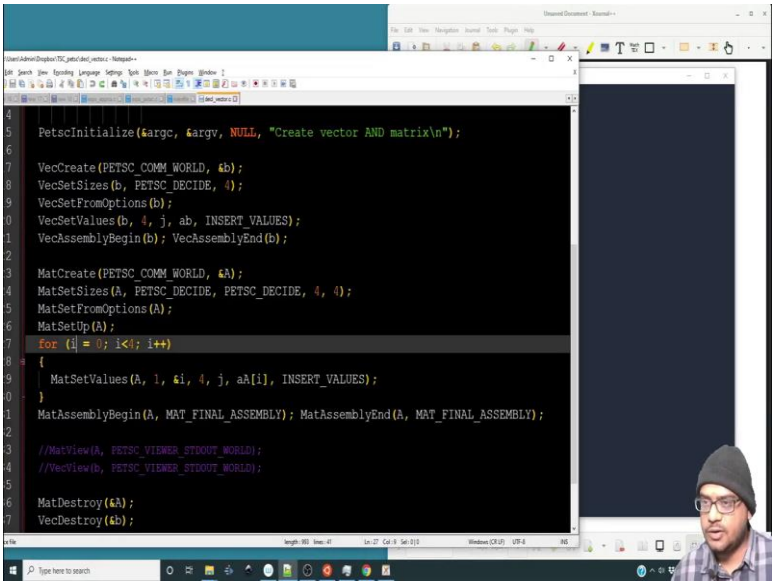
Tools in Scientific Computing
Prof. Aditya Bandopadhyay
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

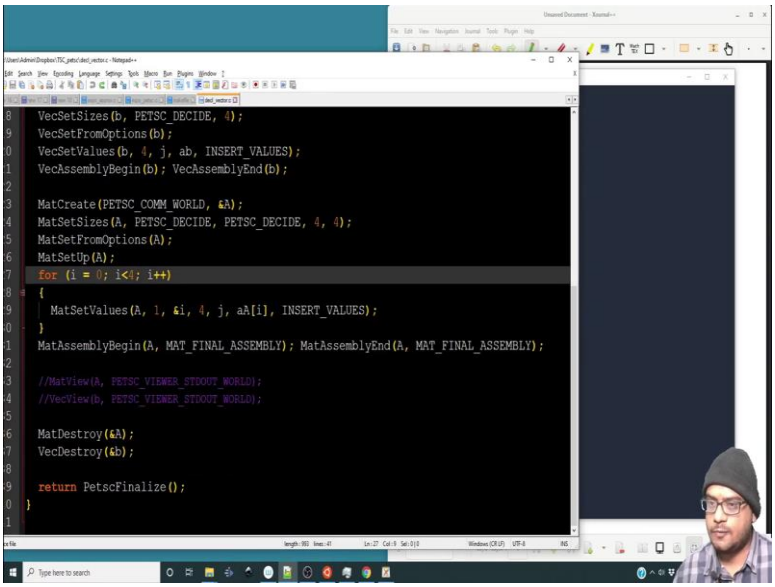
Lecture - 31
KSP object and solving a system

(Refer Slide Time: 00:27)



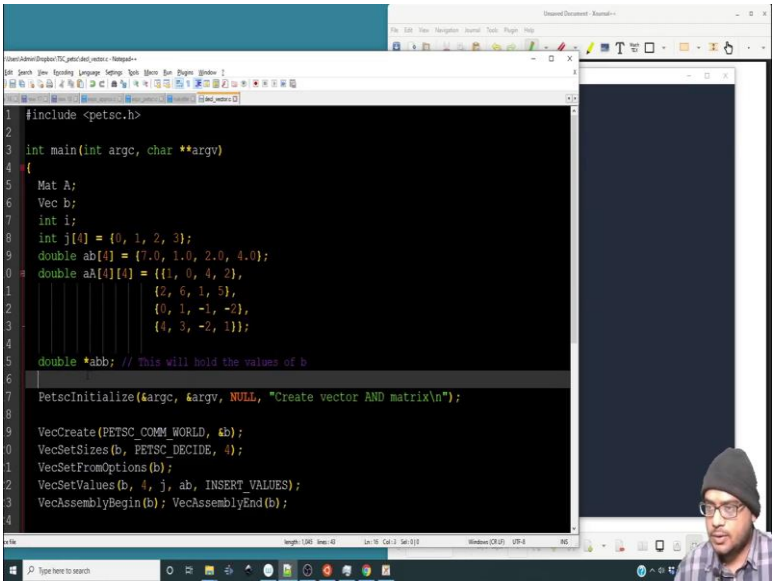
Last class, we saw how to create a vector and a matrix.

(Refer Slide Time: 00:33)



And in this particular video we are going to start off by looking at how we can print values to a command line and effectively to a file as well.

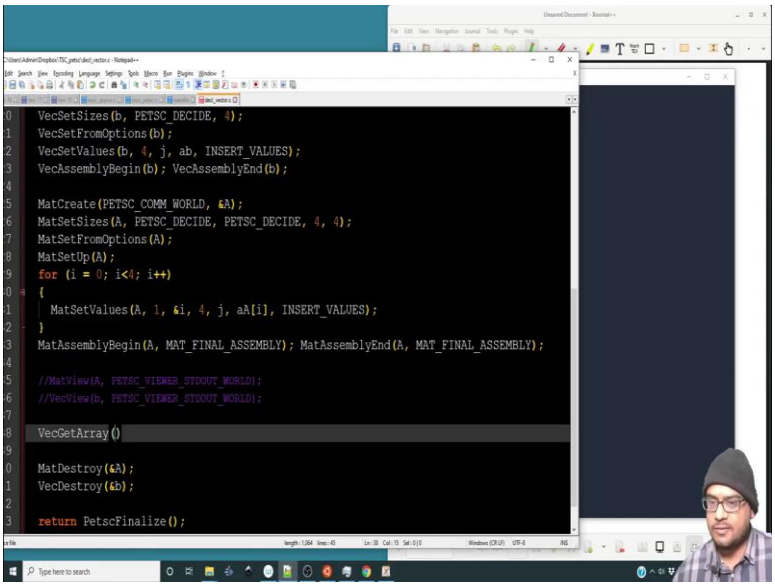
(Refer Slide Time: 00:51)



So, for that we have to first create an array. So, suppose we want to print the vector. So, suppose we want to print the vector like this I mean of course, you can simply loop over the elements of ab, but that is not useful in this particular context because, once you start programmatically declaring your arrays you would not be able to have such an easy access.

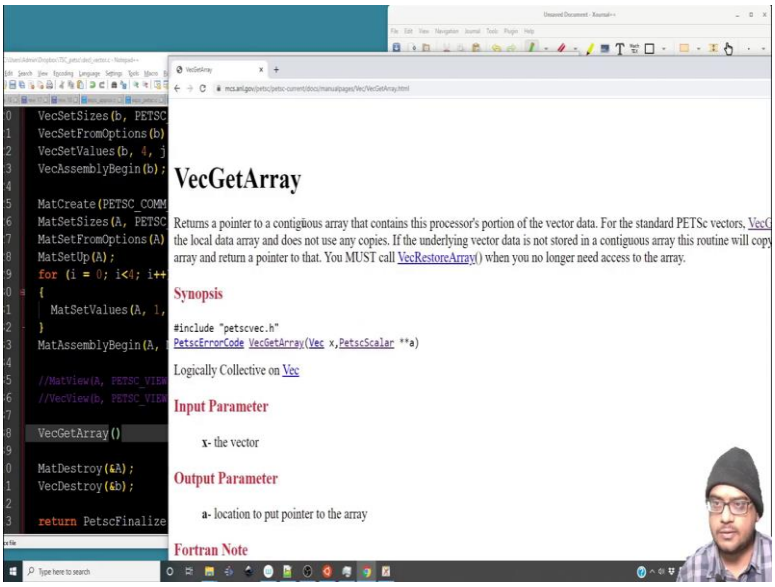
So, what you will have is the Vec object b and eventually you want to print the elements of b alright. So, let us create an array. So, double star abb. So, I am using abb because ab is already used. So, essentially this holds rather this will hold the value of b alright. So, we have declared double abb. So, before destroying we have to write certain lines.

(Refer Slide Time: 02:07)



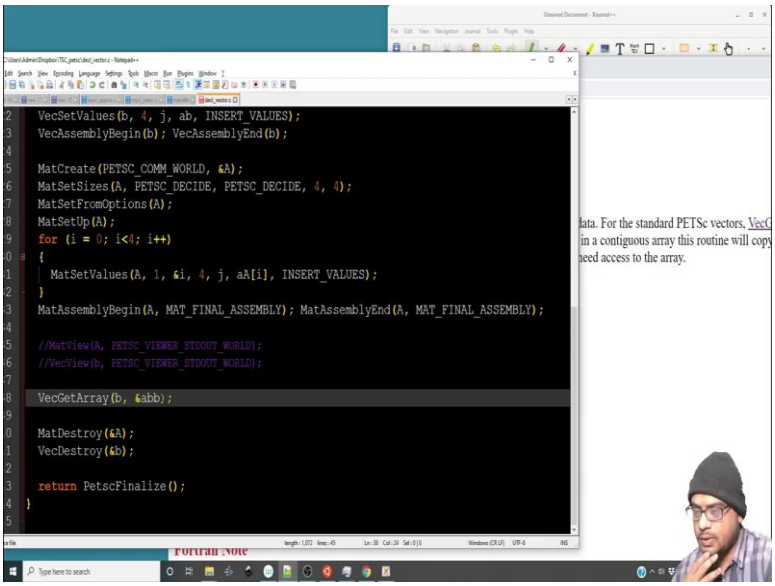
So, we will do VecGetArray. So, VecGetArray is a function.

(Refer Slide Time: 02:20)



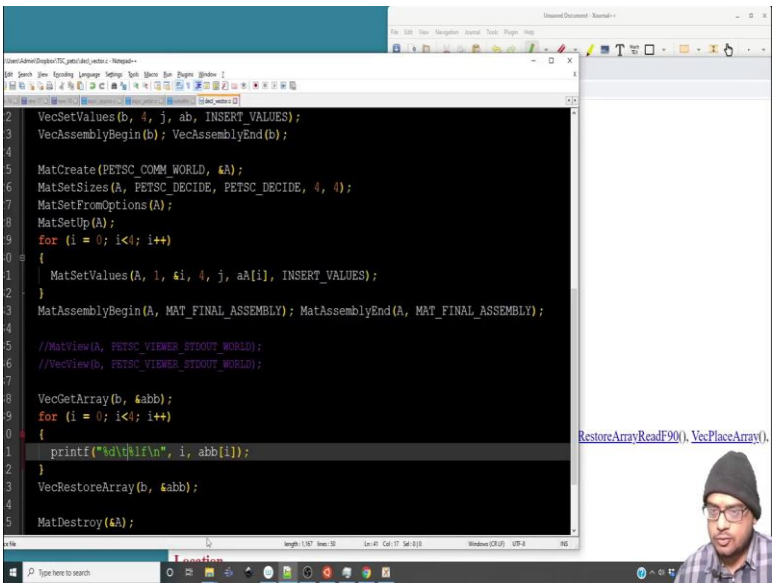
So, let me just take you to the functional reference. So, VecGetArray returns an array which contains the processors portion of the vector data ok. So, VecGetArray requires the vector object x and the Petsc double star a. So, because we have declared it as this we have to pass the address of abb rather than simply passing abb.

(Refer Slide Time: 02:43)



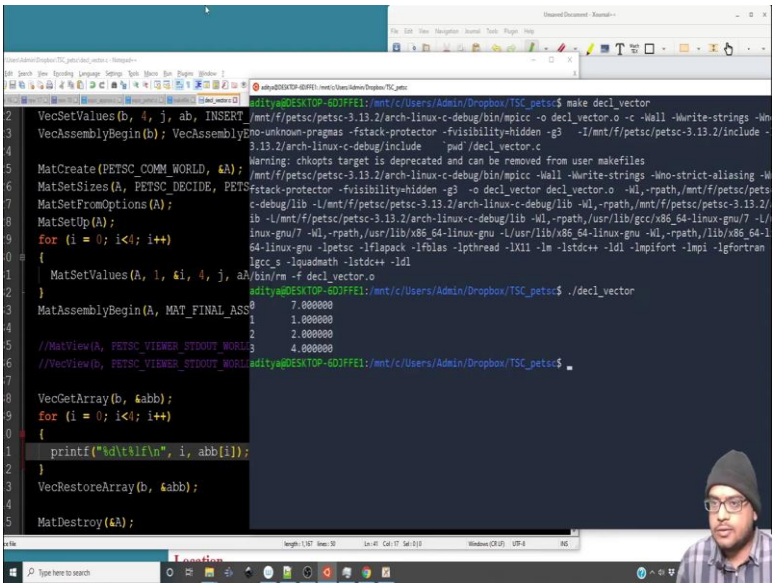
So, we must pass b and the address of abb. So, another useful thing is You MUST call VecRestoreArray when you no longer need access to the array. So, PETSc tries to keep everything in it is own data structure by calling this your actually transferring the vec object to a simple c array abb. So, we must also follow up by the VecRestoreArray it has the same syntax.

(Refer Slide Time: 03:19)



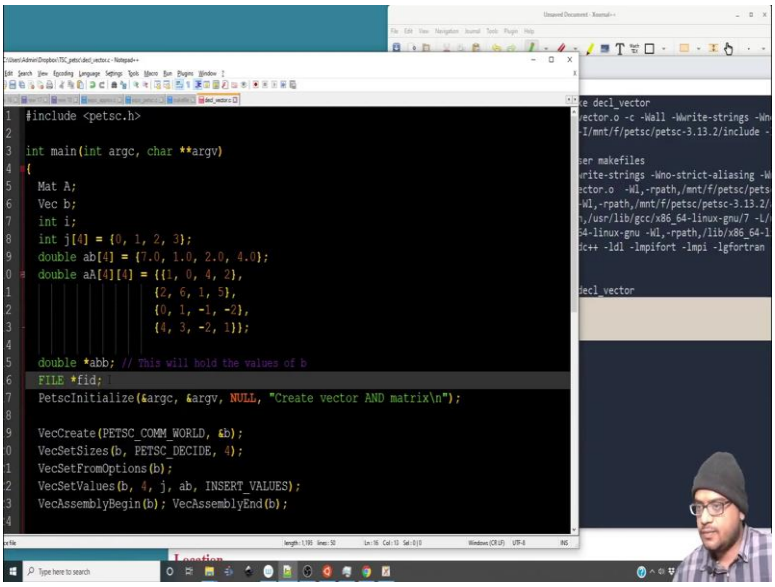
So, let us do that. So, it is simply restore ok. So, this will now help us to print the values of abb. So, what we will do is for i = 0, i < 4, i ++ printf % lf \ n and we will print abb i. In fact, let us print i as well. So, let us just put a % d like \ t alright.

(Refer Slide Time: 04:06)



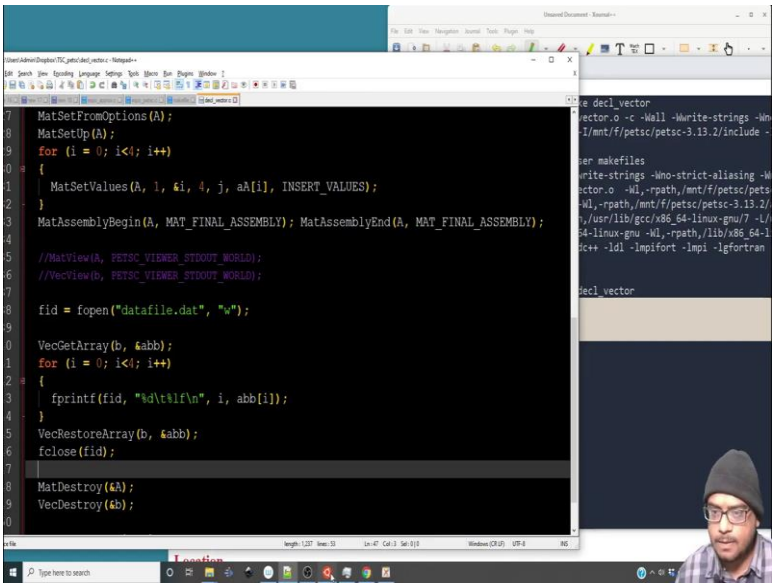
So, let us compile this. So, the make file will be the make target will be make declare vector alright dot slash declare vector ok. So, we could finally print this and at the end of restore it d allocates abb. So, once this line is executed we cannot do this anymore.

(Refer Slide Time: 04:41)



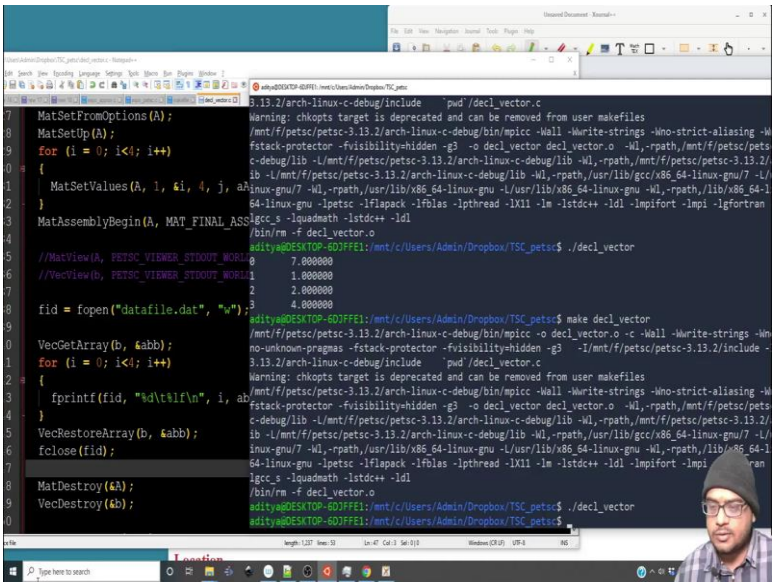
So, now you can imagine instead of having a print f you could alternately do an fprintf. So, there is an fid and then you simply fclose fid. So, for that we have to declare FILE star fid then.

(Refer Slide Time: 05:03)

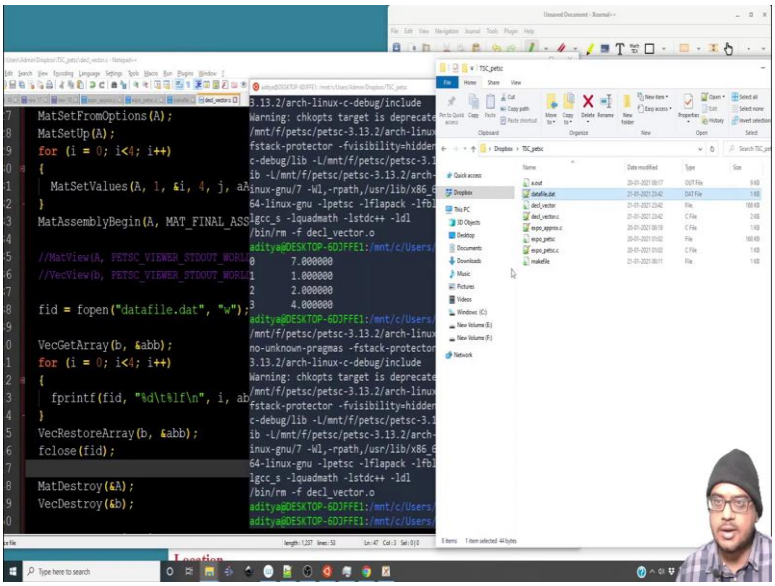


So, it is a pointer to a file then fid equal to fopen datafile dot dat right and yeah that is pretty much it. So, this should allow us to write the data to a file and then close the file as well let us see; let us see what happens.

(Refer Slide Time: 05:27)

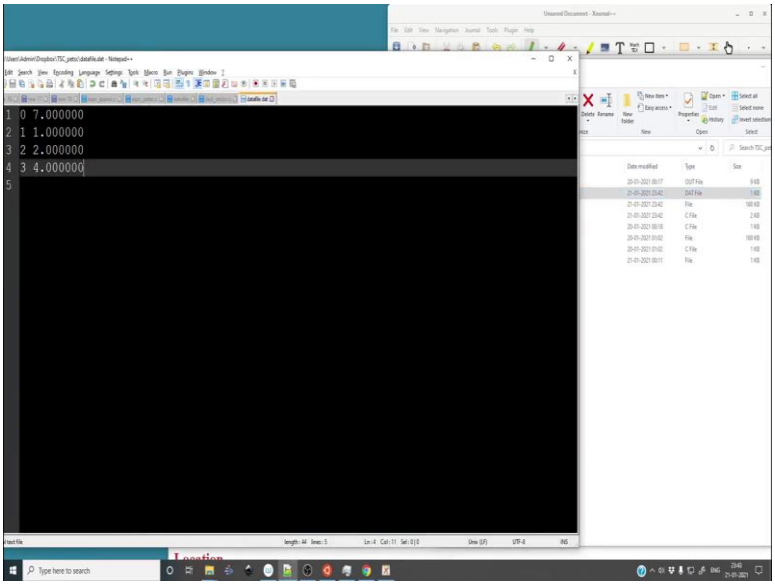


(Refer Slide Time: 05:35)



So, let us go to the folder and see whether we have something datafile dot dat, let me why is it not opening.

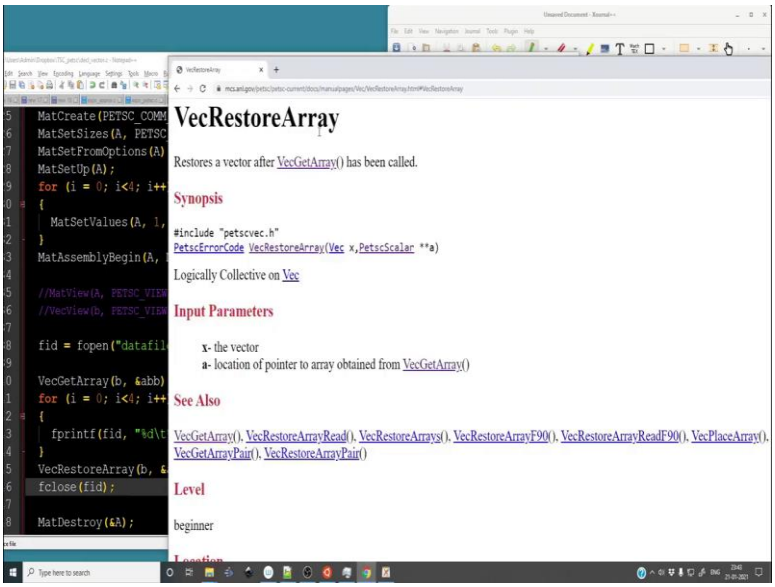
(Refer Slide Time: 05:46)



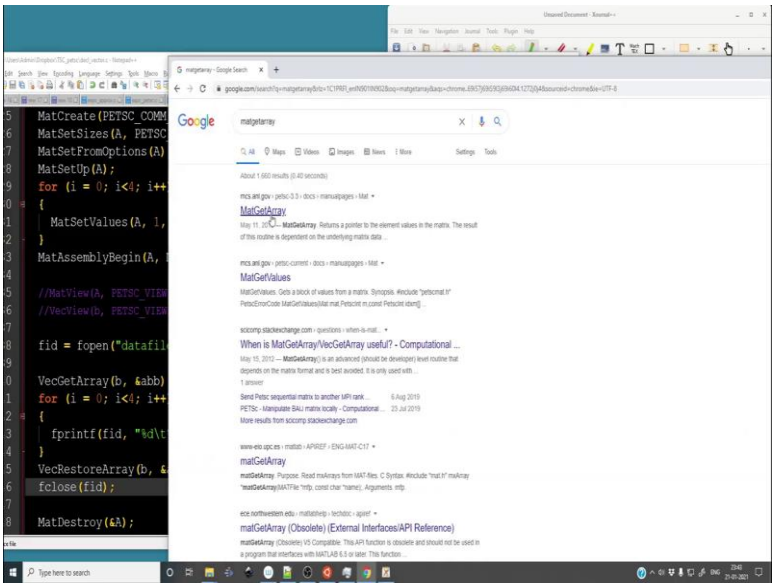
Great. So, we do have the data inside the database. So, it is quite useful when you want to. So, suppose you want to post process data once you have this you can then plot it using Python or glue plot anything you like it is always useful to have the data file when you are working with it.

So, this is a small snippet and I am not going to discuss more about this we will use it when we will need to ok. So, it is quite useful a versatile bit of snippet which you will find useful time to time.

(Refer Slide Time: 06:31)

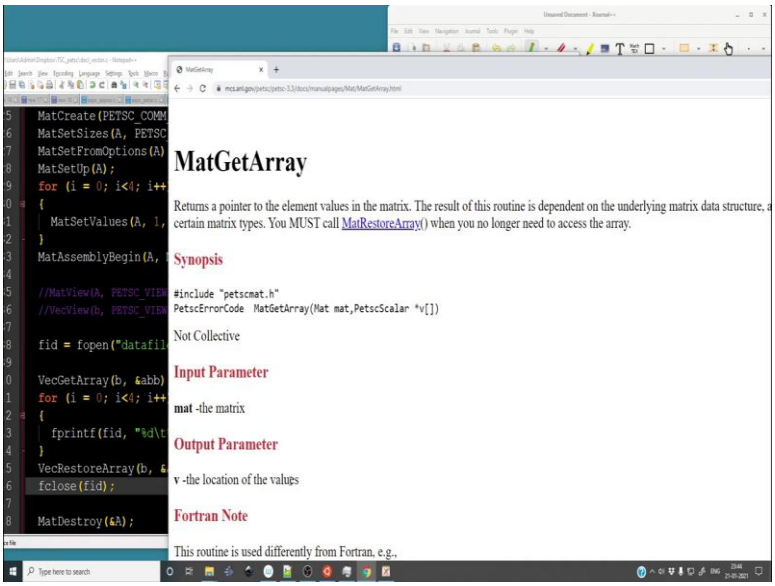


(Refer Slide Time: 06:33)



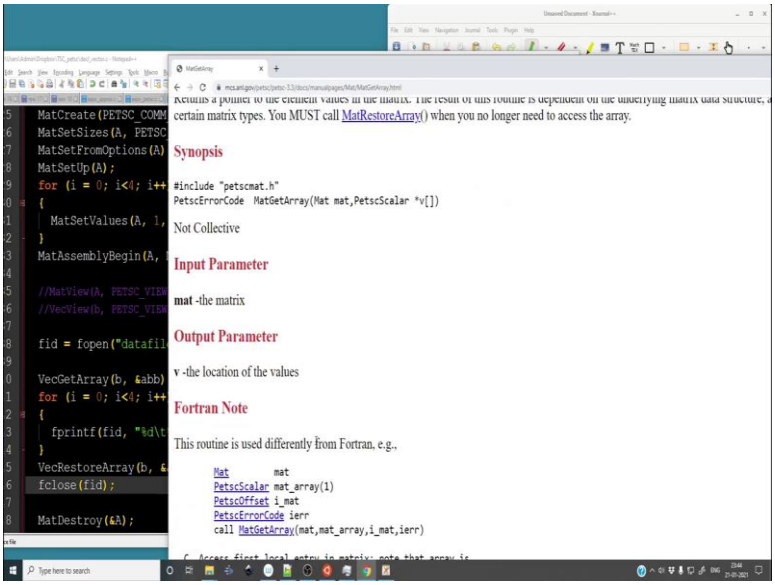
So, similarly similar to the VecGetArray there is also mat get rather MatGetArray.

(Refer Slide Time: 06:37)



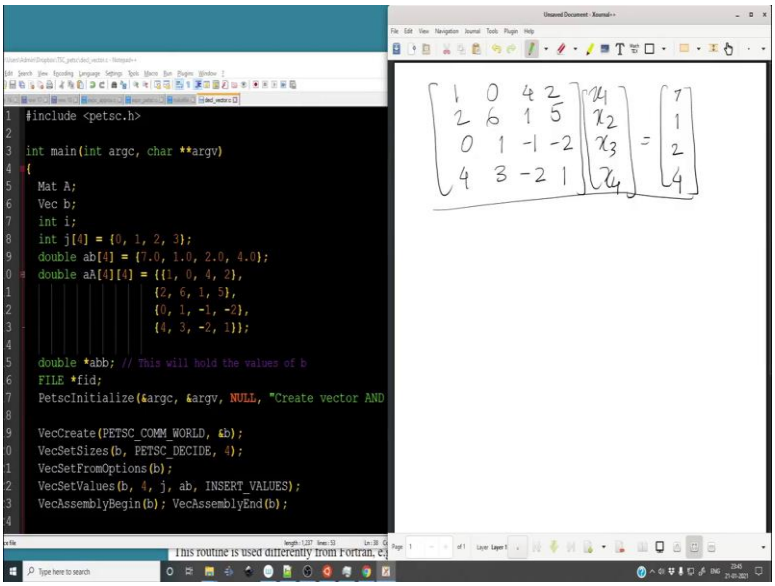
So, it does the same thing right instead it has to be passed with Mat and PetscScalar the pointer to v ok the location of the values.

(Refer Slide Time: 06:48)



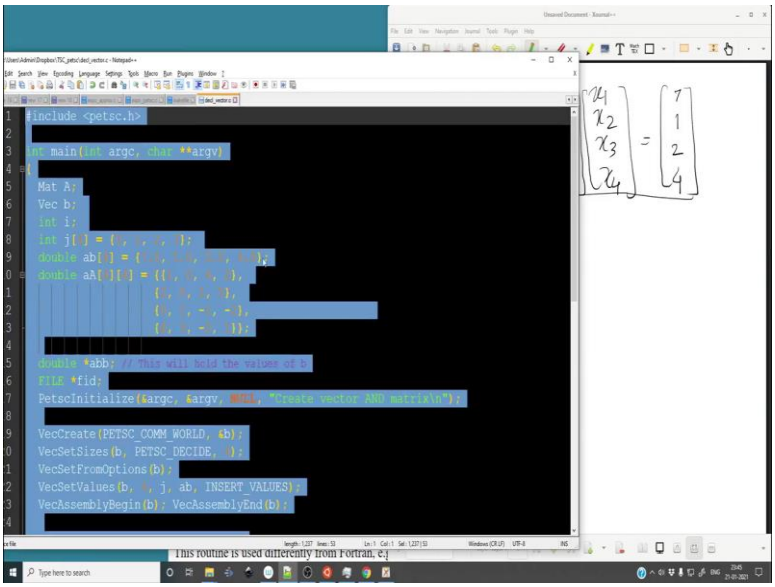
So, we will and it has to be also called with MatRestoreArray. So, we will do it whenever we will require it ok. So, do not worry about that ok. So, let us move on to solving this particular set of equations. So, what do we have? We want to solve we want to solve this and see here.

(Refer Slide Time: 07:10)

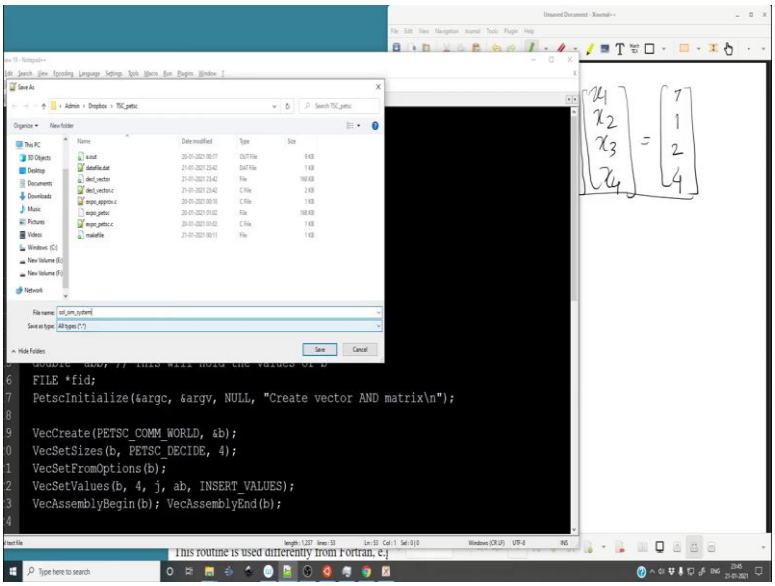


So, let us see how we can solve. So, it is not a very formidable set of algebraic equations it is rather easy to solve it by hand as well, but it is a bit time consuming unless you are a mathematical sound, but anyway. So, we will use.

(Refer Slide Time: 08:02)

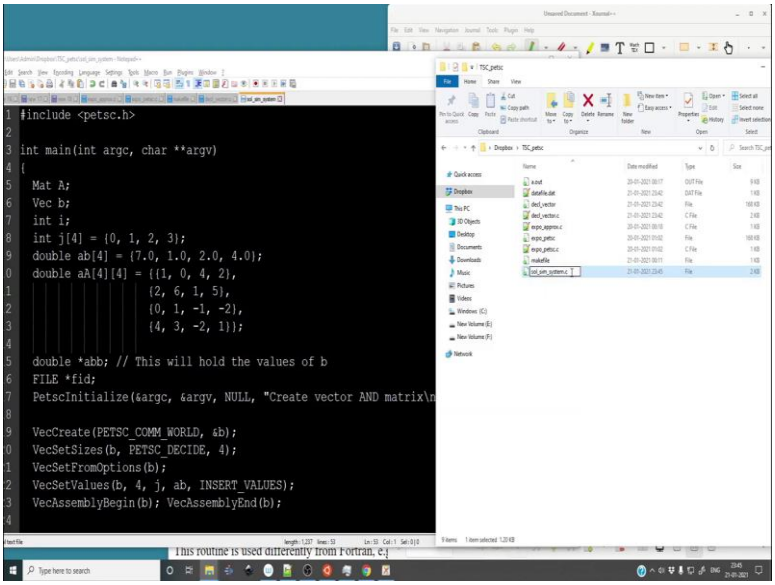


(Refer Slide Time: 08:07)



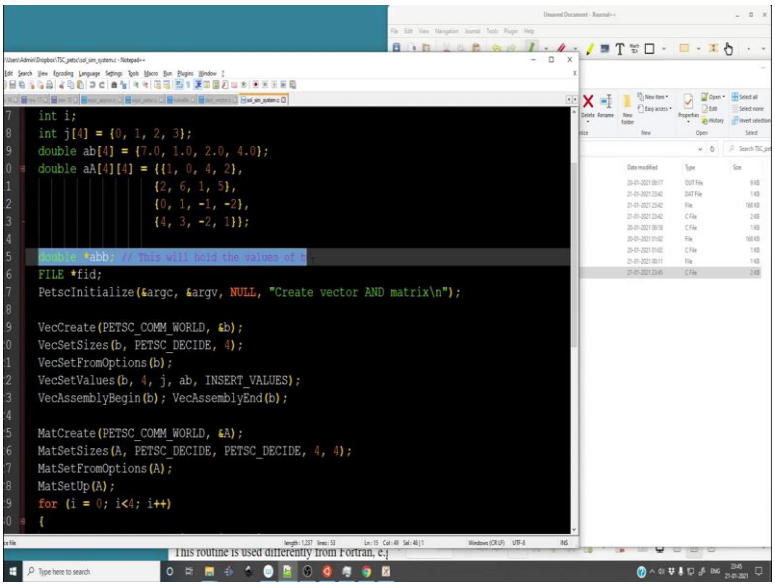
So, let us copy this code let us not. So, let me save it we will write it as solve simple sol sim system.

(Refer Slide Time: 08:25)

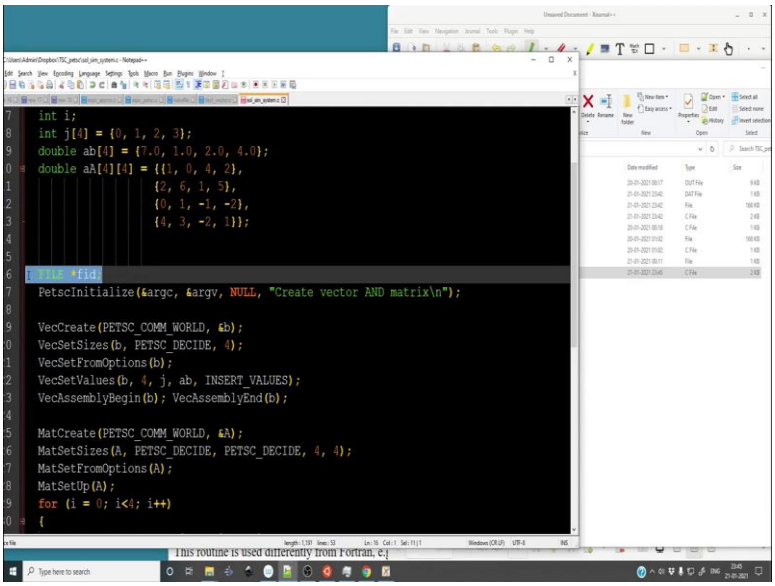


I forgot to put the extension let me just go to the folder and rename it dot c alright yes sorry alright.

(Refer Slide Time: 08:41)

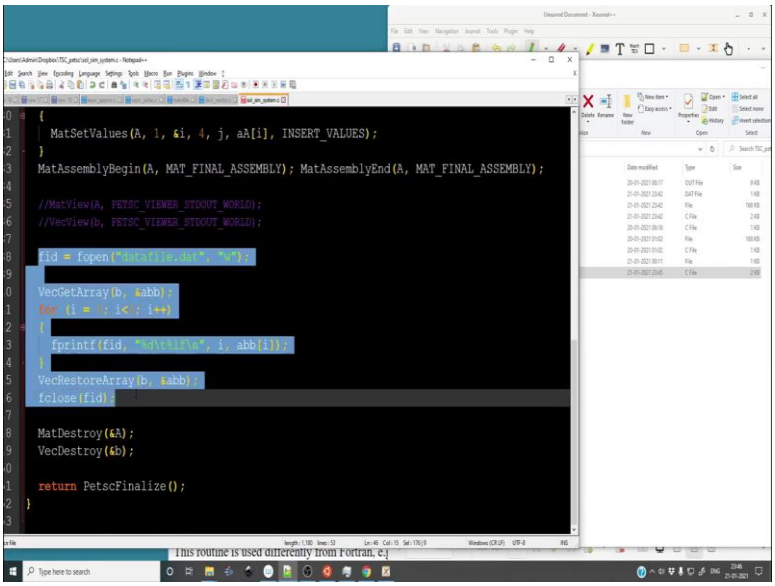


(Refer Slide Time: 08:45)



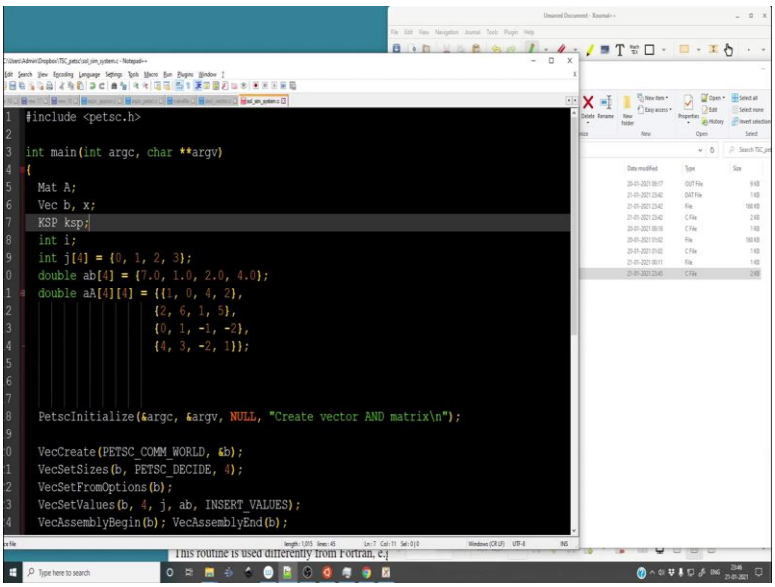
So, let us remove this abb business you do not need it for now let us remove the file id.

(Refer Slide Time: 08:51)



Let us remove this bit of program we do not need it I mean if at all you need it, you can always rewrite that bit of code.

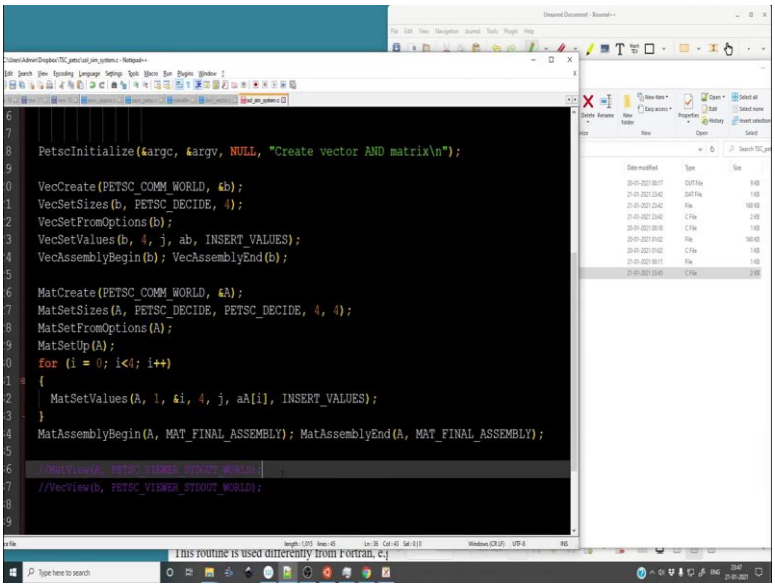
(Refer Slide Time: 08:57)



So, now we want to have another vector x which will store the solution alright and we build the array we build the vector. So, now we need to have an object which will help us actually solve the thing. So, in PETSc the solvers are a part of the object KSP ok.

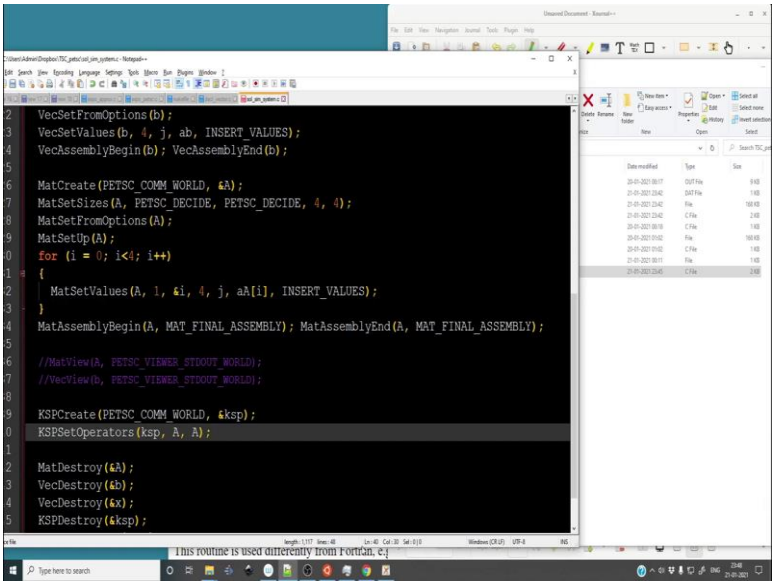
So, we will declare it as KSP this is the data type and we will call it small ksp. So, ksp is the solver object solver object will operate on the vector the matrix A and the vector b to yield the solution x that is how it works.

(Refer Slide Time: 09:46)



So, similar to the matrix or the vector creation. So, this is the vector creation this is the matrix creation there has to be also a solver creation.

(Refer Slide Time: 09:59)

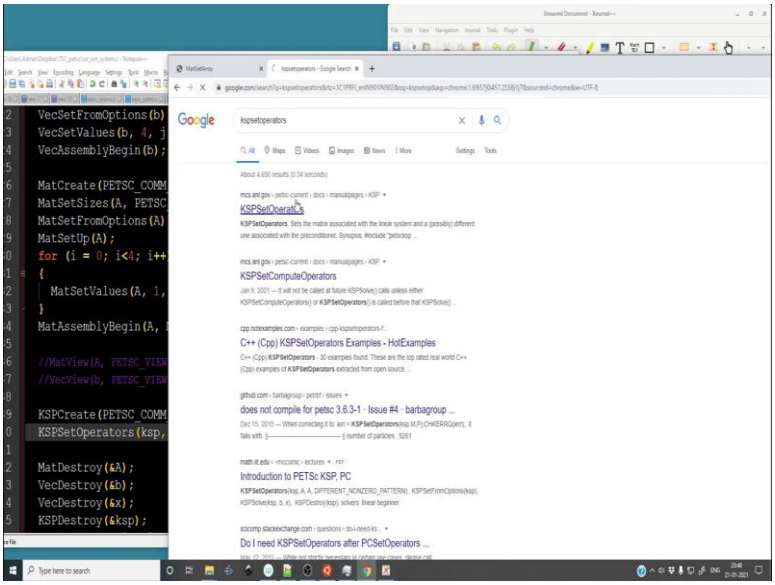


So, steps are quite similar. So, it contains KSPCreate and finally. So, there is no KSPAssembly but ultimately you have to destroy. So, it there will be a KSPDestroy and

we will pass the address of ksp ok and eventually we have to also destroy x. So, these are some things you need to always do.

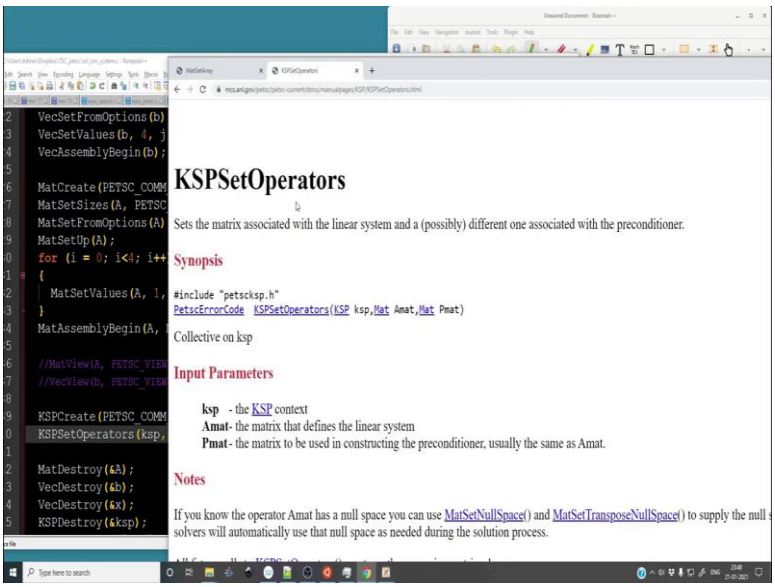
So, once we create KSP again it has to be done on com world we have to pass the address of ksp that is common. Then KSPSetOperators. So, it will have the ksp object and A, A.

(Refer Slide Time: 11:03)



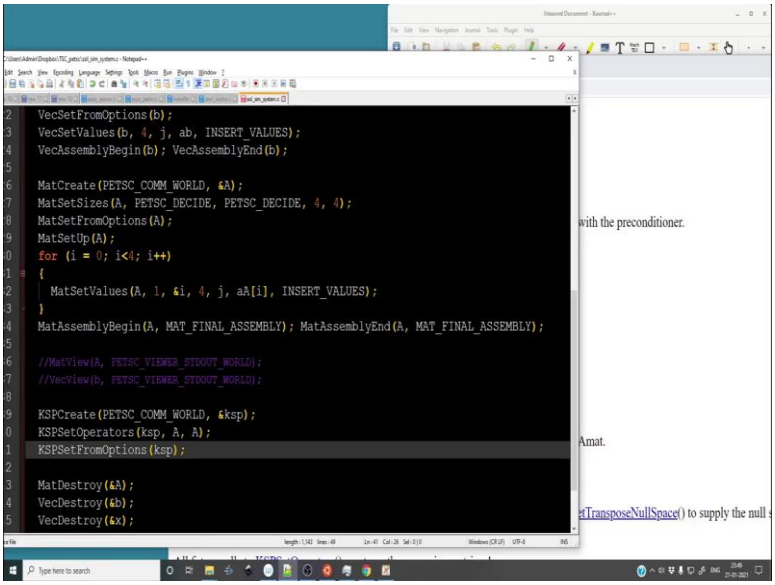
So, KSPSetOperators actually sets the preconditioners let me just show you.

(Refer Slide Time: 11:09)



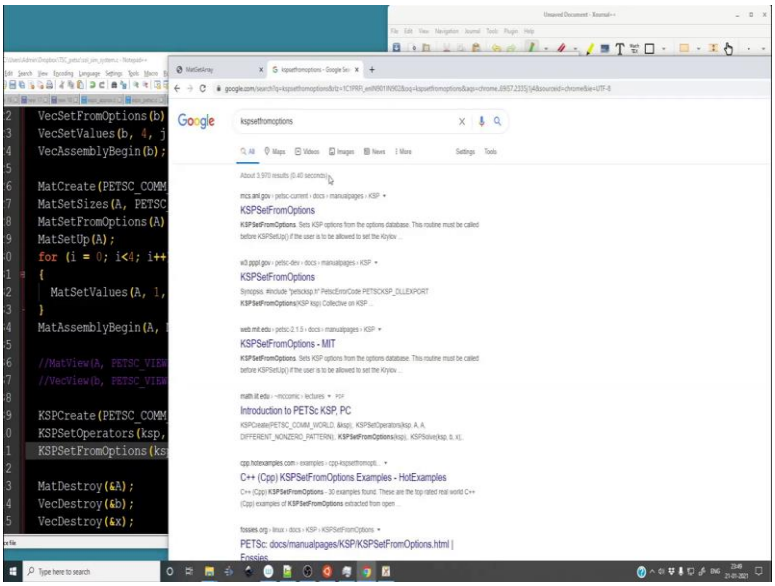
So, it contains the matrix which defines the linear system and the matrix to be used in the preconditioner usually it is the same as Amat that is why we have passed A, A, if it is something else you are always free to give this a different name, but in this case we are going to pass the same matrix to create the default preconditioner. So, we have set the operators.

(Refer Slide Time: 11:43)

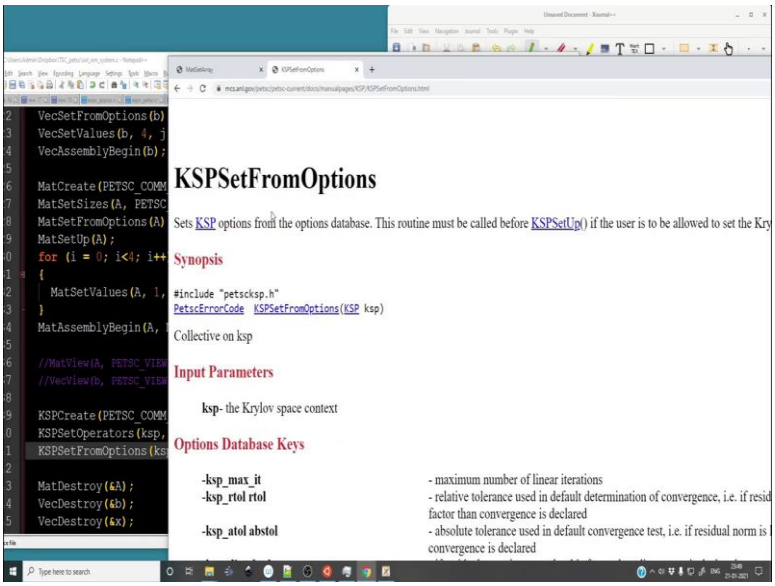


Then we will do KSPSetFromOptions and this will simply act on ksp and set from options is usually going to use the flags that you supply from the command line. So, various kinds of preconditioners, solvers whichever you can you want to choose you can always pass it from the command line to the code and during this particular step set from options it will set the particular options that you set.

(Refer Slide Time: 12:18)

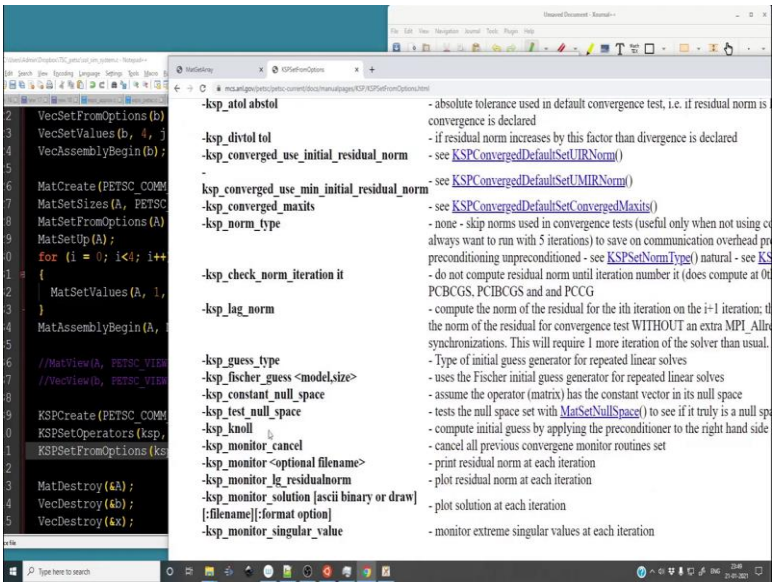


(Refer Slide Time: 12:22)



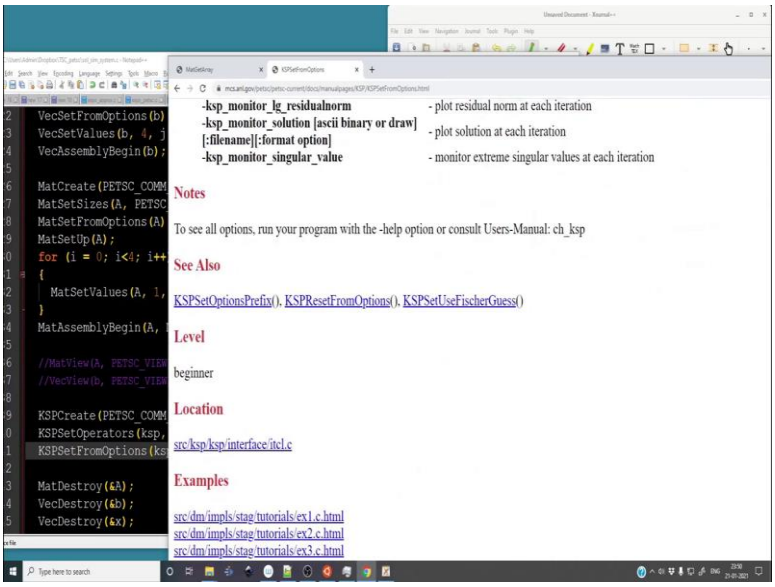
So, kspsetfromoptions. So, the point is the help files you can have an offline version of it, but I am keeping it and I am accessing the online version of the documentation. So, the function reference from KSPSetFromOptions is simply passing ksp.

(Refer Slide Time: 12:35)



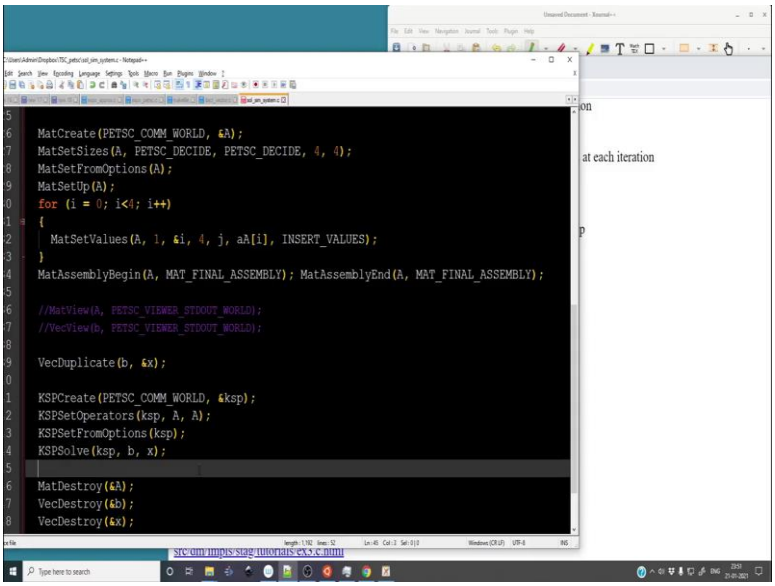
The database options are maximum iterations, relative tolerance, absolute tolerance ok.

(Refer Slide Time: 12:45)



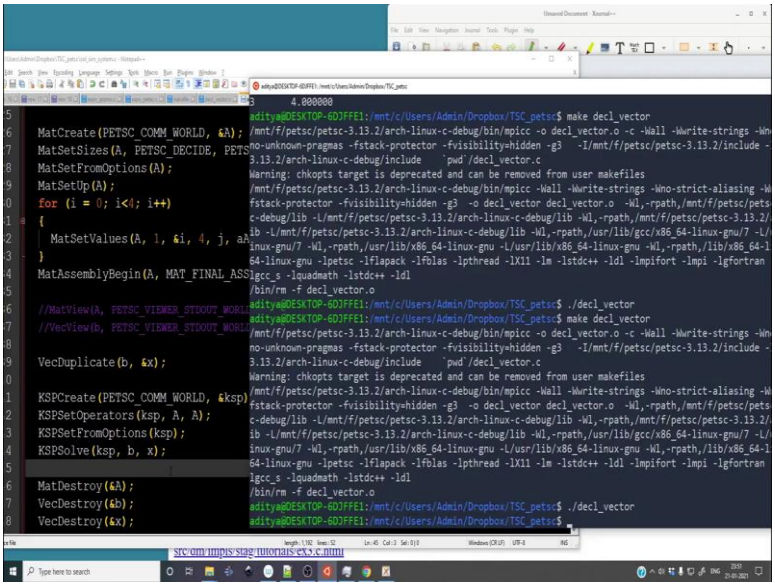
So, the guess type the monitor things like that alright. So, this is something you need to always do then what we will do is. So, before all this we need to sort of initialize what x is. So, what we will do is simply take whatever b we have and initialize x with the value of b.

(Refer Slide Time: 13:12)



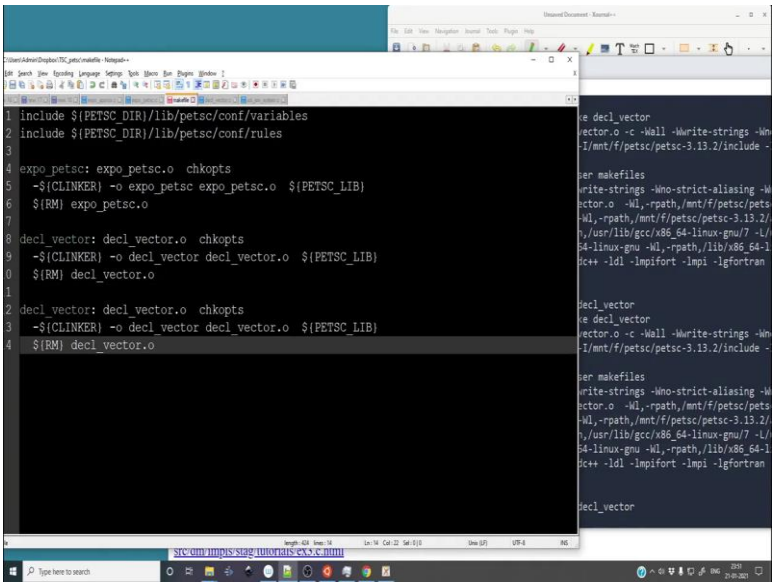
So, we will do VecDuplicate and we will duplicate b into x. So, we have to pass the address of x alright. So, after setting from options we will simply do KSPSolve ksp b comma x. So, ksp already has the information about the matrix A you have to pass the right hand side and the solution vector.

(Refer Slide Time: 13:50)



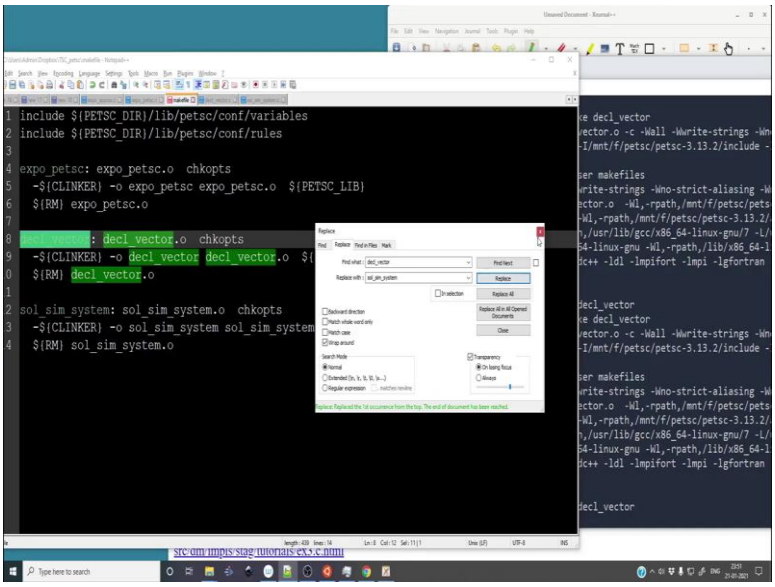
So, before anything let us quickly see whether we have a good compilation excellent. So, no errors fine actually this is a new file. So, we need to modify our make file.

(Refer Slide Time: 14:01)



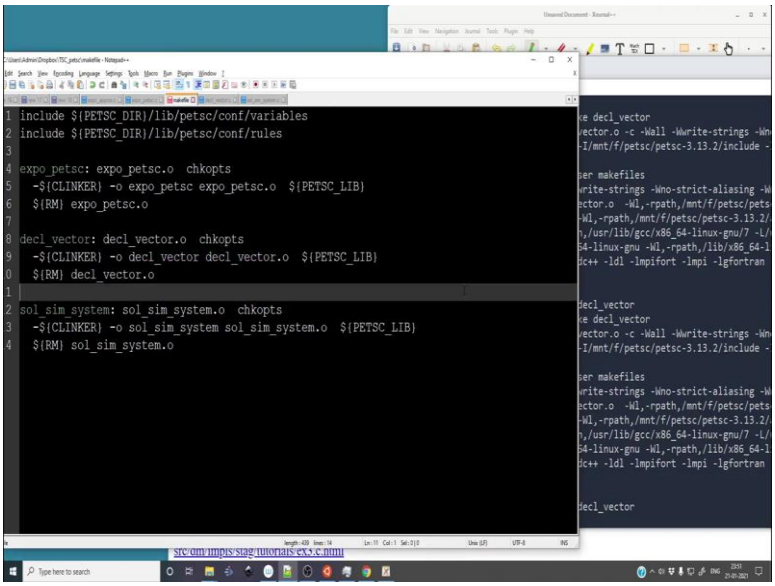
So, let us create a target for this particular for a solution system.

(Refer Slide Time: 14:09)



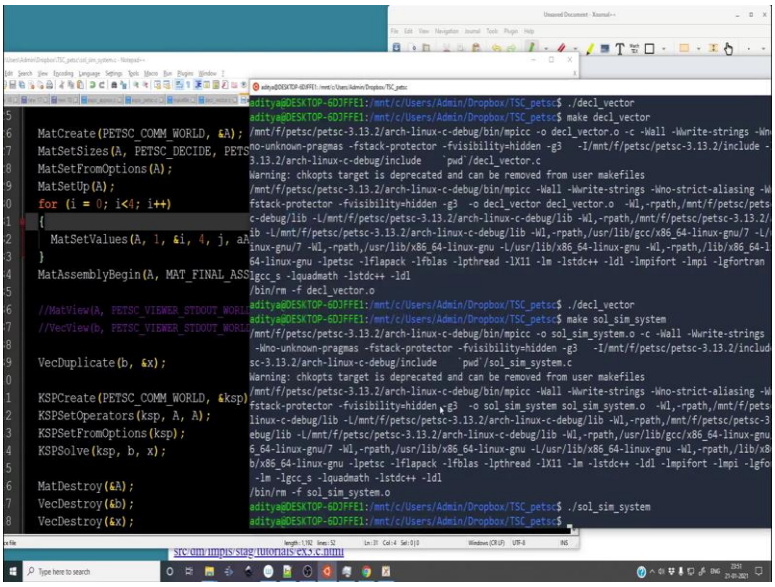
So, we have to make a new target replace, replace, replace, replace that is it.

(Refer Slide Time: 14:20)

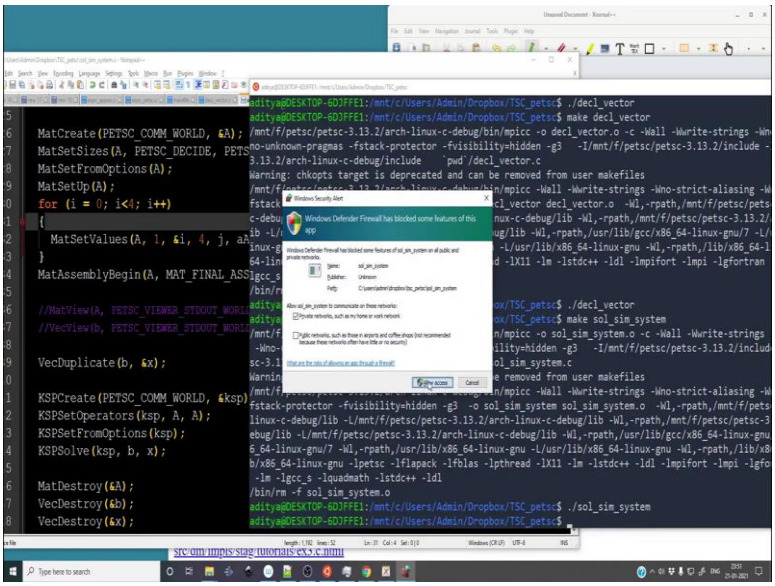


So, we need to now compile this particular target not the previous target alright.

(Refer Slide Time: 14:28)

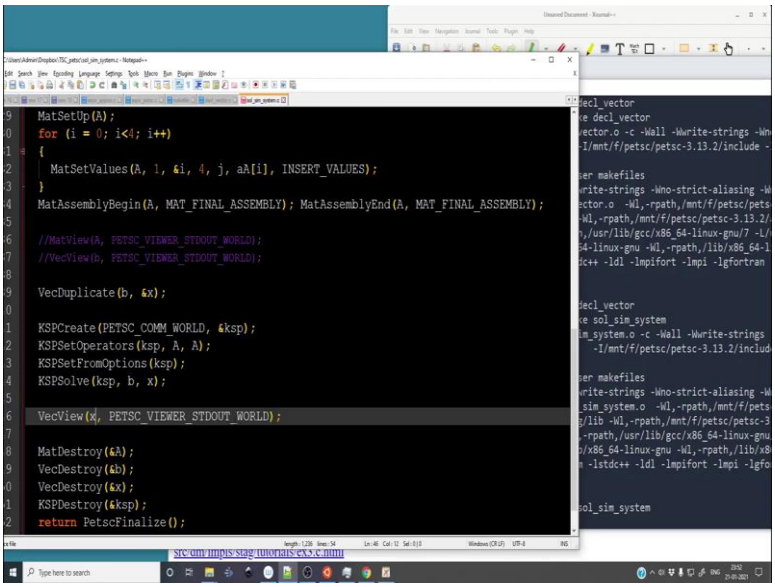


(Refer Slide Time: 14:34)



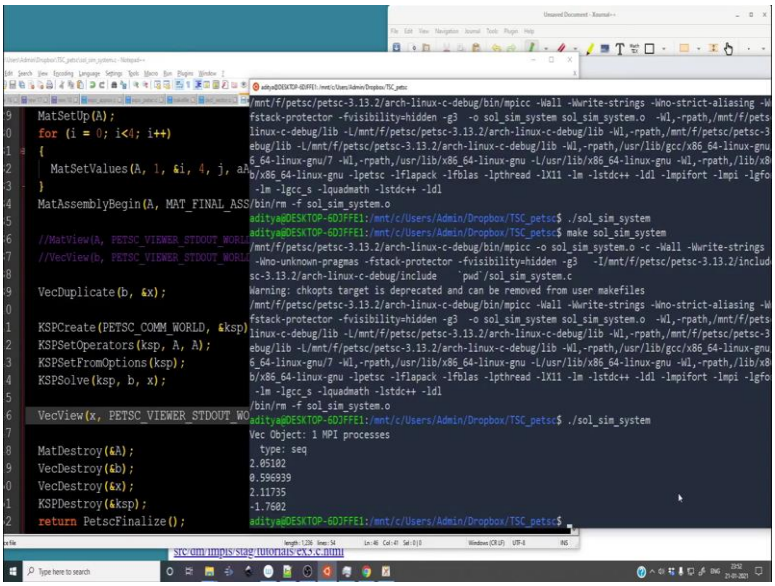
So, we will do make same system dot slash sol sim system you have to allow this do not have to do anything if you are using you know linux by default well everything runs fine. So, now, we can finally, output the vector x.

(Refer Slide Time: 14:55)



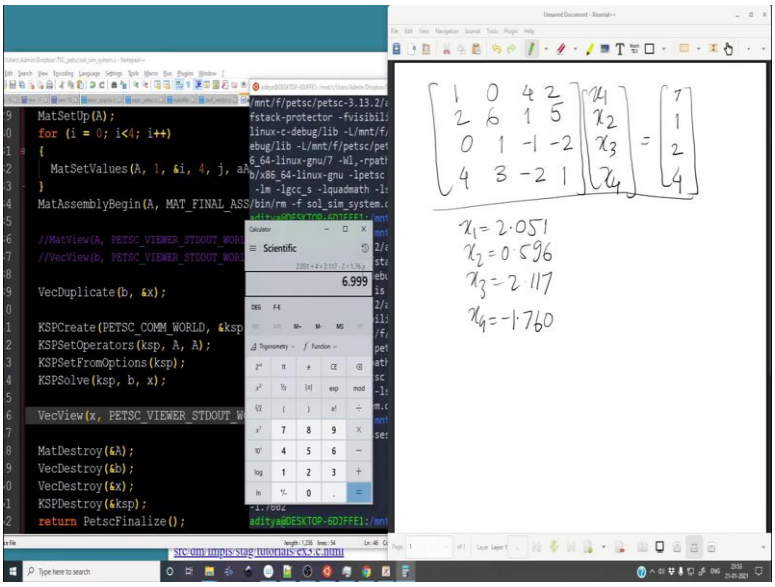
So, we can copy this back view and before destroying we can simply visualize x. So, x has to be an array of size 4.

(Refer Slide Time: 15:00)



So, let me recompile. So, this is the solution that we obtained alright.

(Refer Slide Time: 15:14)



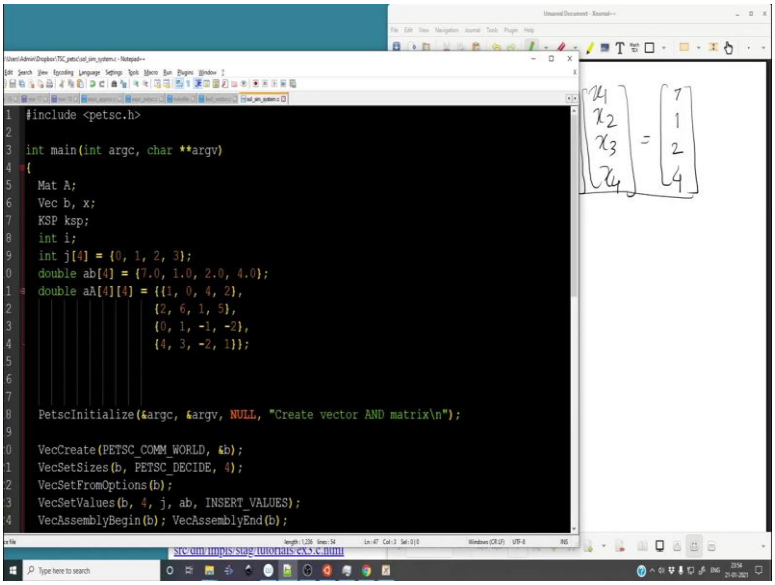
So, x1 is 2.051, x2 is 0.596, x3 is 2.117, x4 is -1.760 ok I am just taking three decimals let us quickly see whether it checks out I am shamelessly opening a calculator, but anyway let us check for the first equation 2.051 + 4 * x3 which is 2.117 -2 *1.760, 6.999 it is as good as 7.

So, I am getting it at three decimal places, but you get the point. So, far we have made a KSP object. So, KSP stands for crew of subspace. So, if you go into the area of solving

large matrices you necessarily have to go into the theories of various kinds of solvers and invariably you will end up with solving with the help of Krylov subspace. So, Krylov subspace contains a lot of methods inside, but they are all part of what they call as Krylov subspace.

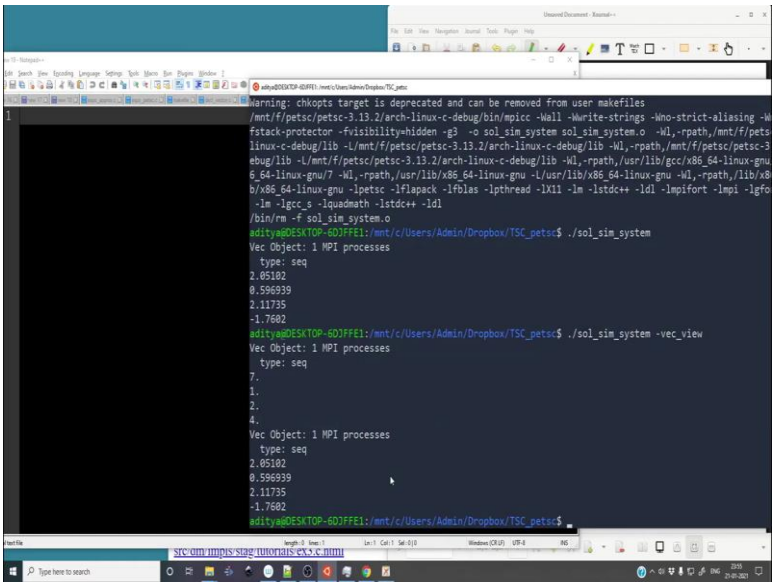
So, we are not going to discuss the theory of all these, they are left for you for a more specialized course on linear algebra perhaps, but once you start doing the course on linear algebra what you can do is make a detour of all those problems that you do you can try to numerically experiment with the help of PETSc; PETSc allows you that massively scalable set of functions data structures libraries if you will and you can do a lot of things you can experiment a lot of things.

(Refer Slide Time: 17:21)



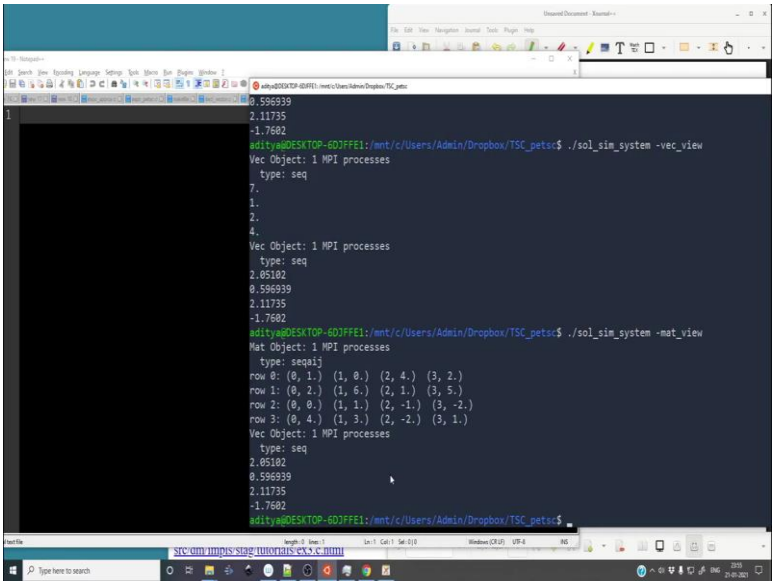
So, this completes the creation of vector matrix the display or the printing of a vector to a file and solving a system using a ksp object. So, in the next example what we are going to do in. In fact, before going to that what we can do.

(Refer Slide Time: 17:49)

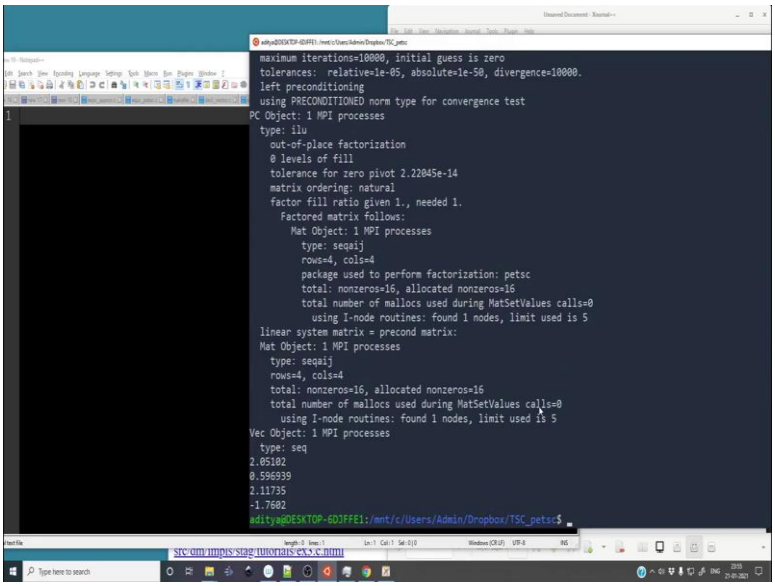


So, what we had was we had seen vec view something like this. So, it showed the different vectors that we had and there was also a mat view something like this.

(Refer Slide Time: 18:03)

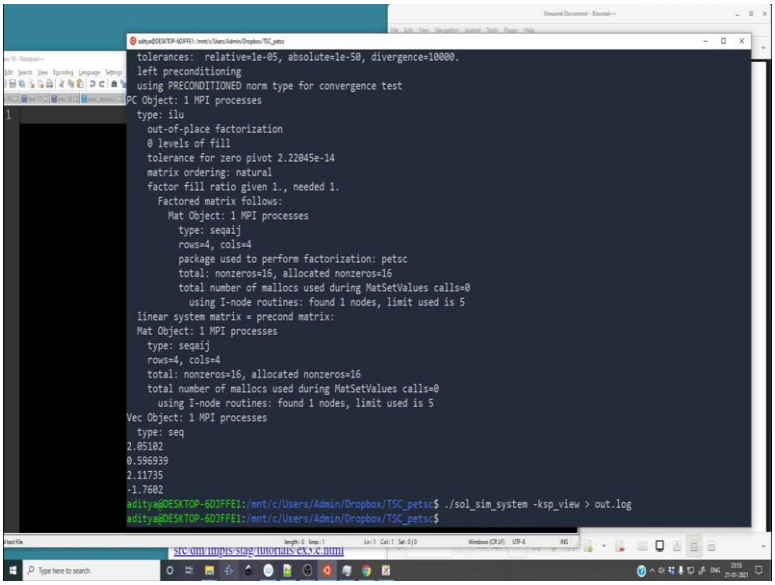


(Refer Slide Time: 18:12)



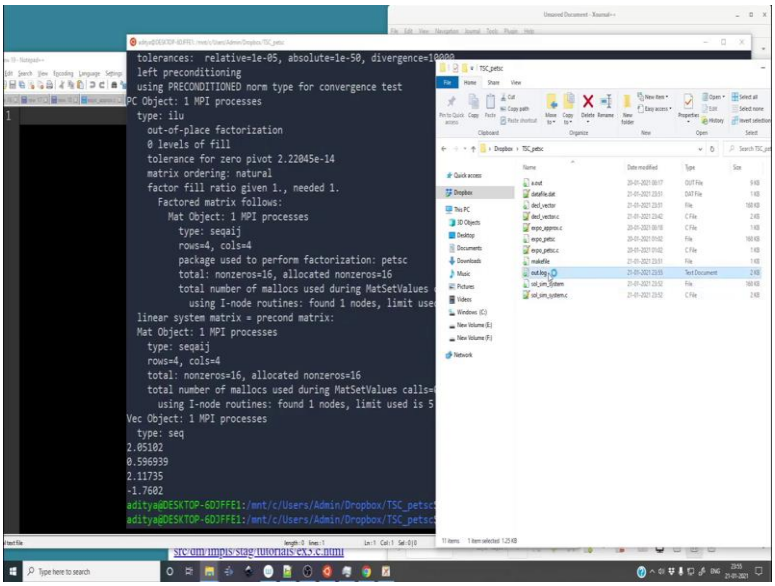
But there is also a ksp view that shows us the information about the method used. So, what do we have? Unable to scroll up.

(Refer Slide Time: 18:36)

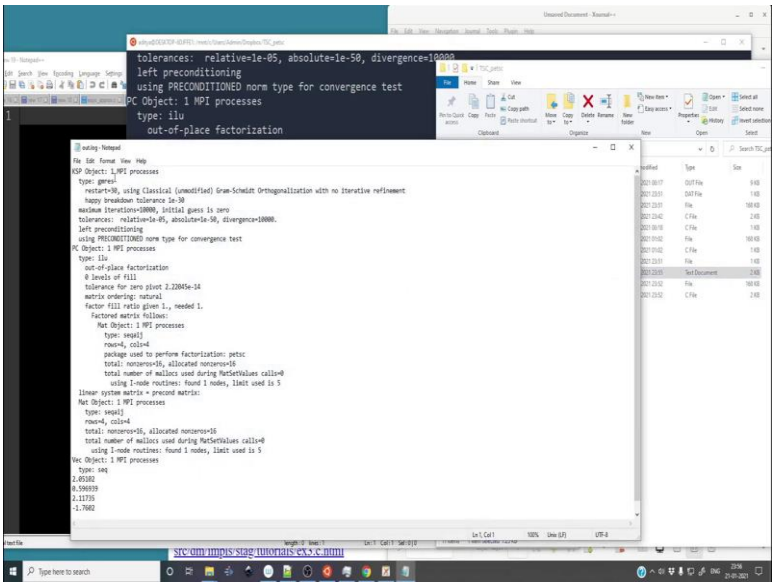


What we can do is; pipe it to a file we can look at the log file.

(Refer Slide Time: 18:44)

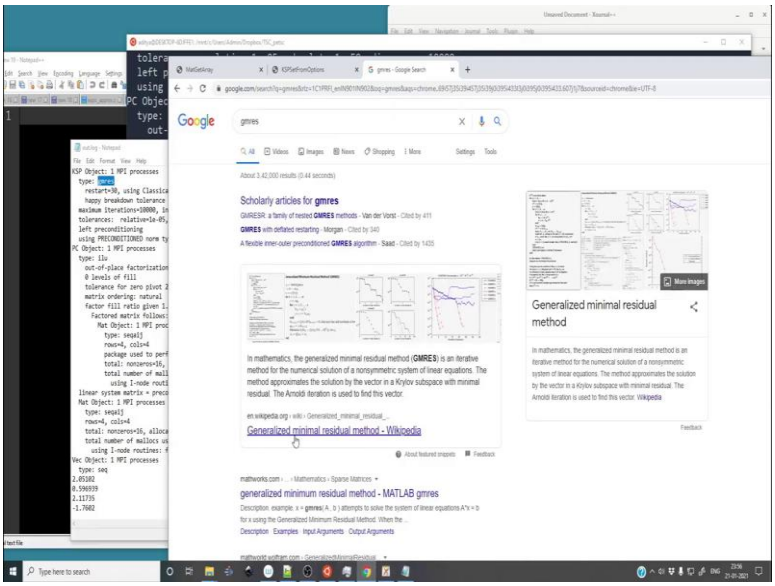


(Refer Slide Time: 18:47)



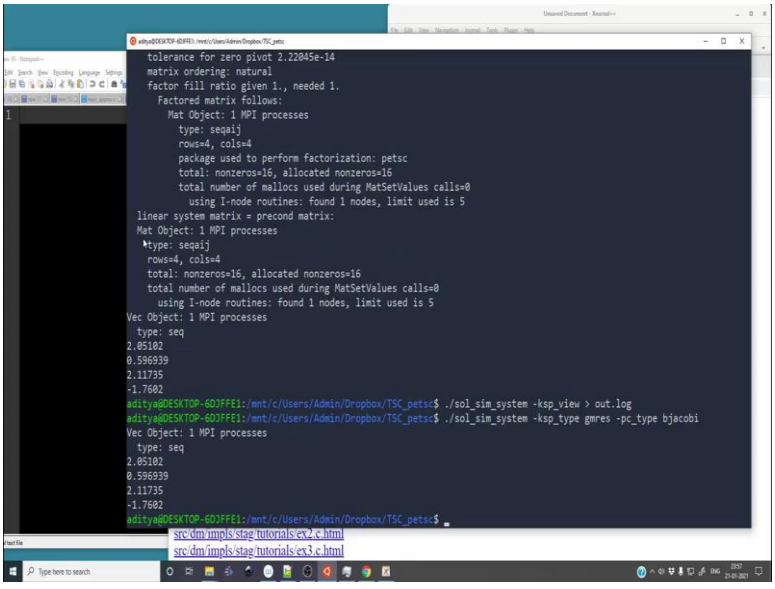
So, KSP object one MPI it is using a gmres. So, gmres stands for generalized minimum well I forget the full form gmres generalized minimum residual method ok.

(Refer Slide Time: 19:00)



So, that particular method has certain number of restarts, it has a breakdown tolerance maximum of iterations PRECONDITIONED norm for convergence test it is using a PRECONDITIONED it is using ilu in complete lu decomposition as the PRECONDITIONED and it shows all the information that you need ok.

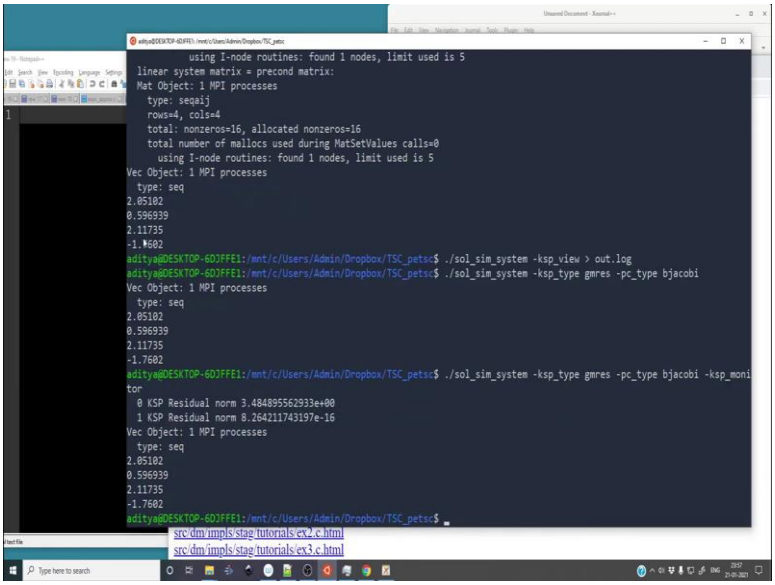
(Refer Slide Time: 19:46)



So, what we can do? In fact, is change the solver type at runtime. So, what we can we see we can change various options. In fact, we can change various options we can do - ksp

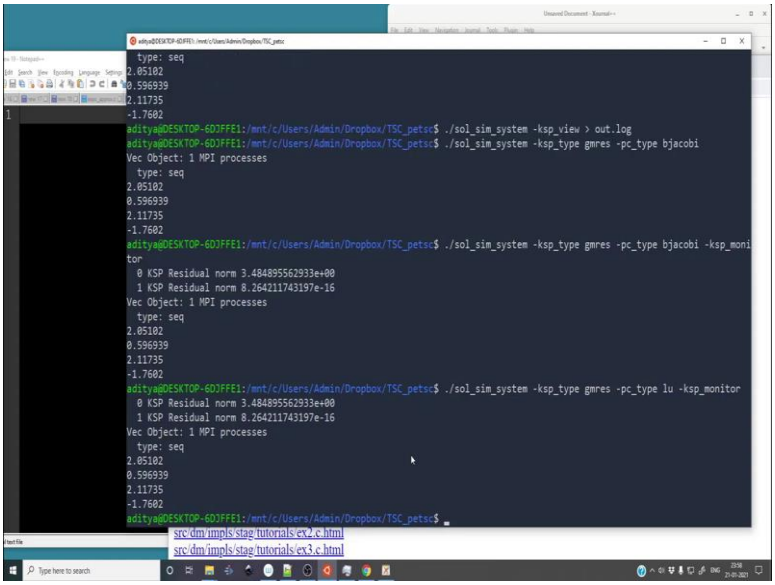
type gmres - pc type bjacobi block jacobi let us see and; obviously, gives the same answer, but it is uses a different pre-conditioner.

(Refer Slide Time: 20:25)



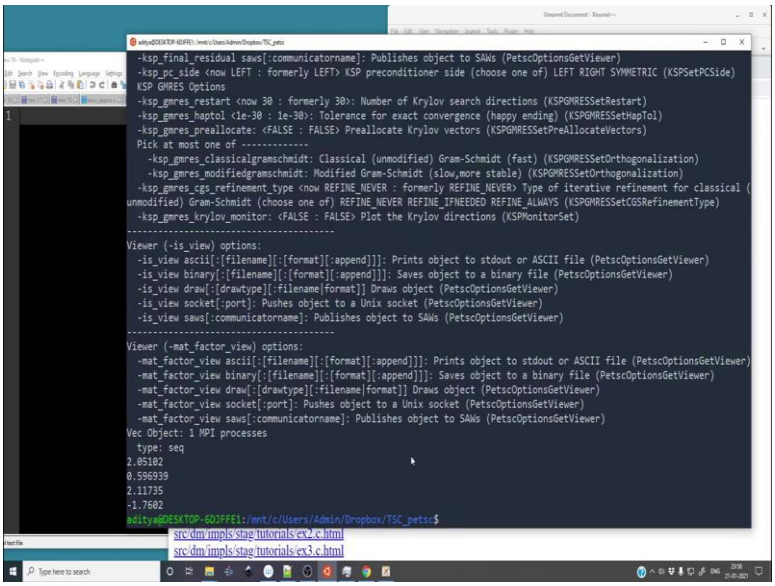
We can also do a minus ksp monitor to see how the progress is. So, immediately after many iteration we have convergence this is expected for such a small system ok. We can use a preconditioner which is not the ilu but in fact, a simple lu decomposition.

(Refer Slide Time: 20:41)



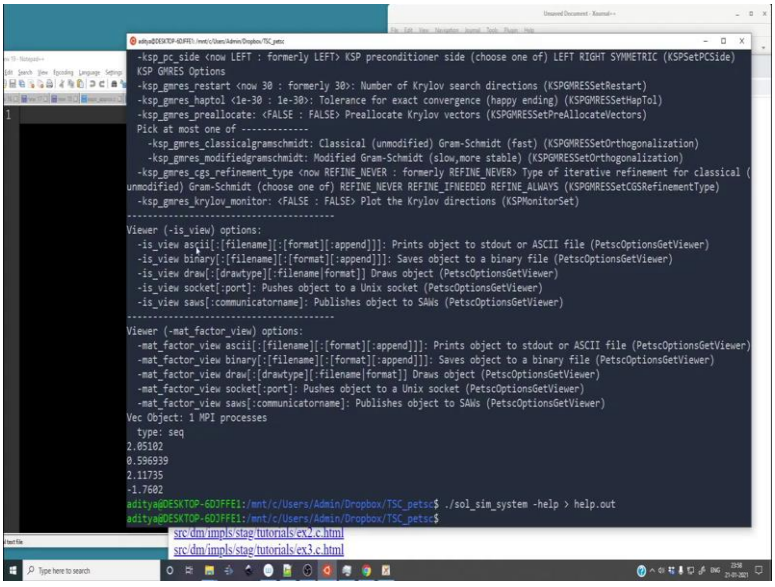
So, we can change various options at runtime ok.

(Refer Slide Time: 20:56)



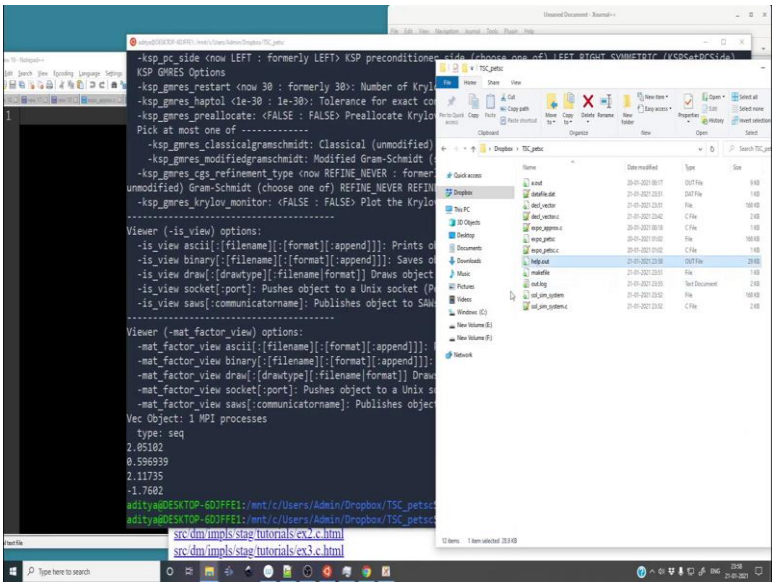
So, what are the different options available to us let us see; let us see minus help. So, it shows all the different ok.

(Refer Slide Time: 21:08)

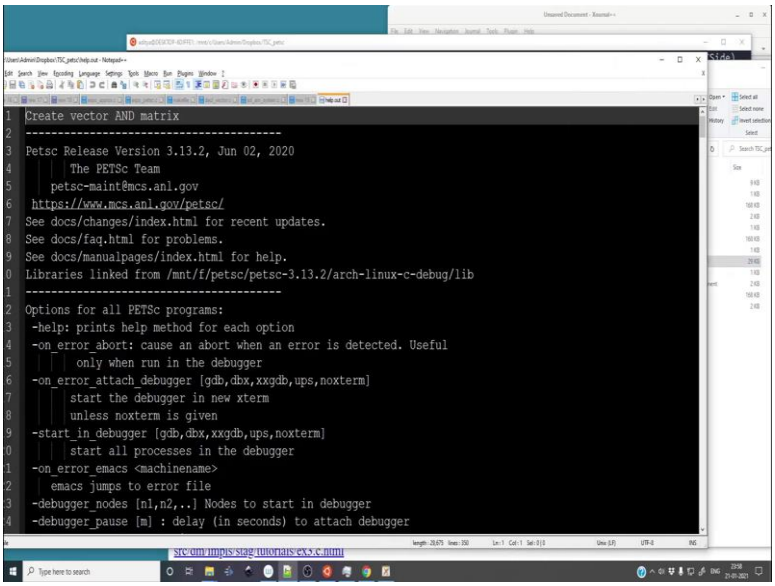


I cannot scroll minus help. In fact, let us get only. So, it has various kinds of. So, let us pipe it to a file let us pipe it to a file. So, that we can open it easily.

(Refer Slide Time: 21:27)

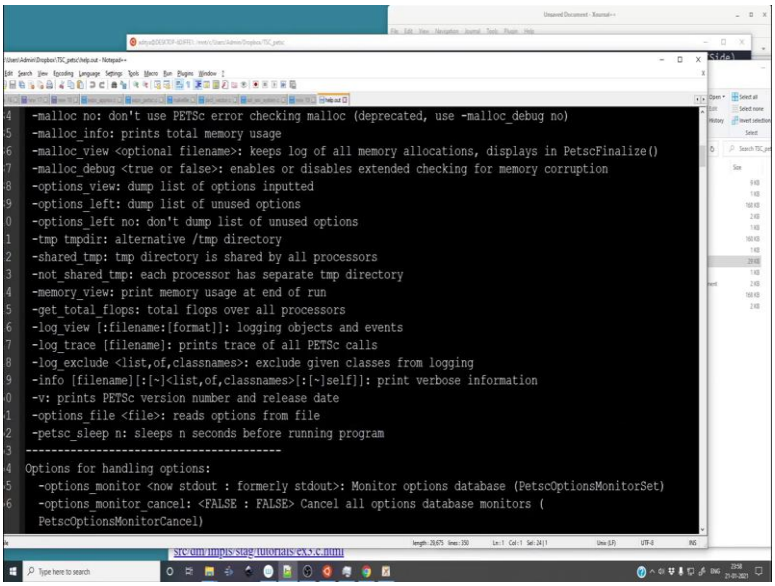


(Refer Slide Time: 21:34)

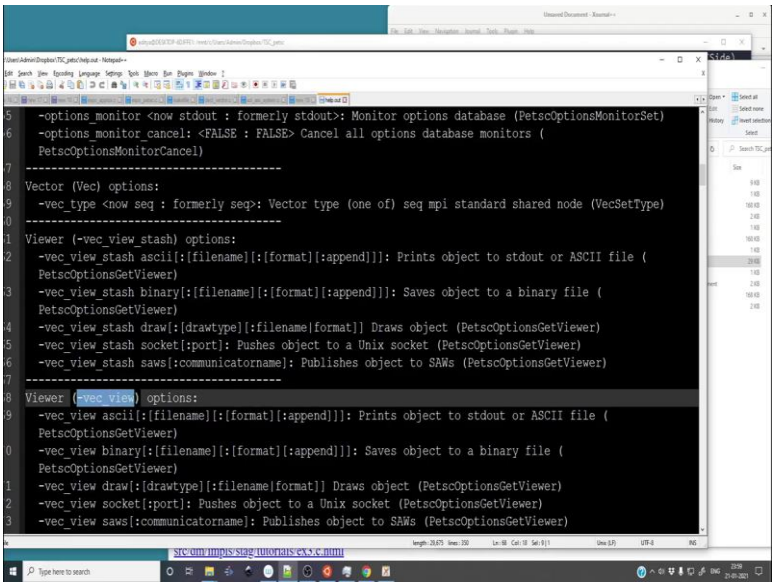


So, help dot out. So, what does help dot out contain it contains the help string that we put then it contains a bunch of options that you can pass.

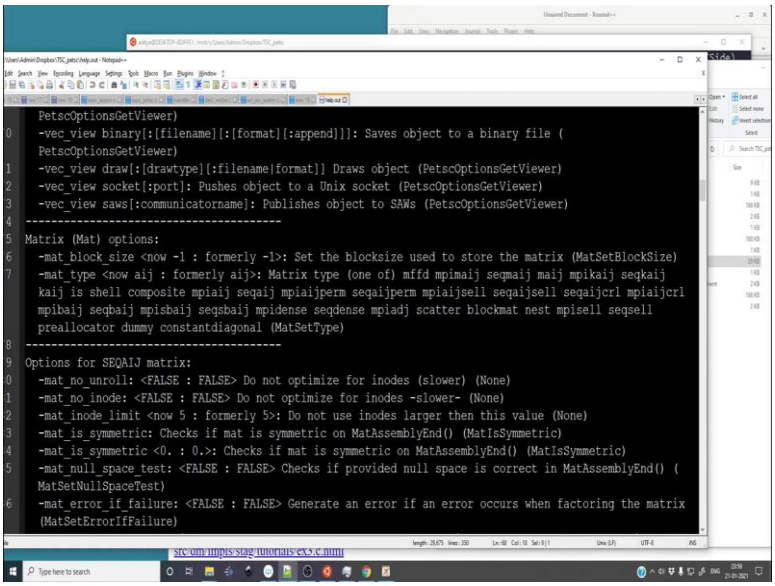
(Refer Slide Time: 21:43)



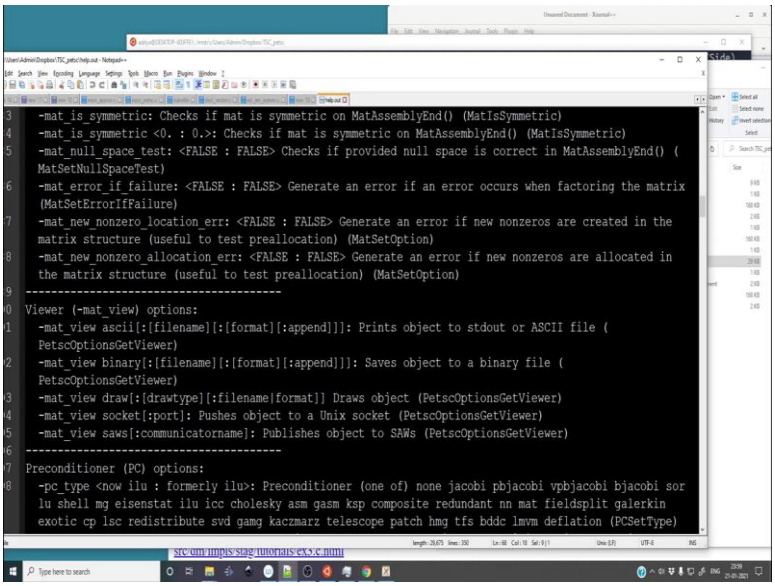
(Refer Slide Time: 21:49)



(Refer Slide Time: 21:55)

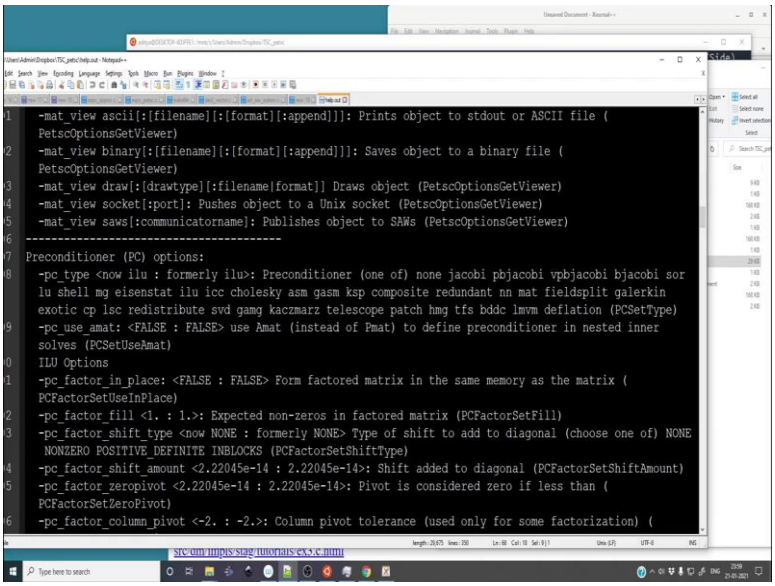


(Refer Slide Time: 21:59)



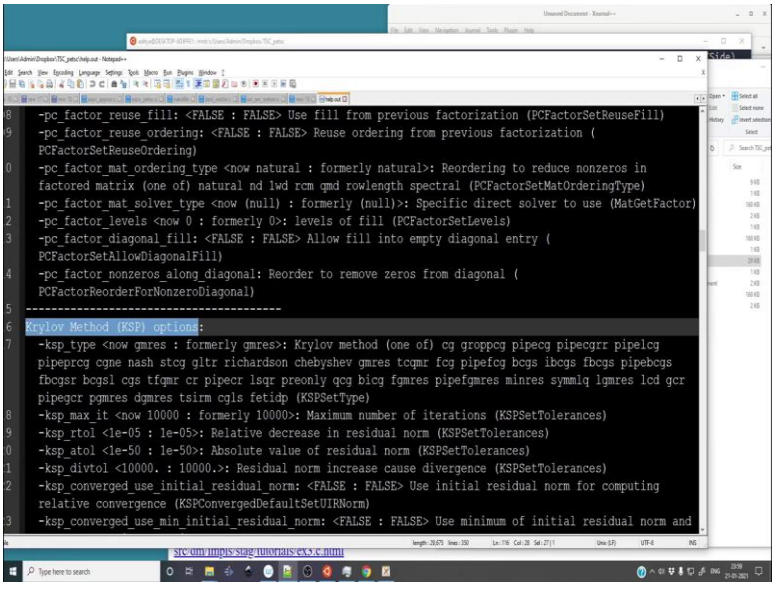
So, what are the different. So, it shows you the different vec view options it shows the different matrix options.

(Refer Slide Time: 22:03)



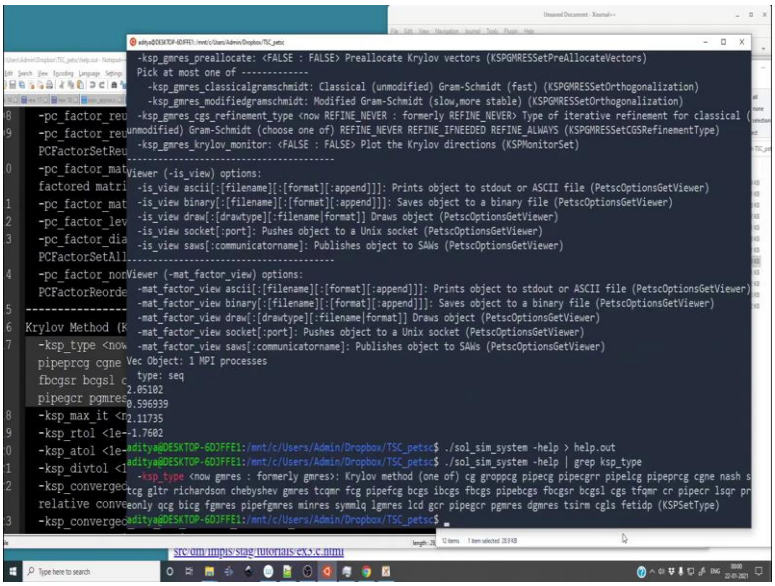
It shows the different viewer options, the different preconditioner options. So, ilu then you have jacobi, pbjacobi, vpbjacobi, bjacobi, sor, shell, mg, eisenstat, ilu, icc, cholesky, asm. So, different kinds of preconditioners are present. So, we are more interested in the KSP options.

(Refer Slide Time: 22:24)



So, there are a bunch of options and we want to extract this line. So, how do you extract this line?

(Refer Slide Time: 22:40)

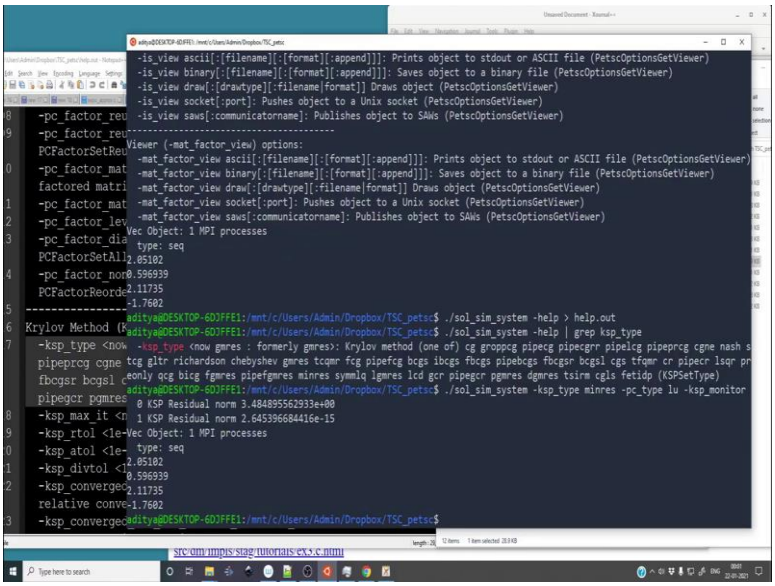


So, there are in linux there are specialized tools one of the specialized tools is called as grip. So, it will scan. So, we are piping this help output to grep which will give us the line which contains the keyword. So, the keyword will be the keyword we want is ksp type.

So, we are going to give it ksp type. So, the output of grep is shown over here. So, grep is a linux tool you do not need to worry about it. So, one of these methods cg, groppcg, pipecg, pipecgr the host of methods minres, fgmres, biconjugate, gradients stuff like that. So, you have a whole set of methods that very efficient and intelligent people have coded so that you do not have to go ahead and code all this.

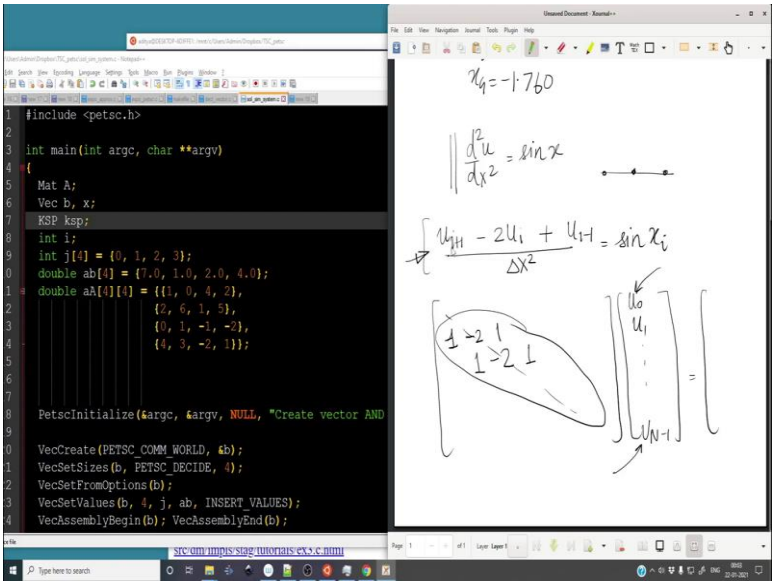
But remember novel it always helps to know what each algorithm does inside out, but it is unlikely that someone like me is going to write a code which is multi-processor which is more efficient than what these people have it is unlikely.

(Refer Slide Time: 24:02)



So in fact, let us change some more options. So, now, that we know these methods exist we can call minres. So, it uses the minres thing it gives us a slightly different norm after one iteration, but it has already converged you do not need to worry about it. So, that is how you can choose solvers at runtime I have shown you a few linux tips and tricks. So, now, let us solve for a tri-diagonal system.

(Refer Slide Time: 24:48)



So, what I am trying to get at is if you are solving a boundary value problem in 1D say $d^2u/dx^2 = \sin x$ right if you are trying to solve this. So, what will you do? You are going

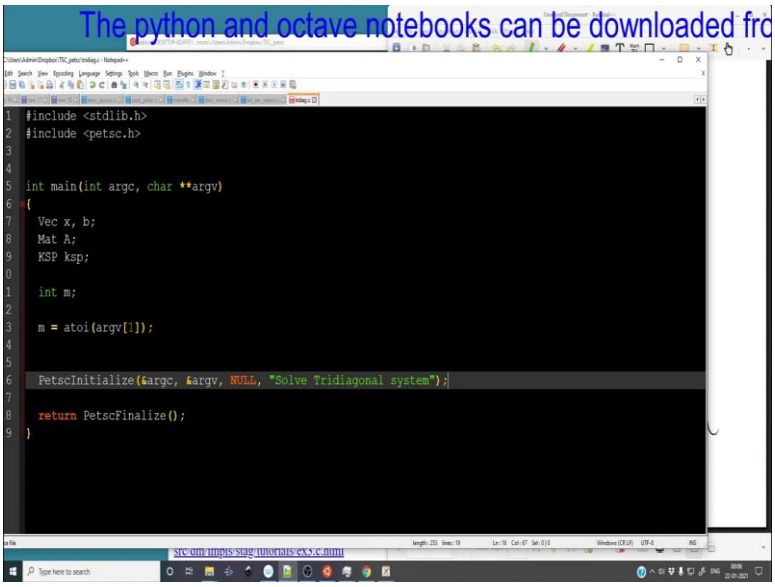
to split this into a discrete form over one central node one right node and one left node. So, what do you have? You have $u_{i+1} - 2u_i + u_{i-1}$ upon $\Delta x^2 = \sin x_i$.

And then as we have seen in the python code you assemble this matrix when assembling this matrix apart from the boundaries you get a tri-diagonal system ok. So, the diagonal elements will be - 2 the off diagonal will be 1 the left angle will be 1 it will keep doing this except the boundary.

Because at the boundary the value of u will be specified or the gradient will be specified after which you can do, you can take a ghost node and you can find out the appropriate boundary condition using this equation itself and we have covered this how a tri diagonal system comes into existence.

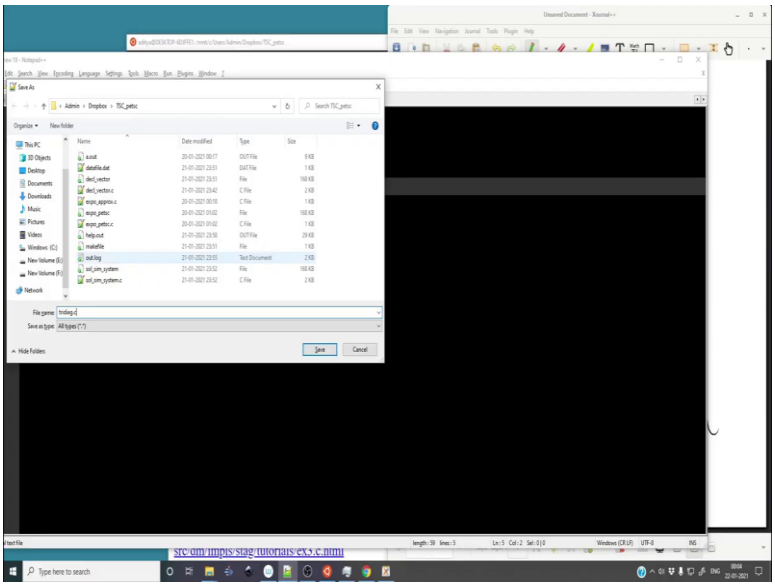
So, this kind of tri diagonal system has to be often times algorithmically done I mean in this particular example it is still very easy, but nevertheless you have to build it for an arbitrary size we do not restrict it to a 4 by 4 matrix. So, we must write the code so that you can assemble the matrix for any arbitrary size. So, let us see how that can be done.

(Refer Slide Time: 26:51)



So, first of all we are going to have hash include std not std petsc dot h int main int argc char start star argv.

(Refer Slide Time: 27:14)

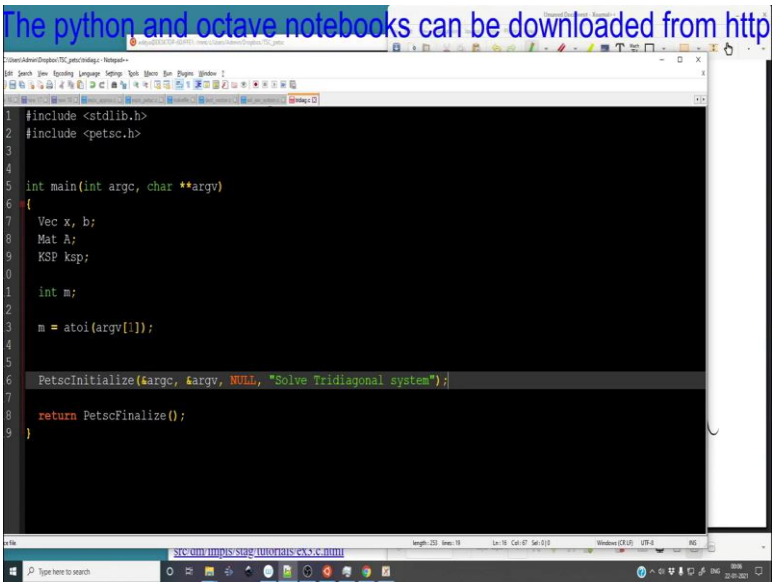


So, let us save it as tridiag dot c and we will return let us see finalize alright. So, we are going to define the vectors as we did in the last code x and b we are going to define the matrix A we are going to define the KSP object ksp and what we are going to do now is define the integer m what will m do? It will hold the size of the matrix and we can put some default or we can overload it with the help of some command line argument.

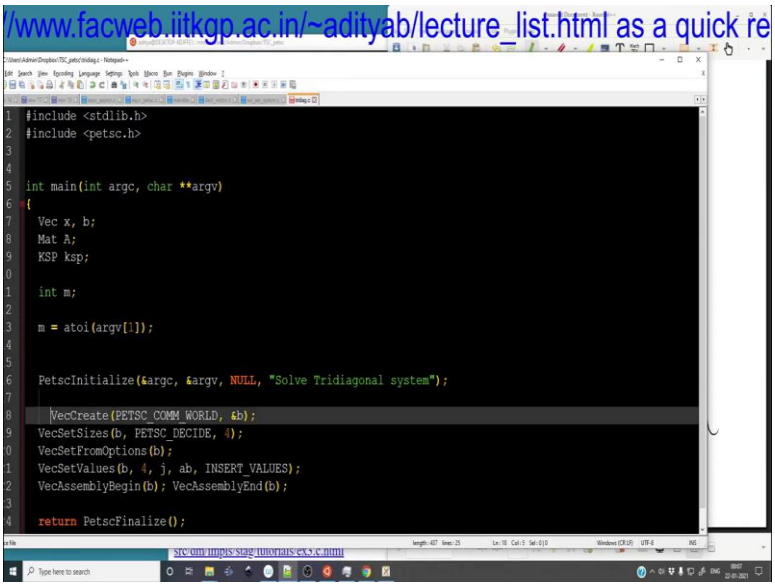
Let us keep it m for now and let us write m equal to atoi of argv first thing. So, we must pass through the command line argument the integer and the size of the matrix. So, let us include stdlib as well just in case let me put this before this ok. So, m contains this and we will define all the other parameters as we go in the code.

So, first things first we must do a PetscInitialize this should contain PETSC COMM WORLD and no this should not contain PETSC COMM WORLD rather it should contain the address of argc address of argv and null and some help string alright.

(Refer Slide Time: 29:11)

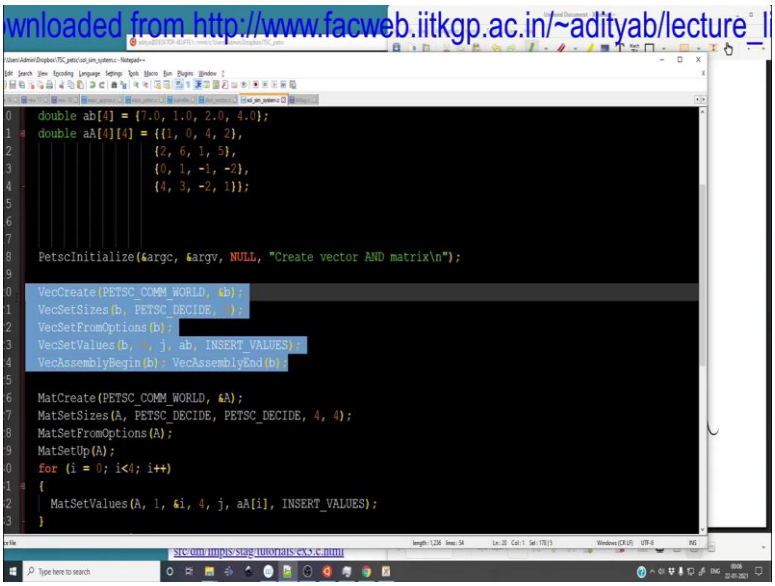


(Refer Slide Time: 29:30)



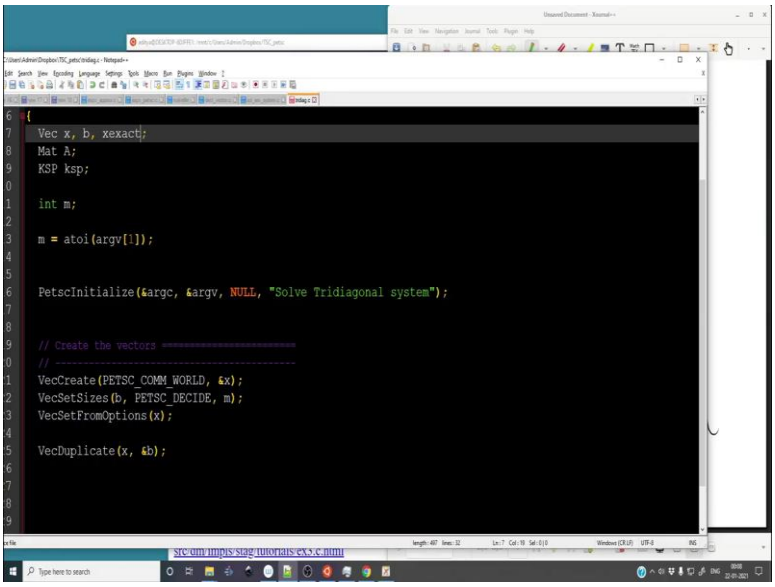
So, now we have to do the same set of lines and which we had done well we had not done, but we need to declare this vector.

(Refer Slide Time: 29:42)



So, let me copy this, let us not waste time in writing all this we have to create the vector.

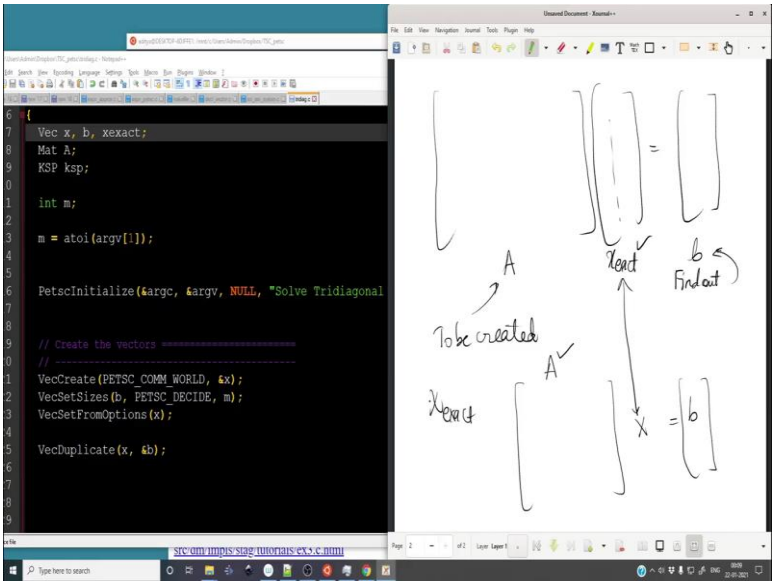
(Refer Slide Time: 29:48)



So, this bit of code is to create the vectors it is good to always type down some comments meaningful comments. So, what kind of vector do we want to build? So, we want to build let us create x and the size will be m it will not be 4 alright it will be m. So, we will build. So, set values we have to get rid of because we are going to do it somewhere else alright; let me keep it we do not need and we can delete it for now.

So, we just do set values of options once x is created we can also create b. So, we can do VecDuplicate. So, we are going to pass x and the value of b. So, essentially we have created duplicate vector b alright then. In fact, we are going to make a very synthetic problem. So, we will need something called as x exact. So, what we are going to do is quite simple.

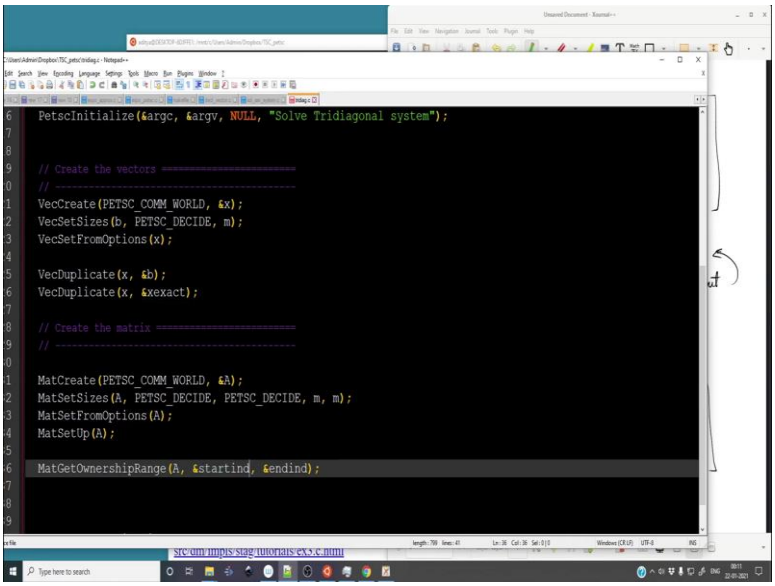
(Refer Slide Time: 31:15)



We are going to take our differentiation matrix A. So, we are going to create this A to be created we are going to create x exact that is we will populate this with something which is known ok and then we are going to multiply these two and find out some matrix b once we have b then we are going to synthetically solve again.

So, A is known X will now be unknown equal to b. So, we are going to solve this afterwards and we are going to compare x exact and X well in the ideal world they should be exactly equal, but let us see because we are using iterative algorithms we will have large matrices and they may be close, but definitely not equal and that is; and that is the thing you have to live with numerical algorithms.

(Refer Slide Time: 32:22)

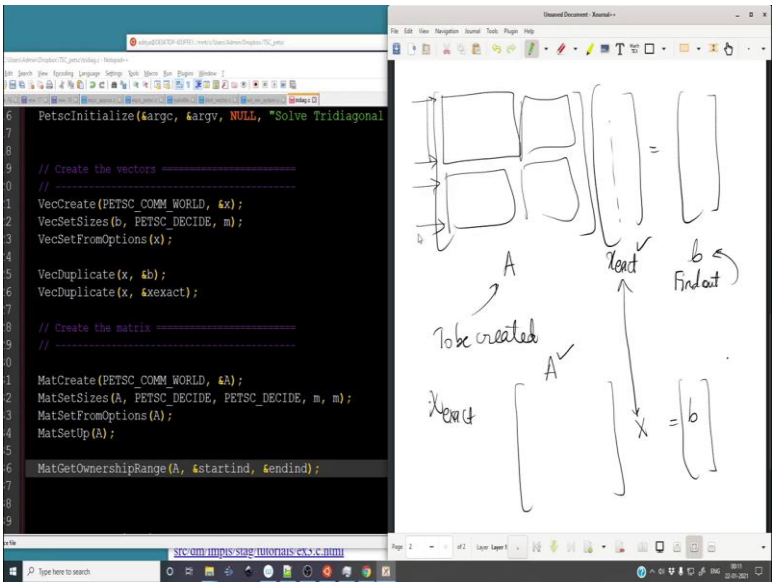


So, set from options x we are going to duplicate we are going to make another duplicate of x comma and there is an xexact alright. So, far we have created the three matrices now we are going to create matrix. So, again if we going to reuse some of the code we are going to reuse this.

So, we have to create the matrix A you have to set sizes instead of 4 it will be m right then MatrixSetFromOptions A good what else and we have to do matrix MatsSetup A something which we have done over here as well. So, once we have this we have now created the two vectors another three vectors and the matrix.

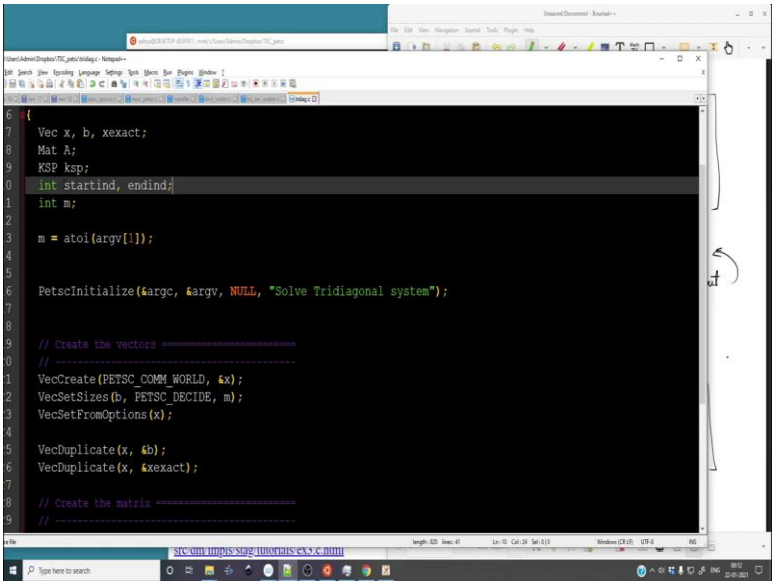
So, what we are going to do is we are going to first obtain the loop range. So, let me show you we are going to do MatGetOwnershipRange A this will be startind this will be endind this means it will look at. So, the program will look at the array A if you have multiple processes it will distribute the index over which the program is going to loop it will divide into sub chunks ok.

(Refer Slide Time: 34:24)



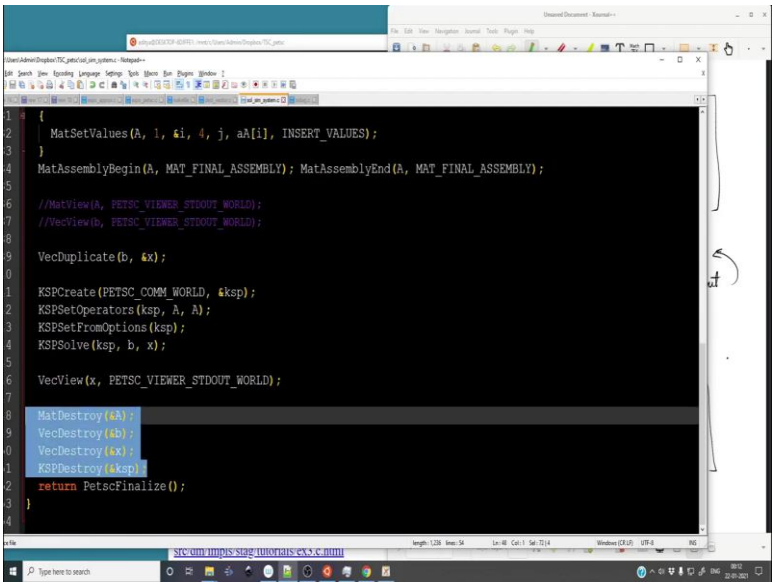
So, it can have something like this startind is this endind is this startind is this it depends if you are doing it in parallel I am not simply speaking startind and endind will simply contain the dimension of the array. So, hence the ownership range different processors will own different ranges and what we can do already is so we have pretend declare startind and endind because if we have not declared them. So, they need to be declared as integers.

(Refer Slide Time: 35:07)

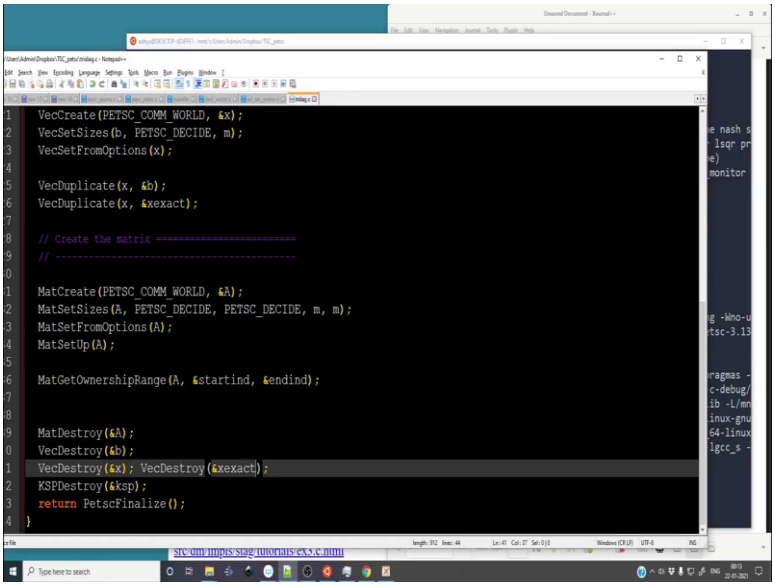


So, let us do that. So, over here we need to make int startind endind. So, we have created the startind and endind. So in fact, we can print it out and I will show you before doing all these let us print out certain parts and we will see what we see. So, let me destroy the different things. So, let me reuse some of the code to destroy we will need all of this.

(Refer Slide Time: 35:35)

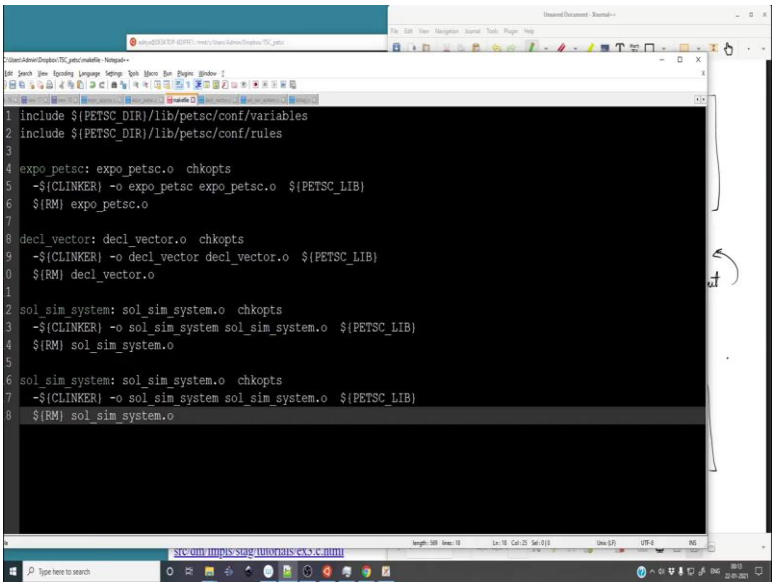


(Refer Slide Time: 35:39)

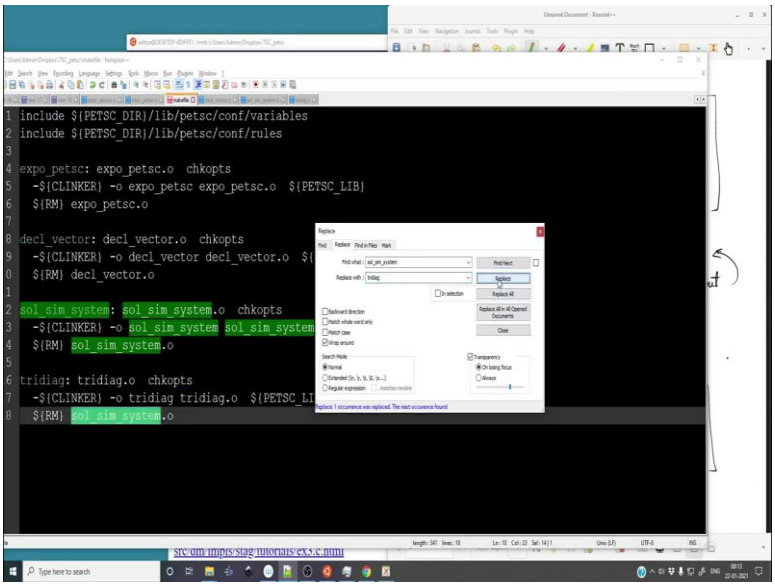


So, destroy a b x you have to destroy xexact as well. So, let me modify the make file to have a new target alright.

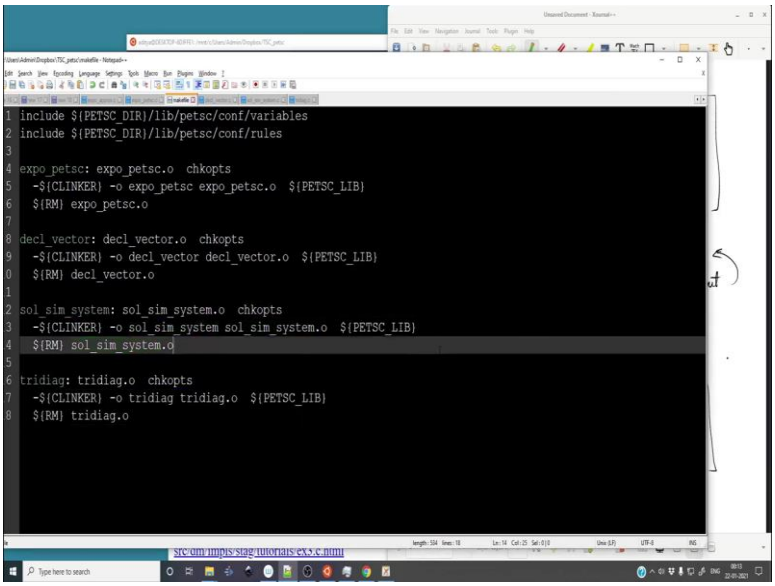
(Refer Slide Time: 35:51)



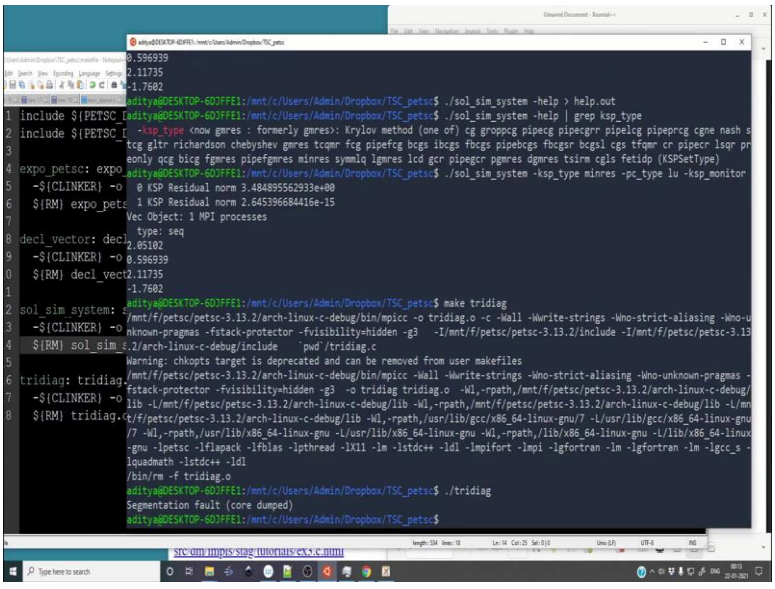
(Refer Slide Time: 36:03)



(Refer Slide Time: 36:09)

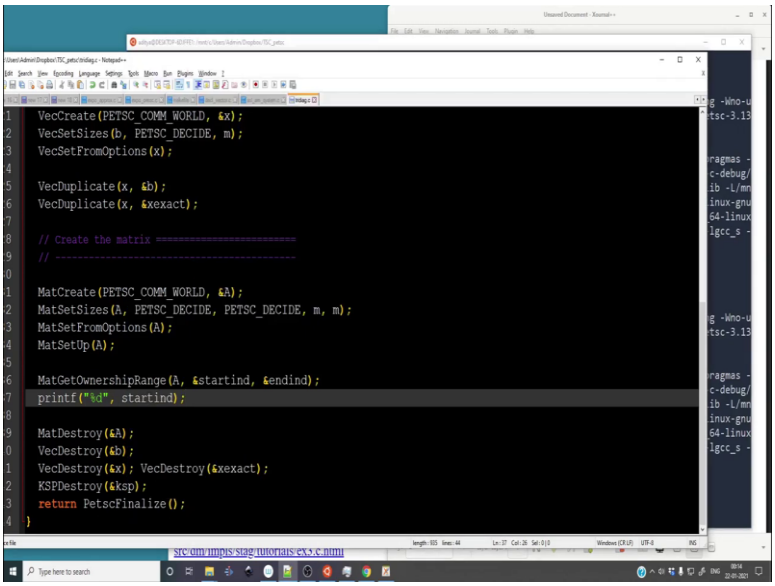


(Refer Slide Time: 36:14)



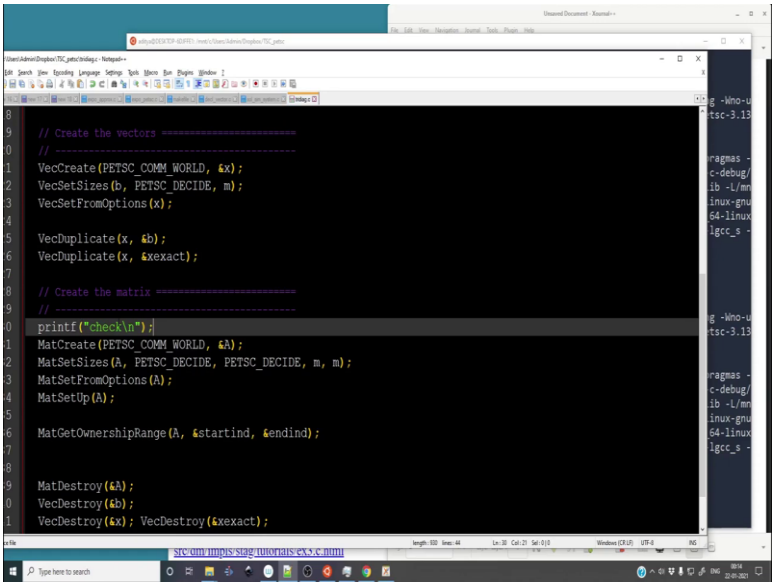
So, we have created a new target. So, let me test it compiles well dot slash tridiag segmentation fault because you have not done anything most likely it has ran into something empty ok.

(Refer Slide Time: 36:36)

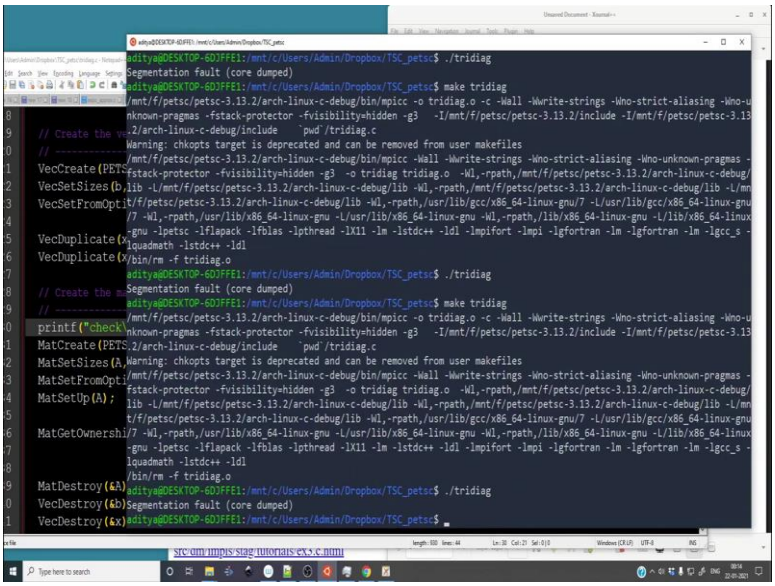


So, I do not know where the segmentation fault is, but let me see if I can print the startind or not the segmentation for much before this and that is ok. So, well you can actually pinpoint where the segmentation fault is most likely it is somewhere over here after setting up we have not done anything.

(Refer Slide Time: 37:12)

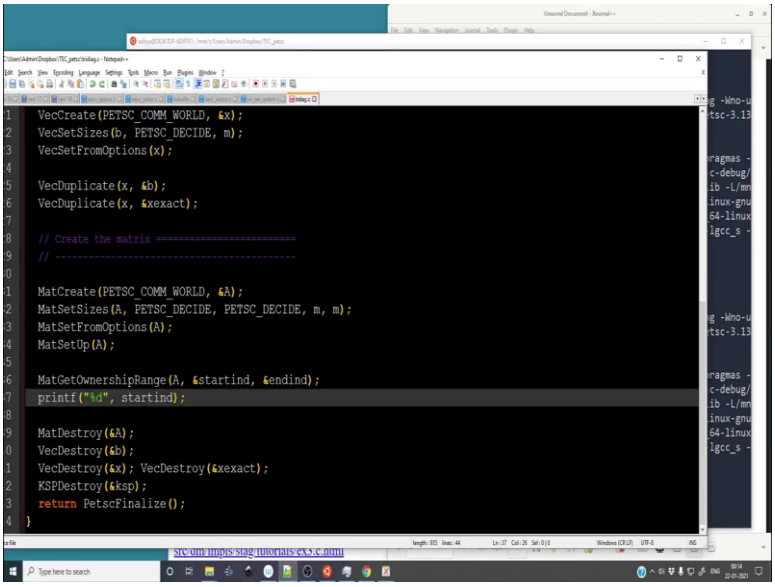


(Refer Slide Time: 37:24)

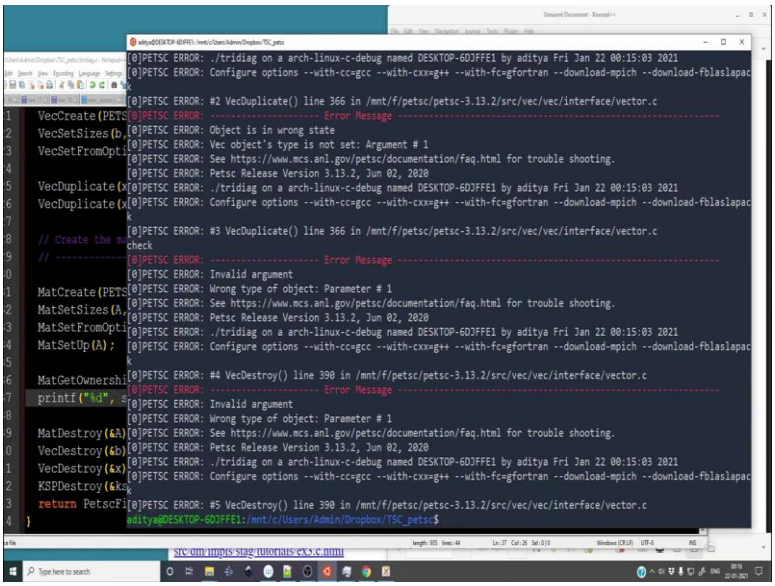


So, you can always do a printf to see I mean you can obviously use a debugger, but if you want to do it the old way then simply do this place it at various positions and see when it actually prints. It is not this we have not passed the number of not pass the elements. So, that is why m remains undefined and hence the segmentation fault well.

(Refer Slide Time: 37:45)

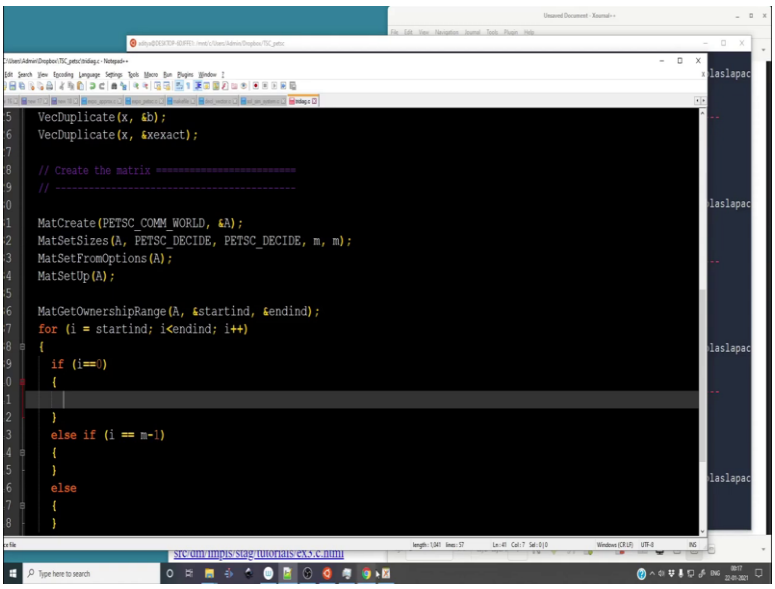


(Refer Slide Time: 37:48)



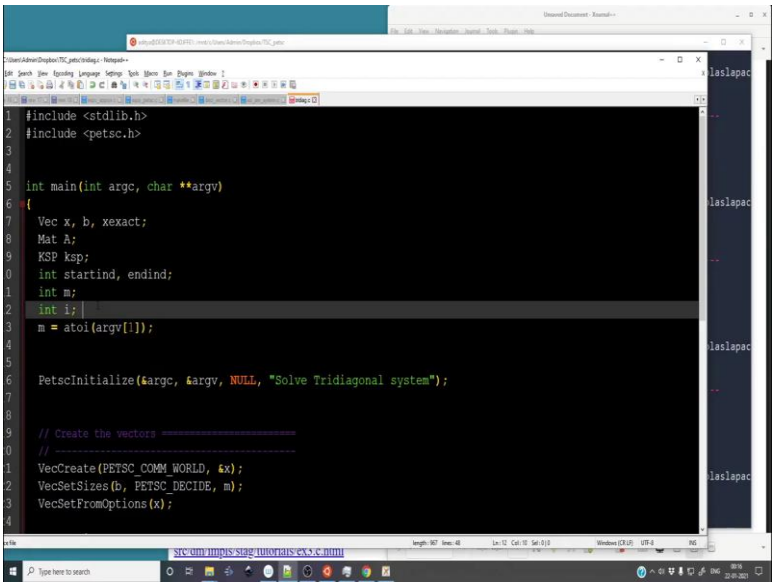
So, let me put this we do have a big set of error and it appears to be because of the lack of declaration, can we scroll? Right. So, do not worry about these errors we will sort it out we will get rid of this. So, we have to get the membership range and what we will do is we will try to fill out the matrix now.

(Refer Slide Time: 39:14)



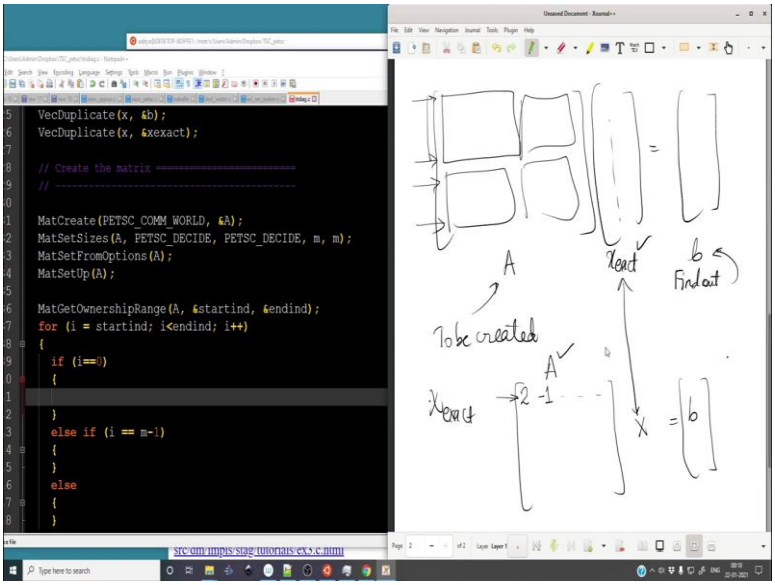
So, for i = startind i < endind i ++. So, we are going to fill in all the elements but we need to define the counter i alright.

(Refer Slide Time: 39:35)



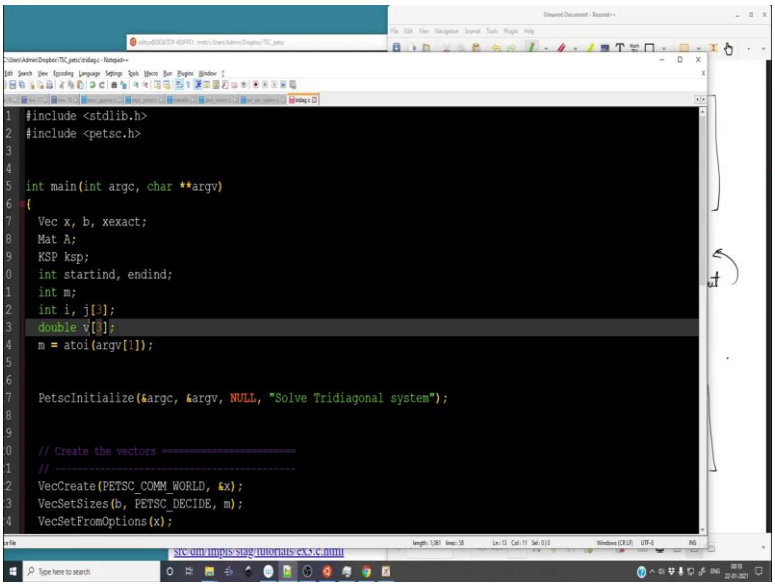
So, over here we will do if i = 0, then we must do something else if i = m - 1 that is the end case we must do something else we must do something alright. So, what are these some things. So, if i = 0 it is like a boundary condition. So, we are going to set two elements. So, let us say we set two elements that is for i = 0 which is the first row we are going to set this to be 2 and this to be -1 everything else is 0.

(Refer Slide Time: 40:37)



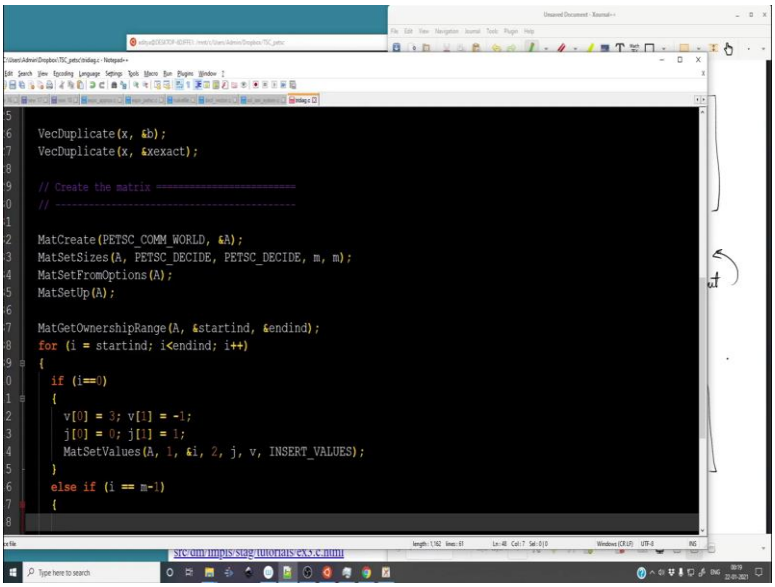
So, we must first create a vector which will sort of insert things into the matrix A. So, this is exactly in line with how we were inserting things.

(Refer Slide Time: 41:00)



So, we must define j as an array of 3 things. So, these 3 things we will define at runtime and we must define double b 3. So, we because at best we are going to insert 3 elements. So, when i = 0 then we must insert 3 and -1.

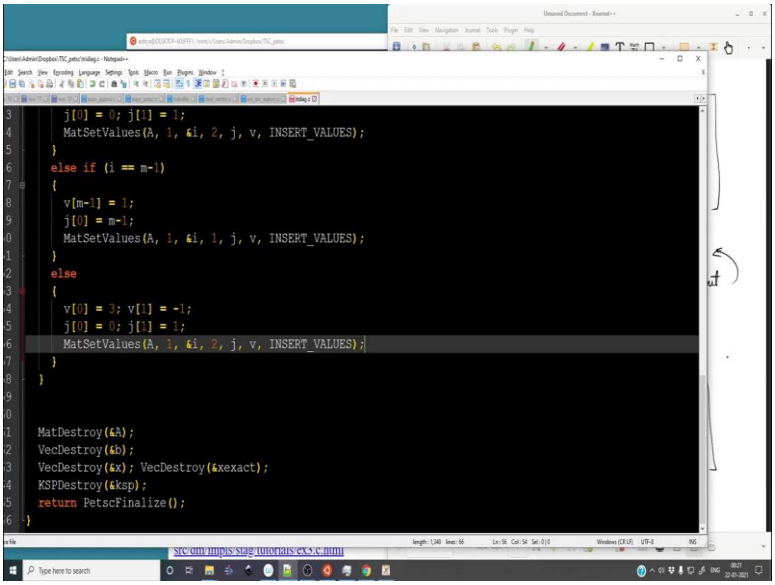
(Refer Slide Time: 41:31)



So, v 0 will be 3, v 1 will be -1. Similarly, j 0 that is the column index where we are going to insert something it is going to 0 and the column index were becoming sorry it should be j 1 where we are going to insert minus 1 has to be 1 essentially this location corresponds to j 0 this corresponding this location corresponds to j = 1.

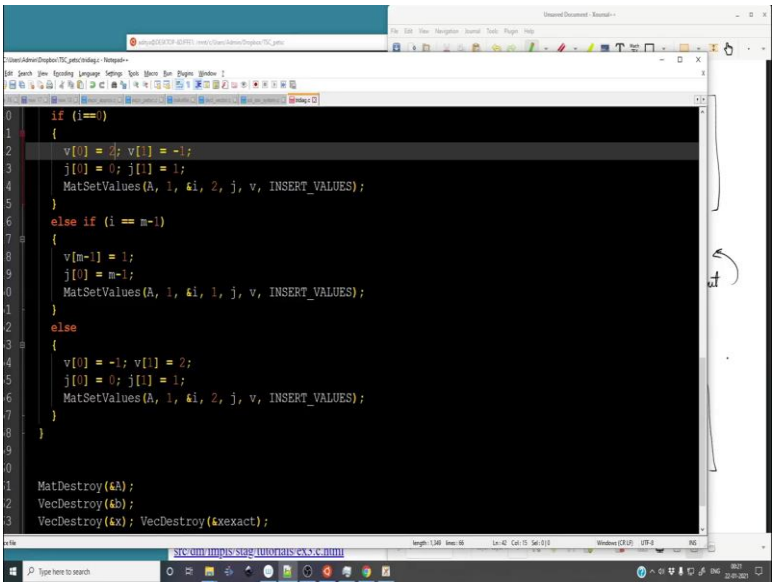
So, now that we have this we going to simply MatSetValues; MatSetValues it will be A. So, we have to insert in A we are going to insert 1 row we are going to pass the row number we are going to insert 2 values we are going to insert at locations j we are going to insert the values of v and we will insert values that is it then if we have the last element we can simply reuse this code.

(Refer Slide Time: 42:51)



So, this will be b m - 1 will be 3 and v m - 2 or rather let us put it as a dirichlet condition. So, the first one is more akin to a Neumann Boundary Condition. So, simply v m - 1 it will be equal to 1 and j 0 will be equal to n - 1 this is one of the last value and we are going to insert only one value else what we are going to do is we are going to we are going to reuse this.

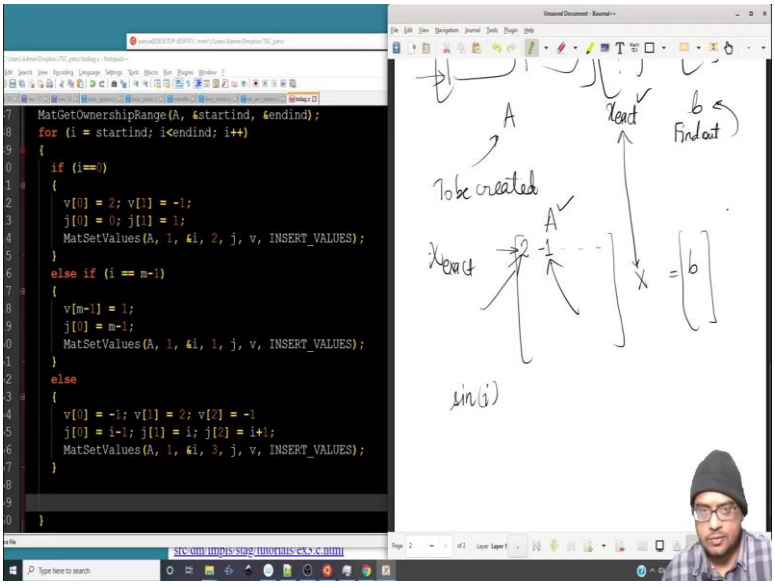
(Refer Slide Time: 44:04)



So, v 0 will be -1 v 1 will be 2 this has to be 2 and v 2 will be -1. So, j 0 will be i-1 j 1 will be i j 2 will be i + 1. So, we are going to insert 3 values this time so far so good we have not done something very remarkable, but we have just created the matrix A alright.

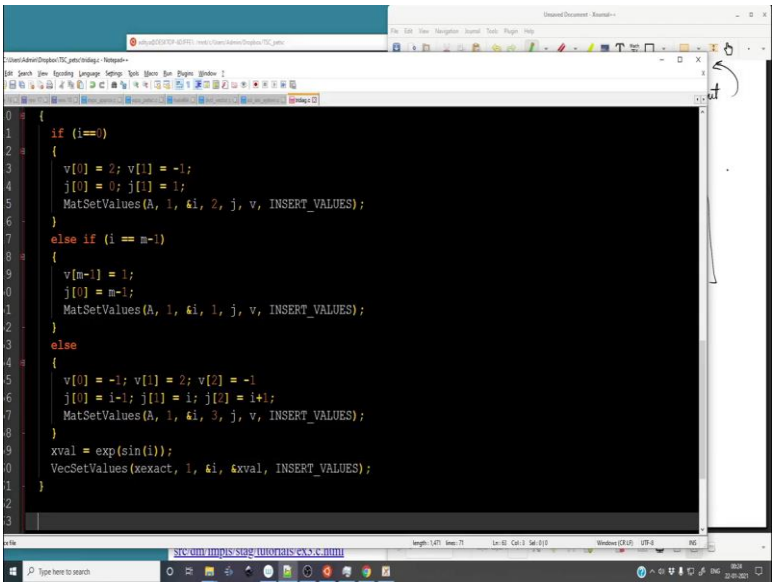
So, once we have made the matrix A let us proceed to set the exact solution as well. So, how do you do that? Well because we are doing this in a loop we can simply set values of the vector as well. So, we can say let us have the solution or something like x or let us have a periodic solution like something like sin i well it is periodic ok.

(Refer Slide Time: 45:39)



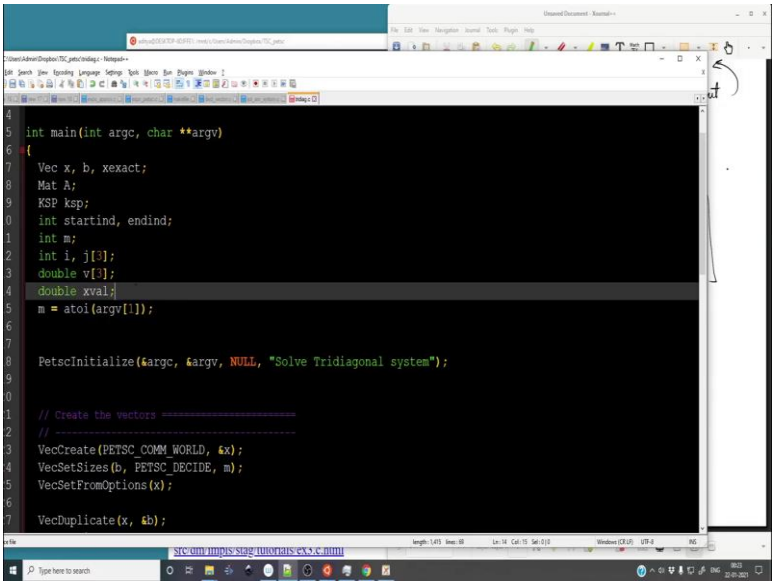
So, let me. So, all these exact values will be simply sin of the row number.

(Refer Slide Time: 45:56)



So, xval equal to. In fact, we can do an exponential of sin i.

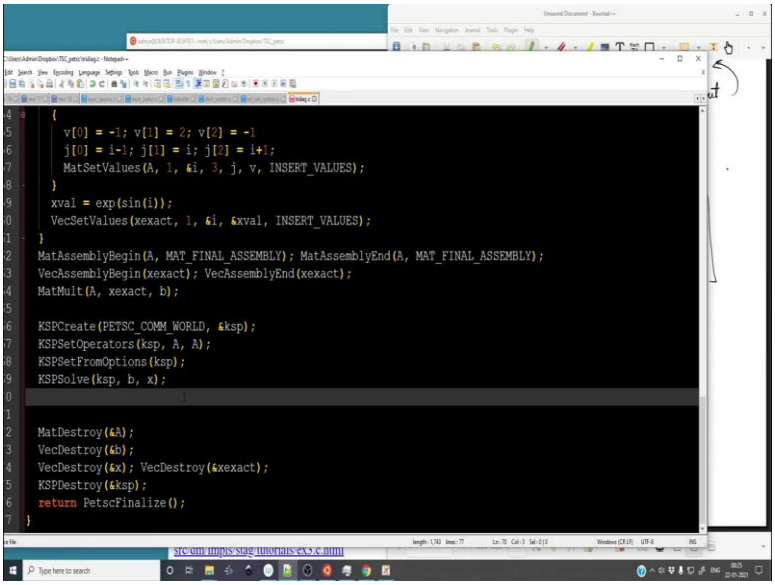
(Refer Slide Time: 46:15)



So, we need to define xval. So, we need to define double xval what else once we have the xval we can simply VecSetValues xexact we are going to assign this value to xexact we are going to put 1 value we have to get the address of i we are going to insert we have to give the address of what we want to insert and finally, we are going to insert values ok. So, this is how we can put the exact solution as well.

So, now, with this we can do the matrix assembly. So, let us go to this code and let us copy this snippet.

(Refer Slide Time: 47:09)

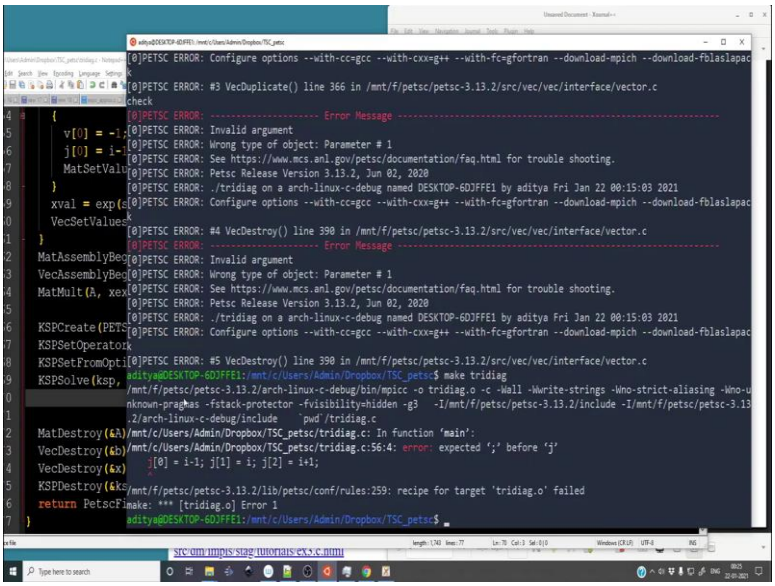


So, with the help of this we are going to begin the matrix assembly and we are going to finish it. Similarly, we have to do the vector assembly as well for xexact because we have constructed xexact also inside the loop. So, this will be xexact this will be xexact alright. So, VecAssembly is done.

Now, what we must do is find out what b will be. So, as discussed b will be actually a times x. So, how do we do that? So, we do a MatMult this is an inbuilt function a comma xexact and it will store the solution in b. So, it in it expects an input of the matrix the vector and the output vector. So, we are assigning b A times xexact great.

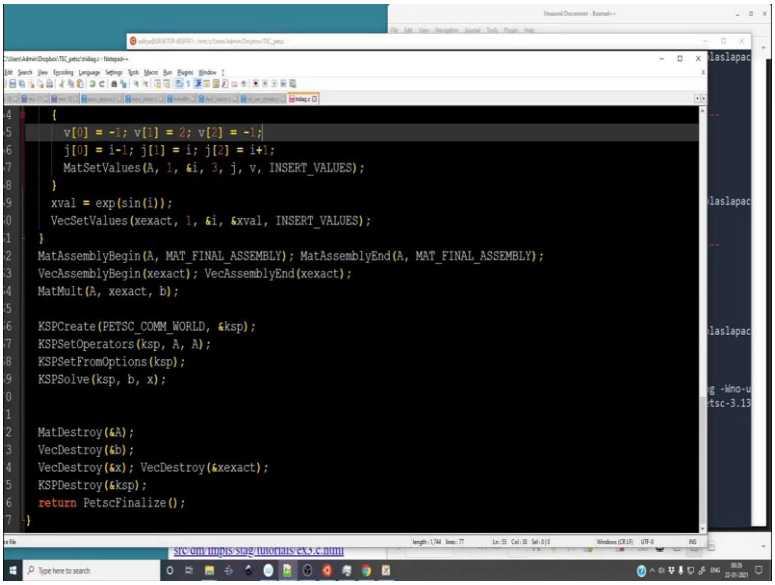
Now, what we are going to do is we are going to create the KSP object. So, we are going to reuse our code well it helps us save some time. So, ksp A, A and everything else remains the same alright.

(Refer Slide Time: 48:34)

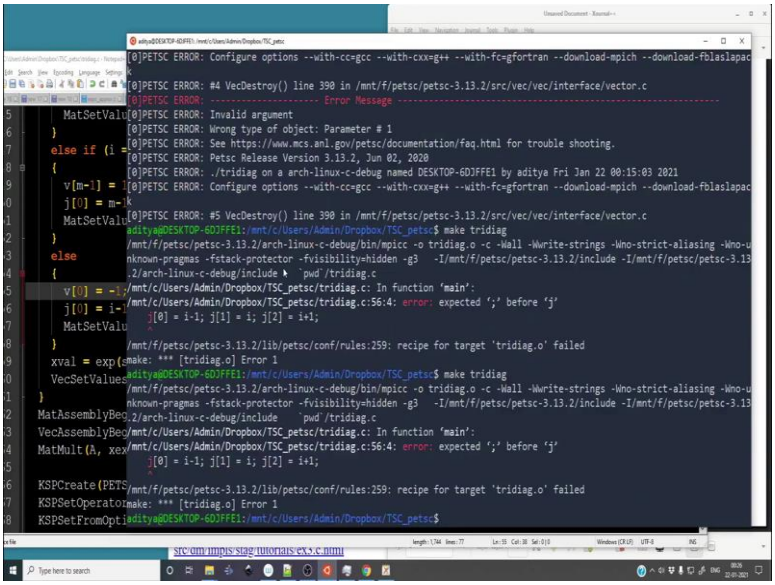


So, let us go to the terminal and see whether we have done some errors unexpected semicolon ok let us check we missed a semicolon over here ok.

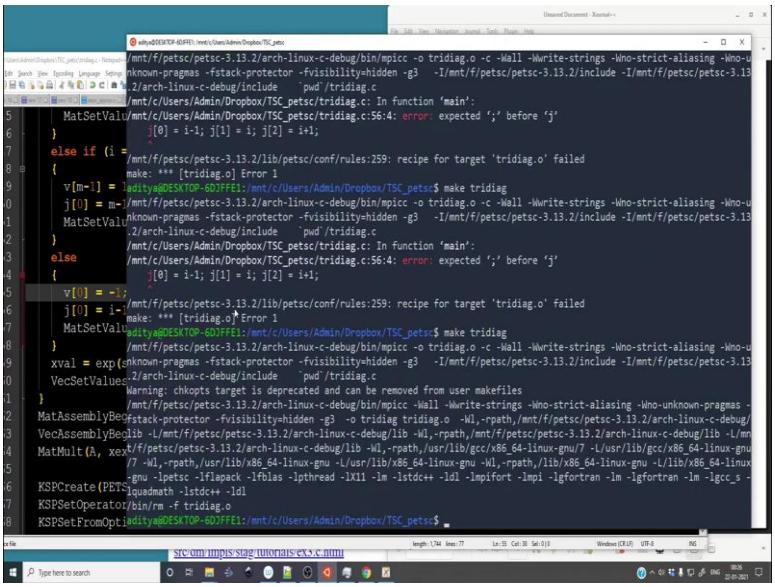
(Refer Slide Time: 48:47)



(Refer Slide Time: 48:52)

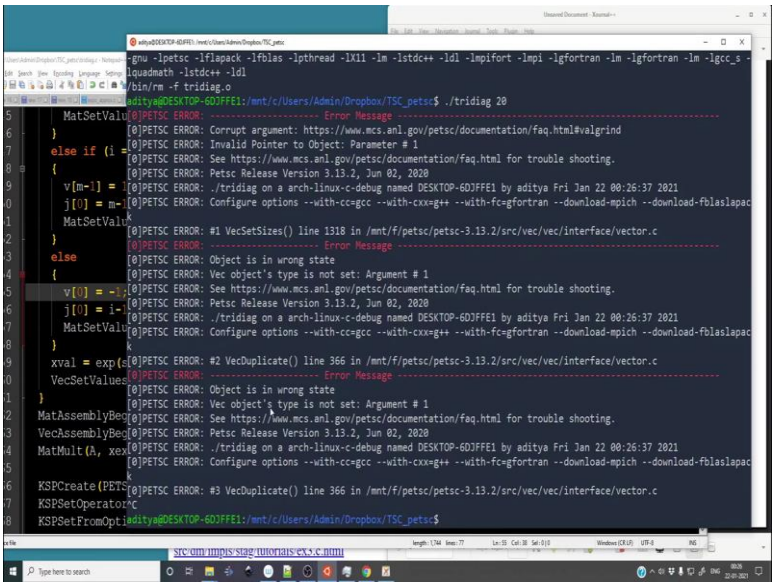


(Refer Slide Time: 49:20)



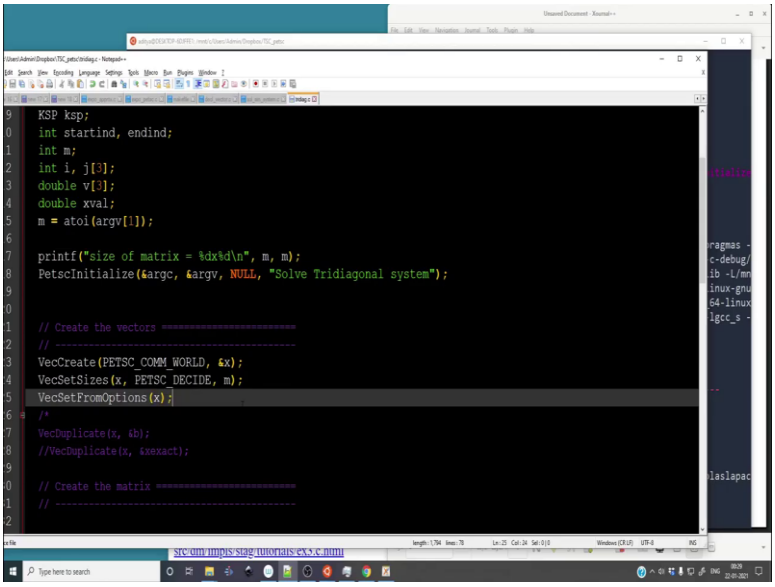
Let us see there is still some error and we missed a semicolon somewhere we have not saved the file do not make these mistakes.

(Refer Slide Time: 49:22)

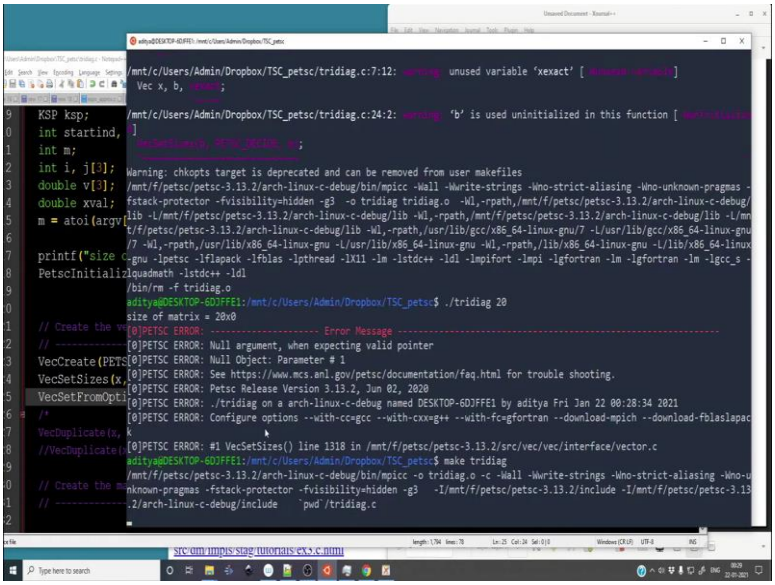


Well there appears to be errors and I am not sure and where it pointed. Let us see there appears to be an error in VecDuplicate xexact there it is there is the small error that was causing the segmentation for well that is the risk you run if you are copying snippets around

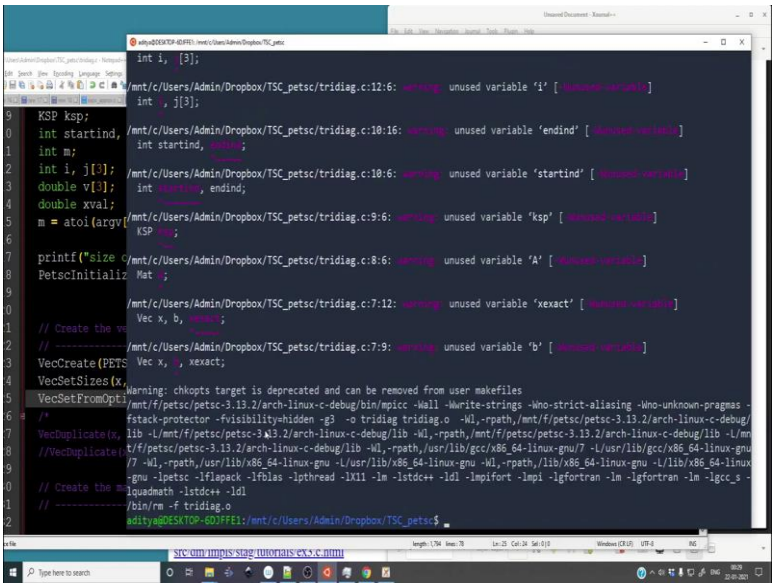
(Refer Slide Time: 50:20)



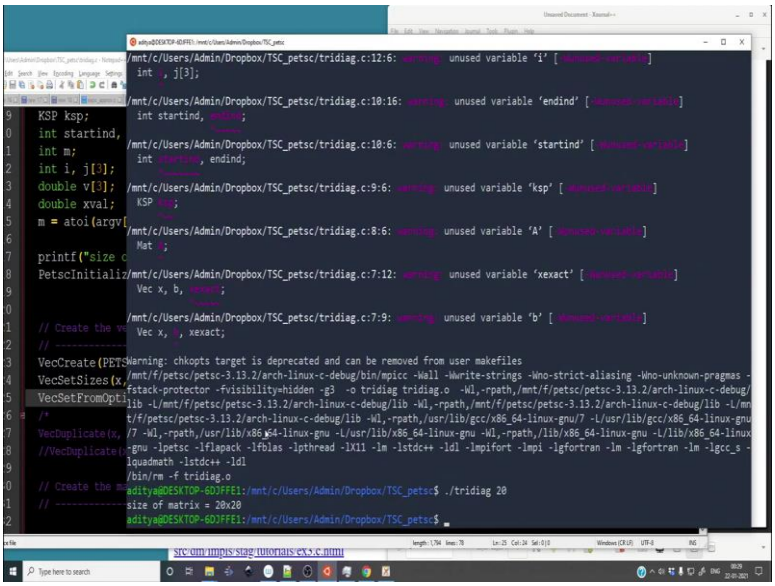
(Refer Slide Time: 50:22)



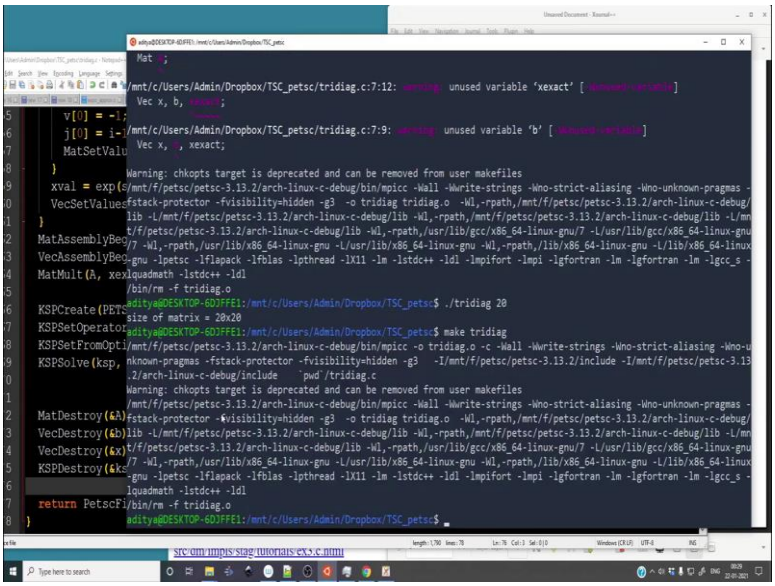
(Refer Slide Time: 50:25)



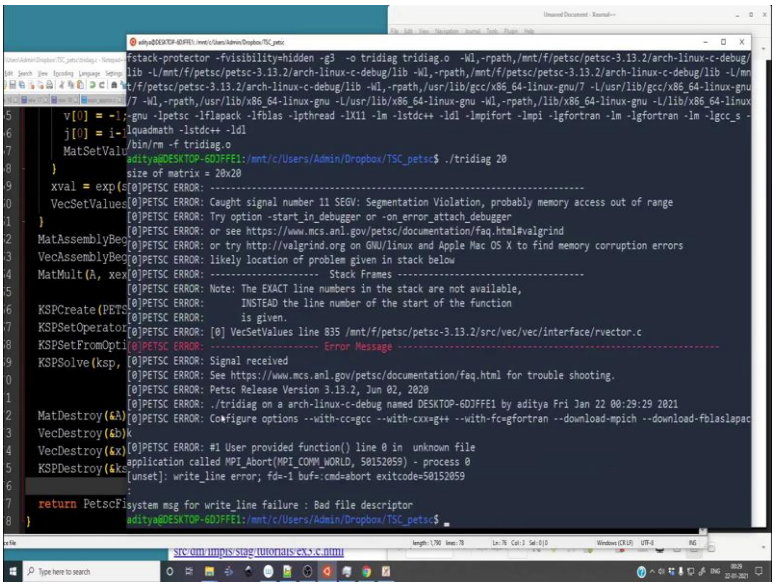
(Refer Slide Time: 50:27)



(Refer Slide Time: 50:40)

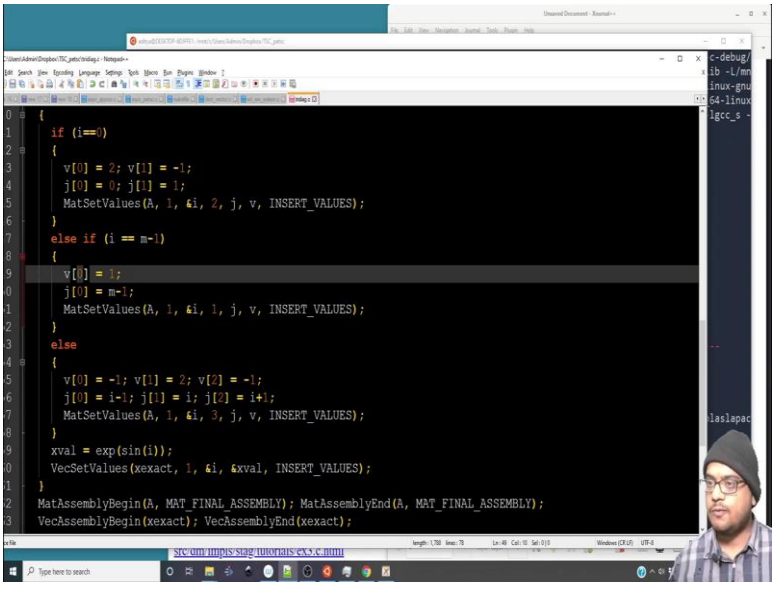


(Refer Slide Time: 50:43)

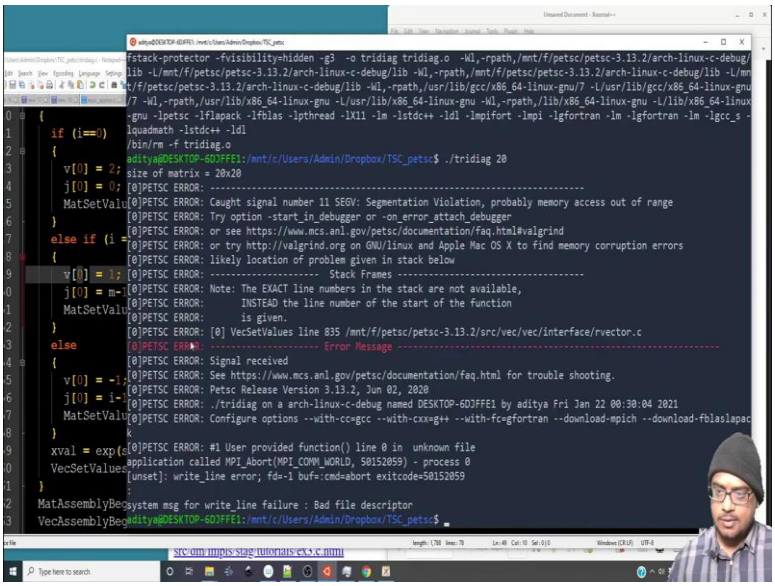


And well over time you learn to be proficient in tracking down the bugs, but the way I have tracked down this bug is not an efficient way of tracking it, but hey if it works it works it is not stupid well there is some error let us see; what is the error segmentation fault out of range let us see.

(Refer Slide Time: 51:12)

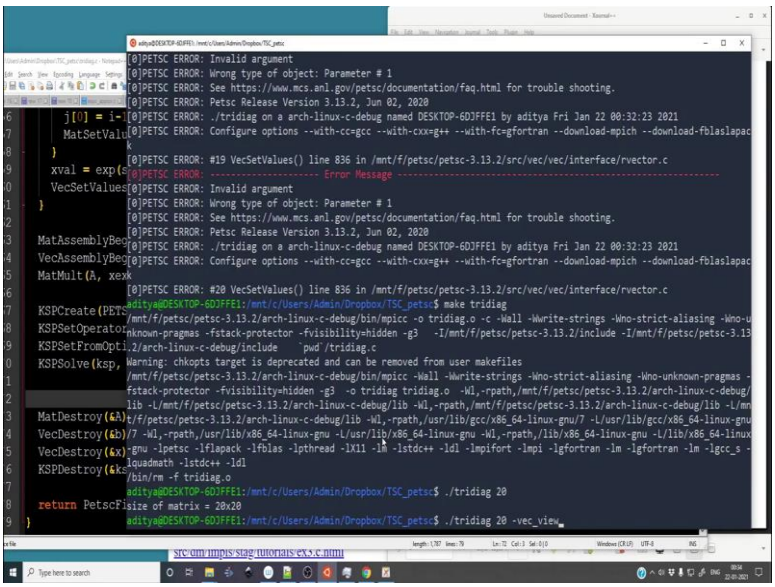


(Refer Slide Time: 51:16)



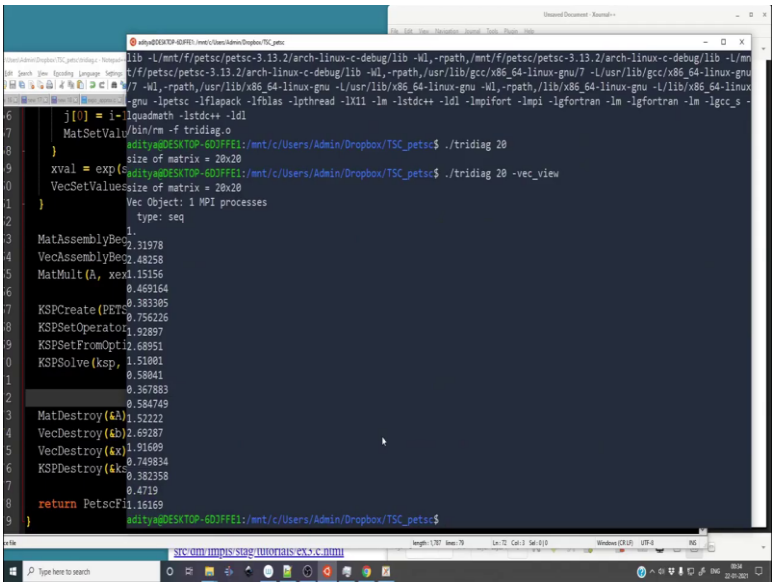
There is an very obvious error well still some error let us try to fix it well that was a very stupid mistake.

(Refer Slide Time: 51:50)

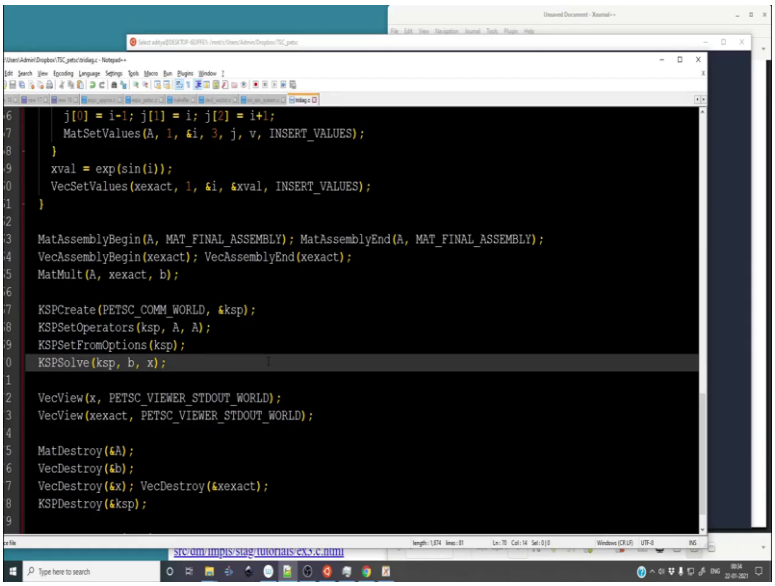


Well this kind of things to happen. So, everything's working nice and well. So, we have been able to solve till KSPSolve so far and now we would like to view the vectors. So, let us do that. So, - vec view ok. So, instead of doing a - vec view let us do vec view inside the code before destroying them we are going to do a vector of x xexact ok. Let us see let us recompile it and ok.

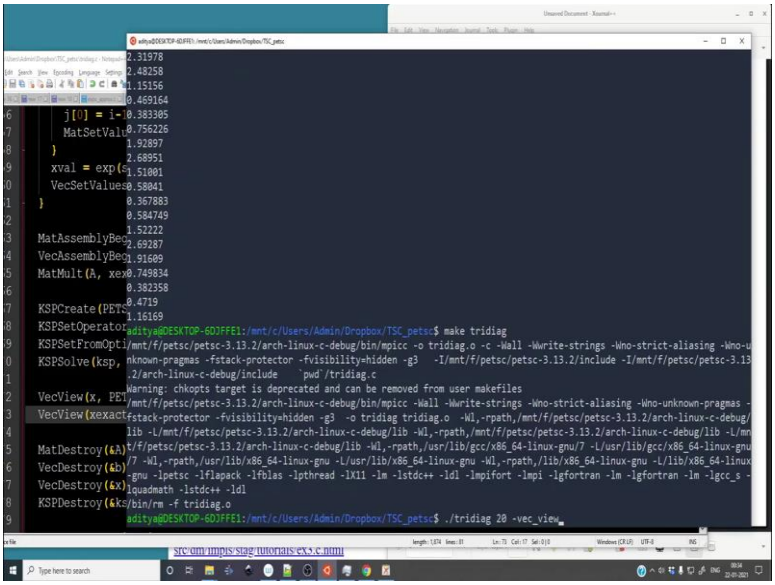
(Refer Slide Time: 52:18)



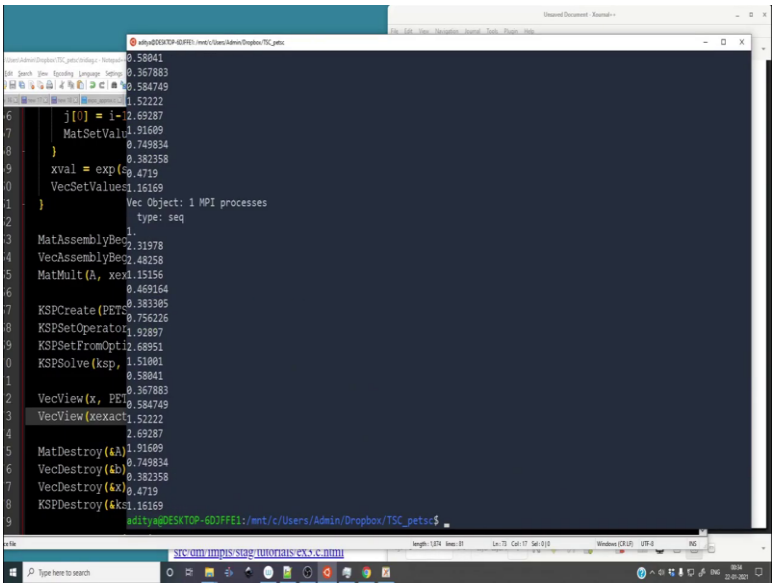
(Refer Slide Time: 52:30)



(Refer Slide Time: 52:36)

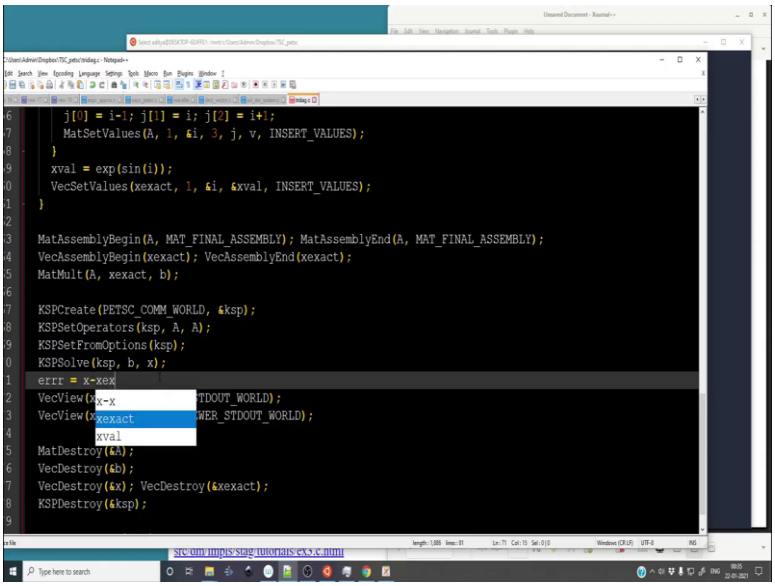


(Refer Slide Time: 52:42)



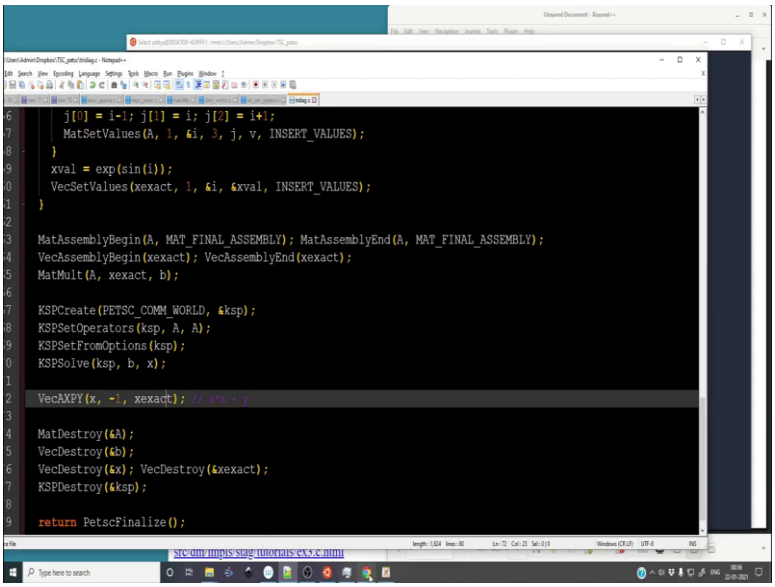
So, they do look close 1.16169 and well this is quite expected and as a matter of fact what we can do is figure out what the norm will be. So, the norm will be the difference between x and xexact ok. So, in order to find out the norm we must first have an operation x - xexact.

(Refer Slide Time: 53:23)



So, we cannot simply do something like what we will do in Python like error equal to x - xexact this is not allowed. So, we must do something which is quite something which resembles something which you would do in blas ok.

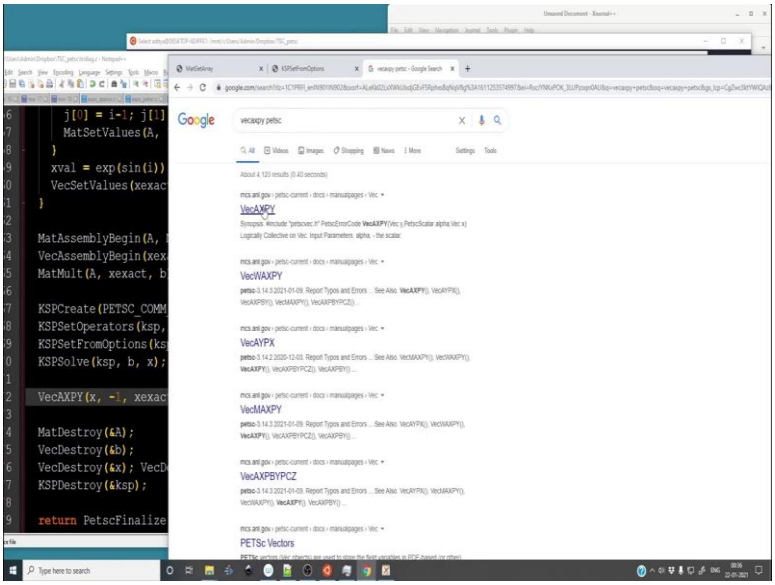
(Refer Slide Time: 53:44)



So, you would do what is called as VecAX + Y. So, what this function does is it does A * X + Y where X and Y are two vectors and A is a scalar ok. So, it does a * x + y so, but here a has to be -1, x and y have to be x and xexact respectively. So, that essentially

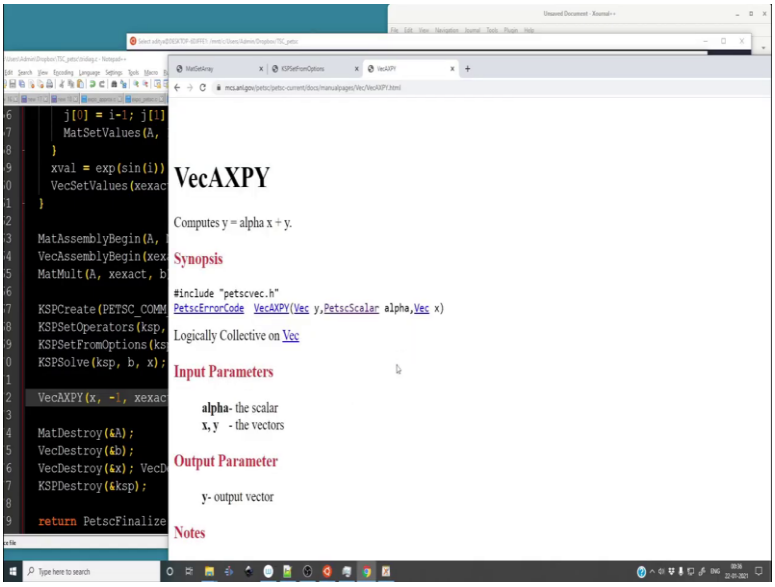
mimics what you want to do and the only caveat is once this operation is done it will overwrite one of the vectors.

(Refer Slide Time: 54:31)



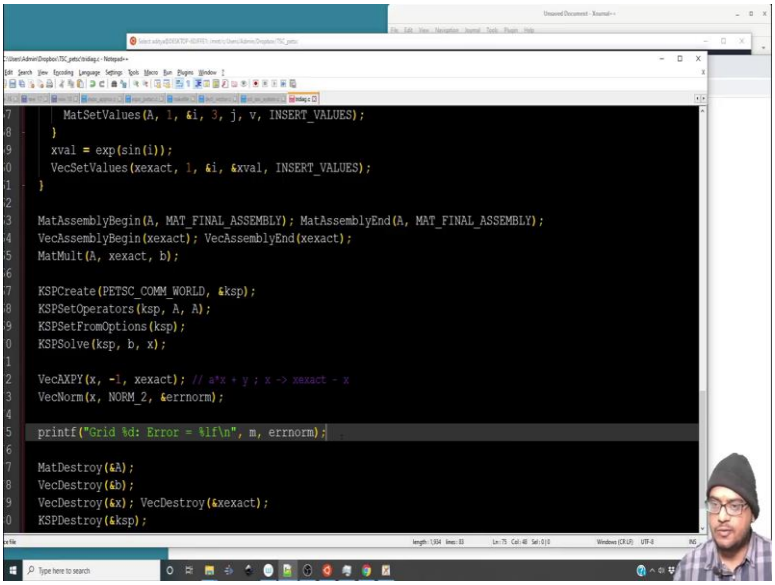
So, let me just show you the function reference and it will tell you exactly what it will overwrite.

(Refer Slide Time: 54:37)



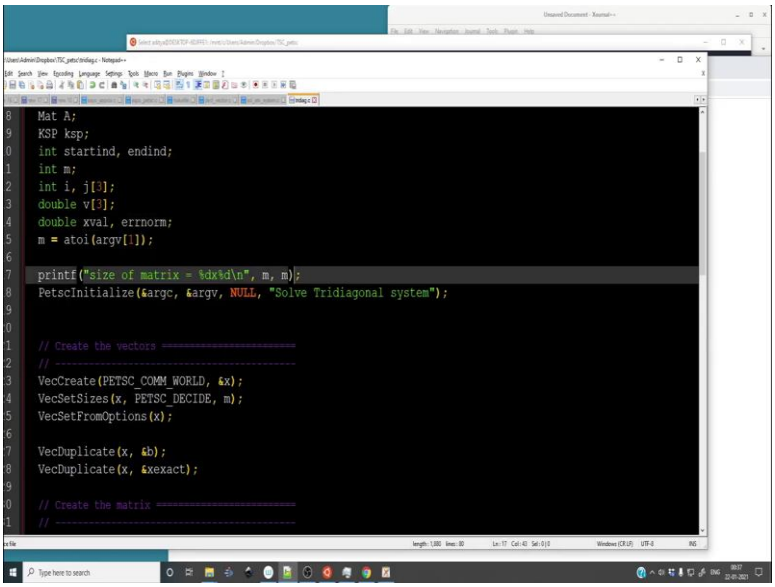
So, it will do y = a x + y ok. So, the first vector gets overwritten.

(Refer Slide Time: 55:00)



So, over here x will get overwritten with the x will be overwritten by xexact - x ok. This thing will cause x to be overwritten by xexact - x and that is ok we just wanted to figure out the norm finally, we can do VecNorm that is the norm of the vector x because that contains the error now and the NORM has to be a l 2 NORM 2 and we will store it inside error of norm. So, error of norm has to be declared as a double.

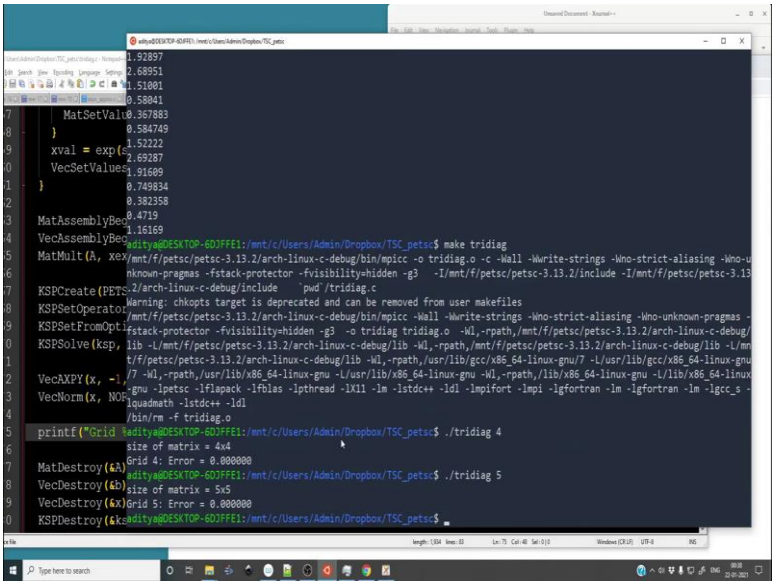
(Refer Slide Time: 55:39)



So, let us declare it as a double ok. So, we have declared it as a norm. So, finally, before destroying everything we can print out printf Error equal to percentage lf errnorm. In fact, we can write down the grid size as well. So, Grid percentage d and error is this we

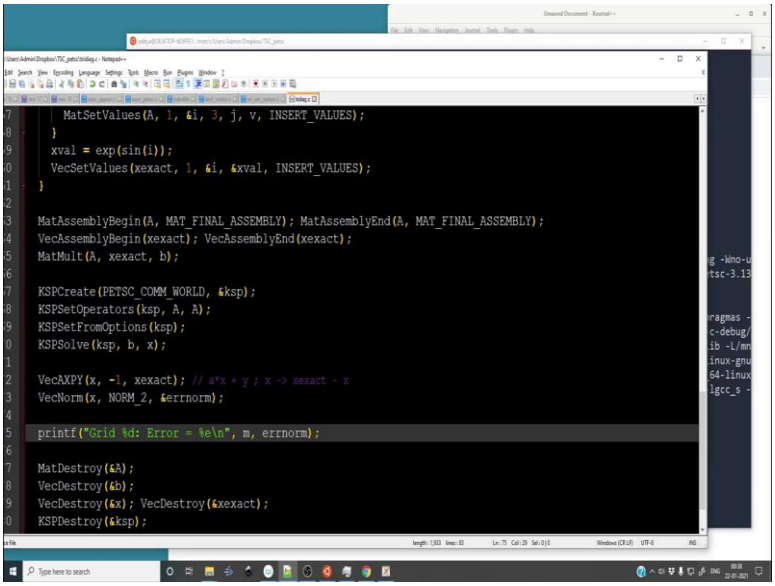
will print m and error norm. So, it will help us evaluate how the error reduces as the number of grids increases.

(Refer Slide Time: 56:30)

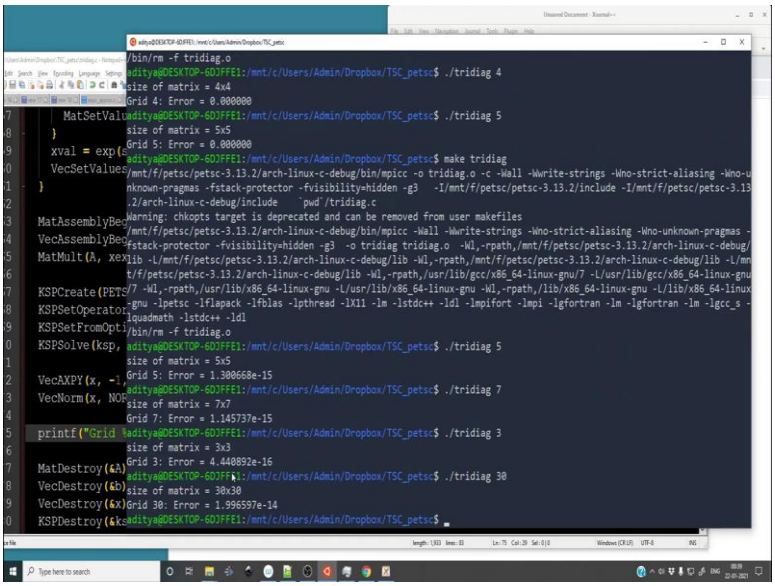


So, let us recompile let us do it with 4 grids, 5 grids well it shows a 0 error let us say percentage e.

(Refer Slide Time: 56:48)

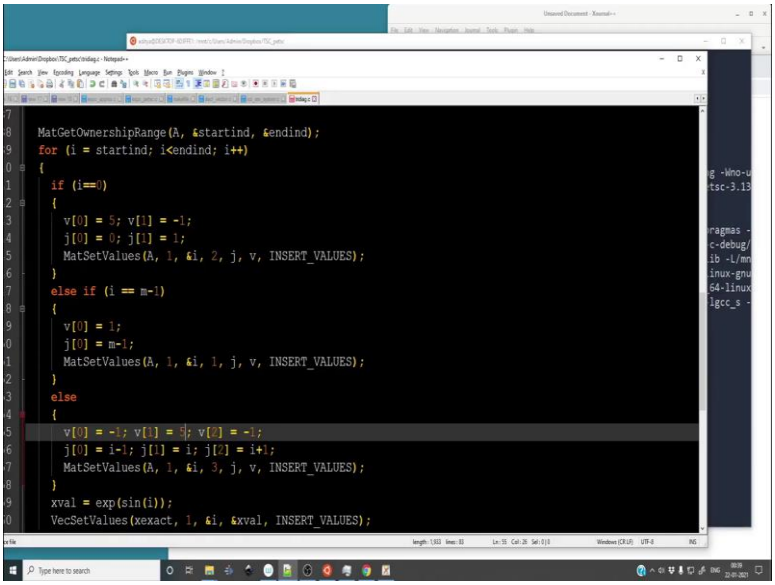


(Refer Slide Time: 56:52)



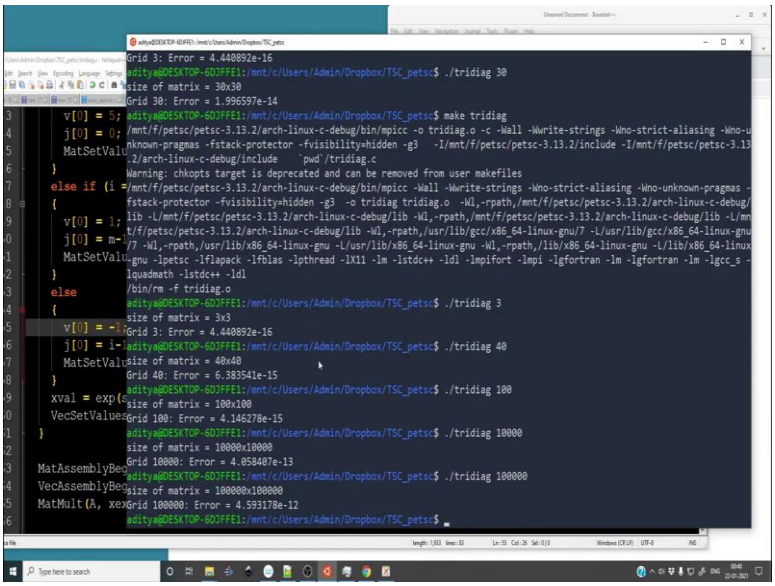
So, the percentage e will have a much better output. So, that is 10^{-15} e^{-15}, e^{-16}, e^{-14}. So, anyway. So, this is how you would go about doing this. So, in this example we ran into some errors, but those were just errors out of sheer haste, but I hope this program gives you a basic structure of how to programmatically create the vectors that you so desire. In fact, let me make the diagonal a bit heavy.

(Refer Slide Time: 57:35)



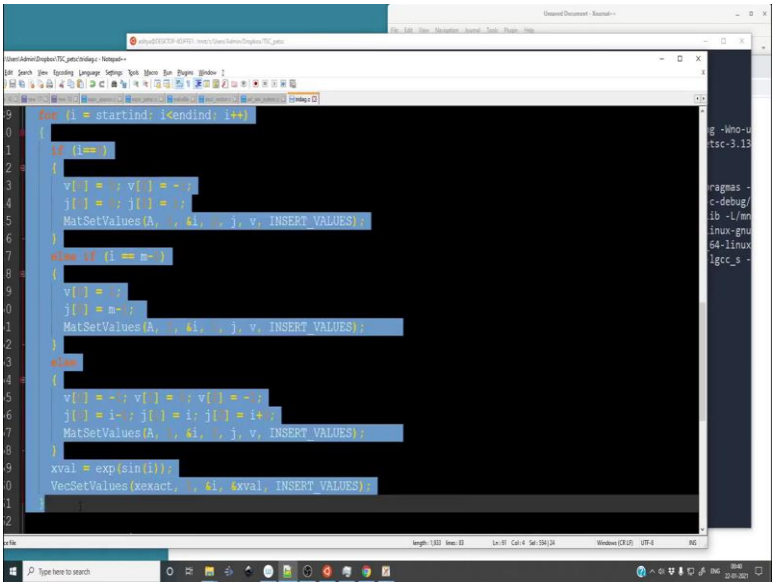
So, that making the diagonal heavy would render a larger error let us see let us see if that happens. No.

(Refer Slide Time: 57:47)



Well, you can make very very large matrices, but anyway yeah. So, you can define the matrix like this we used a very synthetic example to set up the problem after setting up the problem we solved with the problem using the KSP object we found out the norm we printed out the error ok.

(Refer Slide Time: 58:36)

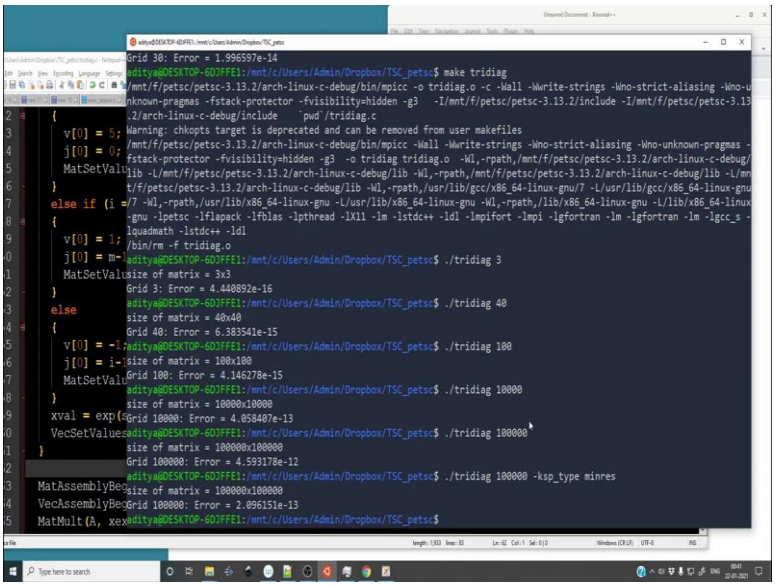


So, in what in the next few lectures we will use this kind of program to sort of solve some PDEs. The rest of the code structure will remain more or less the same and I sincerely request you to have a look at this program write your own program. So, that

you do not do any of the hasty errors that we did, but at least or at least the structure should be clear I mean the syntax and all they can come and go, but it should be clear in your head what you are trying to do because once that is clear you will always find out the syntax to do it.

So, we are right doing it in C, tomorrow maybe you will you will be doing it in Julia and after tomorrow we do it in doing it in some new language that will come out, but the logic will always remain the same. So, with this we end this lecture where we have taken a look at solving a system solving a tri-diagonal system this is tantamount to a tri-diagonal system.

(Refer Slide Time: 59:50)



Go ahead and try out different kinds of solvers by changing the type ok, you can change the different you can change it to different types and find out the solution and with this we end it over here and I will see you next time with a new lecture until then have a nice time bye.