

# Numerical Methods in Differential Equations

Fernando Deeke Sasse

2008

A Maple Companion to  
Mark H. Holmes, Introduction to Numerical Methods in  
Differential Equations, Springer, 2007

## Introduction

### Initial Value Problems

- Even from casual observation it is apparent that most physical phenomena vary both in space and time.
- A consequence of this is that mathematical models of the real world almost inevitably involve both time and space derivatives.

In this course we will learn how solve these problems computationally. We will study:

1. Problems involving only time derivatives,
2. Spatial problems.
3. Space-time problems.

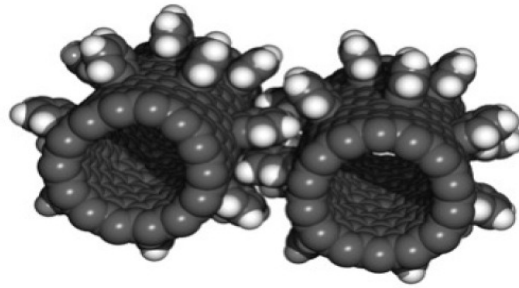
General Initial value problem (IVP):

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}), \quad \text{for } 0 < t, \quad \mathbf{y}(0) = \mathbf{a}$$

It is assumed that the IVP is **well posed** (i.e., there is a unique solution that is a smooth function of time). By **smooth** it is meant that  $\mathbf{y}(t)$  and its various derivatives are defined and continuous.

Fact: Most real-world problems are so complicated that there is no hope of finding an analytical solution.

**Example:** To study molecular machinery such as nanogears it is necessary to solve a system involving thousands of equations with a very complicated nonlinear function  $\mathbf{f}$ .



The nanogears are composed of carbon (the grey spheres) and hydrogen (the white spheres) atoms. The rotation of the tubes, and the resulting meshing of the gear teeth, was carried out by solving a large system of IVPs.

**Steady State Solution:**  $y = Y$  is an equilibrium, or steady-state, solution if it is constant and satisfies the differential equation.

**Stable Solution:** a steady-state  $Y$  is stable if any solution that starts near  $Y$  stays near it.

**Asymptotically Stable Solution:** If, in addition, initial conditions starting near  $Y$  actually result in the solution converging to  $Y$  as  $t \rightarrow \infty$ , then  $y = Y$  is said to be asymptotically stable.

## Examples of IVPs

### 1. Radioactive decay

According to the law of radioactive decay, the mass of a radioactive substance decays at a rate that is proportional to the amount present. To express this in mathematical terms, let  $y(t)$  designate the amount present at time  $t$ . In this case the decay law can be expressed as

$$\frac{dy}{dt} = -ry, \quad \text{for } 0 < t.$$

If we start out with an amount  $\alpha$  of the substance then the corresponding initial condition is

$$y(0) = \alpha.$$

In this decay law,  $r$  is the proportionality constant and it is assumed to be positive. Because the largest derivative in the problem is first order, this is an example of a **first-order** IVP for  $y(t)$ . It is also **linear, homogeneous**, and has **constant** coefficients. Let us find the solution:

```

> restart
> eq:=diff(y(t),t)+r*y(t)=0;
      eq :=  $\frac{d}{dt} y(t) + r y(t) = 0$  (1.1)
> dsolve({eq, y(0)=alpha});
      y(t) =  $\alpha e^{-rt}$  (1.2)
  
```

Consequently, the solution starts at  $\alpha$  and decays exponentially to zero as time increases. Note that  $y = 0$  is an **asymptotically stable equilibrium solution** for (1.1).

### 2. Logistic equation

In the study of populations limited by competition for food one obtains the logistic equation, which is

$$\frac{dy}{dt} = \lambda y(1 - y), \quad \text{for } 0 < t,$$

where

$$y(0) = \alpha.$$

It is assumed that  $\lambda$  and  $\alpha$  are positive. As with radioactive decay, this IVP involves a **first-order** equation for  $y(t)$ . However, because of the  $y^2$  term, this equation is **nonlinear**. Let Maple solve this IVP:

```
> eq:=diff(y(t),t)=lambda*y(t)*(1-y(t));
```

$$eq := \frac{d}{dt} y(t) = \lambda y(t) (1 - y(t)) \quad (1.3)$$

```
> dsolve({eq, y(0)=alpha});
```

$$y(t) = -\frac{\alpha}{-\alpha - e^{-\lambda t} + e^{-\lambda t} \alpha} \quad (1.4)$$

Now, the equilibrium solutions for this equation are  $y = 1$  and  $y = 0$ . Because  $\lambda > 0$ , the solution approaches  $y = 1$  as  $t$  increases. Consequently,  $y = 1$  is an asymptotically stable equilibrium solution, whereas  $y = 0$  is not.

### 3. Newton's Second Law

The reason for the prominence of differential equations in science and engineering is that they are the foundation for the laws of nature. The most well known of these laws is Newton's second, which states that  $F = ma$ . Letting  $y(t)$  designate position then this law takes the form

$$m \frac{d^2 y}{dt^2} = F(t, y, y'), \quad \text{for } 0 < t.$$

The above equation allows for the possibility that the force  $F$  varies in time as well as depends on position and velocity. Assuming that the initial position and velocity are specified, then the initial conditions for this problem take the form

$$y(0) = \alpha \quad \text{and} \quad y'(0) = \beta.$$

This IVP is second order and it is nonlinear if the force depends nonlinearly on either  $y$  or  $y'$ . It is possible to write the problem as a first-order system by introducing the variables

$$\begin{aligned} y_1 &= y, \\ y_2 &= y'. \end{aligned}$$

Differentiating each of these equations, and using the original differential equation, we obtain the following

$$\begin{aligned} y_1' &= y_2, \\ y_2' &= \frac{1}{m} F(t, y_1, y_2). \end{aligned}$$

By introducing the vector  $y(t)$ , defined as

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix},$$

the IVP can be written as

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}), \quad \text{for } 0 < t,$$

where the initial conditions take the form

$$\mathbf{y}(0) = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

and

$$\mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} y_2 \\ \frac{1}{m}F(t, y_1, y_2) \end{pmatrix}.$$

What is significant is that the change of variables has transformed the second order problem for  $y(t)$  into a first-order IVP for  $\mathbf{y}(t)$ . Like the original second order equation, this one is nonlinear if  $F$  depends nonlinearly on either  $y_1$  or  $y_2$ .

## ▼ Methods Obtained from Numerical Differentiation

The task we now undertake is to approximate the differential equation, and its accompanying initial condition, with a problem we can solve using a computer. The question is, can we accurately compute the solution directly from the problem without first finding an analytical solution? As it turns out, most realistic mathematical models of physical and biological systems cannot be solved by hand, so having the ability to find accurate numerical solutions directly from the original equations is an invaluable tool.

We consider the problem of solving the IVP

$$\frac{dy}{dt} = f(t, y), \quad \text{for } 0 < t,$$

where

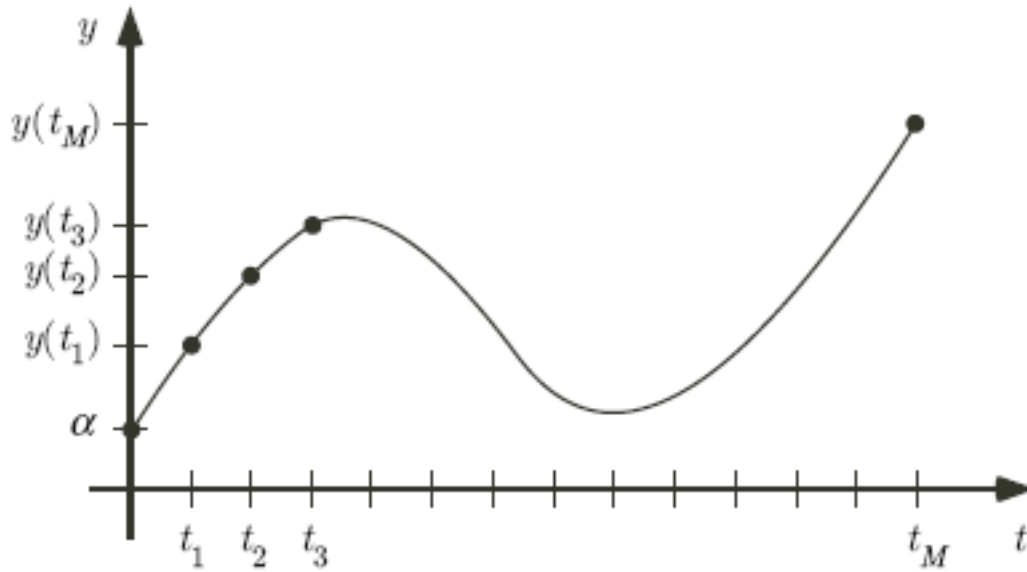
$$y(0) = \alpha.$$

Our objective is to replace these with discrete variables so that the resulting problem is algebraic and therefore solvable using standard numerical methods. The approach we take proceeds in a sequence of five steps.

It is assumed in what follows that the interval is  $0 \leq t \leq T$ .

### Step 1

We first introduce the time points at which we will compute the solution. These points are labeled sequentially as  $t_0, t_1, t_2, \dots, t_M$



We confine our attention to a uniform grid with step size  $k$ , so, the formula for the time points is

$$t_j = jk, \quad \text{for } j = 0, 1, 2, \dots, M.$$

Because the time interval is  $0 \leq t \leq T$  we require  $t_M = T$ . Therefore,  $k$  and  $M$  are connected through the equation

$$k = \frac{T}{M}.$$

### Step 2

Evaluate the differential equation at the time point  $t = t_j$  to obtain

$$y'(t_j) = f(t_j, y(t_j)). \quad (1.22)$$

### Step 3

Replace the derivative term in Step 2 with a finite difference formula using the values of  $y$  at one or more of the grid points in a neighborhood of  $t_j$ . This is where things get a bit interesting, because numerous choices can be made, a few of which are listed in Table 1.1. Different choices result in different numerical procedures, and as it turns out, not all choices will work.

| Type      | Difference Formula   | Truncation Term                          |
|-----------|--|--|
| Forward   | $f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + \tau_i$                   | $\tau_i = -\frac{h}{2} f''(\eta_i)$      |
| Backward  | $f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h} + \tau_i$                   | $\tau_i = \frac{h}{2} f''(\eta_i)$       |
| Centered  | $f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1}))}{2h} + \tau_i$             | $\tau_i = -\frac{h^2}{6} f'''(\eta_i)$   |
| One-sided | $f'(x_i) = \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h} + \tau_i$  | $\tau_i = \frac{h^2}{3} f'''(\eta_i)$    |
| One-sided | $f'(x_i) = \frac{3f(x_i) - 4f(x_{i-1}) + f(x_{i-2}))}{2h} + \tau_i$  | $\tau_i = \frac{h^2}{3} f'''(\eta_i)$    |
| Centered  | $f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1}))}{h^2} + \tau_i$ | $\tau_i = -\frac{h^2}{12} f''''(\eta_i)$ |

**Table 1.1.** Numerical differentiation formulas. The points  $x_1, x_2, x_3, \dots$  are equally spaced with step size  $h = x_{i+1} - x_i$ . The point  $\eta_i$  is located between the left- and rightmost points used in the formula.

### Derivations of difference formulas

Let  $h$  be the stepsize. Then, using Taylor formula,

$$y(x+h) = y(x) + hy'(x) + \frac{h^2}{2}y''(x) + \dots + \frac{h^n}{n!}y^{(n)}(x) + \frac{h^{(n+1)}}{(n+1)!}y^{(n+1)}(\xi), \quad x < \xi < x+h.$$

The last term represents the error in the approximation  $y(x+h)$  caused by the cutoff of the first  $n+1$  terms of the Taylor series.

If  $n=1$  we have

$$y(x+h) = y(x) + hy'(x) + \frac{h^2}{2}y''(\xi),$$

Thus

$$y'(x) = \frac{y(x+h) - y(x)}{h} - \frac{h}{2}y''(\xi).$$

### Centered difference formula

We take  $n=2$  to get

```
> restart;
> eq1:=y(x+h)=convert(taylor(y(x+h),h,3),polynom)+h^3*O3;
```

(2.1.1)

$$eq1 := y(x+h) = y(x) + D(y)(x)h + \frac{1}{2} D^{(2)}(y)(x)h^2 + h^3 O3 \quad (2.1.1)$$

> eq2:=y(x-h)=convert(taylor(y(x-h),h,3),polynom)+h^3\*R3;

$$eq2 := y(x-h) = y(x) - D(y)(x)h + \frac{1}{2} D^{(2)}(y)(x)h^2 + h^3 R3 \quad (2.1.2)$$

> eq1-eq2;

$$y(x+h) - y(x-h) = 2 D(y)(x)h + h^3 O3 - h^3 R3 \quad (2.1.3)$$

> solve(%,(D(y))(x));

$$\frac{1}{2} \frac{y(x+h) - y(x-h) - h^3 O3 + h^3 R3}{h} \quad (2.1.4)$$

In other words,

$$y'(x) = -\frac{y(x+h) - y(x-h)}{2h} - \frac{h^2}{6} y'''(\xi)$$

**Exercise:** Obtain all other formulas given in Table 1.1

To start we take the first entry listed in Table 1.1, which means we use the advanced formula for the first derivative:

$$y'(t_j) = \frac{y(t_{j+1}) - y(t_j)}{k} + \tau_j,$$

where

$$\tau_j = -\frac{k}{2} y''(\eta_j)$$

and  $\eta_j$  is a point between  $t_j$  and  $t_{j+1}$ . Introducing this into the differential equation we obtain

$$\frac{y(t_{j+1}) - y(t_j)}{k} + \tau_j = f(t_j, y(t_j)),$$

or, equivalently,

$$y(t_{j+1}) - y(t_j) + k\tau_j = kf(t_j, y(t_j)).$$

From the initial condition we have that the starting value is

$$y_0 = \alpha.$$

We note that:

- This difference formula uses a value of  $t$  ahead of the current position. For this reason it is referred to as a **forward difference formula** for the first derivative.
- The term  $\tau_j$ , as it appears in the approximation represents how well we have approximated the

original problem. For this reason it is the truncation error for the method and is of order  $O(k)$ .

- It is essential that whatever approximations we use, the truncation error goes to zero as  $k$  goes to zero. This means that, at least in theory, we can approximate the original problem as accurately as we wish by making the time step  $k$  small enough. It is said in this case that the approximation is consistent. Unfortunately, as we demonstrate shortly, consistency is not enough to guarantee an accurate numerical solution.

**Step 4.** Drop the truncation error. This is the step where we go from an exact problem to one that is, hopefully, an accurate approximation of the original. After dropping  $\tau_j$ , the resulting equation is

$$y_{j+1} - y_j = kf(t_j, y_j),$$

or

$$y_{j+1} = y_j + kf(t_j, y_j), \quad \text{for } j = 0, 1, 2, \dots, M - 1. \quad (1.28)$$

with

$$y_0 = \alpha.$$

This finite difference equation is known as the **Euler method** for solving the differential equation. It is a recursive algorithm in which one starts with  $j = 0$  and then uses to determine the solution at  $j = 1$ , then  $j = 2$ , then  $j = 3$ , etc. Because (1.28) gives the unknown  $y_{j+1}$  explicitly in terms of known quantities, it is an **explicit method**.

### Example

Let's see how well Euler's method does with the logistic equation (1.5). Specifically, suppose the IVP is

$$\frac{dy}{dt} = 10y(1 - y), \quad \text{for } 0 < t, \quad (1.30)$$

where

$$y(0) = 0.01. \quad (1.31)$$

We will use the Euler method to calculate the solution for  $0 \leq t \leq 1$ . In this case,  $k$  and  $M$  are connected through the equation

$$k = \frac{1}{M}. \quad (1.32)$$

The finite difference equation in (1.28) now takes the form

$$y_{j+1} = y_j + 10ky_j(1 - y_j), \quad \text{for } j = 0, 1, 2, \dots, M - 1. \quad (1.33)$$

Taking  $M = 6$ , so  $k = 1/6$ , we have

```
> restart:
> eq:=diff(y(t),t)=10*y(t)*(1-y(t));
      eq :=  $\frac{d}{dt} y(t) = 10 y(t) (1 - y(t))$  (2.1)
```

```
> exact_sol:=dsolve({eq, y(0)=0.01}, y(t));
      exact_sol :=  $y(t) = \frac{1}{1 + 99 e^{-10t}}$  (2.2)
```

```
> Yexact:=unapply(op(2,exact_sol),t);
```

$$Y_{exact} := t \rightarrow \frac{1}{1 + 99 e^{-10t}}$$

(2.3)

```
> euler1:=proc(M)
```

```
  local k,j,L,y;
```

```
  k:=evalf(1/M): y[0]:=0.01:
```

```
  L:=[[0,y[0]]]:
```

```
  for j from 0 to M-1 do
```

```
    y[j+1]:=evalf(y[j]+10*k*y[j]*(1-y[j]));
```

```
    L:=[op(L),[(j+1)*k,y[j+1]]];
```

```
  od:
```

```
  end:
```

```
> euler1(6);
```

```
[[0, 0.01], [0.1666666667, 0.02650000000], [0.3333333334, 0.06949625001],  
 [0.5000000001, 0.1772737855], [0.6666666668, 0.4203534363], [0.8333333335,  
 0.8264474778], [1.000000000, 1.065500885]]
```

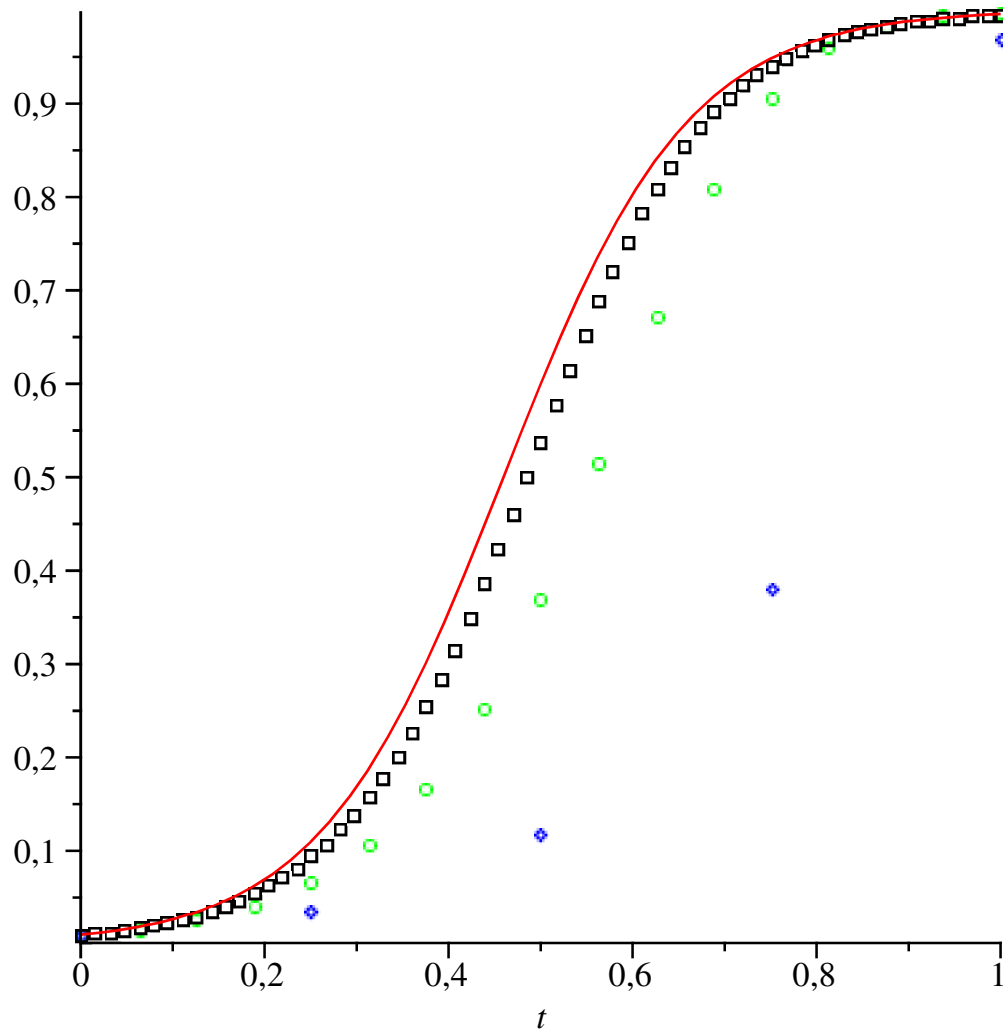
(2.4)

```
> with(plots):
```

```
> Lplots:=[pointplot(euler1(4),color=blue),pointplot(euler1  
  (16),color=green,symbol=circle),pointplot(euler1(64),color=  
  black,symbol=box)]:
```

```
> g2:=plot(Yexact(t),t=0..1,color=red):
```

```
> display([op(Lplots),g2]);
```



## Error

It is seen that the numerical solution with  $M = 4$  is not so good, but the situation improves considerably as more time points are used. In fact, it would appear that if we keep increasing the number of time points that the numerical solution converges to the exact solution. Does this actually happen? Answering this question brings us to the very important concept of error. The following Maple procedure calculates the error at each step of the Euler algorithm.

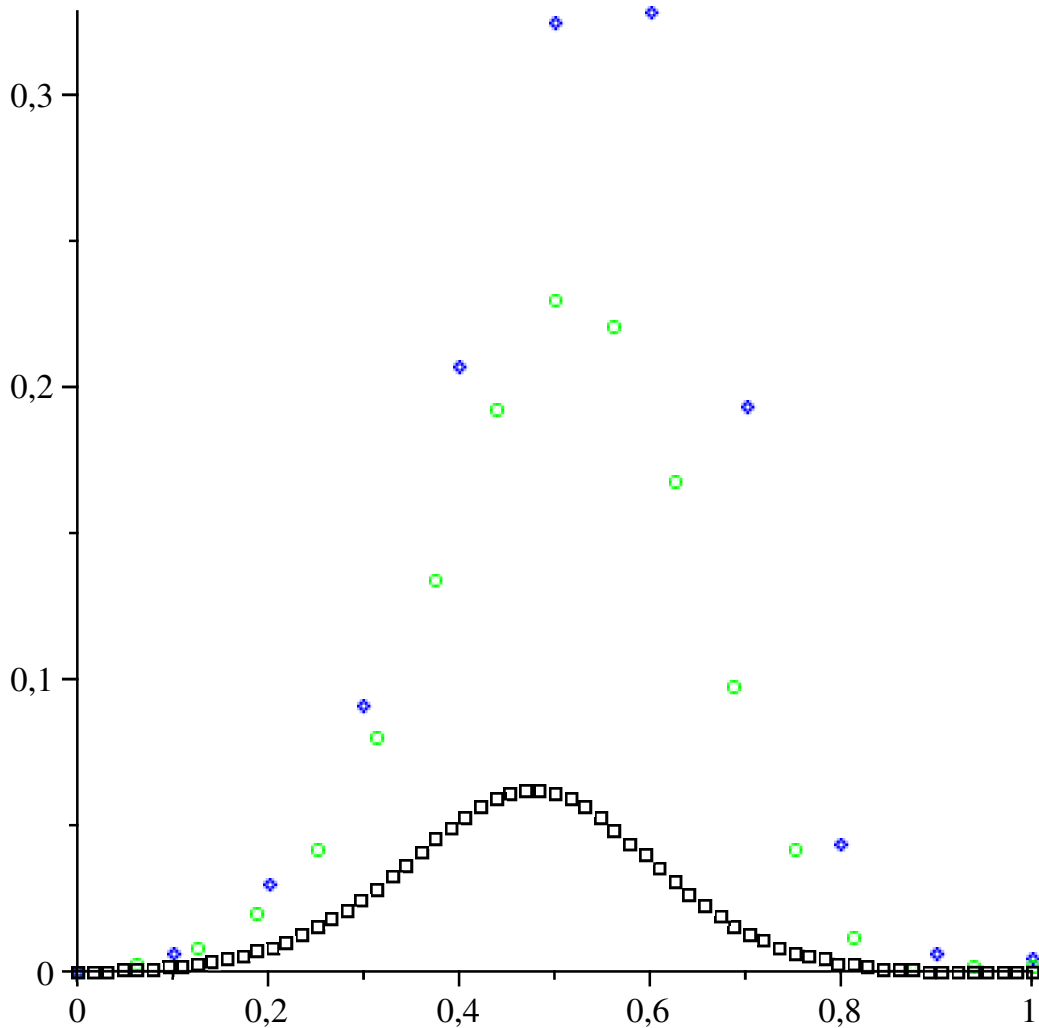
```
> error_euler1:=proc(M)
  local k, j, L, y, L_error, error1;
  k:=evalf(1/M): y[0]:=0.01:
  L_error:=[[0, 0]]:
  for j from 0 to M-1 do
    y[j+1]:=evalf(y[j]+10*k*y[j]*(1-y[j]));
    error1[j+1]:=abs(Yexact((j+1)*k)-y[j+1]);
    L_error:=[op(L_error), [(j+1)*k, error1[j+1]]];
  od:
end:
```

```

> error_euler1(6);
[[0, 0], [0.1666666667, 0.02426481076], [0.3333333334, 0.1511702567],
 [0.5000000001, 0.4225858166], [0.6666666668, 0.4677533297], [0.8333333335,
 0.1503090591], [1.000000000, 0.0699753670]]
> Lplots_error:= [pointplot(error_euler1(10), color=blue),
 pointplot(error_euler1(16), color=green, symbol=circle),
 pointplot(error_euler1(64), color=black, symbol=box)]:
> display(Lplots_error);

```

(2.5)



In what follows we need three different solutions:

$y(t_j) \equiv$  exact solution of the IVP at  $t = t_j$ ;

$y_j \equiv$  exact solution of finite difference equation at  $t = t_j$ ;

$\bar{y}_j \equiv$  solution of difference equation at  $t = t_j$  calculated  
by the computer.

We are interested in the difference between the exact solution of the IVP and the values we actually end up computing using our algorithm. Therefore, we are interested in the error

$$e_j = |y(t_j) - \bar{y}_j|.$$

The question we are going to ask is, if we increase  $M$  will

$$e_M = |y(T) - \bar{y}_M|$$

converge to zero or at least decrease down to the level of the round-off? We want the answer to this question to be yes and, moreover, that it is true no matter what choice we make for  $t = T$ . If this holds then the method is **convergent**.

Let us rewrite the error as follows:

$$e_M = |y(T) - y_M + y_M - \bar{y}_M|.$$

We consider two sources of error:

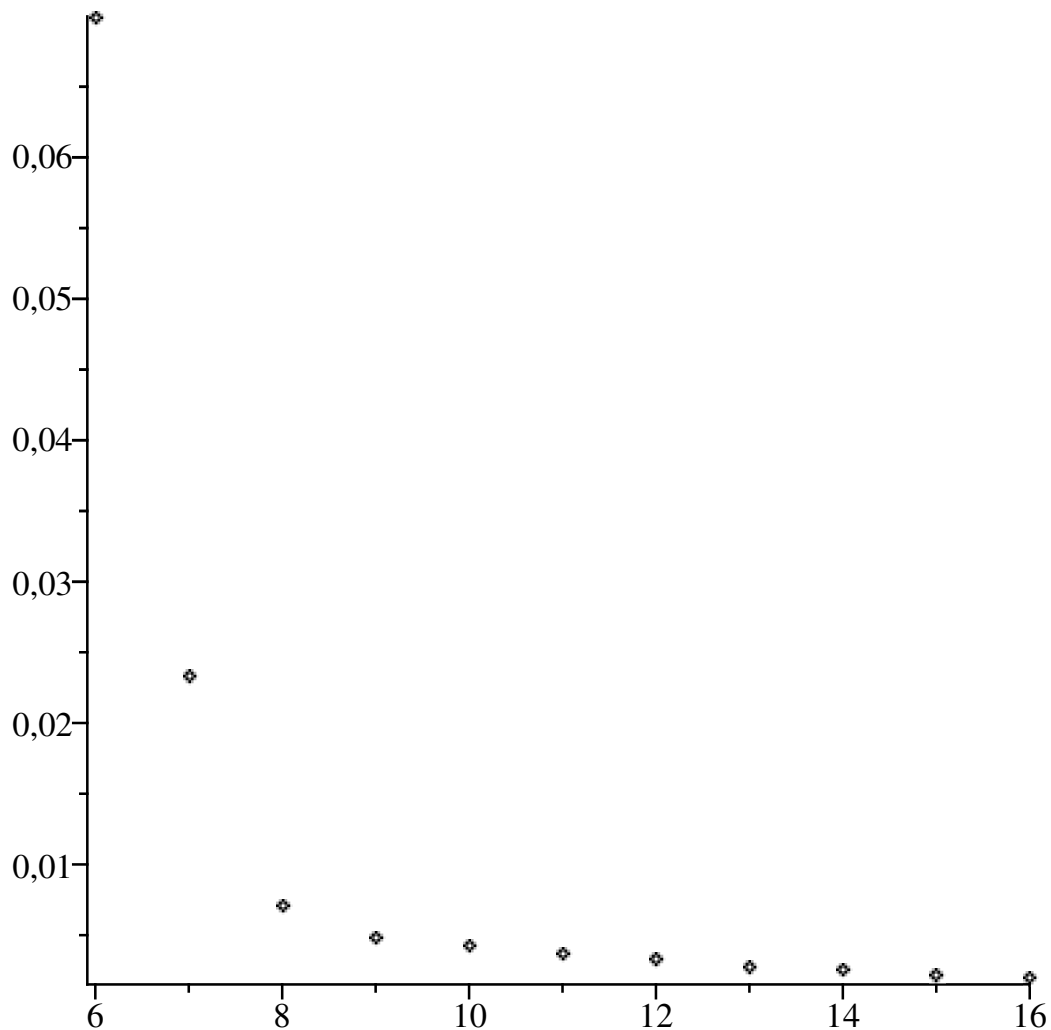
$$y(T) - y_M:$$

This is the difference, at  $t = T$ , between the exact solution of the IVP and the exact solution of the problem we use as its approximation. As illustrated below, this should be the major contributor to the error until  $k$  is small enough that this difference gets down to approximately that of the round-off.

```

> error_euler2:=proc(M)
  local k, j, L, ye;
  k:=1/M: ye[0]:=1/100;
  for j from 0 to M-1 do
    ye[j+1]:=ye[j]+10*k*ye[j]*(1-ye[j]);
  od:
  evalf(abs(ye[M]-Yexact(M*k)));
end:
> error_euler2(7);
                                0.0233879230
(2.6)
> pointplot([seq([M, error_euler2(M)], M=6..16)]);

```



$y_M - \bar{y}_M$ :

This is the error at  $t = T$  that originates from round-off when one uses floating-point calculations to compute the solution of the difference equation. The last column of Table 1.2 gives the values of this error at the first few time points. Getting values of  $10E-18$  or smaller, as occur in this calculation, is about as good as can be expected using double precision. This fact is illustrated in the implementation below:

```

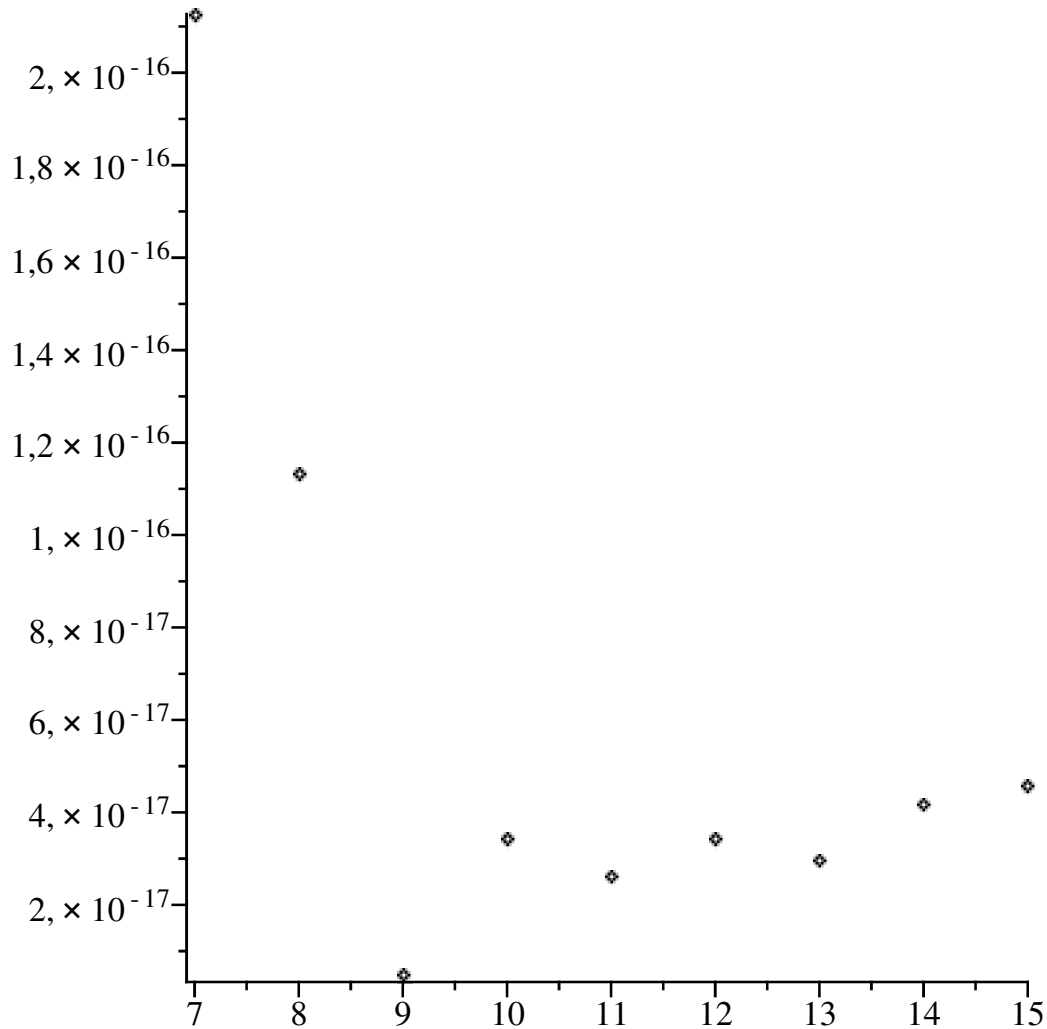
> error_euler3:=proc(M)
  local k, j, L, ye, y;
  k:=1/M: y[0]:=0.01; ye[0]:=1/100;
  for j from 0 to M-1 do
    y[j+1]:=evalf[16](y[j]+10*k*y[j]*(1-y[j]));
    ye[j+1]:=ye[j]+10*k*ye[j]*(1-ye[j]);
  od:
  evalf[22](abs(ye[M]-y[M]));
end:
> error_euler3(6);

```

$3.7321 \cdot 10^{-17}$

(2.7)

```
> pointplot([seq([M,error_euler3(M)],M=7..15)]);
```

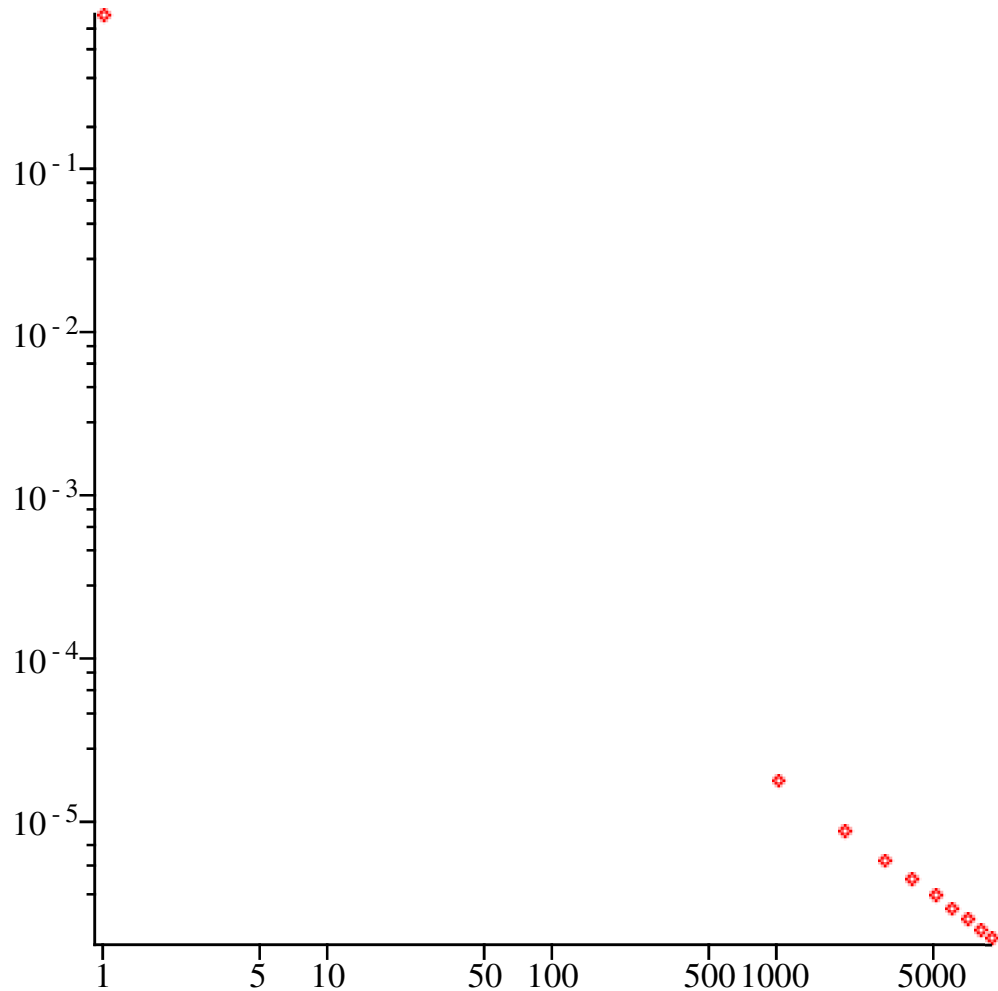


For the logistic equation example considered earlier, the error  $e_M = |y(T) - y_M|$  from the Euler method is plotted in below as a function of the number of time points used to reach  $T = 1$ . Since exact arithmetic calculation is impossible when  $n$  is relatively big, we use 30 digits precision to simulate it.

```
> error_euler4 := proc (M)
  local k, j, L, ye;
  k := 1/M;
  ye[0] := 1/100;
  for j from 0 to M-1 do
    ye[j+1] := evalf[30](ye[j]+10*k*ye[j]*(1-ye[j]));
  od;
  evalf[30](abs(ye[M]-Yexact(M*k)));
end proc;
```

```
> P1:=loglogplot([seq([k,error_euler4(k)], k = 1 .. 10000,1000)
],style=point):
```

```
> display(P1);
```



It is seen that the error decreases linearly in the log-log plot in such a way that increasing  $M$  by a factor of 10 decreases the error by the same factor. In other words, the error decreases as  $k^n$ , ( $k=T/M$ ) with  $n = 1$ . It is not a coincidence that this is the same order as for the truncation error (1.24):

$$y'(t_j) = \frac{y(t_{j+1}) - y(t_j)}{k} + \tau_j, \quad (1.23)$$

where

$$\tau_j = -\frac{k}{2}y''(\eta_j) \quad (1.24)$$

and  $\eta_j$  is a point between  $t_j$  and  $t_{j+1}$ . Introducing this into (1.22) we obtain

$$\frac{y(t_{j+1}) - y(t_j)}{k} + \tau_j = f(t_j, y(t_j)), \quad (1.25)$$

or equivalently,

$$y(t_{j+1}) - y(t_j) + k\tau_j = kf(t_j, y(t_j)). \quad (1.26)$$

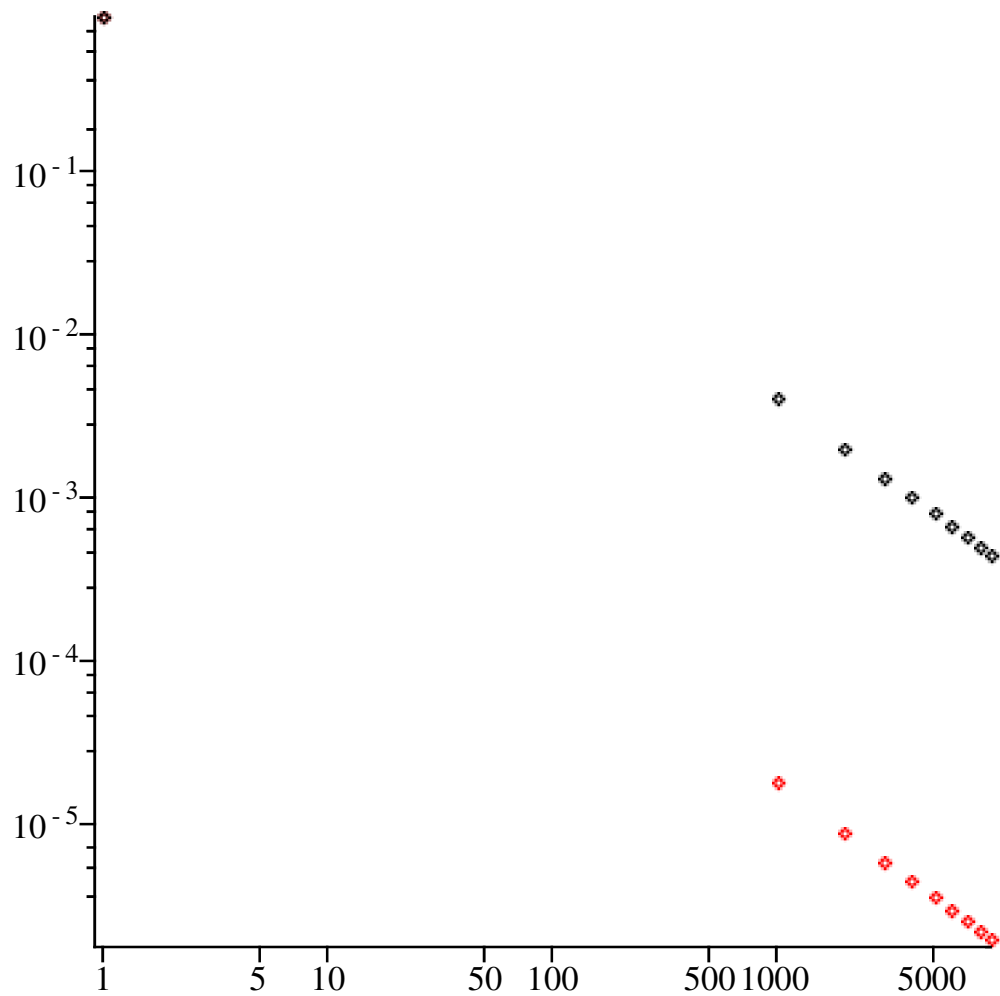
At first glance, because the term that is neglected in (1.26) is  $k\tau_j = O(k^2)$ , one might expect that the error would decrease as  $k^2$ . However,  $k\tau_j$  is the error we generate at each time step. To get to T we take  $M = 1/k$  time steps so the accumulated error we generate in getting to T is reduced by a factor of  $k$ . Therefore, with a convergent method the order of the truncation error determines the order of the error.

We are using the error at  $t = T$  to help determine how the approximation improves as the number of time steps increases. In many applications, however, one is interested in how well the numerical solution approximates the solution throughout the entire interval  $0 \leq t \leq T$ . For this it is more appropriate to consider using a vector norm to define the error. For example, using the maximum norm the error function takes the form

$$e_\infty = \max_{j=0,1,\dots,M} |y(t_j) - \bar{y}_j|. \quad (1.38)$$

To indicate how this differs from the error  $e_M$  used earlier, (1.38) is plotted below, together with  $e_M$ , for the logistic equation example. As expected,  $e_\infty$  is larger than  $e_M$  but its dependence on  $M$  is still  $O(k)$ .

```
> error_euler5 := proc (M)
  local k, j, L, y, error1, error2;
  k := evalf[18](1/M); y[0] := 1/100.;
  error1:=0.;
  for j from 0 to M-1 do
    y[j+1] := evalf[18](y[j]+10*k*y[j]*(1-y[j]));
    error2:=evalf[18](abs(y[j+1]-Yexact((j+1)*k)));
    if error2 > error1 then error1:=error2 fi;
  od;
  error1;
end proc;
> error_euler5(1000);
0.004032652620425356 (2.8)
> P2:=loglogplot([seq([k, error_euler5(k)], k = 1 .. 10000,
1000)], style = point,color=black);
> display([P1, P2]);
```



[ > ;

The earlier logistic equation is typical of what occurs in most applications. Namely, using the laws of physics or some other principles one obtains one or more differential equations to solve, and the numerical method is constructed directly from them. It is informative to see whether the steps can be reversed. Specifically, suppose we start with (1.28) and ask whether it is based on a consistent approximation of (1.18):

$$\frac{dy}{dt} = f(t, y), \quad \text{for } 0 < t, \quad (1.18)$$

where

$$y(0) = \alpha. \quad (1.19)$$

This is determined by plugging the exact solution into (1.28)

$$y_{j+1} = y_j + kf(t_j, y_j), \quad \text{for } j = 0, 1, 2, \dots, M - 1. \quad (1.28)$$

and seeing how close it comes to satisfying this finite difference equation.

Note that from Taylor theorem we have

$$\begin{aligned}
y(t_{j+1}) &= y(t_j + k) \\
&= y(t_j) + ky'(t_j) + \frac{1}{2}k^2y''(t_j) + \dots \\
&= y(t_j) + kf(t_j, y(t_j)) + \frac{1}{2}k^2y''(t_j) + \dots \quad (1.39)
\end{aligned}$$

Now, substituting  $y(t_j)$  for  $t_j$  in (1.28) yields

$$y(t_{j+1}) \stackrel{?}{=} y(t_j) + kf(t_j, y(t_j)). \quad (1.40)$$

A question mark is put above the equal sign here, because we are investigating whether  $y(t)$  satisfies (1.28) or, more precisely, how close it comes to satisfying this equation. With (1.39) the question of whether (1.40) is satisfied can be written as

$$y(t_j) + kf(t_j, y(t_j)) + \frac{1}{2}k^2y''(t_j) + \dots \stackrel{?}{=} y(t_j) + kf(t_j, y(t_j)),$$

or, equivalently,

$$\frac{1}{2}k^2y''(t_j) + \dots \stackrel{?}{=} 0. \quad (1.41)$$

The conclusion from this last step is that  $y(t_j)$  misses satisfying (1.28) by a term that is  $O(k^2)$ , and from this it follows that the truncation error is  $O(k)$ . Because the truncation error goes to zero with  $k$  it follows that the method is consistent. Of course we already knew this, but the above calculation shows that if necessary, it is possible to determine this directly from the finite difference equation.

## Stability

### Step 5

It is not unreasonable to think that as long as the problem is approximated consistently, then the numerical solution will converge to the exact solution as the time step is refined.

Unfortunately, as demonstrated shortly using the leapfrog method, consistency is not enough. To explain what is missing, the approximation that produced Euler's method means that even though  $y_0$  is known exactly, the method computes a value for  $y_1$  that differs a bit from the exact value  $y(t_1)$ . Moreover, this difference affects the values of all the  $y_j$ 's that come afterwards. It is essential that the method not magnify these differences as  $t_j$  increases, and this is true irrespective of the time step at which the error is produced.

This is the idea underlying the concept of **stability**.

There are various ways to express this condition and we will use one of the stronger forms, something known as **A-stability**. This is determined by using the method to solve the radioactive decay equation

$$\frac{dy}{dt} = -ry, \quad (1.42)$$

where

$$y(0) = \alpha. \tag{1.43}$$

As before, it is assumed that  $r$  is positive. For this equation the Euler method (1.28) reduces to

$$y_{j+1} = (1 - rk)y_j. \tag{1.44}$$

The solution of this that satisfies the initial condition is

$$y_j = \alpha(1 - rk)^j. \tag{1.45}$$

In fact,

$$\left[ \begin{array}{l} > \text{simplify}(\alpha \cdot (1 - rk)^{j+1} - (1 - rk) \cdot \alpha \cdot (1 - rk)^j) \\ & \qquad \qquad \qquad 0 \end{array} \right. \tag{2.9}$$

In comparison, the exact solution to the IVP is  $y(t) = \alpha \exp(-rt)$ , and this function approaches zero as  $t$  increases. It is required at the very least that the numerical solution of this problem not grow, and this is the basis for the following definition.

**Definition 1.1.** If the method, when applied to (1.42), produces a bounded solution irrespective of the (positive) value of  $r$  and  $k$ , then the method is said to be A-stable. If boundedness occurs only when  $k$  is small then the method is conditionally A-stable. Otherwise, the method is unstable.

The Euler solution in (1.45) remains bounded as  $j$  increases only as long as  $|1 - rk| \leq 1$ . This occurs if the step size is chosen to satisfy the condition  $k \leq 2/r$ . Therefore, the Euler method is conditionally A-stable. It is worth looking at what happens in the unstable case. If we take a step size that does not satisfy the stability condition, say  $k = 3/r$ , then  $y_j = \alpha \cdot (-2)^j$ .

The solution in this case oscillates with an amplitude that increases as  $t_j$  increases. This is similar to what was seen when the Tacoma bridge collapsed, where relatively small oscillations grew and eventually became so large the bridge came apart. Whenever such growing oscillatory behavior appears in a numerical solution one should seriously consider whether one has been unfortunate enough to have picked a step size that falls in the instability region.

One last point to make here is that one of the central questions arising when using any numerical method to solve a differential equation concerns what properties of the problem the numerical method is able to preserve. For example, if energy is conserved in the original problem then it is natural to ask whether the numerical method does the same. As we will see, preserving particular properties, such as energy conservation or the monotonicity of the solution, can have profound consequences on how well the method works. It is within this context that the requirement of A-stability is introduced.

The radioactive decay problem possesses an asymptotically stable equilibrium solution  $y = 0$ . A-stability is nothing more than the requirement that  $y = 0$  be at least a stable equilibrium solution for the method (i.e., any solution starting near  $y = 0$  will remain near this solution as time increases). As we found earlier, the stability interval for Euler's method is  $k \leq 2/r$ . On the interior of this interval,  $k < 2/r$ , the equilibrium solution  $y = 0$  is asymptotically stable because  $y_j \rightarrow 0$  as  $t_j \rightarrow \infty$ .

This observation accounts for why you will occasionally see a requirement of strict A-stability, where boundedness is replaced with the requirement that  $y_j \rightarrow 0$  as  $t_j \rightarrow \infty$ . The reason is that a strictly A-stable method preserves asymptotic stability. Our conclusion in this case would be that Euler is strictly A-stable when strict inequality holds, namely,  $k < 2/r$ .

## Note

One might wonder why the radioactive decay problem is used as the arbiter for deciding whether a method is A-stable.

To explain how this happens suppose the differential equation is a bit simpler than the one in (1.18) and has the form  $y' = f(y)$ . Also, suppose  $y = Y$  is an asymptotically stable equilibrium solution. This means that the constant  $Y$  is a solution of the equation and any initial condition  $y(0) = \alpha$  chosen close to  $Y$  will result in the solution of the IVP converging to  $Y$  as  $t \rightarrow \infty$ . Now, to determine how the solution behaves near  $y = Y$  let  $v(t) = y(t) - Y$ . Substituting this into the equation, one gets that  $v' = f(Y + v)$ . With  $\alpha$  close to  $Y$  we have that  $v(t)$  starts out relatively small. Consequently, using Taylor's theorem,

$$f(Y + v) = f(Y) + v f'(Y) + O(v^2) \approx f(Y) + v f'(Y).$$

Because  $f(Y) = 0$  we conclude that  $v' = -rv$ , where  $r = -f'(Y)$ .

Therefore the solution near  $y = Y$  is governed by the radioactive decay equation, and that is why it is used to determine A-stability.

For the earlier example using the logistic equation,

$$\frac{dy}{dt} = 10y(1 - y), \quad \text{for } 0 < t, \quad (1.30)$$

$f = \lambda y(1-y)$  and  $r = \lambda(-1+2Y)$ . Because  $Y = 1$ , so  $r = \lambda$ , then the stability requirement when using the Euler method to solve the logistic equation is  $k \leq 2/\lambda$ . Taking  $\lambda = 10$ , then the stability condition is  $k \leq 1/5$ . This helps explain the rather poor showing of the  $M = 4$  curve, since  $k = 1/4$ .

## Systems of equations

Another observation to make is that even though the discussion has centered on problems involving one differential equation, the ideas are easily extended to systems of equations. For example, assuming that the equation is  $y' = f(t, y)$ , then the vector version of the forward difference used in (1.23)

$$y'(t_j) = \frac{y(t_{j+1}) - y(t_j)}{k} + \tau_j, \quad (1.23)$$

where

$$\tau_j = -\frac{k}{2} y''(\eta_j) \quad (1.24)$$

is

$$y'(t_j) = \frac{y(t_{j+1}) - y(t_j)}{k} + O(k). \quad (1.46)$$

Substituting this into the differential equation and dropping the truncation error produces the vector form of the Euler method given in Table 1.3. This formula can be obtained directly from the single-variable version in (1.28)

$$y_{j+1} = y_j + k f(t_j, y_j), \quad \text{for } j = 0, 1, 2, \dots, M - 1. \quad (1.28)$$

by simply converting the appropriate variables to vectors. The same is true for most of the other methods considered in this chapter, the exception arising with Runge–Kutta methods and this is discussed later.

| Methods for solving the differential equation           |  |          |  |
|---|--|----------|--|
| $\frac{d}{dt}\mathbf{y}(t) = \mathbf{f}(t, \mathbf{y})$ |  |          |  |
| Method  | Difference Formula   | $\tau_j$ | Properties                             |
| Euler   | $\mathbf{y}_{j+1} = \mathbf{y}_j + k\mathbf{f}_j$  | $O(k)$   | Explicit;<br>Conditionally<br>A-stable |
| Backward<br>Euler                                       | $\mathbf{y}_{j+1} = \mathbf{y}_j + k\mathbf{f}_{j+1}$  | $O(k)$   | Implicit;<br>A-stable                  |
| Trapezoidal   | $\mathbf{y}_{j+1} = \mathbf{y}_j + \frac{k}{2}(\mathbf{f}_j + \mathbf{f}_{j+1})$   | $O(k^2)$ | Implicit;<br>A-stable                  |
| Heun<br>(RK2)   | $\mathbf{y}_{j+1} = \mathbf{y}_j + \frac{1}{2}(k_1 + k_2)$<br>where<br>$k_1 = k\mathbf{f}_j$<br>$k_2 = k\mathbf{f}(t_{j+1}, \mathbf{y}_j + k_1)$   | $O(k^2)$ | Explicit;<br>Conditionally<br>A-stable |
| Classical<br>Runge–<br>Kutta<br>(RK4)                   | $\mathbf{y}_{j+1} = \mathbf{y}_j + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$<br>where<br>$k_1 = k\mathbf{f}_j$<br>$k_2 = k\mathbf{f}(t_j + \frac{k}{2}, \mathbf{y}_j + \frac{1}{2}k_1)$<br>$k_3 = k\mathbf{f}(t_j + \frac{k}{2}, \mathbf{y}_j + \frac{1}{2}k_2)$<br>$k_4 = k\mathbf{f}(t_{j+1}, \mathbf{y}_j + k_3)$ | $O(k^4)$ | Explicit;<br>Conditionally<br>A-stable |

**Table 1.3.** Finite difference methods for solving an IVP. The points  $t_1, t_2, t_3, \dots$  are equally spaced with step size  $k = t_{j+1} - t_j$ . Also,  $\mathbf{f}_j = \mathbf{f}(t_j, \mathbf{y}_j)$  and  $\tau_j$  is the truncation error for the method.

A limitation of approximating vector derivatives in this way is that every equation is approximated the same way. For example, in (1.46) each component of the vector  $\mathbf{y}'(t_j)$  is approximated using the Euler formula. There are situations in which it is better to use different approximations on different components, and an example of this is explored in Section 1.6.

One last comment to make concerns A-stability for systems. The generalization of (1.42) for systems is the equation  $\mathbf{y}' = -\mathbf{A}\mathbf{y}$ , where  $\mathbf{A}$  is a matrix with constant coefficients.

Similar to what occurred earlier, this matrix can be thought of as derived from a local approximation

of the Jacobian matrix  $\partial f_i / \partial y_j$

The requirement for A-stability remains the same, namely that the method produces bounded solutions for any A that results in  $y = 0$  being an asymptotically stable equilibrium solution of the original problem. To illustrate what is involved, if A is diagonalizable with eigenvalues  $\lambda_1, \lambda_2, \dots, \lambda_n$  then  $y = 0$  is an asymptotically stable equilibrium solution if  $\text{Re}(\lambda_i) > 0, \forall i$ . Now, Euler applied to  $y' = -Ay$  yields the finite difference equation  $y_{j+1} = (I - kA)y_j$ , where I is the identity matrix. Using the diagonalizability of A it is possible to reduce this to scalar equations of the form  $z_{j+1} = (1 - kr)z_j$ , where r is an eigenvalue of A. Consequently, the problem has been reduced to (1.44), except that r is now complex-valued with  $\text{Re}(r) > 0$ . The conclusion is therefore the same, namely that Euler is conditionally A-stable. This example also demonstrates that the scalar equation in (1.42) serves as an adequate test problem for A-stability, and it is the one we use throughout this chapter. Those interested in a more extensive development of A-stability for systems should consult the text by Deuflhard et al. [2002].

### 1.2.2 Additional Difference Methods

The steps used to derive the Euler method can be employed to obtain a host of other finite difference approximations. The point in the derivation that separates one method from another is Step 3, where one makes a choice for the difference formula. Most of the formulas used in this book are listed in Table 1.1. It is interesting to see what sort of numerical methods can be derived using these expressions, and a few of the possibilities are discussed below.

#### Backward Euler

If one uses the backward difference formula in Table 1.1, then in place of (1.23), we get

$$y'(t_j) = \frac{y(t_j) - y(t_{j-1})}{k} + \tau_j, \quad (1.47)$$

where

$$\tau_j = \frac{k}{2} y''(\eta_j). \quad (1.48)$$

Introducing this into (1.26),

$$y(t_{j+1}) - y(t_j) + k\tau_j = kf(t_j, y(t_j)). \quad (1.26)$$

we obtain

$$y(t_j) - y(t_{j-1}) + k\tau_j = kf(t_j, y(t_j)). \quad (1.49)$$

Dropping the truncation error  $\tau_j$ , the resulting finite difference approximation is

$$y_j = y_{j-1} + kf(t_j, y_j), \quad \text{for } j = 1, 2, \dots, M. \quad (1.50)$$

From the initial condition (1.19) we have that the starting value is

$$y_0 = \alpha. \quad (1.51)$$

The difference equation in (1.50) is the backward Euler method. It has the same order of truncation error as the Euler method. However, because of the  $f(t_j, y_j)$  term this method is implicit. This is both good and bad. It is good because it helps make the method A-stable (see below). However, it is bad because it can make finding  $y_j$  computationally difficult. Unless the problem is simple enough that the difference equation can be solved by hand, it is necessary to use something like Newton's method to solve (1.50), and this must be done for each time step. Some of

the issues that arise with this situation are developed in Exercise 1.33.

As for stability (Step 5), for the radioactive decay equation (1.42) one finds that (1.50) reduces to

$$y_{j+1} = \frac{1}{1 + rk} y_j. \quad (1.52)$$

Assuming  $y_0 = \alpha$ , then the solution of this finite difference equation is

$$y_j = \alpha \cdot (1 + rk)^{-j}$$

This goes to zero as  $j$  increases irrespective of the value of  $k$ . Consequently, this method is A-stable. Another point to its credit is that the solution decays monotonically to zero, just as does the exact solution. For this reason backward Euler is said to be a monotone method.

In contrast, recall that for Euler's method the solution is

$$y_j = \alpha(1 - rk)^j$$

If

$$-1 < 1 - rk < 0,$$

which is part of the stability interval for the method, the resulting solution goes to zero, but it oscillates as it does so. In other words, Euler's method is monotone only if

$$0 < 1 - rk < 1$$

(i.e., it is conditionally monotone). The fact that backward Euler preserves the monotonicity of the solution is, for some problems, important, and this is explored in more depth in Exercises 1.11 and 1.12. We will return to this issue of a monotone scheme in Chapter 4, when we investigate how to solve wave propagation problems.

**Example.** Let us solve the following IVP ( $y(0)=1$ ) with the Euler backward method

[> restart

[>  $f := (t, y) \rightarrow \frac{1}{2} - t + 2 \cdot y;$

$$f := (t, y) \rightarrow \frac{1}{2} - t + 2y \quad (2.10)$$

[>  $eq := \text{diff}(y(t), t) = f(t, y(t));$

$$eq := \frac{d}{dt} y(t) = \frac{1}{2} - t + 2y(t) \quad (2.11)$$

[>

Exact solution:

[>  $soln := \text{dsolve}(\{eq, y(0) = 1\});$

$$soln := y(t) = \frac{1}{2} t + e^{2t} \quad (2.12)$$

Let's approximate  $y(1)$  using a step size of  $h = \frac{1}{30}$ . This means we need  $N = 30$  iterations.

[>  $N := 30; h := \frac{1}{N}$

$N := 30$

$$h := \frac{1}{30} \quad (2.13)$$

Here's Backwards Euler's Method. Recall the Backwards Euler's is an implicit method. So we need to solve

for  $y_n$  each time through the loop.

$t_0 = 0$  and  $y_0 = 1$  (from our initial condition)

```
> t[0] := 0; y[0] := 1;
  for n to N do
    t[n] := t[n-1]+h;
    y[n] := solve(X = y[n-1]+f(t[n], X)*h, X)
  end do;
```

$$t_0 := 0$$

$$y_0 := 1$$

$$t_1 := \frac{1}{30}$$

$$y_1 := \frac{457}{420}$$

$$t_2 := \frac{1}{15}$$

$$y_2 := \frac{3473}{2940}$$

$$t_3 := \frac{1}{10}$$

$$y_3 := \frac{17561}{13720}$$

$$t_4 := \frac{2}{15}$$

$$y_4 := \frac{797791}{576240}$$

$$t_5 := \frac{1}{6}$$

$$y_5 := \frac{2412581}{1613472}$$

$$t_6 := \frac{1}{5}$$

$$y_6 := \frac{60717893}{37647680}$$

$$t_7 := \frac{7}{30}$$

$$y_7 := \frac{2747364257}{1581202560}$$

$$t_8 := \frac{4}{15}$$

$$y_8 := \frac{41394937487}{22136835840}$$

$$t_9 := \frac{3}{10}$$

$$y_9 := \frac{207712581963}{103305233920}$$

$$t_{10} := \frac{1}{3}$$

$$y_{10} := \frac{1874578499363}{867763964928}$$

$$t_{11} := \frac{11}{30}$$

$$y_{11} := \frac{140882642107201}{60743477544960}$$

$$t_{12} := \frac{2}{5}$$

$$y_{12} := \frac{705425601828421}{283469561876480}$$

$$t_{13} := \frac{13}{30}$$

$$y_{13} := \frac{31772499038466593}{11905721598812160}$$

$$t_{14} := \frac{7}{15}$$

$$y_{14} := \frac{476785914270312431}{166680102383370240}$$

$$t_{15} := \frac{1}{2}$$

$$y_{15} := \frac{476785914270312431}{155568095557812224}$$

$$t_{16} := \frac{8}{15}$$

$$y_{16} := \frac{107237938686930843919}{32669300067140567040}$$

$$t_{17} := \frac{17}{30}$$

$$y_{17} := \frac{1607480103635057973217}{457370200939967938560}$$

$$t_{18} := \frac{3}{5}$$

$$y_{18} := \frac{8029777681492957067109}{2134394271053183713280}$$

$$t_{19} := \frac{19}{30}$$

$$y_{19} := \frac{360913116812972431277249}{89644559384233715957760}$$

$$t_{20} := \frac{2}{3}$$

$$y_{20} := \frac{1081245274449180065232451}{251004766275854404681728}$$

$$t_{21} := \frac{7}{10}$$

$$y_{21} := \frac{26989297733516859230030987}{5856777879769936109240320}$$

$$t_{22} := \frac{11}{15}$$

$$y_{22} := \frac{1212468525750339187713160303}{245984670950337316588093440}$$

$$t_{23} := \frac{23}{30}$$

$$y_{23} := \frac{18154229930128376173485658753}{3443785393304722432233308160}$$

$$t_{24} := \frac{4}{5}$$

$$y_{24} := \frac{90598960380976644745816628357}{16070998502088704683755438080}$$

$$t_{25} := \frac{5}{6}$$

$$y_{25} := \frac{813783543578580932243974111405}{134996387417545119343545679872}$$

$$t_{26} := \frac{13}{15}$$

$$y_{26} := \frac{60910019079927486892233141482159}{9449747119228158354048197591040}$$

$$t_{27} := \frac{9}{10}$$

$$y_{27} := \frac{303920112258355557237562494238059}{44098819889731405652224922091520}$$

$$t_{28} := \frac{14}{15}$$

$$y_{28} := \frac{13647740818697674662016366041353167}{1852150435368719037393446727843840}$$

$$t_{29} := \frac{29}{30}$$

$$y_{29} := \frac{204283943845545752154853686383800609}{25930106095162066523508254189813760}$$

$$t_{30} := 1$$

$$y_{30} := \frac{203851775410626384379461882147303713}{24201432355484595421941037243826176} \quad (2.14)$$

```
> y[N]
```

$$\frac{203851775410626384379461882147303713}{24201432355484595421941037243826176} \quad (2.15)$$

Let's print out a decimal approximation of  $y_N$ .

```
> yApprox := evalf(y_N);
```

$$yApprox := 8.423128533 \quad (2.16)$$

The exact value of  $y(1)$  is given below:

```
> yExact := subs(t=N*h, rhs(soln));
```

$$yExact := \frac{1}{2} + e^2 \quad (2.17)$$

The difference between the exact value and the approximated value gives us the "global truncation error".

```
> evalf(yExact - yApprox);
```

$$-0.534072434 \quad (2.18)$$

We could also write loop where the the difference equation is already solved.

```
> restart
```

```
> f := (t, y) -> 1/2 - t + 2 y
```

$$f := (t, y) \rightarrow \frac{1}{2} - t + 2 y \quad (2.19)$$

```
> solve(X = y[n-1]+f(t[n], X)*h, X);
```

$$\frac{1}{2} \frac{-2 y_{n-1} - h + 2 h t_n}{-1 + 2 h} \quad (2.20)$$

```
> N := 30; h := 1/N;
```

$$N := 30$$

$$h := \frac{1}{30} \quad (2.21)$$

```
> t[0] := 0;
y[0] := 1;
for n to N do
  t[n] := t[n-1]+h;
  y[n] := (1/2)*(-2*y[n-1]-h+2*h*t[n])/(-1+2*h)
end do;
```

$$t_0 := 0$$

$$y_0 := 1$$

$$t_1 := \frac{1}{30}$$

$$y_1 := \frac{457}{420}$$

$$t_2 := \frac{1}{15}$$

$$y_2 := \frac{3473}{2940}$$

$$t_3 := \frac{1}{10}$$

$$y_3 := \frac{17561}{13720}$$

$$t_4 := \frac{2}{15}$$

$$y_4 := \frac{797791}{576240}$$

$$t_5 := \frac{1}{6}$$

$$y_5 := \frac{2412581}{1613472}$$

$$t_6 := \frac{1}{5}$$

$$y_6 := \frac{60717893}{37647680}$$

$$t_7 := \frac{7}{30}$$

$$y_7 := \frac{2747364257}{1581202560}$$

$$t_8 := \frac{4}{15}$$

$$y_8 := \frac{41394937487}{22136835840}$$

$$t_9 := \frac{3}{10}$$

$$y_9 := \frac{207712581963}{103305233920}$$

$$t_{10} := \frac{1}{3}$$

$$y_{10} := \frac{1874578499363}{867763964928}$$

$$t_{11} := \frac{11}{30}$$

$$y_{11} := \frac{140882642107201}{60743477544960}$$

$$t_{12} := \frac{2}{5}$$

$$y_{12} := \frac{705425601828421}{283469561876480}$$

$$t_{13} := \frac{13}{30}$$

$$y_{13} := \frac{31772499038466593}{11905721598812160}$$

$$t_{14} := \frac{7}{15}$$

$$y_{14} := \frac{476785914270312431}{166680102383370240}$$

$$t_{15} := \frac{1}{2}$$

$$y_{15} := \frac{476785914270312431}{155568095557812224}$$

$$t_{16} := \frac{8}{15}$$

$$y_{16} := \frac{107237938686930843919}{32669300067140567040}$$

$$t_{17} := \frac{17}{30}$$

$$y_{17} := \frac{1607480103635057973217}{457370200939967938560}$$

$$t_{18} := \frac{3}{5}$$

$$y_{18} := \frac{8029777681492957067109}{2134394271053183713280}$$

$$t_{19} := \frac{19}{30}$$

$$y_{19} := \frac{360913116812972431277249}{89644559384233715957760}$$

$$t_{20} := \frac{2}{3}$$

$$y_{20} := \frac{1081245274449180065232451}{251004766275854404681728}$$

$$t_{21} := \frac{7}{10}$$

$$y_{21} := \frac{26989297733516859230030987}{5856777879769936109240320}$$

$$t_{22} := \frac{11}{15}$$

$$y_{22} := \frac{1212468525750339187713160303}{245984670950337316588093440}$$

$$t_{23} := \frac{23}{30}$$

$$\begin{aligned}
y_{23} &:= \frac{18154229930128376173485658753}{3443785393304722432233308160} \\
t_{24} &:= \frac{4}{5} \\
y_{24} &:= \frac{90598960380976644745816628357}{16070998502088704683755438080} \\
t_{25} &:= \frac{5}{6} \\
y_{25} &:= \frac{813783543578580932243974111405}{134996387417545119343545679872} \\
t_{26} &:= \frac{13}{15} \\
y_{26} &:= \frac{60910019079927486892233141482159}{9449747119228158354048197591040} \\
t_{27} &:= \frac{9}{10} \\
y_{27} &:= \frac{303920112258355557237562494238059}{44098819889731405652224922091520} \\
t_{28} &:= \frac{14}{15} \\
y_{28} &:= \frac{13647740818697674662016366041353167}{1852150435368719037393446727843840} \\
t_{29} &:= \frac{29}{30} \\
y_{29} &:= \frac{204283943845545752154853686383800609}{25930106095162066523508254189813760} \\
t_{30} &:= 1 \\
y_{30} &:= \frac{203851775410626384379461882147303713}{24201432355484595421941037243826176}
\end{aligned} \tag{2.22}$$

$$\begin{aligned}
&> \text{evalf}(y[N]); \\
& \qquad \qquad \qquad 8.423128533
\end{aligned} \tag{2.23}$$

Obviously, this is possible only because the difference equation can be algebraically solved. Let us consider next an example where the backward Euler method requires that the difference equation is solved at each step.

### Example

We consider now the nonlinear equation  $y' = \frac{1}{2} - t + \sin(y) - y$ .

$$\begin{aligned}
&> \text{restart}; \\
&> f := (t, y) \rightarrow \frac{1}{2} - t + \sin(y) - y \\
& \qquad \qquad \qquad f := (t, y) \rightarrow \frac{1}{2} - t + \sin(y) - y
\end{aligned} \tag{2.24}$$

$$\begin{aligned}
&> N := 100: h := 1/N:
\end{aligned}$$

```

> t[0] := 0: y[0] := 1:
> L := [[t[0],y[0]]];
                                L:= [[0, 1]]
                                (2.25)
> for n to N do
    t[n] := t[n-1]+h;
    y[n] := fsolve(X = y[n-1]+f(t[n], X)*h, X);
    L:=evalf([op(L), [t[n],y[n]]]);
end do:

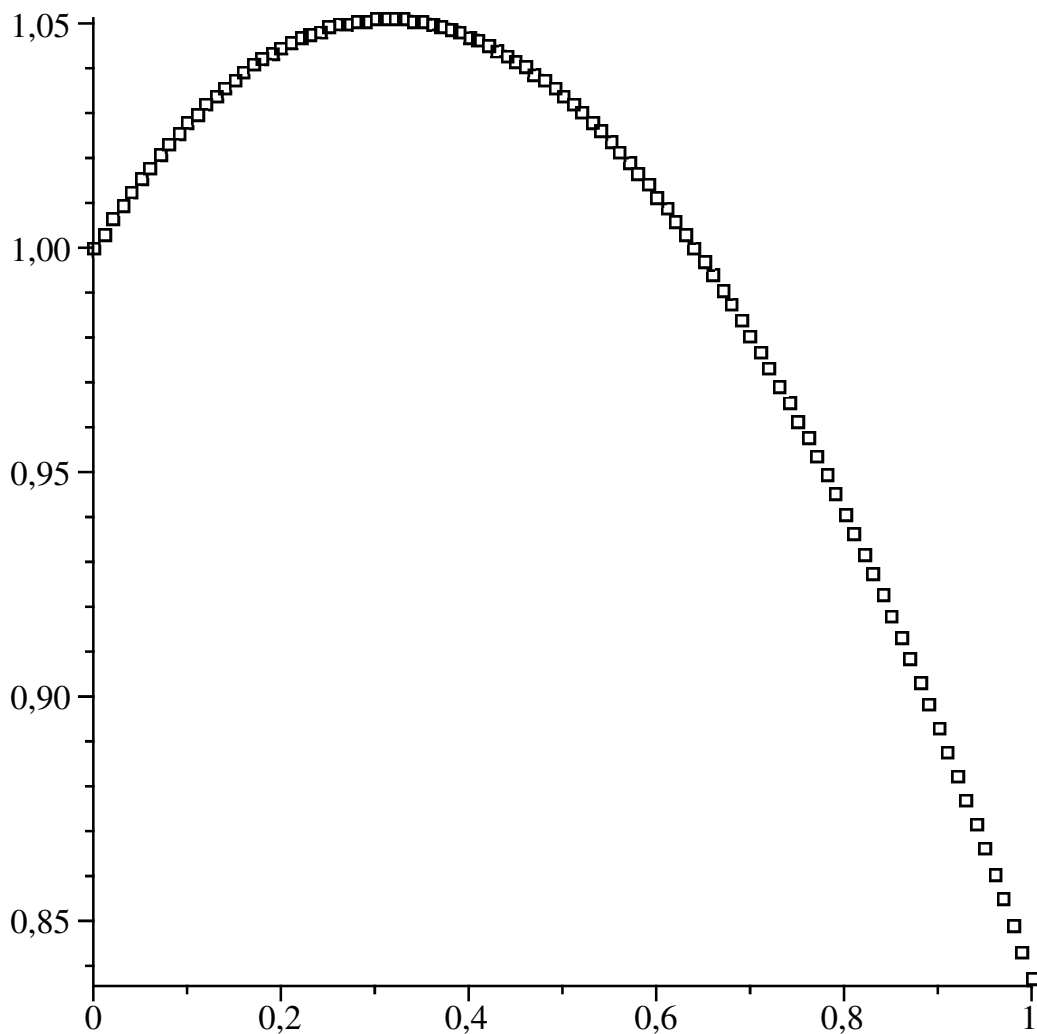
```

Let us graph the numerical solution.

```

> with(plots) :
> p1 := pointplot(L, color = black, symbol = box) :
> display(p1)

```



Note that

```

> y[N]
                                0.8372782260
                                (2.26)

```

An exact solution to this equation does not exist. We compare our solution with that obtained by the rkf45 method (Fehlberg fourth-fifth order Runge-Kutta method with degree four interpolant - we will study this later), implemented in Maple.

```
> eq := diff(Y(T), T) = 1/2-T+sin(Y(T))-Y(T);
      eq :=  $\frac{d}{dT} Y(T) = \frac{1}{2} - T + \sin(Y(T)) - Y(T)$  (2.27)
```

```
> solrk := dsolve({eq, Y(0) = 1}, numeric, method = rkf45)
      solrk := proc(x_rkf45) ... end proc (2.28)
```

The solution is calculated at our request. For example,

```
> solrk(3.6)
      [T=3.6, Y(T) = -2.63846194068569595] (2.29)
```

We can extract the desired value of the function using the command `op`. Next we calculate the difference of the function, calculated by both methods at  $t = 1$ .

```
> op(solrk(4)[2])[2]
      -2.96466218262903870 (2.30)
```

```
>
> op(solrk(4)[2])[2]
      -2.96466218262903870 (2.31)
```

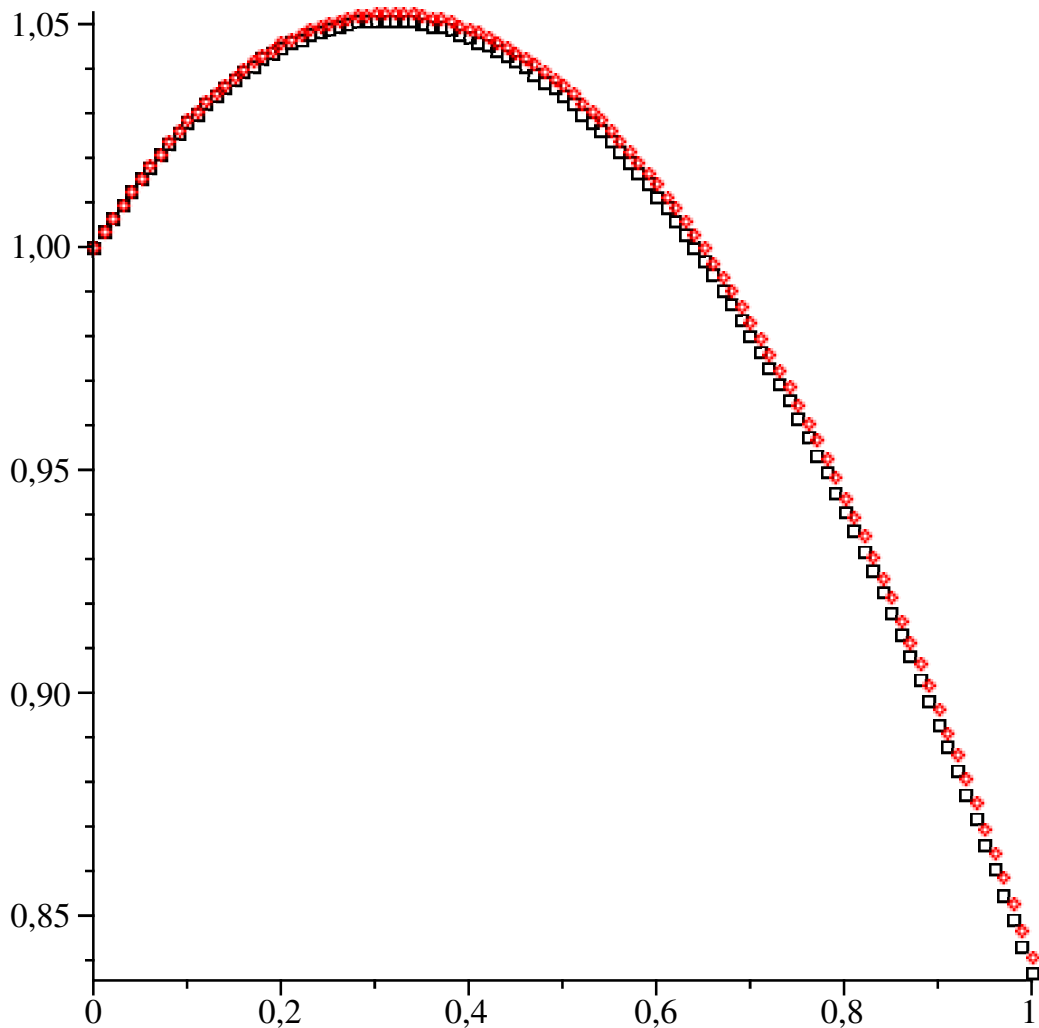
```
> y[N] - op(solrk(1)[2])[2]
      -0.0037372157 (2.32)
```

which is pretty close. In order to get an idea of the relative importance of this error we plot both results:

```
> T[0] := 0 : Y[0] := 1 :
> L2 := [[T[0], Y[0]]];
      L2 := [[0, 1]] (2.33)
```

```
> for n to N do
      T[n] := T[n-1] + h;
      L2 := [op(L2), [T[n], op(solrk(T[n])[2])[2]]]
    od:
> p2 := pointplot(L2, color = red)
      p2 := PLOT(...) (2.34)
```

```
> display([p1, p2])
```



[>

Not bad for backward Euler.

**Exercise:** Verify this result using DEplot command.

[>

### Leapfrog Method

It is natural to expect that a more accurate approximation of the derivative will improve the resulting finite difference approximation of the differential equation. In looking over Table 1.1, the centered difference formula would appear to be a good choice for such an improvement because it has quadratic error (versus linear for the first two formulas listed). Introducing this into (1.18)

$$\frac{dy}{dt} = f(t, y), \quad \text{for } 0 < t, \quad (1.18)$$

where

$$y(0) = \alpha. \quad (1.19)$$

we obtain

$$y(t_{j+1}) - y(t_{j-1}) + 2k\tau_j = 2kf(t_j, y(t_j)), \quad (1.53)$$

where

$$\tau_j = O(k^2).$$

Dropping the truncation error  $\tau_j$ , the resulting finite difference approximation is

$$y_{j+1} = y_{j-1} + 2kf(t_j, y_j), \quad \text{for } j = 1, 2, \dots, M-1. \quad (1.54)$$

This is known as the leapfrog, or explicit midpoint, method. Because this equation uses information from two previous time steps it is an example of a two-step method. In contrast, both Euler methods use information from a single time step back, so they are one-step methods. What this means is that the initial condition (1.19) is not enough information to get leapfrog started, because we also need  $y_1$ . This is a relatively minor inconvenience that will be addressed later. It is more interesting right now to concentrate on the truncation error.

It would seem that the leapfrog method, with its  $O(k^2)$  truncation error, will produce a more accurate numerical solution than either of the two Euler methods. As it turns out, this apparently obvious conclusion **could not be farther from the truth.**

This becomes evident from our stability test. Applying (1.54) to the radioactive decay equation (1.42) yields

$$y_{j+1} = y_{j-1} - 2rky_j$$

This second-order difference equation can be solved by assuming a solution of the form  $y_j = s^j$ . This results in a general solution the form

$$y_j = \alpha_0 s_+^j + \alpha_1 s_-^j$$

where

$$s_{\pm} = -kr \pm \sqrt{1 + k^2 r^2}$$

and  $\alpha_0$  and  $\alpha_1$  are constants.

Proof:

$$\begin{aligned} & \left[ \begin{array}{l} > \text{eq} := s = \frac{1}{s} - 2 \cdot r \cdot k \\ & \text{eq} := s = \frac{1}{s} - 2rk \end{array} \right. \quad (2.35) \end{aligned}$$

$$\begin{aligned} & \left[ \begin{array}{l} > \text{solve}(\text{eq}, s); \\ & -rk + \sqrt{r^2 k^2 + 1}, -rk - \sqrt{r^2 k^2 + 1} \end{array} \right. \quad (2.36) \end{aligned}$$

> Because  $|s| > 1$ , it is impossible to find a step size  $k$  to satisfy the stability condition. Therefore, the leapfrog method is unstable.

**Exercise:** Apply the leapfrog method to solve the the logistic equation. Show how unstabilities appear in the last case.

[>

## Methods Obtained from Numerical Quadrature

Another approach to deriving a finite difference approximation of an IVP is to integrate the differential equation and then use a numerical integration rule.

This is a very useful idea that is best explained by working through an example. To get started, a time grid must be introduced, and so Step 1 is the same as before. However, Step 2 and Step 3 differ from what we did earlier.

**Step 2.** Integrate the differential equation between two time points. We will take  $t_j$  and  $t_{j+1}$ , and so from (1.18) we have

$$\int_{t_j}^{t_{j+1}} \frac{dy}{dt} dt = \int_{t_j}^{t_{j+1}} f(t, y(t)) dt. \quad (1.55)$$

Using the Fundamental Theorem of Calculus we obtain

$$y(t_{j+1}) - y(t_j) = \int_{t_j}^{t_{j+1}} f(t, y(t)) dt. \quad (1.56)$$

**Step 3.** Replace the integral in Step 2 with a finite difference approximation. This is where things get a bit interesting, because there are numerous choices, and they produce different numerical procedures. A few of the most often used possibilities are listed in Table 1.4.

| Rule        | Integration Formula   |
|-------------|---|
| Right Box   | $\int_{x_i}^{x_{i+1}} f(x) dx = hf(x_{i+1}) + O(h^2)$   |
| Left Box    | $\int_{x_i}^{x_{i+1}} f(x) dx = hf(x_i) + O(h^2)$   |
| Midpoint    | $\int_{x_{i-1}}^{x_{i+1}} f(x) dx = 2hf(x_i) + \frac{h^3}{3} f''(\eta_i)$   |
| Trapezoidal | $\int_{x_i}^{x_{i+1}} f(x) dx = \frac{h}{2} (f(x_i) + f(x_{i+1})) - \frac{h^3}{12} f''(\eta_i)$                     |
| Simpson     | $\int_{x_{i-1}}^{x_{i+1}} f(x) dx = \frac{h}{3} (f(x_{i+1}) + 4f(x_i) + f(x_{i-1})) - \frac{h^5}{90} f''''(\eta_i)$ |

**Table 1.4.** Numerical integration formulas. The points  $x_1, x_2, x_3, \dots$  are equally spaced with step size  $h = x_{i+1} - x_i$ . The point  $\eta_i$  is located within the interval of integration.

We will use the trapezoidal rule, and introducing this into (1.56) yields



$$Z := y_{n-1} \quad (3.4)$$

>  $y[n] := \text{sol}[2]$

$$y_n := -\frac{1}{10} \frac{-5h + 1 + \sqrt{25h^2 - 10h + 1 + 100h^2 y_{n-1} + 20h y_{n-1} - 100h^2 y_{n-1}^2}}{h} \quad (3.5)$$

>  $\text{subs}\left(\left\{n = 1, h = \frac{1}{30}, y[0] = 1\right\}, y[n]\right)$

$$-\frac{5}{2} - 3 \sqrt{\frac{25}{36} + \frac{7}{9} y_0 - \frac{1}{9} y_0^2} \quad (3.6)$$

>

>  $N := 30; h := 1/N;$

$$N := 30$$

$$h := \frac{1}{30} \quad (3.7)$$

>  $t[0] := 0; y[0] := 0.01;$

$$t_0 := 0$$

$$y_0 := 0.01 \quad (3.8)$$

>  $L := [[t[0], y[0]]];$

$$L := [[0, 0.01]]$$

$$(3.9)$$

>

> **for**  $n$  **to**  $N$  **do**

$t[n] := t[n-1] + h;$

$y[n] := \text{evalf}((1/10) * (5*h - 1 + \text{sqrt}(25*h^2 - 10*h + 1 + 100*h^2*y[n-1] + 20*h*y[n-1] - 100*h^2*y[n-1]^2)) / h);$

$L := \text{evalf}([\text{op}(L), [t[n], y[n]]]);$

**end do;**

We now generate the plot for this approximated solution:

>  $\text{with}(\text{plots}) :$

>  $p1 := \text{pointplot}(L, \text{color} = \text{blue}, \text{symbol} = \text{diamond})$

$$p1 := \text{PLOT}(\dots)$$

$$(3.10)$$

The exact solution is

>  $\text{eq} := \text{diff}(Y(T), T) = 10 * Y(T) * (1 - Y(T));$

$$\text{eq} := \frac{d}{dT} Y(T) = 10 Y(T) (1 - Y(T)) \quad (3.11)$$

>  $\text{exact\_sol} := \text{dsolve}(\{\text{eq}, Y(0) = 0.01\}, Y(T));$

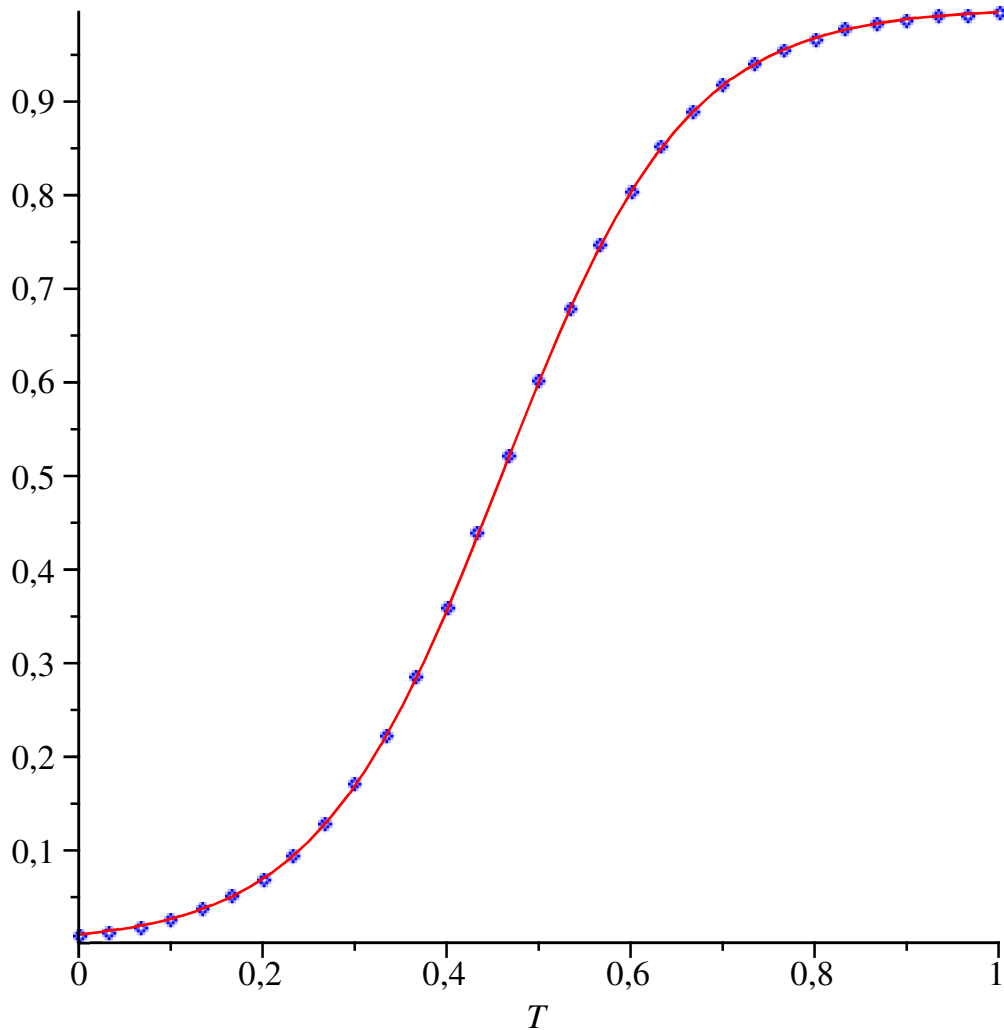
$$\text{exact\_sol} := Y(T) = \frac{1}{1 + 99 e^{-10T}} \quad (3.12)$$

>  $Y_{\text{exact}} := \text{unapply}(\text{op}(2, \text{exact\_sol}), T);$

$$Y_{\text{exact}} := T \rightarrow \frac{1}{1 + 99 e^{-10T}} \quad (3.13)$$

>  $p2 := \text{plot}(Y_{\text{exact}}(T), T = 0..1) :$

```
> display([p1,p2]);
```



```
> Yexact(1.) - L[30][2]
```

0.0015439999

(3.14)

This illustrates the good accuracy of the trapezoidal method

**Exercise.** What is the effect of taking the other solution of the difference equation ?

**Exercise.** Use the trapezoidal method to solve the IVP  $y' = \frac{1}{2} - t + \sin(y) - y$ ,  $y(0) = 1$ . Compare it with the rkf45 and backward Euler methods]

**Exercise.** Modify the trapezoidal method in order to obtain the finite difference equation.

$$y_{j+1} = y_j + \frac{k}{2}(3f_j - f_{j-1}), \quad \text{for } j = 1, 2, \dots, M - 1$$

Because this is explicit it is an example of an Adams–Bashforth method. What is the order of the truncation error ? What about the stability of this method ? Apply it to the previous example.

## Summary of More General Methods

In this course we consider three major types of practical numerical methods for solving initial value problems for ODEs:

- Runge-Kutta methods
- Richardson extrapolation and its particular implementation as the Bulirsch-Stoer method.
- Predictor-corrector methods.

A brief description of each of these types follows.

1. **Runge-Kutta methods** propagate a solution over an interval by combining the information from several Euler-style steps (each involving one evaluation of the right-hand  $f$ 's), and then using the information obtained to match a Taylor series expansion up to some higher order.
2. **Richardson extrapolation** uses the powerful idea of extrapolating a computed result to the value that would have been obtained if the stepsize had been very much smaller than it actually was. In particular, extrapolation to zero stepsize is the desired goal. The first practical ODE integrator that implemented this idea was developed by Bulirsch and Stoer, and so extrapolation methods are often called Bulirsch-Stoer methods.
3. **Predictor-corrector** methods store the solution along the way, and use those results to extrapolate the solution one step advanced; they then correct the extrapolation using derivative information at the new point. These are best for very smooth functions.

Runge-Kutta is what you use when (i) you don't know any better, or (ii) you have an intransigent problem where Bulirsch-Stoer is failing, or (iii) you have a trivial problem where computational efficiency is of no concern. Runge-Kutta succeeds virtually always; but it is not usually fastest, except when evaluating  $f_i$  is cheap and moderate accuracy ( $\leq 10^{-5}$ ) is required.

Predictor-corrector methods, since they use past information, are somewhat more difficult to start up, but, for many smooth problems, they are computationally more efficient than Runge-Kutta. In recent years

Bulirsch-Stoer has been replacing predictor-corrector in many applications, but it is too soon to say that predictor-corrector is dominated in all cases. However, it appears that only rather sophisticated predictor-corrector routines are competitive. Practice shows that relatively simple Runge-Kutta and Bulirsch-Stoer routines are adequate for most problems.

Each of the three types of methods can be organized to monitor internal consistency. This allows numerical errors which are inevitably introduced into the solution to be controlled by automatic, (adaptive) changing of the fundamental stepsize. It is always recommended that adaptive stepsize control be implemented, and we will do so below.

In general, all three types of methods can be applied to any initial value problem. Each comes with its own set of debits and credits that must be understood before it is used.

## ▼ Runge-Kutta Methods

Until now, Euler method was used to illustrate many concepts in the theory of numerical ODEs. However, as illustrated in the previous sections, there are several reasons that Euler's method is not recommended for practical use, among them, (i) the method is not very accurate when compared to other, fancier, methods run at the equivalent stepsize, and (ii) neither is it very stable.

An extraordinarily successful family of numerical approximations for IVPs comes under the general

classification of Runge–Kutta (RK) methods. The derivation is based on the question of whether it is possible to determine an **explicit** method for finding  $y_{j+1}$  that only uses the value of the solution at  $t_j$  and has a predetermined truncation error.

### The idea of Runge-Kutta methods (Peter Stone)

The idea of Runge-Kutta methods for solving differential equations of the form  $\frac{dy}{dx} = f(x, y)$  numerically is to sample the function at several points and use the resulting information to calculate  $y_{k+1}$  from  $y_k$  as accurately as possible when  $x$  increments from  $x_k$  to  $x_{k+1} = x_k + h$ .

In the special case of a differential equation of the form  $\frac{dy}{dx} = f(x)$  we have

$$y_{k+1} = y_k + \int_{x_k}^{x_k+h} f(x) dx.$$

In this case it is possible to estimate  $y_{k+1}$  by evaluating the integral  $\int_{x_k}^{x_k+h} f(x) dx$  by numerical integration. This involves evaluating  $f(x)$  for various values of  $x$  in the interval from  $x_k$  to  $x_{k+1} = x_k + h$  to give a sequence of values  $f_1, f_2, \dots, f_s$ .

The estimate for  $y_k$  then has the form

$$y_{k+1} = y_k + (b_1 f_1 + b_2 f_2 + \dots + f_s b_s) h.$$

Runge-Kutta method may be regarded as being roughly analogous to the previous process. In this more general case  $f(x, y)$  is evaluated for various values of  $x$  in the interval from  $x_k$  to  $x_{k+1} = x_k + h$  and "suitable" corresponding values of  $y$  to give a sequence of values  $f_1, f_2, \dots, f_s$ .

The estimate for  $y_k$  then has the form

$$y_{k+1} = y_k + (b_1 f_1 + b_2 f_2 + \dots + f_s b_s) h.$$

### Improved Euler Method or Heun Method

Let us start with the Taylor expansion

$$y_{i+1} = y_i + h y' (t_i) + \frac{h^2}{2} y''(t_i) + \frac{h^3}{3!} y'''(\xi_i)$$

where  $x_i < \xi_i < x_{i+1}$ . Since  $y'_i = f(t_i, y(t_i))$ , dropping the truncation error term we have

$$y_{i+1} = y_i + h f(t_i, y(t_i)) + \frac{h^2}{2} f'(t_i, y(t_i)).$$

We now approximate  $f'$  using the forward difference (which has a truncation error  $O(h^2)$ ),

$$f'(t_i, y(t_i)) = \frac{(f(t_{i+1}, y(t_{i+1})) - f(t_i, y(t_i)))}{h}$$

Thus,

$$y_{i+1} = y_i + \frac{h}{2} (f(t_i, y_i) + f(t_{i+1}, y_{i+1})).$$

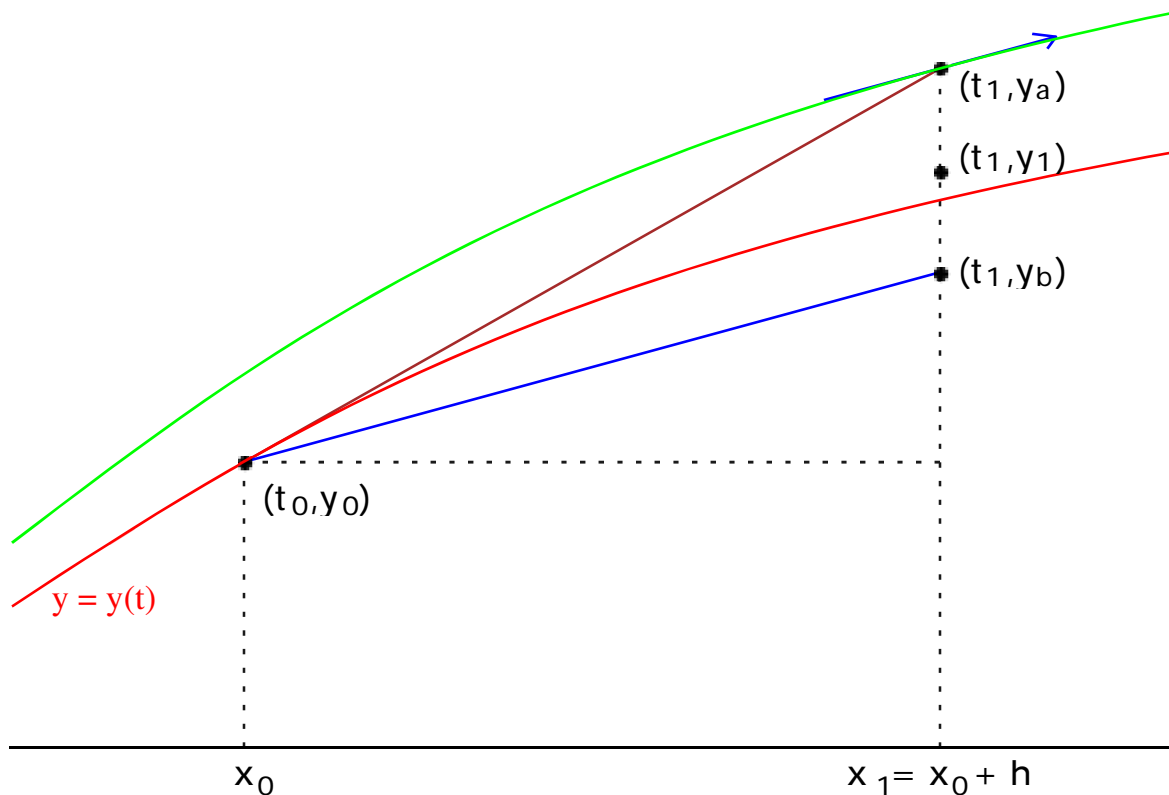
Using  $y_{i+1} = y_i + hf(t_i, y_i)$  we finally get

$$y_{i+1} = y_i + \frac{h}{2} (f(t_i, y_i) + f(t_{i+1}, y_i + hf(t_i, y_i)))$$

There is an interesting geometrical interpretation for this formula. Starting at a point  $(t_0, y_0)$ , we first make an Euler step to obtain a point  $(t_1, y_a)$ .

Then we find the gradient of the tangent to the solution curve through  $(t_1, y_a)$ , and follow a line parallel to this tangent starting at the point  $(t_0, y_0)$  until it meets the vertical line  $t = t_1$ . If this new point is  $(t_1, y_b)$ , we take  $y_1$  to be the average  $\frac{y_a + y_b}{2}$  of the two  $y$  values we have calculated. This gives

the next point  $(t_1, y_1)$  of the discrete numerical solution.



We now compute  $f_1 = f(t_0, y_0)$  followed by  $f_2 = f(t_1, y_0 + f_1 h)$ , where  $t_1 = t_0 + h$ .

Then

$$y_1 = y_0 + \left( \frac{f_1 + f_2}{2} \right) h.$$

**Example.** We consider again the example of the logistic equation:

```
> restart
> with(plots) :
> eq:=diff(y(t),t)=10*y(t)*(1-y(t));
      eq :=  $\frac{d}{dt} y(t) = 10 y(t) (1 - y(t))$  (5.1)
```

```
> exact_sol:=dsolve( {eq, y(0)=0.01}, y(t));
      exact_sol :=  $y(t) = \frac{1}{1 + 99 e^{-10t}}$  (5.2)
```

```
> Yexact:=unapply(op(2,exact_sol),t);
      Yexact :=  $t \rightarrow \frac{1}{1 + 99 e^{-10t}}$  (5.3)
```

```
> f := y -> 10*y*(1-y);
      f :=  $y \rightarrow 10 y (1 - y)$  (5.4)
```

We now compare the errors:

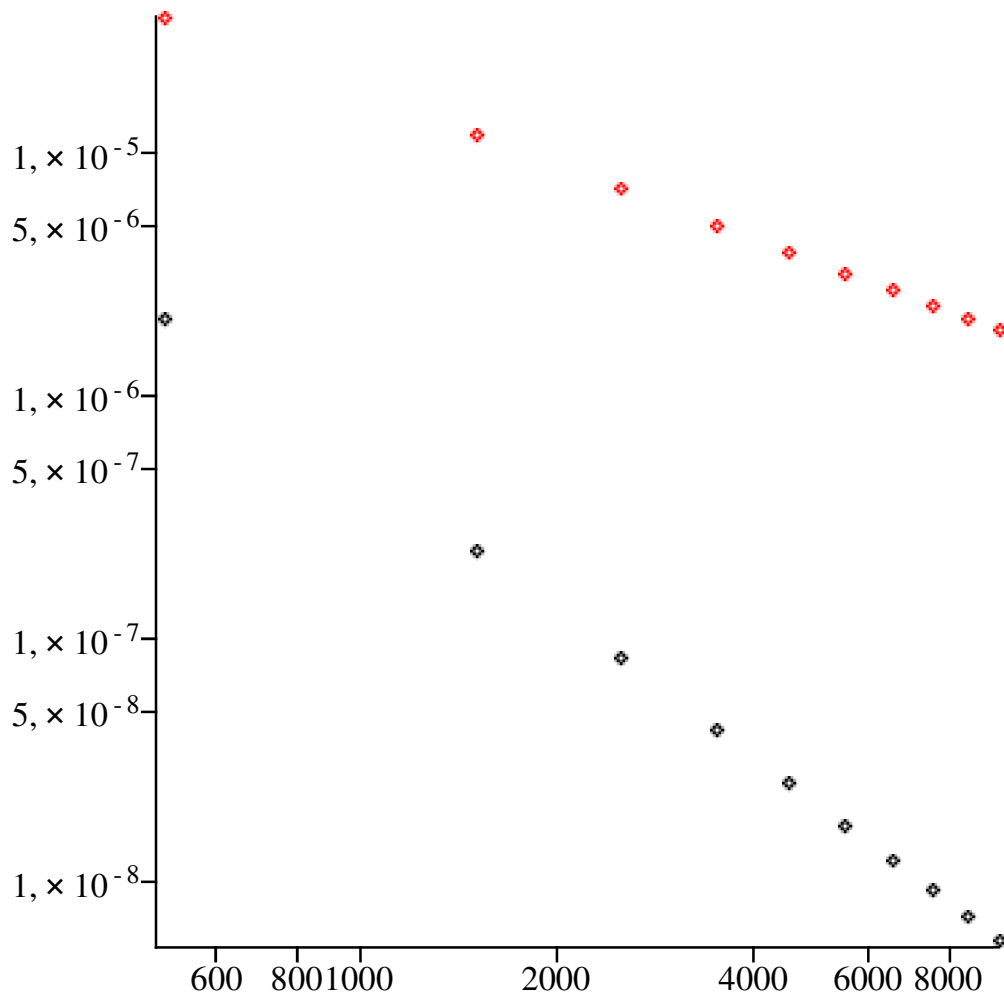
```
> error_euler4 := proc (M)
  local k, j, L, ye;
  k := 1/M; ye[0] := 1/100;
  for j from 0 to M-1 do
    ye[j+1] := evalf[19](ye[j]+10*k*ye[j]*(1-ye[j]))
  end do;
  evalf[30](abs(ye[M]-Yexact(M*k)))
end proc:
```

```
> error_rk2 := proc (M)
  local k, j, L, ye, yrk2,k1,k2;
  Digits:=19;
  k := 1/M;
  ye[0] := 1/100;yrk2[0] := 1/100;
  for j from 0 to M-1 do
    k1:=evalf(k*f(yrk2[j]));
    k2:=evalf(k*f(yrk2[j]+k1));
    yrk2[j+1] := yrk2[j]+evalf((1/2)*(k1+k2));
  od;
  evalf(abs(Yexact(M*k)-yrk2[M]));
end proc:
```

```
> error_rk2(1000);error_euler4(1000);
      5.233913734172 10-7
      0.000017992526604778074406902162 (5.5)
```

```
> P1:=loglogplot([seq([k, error_rk2(k)], k = 500 .. 10000,
  1000)], style = point,color=black):
```

```
> P2 := loglogplot([seq([k, error_euler4(k)], k = 500 .. 10000,
  1000)], style = point, color = red):
> display({P1,P2});
```



```
>
```

This shows the much better accuracy of the rk2 method, relative to Euler's method.

### Example

Let's consider the following IVP:

$$\frac{d}{dx} y(x) + \frac{y(x)}{x} - x = 0, \quad y(1) = 1.$$

The exact solution is:

```
> restart:
```

```
> eq:= diff(y(x),x)+y(x)/x-x^3*cos(x)^2 = 0;
```

$$eq := \frac{d}{dx} y(x) + \frac{y(x)}{x} - x^3 \cos(x)^2 = 0 \tag{5.6}$$

```
> exact_sol:=dsolve( {eq, y(1)=1}, y(x));
```

(5.7)

$$\text{exact\_sol} := y(x) = \frac{1}{40} \frac{1}{x} (10x^4 \sin(2x) + 20x^3 \cos(2x) - 30x^2 \sin(2x) + 15 \sin(2x) - 30x \cos(2x) + 4x^5 + 36 + 5 \sin(2) + 10 \cos(2)) \quad (5.7)$$

Let us solve this problem using the methods of Euler and rk2 in the interval  $I = [1, 10]$ , with  $M=50$ , so  $h = 9/50$ , and compare with the exact solution.

Let us start with Euler method:

```
> M := 50.; X := 10.;
                                     M:= 50.
                                     X:= 10.                                (5.8)
```

```
> h:=(X-1)/M; y[0]:=1.; x[0]:=1.;
                                     h:= 0.1800000000
                                     y0:= 1.
                                     x0:= 1.                                (5.9)
```

```
> f:=-y/x+x^3*cos(x)^2;
                                     f:= -\frac{y}{x} + x^3 \cos(x)^2          (5.10)
```

```
> F:=unapply(f, x, y);
                                     F := (x, y) → -\frac{y}{x} + x^3 \cos(x)^2  (5.11)
```

```
> L1:=[[x[0], y[0]]]:
> for i from 1 to M do
>   y[i]:=y[i-1]+h*F(x[i-1], y[i-1]);
>   x[i]:=x[0]+i*h;
>   L1:=op(L1), [x[i], y[i]];
> od:
```

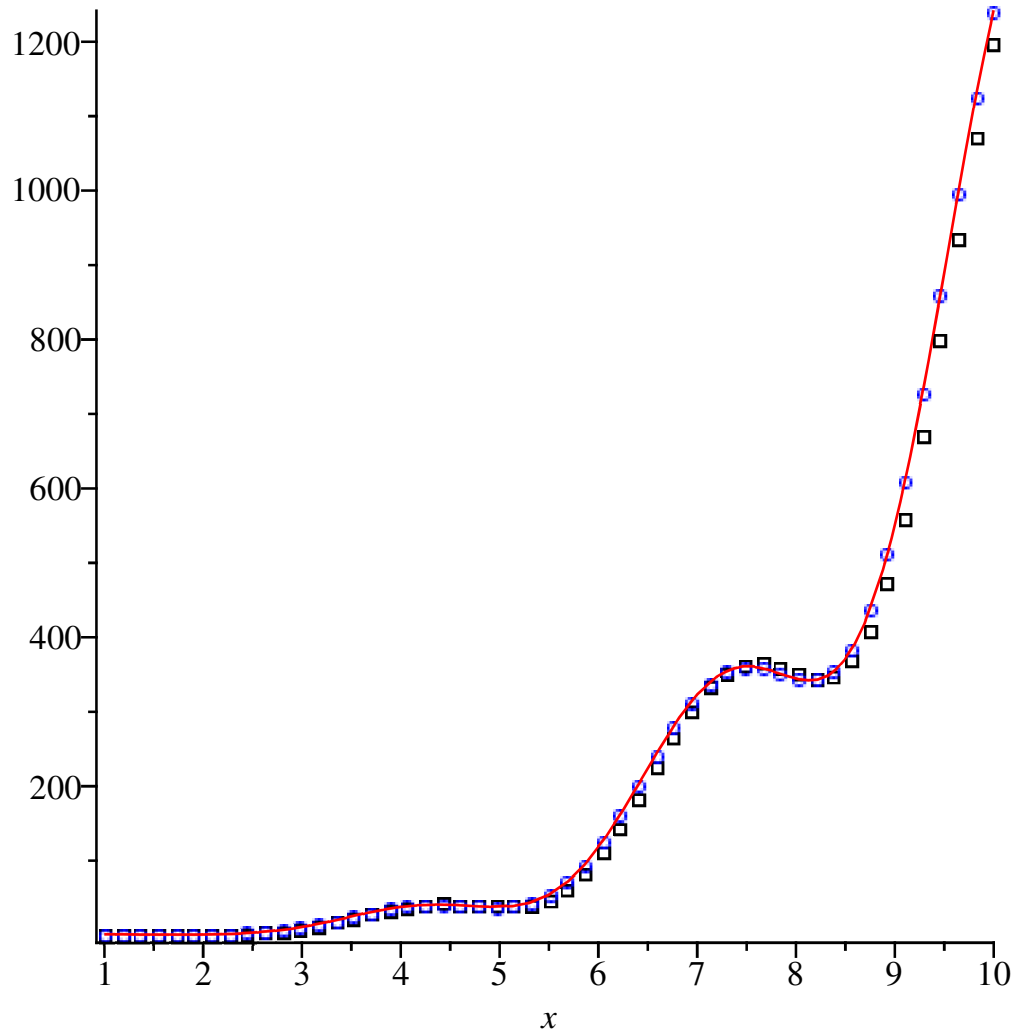
Now we use rk2

```
> L2:=[[x[0], y[0]]]:
> for i from 1 to M do
>   yp[i]:=y[i-1]+h*F(x[i-1], y[i-1]);
>   y[i]:=y[i-1]+(h/2)*(F(x[i], yp[i])+F(x[i-1], y[i-1]));
>   x[i]:=x[0]+i*h;
>   L2:=op(L2), [x[i], y[i]];
> od:
```

We now make the graphs:

```
> with(plots):
> g1 := pointplot(L1, color = black, symbol = box):
> g2 := pointplot(L2, color = blue, symbol = circle):
> yex:=op(2, exact_sol):
> Yex:=unapply(yex, x):
> g3:=plot(yex, x=1..10, color=red):
```

```
> display([g1,g2,g3]);
```



Percentual errors

```
> Error_euler:= evalf(abs((op(2,op(nops(L1),L1))-Yex(10))/Yex(10)*100));
```

*Error\_euler := 3.505242990* (5.12)

```
> Error_rk2:=evalf(abs((op(2,op(nops(L2),L2))-Yex(10))/Yex(10)*100));
```

*Error\_rk2 := 0.1375715724* (5.13)

```
> ;
```

This again shows the relative good accuracy of the method.

**Exercise:** Examine the the error along the the interval [1,10] for both methods. What is your conclusion ?

### General Second Order Runge-Kutta Method

We consider the derivation of coefficient systems for order 2 Runge-Kutta schemes for constructing numerical solutions for the differential equation

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

The plan is to consider the general form

$$y(t+h) \simeq y(t) + (s u_1 + w u_2) h \text{ ----- (i),}$$

where

$$u_1 = f(t, y),$$

and

$$u_2 = f(t_0 + \alpha h, y_0 + \beta u_1 h).$$

Then find values of the coefficients  $s, w, \alpha, \beta$  so that this best approximates  $y(t+h)$  to the 2nd order.

Given

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

we find a 2nd order Taylor approximation to the solution  $y(t)$  and make a list of the coefficients of the various partial derivatives of  $f(t, y(t))$ .

The Taylor series expansion of the solution  $y = y(t)$  about  $t$  is:

$$y(t+h) = y(t) + y'(t) h + y''(t) \frac{h^2}{2!} + \dots \text{ ----- (ii).}$$

Note that  $y'(t)$  can be replaced by  $f(t, y)$ .

We need to find expressions for the 2nd derivative in terms of the gradient field  $f(t, y)$ .

The chain rule for functions of two variables gives:

$$\begin{aligned} \frac{d}{dt} f(t, y) &= \frac{\partial}{\partial t} f(t, y) + \left( \frac{\partial}{\partial y} f(t, y) \right) \left( \frac{dy}{dt} \right) \\ &= \frac{\partial}{\partial t} f(t, y) + \left( \frac{\partial}{\partial y} f(t, y) \right) f(t, y) \\ &= f_t(t, y) + f_y(t, y) f(t, y). \end{aligned}$$

This formula involves the partial derivatives  $\frac{\partial}{\partial t} f(t, y) = f_t(t, y)$  and  $\frac{\partial}{\partial y} f(t, y) = f_y(t, y)$ .

This can be achieved with Maple as follows.

```
> restart
```

```
> fty := f(t, y(t)):
```

```
  fty_ := f(t, y):
```

```
> Diff(fty, t);
```

$$\frac{d}{dt} f(t, y(t)) \tag{5.14}$$

```
> ``=value(%);
```

$$= D_1(f)(t, y(t)) + D_2(f)(t, y(t)) \left( \frac{d}{dt} y(t) \right) \tag{5.15}$$

```
> ``=subs(diff(y(t), t)=fty, rhs(%));
```

$$= D_1(f)(t, y(t)) + D_2(f)(t, y(t)) f(t, y(t)) \quad (5.16)$$

```
> dfty := rhs(%);
```

$$dfty := D_1(f)(t, y(t)) + D_2(f)(t, y(t)) f(t, y(t)) \quad (5.17)$$

Here

$$D_1(f)(t, y(t)) = \frac{\partial}{\partial t} f(t, y) \text{ and } D_2(f)(t, y(t)) = \frac{\partial}{\partial y} f(t, y).$$

Now consider the approximation for  $\frac{y(t+h) - y(t)}{h}$  obtained from the Taylor series (ii) including terms up to the term in  $h^2$ , namely

$$\frac{y(t+h) - y(t)}{h} \simeq y'(t) + y''(t) \frac{h}{2} \text{ ----- (iii).}$$

In Maple we have

```
> fty + dfty*(h/2);
```

```
second_order := expand(%);
```

$$f(t, y(t)) + \frac{1}{2} (D_1(f)(t, y(t)) + D_2(f)(t, y(t)) f(t, y(t))) h$$

$$second\_order := f(t, y(t)) + \frac{1}{2} h D_1(f)(t, y(t)) + \frac{1}{2} h D_2(f)(t, y(t)) f(t, y(t)) \quad (5.18)$$

Next we set up a list whose members are  $f(x, y(x))$  and partial derivatives of  $f(x, y(x))$ .

```
> vars := [f(t, y(t)), D[1](f)(t, y(t)), D[2](f)(t, y(t))];
```

$$vars := [f(t, y(t)), D_1(f)(t, y(t)), D_2(f)(t, y(t))] \quad (5.19)$$

Regarding the expression `second_order` as a polynomial in these variables we can dissect the list into 3 terms with coefficients of the form  $r h^n$ , where  $r$  is rational.

```
> collect(second_order, vars, distributed);
```

$$f(t, y(t)) + \frac{1}{2} h D_1(f)(t, y(t)) + \frac{1}{2} h D_2(f)(t, y(t)) f(t, y(t)) \quad (5.20)$$

```
> C1 := [coffs(%, vars, 'terms')];
```

$$C1 := \left[ 1, \frac{1}{2} h, \frac{1}{2} h \right] \quad (5.21)$$

These coefficients correspond to the terms:

```
> [terms];
```

$$[f(t, y(t)), D_1(f)(t, y(t)), D_2(f)(t, y(t)) f(t, y(t))] \quad (5.22)$$

Now we calculate the corresponding coefficients for the general Runge-Kutta 2nd order scheme.

We have

$$y(t+h) \simeq y(t) + (s u_1 + w u_2) h,$$

where

$$u_1 = f(t, y),$$

$$u_2 = f(t + \alpha h, y + \beta u_1 h).$$

We have

$$\frac{y(t+h) - y(t)}{h} \simeq s u_1 + w u_2 \text{ ----- (iv).}$$

Let

$$v_1 = u_1 = f(t, y),$$

and let  $v_2$  be the 2nd order Taylor series approximation for  $u_2$ .

$v_2$  is obtained by calculating the 2nd order Taylor series of  $f(t+\alpha h, y(t)+\beta v[1]h)$ , and then converting to a polynomial (dropping  $O(h^2)$ ).

$$\begin{aligned} > \mathbf{v[1] := f(t, y(t));} \\ & \qquad \qquad \qquad v_1 := f(t, y(t)) \end{aligned} \tag{5.23}$$

$$\begin{aligned} > \mathbf{v[2] := convert(taylor(f(t+alpha*h, y(t)+beta*v[1]*h), h, 2),} \\ & \mathbf{polynom):} \\ & \mathbf{'v[2]'=v[2];} \\ & \qquad v_2 = f(t, y(t)) + \left( D_1(f)(t, y(t)) \alpha + D_2(f)(t, y(t)) \beta f(t, y(t)) \right) h \end{aligned} \tag{5.24}$$

Now  $\alpha v_1 + \beta v_2$  represents the approximation for  $\alpha u_1 + \beta u_2$  obtained on replacing  $u_2$  by the order 2 Taylor series approximation  $v_2$ .

$$\begin{aligned} > \mathbf{s*v[1]+w*v[2];} \\ & \quad s f(t, y(t)) + w \left( f(t, y(t)) + \left( D_1(f)(t, y(t)) \alpha + D_2(f)(t, y(t)) \beta f(t, y(t)) \right) h \right) \end{aligned} \tag{5.25}$$

$$\begin{aligned} > \mathbf{collect(%, vars, 'distributed');} \\ & \quad w \beta h D_2(f)(t, y(t)) f(t, y(t)) + (s+w) f(t, y(t)) + w D_1(f)(t, y(t)) \alpha h \end{aligned} \tag{5.26}$$

$$\begin{aligned} > \mathbf{C2 := [coeffs(%, vars, 'terms2')];} \\ & \quad C2 := [s+w, w \alpha h, w \beta h] \end{aligned} \tag{5.27}$$

Referring to the approximations (iii) and (iv) for  $\frac{y(t+h) - y(t)}{h}$ , we now equate these coefficients with those found previously and solve.

Note that there are 5 variables (4 Runge-Kutta coefficients and  $h$ ), and 3 equations.

$$\begin{aligned} > \mathbf{eqns := \{seq(C1[i]=C2[i], i=1..nops(C1))\};} \\ & \quad eqns := \left\{ 1 = s + w, \frac{1}{2} h = w \alpha h, \frac{1}{2} h = w \beta h \right\} \end{aligned} \tag{5.28}$$

$$\begin{aligned} > \mathbf{solns := [solve(eqns)];} \\ & \quad solns := \left[ \left\{ h=0, s=1-w, w=w, \alpha=\alpha, \beta=\beta \right\}, \left\{ h=h, s=1-w, w=w, \alpha=\frac{1}{2w}, \beta \right. \right. \\ & \quad \left. \left. = \frac{1}{2w} \right\} \right] \end{aligned} \tag{5.29}$$

Ruling out the trivial solution with  $h=0$ , we have only the one non-trivial solution:

$$s + w = 1, \quad \alpha = \beta$$

$$= \frac{1}{2w}.$$

Let us consider three important solutions:

$$1. s = w = \frac{1}{2}, \quad \alpha = \beta = 1.$$

Here the equations

$$\begin{aligned} y(t+h) &= y(t) + (s u_1 + w u_2) h, \\ u_1 &= f(t, y), \\ u_2 &= f\left(t + \alpha h, y + \beta u_1 h\right). \end{aligned}$$

become

$$y(t+h) \simeq y(t) + \left( \frac{1}{2} f(t, y) + \frac{1}{2} f\left(t+h, y + f(t, y)h\right) \right) h,$$

This is the **improved Euler method** or **Heun method**, seen previously.

$$2. s = 0, \quad w = 1, \quad \alpha = \beta = \frac{1}{2}$$

Now we have

$$y(t+h) = y(t) + f\left(t + \frac{1}{2}h, y + \frac{1}{2}f(t, y)h\right) h,$$

which is called the **Modified Euler method**, or **classical Runge-Kutta mid-point method**

$$3. s = \frac{1}{4}, \quad w = \frac{3}{4}, \quad \alpha = \beta = \frac{2}{3}$$

Now

$$\begin{aligned} y(t+h) &= y(t) + \left( \frac{1}{4} u_1 + \frac{3}{4} u_2 \right) h, \\ u_1 &= f(t, y), \\ u_2 &= f\left(t + \frac{2}{3}h, y + \frac{2}{3}u_1 h\right). \end{aligned}$$

or

$$y(t+h) = y(t) + \left( \frac{1}{4} f(t, y) + \frac{3}{4} f\left(t + \frac{2}{3}h, y + \frac{2}{3}f(t, y)h\right) \right) h,$$

This is called the **two-thirds method**. It can be shown that the bound error is minimum in this case (V. A. Patel, Numerical Analysis, Saunders College Pub., p. 248). A similar derivation can be found in <http://www.solvingproblems.ethz.ch/downloads.html#19>

**Exercise.** Solve the IVP

$$\frac{dy}{dt} = -ty^2, \quad y(0) = 2, \quad 0 \leq t \leq 1.$$

using the three methods described above and compare with the exact solution.

**Exercise.** Consider the differential equation

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

with the initial condition  $y(0) = 1$ .

(a) Find an analytical solution for the differential equation with the given initial condition.

(b) Find a discrete numerical solution for the differential equation with the given initial condition

over the interval from  $t = 0$  to  $t = 5$  using the two-third method method with 20 steps.  
(c) Plot a graph to compare the discrete numerical solution with the analytical solution.

### Classical Runge-Kutta order 4 method (Peter Stone)

The following is an alternative description in terms of the four values  $f_1, f_2, f_3$  and  $f_4$  of the slope field  $f(t, y)$  needed to progress from a point  $(t_k, y_k)$  on the approximate solution curve, to a new point  $(t_{k+1}, y_{k+1})$ , with  $t_{k+1} = t_k + h$ .  
Compute successively:

$$f_1 = f(t_k, y_k),$$

$$f_2 = f\left(t_k + \frac{h}{2}, y_k + f_1\left(\frac{h}{2}\right)\right),$$

$$f_3 = f\left(t_k + \frac{h}{2}, y_k + f_2\left(\frac{h}{2}\right)\right),$$

and

$$f_4 = f(t_k + h, y_k + f_3 h).$$

Then

$$y_{k+1} = y_k + \left(\frac{f_1 + 2f_2 + 2f_3 + f_4}{6}\right) h.$$

### The classical Runge-Kutta order 3 and 4 methods via **dsolve**

We can repeat the example considered previously where the classical Runge Kutta order 4 method is used to obtain an approximate numerical solution for the differential equation

$$\frac{dy}{dx} = x \cdot \sin(y(x)),$$

over the interval from  $x = 0$  to  $x = 1$  with the initial condition  $y(0) = 1$ , and step-size  $h = \frac{1}{20}$ .

Note: The option "method=classical[rk3]" can be inserted in place of "method=classical[rk4]" to obtain a solution by the classical Runge-Kutta order 3 method.

```
> de := diff(y(x), x) = x * sin(y(x));
ic := y(0) = 1;
left := 0;
right := 1.;
numsteps := 20;
h := (right - left) / numsteps;
dsolve({de, ic}, numeric, method = classical[rk4],
        output = array([seq(h * i, i = 0..numsteps)]), stepsize = h);
soln := convert(map(evalf, %[2, 1]), listlist);
```

$$de := \frac{d}{dx} y(x) = x \sin(y(x))$$

```

        ic := y(0) = 1
        left := 0
        right := 1.
        numsteps := 20
        h := 0.050000000000

soln := [[0., 1.], [0.050000000000, 1.001052194], [0.100000000000, 1.004213031],
        [0.150000000000, 1.009495235], [0.200000000000, 1.016919872], [0.250000000000,
        1.026516123], [0.300000000000, 1.038320952], [0.350000000000, 1.052378634],
        [0.400000000000, 1.068740126], [0.450000000000, 1.087462246], [0.500000000000,
        1.108606603], [0.550000000000, 1.132238261], [0.600000000000, 1.158424050],
        [0.650000000000, 1.187230512], [0.700000000000, 1.218721396], [0.750000000000,
        1.252954688], [0.800000000000, 1.289979126], [0.850000000000, 1.329830225],
        [0.900000000000, 1.372525809], [0.950000000000, 1.418061171], [1.000000000000,
        1.466403970]]

```

**(5.30)**

The points of this discrete numerical solution can be plotted along with the curve for the analytical solution

```

> dsolve({de, y(0) = 1}, y(x));

```

$$y(x) = \arctan \left( \frac{2 e^{\frac{1}{2} x^2} \sin(1) (-1 + \cos(1))}{-\sin(1)^2 - 2 \left(e^{\frac{1}{2} x^2}\right)^2 + 2 \left(e^{\frac{1}{2} x^2}\right)^2 \cos(1) + \left(e^{\frac{1}{2} x^2}\right)^2 \sin(1)^2} \right), \quad (5.31)$$

$$- \frac{2 \left(e^{\frac{1}{2} x^2}\right)^2 \cos(1) + \sin(1)^2 + \left(e^{\frac{1}{2} x^2}\right)^2 \sin(1)^2 - 2 \left(e^{\frac{1}{2} x^2}\right)^2}{-\sin(1)^2 - 2 \left(e^{\frac{1}{2} x^2}\right)^2 + 2 \left(e^{\frac{1}{2} x^2}\right)^2 \cos(1) + \left(e^{\frac{1}{2} x^2}\right)^2 \sin(1)^2}$$

```

> exact_sol := simplify(rhs(%));

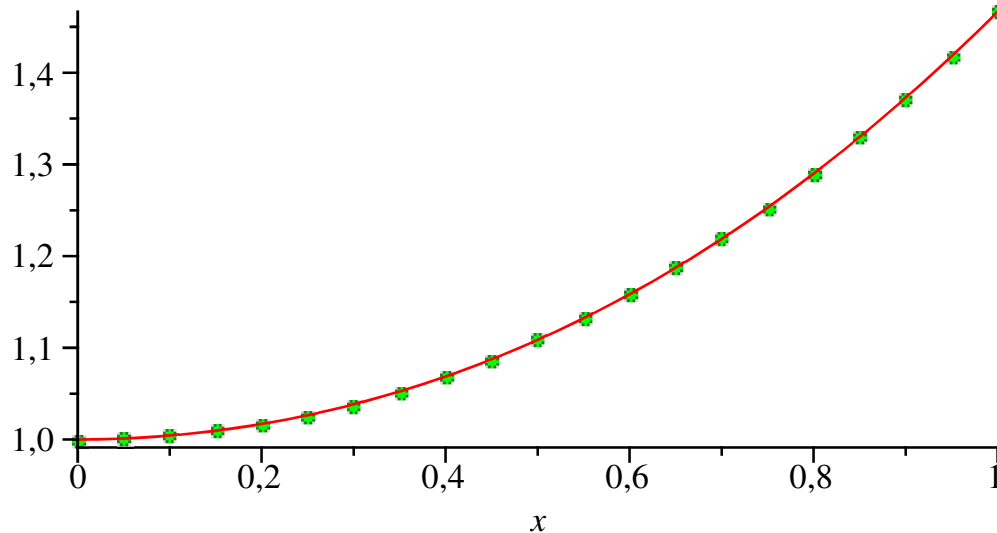
```

$$exact\_sol := \arctan \left( - \frac{2 e^{\frac{1}{2} x^2} \sin(1)}{-\cos(1) + e^{x^2} \cos(1) - e^{x^2} - 1}, - \frac{\cos(1) + e^{x^2} \cos(1) - e^{x^2} + 1}{-\cos(1) + e^{x^2} \cos(1) - e^{x^2} - 1} \right) \quad (5.32)$$

```

> plot([soln$3, exact_sol], x=0..1, style=[point$3, line$2],
color=[black, green$2, red], symbol=[circle, diamond, cross],
scaling=constrained);

```



`>` ;

**Exercise.** Implement the classical RK4 method algorithm and solve the IVP

$$\frac{dy}{dt} = -ty^2, \quad y(0) = 2, \quad 0 \leq t \leq 1.$$

using the three methods described above and compare with the exact solution. Compare the result with the solution obtained with Maple's `dsolve` command with `classical[rk4]`.

## ▼ Taylor Series Method

### ▼ First order ode's (*taylor1*)

Let us consider the IVP

$$y' = f(t, y), \quad a < t, \quad y(a) = y(0) = y_0.$$

Later we shall generalize the method for higher order ODE's.

Let us take, for example,  $f(t, y) = \sin(t) y$ . We shall derive this function many times.

`> restart;`

`> f:=sin(t)*y(t);`

`f:=sin(t) y(t)`

**(6.1.1)**

Now we can introduce a two-variable function  $G$ , which will be used later to determine the points of the solution.

$$\begin{aligned} > \mathbf{F := subs (y (t) = Y, f) ;} \\ & \qquad \qquad \qquad F := \sin(t) Y \end{aligned} \tag{6.1.2}$$

$$\begin{aligned} > \mathbf{G := unapply (F, t, Y) ;} \\ & \qquad \qquad \qquad G := (t, Y) \rightarrow \sin(t) Y \end{aligned} \tag{6.1.3}$$

The next derivative will be  $y''$ . We repeat the above procedures to obtain

$$\begin{aligned} > \mathbf{dy2 := diff (f, t) ;} \\ & \qquad \qquad \qquad dy2 := \cos(t) y(t) + \sin(t) \left( \frac{d}{dt} y(t) \right) \end{aligned} \tag{6.1.4}$$

$$\begin{aligned} > \mathbf{dy2 := subs (diff (y (t), t) = f, dy2) ;} \\ & \qquad \qquad \qquad dy2 := \cos(t) y(t) + \sin(t)^2 y(t) \end{aligned} \tag{6.1.5}$$

$$\begin{aligned} > \mathbf{Dy2 := subs (y (t) = Y, dy2) ;} \\ & \qquad \qquad \qquad Dy2 := \cos(t) Y + \sin(t)^2 Y \end{aligned} \tag{6.1.6}$$

$$\begin{aligned} > \mathbf{DY2 := unapply (Dy2, t, Y) ;} \\ & \qquad \qquad \qquad DY2 := (t, Y) \rightarrow \cos(t) Y + \sin(t)^2 Y \end{aligned} \tag{6.1.7}$$

$$\begin{aligned} > \mathbf{dy3 := diff (dy2, t) ;} \\ & \qquad \qquad \qquad dy3 := -\sin(t) y(t) + \cos(t) \left( \frac{d}{dt} y(t) \right) + 2 \sin(t) y(t) \cos(t) + \sin(t)^2 \left( \frac{d}{dt} y(t) \right) \end{aligned} \tag{6.1.8}$$

$$\begin{aligned} > \mathbf{dy3 := subs (diff (y (t), t) = f, dy3) ;} \\ & \qquad \qquad \qquad dy3 := -\sin(t) y(t) + 3 \sin(t) y(t) \cos(t) + \sin(t)^3 y(t) \end{aligned} \tag{6.1.9}$$

$$\begin{aligned} > \mathbf{Dy3 := subs (y (t) = Y, dy3) ;} \\ & \qquad \qquad \qquad Dy3 := -\sin(t) Y + 3 \sin(t) Y \cos(t) + \sin(t)^3 Y \end{aligned} \tag{6.1.10}$$

$$\begin{aligned} > \mathbf{DY3 := unapply (Dy3, t, Y) ;} \\ & \qquad \qquad \qquad DY3 := (t, Y) \rightarrow -\sin(t) Y + 3 \sin(t) Y \cos(t) + \sin(t)^3 Y \end{aligned} \tag{6.1.11}$$

Let us take the step  $h = 0.05$  with initial condition  $y(1) = 2$  :

**> h:=0.05:**

**> T[1]:=1:Y[1]:=2:**

The iterative process is now given by

```
> L:=[[T[1],Y[1]]]:
> for i to 100 do
>   Y[i+1]:=evalf(Y[i]+h*G(T[i],Y[i])+h^2*DY2(T[i],Y[i])/2+
h^3*DY3(T[i],Y[i])/6);
T[i+1]:=T[i]+h:
L:=[op(L),[T[i+1],Y[i+1]]]
> od:
> L:
```

Let us generate a plot of the result:

**> with (plots) :**

**> d1:=pointplot (L) :**

In order to verify the accuracy of our result, let us compare it with the exact solution:

```
> eq:=diff(y(tt),tt)-sin(tt)*y(tt);
```

$$eq := \frac{d}{dt} y(tt) - \sin(tt) y(tt) \quad (6.1.12)$$

```
> ss:=op(2,dsolve({eq,y(1)=2},y(tt)));
```

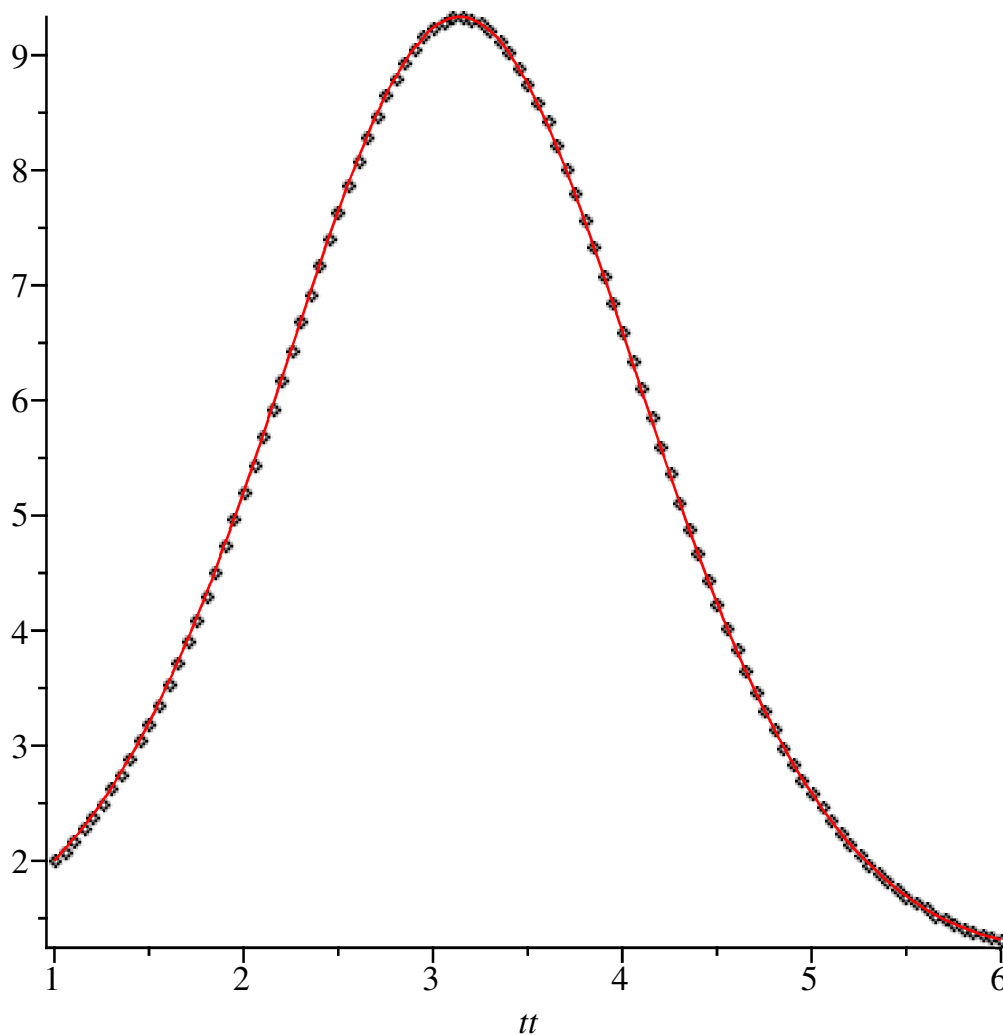
$$ss := \frac{2 e^{-\cos(tt)}}{e^{-\cos(1)}} \quad (6.1.13)$$

```
> ssf:=unapply(ss,tt);
```

$$ssf := tt \rightarrow \frac{2 e^{-\cos(tt)}}{e^{-\cos(1)}} \quad (6.1.14)$$

```
> d2:=plot(ss(tt),tt=1..6):
```

```
> display([d1,d2]);
```



The biggest percentual error in the grid is:

```
> erro3:=seq(abs(evalf((Y[j]-ssf(T[j]))*100/ssf(T[j]))),j=1.  
.100):
```

```
> max(erro3);
```

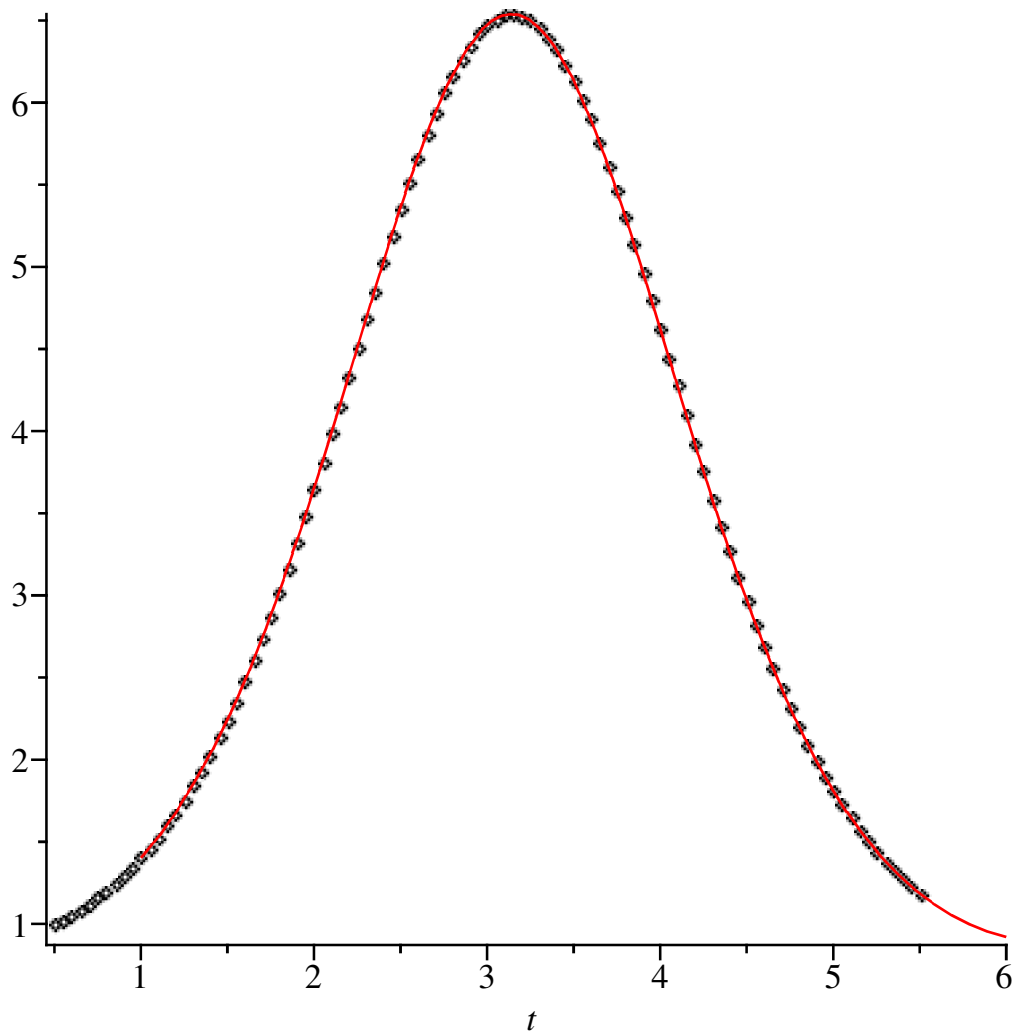
(6.1.15)

We summarize the above commands in a procedure

```

> restart
> with(plots) :
> taylor1:=proc(f,n,h,t0,y0)
  local dy,DY,Dy,i,dys,F,T,Y,L,G,m,F2,aa,d1,eq,ss,ssf,d2,
  erro3;
  F:=f;
  F2:=subs(y(t)=Y,f(t));
  G:=unapply(F2,t,Y);
  for i from 2 to n do
    dy[i]:=diff(F,t);
    dys[i]:=subs(diff(y(t),t)=f,dy[i]);
    Dy[i]:=subs(y(t)=Y,dys[i]);
    DY[i]:=unapply(Dy[i],t,Y);
    F:=dys[i];
  od;
  T[1]:=t0:Y[1]:=y0:
  L:=[[T[1],Y[1]]]:
  for i to 100 do
    aa:=evalf(add(h^m*DY[m](T[i],Y[i])/(m!),m=2..n));
    Y[i+1]:=evalf(Y[i]+h*G(T[i],Y[i])+aa);
    T[i+1]:=T[i]+h:
    L:=[op(L),[T[i+1],Y[i+1]]];
  od:
  d1:=pointplot(L);
  eq:=diff(y(t),t)-f;
  ss:=op(2,dsolve({eq,y(t0)=y0},y(t)));
  ssf:=unapply(ss,t):
  d2:=plot(ss(tt),t=1..6):
> erro3:=seq(abs(evalf((Y[j]-ssf(T[j]))*100/ssf(T[j]))),j=
1..100):
  return [max(erro3),display([d1,d2])];
> end:
> Digits:=20:
> with(plots) :
> f:=sin(t)*y(t);
                                     f:= sin(t) y(t)
                                     (6.1.16)
> taylor1(f,8,0.05,0.5,1)[1];
                                     4.7580247315366462856 10-12
                                     (6.1.17)
> taylor1(f,6,0.05,0.5,1)[2];

```



## Second order ode's (taylor2)

Consider the second order IVP

$$y'' = f(x, y, y'), \quad y(x_0) = y_0, \quad y'(x_0) = y_0'.$$

We can rewrite this equation as a system of two coupled first order DE. Defining

$$u = \begin{bmatrix} y \\ v \end{bmatrix}, \quad F := \begin{bmatrix} v \\ f \end{bmatrix}, \quad u_0 := \begin{bmatrix} y_0 \\ y_0' \end{bmatrix}$$

we have

$$u' = F, \quad u(x_0) = u_0.$$

The previous method can now be applied to  $y$  e  $v$  simultaneously. Let us consider, for example, in the interval  $[1,2]$ , the IVP:

$$y'' = y - x^2, \quad y(1) = 1, \quad y'(1) = 1.$$

Such problem has an exact solution, so we can measure the performance of the method in terms of its accuracy. We define  $v = y'$  e  $y'' = v' = f$ . We write a sequence of commands which list and plot the numerical and exact solution solution, and calculate the maximum error in the grid.

$$\begin{aligned} > \mathbf{f} := \mathbf{y}(\mathbf{x}) - \mathbf{x}^2; \\ & f := y(x) - x^2 \end{aligned} \tag{6.2.1}$$

$$\begin{aligned} > \mathbf{F} := \text{subs}(\mathbf{y}(\mathbf{x}) = \mathbf{Y}, \mathbf{f}); \\ & F := Y - x^2 \end{aligned} \tag{6.2.2}$$

$$\begin{aligned} > \mathbf{F0} := [\mathbf{v}(\mathbf{x}), \mathbf{f}]; \\ & F0 := [v(x), y(x) - x^2] \end{aligned} \tag{6.2.3}$$

$$\begin{aligned} > \mathbf{FFf0} := \text{unapply}(\text{subs}(\{\mathbf{v}(\mathbf{x}) = \mathbf{V}, \mathbf{y}(\mathbf{x}) = \mathbf{Y}\}, \mathbf{F0}), \mathbf{x}, \mathbf{Y}, \mathbf{V}); \\ & FFf0 := (x, Y, V) \rightarrow [V, Y - x^2] \end{aligned} \tag{6.2.4}$$

$$\begin{aligned} > \mathbf{F1} := \text{diff}(\mathbf{F0}, \mathbf{x}); \\ & F1 := \left[ \frac{d}{dx} v(x), \frac{d}{dx} y(x) - 2x \right] \end{aligned} \tag{6.2.5}$$

$$\begin{aligned} > \mathbf{FF1} := \text{subs}(\{\text{diff}(\mathbf{v}(\mathbf{x}), \mathbf{x}) = \mathbf{f}, \text{diff}(\mathbf{y}(\mathbf{x}), \mathbf{x}) = \mathbf{v}(\mathbf{x})\}, \mathbf{F1}); \\ & FF1 := [y(x) - x^2, v(x) - 2x] \end{aligned} \tag{6.2.6}$$

$$\begin{aligned} > \mathbf{FFf1} := \text{unapply}(\text{subs}(\{\mathbf{y}(\mathbf{x}) = \mathbf{Y}, \mathbf{v}(\mathbf{x}) = \mathbf{V}\}, \mathbf{FF1}), \mathbf{x}, \mathbf{Y}, \mathbf{V}); \\ & FFf1 := (x, Y, V) \rightarrow [Y - x^2, V - 2x] \end{aligned} \tag{6.2.7}$$

$$\begin{aligned} > \mathbf{F2} := \text{diff}(\mathbf{FF1}, \mathbf{x}); \\ & F2 := \left[ \frac{d}{dx} y(x) - 2x, \frac{d}{dx} v(x) - 2 \right] \end{aligned} \tag{6.2.8}$$

$$\begin{aligned} > \mathbf{FF2} := \text{subs}(\{\text{diff}(\mathbf{v}(\mathbf{x}), \mathbf{x}) = \mathbf{f}, \text{diff}(\mathbf{y}(\mathbf{x}), \mathbf{x}) = \mathbf{v}(\mathbf{x})\}, \mathbf{F2}); \\ & FF2 := [v(x) - 2x, y(x) - x^2 - 2] \end{aligned} \tag{6.2.9}$$

$$\begin{aligned} > \mathbf{FFf2} := \text{unapply}(\text{subs}(\{\mathbf{y}(\mathbf{x}) = \mathbf{Y}, \mathbf{v}(\mathbf{x}) = \mathbf{V}\}, \mathbf{FF2}), \mathbf{x}, \mathbf{Y}, \mathbf{V}); \\ & FFf2 := (x, Y, V) \rightarrow [V - 2x, Y - x^2 - 2] \end{aligned} \tag{6.2.10}$$

$$\begin{aligned} > \mathbf{x0} := 1; \mathbf{xn} := 2; \mathbf{y0} := 1; \mathbf{dy0} := 1; \\ & x0 := 1 \\ & xn := 2 \\ & y0 := 1 \\ & dy0 := 1 \end{aligned} \tag{6.2.11}$$

$$\begin{aligned} > \mathbf{h} := 0.1; \\ & h := 0.1 \end{aligned} \tag{6.2.12}$$

$$\begin{aligned} > \mathbf{n} := \text{trunc}((\mathbf{xn} - \mathbf{x0}) / \mathbf{h}); \\ & n := 10 \end{aligned} \tag{6.2.13}$$

$$\begin{aligned} > \mathbf{Y[1]} := \mathbf{x0}; \mathbf{X[1]} := \mathbf{y0}; \mathbf{V[1]} := \mathbf{dy0}; \\ & Y_1 := 1 \\ & X_1 := 1 \\ & V_1 := 1 \end{aligned} \tag{6.2.14}$$

```
> with(plots) :
> for i from 1 to n do
    Y[i+1] := Y[i] + h * FFf0(X[i], Y[i], V[i])[1] + (h^2/2) * FFf1(X
```

```

[i],Y[i],V[i])[1]+
  (h^3/3!)*FFf2(X[i],Y[i],V[i])[1];
V[i+1]:=V[i]+h*FFf0(X[i],Y[i],V[i])[2]+(h^2/2)*FFf1(X
[i],Y[i],V[i])[2]+
  (h^3/3!)*FFf2(X[i],Y[i],V[i])[2];
X[i+1]:=X[i]+h;
od:
L:= [seq([X[k],Y[k]],k=1..n+1)]:
eq:=diff(y(x),x$2)=y(x)-x^2:
ss:=op(2,dsolve({eq,y(X[1])=Y[1],D(y)(X[1])=V[1]},y(x)));
ssf:=unapply(ss,x):
erro:=seq(abs(evalf((Y[j]-ssf(X[j]))*100/ssf(X[j]))),j=2..
n+1);
max(erro):
d1:=pointplot(L):
d2:=plot(ssf(x),x=X[1]..X[n+1]):
display([d1,d2]);

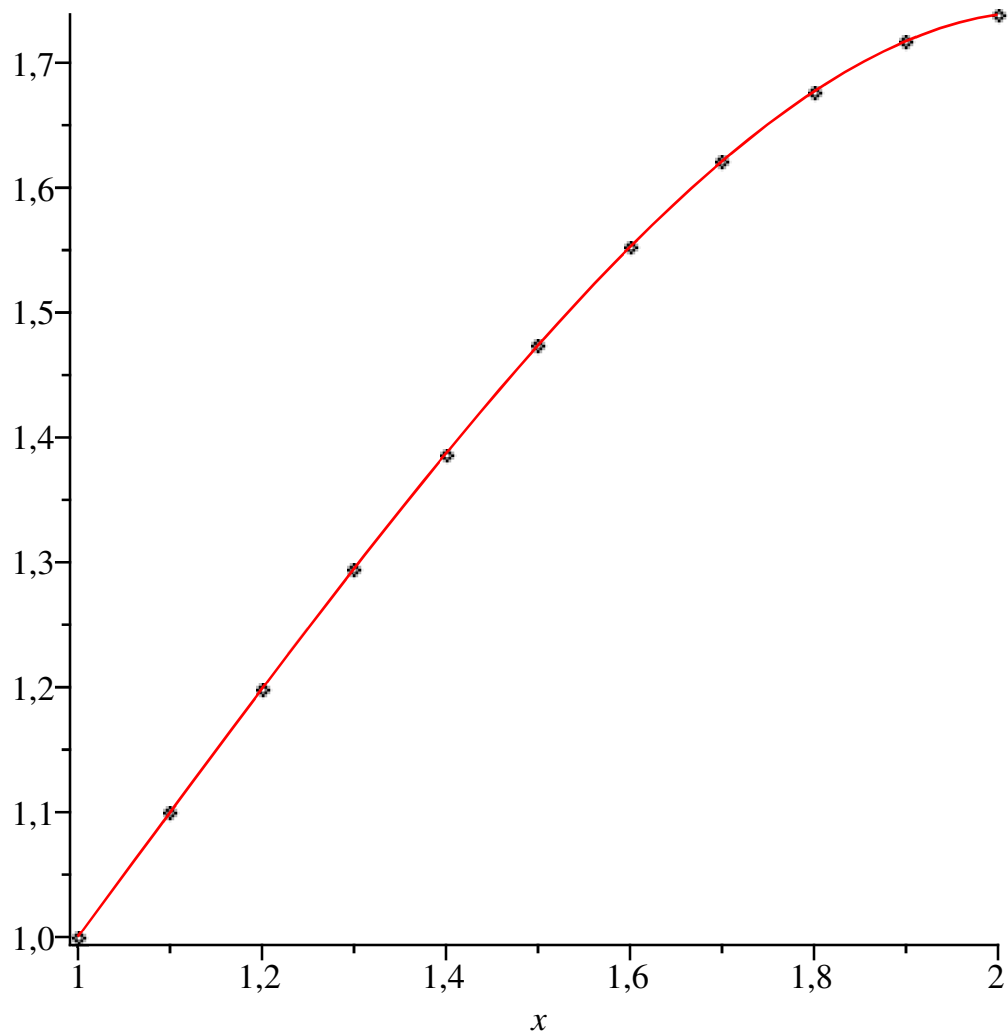
```

$$ss := -\frac{3}{2} \frac{e^x}{e} - \frac{1}{2} \frac{e^{-x}}{e^{-1}} + 2 + x^2$$

```

erro := 0.00076552773795078400816, 0.0014844475987889838122,
0.0021920997984031382525, 0.0029186453904266124940,
0.0036923483990586580305, 0.0045423583737813892493,
0.0055015249649220587763, 0.0066097759870520527325,
0.0079187669831457931966, 0.0094989539089637736838

```



Let us now build a procedure

```

> taylor2:=proc (f, n, h, x0, y0, dy0, xn)
  local i, Y, X, V, FF, N, F, FFf, k, F0, L, FFf0, F1, FF1;
  Y[1]:=x0;X[1]:=y0;V[1]:=dy0;
  N:=trunc((xn-x0)/h);
  > F:=subs(y(x)=Y, f);
  > F0:=[v(x), f];
  > FFf0:=unapply(subs({v(x)=V, y(x)=Y}, F0), x, Y, V);
  > F1:=diff(F0, x);
  > FF1:=subs({diff(v(x), x)=f, diff(y(x), x)=v(x)}, F1);
  > FFf[1]:=unapply(subs({y(x)=Y, v(x)=V}, FF1), x, Y, V);
  for i from 1 to n-1 do
    F[i+1]:=diff(FF[i], x);
  > FF[i+1]:=subs({diff(v(x), x)=f, diff(y(x), x)=v(x)}, F
[i+1]);
  > FFf[i+1]:=unapply(subs({y(x)=Y, v(x)=V}, FF[i+1]), x, Y,
V);

```

```

od;
for k from 1 to N do
  Y[k+1]:=evalf(Y[k]+h*FFf0(X[k],Y[k],V[k])[1]
+add(h^j/(j!)*FFf[j-1](X[k],Y[k],V[k])[1],j=2..n));
  V[k+1]:=evalf(V[k]+h*FFf0(X[k],Y[k],V[k])[2]
+add(h^j/(j!)*FFf[j-1](X[k],Y[k],V[k])[2],j=2..n));
  X[k+1]:=X[k]+h;
od:
L:=[seq([X[k],Y[k]],k=1..N+1)]:
pointplot(L);
end:

```

```
> with(plots):
```

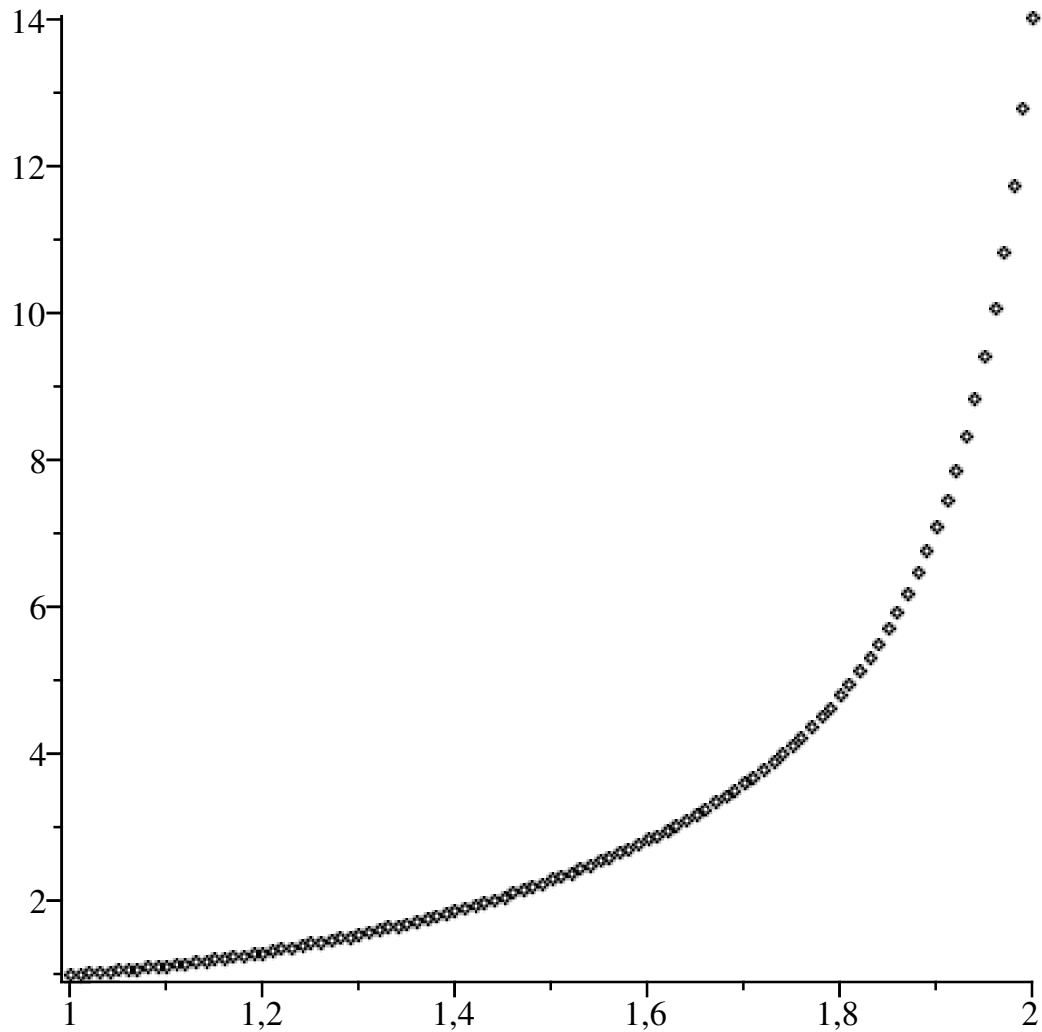
```
> n:=5:
```

```
> f:=y(x)^3-x^3+6*sin(y(x));
```

$$f:=y(x)^3 - x^3 + 6 \sin(y(x))$$

(6.2.15)

```
> taylor2(f,n,0.01,1,1,1,2);
```



**Exercises.**

1. Compare the Taylor series method with  $O(h^4)$  truncation error method with RK4 for a second order IVP. Show an error analysis.
2. Write a Taylor series procedure method for a system with an arbitrary number of equations (defined by the user).