

Numerical Methods and Programming

P. B. Sunil Kumar

Department of Physics

Indian Institute of Technology, Madras

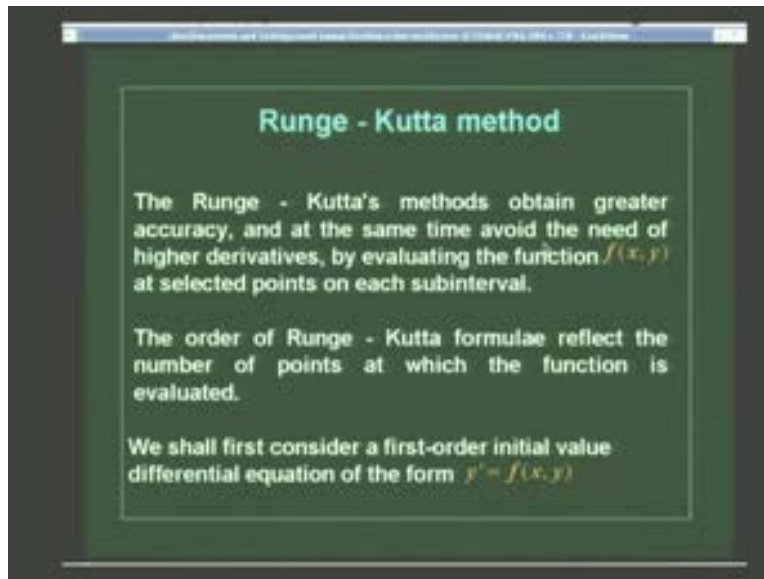
Lecture - 32

Solving Ordinary Differential Equations

Runge Kutta Method

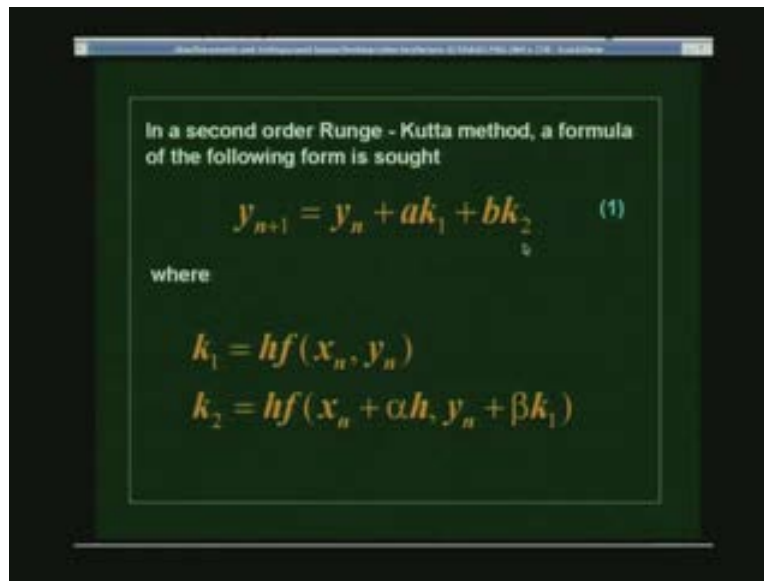
Today we will continue with our discussion on numerical methods to solve differential equations, ordinary differential equations initial value problem that is with all the boundary conditions specified at the beginning. So we were looking at the method called Runge Kutta Method in the last class, in which the function which we had to solve in the differential equations which we had to solve of the form y' is equal to f of x, y was split into 2 steps in the case of second order Runge Kutta Method that is what we looked at in the last time.

(Refer Slide time: 01:51)



So there is, so let us summarize that so we have been writing the differential equations of this form first order differential equations of this form f of x, y . So the prime means the first derivative with respect to x and this is the function of x and y then we said we looked at various different methods of solving such differential equations with all boundary conditions specified at the beginning that is the first order differential equation there is only 1 boundary condition that is specified x is equal to x_0 , that is the beginning as y_0 . So that is the initial value problem that is what we were solving and we looked at simple methods of solving this including the Runge Kutta Method.

(Refer Slide time: 10:06)



So in the second order Runge Kutta Method, we write the solutions as of y_n plus 1 equal to y_n plus ak_1 plus bk_2 remember here or we mean by solving such a differential equations is to actually find, to find out what is the function dependence of y on x is. So y at x given interval x greater than or equal to x_0 to less than or equal to x_1 that is what we wanted to solve. So now we have to solve this numerically, we of course this interval into n interval, n points and then write y of x_n now we will denote this by y_n is that the notation which we use. So y prime means the derivative with respect to x and y x_n is y_n .

So then we wrote y_n plus 1 there is x_n plus the next interval from x_n the next position after x_n that is x_n plus h can be written as we said, y_n plus ak_1 plus bk_2 and we said k_1 is the function at x_n, y_n and k_2 is the function at x_n plus αh and y_n plus βk_1 that was the method, that is that is the second order method Runge Kutta Method in which we actually wrote this as this form there is y_n plus 1 that is given y_n given the y value at n at any point x_n what is the y value at x_n plus 1 that is the next interval x_n plus h is the distance between x_n plus 1 and x_n .

So what is the y value there that is given by y_n plus ak_1 plus bk_2 where, a and b , a and b are some co-efficient just to be determined and k_1 is the function value at x_n, y_n and k_2 is this function value right hand side the derivative x_n plus αh and y_n plus βk_1 where α and β are to be determined. Then the technique we used was to expand this we could 2 things we could say that y at y_n plus 1 that is y at x_n plus h , I can expand it around x_n using a Taylor's series expansion and say that this is equal to Y at x_n that is Y_n plus the derivative of this function Y prime at x_n into h , I could write this in this form and then say the next 1 would be Y double prime of x_n into h square by 2 etcetera, I could expand this in Taylor's series.

(Refer Slide time: 10:16)

$$y_{n+1} = y(x_n+h)$$

$$= y(x_n) + y'(x_n)h + \frac{y''(x_n)}{2}h^2 + \dots \quad (1)$$

$$y_{n+1} = y_n + a f(x_n, y_n) + b f(x_n + \alpha h, y_n + \beta f(x_n, y_n))$$

$$= y_n + a f + b \left[f + f_x \alpha h + f_y \beta f + \dots \right]$$

$$= y_n + (a+b)f + b h^2 (\alpha f_x + \beta f f_y + \dots) \quad (2)$$

$$\left. \begin{aligned} a+b &= 1 \\ \alpha b &= \beta a = \frac{1}{2} \end{aligned} \right\} \begin{aligned} a &= b = \frac{1}{2} \\ \alpha &= \beta = 1 \end{aligned}$$

So that is I would get order h , so remember h , h is the interval here that is x_n plus 1 minus x_n so we could get orders this order h order h square and order 0 terms in this expansion and then we could do we could obtain the same thing by another way that we could take this expression and then again expand the right hand side in the Taylor's series.

So when do that I can write this this we got let me call this as 1 and then we could start from here that is y_n plus 1 as y_n which is y_n plus a times f of x_n, y_n plus b times k_2 now k_2, f is at x_n . So f at x_n plus αh comma y_n plus β at $f(x_n, y_n)$ we call this as f_n now. So now what we could do is just we have expanded this on the right hand side right hand side of this equation we could also expand this function here and do the Taylor's series expansion of that. So that would lead to y_n plus af , we call f is f is at x_n and y_n we call it as f_n , so we call that.

So we say f is the value of f at x_n and y_n and then we could write b times I could expand this around that, that is f plus the derivative of f with respect to x , f_x I call, that is derivative of f with respect to x into αh . So similar as this derivative of f with respect to x times αh and then derivative of f with respect to y and derivative of y . So I will write derivative of f with respect to y as $\frac{\partial f}{\partial y}$ into $\frac{\partial y}{\partial x}$ del x del y so that is 2 times. Okay so I will write this as f times f_y , f times f_x , f_y I am writing, I am writing f_y as $\frac{\partial f}{\partial y}$ by del y into del y by del x that is why I am writing this.

So that is k_1 is f , so just write as βf , so I could write terms all in expansion like that so all these terms. So then I could compare other terms of order this equation from this equation to this equation and then get the co-efficient β b α and I will compare all α terms, order h terms and order h square terms between the equation 2 and the equation 1 and then I would determine a and b that is the what is the method was.

(Refer Slide time: 10:44)

We have to now have a scheme to find the coefficients $(\alpha, b, \alpha, \beta)$. On expanding $y(x_{n+1})$ in a Taylor series through terms of higher order we obtain

$$\begin{aligned}
 y(x_{n+1}) &= y(x_n) + hf'(x_n) + \frac{h^2}{2} y''(x_n) + \frac{h^3}{6} y'''(x_n) + \dots \\
 &= y(x_n) + hf(x_n, y_n) + \frac{h^2}{2} (f_x + \beta f_y) \\
 &\quad + \frac{h^3}{6} (f_{xx} + 2\beta f_{xy} + f_{yy} f^2 + f_x f_y + f_y^2 f) + \Theta(h^4)
 \end{aligned}$$

So let me just summarize that once again here, slide I will write y_{n+1} as y_n plus ak_1 plus bk_2 and then where k_1 is the $hf(x_n, y_n)$ and k_2 is hf at x_n plus αh , y_n plus βk_1 and then I would just expand this term y at x_n plus 1 using Taylor's series expansion. So that is y at x_n plus that is h times f which is the derivative and h^2 times the second derivative etcetera and then I would go back in their expand the terms, the term I would go ahead in the next equation that is I would write this, I would write this right hand side of this equation which is what written in the board and then I would write it as $\alpha h^2 f_x$ plus $\beta k_1 f_y$ plus $\alpha^2 h^2$ by $2f_{xx}$ etcetera.

(Refer Slide time: 12:39)

Substituting the expression for k_1 into (1) and also substituting $k_2 = hf(x_n, y_n)$, we find upon arrangement in powers of h that

$$\begin{aligned}
 y_{n+1} &= y_n + (a + b)hf + bh^2(\alpha f_x + \beta f_y) \\
 &\quad + bh^3 \left(\frac{\alpha^2}{2} f_{xx} + \alpha\beta f_{xy} + \frac{\beta^2}{2} f_y^2 f \right) \\
 &\quad + \Theta(h^4)
 \end{aligned}$$

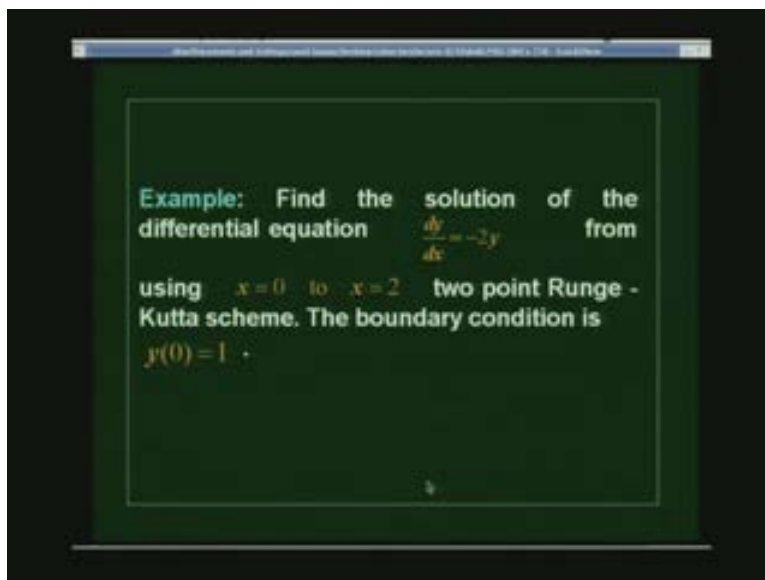
So now get these 2 equations by solving by comparing 1 and 2 and then we would write a equal to b equal to $\frac{1}{2}\alpha$ equal to β equal to 1 as the solution of these equations that is not the only solution of course because there are 4 unknowns a, b and α and

beta but this is 1 of the solution. So we make a choice here. We will make a choice here and we say a equal to b equal to half and alpha equal to beta equal to 1 would satisfy these equations. Okay that is what we have done in the last class let us summarize that here okay.

So basically I take this expansion this, equation which is our solution and expand the right hand side and the left hand side and right hand side independently there is y and plus 1 is y, x plus h which I can Taylor expand using Taylor's series and I can also expand beta k₂ because k₂ is that with term which also I can expand around x_n y_n and then compare the 2 expansions. So orders h square and h terms and then I would get this this term that I would get the expression with a plus b equal to 1 and beta alpha plus b beta equal to 1 that is what I would get.

So these 2 expressions then from these 2 expression I can solve, I get that these 2 expression I can solve get the solution that a equal to b equal to half and alpha equal to beta equal to 1. So that is the summary which we did and that would lead to an equation of the following form.

(Refer Slide time: 12:51)



So we would get the solution as if I use a equal to b equal to half and alpha equal to beta equal to 1 as my solution, as we just written here and then I can write y_n plus 1 as y_n remember, plus a is half so it is half times k₁ plus half times k₂ which is y_n plus half times f at x_n, y_n plus half times f at k₂ was f at x_n plus alpha h, alpha is 1. So x_n plus h, y_n plus beta is again 1 y_n plus 1 k₁, so k₁ is h times f it is h times f so it is h times f that is what we have to obtain. So I forgot to write h here, so there is actually h terms all the sequences we write remember k₁ as h times f of x_n, y_n and k₂ as h times f of x_n plus alpha h plus y_n plus beta k₁ there is h in this steps here okay.

(Refer Slide time: 15:13)

The image shows a chalkboard with the following handwritten equations:

$$a = b = \frac{1}{2}$$
$$d = b = 1$$
$$Y_{n+1} = Y_n + \frac{1}{2} k_1 + \frac{1}{2} k_2$$
$$Y_{n+1} = Y_n + \frac{h}{2} f(x_n, y_n) + \frac{h}{2} f(x_n + h, y_n + hf)$$
$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf(x_n + ah, y_n + hk_1)$$

So I write all that here it become h by 2, h by 2 that is the equation, that is what our y_n plus 1 would be, okay so that is the Y_n plus 1 so we can do we can write it in terms of this basically saying it is Y_n plus half times h f derivative evaluated at x_n y_n and derivative evaluated x_n plus h and y_n plus hf . So now remember that this is nothing but our whole predictor corrector terms method which we looked at.

So we said that there are 2 ways of in the first method we looked at simple extension of the Euler method that is in the Euler method we said that there if we want to go to the next terms next position on the curve there we use the, use the function value at the given position and then multiplied by the derivative of that function, add to that the product of the derivative and the interval that was the Euler method.

That is in the Euler method, we choose to write y in the Euler method we had Euler we wrote y_n plus 1 as y_n plus h times f of x_n , y_n and in the predictor corrector method we said simple extension of this we said I can evaluate in to 2 steps that is 1 is this then I would say that I would evaluate.

So this I call as y_n 0 and in the predictor corrector predictor corrector method, we wrote it as y_n plus 1, 0 as y_n plus h times f of x_n , y_n and then we wrote y_n plus 1 as y_n plus h by 2 into f at x_n y_n plus f evaluated x_n plus h , y_n plus 1, 0. So in that method we had a predictor value for Y_n plus 1 using the derivative at x_n , y_n and then we corrected that by taking a mean of these the 2 derivatives that is derivative evaluated x_n , y_n and the derivative evaluated at x_n plus h y_n 0 that is the predictor value and then we said that if these 2 does not compare this is y_n plus 0, y_n plus 1 did not compare then we go back and then do this again that was the predictor corrector method iteration.

(Refer Slide time: 19:30)

```
main()
{
    FILE *fp1;
    int i,j,Npoints,n;
    float *x,*y,upper_lmt=2.0,lower_lmt=0.0
;
    void runge_kutta_2();
    char fname[20];
    x=(float *)malloc((n+2)*sizeof(float));
    y=(float *)malloc((n+2)*sizeof(float));
    printf("\n Here, upper limit=%f; lower l
imit=%f\n",upper_lmt,lower_lmt);
    printf("\n Enter no of pts you want:");
    scanf("%d",&Npoints);
                                     10,9-16      8%
```

So here we are doing the exactly the same, we just saying that in the second order Runge Kutta we are saying the exactly the same, we are saying what we did by making this choice a equal to b equal to half and alpha equal to beta equal to 1 is to go back to the predictor corrector and write it as Y_n plus h by 2 into f at x_n y_n plus f at x_n plus h , y_n plus hf by 2 that was the simple our simple scheme of second order Runge Kutta scheme.

So we could we could just look at an implementation of this here, is the code which does that. So this code would do a simple implementation of the second order Runge Kutta scheme. So I have again this code just like our numerical integration scheme here again we have x and y as descritised n points. So I have some given some upper limits as 2 and 0 this integral from 0 to 2 of a function.

(Refer Slide time: 20:59)

```
void runge_kutta_2();
char fname[20];
x=(float *)malloc((n+2)*sizeof(float));
y=(float *)malloc((n+2)*sizeof(float));
printf("\n Here, upper limit=%f; lower l
imit=%f\n",upper_lmt,lower_lmt);
printf("\n Enter no of pts you want:");
scanf("%d",&Npoints);

n=Npoints;
runge_kutta_2(n,upper_lmt,lower_lmt,&my
function,x,y);

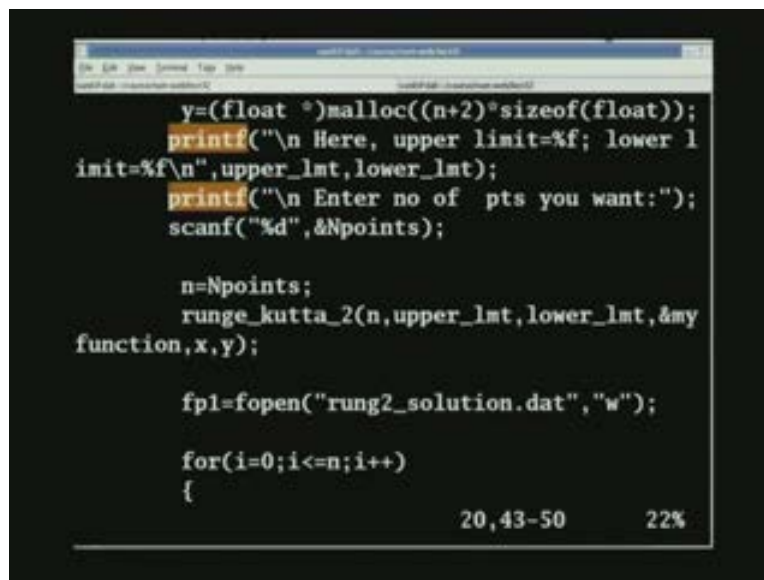
fp1=fopen("rung2_solution.dat","w");
                                     20,38-45      16%
```

So we will see what the differential equations is it takes some differential equation and then we have the first order differential equation and we have to solve the differential equations between 2 limits that is 0 and 2 that is our x_1 and x_0 and I will have end points I will choose end points in between we will see how we choose the end points and then I have 2 arrays here x and y, this is some points is again which I would put the values at which I am going to evaluate the function and that is the independent variable and the dependent variable y.

So first our equation, so we have only y of x is the solution of the equation. So then I have given here some memory to that variables the point is x and y by using malloc function and then I will print out the upper limit and the lower limit in which we are going to solve the equation and that is this point and then we would say this program would ask the number of points which you like to have in between these 2 intervals and that is takes rid out from the screen as end points which is given to n here, so n would be the number of points at which you evaluate use you would use number of points used to solve the equation.

Okay then we had we call this program called Runge Kutta 2, second order Runge Kutta I just denote that I could the function Runge Kutta 2 and to this function we will supply the number of points at which you would like to use the upper limit the lower limit and a point to the function call my function because I need to tell the program which function at which is, what is my right hand side which is because I am going to solve an equation of the form.

(Refer Slide time: 21:17)



```
    y=(float *)malloc((n+2)*sizeof(float));
    printf("\n Here, upper limit=%f; lower l
imit=%f\n", upper_lmt, lower_lmt);
    printf("\n Enter no of pts you want:");
    scanf("%d", &Npoints);

    n=Npoints;
    runge_kutta_2(n, upper_lmt, lower_lmt, &my
function, x, y);

    fp1=fopen("rung2_solution.dat", "w");

    for(i=0; i<=n; i++)
    {

```

20,43-50 22%

I am going to solve an equation of the form Y prime is equal to f of x, y. So I need to tell the program what this function is, that is I would write a program separately for that function for that and I say, I am supplying to this program that is Runge Kutta second order the point to that function and remember, when you supply point to a function like that we need to declare it outside the main program that is what I have done here.

So that the function is declared here as my function and it takes 3 variables and that is all 3 are pointers, **is the 3 points functions**. Okay so then we call that we call pointer to that function and then x and y values and this program should return to the x and y that is the function value evaluated at x that is what this program would written. Okay given the number of points which you like to use the upper limit the lower limit and pointer to the function which would give the right hand side of my differential equation $y' = f(x, y)$ then I would this print out this into a solution a file is called Runge 2 underscore solution dot dat.

(Refer Slide time: 22:17)

```

printf("\n Enter no of pts you want:");
scanf("%d", &Npoints);

n=Npoints;
runge_kutta_2(n, upper_lmt, lower_lmt, &my
function, x, y);

fp1=fopen("Runge2_solution.dat", "w");

for(i=0; i<=n; i++)
{
    fprintf(fp1, "%f %f\n", x[i], y[i]);
}
fclose(fp1);
    
```

22,13-20 25%

(Refer Slide time: 22:54)

```

y[0]=1.0;

for(i=0; i<=n-1; i++)
{
    *xx=x[i];
    *yy=y[i];
    derivs(xx, yy, dydx1);
    yn=y[i]+xp*(^dydx1);
    *xx=x[i+1];
    *yy=yn;
    derivs(xx, yy, dydx);
    y[i+1]=y[i]+xp*((^dydx)+(^dydx1))/
    2.0;
}
    
```

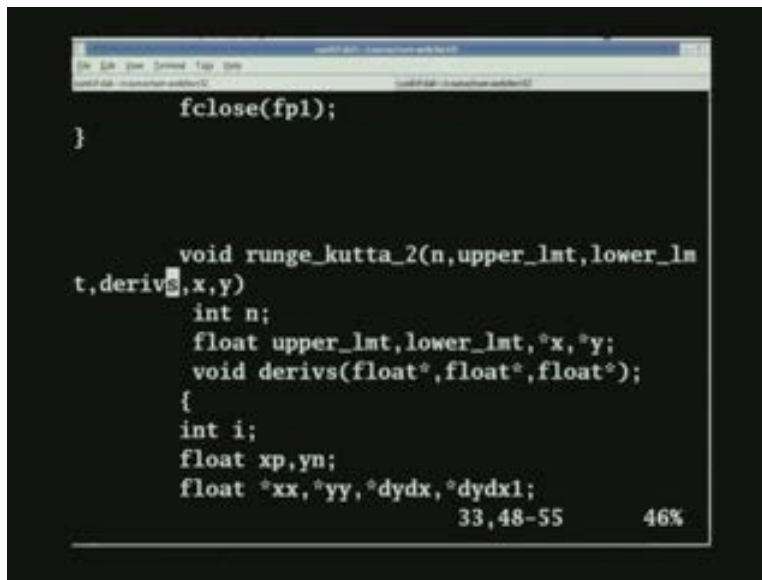
58,26-33 86%

So now what is this program to this Runge underscore Kutta underscore 2 program just simply does the second order differential equation that is it just solves it write the solution as the y_n as y of i plus the interval times dy/dx and then y of i plus 1 as y of i

plus the interval times dy/dx that is derivative evaluated at x_n and y_n and the derivative evaluated x_n plus h and y_n plus the beta n plus f that is what we will see here.

So first the program recall as this program Runge underscore Kutta which we call this program and then that will again the structure of this program is the following that we are supplying to the program the number of points we would like and upper limit lower limit and the pointer to the function which would give the right hand side and x and y .

(Refer Slide time: 24:50)



```
fclose(fp1);
}

void runge_kutta_2(n, upper_lmt, lower_lmt,
t, derivs, x, y)
int n;
float upper_lmt, lower_lmt, *x, *y;
void derivs(float*, float*, float*);
{
int i;
float xp, yp;
float *xx, *yy, *dydx, *dydx1;
33,48-55 46%
```

So and then this program I am going to call in this so first determines what the x values to use so that x values is used as lower limit plus i times h is the interval that is h of our equations is the interval between 2 points, I just divided that determines the intervals by equally dividing the whole interval upper limit to lower by n points, this is not required. We could have unequal intervals but we in this case simple division of equals intervals and then I have my x values. Okay I start with my boundary condition which is $y(0)$ is equal to 1.

So I am going to solve this with boundary condition $y(0)$ equal to 1 and then I would first call that function okay now the $derivs$ is my function here okay so $derivs$ is the function which gives me the derivative of the right hand side.

Okay so you remember this program, I pass to this program a pointer to a my function here. So I call my function and this program receives it as $derivs$ here. So that when I supply here a pointer to the function which is the right hand side of my differential equation as my function and this function here that is the Runge Kutta program is actually receiving it as $derivs$. So this is and then it call the $derivs$ with the x value the y value at which I want to evaluate the derivative and this function would written the derivative at that point x and y , so both xx and yy a point is and dy/dx 1 is also a pointer. So I have allocated memory for all of that here dy/dx xy dy/dx etcetera.

So I give putting some values into xx and yy that is x_i and y_i the starting point i is equal to 0 let us say, i equal to 0. I put in the values xx and yy and it causes the $derivs$ and

the derives would written that the value at where is the derivative at that point x_n and y_n . I do the first step here that is to calculate k_1 .

So the k_1 is now I call y_n here, so it is y_1 plus x times dy/dx 1 that is would be that is that is used to evaluate the first point I need to, remember compute this quantity to compute k_2 I need to first know this point that is y_n plus beta k_1 , k_1 is this. So h times $f(x_n, y_n)$.

(Refer Slide time: 26:03)

```

for(i=0;i<=n-1;i++)
{
    *xx=x[i];
    *yy=y[i];
    derivs(xx,yy,dydx1);
    yn=y[i]+xp*( *dydx1);
    *xx=x[i+1];
    *yy=yn;
    derivs(xx,yy,dydx);
    y[i+1]=y[i]+xp*(( *dydx)+( *dydx1))/
2.0;
}

```

So I need to compute this quantity first I make a call to the derivative program which would return to me this function, this function here and then I would multiply that by h and then add to this one, beta is 1 remember. So I get this then I make a call again to that program with this y_n and then this x value and then I would return written k_2 and then I substitute both back here I get the y_n plus 1 here that is what I am doing here.

So first take the x and y value that is x_0 and y_0 , i equal to 0 and then I evaluated the derivative there and then I computed this y_n that is y_0 plus x times derivative and then went to the next point x_{n1} now i equal to 0 x_1 and then I use yy this value and then evaluate the derivative again and now compute my y next as the y_i plus the derivative times.

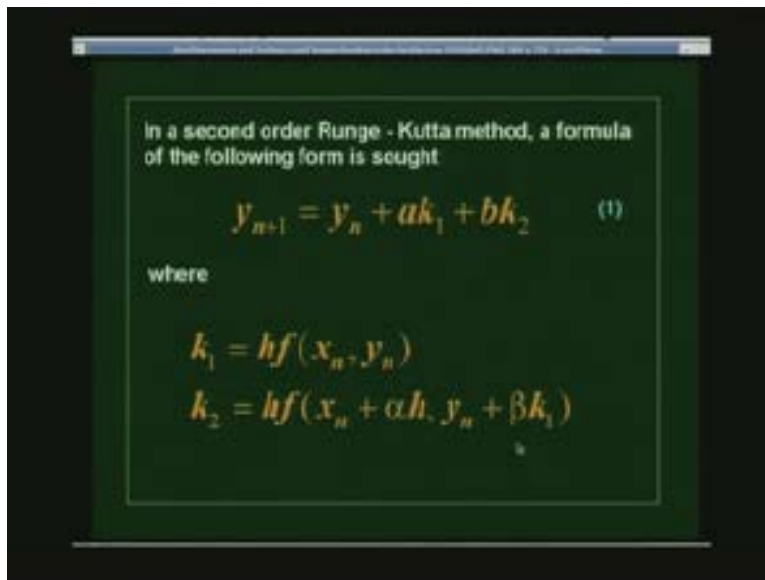
The interval times the 2 derivative e 1 with x_n, y_n another multiplied by h , h is outside here so 1 line derivative evaluated at x_n and y_n and this is evaluated at x_n plus h that is x_i plus 1 times, y_n times h into the derivative at x_n, y_n . So that and then I divided by 2 and now this program will then the loop continues for all i going form 0 to n minus 1 that is n points will be evaluated and would that return that to the main program. So what is my function here, when this calls derives remember is map into my function.

So my function is this simply saying minus 2 y . So what are the equations I am solving here is simply $y' = -2y$, like we have done before, and with the boundary condition that $y(0) = 1$. So that is what we are finding trying to solve.

So we solve this equations using this several scheme that is what the program is, and we can just see how it would run, okay now that is between the upper limit 2, lower limit 0 and the number of points we would use, we will use just use 5 points to evaluate that and then we can plot this. We can plot this function, so I have created a file there call rung 2 solution dot dat and we know the solutions to the actual the equation is exponential of minus 2 star x that is the actual solution the equation. Okay so we can compare these 2 using that is the solution.

Okay so now here is the function we could change the points now this is red parts other 1, we got the solutions of the second order Runge Kutta with these 4points, 5 points here 1, 2, 3, 4, 5, 6 points evaluated and that is green line is the actual solution to the equation so far.

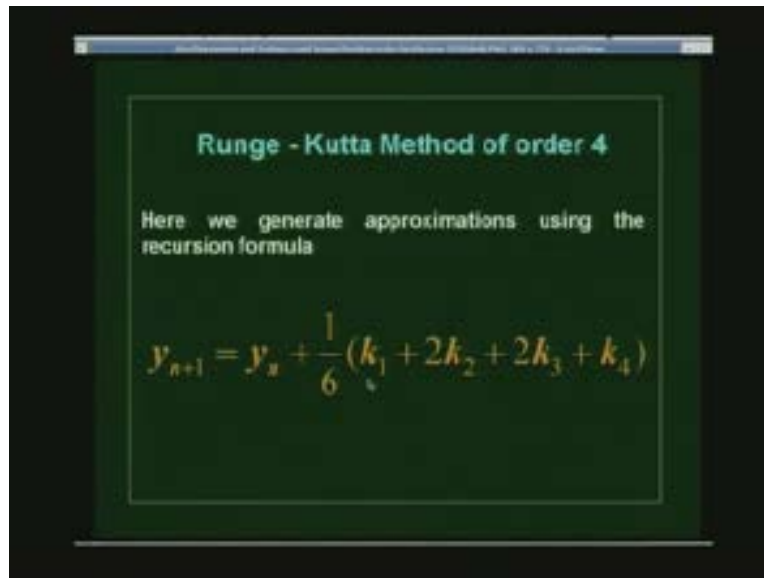
(Refer Slide time: 31:26)



So we can see that the second order Runge Kutta is not all accurate 4, 5 point integration if you want more accuracy we have to go to higher number of points which is not surprising because this method is just a predictor corrector method, just 1 loop and so accuracy is not expected to be extremely high.

So the 1 way to improve the accuracy is to go into larger number of points here we have more number of points or go to higher order Runge Kutta scheme, when you go to higher order Runge Kutta scheme we would use instead of just using a function like ak_1 plus bk_2 we would now use more derivatives that is, now we will evaluate derivative at more number of points. Here now, we are evaluating derivative only a 2 points instead of higher order scheme, we would evaluate derivative at higher number of points. Okay that is the higher Runge Kutta scheme. We can see that here for example the 4th order Runge Kutta scheme we would write y_n as k_1 plus 2, k_2 plus 2, k_3 plus k_4 .

(Refer Slide time: 32:10)



So again, so we will write that now we see we have to evaluate the these are all derivatives the right hand side the function of the differential equations see evaluate now at the 4r parts and it is called the 4th order Runge Kutta scheme function. In this 4th order Runge Kutta scheme, we are going to write Y_n plus 1 as Y_n plus 1 by 6 times k_1 plus 2 k_2 plus 2 k_3 plus k_4 .

So this the this numbers that is 6 and 1 here 2, 2, 1 this is evaluated this also arrived at using a similar scheme as what we have done for the second order scheme that is we would write y_n as y_n plus 1 as y_n plus and we say ak_1 plus bk_2 plus ck_3 plus dk_4 and then we would write k_1 as the function value itself there and then k_2 as the function value evaluated at x plus alpha h plus y plus beta k_1 and then the k_2 , k_3 would be then the function value evaluated at x plus alpha some alpha $1h$ plus beta $2k_1$, k_2 etcetera and then we will have to determine all those co-efficient by a Taylor's expansion. But now we will again, we have to go to order higher than h to get this to get this answers.

So that is rather cumbersome but it can be obtained and then we would get this co-efficient 1 solution to that would be of this form the co-efficient here are 1 over 6, so this will be 1 here 2 and 2 and 1 that is what the 4th order scheme. I am not going to all the derivation here but that is the scheme the scheme would be same as what we did for the second order Runge Kutta scheme.

So now this is what k_1 here is f at x_n , y_n and k_2 would be f at x_n plus h by 2, h by 2, y_n etcetera that is what we have to write. Okay so that the scheme is the full scheme is the following, so y_n plus k_1 by 2, so y_n plus k_1 by 2 and k_3 we would need this we just write this on the board here that is h times h times f at x_n plus h by 2, y_n plus k_2 by 2. So and then k_4 would be hf x_n plus h , y_n plus k_3 that is that is the solution we have to write y_n plus 1 as y_n plus 4 derivatives. So evaluate the derivatives at 4 times.

(Refer Slide time: 35:21)

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$$
$$k_3 = hf\left(x_n + \frac{h}{2}, y_n + k_2\right)$$
$$k_4 = hf(x_n + h, y_n + k_3)$$

Okay and then the orders to determine this these co-efficient sitting here Taylor expand all the way to h^4 accuracy we are going to get that much here okay and then we have k_1 as h times the function value and k_2 as h times the function value evaluated at now x_n plus h_n by 2 half the interval y_n plus k_1 by 2 and k_3 is now h_n the function evaluated at x_n plus h_n by 2 again but now y_n plus k_2 by 2 using this derivative and then k_4 is the function value at x_n plus h and then y_n now is obtained using this derivative as y_n plus k_3 .

So that is the that is the 4 values k_1, k_2, k_3, k_4 . Okay so then we use that k_1, k_2, k_3, k_4 and then write the 4th order Runge Kutta as y_n plus 1 by $2, k_1$ plus $2, k_2$ plus $2, k_3$ plus k_4 . So we can see this also implemented in this simple code, so the code is very similar what we had just used not different.

So it is again same way we have first part of the code is anyway now not different, so everything is same we just call here now instead of calling earlier Runge 2 and now we are calling Runge 4 and again pass the number of points required the upper limits lower limits and pointed to the function which supplies right hand side the differential equation and we have the same the right hand side of the differential equation that will try to solve dy by dx is equal to minus $2y$ with the boundary condition y of 0 equal to 1 and then we would get so now this program in this, in this Runge 4 program which receives all of that.

Okay will have to evaluate the function value at 4 points that is what done here it uses the derives that is the my function which would supply to me given the x and y what is the derivative on the right hand side that point is now that is this function. So here I supplied to start with the first part determine the k_1 , I supply x and y and I got the k_1 here that is my this my k_1 then I say that by I do that in 4 different steps.

(Refer Slide time: 37:18)

```
for(i=0;i<=n-1;i++)
{
    *xx=x[i];
    *yy=y[i];
    derivs(xx,yy,dydx);
    y[i+1]=y[i]+xp*( *dydx)/6.0;

    *xx=x[i]+xp/2.0;
    *yy=y[i]+xp*( *dydx)/2.0;
    derivs(xx,yy,dydx);
    y[i+1]=y[i+1]+xp*( *dydx)/3.0;

    *xx=x[i]+xp/2.0;
    55,16 75%
```

(Refer Slide time: 38:10)



So first I do the way I do this program is to write each type as first I compute this and I say my Y_n plus 1 is this and do first this sum and then to this sum I add $2k_2$ by 6 that is k_2 by 3 and then that to add k_3 by 3 and then I add k_4 by 6 that is the other way which we are going to do that in this in this program I am going to start with saying that y_n plus 1 is equal to y_n plus k_1 by 6 and then I would say, y_n plus 1 is y_n plus 1 plus k_2 by 3 and y_n plus 1 equal to y_n plus 1 plus k_2, k_3 by 3 and y_n plus 1 is y_n plus 1 plus k_4 by 6. So just doing this in a serial order.

(Refer Slide time: 39:01)

```
for(i=0;i<=n-1;i++)
{
    *xx=x[i];
    *yy=y[i];
    derivs(xx,yy,dydx);
    y[i+1]=y[i]+xp*(^dydx)/6.0;

    *xx=x[i]+xp/2.0;
    *yy=y[i]+xp*(^dydx)/2.0;
    derivs(xx,yy,dydx);
    y[i+1]=y[i+1]+xp*(^dydx)/3.0;

    *xx=x[i]+xp/2.0;
    60,16 75%
```

(Refer Slide time: 39:32)

```
*yy=y[i];
derivs(xx,yy,dydx);
y[i+1]=y[i]+xp*(^dydx)/6.0;

*xx=x[i]+xp/2.0;
*yy=y[i]+xp*(^dydx)/2.0;
derivs(xx,yy,dydx);
y[i+1]=y[i+1]+xp*(^dydx)/3.0;

*xx=x[i]+xp/2.0;
*yy=y[i]+xp*(^dydx)/2.0;
derivs(xx,yy,dydx);
y[i+1]=y[i+1]+xp*(^dydx)/3.0;

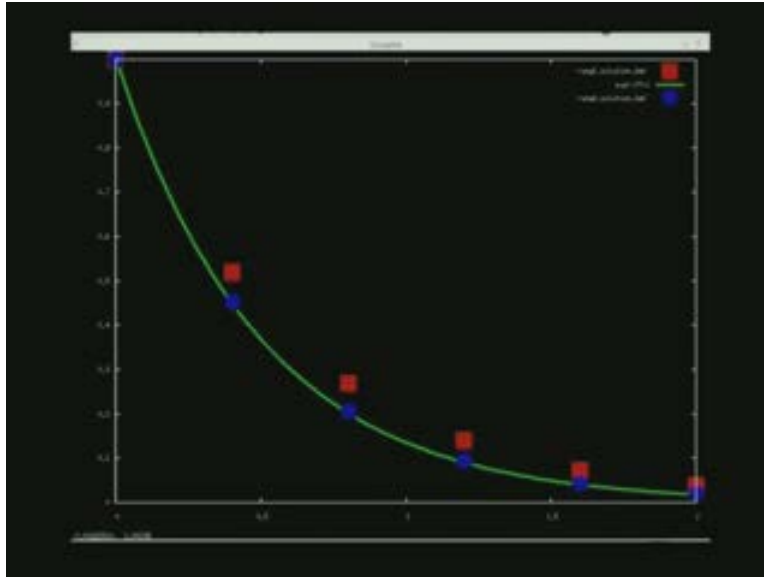
65,18 80%
```

So first i initialize y_{n+1} to this and then keep to that k_2 by 3 k_3 by 3 k_4 by 6. So that is what the program is doing, is this just starting from that y_i plus 1 is y_i plus x times derivative that is the function value divided by 6 and then it will shift now the xx to x plus h by 2 and y to y plus the h times the derivative that is k_1 by 2 and again I will call the same program that is supply to new x value, y value it supplies the new derivative and then whole y_{n+1} , now I am adding that divided by 3 that is h times this derivative evaluated with new x and y value.

Okay i divided by 3 and I do the steps once again with again shifting the x to that and y , x is again taken as the same x is x_i plus xp by 2 y is now y of i plus xp times now this derivatives evaluated here divided by 2 and then I supplied that and then I get a new

derivative and then I add that to y_n plus. So I do that 4 times till the last points where x is now x_i plus h that is x_i plus 1 and y is y_i plus this derivative evaluated here and then I compute the new derivative at that point and add that to the function. So that is the Runge 4, we can run the Runge 4 here.

(Refer Slide time: 41:12)



So we could again supply 5 points to that, okay and we can now upload that along with this function with Runge 2 we will have will also upload Runge 4 solution. So compare the by plotting the solution we got using a 5 points, second order Runge Kutta and 5 points, 4th order Runge Kutta and that is what we do here and this is the plot we obtain. Now green line is the actual solution that is exponential minus $2x$ to a differential equation that y prime equal to minus $2y$ with boundary condition y of 0 equal to 1 and the red squares here are the 1 which obtain with 4th order Runge Kutta 5 points, we use into we got 4th order Runge Kutta here and now this blue circles are the one which we obtain with 4th order Runge Kutta using again 5 points.

So we can see the difference in the accuracy is quite remarkable that we get into by 4th order Runge Kutta we get much more accurate values. So now that is we see that increasing the order of the Runge Kutta scheme but definitely increase the accuracy. So another way to increase the accuracy would be to use the lower order scheme but then evaluated the function value at use more points in some sense both require larger number of reevaluation of the function. So that is now we do 4th order Runge Kutta we are actually evaluated the function valued 4 different points.

Okay while a second order Runge Kutta we are only evaluating the function at 2 points in the in this interval between x_n and x_n plus 1. So that 1, okay we see the this this is 1 way of doing this increasing the accuracy but then again there is a question of what should be the interval between these 2 points should be, okay should I use the same interval for all the points which is what I have done here between if I look at the interval between x_n and x_n plus 1 they are the same.

So the question is always should be use the same interval I would think that since we are trying to solve an equation of this form that is y' we say f of x, y so wherever the derivative is large okay we should go to smaller steps, finally we are writing the solution as something like this. So then we are saying that we using the derivative and to evaluate the function at the next points we use the function value at that point and the derivative multiplied by some interval that is the basically scheme all of these the derivative multiplied by that interval I would think that the derivative is large and then we should use the smaller interval. So that we get the higher accuracy because otherwise, you could have large here.

So we should be able to write a program such a way that the steps sides that is the h function is not a constant but instead it changes depending on the function value itself that we call an adaptive steps side. So evaluate we look at the function value that is the function that is the right hand side of this equation, okay and see how big it is and then depending up on that we could choose the steps sides that is 1 way of doing it, another way of doing this just say that I will choose a steps side and see what I would get and I would change the steps size and again I will go to the same point at x_n value and then see and compare these 2 that is another way of choosing the steps sizes.

(Refer Slide time: 44:55)



Okay we will choose a step size such that we get a higher accuracy and where the function value is not changing to past we could use larger steps sizes and where the function value changing very past we would say smaller steps sizes whereas looking at this graph I would say that we should be using smaller steps size in this this region where the function value decreasing rather fast while we could use the larger steps size here where it is not decreasing as, so fast okay that is the basic idea beyond what is called the adaptive steps sizes algorithms. So we look at that now.

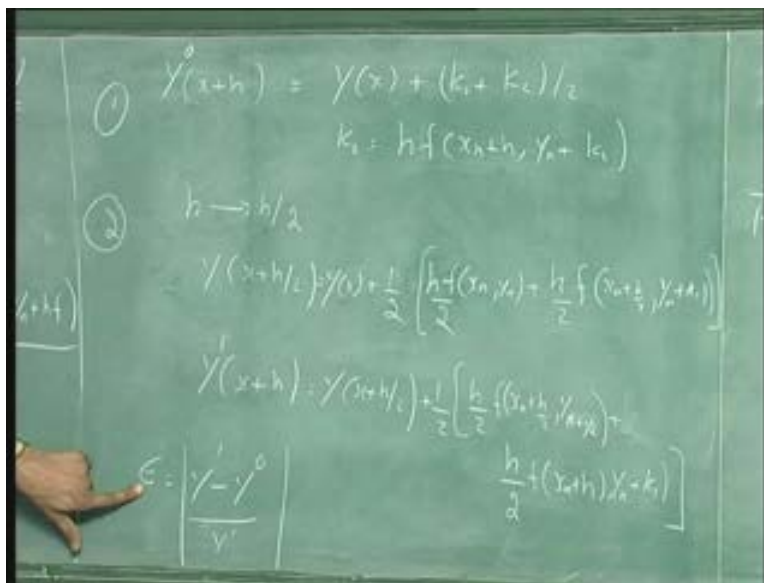
So in adaptive step size Runge Kutta we would get the y_n plus 1 from using a steps, 2 different step sizes we could use the step sizes h first and then we could half the step size and then go to the same x value and that is in 2 steps now and we would now say that I would evaluate y at x plus h by using y at x plus some h times now let me use any

of those scheme let us say Runge Kutta 4, second order Runge Kutta and then I would write this as f at k_1 plus k_2 by 2 right.

So I just use k_1 plus k_2 by 2 where, now to determine k_2 , I would use h times f function value evaluated at x_n plus h and y_n plus k_1 now this h , I use and I get x plus h that may be the first steps and second steps what I do is the take h to h by 2 and then I would evaluate the second steps would use half the step sizes and then I would say I will evaluate x plus h by 2 first, okay from y at x , now k_1 and k_2 evaluated at x_1 plus h by 2. Okay that is function value h times the function value x_n y_n , h by 2 times now I am going by the step size half h is going to h by 2.

So we will do h by 2 steps and then again the function value evaluated at x_n plus h by 2 and y_n plus k_1 and that is what I do and there is half outside again. So that will be second steps now I do this once again then go to y at x plus h , y at now x plus h by 2 and then do the same thing again that is half times h by 2 now the function value x_n plus h by 2 y_n plus half y_n at x plus h by 2. So then h by 2 times f at x_n plus h now, y_n plus k_1 .

(Refer Slide time: 48:33)

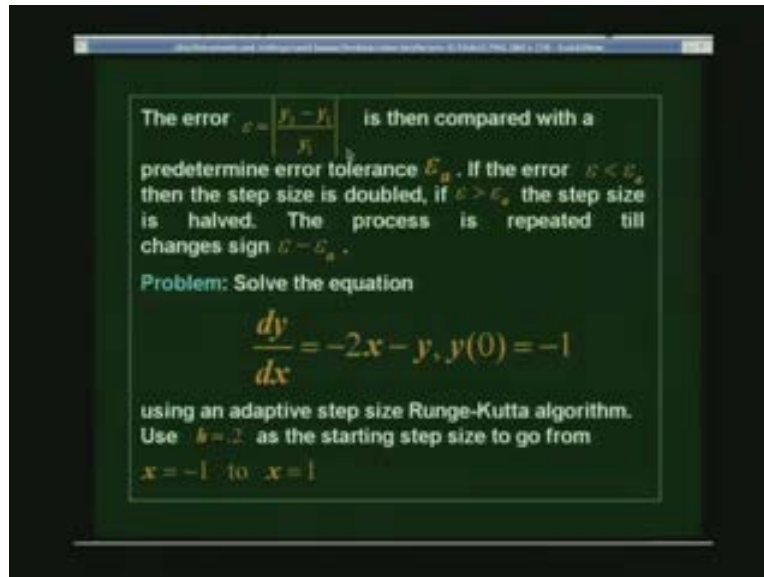


So I do this in 2 steps there is 1 step here and I do it in 2 steps. So let me call this value as y_0 and this value as y_1 and then I compute an error which would be y_1 minus y_0 divided by y_1 mode of that. Okay and then if this method f by values are correct then whether I should go by step h or h by 2, I should get the same answer that is would be this would be 0 or should be very small.

So if this not small then I will go back to loop again and half the step again I will go to half the step again and then I will continue this loop till I get the same that is the idea behind adaptive step size. So what is the summary is the I do I go to x plus h in 1 step or I go to x plus h by 2 steps x plus h by 2 and then x plus h using hx here and x plus h by 2, I go into 2 steps and I compare the values I get from these 2 and compute in error, now this error is significant.

Okay compare to depending about what accuracy you want, if this error is significant then I go back here and again half the steps. Okay, now I say my y_0 is this y_1 and I will compute it y_1 again by passing the steps again going in 2 steps.

(Refer Slide time: 50:55)



So I will keep doing this till I get the same values by this 1 and y_0 and that is the step size and thus determine the step size and then I go to next point of the integration using that step size and as my initial step size prediction and then I will again check this value and if these 2 values are same that is the first steps itself I use these 2 values are same then I can actually increase my step sizes h , I can go from h to $2h$ etcetera.

So I keep doing this that is what we will see in this particular program. So as I said that I will go in step size I choose interval h compute the value y_n plus 1 okay and check all y_n by a single step and half of the step value h and compute y_{n+1} here again y_n plus 1 and compare these 2 and define error here $y_{n+1} - y_n$ by y_n and then I will say I will compare with some predetermine error tolerance the value which I have got.

We can use this equation to solve the differential equation which we just looked at. So that is given here in this program. So I am using the 4th order Runge Kutta using adaptive step size here. Okay now this program is slightly longer because we have to determine step size, I have written it slightly differently.

Now this function is going to call Runge Kutta order 4 now with adaptive step size that is what we are going to call okay I call this Runge Kutta 4 now this with adaptive step size in this part we are doing only this now I have to cannot divide my interval now into some predetermine number of points. So instead of I will ask the program as for the step size initial guess to the step size, you give some step size and then the program will determine automatically determine by using this which I just described what is the actual step size, optimum step size to be used. So the only thing is to supply we have to determine the step size from the program.

So we cannot divide into the number of intervals initially, so what we do is we will go by step by step it starts from x_0 in this program and then we call this Runge Kutta order 4 which will return to the y value at the next point and what be the next point is and that will be determined by what steps it will use right.

So that is the what we have written we just start with give some x_0 as lower limit, okay the y value this is the boundary condition y_0 is 1. So this is the boundary condition I start with this boundary condition and I call this program which would return to me the x and y value and the step size it used to determine the x and y value at next points okay.

So that is the and then I use that as the next initial value, I use the initial value to get my next points and I use that as the initial value call this program again and till I do this till the, it will reach the upper limit. Now I have to go step by step. So we use the x initial value x_0 and y_0 and go to the next step using the this program, okay and which will return to me the step size used and I use that and then go to the next and now I use the x_1 y_1 which would return it will return the x_1 y_1 what is the next point and the y value there use that point and go to the next 1 and determine the x and y the next points etcetera.

Now how does this program determine the step size the method I describes. So it starts with step size which you supplied this program we call that which is step size which we kind of predicted or we guessed then we supply the pointer to the function and xy value this is the thing which would return you come here and then it would use the x_1 y_1 value as the value which we supplied the initial values and then we have tolerance which also we have to set in, set it here as 10 power of minus 2 is my tolerance and then it would determine the first step x_n plus h and it will do this 4th order Runge Kutta points up starting from here.

So we supplied the initial value and it is going to do x_1 as x_1 plus h , x_1 is the initial value y_1 is the initial guess initial value. Okay so it is going to take the x and y and then compute the derivative that is our k_1 and then increase the step size by h by 2 using the step size which we guessed as an appropriate step size and will compute k_2 and it will compute k_3 and then it would compute k_4 and I will add all of them into y . So that y_1 is at the end.

So this is the same as the 4th order Runge Kutta we just look at so once we have the that y value this we now here we first loop at, i t is equal to 0 that is the first loop it will store that y value as y_n and half the step size and reset the x and y values to the initial value which we just used so reset the initial value to this and half the step size it goes back here again and compute whole things again it will compute twice because i t equal to 1 and then what it does it is this is the i t equal to 0 when i t is equal to 1 it will now store the intermediate value in y_1 and x_1 and starts again. So now it use h by 2 starting from x_0 y_0 and went to x_1 y_1 it computes that once again.

Now it goes to h , okay x_1 goes to h , y_1 is now the initial value at here and it goes back and again computes that. Remember, this h by 2 because I put h equal to h by 2 here that is my initial value and it does 1 more loop and in these 2 loops completes that h and it comes out here, okay and now it has 2 y values it has y_n and a new y_1 and now it compute the error y_1 minus y_1 by y_1 .

Okay now the error tolerance is error is below tolerance and then it will say that the h next to be used 2 times h value of it is greater than the tolerance it will reset the y_n value and the x_1 and y_1 value x_0 y_0 and half the step size further. Okay half the step size again and go back and do that again once more it continues this process and comes out with the x next, okay so now we will do the same thing here.

So now we will do same function that is minus $2y$, d_1 equal to minus $2y$ and we will run this and see what we get. So we will run this program here and we can compare this with our results. Okay, so okay we just use the Runge 4 adaptive dot c will compare the so just 1 initial step size we are going from 0 to 2 will use the step size as 1 okay and it has d_1 something and we will just plot that here is the plot of the blue point here the adaptive step size we can see that the interval is now not the same all throughout larger interval and smaller interval there, so and the right point which you obtained here from the Runge Kutta 4 using 5 point integration.

So here we see that again we got a very high accuracy we just started with step size 1 which is different from what we use 1 because the whole interval is 2, so we suggested use step size 1 and which obviously and not satisfactory and g_1 back and chosen a step size which is which fits into the accuracy, you want that 10 power of minus 2. So now that is the different methods which we have to use for solving a first order Runge Kutta, first order differential equation and we have seen that we could use the Euler scheme or use the predictor corrector scheme or use 2 different orders of Runge Kutta scheme, now that is all for the first order equation.

So what we have to do a higher order differential equation let us say a second order, third order differential equations and then what we will have to do is split that equations into many first order equations and then use these schemes and that is what we use the same Runge Kutta or predictor corrector scheme that what is we would see in the next class.