINKER) -o nlin_2 nlin_2.o $(PETSC_LIB)
30 $(RM) nlin_2.o
31
```

Search dialog box: Search for 'nlin_1' in 'intermediate'.

File explorer: Search for 'petsc' in the file system.

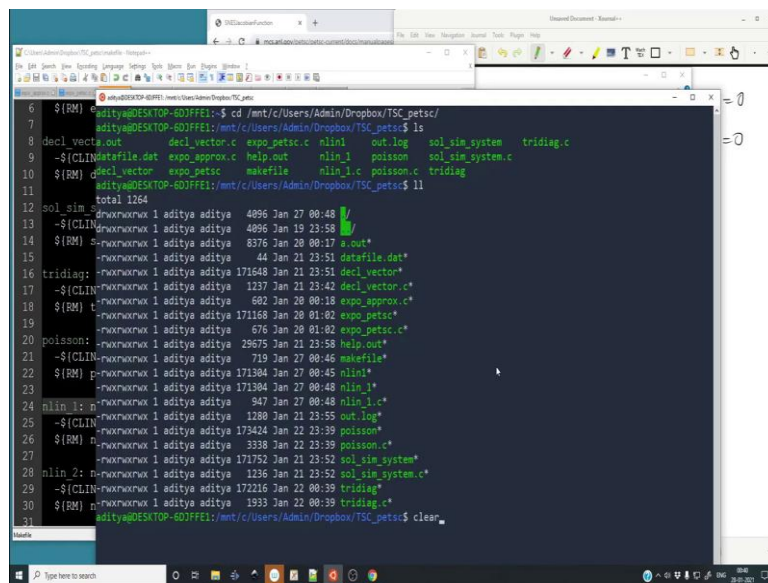
So, we are going to make a new target and nlin 2, control H.

(Refer Slide Time: 19:24)



Yes, you can nlin 2 Replace Replace, Replace, Replace, Replace, Close.

(Refer Slide Time: 19:28)



So, we have created a new target let me go to the terminal and clear.

(Refer Slide Time: 19:30)

```
aditya@DESKTOP-6D3JPFEE:~/mnt/f/petsc/petsc-3.13.2$ ./nlin_2 -snes_monitor_short
0 SNES Function norm 2.87411
1 SNES Function norm 0.859139
2 SNES Function norm 0.168996
3 SNES Function norm 0.0110689
4 SNES Function norm 6.61811e-05
5 SNES Function norm 2.41926e-09
Vec Object: 1 MPI processes
return F type: seq
0.319632
0.947542
aditya@DESKTOP-6D3JPFEE:~/mnt/f/petsc/petsc-3.13.2$ ./nlin_2 -snes_monitor_short -snes_fd
0 SNES Function norm 2.87411
1 SNES Function norm 0.859139
2 SNES Function norm 0.168996
3 SNES Function norm 0.0110689
4 SNES Function norm 6.61814e-05
5 SNES Function norm 2.42078e-09
Vec Object: 1 MPI processes
return F type: seq
0.319632
0.947542
```

So, make nlin_2, let us see if we have some error there is an error. So, let us see before Snes we have missed this semicolon. Do not make these mistakes, alright. So, now we can simply call ./nlin_2 with a petsc monitor or rather snes monitor short and because we have passed Jacobian, we do not need to additionally do the -fd some -snes fd. So, this is using the computed Jacobian instead of the analytical Jacobian.

(Refer Slide Time: 20:30)

```
aditya@DESKTOP-6D3JPFEE:~/mnt/f/petsc/petsc-3.13.2$ ./nlin_2 -snes_monitor_short
0 SNES Function norm 2.87411
1 SNES Function norm 0.859139
2 SNES Function norm 0.168996
3 SNES Function norm 0.0110689
4 SNES Function norm 6.61811e-05
5 SNES Function norm 2.41926e-09
Vec Object: 1 MPI processes
return F type: seq
0.319632
0.947542
aditya@DESKTOP-6D3JPFEE:~/mnt/f/petsc/petsc-3.13.2$ ./nlin_2 -snes_monitor_short -snes_fd
0 SNES Function norm 2.87411
1 SNES Function norm 0.859139
2 SNES Function norm 0.168996
3 SNES Function norm 0.0110689
4 SNES Function norm 6.61814e-05
5 SNES Function norm 2.42078e-09
Vec Object: 1 MPI processes
return F type: seq
0.319632
0.947542
```

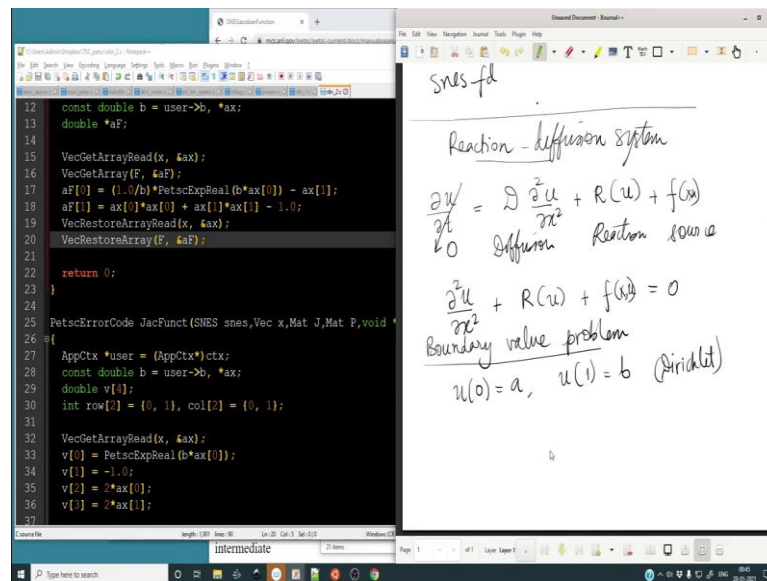
Well, it does give the same answer and it is expected for such a simple problem, ok. But, there is some difference. It converges slightly better than this. The fact is, it does not

change anything in this particular case because it is a simple problem, alright. So, this is how we do it and ok. So, this is how you go about passing the Jacobian to the function and really speaking this is a very simple way to go about things.

So, that if you know what the Jacobian is going to be, I mean passing it to a function will always help you reduce the number of function evaluations that one needs to do for the Jacobian because in this case you need a lot of function evaluations because you are going to loop over the entire matrix, perform a finite difference to find out the Jacobian matrix elements, right.

In the case of these Snes, the Jacobian function you do not you have to you do not have to do all that you simply get what you want in a straightforward manner. So, let us now discuss where we can go with this. So, let us consider a very simple 1-dimensional PDE.

(Refer Slide Time: 22:14)



So, let us consider a reaction diffusion system ok. So, $\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2} + R(u) + f(x)$ where u is some species is equal to some diffusion coefficient times well, application of u and in case of 1D its going to be $\frac{\partial^2 u}{\partial x^2}$ plus some reaction term which may depend on the local concentration plus some source term. So, this is the diffusion term, this is the reaction term and this is the source if it is a steady state problem, then this goes to 0.

And, thus we can write essentially $\frac{\partial^2 u}{\partial x^2} + R(u) + f(x) = 0$, where we have sort of absorbed this diffusion into these terms, really speaking it does not really matter what D is going to be itself it is a constant. So, how do we well it can be a function of u as well.

So, how do we go about solving this equation? Well, first of all, its a boundary value problem and in order to do a boundary value problem we need value at 0 which is a and value at 1 which is b. So, these are the Dirichlet boundary conditions. And, so, let us consider a simple case where we know what the solution is going to be. This is just to sort of verify whether whatever we are doing is correct or not.

(Refer Slide Time: 24:34)

```

12 const double b = user->b, *ax;
13 double *aF;
14
15 VecGetArrayRead(x, &ax);
16 VecGetArray(F, &aF);
17 aF[0] = (1.0/b)*PetscExpReal(b*ax[0]) - ax[1];
18 aF[1] = ax[0]*ax[0] + ax[1]*ax[1] - 1.0;
19 VecRestoreArrayRead(x, &ax);
20 VecRestoreArray(F, &aF);
21
22 return 0;
23 }
24
25 PetscErrorCode JacFunct(SNES snes, Vec x, Mat J, Mat P, void *
26 =
27 AppCtx *user = (AppCtx*)ctx;
28 const double b = user->b, *ax;
29 double v[4];
30 int row[2] = {0, 1}, col[2] = {0, 1};
31
32 VecGetArrayRead(x, &ax);
33 v[0] = PetscExpReal(b*ax[0]);
34 v[1] = -1.0;
35 v[2] = 2*ax[0];
36 v[3] = 2*ax[1];
37

```

Boundary value problem
 $u(0) = a, u(1) = b$ (Dirichlet)

$\frac{\partial^2 u}{\partial x^2} - \sqrt{u} = 0$
 $u'' = \sqrt{u}$

Ed Bueler
 PETSc for
 partial diff
 eq's SIAM

$u = M(1+x)^4, \sqrt{u} = \sqrt{M}(1+x)^2$
 $u' = 4M(1+x)^3$
 $u'' = 12M(1+x)^2$
 $12M(1+x)^2 = \sqrt{M}(1+x)^2$
 $\rightarrow M = \left(\frac{12}{\sqrt{M}}\right)^2$

Analytical solution:

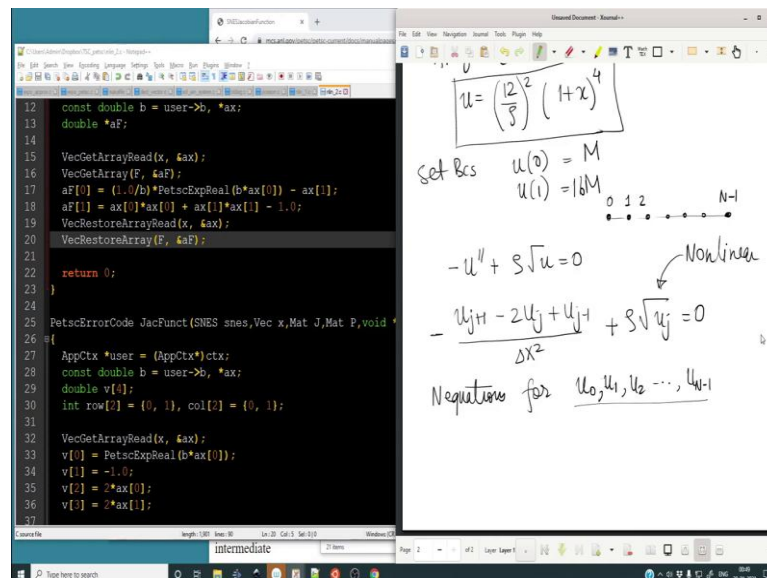
So, let us consider a case that there is no source, but there is only reaction. So, essentially we are going to cover $\frac{\partial^2 u}{\partial x^2} - \rho\sqrt{u} = 0$. This is a very famous example, but I have taken this example from the book of Ed Bueler from University of Alaska and the name of the book is signed let me see.

Let us see for partial differential equations, right. So, its a very its a new book and it is published by SIAM publications. So, its a new book and do check it out if you are interested in building applications with the PESTc (Refer Time: 25:27). I am going to use this one example.

And, what I am going to do is rather what we do is we choose a solution to this and the solution to this we can eyeball it is going to be something like $u = M(1+x)^4$ because two derivatives of this and one square root of this will give the same order

So, $u' = 4M(1+x)^3$, $u'' = 12M(1+x)^2$ and $\sqrt{u} = \sqrt{M}(1+x)^2$. So, now, when you equate this 2 or so, we have $M = \left(\frac{12}{\rho}\right)^2$.

(Refer Slide Time: 26:59)



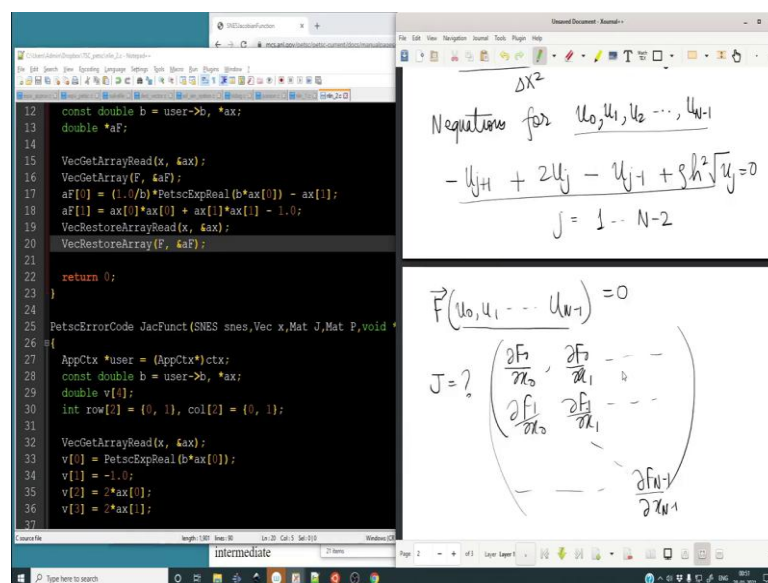
So, when M is this we get the analytical solution we have the analytical solution that $u = \left(\frac{12}{\rho}\right)^2 (1+x)^4$, alright. So, with the help of this we can set the boundary conditions as well. So, $u(0)$ is going to be. So, let us keep it as M and let us keep it as M because we will pass it as a user defined context later and $u(1)$ is going to be M times 2^4 which is 16, alright.

So, these are the this is the test problem. So, that whatever we are going to solve we can sort of test it out whether it is correct or not, alright. So, now how do we go about solving this using Petsc. Let us strategize it first and in the next lecture we are going to solve it. So, what do we have? We have $-u'' + \rho\sqrt{u} = 0$.

This is $-\frac{u_{j+1} - 2u_j + u_{j-1}}{\Delta X^2} + \rho\sqrt{u_j} = 0$. Essentially this is the equation and if we have n number of nodes starting from 0, 1 2 all the way to N-1. So, we will have N equations N equations for u_0, u_1, u_2 all the way to u_{N-1} .

So, we are going to have N- 1 equation and rather N equations for these nodes and. They are non-linear solely because of this term and because they are non-linear, what we need to do is to assemble them and use a non-linear solver to solve.

(Refer Slide Time: 29:20)

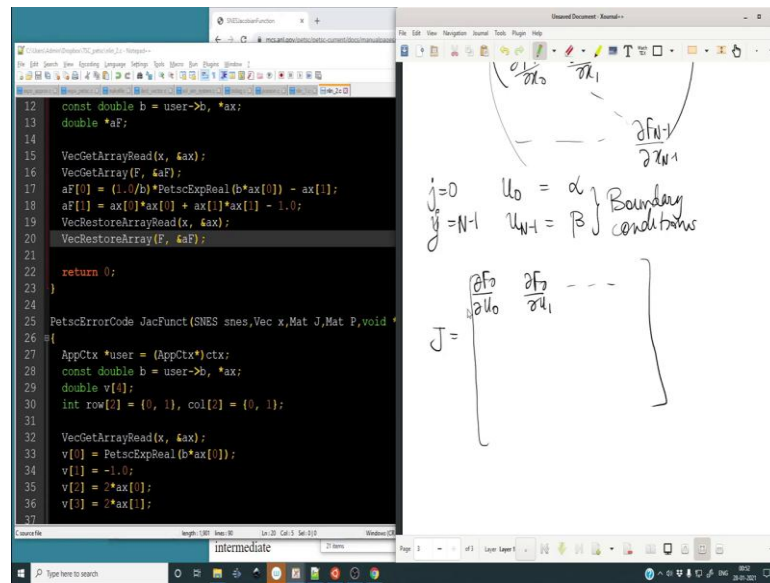


So, essentially we have $\vec{F}(u_0, u_1, \dots, u_{N-1}) = 0$. So, this F vector is equal to 0 and this is what we started with this is what the discussion was in the first place, alright. So, what is

the Jacobian going to be? It is going to be
$$\begin{pmatrix} \frac{\partial F_0}{\partial x_0} & \frac{\partial F_0}{\partial x_1} & \dots \\ \frac{\partial F_1}{\partial x_0} & \ddots & \dots \\ \dots & \dots & \frac{\partial F_{N-1}}{\partial x_{N-1}} \end{pmatrix}$$
.

All these terms are going to comprise Jacobian, alright and really speaking we want to find out u which satisfies. So, whatever the role was of x over here, in this problem, the same as the role of u over here, alright. So, let us first see what the equations are. So, its $-u_{j+1} + 2u_j - u_{j-1} + \rho h^2 \sqrt{u_j} = 0$. So, this for j = 1 to N - 2. Why 1 to N - 2?

(Refer Slide Time: 30:53)



Because $j = 0$ is going to be simply $u_0 = \alpha$ and $j = N - 1$ is simply going to be $u_{N-1} = \beta$. So, these are the boundary conditions. So, how is the Jacobian going to look like? So, Jacobian matrix will look something like this.

So, this will be $\frac{\partial F_0}{\partial x_0}$, $\frac{\partial F_0}{\partial x_1}$ and so on, but $\frac{\partial F_0}{\partial x_0}$ because the first equation is simply

$u_0 = \alpha$. So, $\frac{\partial F_0}{\partial x}$. So, in this case x is u . So, this will be $\frac{\partial F_0}{\partial u_0}$ this will be $\frac{\partial F_0}{\partial u_1}$ and so on.

So, $\frac{\partial F_0}{\partial u_1}$ is going to be 1.

(Refer Slide Time: 31:51)

The code editor shows the following C code:

```

12 const double b = user->b, *ax;
13 double *aF;
14
15 VecGetArrayRead(x, &ax);
16 VecGetArray(F, &aF);
17 aF[0] = (1.0/b)*PetscExpReal(b*ax[0]) - ax[1];
18 aF[1] = ax[0]*ax[0] + ax[1]*ax[1] - 1.0;
19 VecRestoreArrayRead(x, &ax);
20 VecRestoreArray(F, &aF);
21
22 return 0;
23 }
24
25 PetscErrorCode JacFunct(SNES snes, Vec x, Mat J, Mat P, void *
26 {
27 AppCtx *user = (AppCtx*)ctx;
28 const double b = user->b, *ax;
29 double v[4];
30 int row[2] = {0, 1}, col[2] = {0, 1};
31
32 VecGetArrayRead(x, &ax);
33 v[0] = PetscExpReal(b*ax[0]);
34 v[1] = -1.0;
35 v[2] = 2*ax[0];
36 v[3] = 2*ax[1];
37

```

The handwritten document shows the following:

Boundary conditions: $u_0 = \alpha$, $u_{N-1} = \beta$

Jacobian matrix J :

$$J = \begin{bmatrix} 1 & 0 & \dots & \dots \\ -1 & 2\frac{\partial^2}{\partial x_0^2} & -1 & \dots \\ & -1 & 2\frac{\partial^2}{\partial x_1^2} & -1 \\ & & & \dots & \dots & \dots & 1 \end{bmatrix}$$

Equation: $-u_0 + 2u_1 - u_2 + \sum_{i=1}^{N-1} u_i h^2 = 0$

Partial derivatives:

$$\frac{\partial F_1}{\partial b_0} = -1, \quad \frac{\partial F_1}{\partial u_1} = 2 + \frac{2h^2}{2u_1}, \quad \frac{\partial F_1}{\partial u_2} = -1$$

This fellow is going to be 0 because the first equation only consists of u_0 . So, every other term will be 0. Similarly, on the last row this final element is going to be 1 because this equation is this and the last term the last term in the Jacobian is going to be $\frac{\partial F_{N-1}}{\partial x_{N-1}}$ and hence it is going to be 1. All the other terms will be 0 and what about the first term?

(Refer Slide Time: 32:30)

The code editor shows the following C code:

```

12 const double b = user->b, *ax;
13 double *aF;
14
15 VecGetArrayRead(x, &ax);
16 VecGetArray(F, &aF);
17 aF[0] = (1.0/b)*PetscExpReal(b*ax[0]) - ax[1];
18 aF[1] = ax[0]*ax[0] + ax[1]*ax[1] - 1.0;
19 VecRestoreArrayRead(x, &ax);
20 VecRestoreArray(F, &aF);
21
22 return 0;
23 }
24
25 PetscErrorCode JacFunct(SNES snes, Vec x, Mat J, Mat P, void *
26 {
27 AppCtx *user = (AppCtx*)ctx;
28 const double b = user->b, *ax;
29 double v[4];
30 int row[2] = {0, 1}, col[2] = {0, 1};
31
32 VecGetArrayRead(x, &ax);
33 v[0] = PetscExpReal(b*ax[0]);
34 v[1] = -1.0;
35 v[2] = 2*ax[0];
36 v[3] = 2*ax[1];
37

```

The handwritten document shows the following:

Boundary conditions: $u_0 = \alpha$, $u_{N-1} = \beta$

Jacobian matrix J :

$$J = \begin{bmatrix} 1 & 0 & \dots & \dots \\ -1 & 2\frac{\partial^2}{\partial x_0^2} & -1 & \dots \\ & -1 & 2\frac{\partial^2}{\partial x_1^2} & -1 \\ & & & \dots & \dots & \dots & 1 \end{bmatrix}$$

Equation: $-u_0 + 2u_1 - u_2 + \sum_{i=1}^{N-1} u_i h^2 = 0$

Partial derivatives:

$$\frac{\partial F_1}{\partial b_0} = -1, \quad \frac{\partial F_1}{\partial u_1} = 2 + \frac{2h^2}{2u_1}, \quad \frac{\partial F_1}{\partial u_2} = -1$$

So, first the equation for the node with index 1, that is the second node the equation is

$$-u_0 + 2u_1 - u_2 + \rho\sqrt{u_1}h^2 = 0. \text{ So, } \frac{\partial F_1}{\partial u_0} = -1. \text{ So, this is going to be } -1, \frac{\partial F_1}{\partial u_1} = 2 + \frac{\rho h^2}{2\sqrt{u_1}},$$

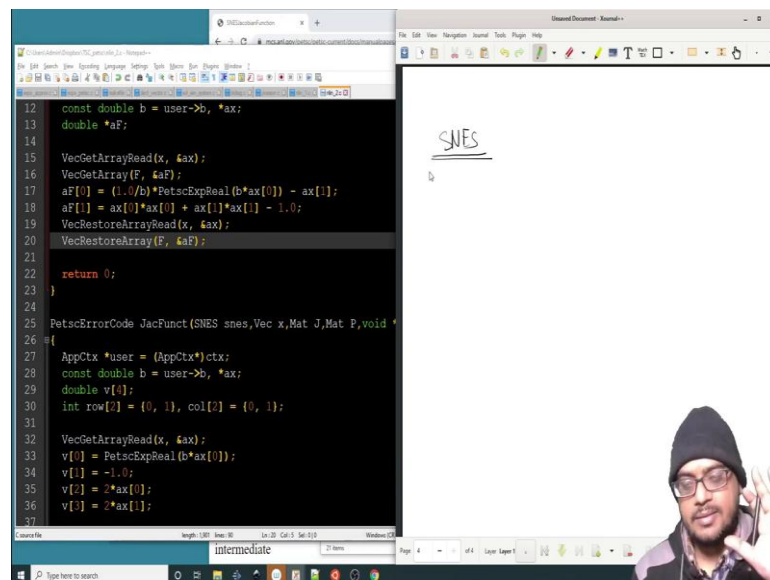
alright. So, this is going to be this h^2 over here as well. I am sorry, I forgot to write.

So, it is $\frac{\rho h^2}{2\sqrt{u_1}}$ and the third term will be $\frac{\partial F_1}{\partial u_2}$ which is going to be -1, all the other terms

are going to be 0 which is $2 + \frac{\rho h^2}{2\sqrt{u_1}}, -1, 0$ and for the next term this will be this and all

the way to $N - 2$ row number row index. So, what have we done? We have constructed the Jacobian, right.

(Refer Slide Time: 33:53)



So, then we can simply use a SNES solver to find out the steady state solution and in the next class we are going to do exactly this. Until then, have a good day.

Bye.