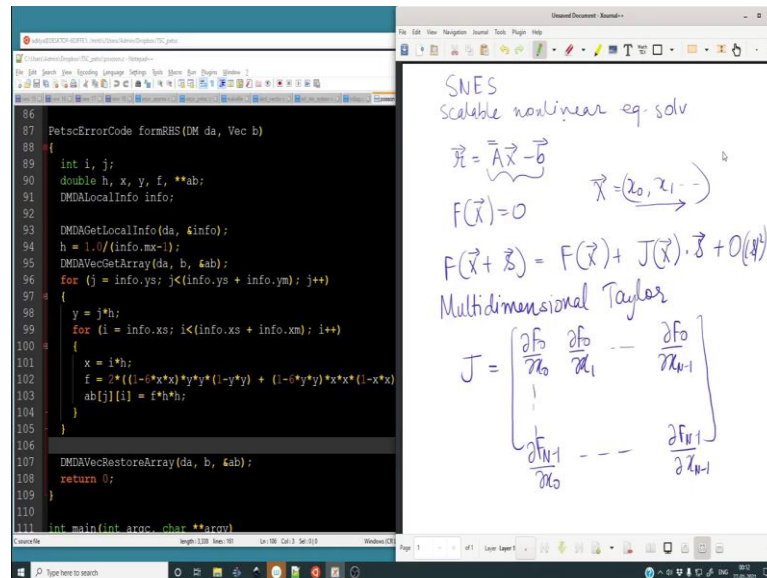


**Tools in Scientific Computing**  
**Prof. Aditya Bandopadhyay**  
**Department of Mechanical Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 33**  
**Nonlinear Solver of PETSc**

(Refer Slide Time: 00:26)



Hi, everyone in this particular lecture we are going to look at Non-linear Solver in PETSc. It is called as SNES and it stands for not the Super Nintendo Entertainment System, but it stands for Scalable Non-linear Equation Solver.

And we have had a look at the newton iterations and all that in python, but the fact is when you are trying to solve differential equations when you are trying to construct some residual so the residual is defined as  $A \times X - b$  and your target is to minimize this residual. So, often this can be non-linear, it can be polynomial, it can be transcendental.

But what happens when it is non-linear? So you need to solve for non-linear algebraic equations and that requires a scalable non-linear solver. So, the crux of solving something which we have already done say for example, you want to solve some  $F(X) = 0$  and this is a column vector which contains all the functions at  $X$  is the vector  $x_0, x_1$  and so on.

So, we will see in an example what exactly this means. So, if you have a multi-dimensional function then  $F(X) + \text{some step} = F(X) + \text{the Jacobian evaluated at } X \times \text{the step} + O(s^2)$ . So, this comes from a multi-dimensional Taylor series approximation and this Jacobian.

So, what is this Jacobian? So, Jacobian is defined as this particular matrix. So,  $\partial F_0 / \partial x_0, \partial F_0 / \partial x_1$  so on  $\partial F_0 / \partial x_{N-1}$ . If there are N number of elements in this and similarly this becomes  $\partial F_{N-1} / \partial x_0$  all the way to  $\partial F_{N-1} / \partial x_{N-1}$ . So, this is called as the Jacobian of this particular Taylor series. And so the point is if we are near a root right.

(Refer Slide Time: 03:35)

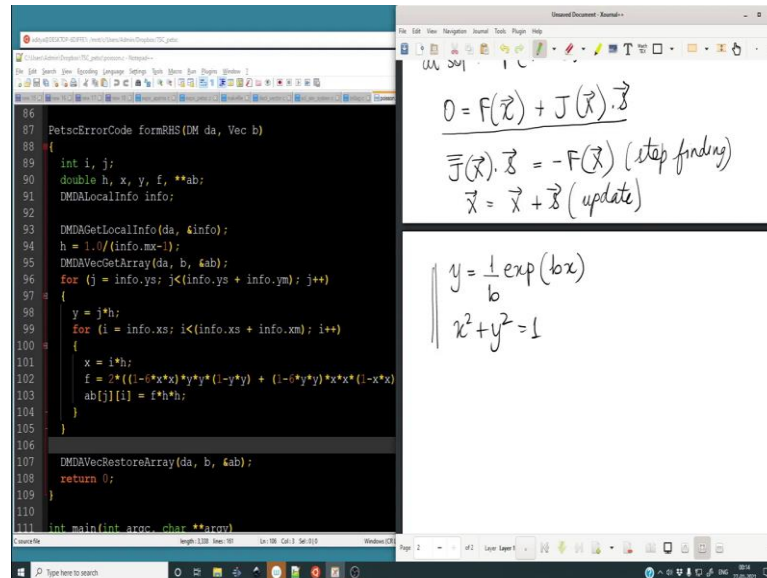
The image shows a presentation slide with two main parts. On the left, a code editor displays C++ code for a Newton-Raphson method using PETSc. The code includes comments and function calls like `PetscErrorCode formRHS(DM da, Vec b)` and `DMDAVecRestoreArray(da, b, &ab)`. On the right, a whiteboard contains handwritten mathematical notes. At the top, it says 'Multidimensional' and 'Jacobian'. Below that, the Jacobian matrix  $J$  is defined as a matrix of partial derivatives:  $J = \begin{bmatrix} \frac{\partial F_0}{\partial x_0} & \frac{\partial F_0}{\partial x_1} & \dots & \frac{\partial F_0}{\partial x_{N-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_{N-1}}{\partial x_0} & \frac{\partial F_{N-1}}{\partial x_1} & \dots & \frac{\partial F_{N-1}}{\partial x_{N-1}} \end{bmatrix}$ . Below the matrix, it says 'at sol<sup>n</sup>:  $F(\vec{x} + \vec{\delta}) = 0$ '. Then, the equation  $0 = F(\vec{x}) + J(\vec{x}) \cdot \vec{\delta}$  is written, followed by the formula for the next iteration:  $\vec{\delta}(\vec{x}) = -F(\vec{x}) / J(\vec{x})$ .

So, you are at some point in the parameter space not the parameter space, but the variable space  $X$  and this is the solution. So, if this were to be true then  $F$  at  $X + s$  would be equal to 0 this is that step  $s$ . So, if you are having a good knowledge of what  $s$  is going to be, then after that particular step you would arrive at the solution.

Therefore, add the solution  $F(X + s)$  is equal to 0. This implies 0 is equal to  $F(X) + \text{the Jacobian of } X \times s$  and if  $s$  is not the appropriate step. So, if this is the solution this is  $X$ . So, if the Jacobian approximation at this point makes you go over here then you hope that in the next iteration it will take you like this, then in the next iteration like this, then in the next iteration like this.

And we have we have seen this in python how this looks right. So, the step in order to evaluate what the step will be, we need to solve this particular linear equation right. We need to solve this equation and once this is solved we need to update what X is.

(Refer Slide Time: 05:27)



So, X will be simply updated by X + s. So, this is nothing but the update. So, this is the step finding and this is the update right. Let us consider a very simple example. So,  $y = \frac{1}{b} \exp(bx)$  right and  $x^2 + y^2 = 1$ . So, we need to solve these two equations simultaneously and we know how the solution will look like. How? Because we have seen this already that these two equations the root will correspond to the intersection of these two equations.

(Refer Slide Time: 06:25)

The image shows a slide with two parts. On the left, a code editor displays the following C code:

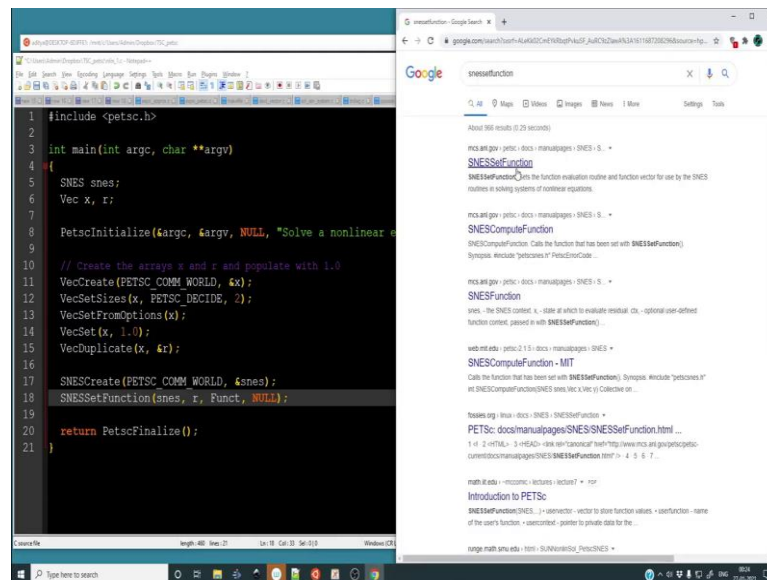
```
1 #include <petsc.h>
2
3 int main(int argc, char **argv)
4 {
5     SNES snes;
6     Vec x, r;
7
8     PetscInitialize();
9
10    return PetscFinalize();
11 }
```

On the right, a hand-drawn diagram illustrates the problem. It shows a coordinate system with a green circle centered at the origin, representing the equation  $x^2 + y^2 = 1$ . A purple curve, representing the exponential function  $y = \frac{1}{b} \exp(bx)$ , is plotted. The intersection point of the circle and the curve is marked with a purple dot and labeled  $-x.root$ . Handwritten equations include  $x_1 = \frac{1}{b} \exp(bx_0)$ ,  $x_0^2 - x_1^2 = 1$ , and the residual equation  $F(\vec{x}) = 0 \rightarrow \begin{bmatrix} \frac{1}{b} \exp(bx_0) - x_1 \\ x_0^2 - x_1^2 - 1 \end{bmatrix} = 0$ . The vector  $\vec{x}$  is defined as  $\vec{x} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$ . The residual is labeled  $\vec{r}$ .

So, let me draw an axis and let us draw the circle. So, this is a circle at the origin it is something like this right. So, we will poorly done circle, but you get the point and the exponential will be something like this right. So, obviously it will intersect somewhere along minus x as well some minus root, but it will intersect over here as well.

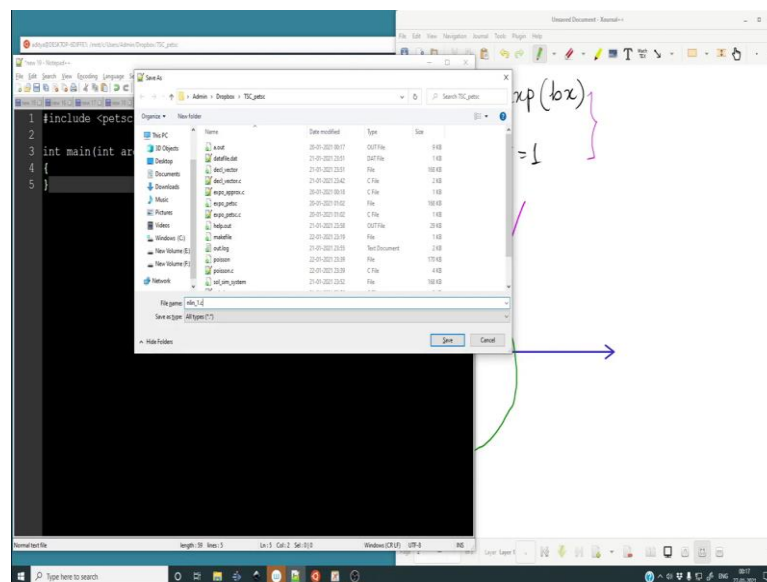
So, it is quite straight forward to graphically draw the root. But solving it algebraically is not possible and therefore you have to resort to these kind of iterative methods. So, what is the way we can achieve this in PETSc. So, let us start programming and I will explain the different terms as and when they arrive.

(Refer Slide Time: 07:48)



So, let us create a new file alright. So, first things first hash include petsc.h. Then int main argc int argc char star star argv.

(Refer Slide Time: 08:07)



Let us save this let's call this nlin 1 dot c alright. So, first things first we have to return PetscFinalize and we have to do an initialize. Apart from this we must create the object that will help us in solving a non-linear problem. In the case of the matrix solvers, we were using a KSP object. In this particular case we will be using SNES object.

So, simply we declare SNES snes like we did for KSP we will need two vectors one vector will be x and one vector will be y was r. So,  $\text{vec } x, r$ . So, what is x? x is a vector which consists of  $x_0$  and  $x_1$  which is a standard for x and y because we have a single array which will hold all the variables.

So, in that case this particular equation can be written as  $x_1 = 1 / b \exp(bx_0)$  this will be  $x_0^2 - x_1^2 = 1$ . And so let us try to write it in this particular form  $F(x) = 0$ . So, this particular form will be  $1 / b \exp(bx_0) - x_1$  and  $x_0^2 - x_1^2 - 1 = 0$ . So, this is that form that we spoke about ok.

So, let us continue. So, we need ok. So, I did not explain what r is r is the residual ok. Residual of these two equations. So, it is obviously an array ok. So, this is required by the SNES solver, because it provides us what residual has remained from solving this particular set of equations alright.

So, now that we have defined this we will do a PETScInitialize. So, let me copy the code because PETScInitialize looks the same. So, instead of this we will have solve a non-linear equation alright after this we will create the vectors that we will require for our computation.

So, `VecCreate PETSc COMM WORLD ampersand x`. So, address of x `VecSetSizes x comma PETSC DECIDE, 2`. So, we want it to be a size 2 then `VecSetFromOptions x` in case we pass command line arguments and finally `VecSet`. So, this is a new function that we are using `VecSet` is to declare that all the values in x will be set to 1.

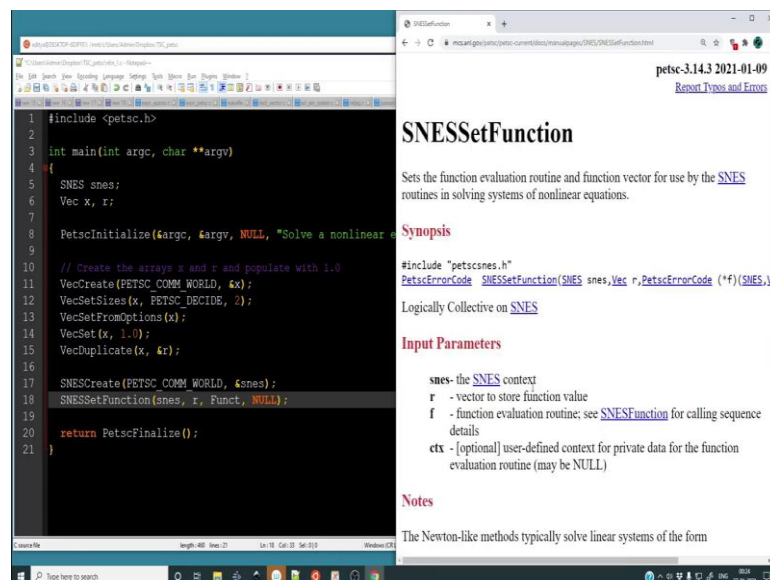
So this sets both the values to be 1 and why do we do this? Because we are going to start with an initial guess of 1, 1. If this is not required we can do the entire sequence of setting a vector like we had done in the previous example in creating a right hand side of the Poisson solver we had created the vector in this fashion.

We can do that well you have need to do all this, but you simply need to assign a value. But because we will start with the guess of 1, 1 it is convenient to use this particular function quickly. Finally let us duplicate x into r. So, `VecDuplicate x into r`. So, the address of r is passed. So, at this particular step we have created the arrays.

So, this alright. So, we have populated them with 1. Now we must create the SNES object it follows the exact same sequence that we had followed for KSP. So, first SNESCreate PETSC COMM WORLD we have to pass the address of snes. Then SNESSetFunction snes, residual, the function, NULL.

So, let me show the command reference of set function. So, what it does is it tells the SNES solver and whatever residual you have you store it in r and the function call back is Funct. So, we will create it have not yet created the function, we will create a function which will provide these two values. It is a simple function which will output a vector which will be the comprise of these two things.

(Refer Slide Time: 15:22)



So, alright ok well this one's ok. So, it has the SNES context, the residual storer, the function routine and the user define parameters that you want to pass, but in this case will not pass any user defined parameters ok.

(Refer Slide Time: 15:41)

The image shows a code editor on the left with the following C code:

```

1 #include <petsc.h>
2
3 int main(int argc, char **argv)
4 {
5     SNES snes;
6     Vec x, r;
7
8     PetscInitialize(&argc, &argv, NULL, "Solve a nonlinear equation");
9
10    // Create the arrays x and r and populate with 1.0
11    VecCreate(PETSC_COMM_WORLD, &x);
12    VecSetSizes(x, PETSC_DECIDE, 2);
13    VecSetFromOptions(x);
14    VecSet(x, 1.0);
15    VecDuplicate(x, &r);
16
17    SNESCreate(PETSC_COMM_WORLD, &snes);
18    SNESSetFunction(snes, r, Funct, NULL);
19
20    return PetscFinalize();
21 }

```

The browser window on the right shows the documentation for `SNESFunction` with the following content:

**Input Parameters**

- `snes` - the `SNES` context
- `r` - vector to store function value
- `f` - function evaluation routine; see `SNESFunction` for calling sequence details
- `ctx` - [optional] user-defined context for private data for the function evaluation routine (may be `NULL`)

**Notes**

The Newton-like methods typically solve linear systems of the form  $f'(x) x = -f(x)$ , where  $f(x)$  denotes the Jacobian matrix and  $f(x)$  is the function.

**See Also**

[SNESGetFunction\(\)](#), [SNESComputeFunction\(\)](#), [SNESSetJacobian\(\)](#), [SNESSetPicard\(\)](#), [SNESFunction](#)

**Level**

beginner

**Location**

So, that is it as it as easy as that. So, we have defined this once we have set the function we can actually go ahead and create the function right now.

(Refer Slide Time: 15:58)

The image shows a code editor on the left with the following C code:

```

1 #include <petsc.h>
2
3 PetscErrorCode Funct(SNES snes, Vec x, Vec F, void *ctx)
4 {
5     const double b = 2.0, *ax;
6     double *aF;
7
8     VecGetArrayRead(x, &ax);
9     VecGetArray(F, &aF);
10    aF[0] = 1.0/b * PetscExpReal(b*ax[0]) - ax[1];
11    aF[1] = ax[0]*ax[0] + ax[1]*ax[1] - 1;
12    VecRestoreArrayRead(x, &ax);
13    VecRestoreArray(x, &aF);
14
15    return 0;
16 }
17
18 int main(int argc, char **argv)
19 {
20     SNES snes;
21     Vec x, r;
22
23     PetscInitialize(&argc, &argv, NULL, "Solve a nonlinear equation");
24
25     // Create the arrays x and r and populate with 1.0
26     VecCreate(PETSC_COMM_WORLD, &x);

```

The hand-drawn diagram on the right illustrates the function and its residual. It shows a coordinate system with a curve  $F(\vec{x}) = 0$  and a point  $\vec{x}_0 = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$ . The residual is defined as  $\vec{r} = F(\vec{x}_0)$ . The diagram also shows the function  $F(\vec{x}) = \begin{pmatrix} \frac{1}{b} \exp(bx) - x_1 \\ x_0^2 + x_1^2 - 1 \end{pmatrix}$  and the initial guess  $\vec{x}_0 = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$  with the corresponding function values  $a(x_0) = x_0$  and  $a(x_1) = x_1$ .

Let us not wait for it. So, `PetscErrorCode Funct`. So, the `Funct` will sort of take as an input `SNES snes` the vector `x` the vector `F` will be sort of returned and the additional thing we need is the user context. So, in this case it will be void pointer to the user context alright.



So, the vector  $F$  is something which is going to hold the return value. So,  $Funct$  in this case is going to hold the return value. So, we have `const double b = 2.0`. So, by a `const double` we say that we are not even accidentally going to change the value of  $b$  alright.

So, apart from this we need a pointer to the  $ax$ ; because whatever so  $ax$  is the auxiliary array for the vector  $x$ . Remember for defining a vector in `petsc` in case we want to print everything or run something in a loop we need an auxiliary array like over here. We had a `Vec uexact`, but we needed an auxiliary array `auexact`. So, that is the same thing alright.

So, it is a constant array and the reason is we will not modify values of  $x$ . We simply want to modify values of  $F$  that is why we will declare it as a constant double pointer. If you want you cannot do this, but you run the risk of modifying  $x$  by mistake. Although you will feel I am not going to make that mistake, but you know you should know never underestimate your own sense of making mistakes alright.

So, we have constant double  $b$  equal to 0 and a pointer to  $ax$  which will hold the vectors for  $x$  alright. And we need a double star  $aF$  that is an auxiliary array to hold the elements of the `Petsc` object vector  $F$  alright. Then what do we need to do? We need to tell that  $F_0$  will be this thing and  $F_1$  will be this thing this is what we need to tell and  $x_0$  and  $x_1$  are in the `Vec x`.

So, we must take these two give them to  $ax$ . So, that  $ax_0$  will be  $x_0$  and  $ax_1$  will be  $x_1$  this is what we need to do. So, first things first we will read the vector  $x$  into the array  $ax$ . So, `VecGetArrayRead x ampersand ax`. So, why have I used `VecGetArrayRead` instead of simply `get vec array`. It is a safety interlock to say that  $ax$  is only a read only variable by virtue of being a `const double`.

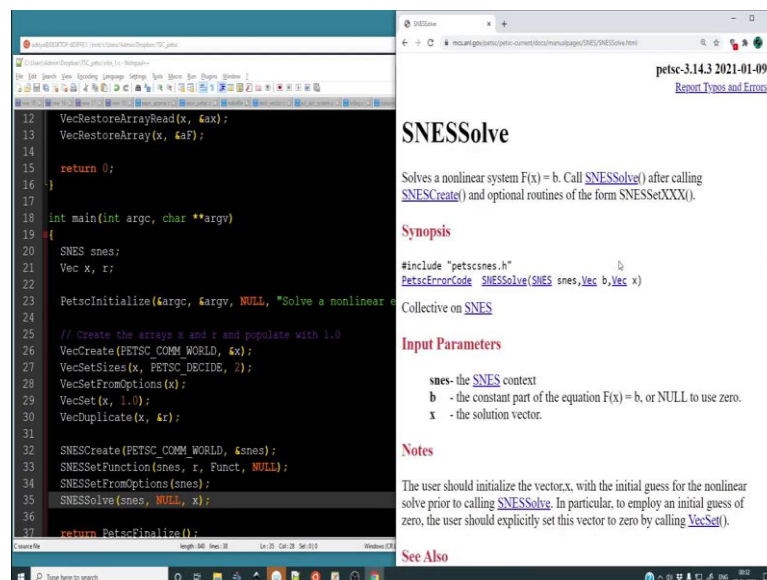
If it were to be only `double` we would have written only `get VecGetArray`, which we are going to do for  $f$  in fact, `VecGetArray F comma ampersand of aF`. So, these two extract the `PETSc` objects into `c` objects this is what it does and we have discussed this in the previous lectures as well alright. So, once we have this we are going to define  $aF_0$  as  $1$  over  $b$ . So,  $1.0$  over  $b$  times `exp` of `PetscExpReal` then  $b$  times  $ax$  naught minus  $ax_1$ .

Similarly,  $aF_1 = ax_0 \times ax_0 - ax_1 \times ax_1$  not - this should be + I think I wrote the equation the circle incorrectly well. This is where the mistake was it is actually plus this

should be a + this should be a + alright. So, aF 1 is this now we simply need to restore the c object aF into the PETSc of vector F just to complete the entire sequence.

So, Vec VecRestoreArrayRead x comma address of ax and then VecRestoreArray x comma address of aF and we will return 0 if everything is successful. So, this is how we will create the code and if you feel like you want to modify these things to make your own solver the code will be available on the website you can have a go ahead alright. So, at this particular step we have created the function the function call back and the residual so all that we have passed.

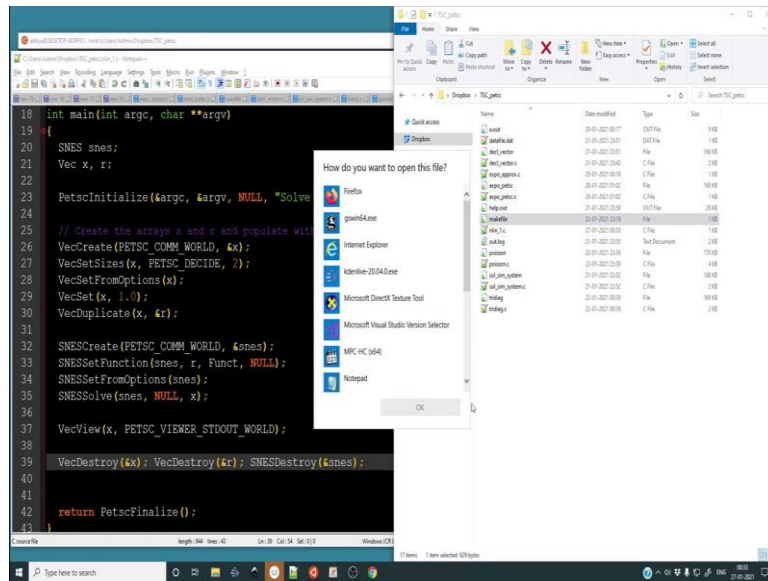
(Refer Slide Time: 22:53)



So, once we have set the function we need to tell SNES that in case I pass some command line arguments it should set the SNES object accordingly. So, SetFromOptions snes and we will use this. Lastly we need to solve, SNESolve snes NULL x. So, let me show you the function reference. There appears to be some internet issue ok alright.

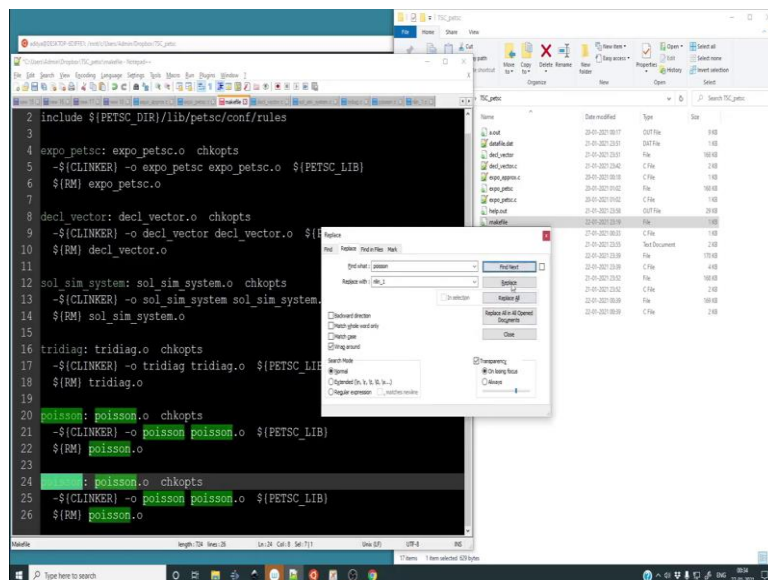
So, if  $F(x) = b$  then we need to pass b, but because we have cast our equation in the form of  $x = 0$ . We pass a as a NULL and x is the solution vector. So, we finally get x as the solution vector. So, what do we do once we have a solution vector? Obviously we are going to have a look at the solution vector.

(Refer Slide Time: 24:00)



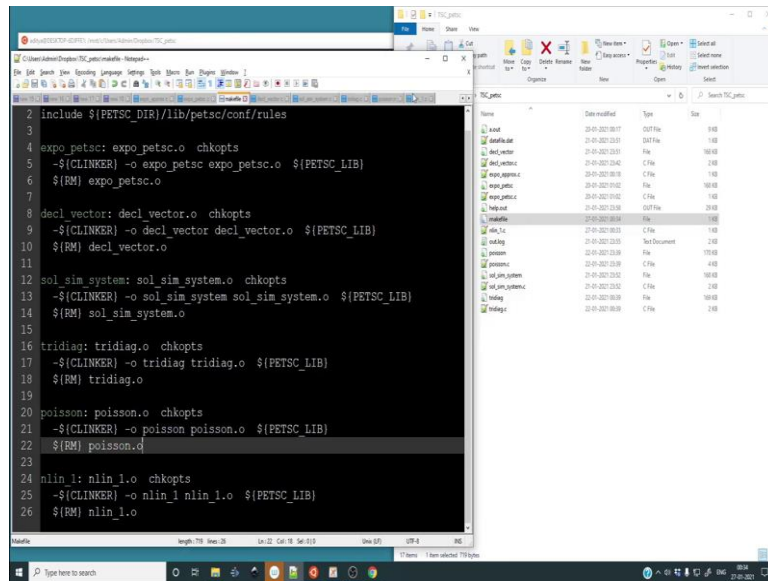
So, we are going to do VecView x PETSC VIEWER STDOUT WORLD. Finally, we must destroy all the variables that we have created. So, VecDestroy x; VecDestroy r we have to pass the addresses not just the. So, we have to destroy the memory addresses you get the point the point is an SNESDestroy ampersand snes. So, all the addresses are passed so that they destroy alright. So, that takes care of this little program. So, let me go ahead and change or change create a new target in our make file.

(Refer Slide Time: 25:09)



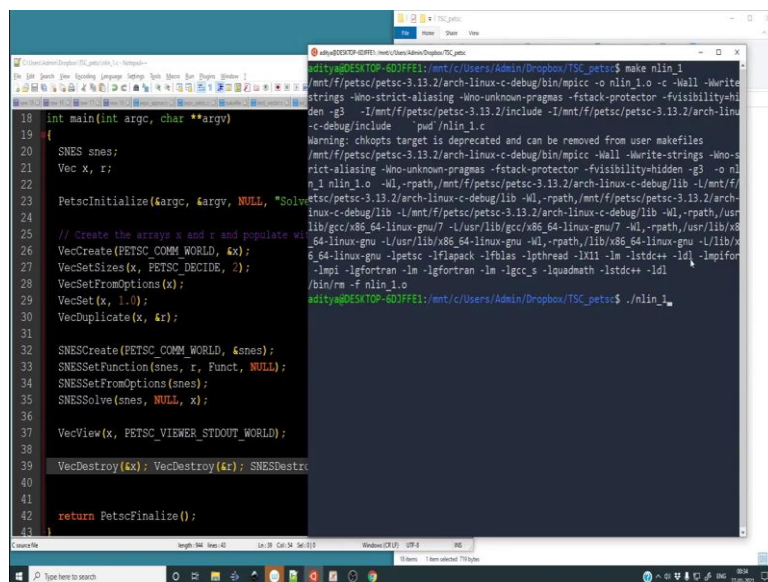
So, we will make a new target we will call it nlin 1. So, control H nlin 1 poisson nlin 1.

(Refer Slide Time: 25:36)



So, my next point [vocalized- noise] my next replace, replace, replace, replace it's as easy as that no hassle ok.

(Refer Slide Time: 25:42)



So, we have our terminal open. So, let us make the target. So, make and then 1 everything is fine. So, .\nlin\_1.

(Refer Slide Time: 26:05)

```
18 int main(int argc, char **argv)
19 {
20     SNES snes;
21     Vec x, r;
22
23     PetscInitialize(&argc, &argv, NULL, "S
24
25     // Create the arrays x and r and populate
26     VecCreate(PETSC_COMM_WORLD, &x);
27     VecSetSizes(x, PETSC_DECIDE, 2);
28     VecSetFromOptions(x);
29     VecSet(x, 1.0);
30     VecDuplicate(x, &r);
31
32     SNESCreate(PETSC_COMM_WORLD, &snes);
33     SNESSetFunction(snes, r, Funct, NULL);
34     SNESSetFromOptions(snes);
35     SNESolve(snes, NULL, x);
36
37     VecView(x, PETSC_VIEWER_STDOUT_WORLD);
38
39     VecDestroy(&x); VecDestroy(&r); SNESDestro
40
41     return PetscFinalize();
42 }
43 }
```

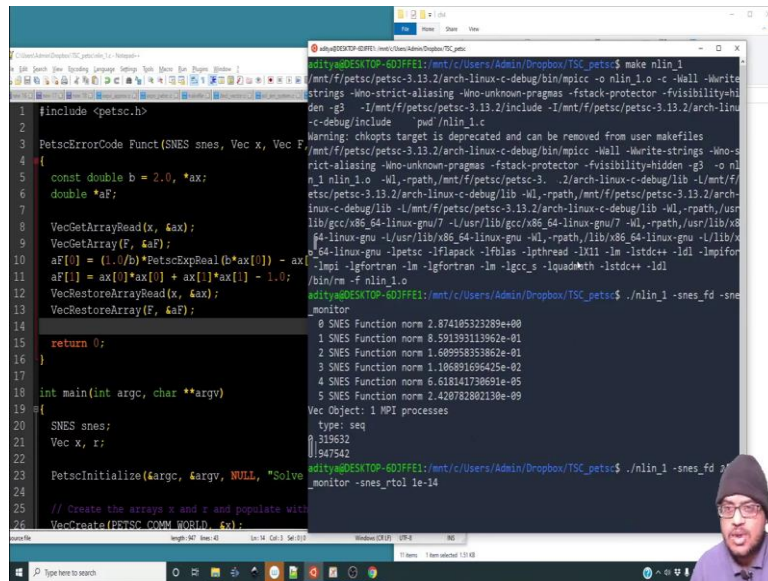
```
[0]PETSC ERROR: [0] MatSetColumnIJ line 7435 /mnt/f/petsc/petsc-3.13.2/src/mat/int
face/matrix.c
[0]PETSC ERROR: [0] MatColoringApply_S1 line 60 /mnt/f/petsc/petsc-3.13.2/src/mat/a
lor/impls/minpack/color.c
[0]PETSC ERROR: [0] MatColoringApply line 354 /mnt/f/petsc/petsc-3.13.2/src/mat/coe
r/interface/matcoloring.c
[0]PETSC ERROR: [0] SNESComputeJacobianDefaultColor line 65 /mnt/f/petsc/petsc-3.1c
2/src/snes/interface/snes2.c
[0]PETSC ERROR: [0] SNES user Jacobian function line 2715 /mnt/f/petsc/petsc-3.13.1
an line 2674 /mnt/f/petsc/petsc-3.13.2/src/sn
S line 144 /mnt/f/petsc/petsc-3.13.2/src/snes2
/83 /mnt/f/petsc/petsc-3.13.2/src/snes/interfa
Error Message -----/
n.gov/petsc/documentation/faq.html for troub
3.13.2, Jun 02, 2020
[0]PETSC ERROR: [0] MPI_Abort on a arch-linux-c-debug named DESKTOP-6D07FE1 by aditya W
Jan 27 00:34:52 2021
[0]PETSC ERROR: Configure options --with-cc-gcc --with-cxx-g++ --with-fc=gfortran
download-mpich --download-fblaslapack
[0]PETSC ERROR: #1 User provided function() line 0 in unknown file
application called MPI_Abort(MPI_COMM_WORLD, 50152859) - process 0
[unset]: write_line error; fd=-1 buf=c:cmd-abort exitcode=50152859
system msg for write_line failure : Bad file descriptor
aditya@DESKTOP-6D07FE1:~/mnt/c/Users/Admin/Dropbox/TSC_petsc$
```

(Refer Slide Time: 26:09)

```
2/src/snes/interface/snes2.c
[0]PETSC ERROR: [0] SNES user Jacobian function line 2715 /mnt/f/petsc/petsc-3.13.
src/snes/interface/snes.c
[0]PETSC ERROR: [0] SNESComputeJacobian line 2674 /mnt/f/petsc/petsc-3.13.2/src/sn
/interface/snes.c
[0]PETSC ERROR: [0] SNESolve_NEWTONLS line 144 /mnt/f/petsc/petsc-3.13.2/src/snes
mpls/lsls.c
[0]PETSC ERROR: [0] SNESolve line 4403 /mnt/f/petsc/petsc-3.13.2/src/snes/interfa
/snes.c
[0]PETSC ERROR: [0] Signal received
[0]PETSC ERROR: See https://www.mcs.anl.gov/petsc/documentation/faq.html for troub
shooting.
[0]PETSC ERROR: Petsc Release Version 3.13.2, Jun 02, 2020
[0]PETSC ERROR: ./lin_1 on a arch-linux-c-debug named DESKTOP-6D07FE1 by aditya W
Jan 27 00:34:52 2021
[0]PETSC ERROR: Configure options --with-cc-gcc --with-cxx-g++ --with-fc=gfortran
download-mpich --download-fblaslapack
[0]PETSC ERROR: #1 User provided function() line 0 in unknown file
application called MPI_Abort(MPI_COMM_WORLD, 50152859) - process 0
[unset]: write_line error; fd=-1 buf=c:cmd-abort exitcode=50152859
system msg for write_line failure : Bad file descriptor
aditya@DESKTOP-6D07FE1:~/mnt/c/Users/Admin/Dropbox/TSC_petsc$ ./lin_1 -snes_fd -sn
_monitor
0 SNES Function norm 2.874185323289e+00
1 SNES Function norm 8.591393113962e-01
Vec Object: 1 MPI processes
type: seq
0.619203
0.888797
aditya@DESKTOP-6D07FE1:~/mnt/c/Users/Admin/Dropbox/TSC_petsc$
```

This will not run there is a bunch of errors the reason is we are not told how to make the Jacobian. So, if you notice in this entire code we do not have a Jacobian. So, -snes fd. So, it says that it has to create the Jacobian using a finite difference approximation and it will do that automatically you do not need to worry about it. And finally lets have an snes monitor great. So, after just 2 iterations we have convergence what wait? It does not look entirely correct well let us check our function callback.

(Refer Slide Time: 26:59)



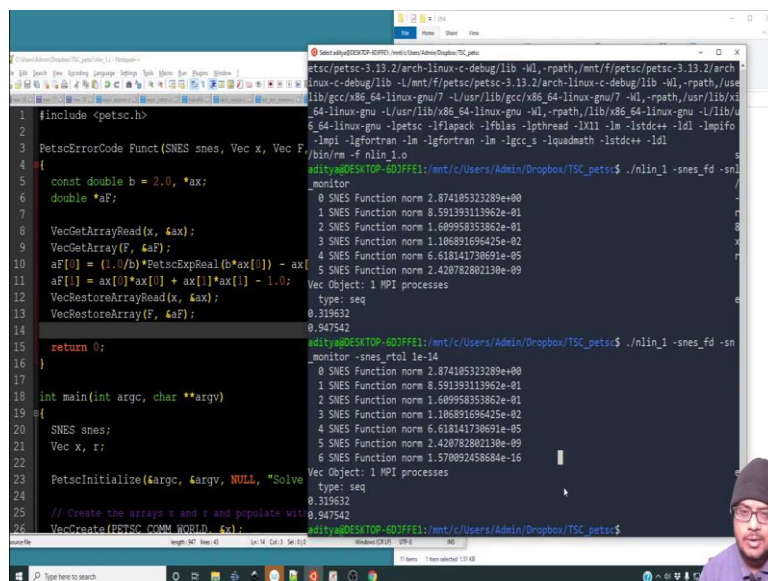
```
#include <petsc.h>
2
3 PetscErrorCode Funct(SNES snes, Vec x, Vec F)
4 {
5     const double b = 2.0, *ax;
6     double *aF;
7
8     VecGetArrayRead(x, &ax);
9     VecGetArray(F, &aF);
10    aF[0] = (1.0/b)*PetscExpReal(b*ax[0]) - ax[0];
11    aF[1] = ax[0]*ax[0] + ax[1]*ax[1] - 1.0;
12    VecRestoreArrayRead(x, &ax);
13    VecRestoreArray(F, &aF);
14
15    return 0;
16 }
17
18 int main(int argc, char **argv)
19 {
20     SNES snes;
21     Vec x, r;
22
23     PetscInitialize(&argc, &argv, NULL, "Solve
24     // Create the arrays x and r and populate with
25     VecCreate(PETSC_COMM_WORLD, &x);
26
```

```
aditya@DESKTOP-6DJFFE1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ make nlin_1
Warning: chkopts target is deprecated and can be removed from user makefiles
aditya@DESKTOP-6DJFFE1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./nlin_1 -snes_monitor -snes_rtol 1e-14
0 SNES Function norm 2.874195323289e+08
1 SNES Function norm 0.591393113962e-01
2 SNES Function norm 1.689958333862e-01
3 SNES Function norm 1.186891696425e-02
4 SNES Function norm 6.618141730691e-05
5 SNES Function norm 2.420782802130e-09
Vec Object: 1 MPI processes
type: seq
0.319632
0.947542
aditya@DESKTOP-6DJFFE1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./nlin_1 -snes_monitor -snes_rtol 1e-14
```

Alright guys I think there was a small error this should have been F. Now we are in a position to make because earlier it was only iterating for 1 iteration and I thought ok that is quite odd it cannot just converge in 1 iteration.

So, let me make it again. So, now it should work fine. So, `./nlin_1 -snes_monitor` ok. So, at the end of 5 iterations we do have convergence to the solution. So, what we can do is we can alternately have `-snes_rtol 1e-14` high degree of convergence.

(Refer Slide Time: 27:55)



```
#include <petsc.h>
2
3 PetscErrorCode Funct(SNES snes, Vec x, Vec F)
4 {
5     const double b = 2.0, *ax;
6     double *aF;
7
8     VecGetArrayRead(x, &ax);
9     VecGetArray(F, &aF);
10    aF[0] = (1.0/b)*PetscExpReal(b*ax[0]) - ax[0];
11    aF[1] = ax[0]*ax[0] + ax[1]*ax[1] - 1.0;
12    VecRestoreArrayRead(x, &ax);
13    VecRestoreArray(F, &aF);
14
15    return 0;
16 }
17
18 int main(int argc, char **argv)
19 {
20     SNES snes;
21     Vec x, r;
22
23     PetscInitialize(&argc, &argv, NULL, "Solve
24     // Create the arrays x and r and populate with
25     VecCreate(PETSC_COMM_WORLD, &x);
26
```

```
aditya@DESKTOP-6DJFFE1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ make nlin_1 -snes_monitor -snes_rtol 1e-14
aditya@DESKTOP-6DJFFE1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./nlin_1 -snes_monitor -snes_rtol 1e-14
0 SNES Function norm 2.874195323289e+08
1 SNES Function norm 0.591393113962e-01
2 SNES Function norm 1.689958333862e-01
3 SNES Function norm 1.186891696425e-02
4 SNES Function norm 6.618141730691e-05
5 SNES Function norm 2.420782802130e-09
6 SNES Function norm 1.578092458684e-16
Vec Object: 1 MPI processes
type: seq
0.319632
0.947542
aditya@DESKTOP-6DJFFE1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$
```

So, let us run it. So, it took 1 more iteration to converge to  $10^{-16}$ . So, once you are near a root you sort of have a rapid convergence towards the root, but far away from the root there is no guarantee you will converge quickly towards. In fact, the direction where convergence will happen ok.

So, let me conclude this lecture over here in this particular lecture, to summarize we have looked at how we can solve a non-linear solver using how we can solve a non-linear equation using the SNES object in PETSc. Next lecture we are going to look at how to pass a Jacobian and we will try to do a problem in of a non-linear reaction using PETSc until then its goodbye.