10 Lecture #7: Tuesday, March 10th, 2026

10.1 Solving ODEs numerically

We consider a first-order differential equation $y' = f(x, y)$ with an initial condition $y(x_0) = y_0$. In previous discussions our goal was to understand the qualitative behavior of the solutions or to derive a general analytical solution through separation or variation methods. Now that we have some background on Numerical Analysis, we adopt a different perspective as we focus on *numerical* solution methods.

Rather than studying all possible solutions, we focus on a single solution determined by an initial condition. We are given a point (x_0, y_0) in the plane and look for the solution that passes through this point. If the differential equation satisfies appropriate assumptions (for example, those ensuring existence and uniqueness, such as the hypotheses of the Picard theorem - see Theorem 2.10), then such a solution exists. In favorable cases, the differential equation can be solved analytically and an explicit formula for the solution can be obtained. However, as we have discussed before, in many practically relevant situations this is impossible, and we must turn to Numerical Analysis. The central topic is therefore the following: given an initial value problem, we wish to construct a numerical method that approximates its solution. The exact solution of an initial value problem is a function. Numerical methods, on the other hand, produce only a finite set of numbers. Consequently, we must reformulate our goal.

First, we restrict our attention to an interval of the form $[x_0, x_0 + T]$, where the initial point x_0 is naturally chosen as the starting point of the analysis (see Figure 50). In many applications, the behavior prior to x_0 is irrelevant, and we are interested only in the evolution from this moment onward. Even this goal is still too ambitious, since a numerical method cannot represent the whole function continuously. Instead, we select a finite set of points within the interval by introducing a partition where the first point is x_0 and the last is $x_0 + T$, that is, $\{x_0 < x_1 < \ldots < x_n = x_0 + T\}$. Our goal, therefore, is to approximate the value of the exact solution at these discrete points.

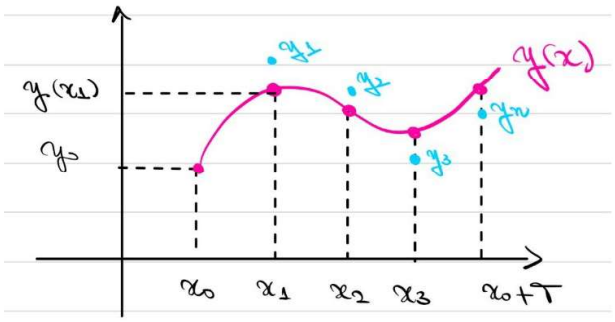

Figure 50: General framework for a numerical method to solve a differential equation.

Let $y(x)$ denote the exact solution of the initial value problem. For each node x_i we compute a numerical approximation, denoted by y_i, which should satisfy $y_i \approx y(x_i)$ (see once again Figure 50). Notice that the value at the initial point is known exactly $y_0 = y(x_0)$, since it is prescribed by the initial condition. Thus, the output of a numerical method is a sequence of numbers $y_0, y_1, \ldots, y_n$, which approximates the corresponding values of the exact solution (again take a look at Figure 50).

Once such approximations are obtained, we must quantify their accuracy. The natural measure is the difference between the exact value and its numerical approximation. At each node we define the absolute

error $|y(x_i) - y_i|$. Since we are interested in the performance of the method over the entire interval $[x_0, x_0 + T]$, we consider the worst-case scenario and define the **global error** as

$$E_{\text{global}} = \max_{0 \leqslant i \leqslant n} |y(x_i) - y_i|.$$

The aim of numerical methods is to produce approximations such that this global error is as small as possible.

To simplify the discussion, we restrict ourselves to a particularly convenient choice of partition: an equidistant grid (see Figure 51). The interval $[x_0, x_0 + T]$ is divided into n subintervals of equal length. The key parameter is therefore the number of subintervals n, from which we obtain the *step size*

$$h = \frac{T}{n}.$$

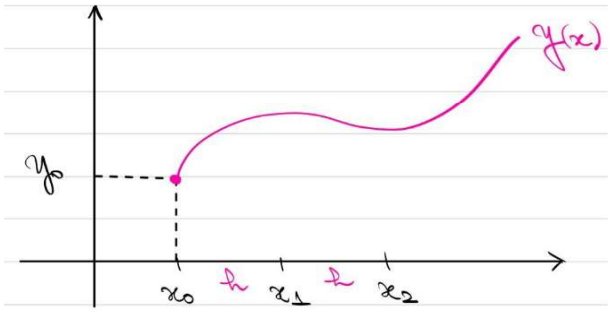

Figure 51: The partition should be equidistant.

Then, the grid points are generated recursively as

$$x_{i+1} = x_i + h$$

starting from x_0. In other words, each node can be written explicitly as

$$x_i = x_0 + ih$$

for every $i = 1, \ldots, n$. We now consider a solution passing through the initial point (x_0, y_0) and seek numerical approximations at these grid points. This setting provides the framework for all the methods studied in this lecture and the next. In particular, we begin with a very simple and yet fundamental method, which serves as a baseline for many more advanced numerical methods and illustrates the essential ideas underlying numerical solution of initial value problems.

10.2 The Euler Method

In order to explain this method, we will start with a simple example. In this example, we will solve an equation of the form $y' = f(x, y)$ with some initial condition $y(x_0) = y_0$. From this equation we know that the slope of the solution curve at the point (x_0, y_0) is given by

$$k = f(x_0, y_0).$$

The idea of Euler's method is to use this slope information to construct a linear approximation of the solution and then iterate the process step by step.

Example 10.1. Consider the following differential equation

$$y' = \frac{1}{1+xy}$$

with initial condition $y(0) = 0$. Observe that this equation is neither separable nor linear, so the analytical techniques previously studied do not apply. In fact, this equation does not admit a closed-form solution in terms of elementary functions.

In order to obtain an approximation of the solution in the interval $[0, 3]$ with partition size $n = 3$, we use the so-called **Euler method**. In this case, the step size is

$$h = \frac{3-0}{3} = 1.$$

The initial point is $(x_0, y_0) = (0, 0)$. To compute the next approximation, we evaluate the slope at $(0, 0)$. From the differential equation, the slope is given by

$$k_0 = f(0, 0) = \frac{1}{1+0\cdot 0} = 1.$$

The next point x_1 should be calculated in the following way

$$x_1 = x_0 + 1 \cdot h = 0 + 1 \cdot 1 = 1.$$

Let us notice here that, in order to obtain x_2, we could have used the formula

$$x_2 = x_1 + h = 1 + 1 = 2$$

which means geometrically that we are going to the right by using the step size. In order to find the point y_1, we observe Figure 52, and conclude that

$$y_1 = y_0 + k \cdot h$$

which is the formula for moving along the straight line. This implies that

$$y_1 = y_0 + k \cdot h = 0 + 1 \cdot 1 = 1.$$

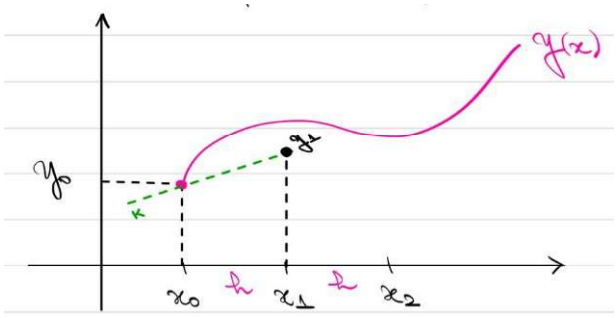

Figure 52: First step of the Euler method.

We have obtained, then, the point $(x_1, y_1) = (1, 1)$ as in the Figure 53.

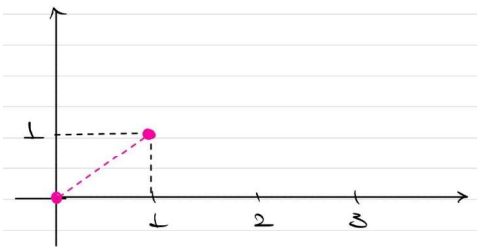

Figure 53: Still the first step of the Euler method.

This simply means that the first approximation to the solution at $x_1 = 1$ is $y_1 = 1$. We repeat the procedure (see Figure 54). At $(x_1, y_1) = (1, 1)$, the slope is

$$k_1 = f(1,1) = \frac{1}{1+1\cdot 1} = \frac{1}{2}$$

and the next point x_2 is

$$x_2 = x_1 + h = 1 + 1 = 2.$$

Then, we have that

$$y_2 = y_1 + hk_1 = 1 + 1\cdot\frac{1}{2} = \frac{3}{2}.$$

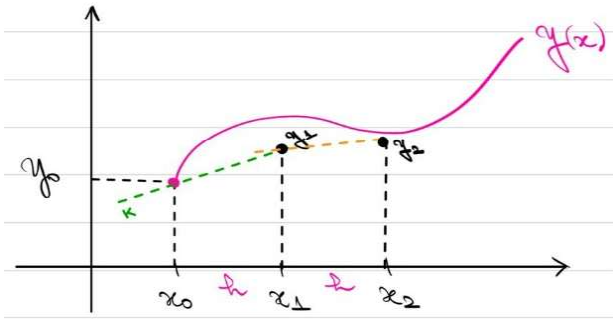

Figure 54: The second step of the Euler method.

At $(x_2, y_2) = (2, \frac{3}{2})$, we compute

$$k_2 = f\left(2, \tfrac{3}{2}\right) = \frac{1}{1+2\cdot\frac{3}{2}} = \frac{1}{1+3} = \frac{1}{4},$$

$x_3 = x_2 + h = 2 + 1 = 3$ and

$$y_3 = y_2 + hk_2 = \frac{3}{2} + 1\cdot\frac{1}{4} = \frac{7}{4}.$$

Therefore, the approximations at the nodes $x_0 = 0$, $x_1 = 1$, $x_2 = 2$ and $x_3 = 3$ are

$$y_0 = 0, \qquad y_1 = 1, \qquad y_2 = \frac{3}{2} \qquad \text{and} \qquad y_3 = \frac{7}{4}$$

(see Figure 55).

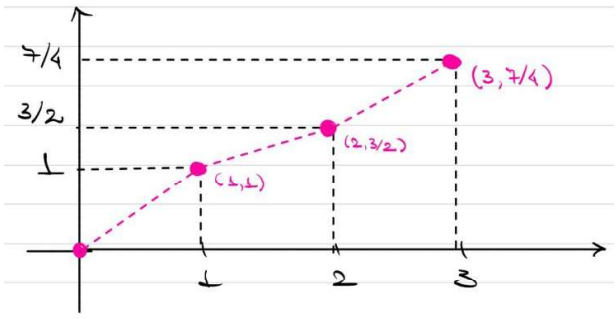

Figure 55: This is what we have done so far.

These values provide a step-by-step linear approximation to the true solution on the interval $[0, 3]$.

Let us formalize this procedure.

Algorithm 10.2 (Euler (forward) formula for IVP $y' = f(x, y)$)**.** Given an ODE $y' = f(x, y)$ on $[x_0, x_0+T]$ with initial value y_0 and $n \in \mathbb{N}$, we have the following steps.

0. Set $h = \frac{T}{n}$.

1. x_0 and y_0 are given.

2. For $i = 0, \ldots, n-1$, set $x_{i+1} = x_i + h$ and $y_{i+1} = y_i + f(x_i, y_i) \cdot h$.

So, how can we determine whether these approximations are sufficiently accurate? At this stage, we are not able to answer this question, since we do not know the exact solution of the previous initial value problem. Without an explicit expression for the true solution, we cannot directly compute the error of our numerical approximations.

We will postpone this discussion for the moment and turn to an even simpler example, where the exact solution is known and a precise comparison can be carried out.

Example 10.3. Consider the differential equation

$$y' = \frac{1}{2}y$$

with initial condition $y(0) = 1$. Notice that we do know how to solve this equation (this is in fact the exponential growth) and its general solution, as we have seen before, is given by $y(x) = e^{x/2}$. Let us pretend we do not know how to handle it and use the Euler approximations in the interval $[0, T]$ for some $T > 0$ with partition size $n \in \mathbb{N}$. The step size is given by $h = \frac{T}{n}$.

In our case, we have that $x_0 = 0$ and $y_0 = 1$. We write the procedure to generate the approximations. We have that

$$x_{i+1} = x_i + h$$

and

$$y_{i+1} = y_i + \left(\frac{1}{2}y_i\right) \cdot h$$

for every $i = 0, 1, \ldots, n-1$. Let us Maple on it (see Figure 56).

```
> r:=0.5:
  ODEe:=diff(y(x),x)=r*y(x);
  ae:=0:ystarte:=1:
  be:=5:
  yviewe:=exp(r*be):
  sole:=ystarte*exp(r*x):
  gsole:=plot(sole,x=ae..be,thickness=2,color=navy,view=[ae..be,0..exp(r*be)],legend="solution"):
```

$$ODEe := \frac{d}{dx} y(x) = 0.5\,y(x)$$

Figure 56: Let us consider our problem on Maple.

If we consider the interval $[0, 5]$ with step size $h = 1$ (that is, $n = 5$), then we get the following approximations (see Figure 57).

```
> n:=5:
  ODENumeric(ODEe,yinit=ystarte,x=ae..be,method=foreuler,numsteps=n,style=point,output=graph,solution=
  sole,plotops=[symbolsize=15]);
Using step size 1.000000.
```

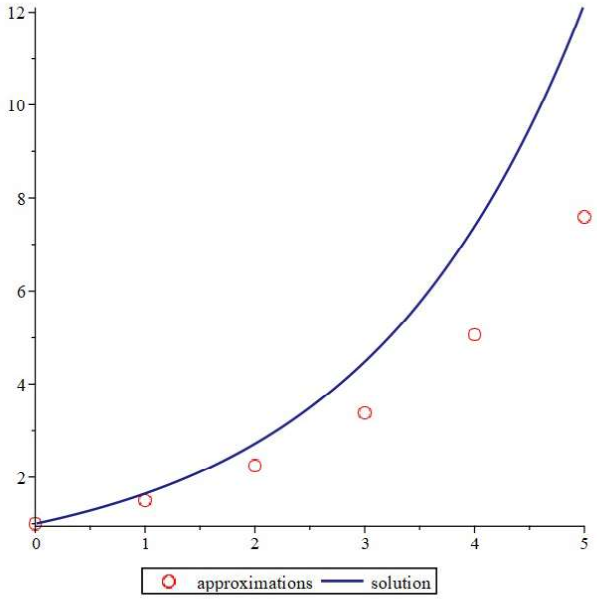

Figure 57: Approximations for the solution using the interval $[0, 5]$ and step size $h = 1$.

There is another way to visualize the method: we may connect the computed points with straight line segments (see Figure 58). It is important to emphasize that these segments are only a graphical aid. They do not represent the true solution, nor do they provide additional analytical information; they merely illustrate how Euler's method advances from one node to the next using successive linear approximations.

If we refine the partition, that is, if we decrease the step size h (by increasing n), the numerical approximation improves, as suggested in Figure 59. By taking an even finer partition (now with $n = 135$), the agreement becomes more pronounced (see Figure 60).

```
> n:=5:
  ODENumeric(ODEe,yinit=ystarte,x=ae..be,method=foreuler,numsteps=n,style=both,output=graph,solution=
  sole,plotops=[symbolsize=15]);
Using step size 1.000000.
```

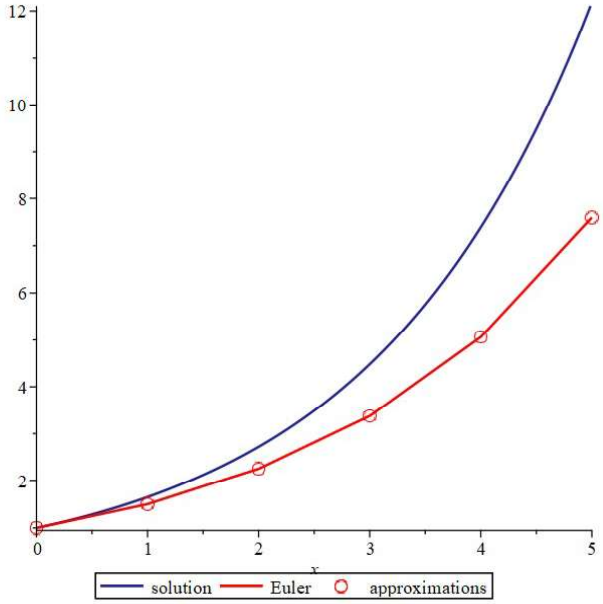

Figure 58: Approximations using straight lines for the solution using the interval $[0, 5]$ and step size $h = 1$.

```
> n:=35:
  ODENumeric(ODEe,yinit=ystarte,x=ae..be,method=foreuler,numsteps=n,style=both,output=graph,solution=
  sole,plotops=[symbolsize=15]);
Using step size 0.142857.
```

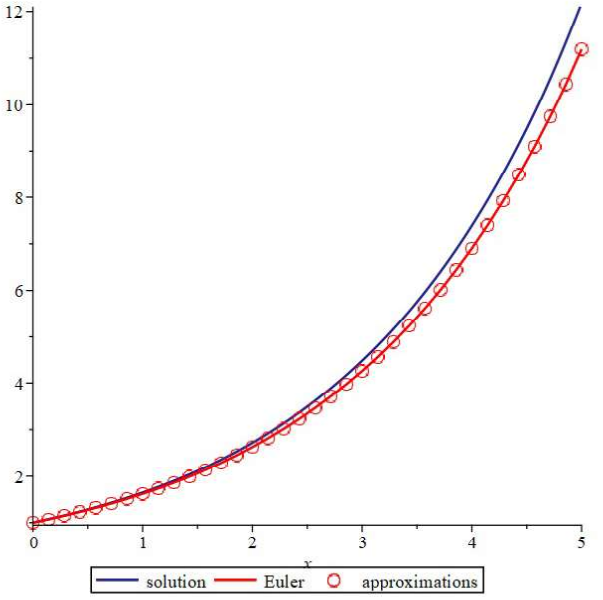

Figure 59: We increase the number of partitions to $n = 35$.

```
> n:=135:
  ODENumeric(ODEe,yinit=ystarte,x=ae..be,method=foreuler,numsteps=n,style=line,output=graph,solution=
  sole,plotops=[symbolsize=15]);
Using step size 0.037037.
```

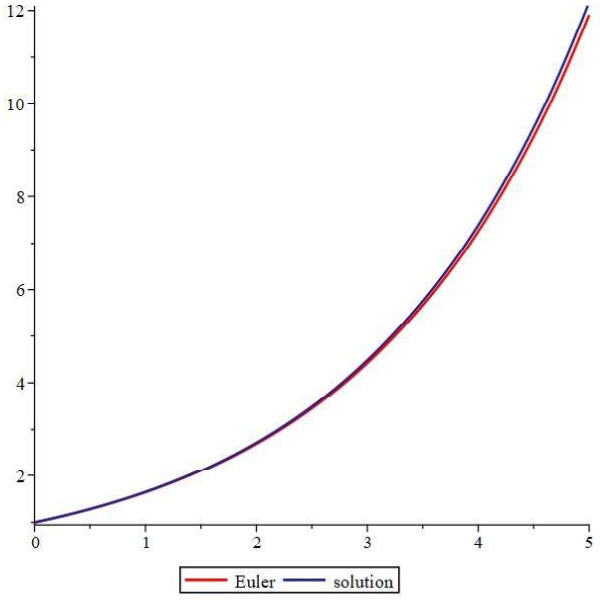

Figure 60: We increase the number of partitions to $n = 135$.

This shows that, at least for this example, the Euler method does a good job by refining the interval, that is, by increasing the partition size $n \in \mathbb{N}$. Let us see that we can in fact get much more than that.

Still in our example, we have observed that

$$y_{i+1} = y_i + \left(\frac{1}{2}y_i\right) h.$$

Factoring out y_i, we obtain

$$y_{i+1} = \left(1 + \frac{1}{2}h\right) y_i,$$

for every $i = 0, 1, \ldots, n-1$. This recursive relation allows us to compute the successive approximations explicitly. Indeed,

$$y_1 = \left(1 + \frac{1}{2}h\right) y_0,$$

and therefore

$$y_2 = \left(1 + \frac{1}{2}h\right) y_1 = \left(1 + \frac{1}{2}h\right)^2 y_0.$$

Proceeding in the same way,

$$y_3 = \left(1 + \frac{1}{2}h\right)^3 y_0.$$

By induction, this pattern suggests that

$$y_i = \left(1 + \frac{1}{2}h\right)^i y_0$$

for every $i \in \mathbb{N}$. As $h = \frac{T}{n}$, for the last point of our partition y_n, we have that

$$y_n = \left(1 + \frac{1}{2} \cdot \frac{T}{n}\right)^n y_0.$$

If we see this last expression as a sequence in n, then we can calculate the limit to get

$$\lim_{n\to\infty} y_n = \lim_{n\to\infty} \left(1 + \frac{1}{2} \cdot \frac{T}{n}\right)^n y_0 = y_0 \cdot e^{\frac{T}{2}}$$

which is the actual solution of the equation recalling that $y_0 = 1$.

Example 10.4. Let us now consider the logistic growth model. The differential equation is

$$y' = \frac{y(10 - y)}{10},$$

where the carrying capacity of the environment is $K = 10$ (see Figure 61). We apply Euler's method to approximate its solutions (see Figure 62).

```
> r1:=1:K:=10:
  ODE1:=diff(y(x),x)=(r1/K)*y(x)*(K-y(x));
  a1:=0:ystart1:=0.9:
  b1:=12:
  yview1:=K+1:
  sol1:=(K*ystart1*exp(r1*x))/(K-ystart1+ystart1*exp(r1*x)):
  gsol1:=plot(sol1,x=a1..b1,thickness=2,color=navy,view=[a1..b1,0..yview1],legend="solution")
```

$$ODE1 := \frac{d}{dx} y(x) = \frac{y(x)\,(10 - y(x))}{10}$$

Figure 61: The logistic growth computed in Maple with carrying capacity $K = 10$.

```
> hstep:=1.3:
  ODENumeric(ODE1,yinit=ystart1,x=a1..b1,method=foreuler,stepsize=hstep,output=graph,solution=sol1);
Using step size 1.333333.
```

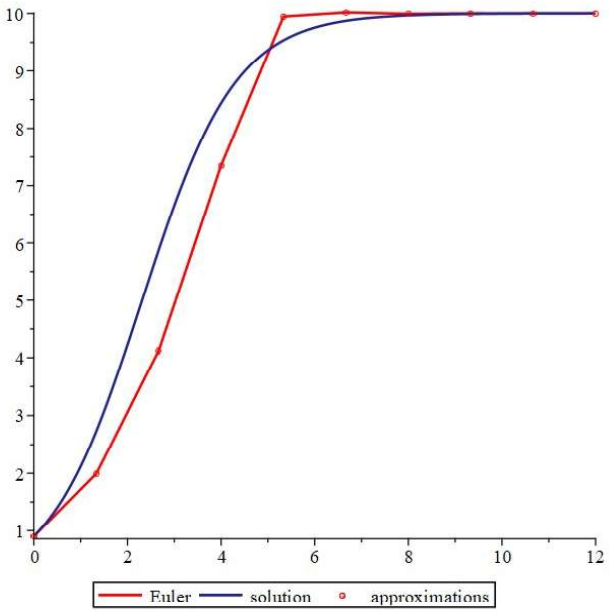

Figure 62: Application of Euler's method to the logistic growth model.

We observe that Euler's method is still able to capture the qualitative behavior of the solution in a reasonable way, including the saturation effect as y approaches the carrying capacity. Let us now decrease the step size in order to refine the approximation (see Figure 63).

```
> hstep:=0.3:
  ODENumeric(ODE1,yinit=ystart1,x=a1..b1,method=foreuler,stepsize=hstep,output=graph,solution=sol1);
```

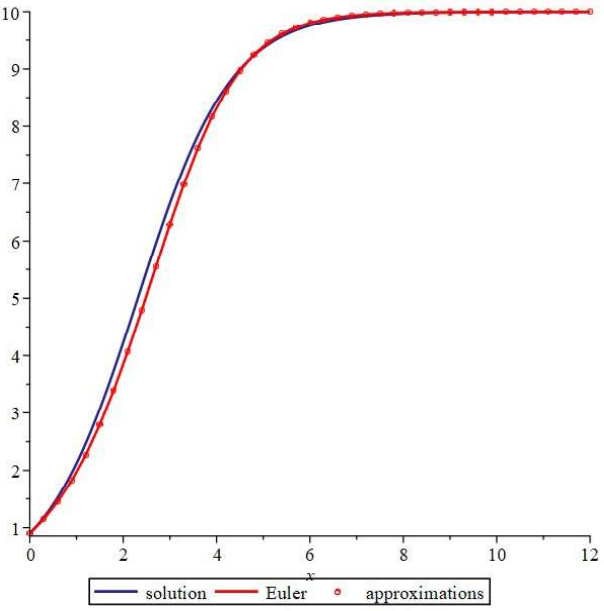

Figure 63: Euler's method applied to the logistic model with a smaller step size.

Once again, we see that reducing the step size improves the numerical approximation, yielding a curve that more closely follows the true solution.

10.3 Global and local errors, and order of the method

In this section, we analyze the accuracy of the Euler's method. Consider a Cauchy problem $y' = f(x, y)$ with initial condition $y(x_0) = y_0$ on the interval $[x_0, x_0 + T]$. For a fixed partition size $n \in \mathbb{N}$, Euler's method generates a discrete set of approximations $\{(x_i, y_i)\}$ where we emphasize that both the nodes and the numerical values depend on n (equivalently, on the step size $h = T/n$). As we have discussed before, the global error associated with this partition is defined as

$$E_n = \max_i |y(x_i) - y_i|,$$

that is, the maximum difference between the exact solution and the numerical approximation at the grid points. The central question is whether this error tends to zero as the partition is refined. In other words, whether

$$E_n \longrightarrow 0 \quad \text{as } n \to \infty.$$

We introduce the following definition, which applies to numerical methods such as Euler's method.

Definition 10.5. Consider some numerical method for solving initial value problems $y' = f(x, y(x))$, $y(x_0) = y_0$ that for given $T > 0$ and $n \in \mathbb{N}$ produces approximations $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$ of the

solution on $[x_0, x_0+T]$. We say that this method is **convergent** if the following is true: for any IVP with f Lipschitz in the second variable that has a solution $y(x)$ on some $[x_0, x_0+T]$ and every $n \in \mathbb{N}$, consider the corresponding approximations $\{(x_0, y_0), \ldots, (x_n, y_n)\}$ and define

$$E_n = \max_i |y(x_i) - y_i|.$$

We require that $E_n \to 0$ as $n \to \infty$.

In fact, our main interest is not only to determine whether the error converges to zero, but also to quantify the *rate* of convergence. More precisely, we seek an estimate of the form

$$E_h \approx c \cdot h^p$$

for some constant $c > 0$ and some exponent $p \in \mathbb{R}$. The number p is called the **order of convergence** of the method and measures how rapidly the global error decreases as the step size h tends to zero.

It is important to emphasize that Euler's method is, by construction, a *local* method. At each step, the new approximation

$$y_{n+1} = y_n + hf(x_n, y_n)$$

is obtained using only the information available at the current point (x_n, y_n), namely the value of the vector field at that point. The method replaces the true solution locally by the tangent line and advances one step forward. Consequently, the error analysis naturally begins with the local truncation error, which measures the error committed in a single step. The global error E_h then arises from the accumulation of these local errors along the entire partition.

Definition 10.6. Consider a one-step method Φ_f for solving initial value problems that for an equation $y' = f(x, y(x))$ and a point (x^*, y^*) generates an estimate $\Phi_f(x^*, y^*, h)$ for the value for the corresponding solution at $x^* + h$. Given an ODE $y' = f(x, y)$, we define the **local error** of the method as

$$d_f(x^*, y^*, h) = y_*(x^* + h) - \Phi_f(x^*, y^*, h)$$

for all $x^*, y^* \in \mathbb{R}$ and all step sizes $h > 0$ such that the IVP $y' = f(x, y)$, $y(x^*) = y^*$ has a solution $y_*(x)$ on $[x^*, x^* + h]$.

Intuitively, the local error measures what happens in a single step of the method under ideal conditions (in order to apply Picard Theorem 2.10). Suppose we start at a point (x^*, y^*) that lies exactly on the true solution curve. The exact solution then gives us the value $y_*(x^* + h)$ after one step of length h, while the numerical method produces the approximation $\Phi_f(x^*, y^*, h)$. The difference between these two values is the local error. In other words, it quantifies how accurately the method reproduces the true solution over just one step, assuming no previous numerical errors have accumulated.

Definition 10.7. Consider a one-step method Φ_f for solving initial value problems that for an equation $y' = f(x, y(x))$ and a point (x^*, y^*) generates an estimate $\Phi_f(x^*, y^*, h)$ for the value of the corresponding solution at $x^* + h$. We say that the method is of **order** p, or that it has error of order p, if the following is true: for every differential equation $y' = f(x, y)$ and rectangle $I \times J$ such that f is Lispchitz with respect to y in $I \times J$ and sufficiently smooth there is some $C > 0$ so that

$$|d_f(x^*, y^*, h)| \leqslant C \cdot h^{p+1}$$

for all $(x^*, y^*) \in I \times J$ and $h > 0$ such that $(x^* + h, \Phi_f(x^*, y^*, h)) \in I \times J$.

The notion of order (see Figure 64) formalizes how small this one-step error is when the step size h becomes small. If the method is of order p, then the local error behaves like a constant times h^{p+1} for sufficiently small h. This means that each individual step becomes rapidly more accurate as we refine the mesh. The higher the order p, the faster the local error decreases when h is reduced, and consequently, the more efficient the method is in achieving high accuracy.

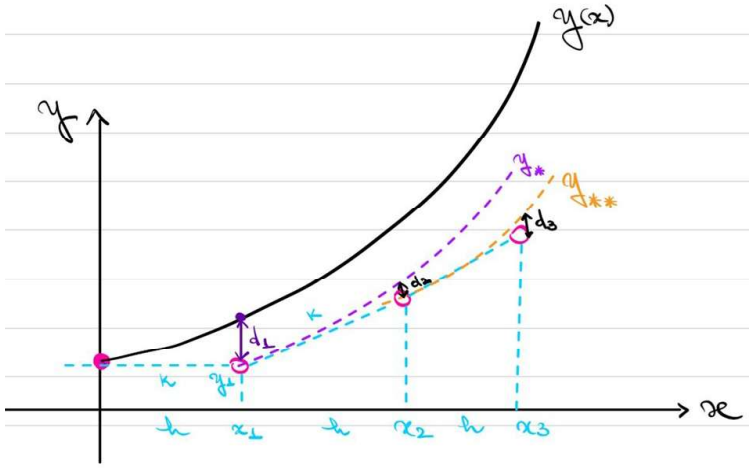

Figure 64: Local error in Euler's method.

For Euler's method, it is possible to carry out a rigorous analysis of the local truncation error by means of a Taylor expansion of the exact solution. This analysis leads to a precise quantitative estimate and yields the following result. The local truncation error measures the error made in a single step of a one-step method, assuming that the step starts from the exact solution value, and for a method of order p it is $O(h^{p+1})$. The global error, on the other hand, is the total error between the exact solution and the numerical approximation after many steps, so it includes both the new error introduced at each step and the accumulation of previous errors. Since on a fixed interval the number of steps is proportional to $1/h$, the $O(h^{p+1})$ local errors are accumulated over about $1/h$ steps, and under stability this leads to a total error of order $O(h^p)$. Therefore, the global error loses one power of h compared with the local error: local order $p+1$ becomes global order p. In particular, Euler's method has local error $O(h^2)$, so its global error is $O(h)$, and therefore Euler's method is of order 1 as we can see in the next result.

Theorem 10.8. The Euler's method is of order 1 with respect to differential equations $y' = f(x, y)$ such that f is differentiable on its domain and its derivatives are bounded on bounded rectangles.

10.4 The integral approach for the Euler's method

Notice that we have developed the following. Under suitable assumptions, we considered Cauchy problems of the form $y' = f(x, y)$ with $y(x_0) = y_0$, and applied Euler's method to obtain approximations $y_i \approx y(x_i)$. In particular, for the x-values we advance as $x^* \mapsto x^* + h$, and for the y-valued we approximated them by $\Phi_f(x^*, y^*, h)$ using the expression $y^* + k \cdot h$.

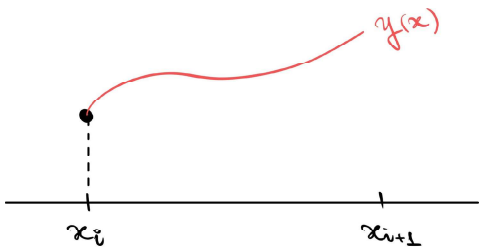

Figure 65: Analysis of the solution in a single panel.

We now adopt a different perspective by using numerical integration, as previously discussed (see Section 8.2). Starting from the differential equation $y' = f(x, y)$, we integrate both sides over the interval $[x_i, x_{i+1}]$ (let us just see what happens in one panel as in Figure 65) to obtain

$$\int_{x_i}^{x_{i+1}} y'(x)\,dx = \int_{x_i}^{x_{i+1}} f(x, y(x))\,dx.$$

By the Fundamental Theorem of Calculus, we have that

$$\int_{x_i}^{x_{i+1}} y'(x)\,dx = y(x_{i+1}) - y(x_i).$$

Hence,

$$y(x_{i+1}) - y(x_i) = \int_{x_i}^{x_{i+1}} f(x, y(x))\,dx.$$

Equivalently,

$$y(x_{i+1}) = y(x_i) + \int_{x_i}^{x_{i+1}} f(x, y(x))\,dx. \tag{18}$$

We now approximate the integral on the right-hand side by using the left rectangle method to yield

$$\int_{x_i}^{x_{i+1}} f(x, y(x))\,dx \approx f(x_i, y(x_i)) \cdot h = f(x_i, y_i) \cdot h.$$

Therefore,

$$y(x_{i+1}) \approx y_{i+1} = y(x_i) + f(x_i, y_i) \cdot h,$$

which recovers Euler's method.

Now we wonder what happens if, instead, we apply the right rectangle method. In this case, using (18) we obtain a new approximation for the integral. Applying this quadrature rule gives

$$y_{i+1} = y_i + f(x_{i+1}, y(x_{i+1})) \cdot h.$$

However, a difficulty arises: the value $y(x_{i+1})$ is unknown. Suppose that we approximate it by $y_{i+1} \approx y(x_{i+1})$. Then we obtain

$$y_{i+1} = y_i + f(x_{i+1}, y_{i+1}) \cdot h.$$

This scheme is known as the **backward (implicit) Euler method**.

Algorithm 10.9 (backward (implicit) Euler formula)**.** Given: ODE $y' = f(x, y)$ on $[x_0, x_0 + T]$, initial value y_0 and $n \in \mathbb{N}$.

0. Set $h = \frac{T}{n}$.

1. x_0 and y_0 are given.

2. For $i = 0, 1, \ldots, n-1$, set $x_{i+1} = x_i + h$ and $y_{i+1} = y_i + f(x_{i+1}, y_{i+1}) \cdot h$ for y_{i+1}.

Of course, one issue we must address is whether the point y_{i+1} actually exists. This is precisely why the method is called *implicit*. The value y_{i+1} appears on both sides of the equation, so it must be determined by solving the resulting equation. In this sense, the procedure is circular, and only if we are fortunate (for instance, if the equation admits a solution that we can compute) will we obtain the desired value. Let us see an example.

Example 10.10. Consider once again the exponential growth equation $y' = \frac{1}{2}y$ with $y(0) = 1$, and apply the implicit Euler method on an interval $[0, T]$ with partition size $n \in \mathbb{N}$. This yields the step size $h = \frac{T}{n}$. The initial values x_0 and y_0 are given. In this case, $x_{i+1} = x_i + h$ and

$$y_{i+1} = y_i + \left(\frac{1}{2}y_{i+1}\right) \cdot h,$$

which is an implicit equation for y_{i+1}. We can solve it explicitly by isolating y_{i+1} and therefore

$$y_{i+1} = \frac{y_i}{1 - \frac{1}{2}h}.$$

Notice that we may encounter issues if $h = 2$, since the denominator vanishes. Moreover, if $h > 2$ the factor $1 - \frac{1}{2}h$ becomes negative, so the iterates can alternate in sign and the approximation may behave poorly. Let us now take a look at the performance of the implicit Euler method compared to the (explicit) Euler method. See Figures 66 and 67.

```
> hstep:=1:
  gEf:=ODENumeric(ODEe,yinit=ystarte,x=ae..be,method=foreuler,stepsize=hstep,style=point,plotops=[symbol=
  circle,color=red,symbolsize=10,legend="Euler"]):
  gEb:=ODENumeric(ODEe,yinit=ystarte,x=ae..be,method=backeuler,stepsize=hstep,style=point,plotops=[symbol=
  diamond,color=brown,symbolsize=15,legend="implicit Euler"],silent=on):
  plots[display]({gEf,gEb,gsole},view=[ae..be,0..yviewe]);
```

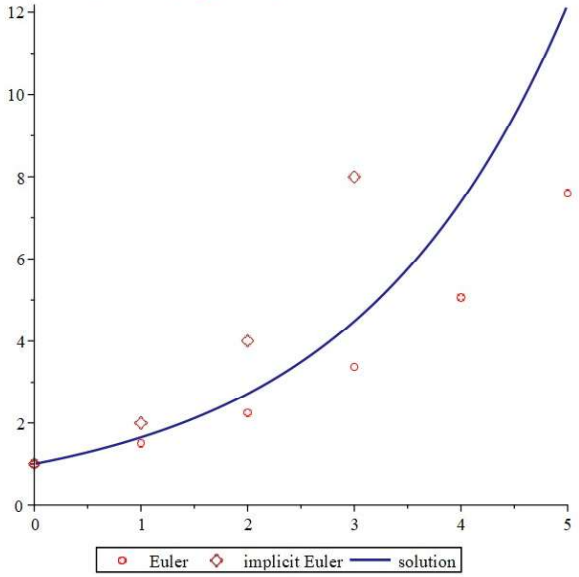

Figure 66: Comparison between the implicit Euler's method and Euler's method.

```
> hstep:=0.5:
  gEf:=ODENumeric(ODEe,yinit=ystarte,x=ae..be,method=foreuler,stepsize=hstep,style=point,plotops=[symbol=
  circle,color=red,symbolsize=10,legend="Euler"]):
  gEb:=ODENumeric(ODEe,yinit=ystarte,x=ae..be,method=backeuler,stepsize=hstep,style=point,plotops=[symbol=
  diamond,color=brown,symbolsize=15,legend="implicit Euler"],silent=on):
  plots[display]({gEf,gEb,gsole},view=[ae..be,0..yviewe]);
```

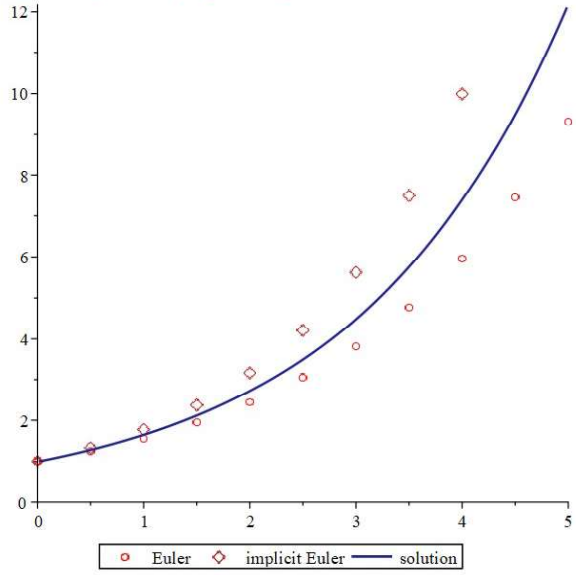

Figure 67: Comparison between the implicit Euler's method and Euler's method with smaller step size.