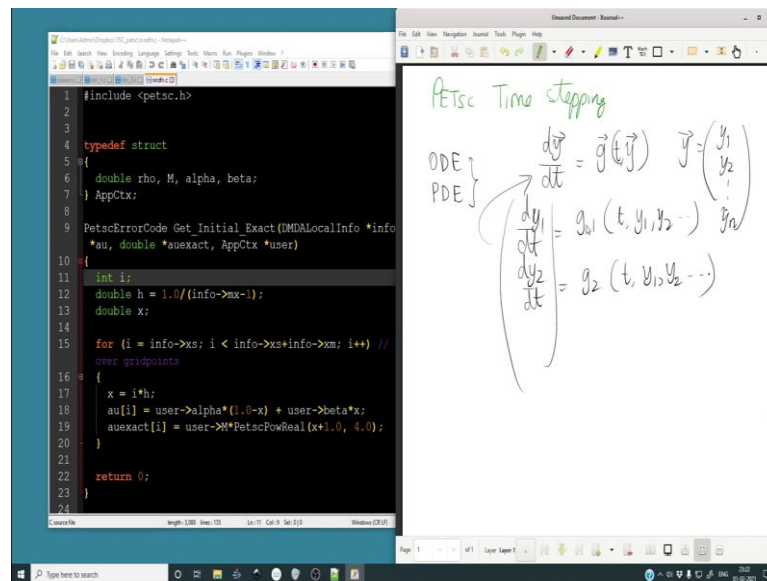


Tools in Scientific Computing
Prof. Aditya Bandopadhyay
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture - 36
Time Stepping in PETSc

[noise]

(Refer Slide Time: 00:27)



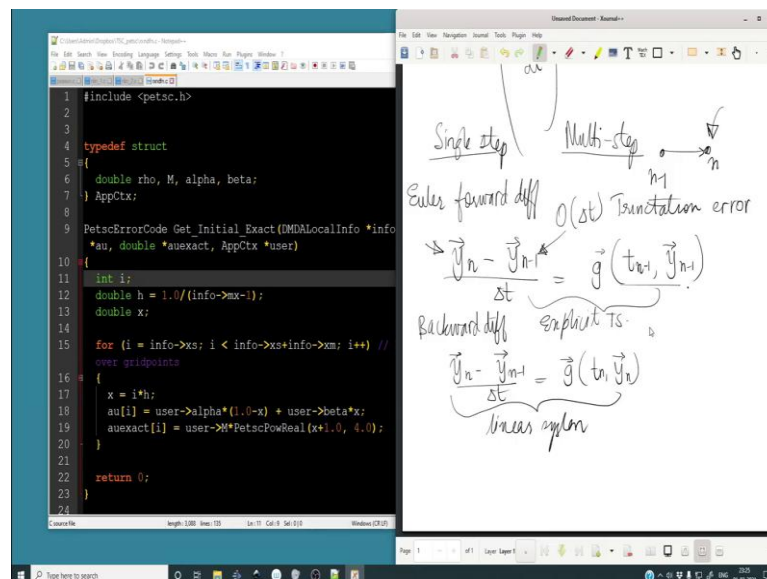
Hi everyone in this lecture we are going to study about Time Stepping. In particular we are going to look at how we can achieve not just linear, but also non-linear time stepping in PETSC. So, like everything else time stepping is also encoded inside an object a time stepping object and as you can imagine there is a host of algorithms. But what really do we mean by time stepping?

So, by time stepping we refer to marching an equation in time. So, it could be for an ordinary differential equation or partial differential equation which has an unsteady term and so on. So, basically it can happen for both ordinary and partial differential equations and so if you have an ordinary differential equation of the form $\frac{d\vec{y}}{dt} = \vec{g}(t, \vec{y})$ right.

Where \vec{g} is a sort of function, so I mean $\vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$. So, $\frac{dy_1}{dt} = g_1(t, y_1, y_2, \dots)$,

$\frac{dy_2}{dt} = g_2(t, y_1, y_2, \dots)$ and so obviously you can sort of assemble all this in the form of this vector equation right. So, then how do you actually achieve time stepping? I mean there are various algorithms that exist.

(Refer Slide Time: 02:45)



So, algorithms can be broken up into Single step algorithms or Multi step algorithms. The single step algorithms are usually much faster because they involve only maybe one set of function evaluations, multistep methods will involve usually function evaluations which will have multiple intermediate steps or it may drop on multiples time steps at a time.

So, that requires more computation. So, single step so let us look at the simplest one that is the Euler time stepping so forward difference. So, we can say y at n - y I mean let us

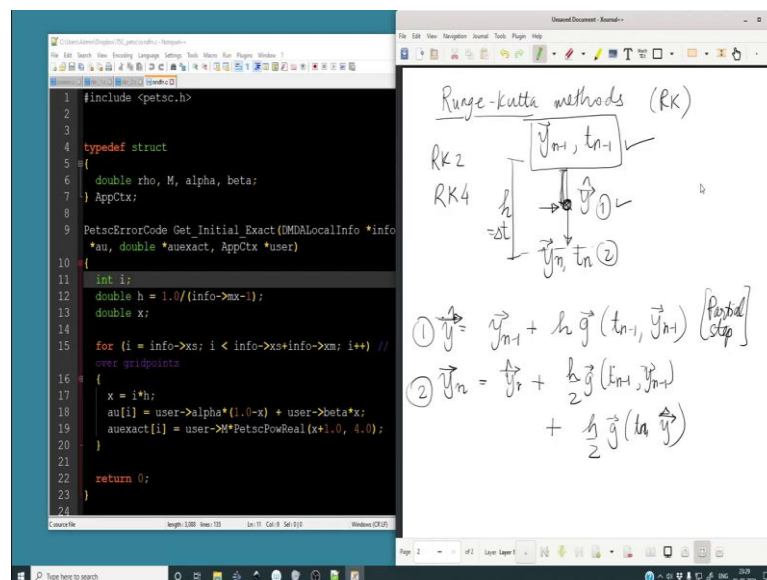
write it in the subscript form, so $\frac{\vec{y}_n - \vec{y}_{n-1}}{\Delta t} = \vec{g}(t_{n-1}, \vec{y}_{n-1})$. This is how you would write

forward difference approximation for y and you can obviously integrate this I mean not integrate this, but easily step this in time.

But this method has order Δt truncation error because this as you can imagine is a vector series expansion about \vec{y}_{n-1} . Similarly, you can have a backward difference method where instead of sort of going from n-1 to n you can also write everything in terms of n. But there will be a prime difference and I mean you have most likely gone through this in your first year or whenever you have come across this this will instead be $\vec{g}(t_n, \vec{y}_n)$.

And this usually culminates in the form of linear system in \vec{y}_n ok. This thing is usually an explicit time stepping, so there is no matrix inversion as such. Well apart from these methods both of these methods suffer from order Δt truncation error.

(Refer Slide Time: 05:36)



But we can also have the higher order methods and the most famous higher order method is called as a Runge-Kutta methods and essentially apologize for incorrect pronunciation of the name I am not sure what this is called I am pretty sure it is called Runge, Runge ok. I am not sure about that anymore, but anyway it does not matter what you call them we call them the RK methods in short and they are widely used ok.

So, there are various RK methods, there are RK2 methods which is order Δt^2 accurate this the very famous RK4 which is 4th order accurate and there are other higher order methods as well you can actually construct various methods as you go. So, let me write down the second order RK4.

So, suppose you have the information about y_{n-1} step at t_{n-1} time and your objective is to time step from this to y_n which is at t_n . So, we declare an intermediate time \hat{y} right. So, let $\hat{y} = \bar{y}_{n-1} + h\bar{g}(t_{n-1}, \bar{y}_{n-1})$. Where h is the time stepping is essentially Δt we have borrowed the same literature from spatial derivatives over here. So, far it should not create any confusion at all.

And so essentially you are writing at an intermediate step you are sort of doing a partial step towards the final goal. So, this is that partial step ok. So, beyond this we have $\bar{y}_n = \hat{y} + \frac{h}{2}\bar{g}(t_{n-1}, \bar{y}_{n-1}) + \frac{h}{2}\bar{g}(t_n, \hat{y})$ ok. So, it is like a partial step you if you first evaluate this. So, this is that first step and then you sort of take an average of the evaluation at this step and this step and you take an average and then you perform the final time stepping ok.

So, this you go in 2 steps and this is order Δt^2 accurate the truncation order result of the order of Δt^2 and you can easily prove this I have given the form over here you just do a back calculation using the Taylor series. The most famous form that is the RK4 which is the workhorse of solving ordinary differential equations, because it is quite robust and it gives us a great deal of accuracy with very few number of computations.

(Refer Slide Time: 09:25)

```

1 #include <petsc.h>
2
3
4 typedef struct
5 {
6     double rho, M, alpha, beta;
7 } AppCtx;
8
9 PetscErrorCode Get_Initial_Exact (DMDataLocalInfo *info
10 *su, double *auexact, AppCtx *user)
11 {
12     int i;
13     double h = 1.0/(info->mx-1);
14     double x;
15     for (i = info->xs; i < info->xs+info->xm; i++) //
16         over gridpoints
17     {
18         x = i*h;
19         au[i] = user->alpha*(1.0-x) + user->beta*x;
20         auexact[i] = user->M*PetscPowReal(x+1.0, 4.0);
21     }
22     return 0;
23 }
24

```

① $\hat{y} = \bar{y}_{n-1} + h\bar{g}(t_{n-1}, \bar{y}_{n-1})$ (Partial Step)

② $\bar{y}_n = \hat{y} + \frac{h}{2}\bar{g}(t_{n-1}, \bar{y}_{n-1}) + \frac{h}{2}\bar{g}(t_n, \hat{y})$

RK4

$k_1 = f(t_n, \bar{y}_n)$

$k_2 = f(t_n + \frac{h}{2}, \bar{y}_n + \frac{h}{2}k_1)$

$k_3 = f(t_n + \frac{h}{2}, \bar{y}_n + \frac{h}{2}k_2)$

$k_4 = f(t_n, \bar{y}_n + hk_3)$

$\bar{y}_n = \bar{y}_{n-1} + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

So, the RK4 method may be summarized as going from t_{n-1}, \vec{y}_{n-1} to t_n, \vec{y}_n in multiple steps. So, the first step is to write $K_1 = f(t_{n-1}, y_{n-1})$, then you can write $K_2 = f(t_{n-1} + \frac{h}{2}, y_{n-1} + \frac{h}{2}K_1)$, $K_3 = f(t_{n-1} + \frac{h}{2}, y_{n-1} + \frac{h}{2}K_2)$, $K_4 = f(t_n, y_n + hK_3)$. And finally after finding out these 4 intermediate steps, so this K_1, K_2, K_3, K_4 you finally write $\vec{y}_n = \left(\frac{1}{6}K_1 + \frac{1}{3}K_2 + \frac{1}{3}K_3 + \frac{1}{6}K_4\right) + \vec{y}_{n-1}$.

(Refer Slide Time: 11:26)

The image shows a code editor on the left and a handwritten slide on the right. The code editor displays a C++ implementation of the RK4 method using PETSc. The handwritten slide contains the following content:

① $\vec{y}_1 = \vec{y}_{n-1} + h \vec{g}(t_{n-1}, \vec{y}_{n-1})$ (Partial Step)

② $\vec{y}_n = \vec{y}_1 + \frac{h}{2} \vec{g}(t_{n-1}, \vec{y}_{n-1}) + \frac{h}{2} \vec{g}(t_n, \vec{y}_1)$

rk4

$K_1 = f(t_{n-1}, y_{n-1})$
 $K_2 = f(t_{n-1} + \frac{h}{2}, y_{n-1} + \frac{h}{2}K_1)$
 $K_3 = f(t_{n-1} + \frac{h}{2}, y_{n-1} + \frac{h}{2}K_2)$
 $K_4 = f(t_n, y_n + hK_3)$

$\vec{y}_n = \frac{1}{6}K_1 + \frac{1}{3}K_2 + \frac{1}{3}K_3 + \frac{1}{6}K_4 + \vec{y}_{n-1}$

(Refer Slide Time: 12:03)

The image shows a code editor on the left and a handwritten slide on the right. The code editor displays a C++ implementation of the BDF method using PETSc. The handwritten slide contains the following content:

2nd order backward differentiation formula
BDF

$\vec{y}_n = \frac{4}{3}\vec{y}_{n-1} - \frac{1}{3}\vec{y}_{n-2} + \frac{2}{3}h \vec{g}(t_n, y_n)$

$\frac{dy}{dt} = \bar{A} \vec{y} + \vec{f}(t)$

$\bar{A} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ $\vec{f} = \begin{pmatrix} 0 \\ t \end{pmatrix}$

$\frac{dy_0}{dt} = y_1 + 0$ $y_0 = t - \sin t$
 $\frac{dy_1}{dt} = -y_0 + t$ $y_1 = 1 - \cos t$

So, this is how you find out using 4 intermediate steps what the Runge Kutta terms will be. So, apart from this these Runge Kutta methods there is also multi step methods like the very famous 2nd order backward differentiation formula and you will find many of these things in commercial solvers like mat lab are not I mean you will find this in mat lab as well, but in comsol fluent and all these things as well.

So, the BDF solver so $\bar{y}_n = \frac{4}{3}\bar{y}_{n-1} - \frac{1}{3}\bar{y}_{n-2} + \frac{2}{3}h\bar{g}(t_n, y_n)$ ok. So, this is order Δt^2 so it is a multi step method ok.

So, there can also be various time stepping methods which make partial use of such kinds of backward differentiation formulae and forward stepping. So, typically we have to remember that implicit methods require more time, because you have to in fact solve for an algebraic system. Whereas, explicit methods do not suffer such a kind of drawback.

So, now let us proceed to solve a very simple ordinary differential equation with the help of the PETSc time stepping. So, let us first write down a governing equation that we intend to solve. So, $\frac{d\bar{y}}{dt} = \bar{A}(\bar{y}) + \bar{f}(t)$, where $\bar{A} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$, $\bar{f} = \begin{pmatrix} 0 \\ t \end{pmatrix}$, so this is a driven

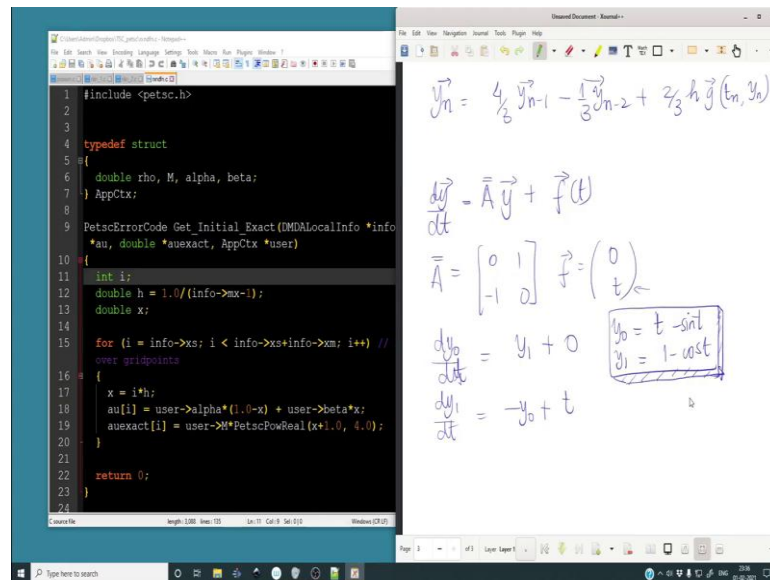
term. So, what do we have $\frac{dy_0}{dt} = y_1 + 0$ and $\frac{dy_1}{dt} = -y_0 + t$.

So, these are the 2 equations and yeah so let us see how we can solve this using PETSc, but before that let me just point out that $y_0 = t - \sin t$ and $y_1 = 1 - \cos t$ they are solutions

and we can easily verify this. So, $\frac{dy_0}{dt} = 1 - \cos t = y_1$, $\frac{dy_1}{dt} = \sin t$. So, $-y_0 = -t + \sin t$.

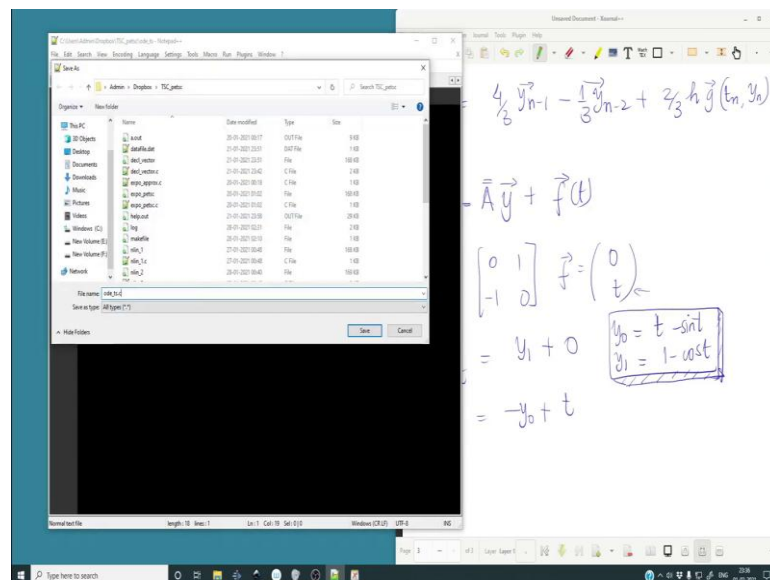
So, $-t + \sin t + t$ so the t cancels out and you get this.

(Refer Slide Time: 15:23)



So, this is what the solution is and it will be useful for us, because then we can compare whatever we obtain using PETSc numerically we can easily compare that alright. So, let us proceed towards writing the program.

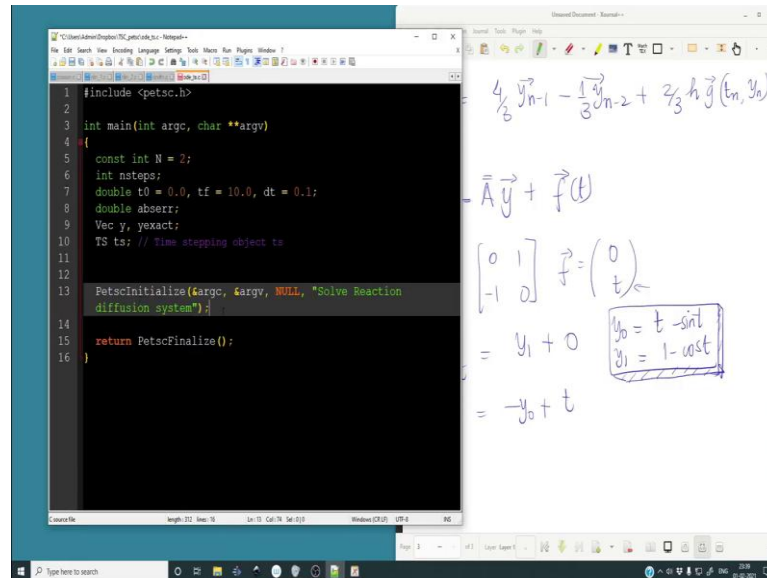
(Refer Slide Time: 15:54)



So, let me create a new file alright. So, first things first we must create a time step object the time stepping object.

And so let us go line by line. So, the first thing we will do `#include <petsc.h>` let me save this file. So, let me save it as `ode_ts` oops, I forgot to declare this the extension of the file as `.c` alright.

(Refer Slide Time: 16:14)



So, we have declared it as a `.c` file `#include <petsc.h>`, then let us write the main so `int main(int argc, char **argv) return PetscFinalize()` alright.

So, over here we are going to define the number of equations that we are going to solve. So, `int N = 2` and actually because `N` is not going to change or rather we do not want `N` to be changed we can declare it always as a `const int 2`. Alright we are going to need the number of steps that we want to integrate the equations. So, `int N steps` we also are going to need the initial time.

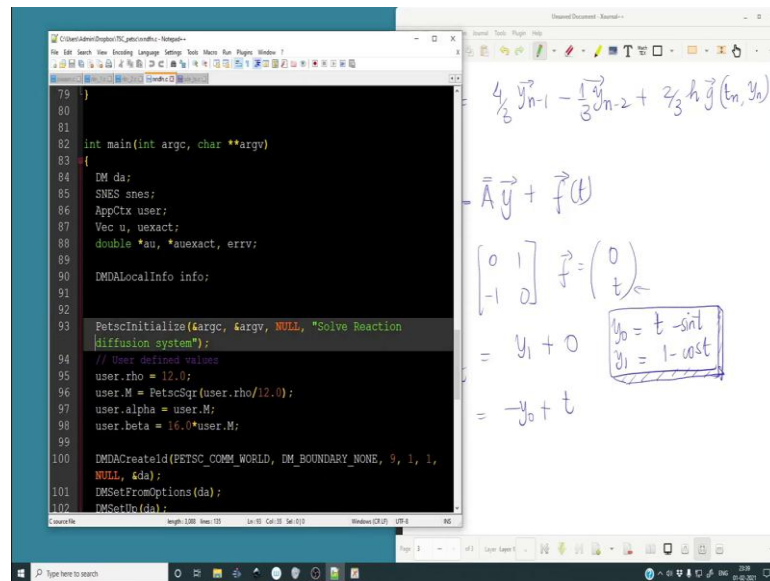
So, `double t0 = 0.0, tf = 10.0` the final time this has to be a comma and finally the time stepping variable that the `dt` is going to be `0.01` we are also going to need a variable to store the error between the analytical solution and the numerical solution find out using time stepping. So, we are going to declare the error so we `abs error` ok.

So, now apart from this we are going to need a vector which is going to sort of keep on updating depending on the time stepping and we are going to need a vector which will store the exact sort of values that we have from the analytical solution, just to make a comparison on how everything looks like alright. So, we are going to need `Vec y, yexact`

and apart from this we are going to need a time stepping object. so TS ts. So, this is the time stepping object alright.

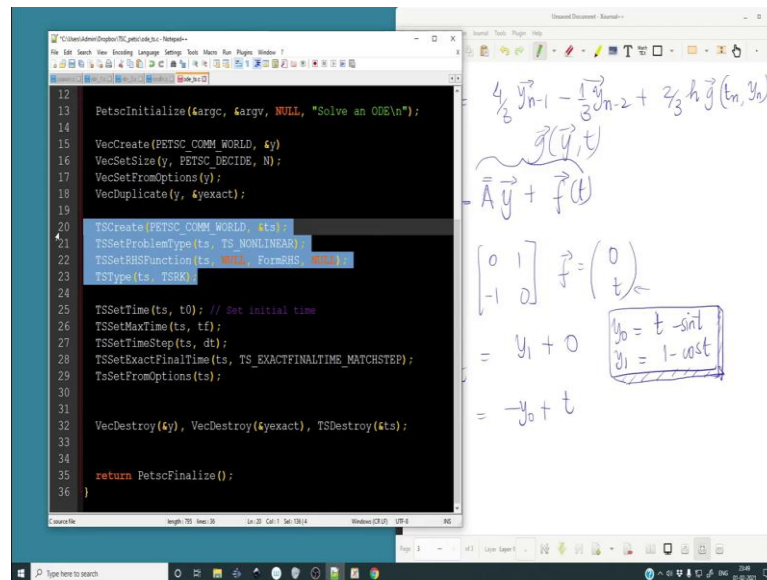
So, with the help of this we can perform the PetscInitialize. So, in order to have a PetscInitialize we will need certain arguments.

(Refer Slide Time: 19:03)



So, let me borrow that from one of our previous code and the reason why we can borrow it because nothing it really changes across the different programs in this particular line at least and instead of having a help string that solve the reaction diffusion system we will say that it is to solve an ode alright.

(Refer Slide Time: 19:20)



```
12 PetscInitialize(&argc, &argv, NULL, "Solve an ODE\n");
13
14
15 VecCreate(PETSC_COMM_WORLD, &y);
16 VecSetSize(y, PETSC_DECIDE, N);
17 VecSetFromOptions(y);
18 VecDuplicate(y, &yexact);
19
20 TSCreate(PETSC_COMM_WORLD, &ts);
21 TSSetProblemType(ts, TS_NONLINEAR);
22 TSSetRHSFunction(ts, RHS, FormRHS);
23 TSType(ts, TSRK);
24
25 TSSetTime(ts, t0); // Set initial time
26 TSSetMaxTime(ts, tf);
27 TSSetTimeStep(ts, dt);
28 TSSetExactFinalTime(ts, TS_EXACTFINALTIME_MATCHSTEP);
29 TSSetFromOptions(ts);
30
31
32 VecDestroy(&y), VecDestroy(&yexact), TSDestroy(&ts);
33
34
35 return PetscFinalize();
36 }
```

Handwritten notes on the right side of the screenshot:

$$\frac{4}{6} y_{n-1} - \frac{1}{3} y_{n-2} + \frac{2}{3} h \vec{f}(t_n, y_n)$$
$$\vec{f}(\vec{y}, t)$$
$$= \vec{A} \vec{y} + \vec{f}(t)$$
$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \vec{f} = \begin{pmatrix} 0 \\ t \end{pmatrix}$$
$$= y_1 + 0$$
$$= -y_0 + t$$

$$\begin{aligned} y_0 &= t - \sin t \\ y_1 &= 1 - \cos t \end{aligned}$$

So, we have initialized PETSc, now let us focus on creating the vectors. So, we will do VecCreate, so this will be PETSc com world and let us pass the address of the y array this has to be small y.

So, essentially we are creating the vector y and using various processes that we may or may not use, then we will do go through the usual grind. So, it is going to be VecSetSizes then it is going to have y, PETSC_DECIDE and finally, you will say you will you will pass the address of y. So, with this we have done oh sorry in set sizes we have to tell the number of elements.

So, it will be simply capital N that is 2 number of elements. I mean I could have hardly coded to do that s 2, but still we like to maintain the modularity, so nothing to worry about. Finally, we write VecSetFromOptions(y). So, that when you pass command line arguments in in running the C program you may pass some additional flags to specify either the solver or the Jacobean or the coloring of the Jacobean, so many things that is why this line is quite important to accept command and arguments.

Finally we have VecDuplicate(y, &yexact) alright. So, we are sort of creating this this variable and making it duplicate I am calling it yexact. So finally, we will construct whatever is going to be there in yexact and put it in yexact we going to do the time stepping and save it in y and finally we are going to take the error between y and yexact and find out what how the function progresses.

So, far we have created the 2 vectors now we are going to create the time stepping object. So, TSCreate PETSC COM WORLD and we must pass the address of the time stepping object. And this is the same as dmda or ksp or anything ok, usually all the objects in PETSc have their similar calling sequence.

Once we have created we must set from options. So, TSSet from options and we will pass the time stepping object and we are going to actually TSSet options will come much later. So, in this sequence we must first also specify what is the problem kind? So, we will do TSSetProblemType(ts, TS_NONLINEAR).

So, this is just to make use of any flex I mean flexibility that we may need later on to make the program non-linear, rather than having simply a linear program. But the linear program will much will run much faster than this, but anyway we do not mind it so much. After this we must do TSSetRHSFunction and this is the RHSFunction in the time stepping this entire thing which we call as $g, \bar{g}(\bar{y}, t)$.

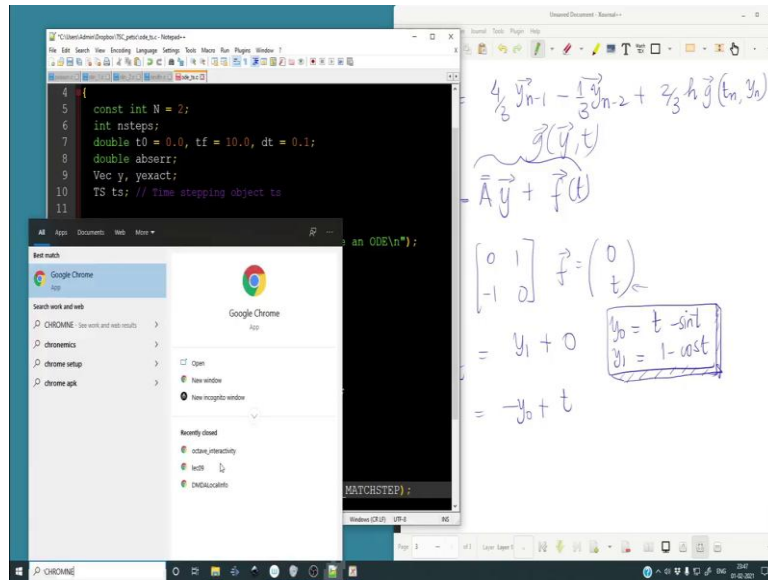
So, that is the entire function. So, in this case we will write down a user defined function which will be used to sort of make the function call back. So, we have ts, NULL, FormRHS, NULL. So, with the help of this you are telling the time stepping object that whatever you are going to do the right hand side callback has to be through the function which is called as form RHS. It can be any name you want in this particular case it is form RHS.

Then we are going to set the test ts type. So, ts type and the default will be the Runge Kutta, Runge Kutta integrator, so ts, TSRK ok. So, TSRK is the Runge Kutta time stepper. So, once these 4 lines are done you have created the time stepping object in which the problem type is considered to be non-linear despite being linear in this case, after that you are going to declare the callback to the function when it tries to evaluate the right hand side.

Now, we are going to set the time. So, TSSetTime(ts, t0). So, this is to set the initial time, then what do we have TSSet problem we have already set the problem type. So, set the initial time after that we are going to set the max time. So, TSSetMaxTime(ts, tf), then we are going to define the dt that we want to output the data. So, TSSetTimeStep(ts, dt).

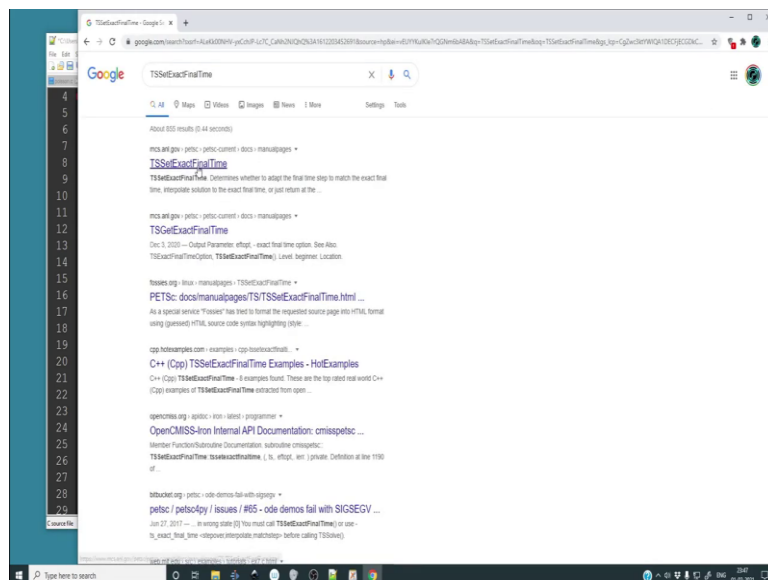
And finally we want to match the final time step. So, we must do an additional function call and it is called as `TSSetExactFinalTime(ts, TS_EXACTFINALTIME_MATCHSTEP)` no not extra exact final time match step ok.

(Refer Slide Time: 27:00)

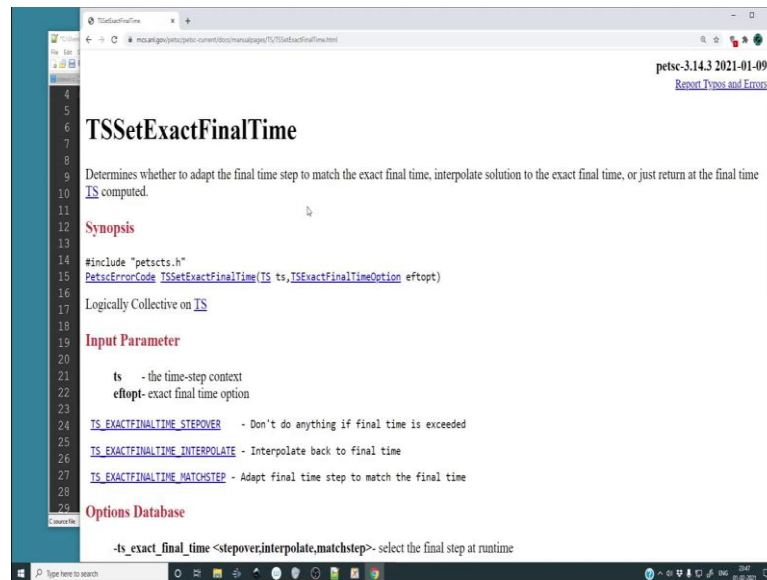


So, let me quickly open the function reference so `TSSetExactFinalTime`.

(Refer Slide Time: 27:09)



(Refer Slide Time: 27:14)



```
4  
5  
6 TSSetExactFinalTime  
7  
8  
9 Determines whether to adapt the final time step to match the exact final time, interpolate solution to the exact final time, or just return at the final time  
10 TS computed.  
11  
12 Synopsis  
13  
14 #include "petscts.h"  
15 PetscErrorCode TSSetExactFinalTime(TS ts, TSExactFinalTimeOption eftopt)  
16  
17 Logically Collective on TS  
18  
19 Input Parameter  
20  
21 ts - the time-step context  
22 eftopt- exact final time option  
23  
24 TS\_EXACTFINALTIME\_STEPOVER - Don't do anything if final time is exceeded  
25  
26 TS\_EXACTFINALTIME\_INTERPOLATE - Interpolate back to final time  
27  
28 TS\_EXACTFINALTIME\_MATCHSTEP - Adapt final time step to match the final time  
29  
30 Options Database  
31  
-ts_exact_final_time <steptover,interpolate,matchstep>- select the final step at runtime
```

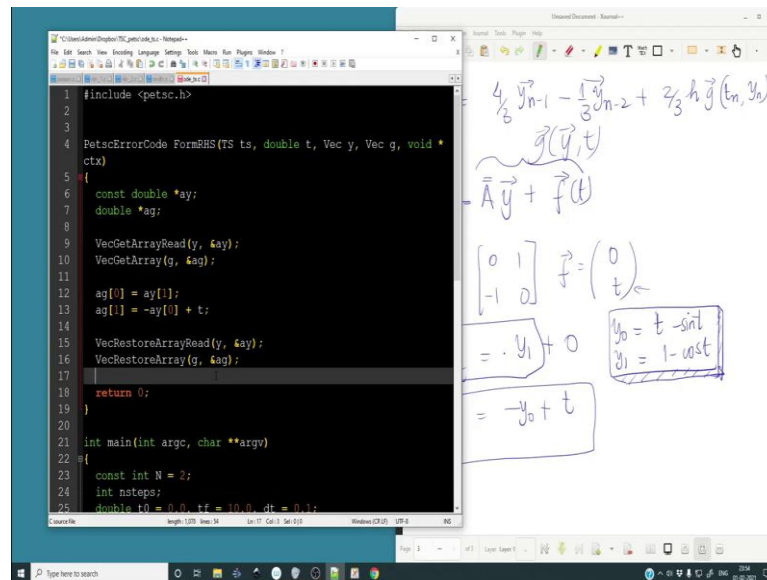
So, it determines whether to adapt the final time step to match the final step or just to return whatever final time you have, but it is good to have exact final time match step ok. So, using this it will sort of balance the final step alright. So, once this is done we have to do a TSSet from options and the reason why we do this is to allow for user to pass modifiers or arguments to the time stepping object through the command line ok. If you do not do this it will not take form ok.

So, that takes care of this so now we are yet to define what the form RHS function is going to be. But even before that we know that because we have declared so many variables we also have to remove a bunch of variables or destroy a bunch of variables. So, we will say VecDestroy(&y) that is the address of y, VecDestroy(&yexact) then you have TSDestroy much in line to the right destroy my destroy and all that thing.

So, here we have to pass the address of the time stepper ok, finally yeah that is that is pretty much it. So, so far what we have done is created the vector created the exact vector created the problem and the time stepper kind is in this case the Runge Kutta, then we have defined the times of which the integration has to occur ok. So, it is going from t_0 to t_f in steps of dt that is what is going on.

So, with this in mind what how can we proceed. So, we must define the callbacks and yeah. So, let us see how we can proceed from there. So, first of all let us definitely define what the callbacks will be, otherwise without that we will have no function evaluations.

(Refer Slide Time: 29:55)



So, we will write a functions. So, PetscErrorCode.

So, this is the return type so PetscErrorCode FormRHS(TS ts, double t), so there is the time the y vector and the g vector will be needed to populate it later on and any user defined context, so y *c t x. So, that is some user defined user con.

So, application user context so depending on what you are defining the structure as you are going to pass this, later on you can cast it and we have seen this in one of the previous lectures as well. So, in order to form the RHS we are passing it the y vector the time and the g vector so we must first convert this PETSc vec object into a c type array object.

So, the way to do it is to declare arrays. So, because the vector y is not going to change. So, const double *ay we will need a double *ag. So, these are the auxiliary arrays after this we have to first set the array from the vector, then afterwards you have to answer it. So, we have done this plenty of times before. So, we get array read for the case of y it is going to be an error read.

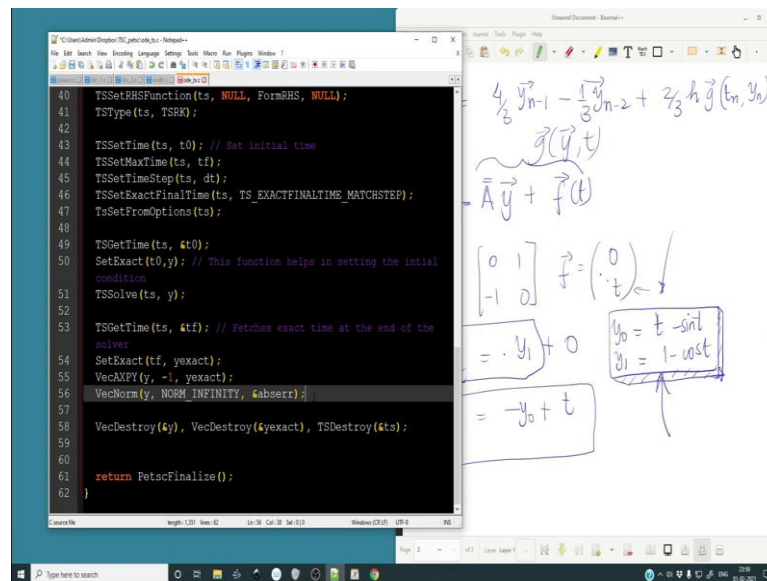
So, it is going to be y, the address of auxiliary y. Similarly we can do oops VecGetArray and we can pass g and the auxiliary function to g, then we can simply say ag[0] is something ag[1] is something and then we can restore. So, VecRestoreArrayRead we are going to do y and &ay and similarly we are going to do a VecRestoreArray(g, &ag).

So, $ag[0] = ay[1]$ and this is going to be $-y[0] + t$. So, let us proceed to write that down nothing wrong. So, $ay[1]$ oops and this will be $-ay[0] + t$. So, once this is successful we can simply return 0 to say everything is running fine alright.

So, now we have defined the callback function which the time stepper will evaluate when doing the time integration and more importantly it follows the similar way of declaring things as we have done in the previous lectures. Pertaining to anything which makes use of such kinds of PETSc vectors you have to first convert it into C vector, then convert it back into a PETSc vector ok .

So, let us continue not over here this is the function we have to go back to main. So, what we have done is we have set from options and now we must actually solve the actual thing.

(Refer Slide Time: 34:24)



So, let us do `TSSetTime` the time stepping object and the initial time, `SetExact t0 y0` well because there is no y_0 this simply going to be y because we have not yet done any time integration.

So, it y is so we have actually not defined what the initial condition is as well ok. But we are asking ok, so we are asking the function set exact to take t_0 and define what y will be. So, this function helps in setting the initial condition alright. So, set exact then

once we have the initial guess right not the initial guess, but the initial condition then we can simply do `TSSolve(ts, y)`.

So, now the time stepper is going to update each value of `y` until it reaches the final step alright and the final steps are sort of fixed using this set of code not the final step. But all the time stepping `dts` and final time set using that. So, at the end of this code we should have the numerical solution in the variable `y` alright.

So, now what we can do is we can proceed and simply find out the error ok. So, let us find out what the exact final time is in case it does not match the final step rather match the final time we can actually fetch what the final step is going to be. So, the function is `TSGetTime TS` and you have to return the so you have to pass the final time step variable. So now, this will be the exact solution not the exact solution with the exact time at the end of the solver.

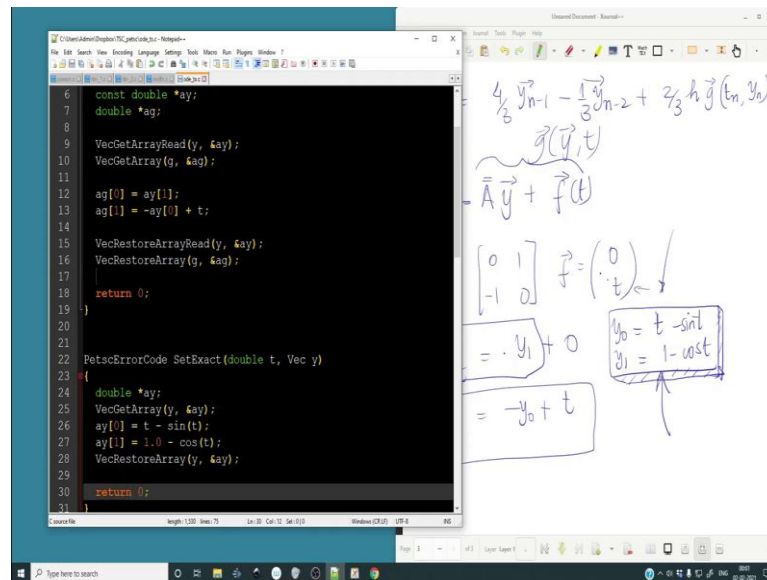
So, this fetches exact time at the end of the solver alright. So, once we have the time we can actually find out the exact solution at the final time ok. So, we can set the exact solution at the final time. So what is the exact solution? So, set from not set from so `SetExact(tf, yexact)` we do not want to overwrite `y` ok.

So, now that we have `yexact` as the solution exact solution which is this which is this, we can compare it with the approximate solution using the time stepping. Actually over here when we are setting the initial condition this is where we set the initial condition, the reason why this works is we know the analytical solution and. So, we can set the initial condition simply by setting `t0` into this expression that is why this works.

Otherwise in general you would have to specify clearly what the initial condition is supposed to be ok. But in this case because it is simply this analytical solution with `t` replaced by `0`. So now, we have the exact solution, so now as usual we will find out the error using `VecAXPY`. So, let us do that. So, `VecAXPY(y, -1, yexact)` alright.

So, then we simply do a norm. So, `VecNorm(y, NORM_INFINITY, &abserr)`; so we have abs error alright. So, with the help of this we have the error we will then destroy the variables which are not in use and finalize the program. So, now all we need to do is make a function definition for set exact ok.

(Refer Slide Time: 39:25)

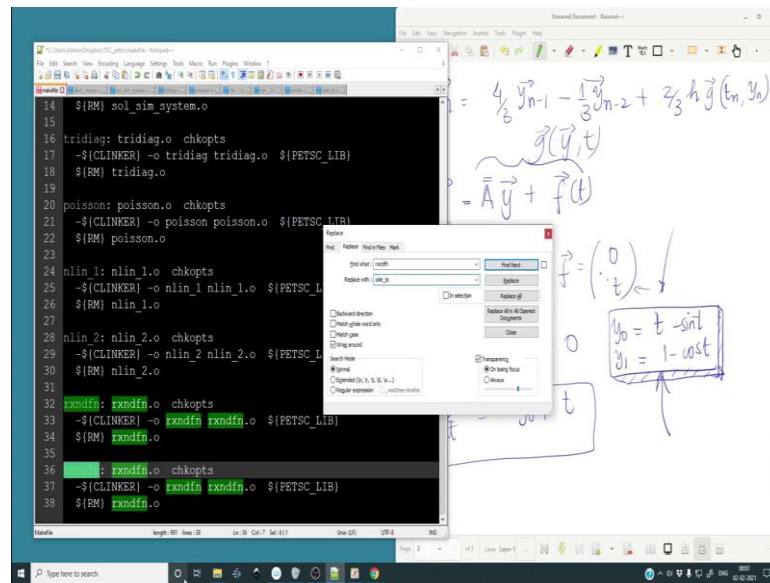


So let us go ahead and do that, so the return value will obviously be equal to PetscErrorCode SetExact(double t, Vec y) then we have we have to create the auxiliary array double *ay and VecGetArray(y, &ay). So, we are passing the address of a y you have to eventually do VecRestoreArray(y, &ay).

So, this is these are the things you need to do and once you have extracted the PETS vector y into the c array ay we can find out the exact solution estimate not the estimate. But the exact solution we can replace it by the exact solution expression. So, in that case $ay[0] = t - \sin(t)$ and $ay[1] = 1.0 - \cos(t)$.

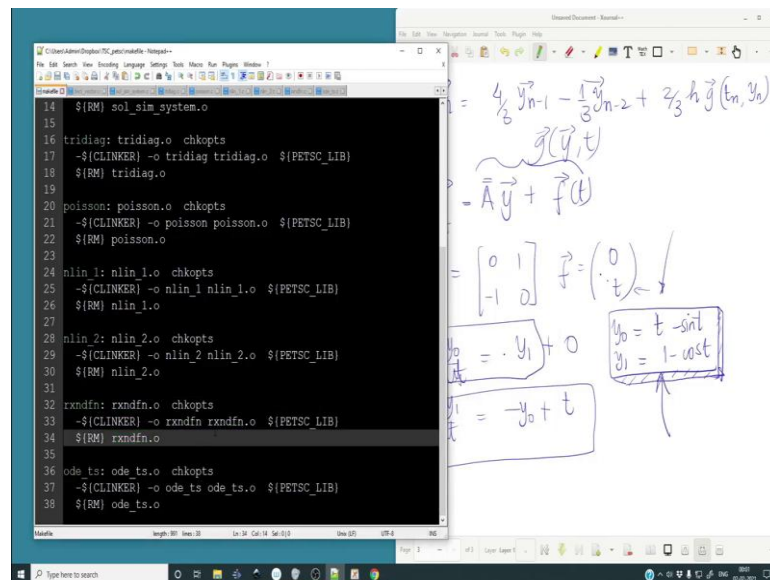
And if the program is successful you return what do you return a big 0 it is quite contradictory if everything runs successfully ok. So, that is the entire code let me run through it once again. In fact, let me compile it first, so let me modify the make file to create a new target ok.

(Refer Slide Time: 41:09)

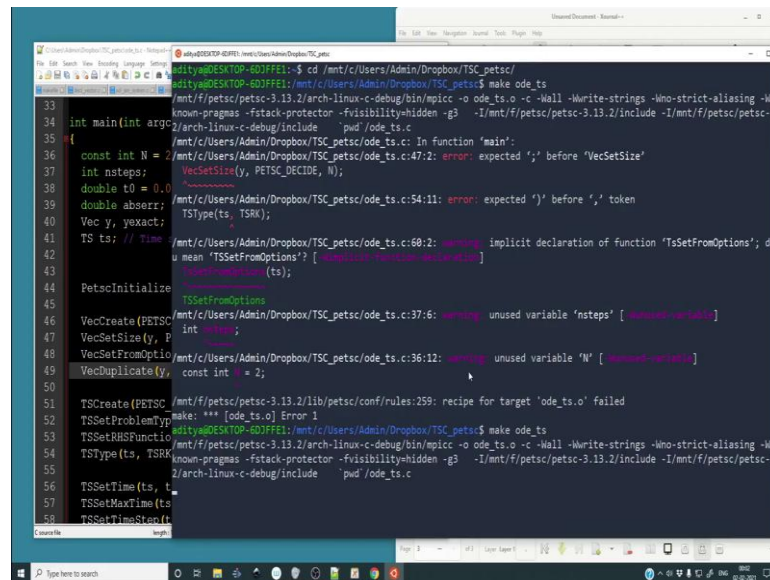


So, let me copy this. So, instead of the reaction diffusion control h you replace it by ode_ts.

(Refer Slide Time: 41:25)



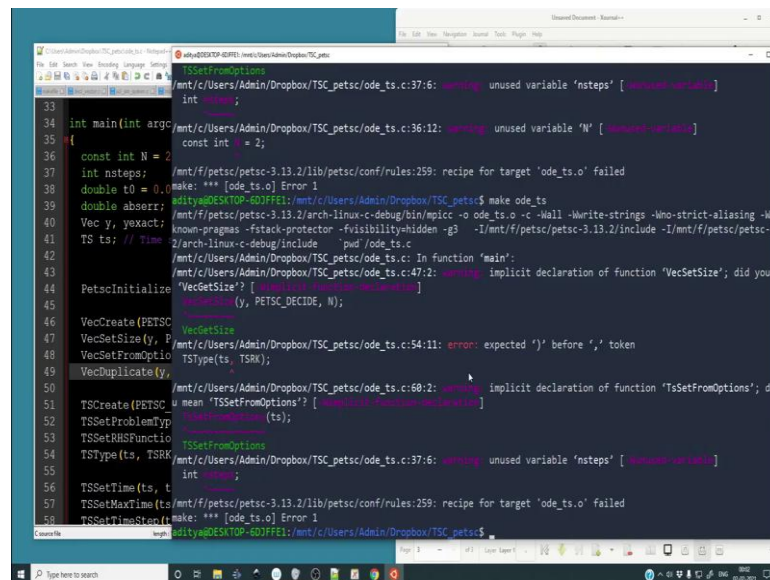
(Refer Slide Time: 41:31)



```
33
34 int main(int argc
35 {
36     const int N = 2
37     int nsteps;
38     double t0 = 0.0
39     double abserr;
40     Vec y, yexact;
41     TS ts; // Time
42     PetscInitialize
43     TSetFromOptions
44     VecCreate(PETSC
45     VecSetSize(y, P
46     VecSetFromOptio
47     VecDuplicate(y,
48     const int = 2;
49
50     TSCreate(PETSC
51     TSetProblemType
52     TSetRHSFunction
53     TSType(ts, TSRK
54     TSetTime(ts, t
55     TSetMaxTime(ts
56     TSetTimeStep(t
57
58
```

So, then you do replace alright. So, let us save it let me launch the ubuntu app and after that I have to navigate all the way to. So, cd mnt c Users Admin Dropbox TSC. So, then make ode_ts and let us see whether we have some errors or not ok. So, I think we have missed a semicolon somewhere I do not do not have to worry too much about this ok.

(Refer Slide Time: 42:23)



```
33
34 int main(int argc
35 {
36     const int N = 2
37     int nsteps;
38     double t0 = 0.0
39     double abserr;
40     Vec y, yexact;
41     TS ts; // Time
42     PetscInitialize
43     TSetFromOptions
44     VecCreate(PETSC
45     VecSetSize(y, P
46     VecSetFromOptio
47     VecDuplicate(y,
48     const int = 2;
49
50     TSCreate(PETSC
51     TSetProblemType
52     TSetRHSFunction
53     TSType(ts, TSRK
54     TSetTime(ts, t
55     TSetMaxTime(ts
56     TSetTimeStep(t
57
58
```

(Refer Slide Time: 42:38)

```
39 double abserr;
40 Vec y, yexact;
41 TS ts; // Time stepping object ts
42
43
44 PetscInitialize(&argc, &argv, NULL, "Solve an ODE\n");
45
46 VecCreate(PETSC_COMM_WORLD, &y);
47 VecSetSizes(y, PETSC_DECIDE, N);
48 VecSetFromOptions(y);
49 VecDuplicate(y, &yexact);
50
51 TSCreate(PETSC_COMM_WORLD, &ts);
52 TSSetProblemType(ts, TS_NONLINEAR);
53 TSSetRHSFunction(ts, NULL, FormRHS, NULL);
54 TSSetType(ts, TSEK);
55
56 TSSetTime(ts, t0); // Set initial time
57 TSSetMaxTime(ts, tf);
58 TSSetTimeStep(ts, dt);
59 TSSetExactFinalTime(ts, TS_EXACTFINALTIME_MATCHSTEP);
60 TSSetFromOptions(ts);
61
62 TSSetTime(ts, &t0);
63 SetExact(t0, y); // This function helps in setting the initial condition
```

Warnings:

- implicit declaration of function 'VecSetSize'; did you mean 'VecGetSize'?
- implicit declaration of function 'TSSetFromOptions'; did you mean 'TSSetFromOptions'?
- unused variable 'nsteps' [unused-variable]

(Refer Slide Time: 42:44)

```
47 VecSetSize(y, PETSC_DECIDE, N);
```

Warnings:

- Warning: chokpt target is deprecated and can be removed from user makefiles
- Warning: implicit declaration of function 'VecSetSize'; did you mean 'VecGetSize'?
- Warning: implicit declaration of function 'TSSetFromOptions'; did you mean 'TSSetFromOptions'?
- Warning: unused variable 'nsteps' [unused-variable]

Errors:

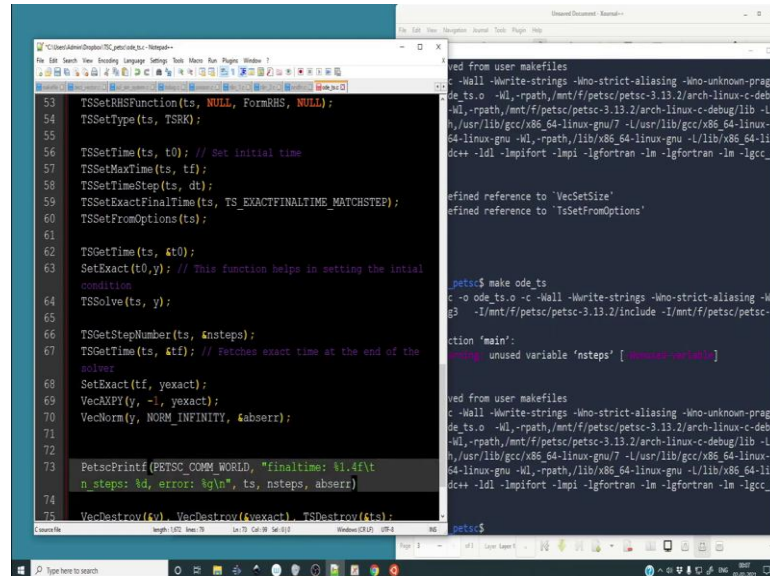
- undefined reference to 'VecSetSize'
- undefined reference to 'TSSetFromOptions'
- collect2: error: ld returned 1 exit status
- makefile:37: recipe for target 'ode_ts' failed
- make: [ode_ts] Error 1 ignored

So, there is a semicolon missing over here, what else hm? In fact, let me redo this there appears to be an error pertaining to semicolon once again ah. So, the function name I have put incorrectly.

So, it is TSSetType ok, so I have saved it let me make it ok. So, we all we have are a bunch of warnings ah, but there are some errors. So, vec set size and it is incorrect because the function is VecSetSizes and TSSetFromOptions is actually the S should be

capital. So, these are some silly mistakes that happen if you try to code too quickly, but anyway it just goes to show that you have to be super careful ok.

(Refer Slide Time: 43:30)



```
53 TSSetRHSFunction(ts, NULL, FormRHS, NULL);
54 TSSetType(ts, TSRK);
55
56 TSSetTime(ts, t0); // Set initial time
57 TSSetMaxTime(ts, tf);
58 TSSetTimeStep(ts, dt);
59 TSSetExactFinalTime(ts, TS_EXACTFINALTIME_MATCHSTEP);
60 TSSetFromOptions(ts);
61
62 TSSetTime(ts, t0);
63 SetExact(t0, y); // This function helps in setting the initial
    condition
64 TSSolve(ts, y);
65
66 TSSetStepNumber(ts, nsteps);
67 TSSetTime(ts, ttf); // Petscas exact time at the end of the
    solve
68 SetExact(tf, yexact);
69 VecXPY(y, -1, yexact);
70 VecNorm(y, NORM_INFINITY, &abserr);
71
72
73 PetscPrintf(PETSC_COMM_WORLD, "finaltime: %1.4f\t
    n_steps: %d, error: %g\n", ts, nsteps, abserr);
74
75 VecDestroy(v); VecDestroy(vexact); TSSetDestroy(ts);
```

```
ved from user makefiles
-c -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas
de ts.o -Wl,-rpath,/mnt/f/petasc/petasc-3.13.2/arch-linux-c-debug
-Wl,-rpath,/mnt/f/petasc/petasc-3.13.2/arch-linux-c-debug/lib-L/n
h,/usr/lib/gcc/x86_64-linux-gnu/7 -L/usr/lib/gcc/x86_64-linux-gr
64-linux-gnu -Wl,-rpath,/lib/x86_64-linux-gnu -L/lib/x86_64-lin
dccc++ -ldl -lmpifort -lmpi -lgfortran -lm -lgfortran -lm -lgcc_s
efined reference to 'VecSetSize'
efined reference to 'TSSetFromOptions'
petsc$ make ode ts
c -o ode_ts.o -c -Wall -Wwrite-strings -Wno-strict-aliasing -Wno
g3 -I/mnt/f/petasc/petasc-3.13.2/include -I/mnt/f/petasc/petasc-3.
ction 'main':
warning: unused variable 'nsteps' [ -Werror,-Wunused]
ved from user makefiles
-c -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas
de ts.o -Wl,-rpath,/mnt/f/petasc/petasc-3.13.2/arch-linux-c-debug
-Wl,-rpath,/mnt/f/petasc/petasc-3.13.2/arch-linux-c-debug/lib-L/n
h,/usr/lib/gcc/x86_64-linux-gnu/7 -L/usr/lib/gcc/x86_64-linux-gr
64-linux-gnu -Wl,-rpath,/lib/x86_64-linux-gnu -L/lib/x86_64-lin
dccc++ -ldl -lmpifort -lmpi -lgfortran -lm -lgfortran -lm -lgcc_s
petsc$
```

So, the only warning is then steps is unused. In fact, why do not we make use of n steps. What we can do is once the solver is done we can say TSSetStepNumber from the time stepping object and put it into steps. So, at the end of it all what we can do is before destroying everything we can print. So, we can do a PetscPrintf(PETSC_COMM_WORLD, "finaltime: %1.4f\t n_steps: %d, error: %g\n", ts, nsteps, abserr);

So, n_steps: %d because it is going to be an integer \n. So, we can simply go over here and set ts, nsteps and let us additionally write down error %g and this will be simply y not y it will be error yeah because not errors I mean it has to be abs error.

So, what we are doing is finding out the AX + y that is y - yexact over here and then finding the infinity norm over here and assigning it to abs error which we are printing over here let us not forget to put a semicolon over here alright.

(Refer Slide Time: 45:28)

```
make: [ode_ts] Error 1 (ignored)
/bin/rm -f ode_ts.o
sdity@DESKTOP-GD3JFPE1: /mnt/c/Users/Admin/Dropbox/TSC_petsc$ make ode_ts
sdity@DESKTOP-GD3JFPE1: /mnt/c/Users/Admin/Dropbox/TSC_petsc$ make ode_ts
53 TSSetRHSFunction(mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -o ode_ts.o -c -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-
54 TSSetType(ts, known-pragmas -fstack-protector -fvisibility=hidden -g3 -O -I/mnt/f/petsc/petsc-3.13.2/include -I/mnt/f/petsc/petsc-3.
55 /2/arch-linux-c-debug/include -pud /ode_ts.c
56 TSSetTime(ts, t/mnt/c/Users/Admin/Dropbox/TSC_petsc/ode_ts.c: In function 'main':
57 TSSetMaxTime(ts, t/mnt/c/Users/Admin/Dropbox/TSC_petsc/ode_ts.c:37:6: warning: unused variable 'nsteps' [ -Wunused-variable]
58 TSSetTimeStep(t, int nsteps;
59 TSSetExactFinal
60 TSSetFromOptionWarning: chkopts target is deprecated and can be removed from user makefiles
61 TSSetTime(ts, t/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragm
62 SetExact(t0, y), -L/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl, -rpath, /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/
63 f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl, -rpath, /usr/lib/gcc/x86_64-linux-gnu/7 -L/usr/lib/gcc/x86_64-linux-gr
64 condition -Wl, -rpath, /usr/lib/x86_64-linux-gnu -L/usr/lib/x86_64-linux-gnu -Wl, -rpath, /lib/x86_64-linux-gnu -L/lib/x86_64-lin
65 TSSolve(ts, y), -nu -lpetsc -lflapack -lflblas -lthreads -lX11 -lm -lstdc++ -ldl -lmpifort -lmpi -lgfortran -lm -lgcc_s
66 /bin/rm -f ode_ts.o
67 TSSetStepNumber(mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -o ode_ts.o -c -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-
68 TSSetTime(ts, known-pragmas -fstack-protector -fvisibility=hidden -g3 -O -I/mnt/f/petsc/petsc-3.13.2/include -I/mnt/f/petsc/petsc-3.
69 SetExact(tf, yelWarning: chkopts target is deprecated and can be removed from user makefiles
70 VecNorm(y, NORMstack-protector -fvisibility=hidden -g3 -O ode_ts ode_ts.o -Wl, -rpath, /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug
71 b -L/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl, -rpath, /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/
72 f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl, -rpath, /usr/lib/gcc/x86_64-linux-gnu/7 -L/usr/lib/gcc/x86_64-linux-gr
73 n_steps: 1d, @ru -nu -lpetsc -lflapack -lflblas -lthreads -lX11 -lm -lstdc++ -ldl -lmpifort -lmpi -lgfortran -lm -lgcc_s
74 uadmath -lstdc++ -ldl
75 VecDestroy(v), /bin/rm -f ode_ts.o
```

(Refer Slide Time: 45:36)

```
sdity@DESKTOP-GD3JFPE1: /mnt/c/Users/Admin/Dropbox/TSC_petsc$ make ode_ts
sdity@DESKTOP-GD3JFPE1: /mnt/c/Users/Admin/Dropbox/TSC_petsc$ make ode_ts
53 TSSetRHSFunction(mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -o ode_ts.o -c -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-
54 TSSetType(ts, known-pragmas -fstack-protector -fvisibility=hidden -g3 -O -I/mnt/f/petsc/petsc-3.13.2/include -I/mnt/f/petsc/petsc-3.
55 /2/arch-linux-c-debug/include -pud /ode_ts.c
56 TSSetTime(ts, t/mnt/c/Users/Admin/Dropbox/TSC_petsc/ode_ts.c: In function 'main':
57 TSSetMaxTime(ts, t/mnt/c/Users/Admin/Dropbox/TSC_petsc/ode_ts.c:37:6: warning: unused variable 'nsteps' [ -Wunused-variable]
58 TSSetTimeStep(t, int nsteps;
59 TSSetExactFinal
60 TSSetFromOptionWarning: chkopts target is deprecated and can be removed from user makefiles
61 TSSetTime(ts, t/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragm
62 SetExact(t0, y), -L/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl, -rpath, /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/
63 f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl, -rpath, /usr/lib/gcc/x86_64-linux-gnu/7 -L/usr/lib/gcc/x86_64-linux-gr
64 condition -Wl, -rpath, /usr/lib/x86_64-linux-gnu -L/usr/lib/x86_64-linux-gnu -Wl, -rpath, /lib/x86_64-linux-gnu -L/lib/x86_64-lin
65 TSSolve(ts, y), -nu -lpetsc -lflapack -lflblas -lthreads -lX11 -lm -lstdc++ -ldl -lmpifort -lmpi -lgfortran -lm -lgcc_s
66 /bin/rm -f ode_ts.o
67 TSSetStepNumber(mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -o ode_ts.o -c -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-
68 TSSetTime(ts, known-pragmas -fstack-
69 SetExact(tf, yelWarning: chkopts target is deprecated and can be removed from user makefiles
70 VecNorm(y, NORMstack-protector -fvisibility=hidden -g3 -O ode_ts ode_ts.o -Wl, -rpath, /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug
71 b -L/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl, -rpath, /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/
72 f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl, -rpath, /usr/lib/gcc/x86_64-linux-gnu/7 -L/usr/lib/gcc/x86_64-linux-gr
73 n_steps: 1d, @ru -nu -lpetsc -lflapack -lflblas -lthreads -lX11 -lm -lstdc++ -ldl -lmpifort -lmpi -lgfortran -lm -lgcc_s
74 uadmath -lstdc++ -ldl
75 VecDestroy(v), /bin/rm -f ode_ts.o
```

(Refer Slide Time: 45:49)

```
58 TSSolve(ts, y) 15 TS dt 0.259799 time 2.91839
59 TSSetExactFinal 16 TS dt 0.277075 time 3.45053
60 TSSetFromOptions 17 TS dt 0.267318 time 3.72761
61 18 TS dt 0.250457 time 3.99492
62 19 TS dt 0.234878 time 4.24538
63 20 TS dt 0.222686 time 4.48026
64 TSSolve(ts, y) 21 TS dt 0.213469 time 4.70286
65 SetExact(t0, y) 22 TS dt 0.207135 time 4.91633
66 condition 23 TS dt 0.203554 time 5.12347
67 TSSolve(ts, y) 24 TS dt 0.203123 time 5.32702
68 25 TS dt 0.206984 time 5.53015
69 26 TS dt 0.217945 time 5.93213
70 TSSetStepNumber 27 TS dt 0.243914 time 5.95507
71 TSSolve(ts, y) 28 TS dt 0.389594 time 6.19899
72 29 TS dt 0.279473 time 6.46629
73 SetExact(tf, y) 30 TS dt 0.232617 time 6.69195
74 VecXPY(y, -1) 31 TS dt 0.211995 time 6.92457
75 VecNorm(y, NORM_2) 32 TS dt 0.204735 time 7.13656
76 33 TS dt 0.203236 time 7.3413
77 34 TS dt 0.205312 time 7.54453
78 35 TS dt 0.21013 time 7.74985
79 PetscPrintf(PET2/arch-linux-c-debug/include 'pwd' /ode_ts.c
n_steps: 3d, error: -0.00499877
VecDestroy(&y) 36 TS dt 0.217492 time 7.95998
37 TS dt 0.227658 time 8.17747
38 TS dt 0.241511 time 8.40514
39 TS dt 0.260924 time 8.64665
40 TS dt 0.290128 time 8.90757
41 TS dt 0.336881 time 9.1977
42 TS dt 0.23275 time 9.5345
43 TS dt 0.287223 time 10.
Finaltime: 0.0059 n_steps: 852791312, error: -0.00499877
```

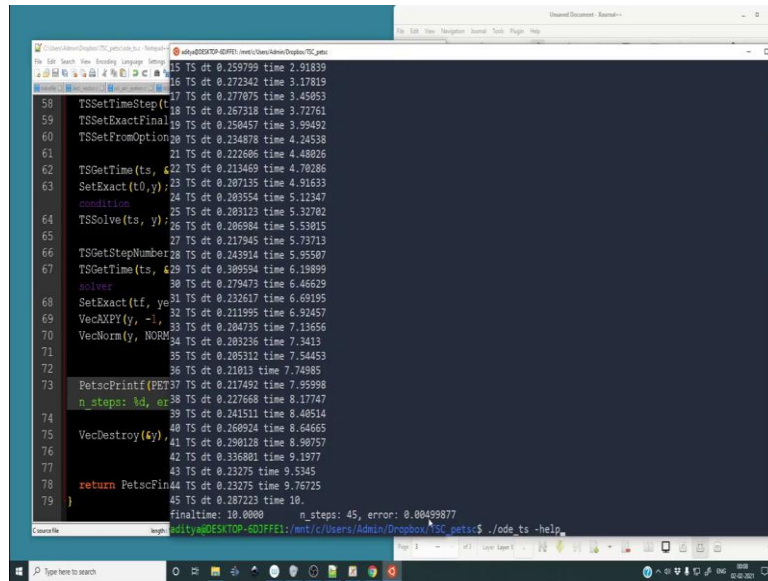
So, now we can compile this and great there is no error there is no warning everything is fine. So now, we can simply do `./ode_ts -ts_monitor`. So, it is a flag to show the monitor. So now, we have to accept this. So, what it looks like is ok this there appears to be an overflow for the number of steps.

So, is there some issue? So, it is int n steps I wonder why there is an error in the number of steps, this there appears to be some overflow happening over here ha. And ok so this has to be `df` because it is different term so let me recompile yeah.

(Refer Slide Time: 46:31)

```
58 TSSolve(ts, y) 15 TS dt 0.243914 time 5.95507
59 TSSetExactFinal 16 TS dt 0.389594 time 6.19899
60 TSSetFromOptions 17 TS dt 0.279473 time 6.46629
61 18 TS dt 0.232617 time 6.69195
62 TSSolve(ts, y) 19 TS dt 0.211995 time 6.92457
63 20 TS dt 0.203236 time 7.13656
64 21 TS dt 0.205312 time 7.3413
65 TSSolve(ts, y) 22 TS dt 0.205312 time 7.54453
66 SetExact(t0, y) 23 TS dt 0.21013 time 7.74985
67 condition 24 TS dt 0.217492 time 7.95998
68 TSSolve(ts, y) 25 TS dt 0.227658 time 8.17747
69 26 TS dt 0.241511 time 8.40514
70 27 TS dt 0.260924 time 8.64665
71 TSSetStepNumber 28 TS dt 0.290128 time 8.90757
72 TSSolve(ts, y) 29 TS dt 0.336881 time 9.1977
73 30 TS dt 0.23275 time 9.5345
74 SetExact(tf, y) 31 TS dt 0.287223 time 10.
75 VecXPY(y, -1) 32 TS dt 0.0059 n_steps: 852791312, error: -0.00499877
76 VecNorm(y, NORM_2) 33 TS dt 0.0059 n_steps: 852791312, error: -0.00499877
Warning: chkopts target is deprecated and can be removed from user makefiles
n_steps: 3d, error: -0.00499877
VecDestroy(&y) 34 TS dt 0.0059 n_steps: 852791312, error: -0.00499877
```

(Refer Slide Time: 46:33)

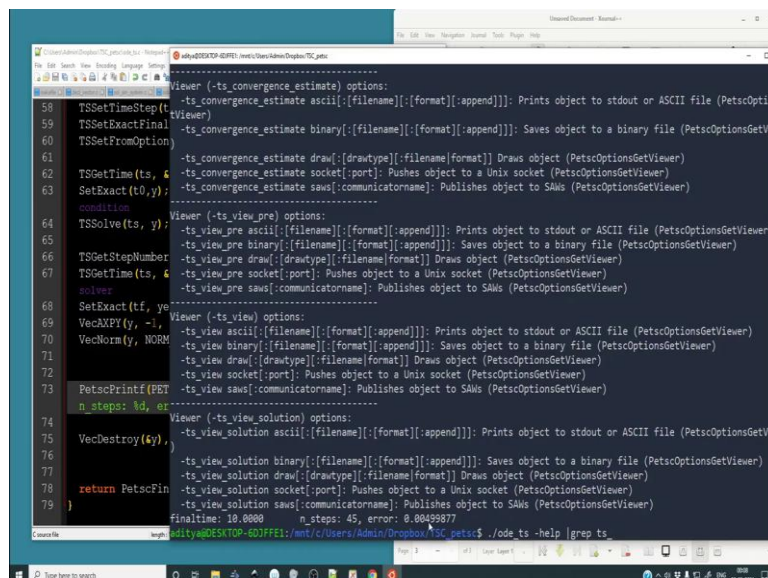


```
15 TS dt 0.259799 time 2.91839
16 TS dt 0.272342 time 3.17819
17 TS dt 0.277075 time 3.45053
58 TSSetTimeStep(t, 18 TS dt 0.267318 time 3.72761
59 TSSetExactFinal 19 TS dt 0.250457 time 3.99492
60 TSSetFromOption 20 TS dt 0.234878 time 4.24538
61 21 TS dt 0.222686 time 4.48026
62 TSGetTime(ts, 22 TS dt 0.213469 time 4.70286
63 SetExact(t0, y), 23 TS dt 0.207135 time 4.91633
64 condition, 24 TS dt 0.203554 time 5.12347
65 TSSolve(ts, y), 25 TS dt 0.203123 time 5.32702
66 TSGetStepNumber 26 TS dt 0.206984 time 5.53015
67 TSGetTime(ts, 27 TS dt 0.217945 time 5.73713
68 solve, 28 TS dt 0.243914 time 5.95587
69 SetExact(tf, ye 29 TS dt 0.389594 time 6.19899
70 VecXMPY(y, -1, 30 TS dt 0.279473 time 6.46529
71 VecNorm(y, NORM 31 TS dt 0.232617 time 6.69195
72 32 TS dt 0.211995 time 6.92457
73 PetscPrintf(PET 33 TS dt 0.204735 time 7.13656
74 n_steps: %d, er 34 TS dt 0.203236 time 7.3413
75 VecDestroy(&y), 35 TS dt 0.205312 time 7.54453
76 36 TS dt 0.21013 time 7.74906
77 37 TS dt 0.217492 time 7.95998
78 38 TS dt 0.227658 time 8.17747
79 39 TS dt 0.241511 time 8.40514
40 TS dt 0.260924 time 8.64665
41 TS dt 0.290128 time 8.90757
42 TS dt 0.336881 time 9.1977
43 TS dt 0.23275 time 9.5345
44 TS dt 0.23275 time 9.76725
Finaltime: 10.0000 n_steps: 45, error: 0.00499877
```

So, the final time is ten the number of steps is 45 and the error is this ok fine.

So, everything looks quite ok alright. So, so how do you actually go ahead and control this time stepping parameter. So, what we can do is we can do the grep business that we had done earlier as well. So, we can do `./ode_ts`.

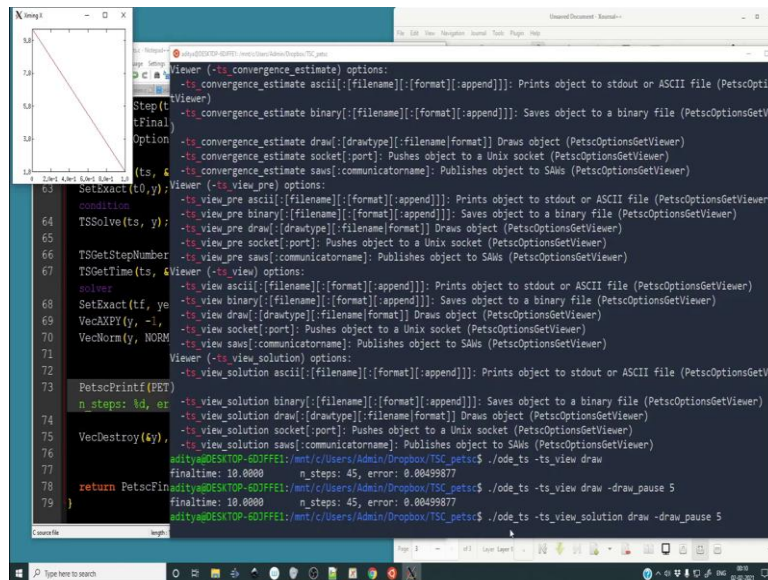
(Refer Slide Time: 47:04)



```
Viewer (-ts_convergence_estimate) options:
-ts_convergence_estimate ascii[:filename][:format][:append]]: Prints object to stdout or ASCII file (PetscOptionsGetViewer)
-ts_convergence_estimate binary[:filename][:format][:append]]: Saves object to a binary file (PetscOptionsGetViewer)
-ts_convergence_estimate draw[:drawtype][:filename[:format]] Draw object (PetscOptionsGetViewer)
-ts_convergence_estimate socket[:port]: Pushes object to a Unix socket (PetscOptionsGetViewer)
-ts_convergence_estimate saws[:communicatorname]: Publishes object to SAMs (PetscOptionsGetViewer)
Viewer (-ts_view_pre) options:
-ts_view_pre ascii[:filename][:format][:append]]: Prints object to stdout or ASCII file (PetscOptionsGetViewer)
-ts_view_pre binary[:filename][:format][:append]]: Saves object to a binary file (PetscOptionsGetViewer)
-ts_view_pre draw[:drawtype][:filename[:format]] Draw object (PetscOptionsGetViewer)
-ts_view_pre socket[:port]: Pushes object to a Unix socket (PetscOptionsGetViewer)
-ts_view_pre saws[:communicatorname]: Publishes object to SAMs (PetscOptionsGetViewer)
Viewer (-ts_view) options:
-ts_view ascii[:filename][:format][:append]]: Prints object to stdout or ASCII file (PetscOptionsGetViewer)
-ts_view binary[:filename][:format][:append]]: Saves object to a binary file (PetscOptionsGetViewer)
-ts_view draw[:drawtype][:filename[:format]] Draw object (PetscOptionsGetViewer)
-ts_view socket[:port]: Pushes object to a Unix socket (PetscOptionsGetViewer)
-ts_view saws[:communicatorname]: Publishes object to SAMs (PetscOptionsGetViewer)
Viewer (-ts_view_solution) options:
-ts_view_solution ascii[:filename][:format][:append]]: Prints object to stdout or ASCII file (PetscOptionsGetViewer)
-ts_view_solution binary[:filename][:format][:append]]: Saves object to a binary file (PetscOptionsGetViewer)
-ts_view_solution draw[:drawtype][:filename[:format]] Draw object (PetscOptionsGetViewer)
-ts_view_solution socket[:port]: Pushes object to a Unix socket (PetscOptionsGetViewer)
-ts_view_solution saws[:communicatorname]: Publishes object to SAMs (PetscOptionsGetViewer)
Finaltime: 10.0000 n_steps: 45, error: 0.00499877
```

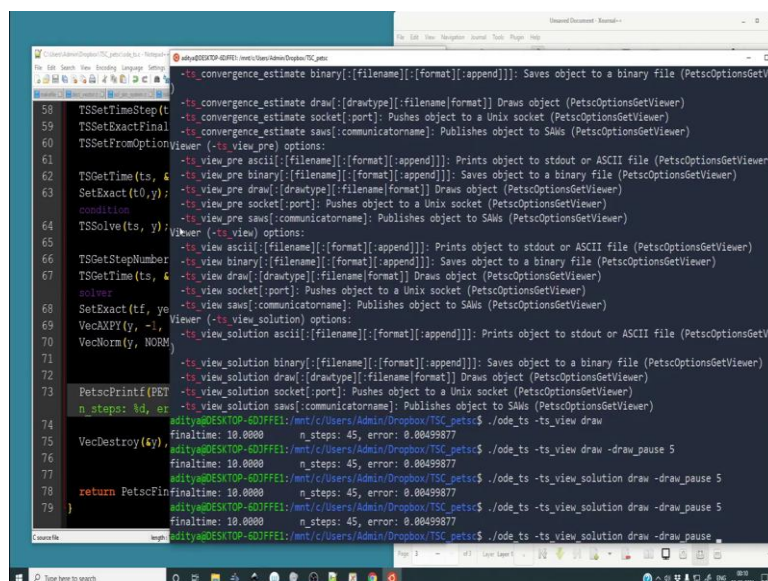
And we can say help so this help will return us a huge amount of information which we probably do not need. So, out of this we will grep. So, we will pipe this output to the grep utility of Linux and we will say show me options which have the prefix TS.

(Refer Slide Time: 47:23)

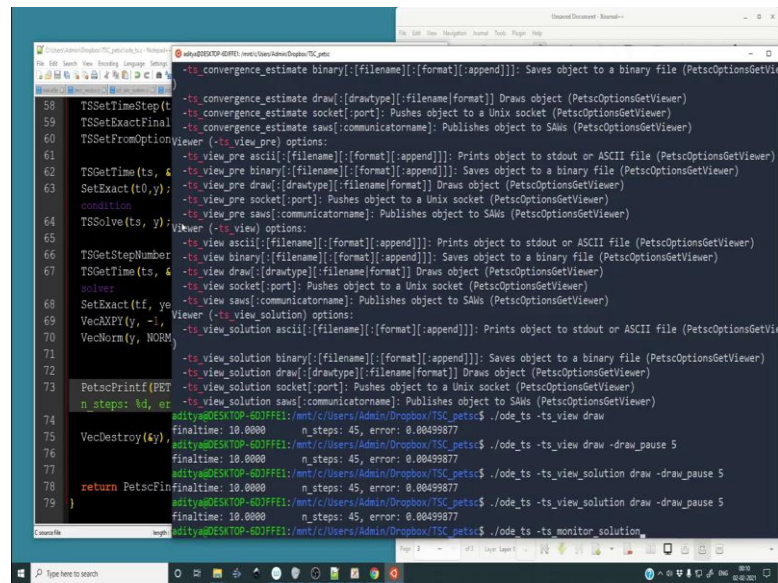


So, these are the options, so we can say TS view binary TS view draw. So, let me launch x min. So, we can see how it looks. So, let us do that let us see so minus ts view draw is there some draw delay let me execute it and say ok there was something minus draw pause underscore pause let us pause for 5 seconds oops. So, ts view solution let us yeah. So, that is how the ode goes (Refer Time: 48:16) until t equal to 10. So, let us have a look again ok.

(Refer Slide Time: 48:38)

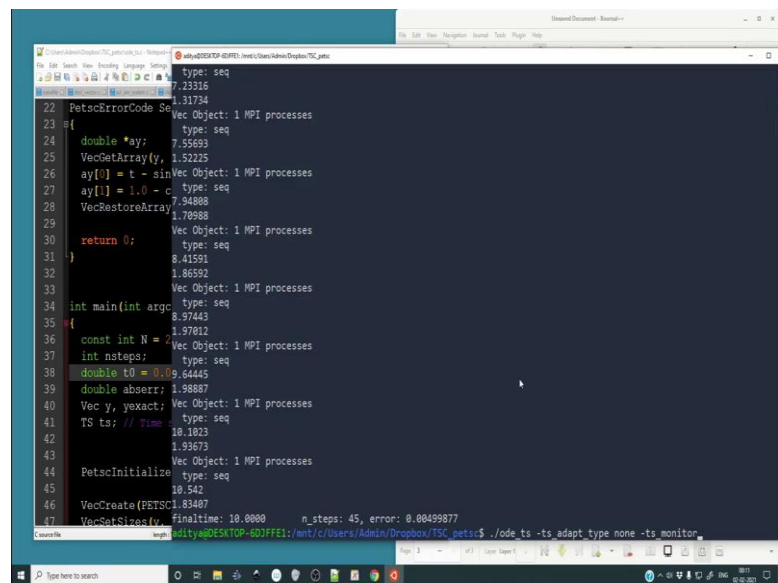


(Refer Slide Time: 48:39)



```
58 TSSetTimeStep(t, ...);
59 TSSetExactFinalTime(t, ...);
60 TSSetFromOptions(t, ...);
61 TSGetTime(ts, ...);
62 SetExact(t0, y0, ...);
63 TSolve(ts, y);
64 VecDestroy(y);
65 TSGetStepNumber(t, ...);
66 TSGetTime(ts, ...);
67 SetExact(tf, ye, ...);
68 VecXPY(y, -1, ...);
69 VecNorm(y, NORM_2, ...);
70 PetscPrintf(PETSC_COMM_WORLD, ...);
71 PetscMPISize(&nsteps);
72 PetscMPITypeSize(&type);
73 PetscMPITypeSize(&type);
74 PetscMPITypeSize(&type);
75 PetscMPITypeSize(&type);
76 PetscMPITypeSize(&type);
77 PetscMPITypeSize(&type);
78 PetscMPITypeSize(&type);
79 PetscMPITypeSize(&type);
```

(Refer Slide Time: 48:45)



```
22 PetscErrorCode Se...
23 {
24     double *ay;
25     VecGetArray(y, ...);
26     ay[0] = t - sin(t);
27     ay[1] = 1.0 - cos(t);
28     VecRestoreArray(y, ...);
29     return 0;
30 }
31
32 int main(int argc, char **argv)
33 {
34     const int N = 2;
35     int nsteps;
36     double t0 = 0.0;
37     double abserr;
38     Vec y, yexact;
39     TS ts;
40     PetscInitialize(&argc, &argv, ...);
41     VecCreate(PETSC_COMM_WORLD, ...);
42     VecSetSizes(y, ...);
```

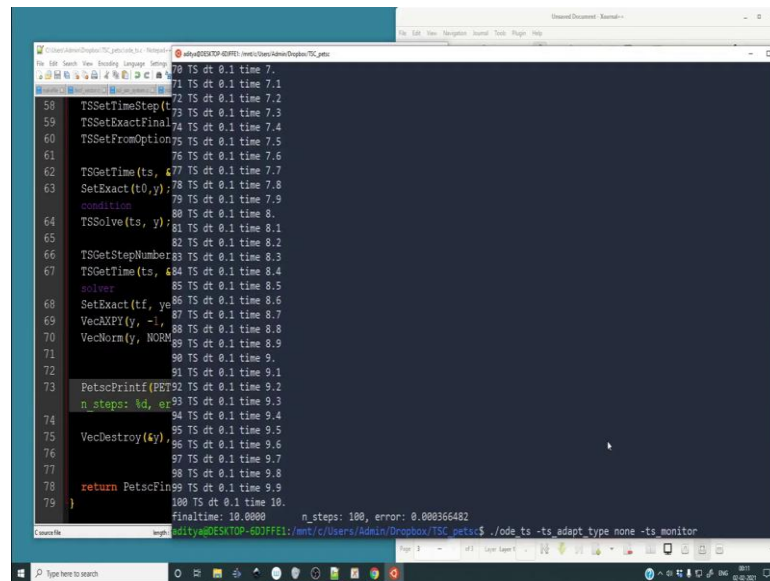
So, let us do TSC to monitor solution. So, it tells you what the value of the solution is at the final time ok or you could ideally do. So, we can turn off the adaptive time stepping. So, you can note over here that the time step is sort of dt, but the time step is kept adaptive and the reason why time step is kept adaptive, because the Runge Kutta really runs well with adaptive time stepping.

So, whenever it has a smooth function it can have large steps, whenever we have a fast varying function we can always choose small steps. So now, we can turn off time stepping

adaptive time stepping and ask it to do time steps which are fixed it have to be defined so yeah. So, if when time adaptive time stepping is not used it will fall back on this default time stepping of 0.1.

So, we can do `-ts_adapt-type none -ts_monitor`.

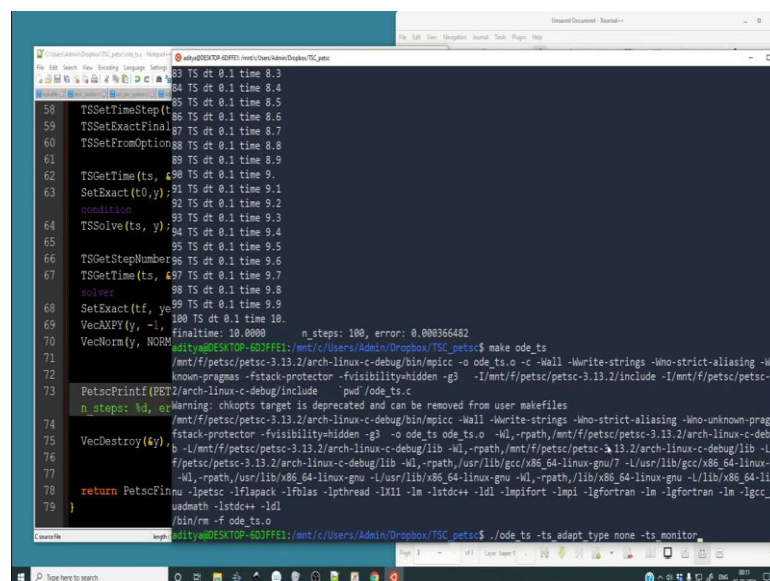
(Refer Slide Time: 50:02)



```
70 TS dt 0.1 time 7.0
71 TS dt 0.1 time 7.1
72 TS dt 0.1 time 7.2
58 TSSetTimeStep(t 73 TS dt 0.1 time 7.3
59 TSSetExactFinal 74 TS dt 0.1 time 7.4
60 TSSetFromOption 75 TS dt 0.1 time 7.5
61 76 TS dt 0.1 time 7.6
62 TSGetTime(ts, 77 TS dt 0.1 time 7.7
63 SetExact(t0,y) 78 TS dt 0.1 time 7.8
79 TS dt 0.1 time 7.9
condition 80 TS dt 0.1 time 8.0
64 TSSolve(ts, y) 81 TS dt 0.1 time 8.1
65 82 TS dt 0.1 time 8.2
66 TSGetStepNumber 83 TS dt 0.1 time 8.3
67 TSGetTime(ts, 84 TS dt 0.1 time 8.4
85 TS dt 0.1 time 8.5
68 SetExact(tf, ye 86 TS dt 0.1 time 8.6
69 VecAXPY(y, -1, 87 TS dt 0.1 time 8.7
70 VecNorm(y, NORM 88 TS dt 0.1 time 8.8
71 89 TS dt 0.1 time 8.9
72 90 TS dt 0.1 time 9.0
73 PetscPrintf(PET 91 TS dt 0.1 time 9.1
n_steps: %d, ex 92 TS dt 0.1 time 9.2
93 TS dt 0.1 time 9.3
74 94 TS dt 0.1 time 9.4
75 VecDestroy(&y) 95 TS dt 0.1 time 9.5
76 96 TS dt 0.1 time 9.6
77 97 TS dt 0.1 time 9.7
78 98 TS dt 0.1 time 9.8
79 99 TS dt 0.1 time 9.9
100 TS dt 0.1 time 10.0
Finaltime: 10.0000 n_steps: 100, error: 0.000366482
```

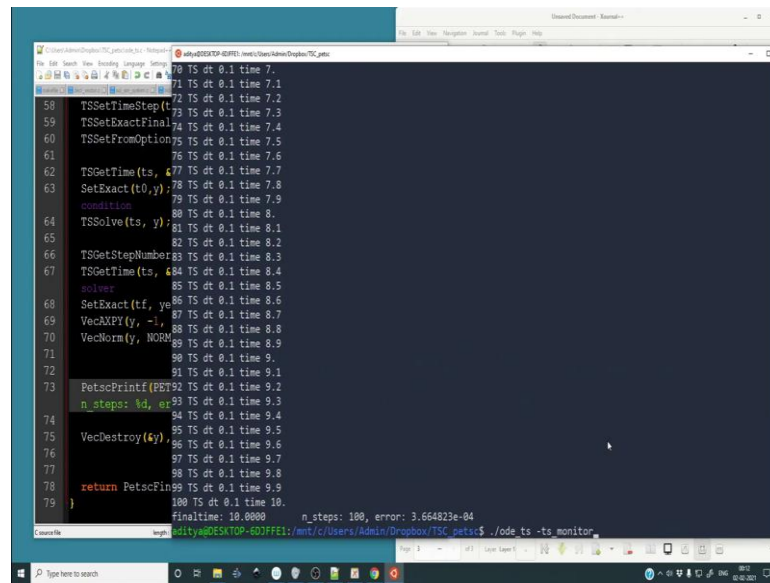
So, look it is its going to take all the time steps like this 3 10 to the sorry, why is it not giving me (Refer Time: 50:13) so on.

(Refer Slide Time: 50:19)



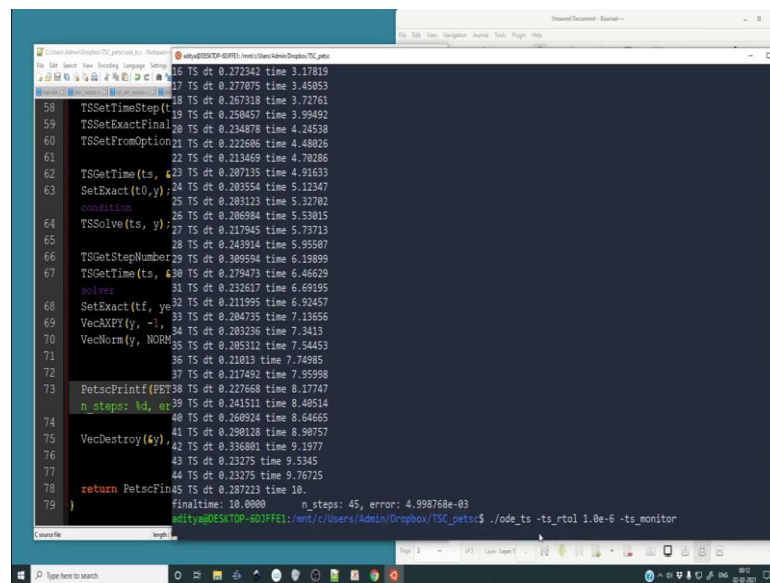
```
83 TS dt 0.1 time 8.3
84 TS dt 0.1 time 8.4
85 TS dt 0.1 time 8.5
58 TSSetTimeStep(t 86 TS dt 0.1 time 8.6
59 TSSetExactFinal 87 TS dt 0.1 time 8.7
60 TSSetFromOption 88 TS dt 0.1 time 8.8
61 89 TS dt 0.1 time 8.9
62 TSGetTime(ts, 90 TS dt 0.1 time 9.0
63 SetExact(t0,y) 91 TS dt 0.1 time 9.1
condition 92 TS dt 0.1 time 9.2
64 TSSolve(ts, y) 93 TS dt 0.1 time 9.3
65 94 TS dt 0.1 time 9.4
66 TSGetStepNumber 95 TS dt 0.1 time 9.5
67 TSGetTime(ts, 96 TS dt 0.1 time 9.6
97 TS dt 0.1 time 9.7
68 SetExact(tf, ye 98 TS dt 0.1 time 9.8
69 VecAXPY(y, -1, 99 TS dt 0.1 time 9.9
100 TS dt 0.1 time 10.0
Finaltime: 10.0000 n_steps: 100, error: 0.000366482
```

(Refer Slide Time: 50:22)



```
58 TSSetTimeStep(t, 70 TS dt 0.1 time 7.1
59 TSSetExactFinal 71 TS dt 0.1 time 7.2
60 TSSetFromOption 72 TS dt 0.1 time 7.3
61 73 TS dt 0.1 time 7.4
62 TSSolve(ts, y) 74 TS dt 0.1 time 7.5
63 SetExact(t0, y) 75 TS dt 0.1 time 7.6
64 condition 76 TS dt 0.1 time 7.7
65 TSSolve(ts, y) 77 TS dt 0.1 time 7.8
66 TSSetStepNumber 78 TS dt 0.1 time 7.9
67 TSSolve(ts, y) 79 TS dt 0.1 time 8.0
68 SetExact(tf, y) 80 TS dt 0.1 time 8.1
69 VecXPY(y, -1) 81 TS dt 0.1 time 8.2
70 VecNorm(y, NORM) 82 TS dt 0.1 time 8.3
71 83 TS dt 0.1 time 8.4
72 84 TS dt 0.1 time 8.5
73 PetscPrintf(PET 85 TS dt 0.1 time 8.6
74 n_steps: %d, er 86 TS dt 0.1 time 8.7
75 VecDestroy(&y) 87 TS dt 0.1 time 8.8
76 88 TS dt 0.1 time 8.9
77 89 TS dt 0.1 time 9.0
78 90 TS dt 0.1 time 9.1
79 91 TS dt 0.1 time 9.2
Finaltime: 10.0000 n_steps: 100 error: 3.664823e-04
```

(Refer Slide Time: 50:33)

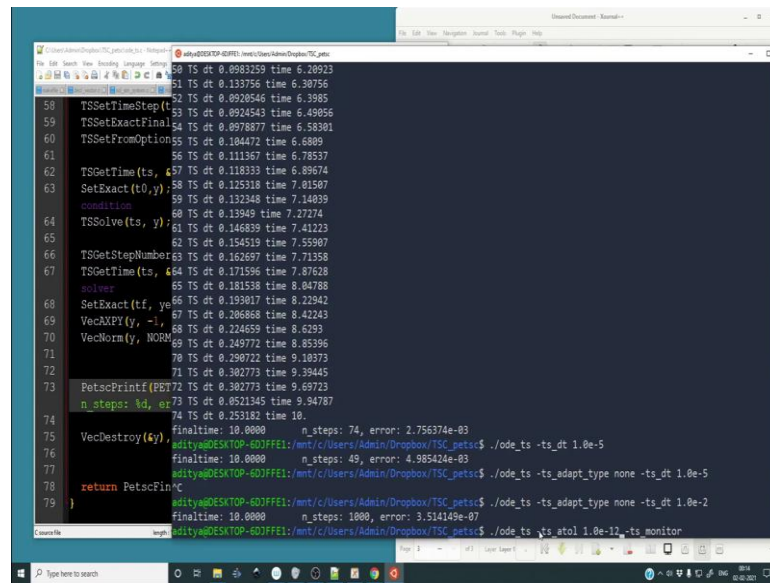


```
58 TSSetTimeStep(t, 16 TS dt 0.272342 time 3.17819
59 TSSetExactFinal 17 TS dt 0.277075 time 3.45053
60 TSSetFromOption 18 TS dt 0.267318 time 3.72761
61 TSSolve(ts, y) 19 TS dt 0.250457 time 3.99492
62 SetExact(t0, y) 20 TS dt 0.234878 time 4.24538
63 condition 21 TS dt 0.222886 time 4.48026
64 TSSolve(ts, y) 22 TS dt 0.213469 time 4.70286
65 TSSetStepNumber 23 TS dt 0.207135 time 4.91653
66 TSSolve(ts, y) 24 TS dt 0.203554 time 5.12347
67 SetExact(tf, y) 25 TS dt 0.203123 time 5.32702
68 VecXPY(y, -1) 26 TS dt 0.206984 time 5.53015
69 VecNorm(y, NORM) 27 TS dt 0.217945 time 5.73713
70 28 TS dt 0.243914 time 5.95507
71 TSSolve(ts, y) 29 TS dt 0.309594 time 6.18899
72 SetExact(t0, y) 30 TS dt 0.279473 time 6.46629
73 condition 31 TS dt 0.232617 time 6.69195
74 TSSolve(ts, y) 32 TS dt 0.211995 time 6.92457
75 SetExact(tf, y) 33 TS dt 0.208735 time 7.13656
76 VecXPY(y, -1) 34 TS dt 0.203236 time 7.3413
77 VecNorm(y, NORM) 35 TS dt 0.205312 time 7.54433
78 36 TS dt 0.21013 time 7.74985
79 37 TS dt 0.217492 time 7.95998
80 PetscPrintf(PET 38 TS dt 0.227668 time 8.17747
81 n_steps: %d, er 39 TS dt 0.241511 time 8.40514
82 VecDestroy(&y) 40 TS dt 0.260924 time 8.64665
83 41 TS dt 0.290128 time 8.90757
84 42 TS dt 0.336001 time 9.1977
85 43 TS dt 0.23275 time 9.5345
86 44 TS dt 0.23275 time 9.76725
87 45 TS dt 0.287223 time 10.0000
Finaltime: 10.0000 n_steps: 45 error: 4.998768e-03
```

So, let me recompile. So, the error is $3.664 \cdot 10^{-4}$ it took less steps it gave us a fine error ok. So, you can also change the tolerances that you choose in the error.

So, this particular error is based on a tolerance of 10^{-4} that is the default value, but we can change that particular tolerance as well. So, we can say minus ts_rtol 1.0e-6 – ts_monitor.

(Refer Slide Time: 51:12)



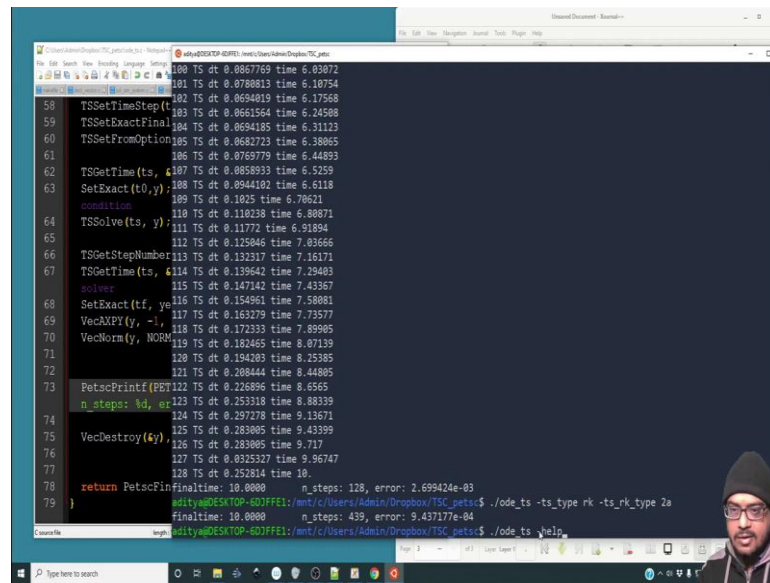
```
50 TS dt 0.0983259 time 6.20923
51 TS dt 0.133756 time 6.30756
52 TS dt 0.0920546 time 6.39385
58 TSSetTimeStep(t, 50) TS dt 0.0924543 time 6.49856
59 TSSetExactFinal 54 TS dt 0.0978877 time 6.58301
60 TSSetFromOptions 55 TS dt 0.104472 time 6.6889
61 56 TS dt 0.111367 time 6.78537
62 TSGetTime(ts, 57) TS dt 0.118333 time 6.89674
63 SetExact(t0, y) 58 TS dt 0.125318 time 7.01507
64 condition 59 TS dt 0.132348 time 7.14039
65 TSSolve(ts, y) 60 TS dt 0.13949 time 7.27274
66 61 TS dt 0.146839 time 7.41223
67 TSGetStepNumber 63 TS dt 0.154519 time 7.55907
68 TSGetTime(ts, 64) TS dt 0.162697 time 7.71358
69 65 TS dt 0.171596 time 7.87628
70 solvers 66 TS dt 0.181538 time 8.04788
71 SetExact(tf, y) 67 TS dt 0.193017 time 8.22942
72 VecXPY(y, -1) 68 TS dt 0.206868 time 8.42243
73 VecNorm(y, NORM) 69 TS dt 0.224659 time 8.6293
74 70 TS dt 0.249772 time 8.85396
75 71 TS dt 0.290722 time 9.10373
76 PetscPrintf(PET) 72 TS dt 0.302773 time 9.30445
77 n_steps: 3d, error 73 TS dt 0.302773 time 9.69723
78 74 TS dt 0.0521345 time 9.94787
79 75 TS dt 0.253182 time 10.
76 finaltime: 10.0000 n_steps: 74, error: 2.756374e-03
77 VecDestroy(&y) 76 finaltime: 10.0000 n_steps: 49, error: 4.985424e-03
78 77 finaltime: 10.0000 n_steps: 49, error: 4.985424e-03
79 return PetscFinC 78 finaltime: 10.0000 n_steps: 49, error: 4.985424e-03
80 79 finaltime: 10.0000 n_steps: 1800, error: 3.514149e-07
81 80 finaltime: 10.0000 n_steps: 1800, error: 3.514149e-07
```

You can also change this. So, the numerical error actually keeps on accumulating each step, but each step you are actually in accruing a very small error. I mean this is quite common for all time stepping methods.

But yeah I mean this kind of thing keeps on happening and one should not be too surprised. But the fact that you can modify the time step you can choose the time step that you want you can choose the dt to be 1.0×10^{-5} and you have to turn off the adaptive as well. So it will take more time, but it will give you a really accurate solution, but probably it is a bit too small.

So, let me cancel execution let me make it 10^{-2} . So, you get a very small error. So, it is all a matter of how small or how large you choose the errors to be. In fact, if you choose this to be -12 the error keeps on argument because the Δt is quite large.

(Refer Slide Time: 52:52)

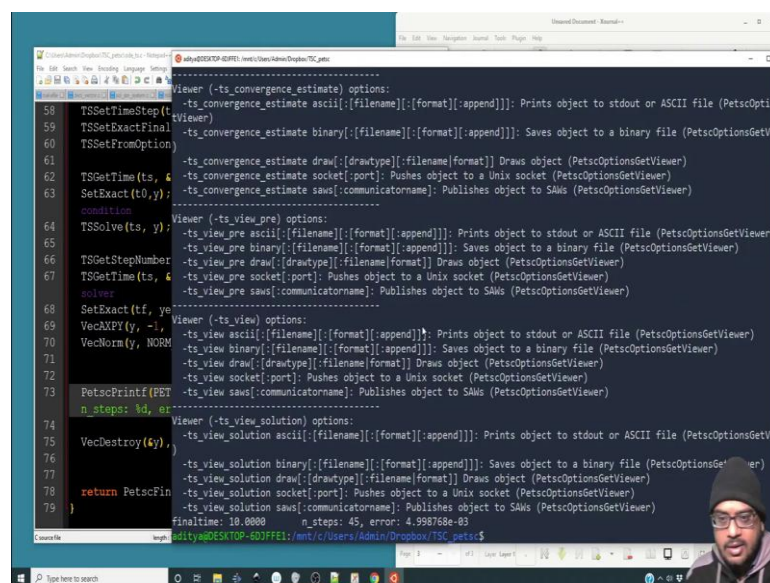


```
100 TS dt 0.0867769 time 6.03072
101 TS dt 0.0786813 time 6.10754
102 TS dt 0.06944919 time 6.17958
58 TSSetTimeStep(t, 103 TS dt 0.0661564 time 6.24598
59 TSSetExactFinal 104 TS dt 0.0694185 time 6.31123
60 TSSetFromOption 105 TS dt 0.0682723 time 6.38065
61 106 TS dt 0.0769779 time 6.44893
62 TSGetTime(ts, 107 TS dt 0.0858933 time 6.5259
63 SetExact(t0, y) 108 TS dt 0.0944102 time 6.6118
condition 109 TS dt 0.1025 time 6.70621
64 TSSolve(ts, y) 110 TS dt 0.110238 time 6.80671
65 111 TS dt 0.11772 time 6.91094
112 TS dt 0.125046 time 7.03666
66 TSGetStepNumber 113 TS dt 0.132317 time 7.16171
67 TSGetTime(ts, 114 TS dt 0.139542 time 7.29403
solvers 115 TS dt 0.147142 time 7.43367
68 SetExact(tf, ye 116 TS dt 0.154961 time 7.58081
69 VecXPY(y, -1, 117 TS dt 0.163279 time 7.73577
70 VecNorm(y, NORM 118 TS dt 0.172333 time 7.89905
119 TS dt 0.182465 time 8.07139
120 TS dt 0.194283 time 8.25285
121 TS dt 0.208444 time 8.44805
73 PetscPrintf(PET 122 TS dt 0.226896 time 8.6585
n_steps: 3d, er 123 TS dt 0.253318 time 8.88339
124 TS dt 0.297278 time 9.13671
74 VecDestroy(&y) 125 TS dt 0.283005 time 9.43399
126 TS dt 0.283005 time 9.717
76 127 TS dt 0.0325327 time 9.96747
77 128 TS dt 0.252814 time 10.
78 return PetscFin 10.0000 n_steps: 128, error: 2.699424e-03
79 } gdiya@DESKTOP-GD3JPFEL: /mnt/c/Users/Admin/Dropbox/TSC_petsc$ ./ode_ts -ts_type rk -ts_rk_type 2a
Finaltime: 10.0000 n_steps: 439, error: 9.437177e-04
gdiya@DESKTOP-GD3JPFEL: /mnt/c/Users/Admin/Dropbox/TSC_petsc$ ./ode_ts -help
```

So, you can actually set a maximum Δ maximum dt there is an option to change it you can find it easily in the function reference ok. So, this is how you can go about doing simple od integration with the help of petsc.

In fact, let me just show you one last option which will be quite useful. So, this will be TS type. So, TS type can be m crank Nicholson maybe no or Runge Kutta. So, TS type RK minus TS RK type 2 a. So, let us see so that is the second order.

(Refer Slide Time: 53:50)



```
Viewer (-ts_convergence_estimate) options:
-ts_convergence_estimate ascii[:filename][:format][:append]]: Prints object to stdout or ASCII file (PetscOptionsGetViewer)
-ts_convergence_estimate binary[:filename][:format][:append]]: Saves object to a binary file (PetscOptionsGetViewer)
-ts_convergence_estimate draw[:drawtype][:filename[:format]] Draw object (PetscOptionsGetViewer)
-ts_convergence_estimate socket[:port]: Pushes object to a Unix socket (PetscOptionsGetViewer)
-ts_convergence_estimate saws[:communicatorname]: Publishes object to SAs (PetscOptionsGetViewer)
-----
Viewer (-ts_view_pre) options:
-ts_view_pre ascii[:filename][:format][:append]]: Prints object to stdout or ASCII file (PetscOptionsGetViewer)
-ts_view_pre binary[:filename][:format][:append]]: Saves object to a binary file (PetscOptionsGetViewer)
-ts_view_pre draw[:drawtype][:filename[:format]] Draw object (PetscOptionsGetViewer)
-ts_view_pre socket[:port]: Pushes object to a Unix socket (PetscOptionsGetViewer)
-ts_view_pre saws[:communicatorname]: Publishes object to SAs (PetscOptionsGetViewer)
-----
Viewer (-ts_view) options:
-ts_view ascii[:filename][:format][:append]]: Prints object to stdout or ASCII file (PetscOptionsGetViewer)
-ts_view binary[:filename][:format][:append]]: Saves object to a binary file (PetscOptionsGetViewer)
-ts_view draw[:drawtype][:filename[:format]] Draw object (PetscOptionsGetViewer)
-ts_view socket[:port]: Pushes object to a Unix socket (PetscOptionsGetViewer)
-ts_view saws[:communicatorname]: Publishes object to SAs (PetscOptionsGetViewer)
-----
Viewer (-ts_view_solution) options:
-ts_view_solution ascii[:filename][:format][:append]]: Prints object to stdout or ASCII file (PetscOptionsGetViewer)
-ts_view_solution binary[:filename][:format][:append]]: Saves object to a binary file (PetscOptionsGetViewer)
-ts_view_solution draw[:drawtype][:filename[:format]] Draw object (PetscOptionsGetViewer)
-ts_view_solution socket[:port]: Pushes object to a Unix socket (PetscOptionsGetViewer)
-ts_view_solution saws[:communicatorname]: Publishes object to SAs (PetscOptionsGetViewer)
-----
Finaltime: 10.0000 n_steps: 45, error: 4.998768e-03
gdiya@DESKTOP-GD3JPFEL: /mnt/c/Users/Admin/Dropbox/TSC_petsc$
```

So, there is a whole host of errors which you can easily obtain using this command and unfortunately my scrolling does not seem to work, but you can scroll up and see all the different options that are available to you.

So, with this simple example I am going to end this particular lecture and you can take a look into your textbooks you can solve a host of ordinary differential equations using PETSc just to get confidence with whether everything works properly or not. So, with this I end this particular lecture and I will see you next time bye.