- **Euler's Method** This is a numerical methods that will allow us to approximate solutions to differential equations. There are many different methods that can be used to approximate solutions to a differential equation and in fact whole classes can be taught just dealing with the various methods. We are going to look at one of the oldest and easiest to use here. This method was originally devised by Euler and is called, oddly enough, Eulers Method. Let us start with a general first order differential equation along with the initial condition,

$$\frac{dy}{dt} = f(t, y), \qquad y(t_0) = y_0$$

Where $f(y, t)$ is a known function and the values in the initial condition are also known numbers. If f and f_y are continuous functions then there is a unique solution to the ODE in some surrounding of $t = t_0$. So let us assume that everything is nice and continuous so that we know that a solution will in fact exist. And we want to approximate the solution near $t = t_0$. Well start with the two pieces of information that we do know about the solution. First, we know the value of the solution at $t = t_0$ from the initial condition. Second, we also know the value of the derivative at $t = t_0$. We can get this by plugging the initial condition into $f(t, y)$ into the differential equation itself. So, the derivative at this point is.

$$\left.\frac{dy}{dt}\right|_{t=t_0} = f(t_0, y_0)$$

Now, recall from your Calculus I class that these two pieces of information are enough for us to write down the equation of the tangent line to the solution at $t = t_0$. The tangent line is

$$y = y_0 + f(t_0, y_0)(t - t_0)$$

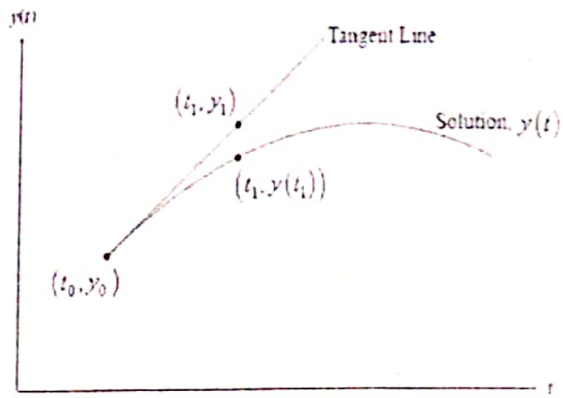


Figure 1: The tangent line

If t_1 is close enough to t_0 then the point y_1 on the tangent line should be fairly close to the actual value of the solution at t_1, or $y(t_1)$. Finding y_1 is easy enough. All we need to do is plug t_1 in the equation for the tangent line.

$$y_1 = y_0 + f(t_0, y_0)(t_1 - t_0)$$

Scanned with CamScanner

Now, we would like to proceed in a similar manner, but we dont have the value of the solution at t_1 and so we wont know the slope of the tangent line to the solution at this point. This is a problem. We can partially solve it however, by recalling that y_1 is an approximation to the solution at t_1. If y_1 is a very good approximation to the actual value of the solution then we can use that to estimate the slope of the tangent line at t_1.

So, lets hope that y_1 is a good approximation to the solution and construct a line through the point (t_1, y_1) that has slope $f(t_1, y_1)$. This gives

$$y = y_1 + f(t_1, y_1)(t - t_1)$$

Now, to get an approximation to the solution at $t = t_2$ we will hope that this new line will be fairly close to the actual solution at t_2 and use the value of the line at t_2 as an approximation to the actual solution. This gives.

$$y_2 = y_1 + f(t_1, y_1)(t_2 - t_1)$$

We can continue in this fashion. Use the previously computed approximation to get the next approximation. So,

$$y_3 = y_2 + f(t_2, y_2)(t_3 - t_2)$$
$$y_4 = y_3 + f(t_3, y_3)(t_4 - t_3)$$

and so on In general, if we have tn and the approximation to the solution at this point, y_n, and we want to find the approximation at t_{n+1} all we need to do is use the following.

$$y_{n+1} = y_n + f(t_n, y_n)(t_{n+1} - t_n)$$

If we define $f(t_n, y_n) = f_n$ and $(t_{n+1} - t_n) = h$, then the formula become,

$$y_{n+1} = y_n + hf_n$$

So, how do we use Eulers Method? Its fairly simple. We start with the first equation and then decide if we want to use a uniform step size or not. Then starting with (t_0, y_0) we repeatedly evaluate the last equation depending on whether we chose to use a uniform step size or not. We continue until weve gone the desired number of steps or reached the desired time. This will give us a sequence of numbers y_1 , y_2 , y_3 , y_n that will approximate the value of the actual solution at t_1 , t_2 , t_3 , t_n.

What do we do if we want a value of the solution at some other point than those used here? One possibility is to go back and redefine our set of points to a new set that will include the points we are after and redo Eulers Method using this new set of points. However this is cumbersome and could take a lot of time especially if we had to make changes to the set of points more than once.

In practice you would need to write a computer program to do these computations for you. In most cases the function f(t,y) would be too large and/or complicated to use by hand and in most serious uses of Eulers Method you would want to use hundreds of steps which would make doing this by hand prohibitive. So, here is a bit of pseudo-code that you can use to write a program for Eulers Method that uses a uniform step size, h.

- define $f(t, y)$

Scanned with CamScanner

- ⋆ input t_0 and y_0
- ⋆ input step size, h and the number os steps, n.
- ⋆ for i form 1 to n do
  - ◇ $m = f(t_0, y_0)$
  - ◇ $y_1 = y_0 + h * m$
  - ◇ $t_1 = t_0 + h$
  - ◇ Print t_1 and y_1
  - ◇ $t_0 = t_1$
  - ◇ $y_0 = y_1$
- ⋆ end

The pseudo-code for a non-uniform step size would be a little more complicated, but it would essentially be the same.

- ⋆ Solve the differential equation

$$\frac{dy}{dt} + 2y = 2 - e^{-4t}, \qquad y(0) = 1$$

```
#include<iostream>
#include<fstream>
#include<cmath>
using namespace std;
ofstream fout("euler1.dat");
double N=500, h=0.1;
double f(double t, double y){
return 2-exp(-4*t)-2*y; }
void euler1(double h, double t0, double y0){
double y=y0;
for (double t=t0;t<=5.0; t+=h){
fout<<t<<" "<<y<<" "<< (1 + 0.5 * exp(-4 * t) - 0.5 * exp(-2 * t)) <<endl;
y+=h*f(t,y); }
}
int main(){
euler1(0.1,0.0,1.0);
return 0; }
```

The obtained result is plotted with the exact result below;

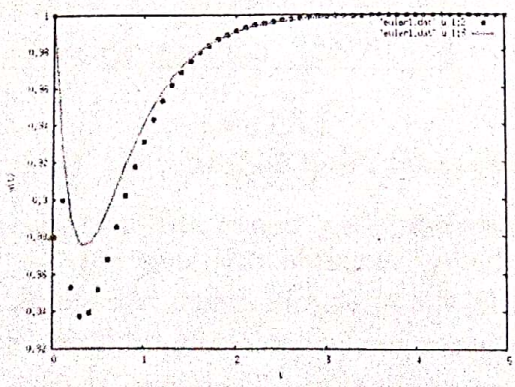


Figure 2: The green line is the exact result and red dot is the calculation using Euler's method.

Scanned with CamScanner

It is clear from the figure that the Euler's method does not work where the function is oscillatory. And it works better where the function is smooth. The exact solution of the above equation is

$$y(t) = 1 + 0.5e^{-4t} - 0.5e^{-2t}$$

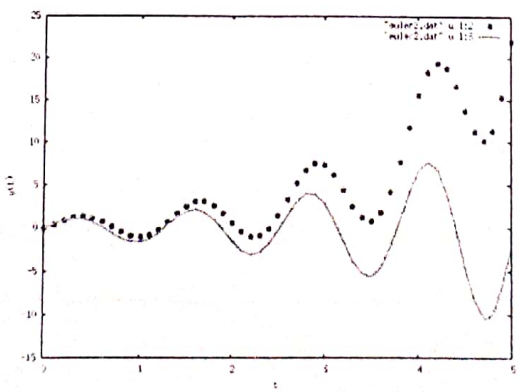
* Let us solve another example;

$$\frac{dy}{dt} - y = -0.5e^{t/2}sin(5t) + 5e^{t/2}cos(5t) \qquad y(0) = 0$$

The exact solution of the above equation is $y(t) = e^{t/2}sin(5t)$

```
#include<iostream>
#include<fstream>
#include<cmath>
using namespace std;
ofstream fout("euler2.dat");
double N=500, h=0.1;
double f(double t, double y){
return y-0.5*exp(0.5*t)*sin(5*t)+5*exp(0.5*t)*cos(5*t); }
void euler1(double h, double t0, double y0){
double y=y0;
for (double t=t0;t<=5.0; t+=h){
fout<<t<<" "<<y<<" "<<(exp(0.5*t)*sin(5*t))<<endl;
y+=h*f(t,y); }
}
int main(){
euler1(0.1,0.0,0.0);
return 0;
}
```

The obtained results are presented in the figure below.



ıre 3: The green line is the exact result and red dot is the calculation using Euler's method.

However, unlike the last example increasing t sees an increasing error. This behavior is fairly common in the approximations. We shouldnt expect the error to decrease as t increases as we saw in the last example. Each successive approximation is found using a previous approximation. Therefore, at each step we introduce error and so approximations should, in general, get worse as t increases.

Scanned with CamScanner

```
#include<iostream>
#include<fstream>
#include<cmath>
using namespace std;
ofstream fout("euler2.dat");
double N=500, h=0.001;

double f(double t, double y){
return y-0.5*exp(0.5*t)*sin(5*t)+5*exp(0.5*t)*cos(5*t);
}
void euler1(double h, double t0, double y0){
double y1=y0;
double y=0.0;
for (double t=t0;t<=5.0; t+=h){
fout<<t<<" "<<y1<<" "<<y<<" "<<(exp(0.5*t)*sin(5*t))<<endl;
y1+=h*f(t,y);
y+=0.5*h*(f(t,y)+f(t+h,y1)); }
}
int main(){
euler1(0.1,0.0,0.0);

return 0; }
```

- **2nd order ODE:** For second order ODE one need to decouple the equation into two first order ODE. For example

$$\frac{d^2x}{dt^2} = -kx$$

can be written as

$$v = \frac{dx}{dt} \qquad \text{and} \qquad \frac{dv}{dt} = -kx$$

And, one apply Euler's method on both v and x simultaneously as follows,

$$x_{n+1} = x_n + f(v, x, t) * h$$

$$v_{n+1} = v_n + f(x, t) * h$$

Simple Harmonic Pendulum

A very widely used problem in computational physics is to solve the 2nd order differential equation of a Simple Harmonic Pendulum. The 2nd order differential equation of a simple pendulum can be given as

$$\frac{d^2\theta}{dt^2} + \frac{g}{L}\theta = 0$$

, One can decouple the equation into two 1st order differential equation as follows:

$$\frac{d\theta}{dt} = \omega$$

$$\frac{d\omega}{dt} = -\frac{g}{L}\theta$$

Scanned with CamScanner

- One can rightly use Euler's method for this as follows:
  - ⋆ initialize $t = 0$, $\theta(0)$ and $\omega(0)$
  - ⋆ repeat until $t \leq t_{max}$
    - ◇ $\omega(t + dt) = \omega(t) - \frac{g}{L} * \theta(t) * dt$
    - ◇ $\theta(t + dt) = \theta(t) + \omega(t) * dt$
    - ◇ $t = t + dt$

  Euler algorithm is unstable- energy does not conserved in this approach (You Will See in a Few Seconds!!)

- Euler-Cromer method. A very simple modification of Euler's method is done by Cromer, and it work very nicely for 2nd order differential equation. The modification is to **Use the Updated velocity while calculating updated position.** The euler-cromer algorithm is as follows.
  - ⋆ initialize $t = 0$, $\theta(0)$ and $\omega(0)$
  - ⋆ repeat until $t \leq t_{max}$
    - ◇ $\omega(t + dt) = \omega(t) - \frac{g}{L} * \theta(t) * dt$
    - ◇ $\theta(t + dt) = \theta(t) + \omega(t + dt) * dt$
    - ◇ $t = t + dt$

  Euler-Cromer method conserve energy in each period of a simple harmonic motion.

Example:In the following example we will solve the equation of a simple harmonic motion by using both the Euler method and Euler-Cromer method. And will compare the results.

```
#include <cmath>
#include <cstdlib>
#include <fstream>
#include <iostream>
using namespace std;
const double g = 9.80;
const double L = 1;
double t;
double theta;
double omega;
void EulerStep( double t, double theta, double omega, double dt){
double omegaOld;
ofstream file("euler-file.dat");
for(int i=0;i<=100;i++){
omegaOld=omega;
omega -= (g/L) * theta * dt;
theta += omegaOld * dt;
t += dt;
file<< t <<" " << theta <<" " << omega<<endl; } }
void EulerCromerStep( double t, double theta, double omega,double dt){
ofstream file("cromer-file.dat");
for(int i=0;i <=100;i++){
omega -= (g/L) * theta * dt;
```

Scanned with CamScanner

```
theta += omega * dt;
t += dt;
file<< t <<" " << theta <<" " << omega<<endl;
} }
int main(){
t = 0;
theta = 0;
omega = 1;
double dt=0.1;
EulerCromerStep(t, theta, omega,dt);
EulerStep(t, theta, omega,dt);
return 0; }
```

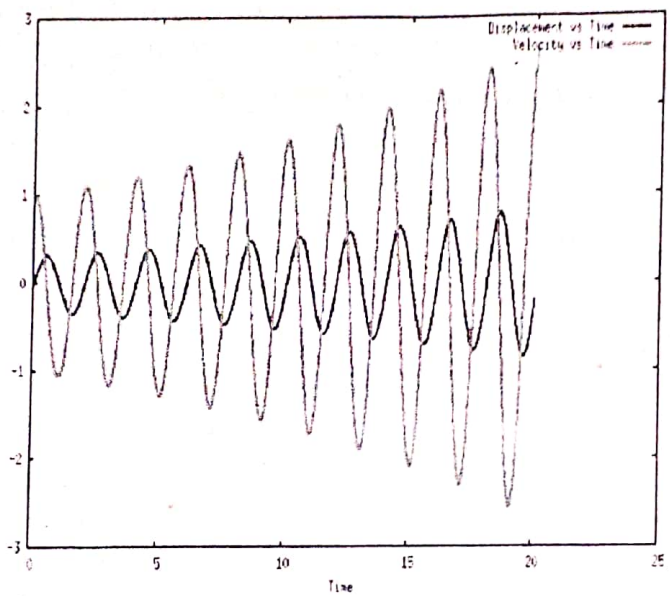


Figure 4: The green line is the Displacement vs Time graph and the Green line is the Velocity vs Time graph, Obtained from the Euler's method.

It is clear from the figure that the displacement amplitude and also the velocity amplitude is increasing with time, which means that the energy is not conserved in the system. Now let us see the same results obtained in the Euler-Cromer method. In the Euler-Cromer method (figure below) we see that both the displacement and the velocity amplitude remain constant over time. So, in the Euler-Cromer method the energy remain constant. **One can plot the phase diagram obtained in both the above to check if energy really conserve or not.**

Scanned with CamScanner

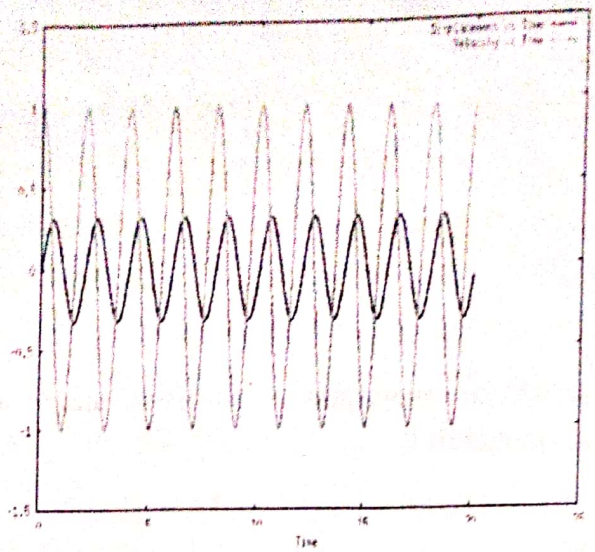


Figure 5: The green line is the Displacement vs Time graph and the Green line is the Velocity vs Time graph, Obtained from the Euler-Cromer method.

Runge-Kutta Method

Runge-Kutta (RK) methods are based on Taylor expansion formulae, but yield in general better algorithms for solutions of an ODE. The basic philosophy is that it provides an intermediate step in the computation of y_{i+1}
Consider the following differential equation

$$\frac{dy}{dt} = f(t, y)$$

and

$$y(t) = \int f(t, y)dt$$

and

$$y_{i+1} = y_i + \int_{t_i}^{t_{i+1}} f(t, y)dt$$

To demonstrate the philosophy behind RK methods, let us consider the second-order RK method, RK2. The first approximation consists in Taylor expanding $f(t, y)$ around the center of the integration interval t_i to t_{i+1}, i. e at $t_i + h/2$, h being the step size. Using the midpoint formula for an integral, defining $y(t_i + h/2) = y_{i+1/2}$ and $t_i + h/2 = t_{i+1/2}$, we obtain

$$\int_{t_i}^{t_{i+1}} f(t, y)dt \approx hf(t_{i+1/2}, y_{i+1/2}) + O(h^3)$$

Which means that we have

$$y_{i+1} = y_i + hf(t_{i+1/2}, y_{i+1/2}) + O(h^3)$$

However, we do not know the value of $y_{i+1/2}$. Here comes thus the next approximation, namely, we use Euler's method to approximate $y_{i+1/2}$. We then have

$$y_{(i+1/2)} = y_i + \frac{h}{2}\frac{dy}{dt} = y(t_i) + \frac{h}{2}f(t_i, y_i)$$

This means that we can define the following algorithm for the second-order Runge-Kutta method, RK2.

$$k_1 = hf(t_i, y_i)$$

Scanned with CamScanner

$$k_2 = hf(t_{(i+1/2)}, y_i + k_1/2)$$

with the final value

$$y_{i+1} \approx y_i + k_2 + O(h^3).$$

The difference between the previous one-step methods is that we now need an intermediate step in our evaluation, namely $t_{(i+h/2)} = t_{(i+1/2)}$ where we evaluate the derivative f. This involves more operations, but the gain is a better stability in the solution. The fourth-order Runge-Kutta, RK4, which we will employ in the solution of various differential equations below, is easily derived. The steps are as follows. We start again with the equation

$$y_{i+1} = y_i + \int_{t_i}^{t_{i+1}} f(t, y)dt$$

but instead of approximating the integral with the midpoint rule, we use now Simpsons rule at $t_i + h/2$, h being the step. Using Simpsons formula for an integral, defining $y(ti + h/2) = y_{(i+1/2)}$ and $t_i + h/2 = t_{(i+1/2)}$, we obtain

$$\int_{t_i}^{t_{i+1}} f(t, y)dt \approx \frac{h}{6}[f(t_i, y_i) + 4f(t_{i+1/2}, y_{i+1/2}) + f(t_{i+1}, y_{i+1})] + O(h^5)$$

This means that, we have

$$y_{i+1} = y_i + \frac{h}{6}[f(t_i, y_i) + 4f(t_{i+1/2}, y_{i+1/2}) + f(t_{i+1}, y_{i+1})] + O(h^5)$$

However, we do not know the values of $y_{i+1/2}$ and y_{i+1}. The fourth-order Runge-Kutta method splits the midpoint evaluations in two steps, that is we have

$$y_{i+1} = y_i + \frac{h}{6}[f(t_i, y_i) + 2f(t_{i+1/2}, y_{i+1/2}) + f(t_{i+1}, y_{i+1}) + 2f(t_{i+1/2}, y_{i+1/2}) + f(t_{i+1}, y_{i+1})]$$

since we want to approximate the slope at $y_{i+1/2}$ in two steps. The first two function evaluations are as for the second order Runge-Kutta method. The algorithm is as follows

- ⋆ We compute first

  $$k_1 = hf(t_i, y_i)$$

  which is nothing but the slope at t_i. If we stop here we have Euler's method.

- ⋆ Then we compute the slope at the midpoint using Eulers method to predict $y_{i+1/2}$, as in the secondorder Runge-Kutta method. This leads to the computation of

  $$k_2 = hf(t_i + h/2, y_i + k_1/2).$$

- ⋆ The improved slope at the midpoint is used to further improve the slope of $y_{i+1/2}$ by computing

  $$k_3 = hf(t_i + h/2, y_i + k_2/2).$$

- ⋆ With the latter slope we can in turn predict the value of y_{i+1} via the computation of

  $$k_4 = hf(t_i + h, y_i + k_3).$$

Scanned with CamScanner

* The final algorithm becomes then

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Thus, the algorithm consists in first calculating k_1 with t_i, y_1 and f as inputs. Thereafter, we increasethe step size by $h/2$ and calculate k_2, then k_3 and finally k_4. With this caveat, we can then obtain thenew value for the variable y. It results in four function evaluations, but the accuracy is increased bytwo orders compared with the second-order Runge-Kutta method. The fourth order Runge-Kutta methodhas a global truncation error which goes like $O(h^4)$. Fig. gives a geometrical interpretation of the fourth-order Runge-Kutta method.

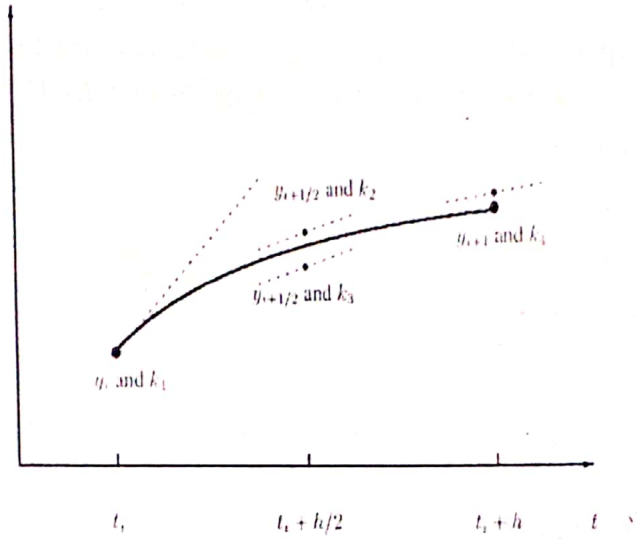


Figure 6: Geometrical interpretation of the fourth-order Runge-Kutta method. The derivative is evaluated at four points, once at the intial point, twice at the trial midpoint and once at the trial endpoint. These four derivatives constitute one Runge-Kutta step resulting in the final value.

Example1:
Solve the differential equation

$$\frac{dx}{dt} = t\sqrt{x}$$

with the initial condition $t_0 = 0$, and $x_0 = 1$. Given the exact solution is

$$x = \frac{(t^2 + 4)^2}{16}$$

Solution:

```
#include<iostream>
#include<fstream>
#include<cmath>

using namespace std;

double f(double indep, double dep){
return indep*sqrt(dep);
```

Scanned with CamScanner

say we wish to compute y_{n+1}

$$K_1 = hf(y_n, t_n)$$

$$K_2 = hf(y_n + \beta K_1, t_n + \alpha h) \quad \text{—} (*)$$

$$y_{n+1} = y_n + aK_1 + bK_2$$

Where α, β, a, b are const to be determined, If $K_2 = 0$, $a = 1$ that will be Euler method.

Now T. S expansion,

$$y(t_{n+1}) = y_n + h\frac{dy}{dt}\Big|_{t_n} + \frac{h^2}{2}\frac{d^2y}{dt^2}\Big|_{t_n} + O(h^3)$$

And for IVP $\frac{dy}{dt} = f(y, t)$ so

$$\frac{d^2y}{dt^2} = \frac{df(y,t)}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}\cdot\frac{\partial y}{\partial t}$$

$$= \frac{\partial f}{\partial t} + f\frac{\partial y}{\partial t}$$

So,

$$y_{n+1} = y_n + hf(y_n, t_n) + \frac{h^2}{2}\left(\frac{\partial f(y_n,t_n)}{\partial t} + f\frac{\partial y}{\partial t}\right) \quad \text{—} (1)$$

However K_2 is RK method can be expanded as -

$$K_2 = hf(y_n + \beta K_1, t_n + \alpha h)$$

$$= h\left(f(y_n, t_n) + \alpha h\frac{\partial f}{\partial t}(y_n, t_n) + \beta K_1\frac{\partial f}{\partial y}(y_n, t_n)\right) + O(h^3)$$

Scanned with CamScanner

So by substitution to *

$$y_{n+1} = y_n + (a+b)hf(y_n, t_n) + bh^2\left(\alpha \frac{\partial f}{\partial t} + \beta f \frac{\partial f}{\partial y}\right)(y_n, t_n) + O(h^3) \quad (2)$$

from (II) and (I)

$$a+b=1 \qquad ab = \tfrac{1}{2} \qquad \beta b = \tfrac{1}{2}$$

Give $\alpha = \beta = 1$, $a = b = \tfrac{1}{2}$

So RK2 method

$$K_1 = hf(y_n, t_n)$$

$$K_2 = hf(y_n + K_1, t_n + h)$$

$$y_{n+1} = y_n + (K_1 + K_2)/2$$ similar

RK4

Scanned with CamScanner