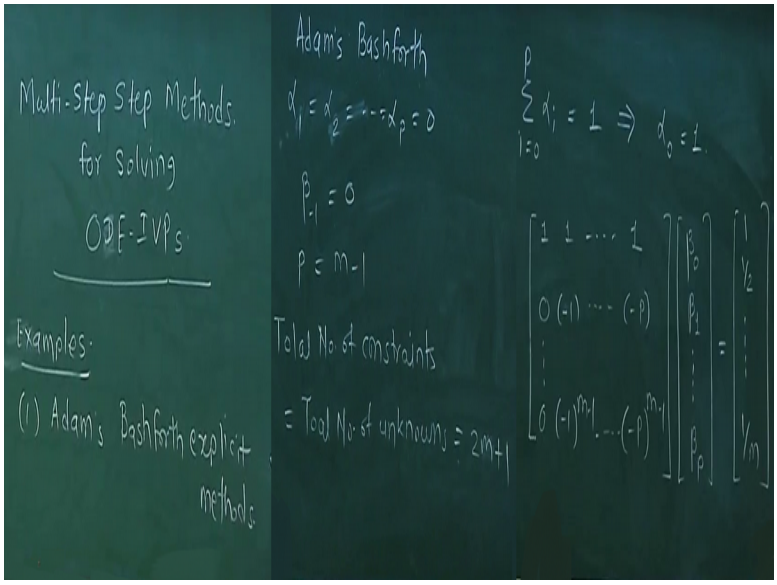
Advanced Numerical Analysis
Prof. Sachin Patwardhan
Department of Chemical Engineering
Indian Institute of Technology - Bombay

Lecture - 44
Solving ODE-IVPs: Multi-step Methods (contd.) and Orthogonal Collocations Method

So we have been looking at predictor-corrector method or multi-step methods for solving ODE initial value problems, and we looked at one particular class last time that was Adams-Bashforth explicit methods.

(Refer Slide Time: 00:48)



So what we have done till now is multi-step methods for solving ODE-IVPs, and under this class I have derived constraints that need to be satisfied interpolating polynomial coefficients, and from that we have derived a generic formula for arriving at any method, and I am describing some popular methods. So one of them was Adams-Bashforth explicit methods, so these methods are multi-step methods which only certain class of or it only makes certain assumptions regarding what in the past you had to use.

So it leads to the formula where you set alpha 0=alpha 1 sorry, alpha 1=alpha 2=alpha p=0, so we are not going to use past x values, we are going to use past derivative values. And it is an explicit method, so beta-1=0. And we set p=m-1 okay the polynomial the multi-step p is number of steps=m-1, where m is the polynomial order which you have to fit okay, so we have p+1

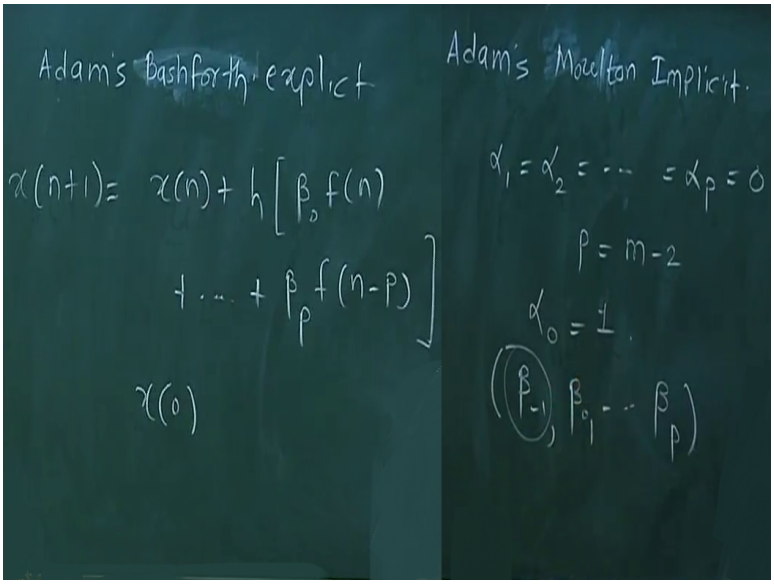
additional equations and total number of constraints this=total number of unknowns, and it is turns out to be 2 m +1, where m is the polynomial order okay.

So this is an explicit method using this first constraint that is alpha i=0 to p alpha i=1 together with the assumptions that we are not going to use past x, this implies that alpha 0=1 okay, and the remaining coefficients can be found out by setting up the constraints. So I am just writing the final from here, so I have to solve for alpha 0, I have to solve for beta 0, beta 1 to beta p. I am just setting up those constraints which we were derived earlier for the coefficients.

And then once I solve for beta 0 to beta p and alpha 0, I get a particular method multi-step method. The final form of this multi-step method, well one assumption when I wrote all these equations I forgot to mention, when we have derived those constraints on alpha and beta okay, we have made an assumption that is 0 raised to 0=1, you get 1 term in that those constraints, it is assumed that 0 raised to 0 is 1 for the sake of writing those constraints.

So whenever I do not want to say anything about whether 0 raised to 0 is truly=1, for these constraints whenever 0 raised to 0 comes substitute by 1 okay that is what makes it easy to write these constraints okay, do not assume that this is this equality is this is only for these constraints, which we already derived okay.

(Refer Slide Time: 06:48)



So Adams-Moulton explicit method finally we looked like this x n +1=x n +h beta 0 f n +beta p f n-p okay, the problem arises at time 0, at time 0 you do not know what are the derivatives in the past, one simplification you can do is that at time 0 you can use all the past you can make assumption that x 0 was the value which was also prevalent in the past, so at all the past instances before 0 you can compute derivative using x 0 okay.

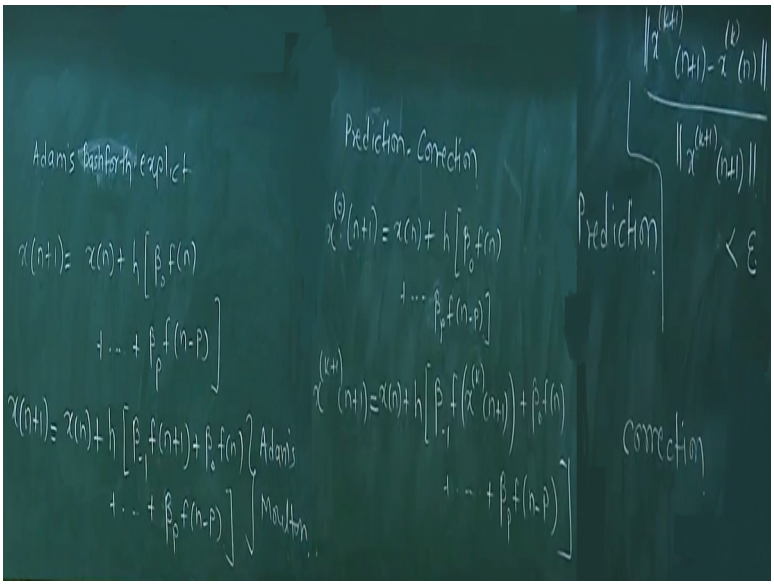
So the problem will vanish after p steps okay, first p steps you have problem because or the other way to think about it is that you have to give first, you have to when you initialize this algorithm we will have to give for p values p initial values okay, one simple way of the give p initial values is to set them=x 0. If you set and you start integration from p+1, so you have to start your algorithm from p+1, 0 to p you will have to specify and those can be set=x 0, and then you can pick up your algorithm.

After sometime okay you will have past values and okay, so these are explicit algorithms okay, just a correction, so here this is not Adams-Moulton this is Adams-Bashforth, Adam-Bashforth are explicit methods, so these constraints here are for Adams-Bashforth. And Adams-Moulton are implicit methods okay, so these constraints are same we are not going to use past x values okay, we are going to use past derivative values okay.

And we set p=m-2, and well I will not set up the equations for beta 1, beta 0, so you now the unknowns are now not just beta 0 to beta p, but these are implicit method for beta-1 is not 0 okay. So the unknowns well the first equation will give you alpha 0=1 okay, and you have to set up equations remaining equations for beta-1, beta 0, beta p you have to set up constraints for these and solve them.

You will get a matrix equation in these unknowns you have to solve for beta-1 to beta p okay, it is an implicit method, so you will get only difference between the implicit and explicit method is you will have a, of course this beta 1 to beta p which you get by this approach are not going to be identical with Adams-Bashforth, so these are 2 different approaches, you will get 2 different set of coefficients okay. So this will give you an implicit method, so I will just go there and write it down.

(Refer Slide Time: 11:38)



So this is my x n+1=x n +h beta-1 f n+1+beta 0 f n, well just because you are using the same notation alpha beta or beta here does not mean these 2 will give you same, this is Adams-Moulton. So this is an implicit method f n+1 appears here on the right hand side, this is an explicit method okay. What is typically done is that when you want to solve or when you want to get the solution by implicit method okay, you use a corresponding explicit method to generate the initial guess.

Because this is the iterative this has to be solved iteratively right, we saw this we have seen this in implicit Euler, explicit Euler, we use explicit Euler to initialize implicit Euler or trapezoidal rule. So likewise in this particular case okay we are going to use this to give initial guess good initial guess for this okay, so now this is done in 2 ways okay, one way is to do iteratively okay. And the other approach is non-iteratively, so which means you do a prediction and then you do a correction okay.

So I will just describe this prediction correction first okay. So this is one is non-iterative method, in non-iterative method you just do one prediction, so my prediction is going to be x tilde n+1=x n +h beta 0 f n, I am calling this prediction as x tilde okay, and I am going to use this prediction to do a correction. Now my correction is x n +1=x n +h beta-1 f x tilde n+1 okay, I am not going to do an iteration in this by this approach.

I am just going to use x tilde n+1 which was computed here, do it only once substitute here okay, since this is now known to me okay I can compute this okay, using this I am going to compute x n+1, so + okay. So if you do it this way only once it is non-iterative okay, this step is called as prediction step, this is called as correction step, prediction correction, prediction correction okay. Suppose you do not want to get into iterations at every point at least do a good prediction and do a correction okay.

Well of course best thing would be to do iterations, so in that case the same approach can be used except this second part will be iterative okay. So in that case the way I would change this algorithm is, so this is non-iterative algorithm. Iterative case I would call this I will go back here and call this my initial guess x 0 n+1, and then I will change this to x k n+ 1, and this side, this is still prediction correction, except now the correction is iterative okay.

And only first time this prediction is used as the initial guess, next time this itself will feed to itself, and then you will wait for the convergence to occur okay. And then we first look at this x k+1, this to become smaller than some epsilon, where epsilon is the limit 10 to the power -8 or something small number. So when you do prediction correction iteratively you use this only once the prediction only once, and then you keep doing iterative calculations till you get convergence.

If you just want to stop with prediction correction okay that is also good enough many times, and you typically use same order algorithms for initializing. So Adams-Bashforth and Adams-Moulton can be used together in this prediction correction form okay. There is one more class of algorithm, now as I told you that Adams algorithms which use past derivative values okay, what you should do of course when you write a program okay you should memorize past derivative values.

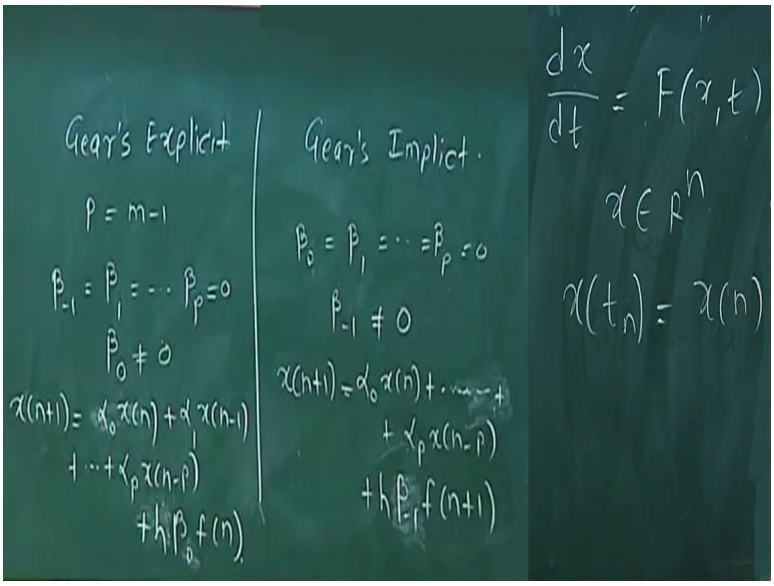
You can store them in some array okay or multi-dimensional array, and then you should not compute it every time okay, you should store it in the array and update the array every time you move in time okay, you have to remember past p values okay, when the new value comes the

new derivative gets in old derivative goes out, you can create a matrix kind of a structure. And you can write an efficient algorithm for doing these calculations okay.

Well what happens when you go to multi-dimensional case where you have vector differential equation, we use exactly same approach, we use the same coefficients which are derived using scalar case, and just instead of f which is scalar here we will substitute by f which is vector nothing is going to change, there are no separate derivations for the multi-dimensional case okay. Multi-dimensional case we just use the coefficients that you derive for the scalar case okay, f here will be a function vector, x here will be a vector, x n+1 will be a vector and so on.

So that is the only difference which comes it when it comes to multi-dimensional methods. Now another class of methods like Adams method another class of methods which are very, very popular are Gear's method.

(Refer Slide Time: 19:58)



So Gear's explicit method and Gear's implicit method. In Gear's method we do not use past derivatives we use past x okay, we do not have to save past derivative values, we have to keep saving past x values, anyway past x values we are saving because you want to see the profile. So Gear's method are, in Gear's explicit method we put the constraints that beta -1 beta 1 up to beta p=0, we of course do not set beta 0=to 0 okay.

So this method would look something like this that is x n+1=alpha 0 x n okay +alpha 1 x n-1 up to alpha p x n-p +beta 0 f n only 1 derivative value the current derivative value at initial point okay, and then we use x n, x n-1, x n-2, x n-p past p x values okay, but this is an explicit method so only f n **"Professor - student conversation starts"** (()) (22:12) h correct, h times yeah you are right, h times beta 0 f n okay. **"Professor - student conversation ends."**

As you can guess now Gear's implicit would be beta 0=beta 1=beta p=0, so these are the constraints, what is non 0 here is beta-1 is !=0, so here you set beta 1=0 beta 0=0 okay, beta 0=0 up to beta p=0 only beta-1 is !=0. So only the way algorithm changes is x n+1=alpha 0 x n up to alpha p okay. So instead of h beta 0 f n which appears here okay you will get h beta-1 f n+1, this is an implicit method, explicit method.

They can be tied up again if you want to implement Gear's method okay in the non-iterative way, you generate x tilde x n+1 using Gear's explicit use it here and do the correction, so prediction correction non-iterative, iterative prediction correction is you initialize your algorithm using this and then solve this iteratively till you get convergence okay. And just to emphasize any of these algorithms, for any of these algorithms if I am solving for dx/dt=f of x t, where x belongs to R n.

And then we are starting from x t n=x n okay, where f is the function vector, all that I do here is if I just go back here all that I do is here this will be my function vector f n, this all will be vectors alpha 0 alpha 1 are same they are not different okay, same thing here this would change to be a function vector. Typically, these iterative suppose you want to solve this iteratively typically you solve it using successive substitution.

The reason being successive substitution will give you a good solution or it will give you a convergence quick convergence provided you have a good initial guess and we assume that we have a good initial guess because of this, so typically it will converge okay, you have a good initial guess and you do not have to. See the advantage of simple successive substitution is that you do not have to compute derivatives right, no derivative calculations involved.

It is just generating a guess and putting it back, and so those are derivative free methods which are computationally less intensive, so you would solve them using successive substitution okay. So as I said these are 2 popular schemes Adam scheme and gear scheme okay, one can create one once own mix you know you might say well I do not like only derivative values or only x values, I want a mix of few derivatives and few and you can do that.

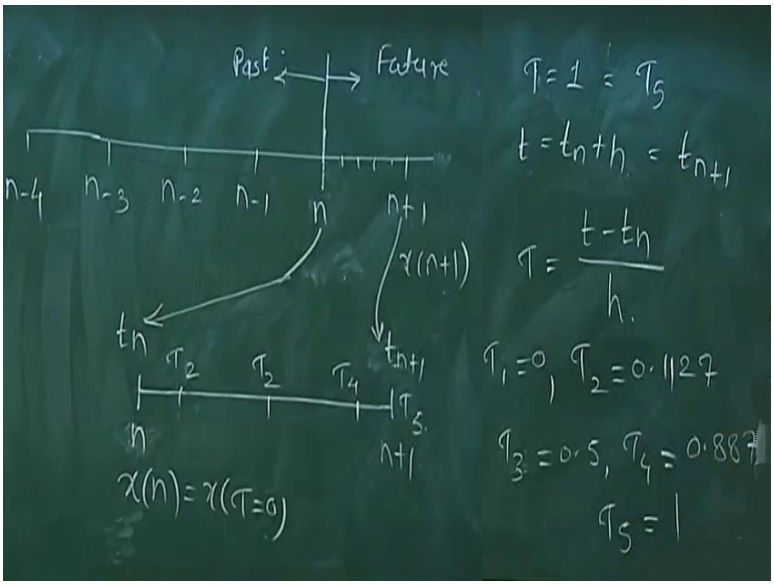
You can choose to generate the method of your liking and develop your own program, develop the coefficients, find them for once and develop a generic program in which you would integrate your differential equation using your own recipe for okay. Just remember what we have learnt is how to arrive at integration algorithms or how to arrive at algorithms for solving ODE initial value problems okay, either through Runge-kutta class or through multi-step class predictor-corrector class okay any one of them will.

And then multivariate case is simply as I said, we use the same coefficients as the univariate case and then instead of derivative scalar derivatives we will substitute by derivative vectors and you get the algorithm. The third method which I promised to do in the class but I am going to leave it more as the reading exercise, because I want to move on to something else, is orthogonal collocation because we are looking at orthogonal collocation very much in great detail okay.

I will just give you an idea what is done now, and then we will move on, the details are there in the notes you should read this okay, because it is just a repetition of what we have already learnt about orthogonal locations. We have learned about orthogonal collocation in the context of solving boundary value problems okay, now I want to use orthogonal collocations idea in the context of initial value problem okay that is the difference I want to use it in the context of initial value problem.

So in the same class that is except one thing which changes here is that the philosophy changes, though is in the same class of interpolating polynomials, we are still going to use interpolating polynomials okay. But when you use orthogonal collocation in some sense the philosophy is similar to Runge-kutta method, so what happens in the Runge-kutta method? What is philosophically difference between multi-step and Runge-kutta methods?

(Refer Slide Time: 28:53)



See in multi-step methods and Runge-kutta methods, suppose you are going from time step n to n+1, this is n-1 this is n-2, this is n-3, in multi-step methods we used x n-1, x n-2, x n-3 derivative values at these points in the past. See this is my current time, this is future and this is passed okay, in multi-step methods we used derivative values or we use the x values in the past. What happened in Runge-kutta method? In Runge-kutta method we created some intermediate points okay.

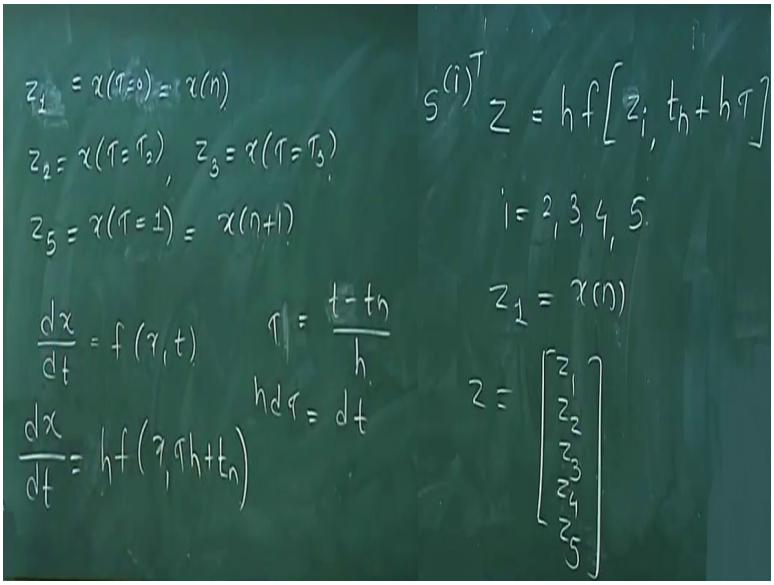
And then we evaluated derivatives at those points, we never worried about what happened in the past okay, we moved from x n to x n+1 by doing some intermediate calculations okay, and what happened in the past is all contained in x n we did not bother about using it again okay. What happens in orthogonal collocation is somewhat similar to what happens in Runge-kutta, in orthogonal locations you are going to still use polynomial interpolation, idea of polynomial interpolation remains there, because it is orthogonal collocation okay.

Except I am not going to use past now, I am just going to use from here to here okay, so what I am going to do now is this section okay I am just blowing it up, this is my time n or this is my time tn or integration instant n and this is n+1 okay. Now what I am going to do is I am going to use shifted Legendre polynomials or the roots of the shifted Legendre polynomials at over this interval, but the problem is this is not 0 to 1 okay.

So what I do is to transform the time axis using tau=t-tn/h, where h is my integration interval okay, and then I have to say that I am standing here at n, so I know x n which is same as x tau=0 okay, and then I am going to place the collocation points now at the roots of the suitable shifted Legendre polynomial inside this interval okay, suppose you place them at you know third order polynomial then this will be the first root will be let us all this okay.

I am going to place roots at tau 1 which=0 okay, tau 1=0, tau 2=0.11, we have seen these roots 0.1127 okay, tau 3=0.5, tau 4=0.8873 and tau 5=1 okay. So I am going to place these knots okay, what is the solution? The solution is the value of x at tau 5 this is x n+1, see my x n+1 where tau=1 that=tau 5, t=tn +h which=tn+1 okay, so once I reach tau 5 okay I get the solution okay.

(Refer Slide Time: 34:46)



And so what I do here is so I am defining this intermediate variables x1=x at time 0, let us all this by some other notation say z1=x at time 0 which is same as or x at tau=0 which is same as x n right initial point okay. Then and I am going to call z2 this is x at tau=tau 2, z3 is x at tau=tau 3 and so on okay. My aim is to find out z5 which is x tau=1 which is x n+1, this is what I want to find out okay, now what I do is I have this differential equation dx/dt okay=f of x t okay.

How will you transform this to tau? So t= or tau=t-tn/h, so d tau= or h d tau=dt right, so this equation will become dx/1/h d tau=f x tau h +tn right okay, and this is my h. I am just going to

multiply this h on the right hand side, so I will just say dx/dt=h times this. Now how do you use orthogonal collocation yeah this dx/dt you have to convert okay, dx/dt you have to convert using these x n t matrices okay, and then instead of working with sorry one mistake this should be dx/d tau.

And then instead of working with x we could work with z okay, so I could work with new notation z okay. Then all that I need to do is to set up these equations that is S i transpose z=h okay, where i goes from 2, 3, 4 and 5, we have taken. See there is one difference here okay, when you were solving boundary value problems okay you have to use 2 boundary conditions okay, here we have to use initial condition.

So you cannot set derivative at time tau=0, because at tau=0 you know the value of z1, z1=x n okay, we know that z1=x n this value is known okay, so what is this vector z here? z consists of 5 elements z is z1, z2, z3, z4 and z5 out of this z1 is known, what are unknowns? z2, z3, z4, z5 okay. What is the solution finally when you solve this is z5, because z5 is x n+1 okay, so you set up these equations, these equations could be non-linear equations okay?

They could be non-linear equations; they could be linear equations depends upon what is f? If f is the non-linear function this will be non-linear algebraic equations, they have to be solved using Newton-Raphson or Newton's method or whatever some iterative methods okay. The difference is even though you are using interpolation polynomial, you are doing function evaluations at intermediate points, at different intermediate points okay.

This is h tau i, and tau i we know what tau i values, tau i values are roots of the shifted Legendre polynomial, so those things we know. So these are at intermediate points we do the function evaluations, we solve this set of non-linear typically non-linear algebraic equations simultaneously. And when you solve it okay the final value that is z5 will be your solution okay, how do modify this for the vector differential equation, I have discussed in the notes you can just go through it, slightly it will become slightly complicated but not too much.

Here, of course you will have to solve this algebraic equations, and then may not be a good initial guess, in which case you have to solve them using derivative best methods okay. There are good collocation based packages available Carnegie Mellon University has put up a package for (()) (41:26) as from Carnegie Mellon University has put up a package on solving large number of differential equations using orthogonal collocation.

You can just download, setup your problem gives number of grid points, it will do all the calculations for you it will also, it has a solver inside which will solve, of course these things you can use when you go to your projects. Now in the course you should not download the package, you should program yourself to understand what is going on. So what is going on is you know intermediate calculations going from n to n+1 okay.

So in some sense philosophically what is happening is similar to that of the Runge-kutta method we are not going to use past derivatives okay. Now so with this method okay we have a wide variety of approaches for solving ODE-IVP, which one do you use okay, so there are 100s of methods now not just one. Runge-kutta method is a class of methods you can derive third order, 4th order.

And here that to within each order depending upon how you choose the free parameters, you will get one method which belongs to second order Runge-kutta method class or third order Runge-kutta class and so on, because they always free parameters. As you have seen here also, there are free parameters if you set certain things=0, you will get some constraints and then you will get a method. So there are so many of methods many methods.

We need to get some insight into their behaviour their convergence behaviour, what is convergence? First of all, I need to know if in certain situations I know the true solution okay, and then I construct an approximate solution using one of these methods, how close is the approximation to the truth? That is one fundamental question okay. And related to this question is how do I use okay interval of integration.

How do I choose my integral of integration? h is the most difficult part in solving ODE-IVP okay. So we will get some insights into this in next 1 or 2 lectures, as to how to exactly what about choosing selecting h okay. So if we are willing to choose h to be very very small, even simple Euler's method will work okay, but sometimes okay this small becomes too small and then it is not useful.

Suppose you are doing dynamic simulation of a chemical plant, some differential equations act on a very fast time scale, some differential equations act on a very slow time scale, choosing 1 h okay which will so you may have to choose h to cater to the small time scale fast dynamics, you may have to choose h very, very small you know milliseconds. And to cater to dynamics of the temperature in a furnace you may have to choose h to be 1 minutes, because nothing happens in 1 hour you know.

So how do you choose h, if you start choosing millisecond you will have too many computations, if you start using minutes you will miss some dynamics okay, so there is a balance. And how do you go about choosing integration step size, these analysis of integration step size gives us insight into you know some comparative behaviour or some relative behaviour of each methods okay.

At the end of it I am not going to prescribe one method ultimately when you actually start solving real problems you will develop your own preferences okay. Some of you will start using Runge-kutta, some of you will start using multi-step, and you will know how to tweak the free parameters or how to choose the integration interval appropriately, so that you can make your algorithm work okay. So there is no one unique you know recipe which will solve all the problems okay.

So you will typically develop your own solutions, I will just mention one approach before we move on to actually getting insights into integration step size okay, so this is called as a variable step size approach. I will just mention this, now before I moved to variable step size approach, is orthogonal collocation idea clear? I have just cased here, I am not derived all the equations, the equations are given in the notes.

And we have looked at orthogonal collocation thoroughly for boundary value problems, only difference here is okay we are solving it for you are using it for initial value problems okay, so just have a look at okay.

(Refer Slide Time: 46:49)

So let us look at this variable step size implementation, the detailed algorithm I will describe in the next class. I will just give you the philosophy is let us say you have reached up to this point, you have started from time t=0 and you are at point tn, and of course you have x n with you, and then you want to move and make the new step okay, you want to make a new step. Variable step size implementation idea is possible only with Runge-kutta class of methods okay.

Not possible or only with the methods in which you are marching I had n time you are not going to use past information okay, multi-step methods variable step size does not work, or it will need a lot of work to make it work in variable step size. Here, in Runge-kutta methods you are just marching from tn to tn +1 okay, so the philosophy is very simple, now the question is I am not going to fix h okay, I want to move from tn to tn +h, what should be h? Okay.

What I do is a simple idea okay, I choose some h some guess h okay, and then I make one move from here to here okay, assuming that step size is h. Then what I do is I assume that step is not h but step is h/2 okay, so I made 2 moves from here to tn +h/2 okay, and then from here to here, so

this is h/2 and this is h okay. Now if the solution I obtained by making 2 steps, and by making one step is not too different, then I accept that h okay, you get what I am saying.

See what I am going to do is in variable step size implementation, I do not know, what is the step size to choose, is it 1 minute okay or is it 10 seconds okay, let us say I start with guest of 1 minute okay. So I make integration from tn to tn+1 minute okay, then I go from tn to tn +1/2 minute, tn +1/2 minute to tn+1 minute, I go to the end point once in 2 steps and once in single step okay, then I compare the result.

If the result is too different okay then I say well I do not accept this initial 1 minute, I will reduce my step size to tn +h/2 okay, now what I will do is I will go in one shot here to here okay, and I will go hopping twice okay. Compare the results, if these 2 results are similar I accept it, if not you know I shrink this further okay. So I take some initial step size, I go there in 2 steps, I go there in 1 step. If the 2 solutions are very, very close I accept that solution, the 2 solutions are too different I shrink the step okay, I reduce the step.

So I might start with 1 minute as my step size, and I might reduce it to half minute okay, to quarter minute, to 1/8th of a minute, till I get this you know 2 solutions matching. One step solution should match with the 2 step solution okay, so this way if you implement Runge-kutta method, you have a very robust method. You do not have to worry about how to select the step size.

It will keep doing lot of calculations but those calculations will help you to give a robust algorithm, which will not fail okay. We will describe this algorithm in detail next class, and we will also get insight into what really matters okay. Well, unfortunately or fortunately again what will reappear is eigenvalues okay, and they will again help us to find our way out okay. So let us look at the convergence aspect in the next class.