# 11 Lecture #8: Wednesday, March 11th, 2026

In this class, we will see second order methods.

## 11.1 The Heun method

Recall that, in Subsection 10.4, we used both the left and the right rectangle rules to approximate the integral in

$$y(x_{i+1}) = y_i + \int_{x_i}^{x_{i+1}} f(x, y(x))\, dx. \tag{19}$$

To obtain a better approximation, and a method of higher order than Euler's, we can instead use the trapezoid method. In this case,

$$\int_{x_i}^{x_{i+1}} f(x, y(x))\, dx \approx \frac{1}{2}\Big[f(x_i, y(x_i)) + f(x_{i+1}, y(x_{i+1}))\Big] \cdot h,$$

and therefore

$$y_{i+1} = y_i + \frac{1}{2}\Big[f(x_i, y_i) + f(x_{i+1}, y(x_{i+1}))\Big] \cdot h.$$

Once again, we do not know the value of $y(x_{i+1})$. A common strategy is to approximate it using Euler's formula. In this way, we first compute a provisional value $y^*_{i+1}$ (see Figure 68) given by

$$y^*_{i+1} = y_i + h\, f(x_i, y_i),$$

(in Figure 68, $k_1$ is the slope given by $f(x_i, y_i)$) and then define

$$y_{i+1} = y_i + \frac{1}{2}\Big[f(x_i, y_i) + f(x_{i+1}, y^*_{i+1})\Big] \cdot h. \tag{20}$$

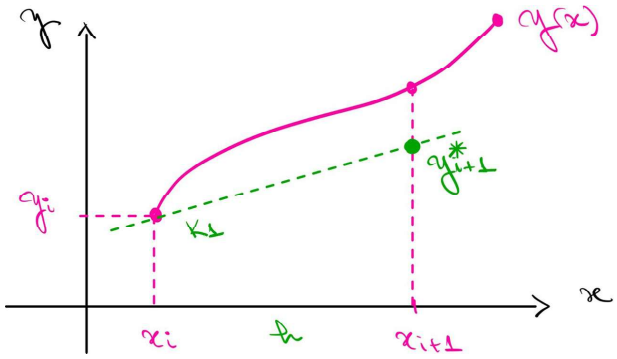


Figure 68: Temporary value for $y(x_{i+1})$.

If $k_2$ denotes the slope $f(x_{i+1}, y^*_{i+1})$, then in (20) we are simply averaging the slopes $k_1$ and $k_2$ to obtain

$$k = \frac{k_1 + k_2}{2}.$$

The value of $y_{i+1}$ is then given by the straight line of slope $k$ passing through the point $(x_i, y_i)$, as illustrated in Figure 70.

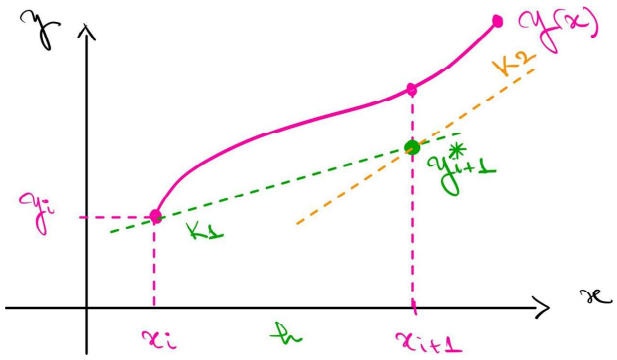


Figure 69: Slope $k_2 = f(x_{i+1}, y^*_{i+1})$.

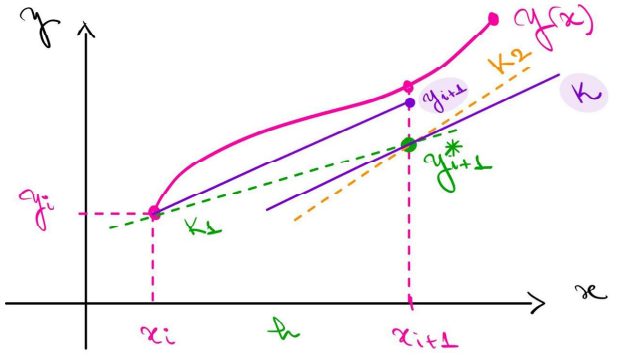


Figure 70: We find the average between the slopes $k_1$ and $k_2$. We then call it $k$ and then the value for $y_{i+1}$ is going to be as in the picture.

Let us formalize it.

**Algorithm 11.1** (Heun formula (improved 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:

(a) Estimate the slope $y'(x_i) : k_1 f(x_i, y_i)$.

(b) Estimate $y_{i+1} : y^*_{i+1} = y_i + k_1 h$, then estimate the slope $y'(x_{i+1}) : k_2 = f(x_{i+1}, y^*_{i+1})$.

(c) Set $y_{i+1} = y_i + \frac{1}{2}(k_1 + k_2) \cdot h$.

We have the following expected result.

**Fact 11.2.** The Heun formula method is of order 2.

The Heun method is a *predictor-corrector* scheme: Euler's method is first used to predict a provisional value, and this prediction is then used to correct the slope through an averaged evaluation of the derivative.

## 11.2 The midpoint method (RK2)

There is another method of order 2 that also approximates the integral appearing in (19). This is the **midpoint method**. The idea is to evaluate the slope at the midpoint of the interval $[x_i, x_{i+1}]$. We denote this midpoint by $x_{i+\frac{1}{2}}$. Thus, while $x_{i+1} = x_i + h$, we have

$$x_{i+\frac{1}{2}} = x_i + \frac{1}{2}h.$$

Using Euler's method, we compute a provisional approximation $y^*_{i+\frac{1}{2}}$ given by

$$y^*_{i+\frac{1}{2}} = y_i + k_1 \cdot \left(\frac{1}{2}h\right)$$

where $k_1 = f(x_i, y_i)$ is the slope at $x_i$ (see Figure 71).

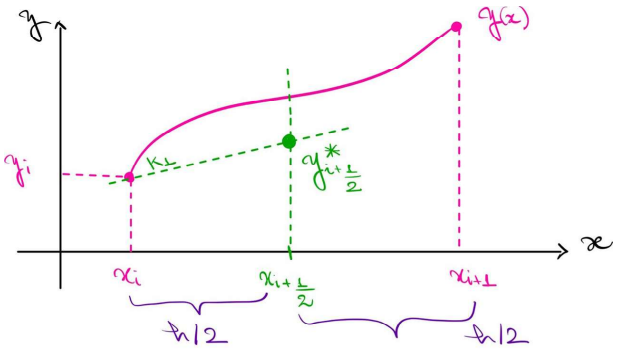


Figure 71: The provisional point $y^*_{i+\frac{1}{2}}$ obtained using Euler's method.

We then evaluate the slope at $x_{i+\frac{1}{2}}$ using the provisional point $y^*_{i+\frac{1}{2}}$. This slope is used to construct a line parallel to the one determined by $k_2$, starting from the point $(x_i, y_i)$, until we reach the point $y_{i+1}$ as illustrated in Figure 72. The value of $y_{i+1}$ is therefore given by

$$y_{i+1} = y_i + k_2 \cdot h$$

where $k_2 = f\left(x_{i+\frac{1}{2}},\, y^*_{i+\frac{1}{2}}\right)$.

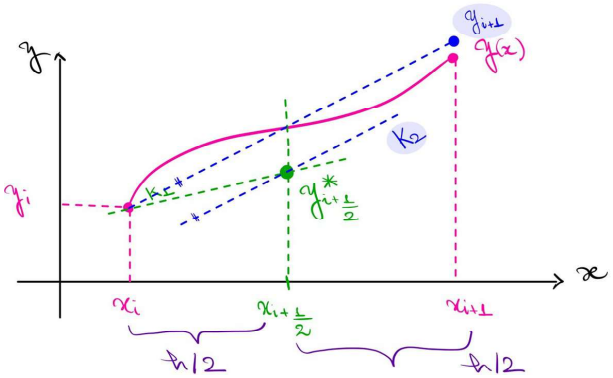


Figure 72: The provisional point $y^*_{i+\frac{1}{2}}$ obtained using Euler's method.

Formally, we have the following algorithm.

**Algorithm 11.3** (RK2 (midpoint, modified Euler formula, improved polygon))**.** 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 $x_{i+1} = x_i + h$.

(a) Estimate the slope $y'(x_i) : k_1 = f(x_i, y_i)$.

(b) Estimate $y(x_i + \frac{1}{2}h)$: $y^*_{i+\frac{1}{2}} = y_i + k_1 \cdot \left(\frac{1}{2}h\right)$, then estimate the slope

$$y'\left(x_i + \frac{1}{2}h\right) : \; k_2 = f\left(x_i + \frac{1}{2}h, y^*_{i+\frac{1}{2}}\right).$$

(c) Set $y_{i+1} = y_i + k_2 \cdot h$.

It turns out that

**Fact 11.4.** The midpoint method is of order 2.

We will see soon why this method is also called RK2. Let us notice that we consider different methods of the same order because, although they share the same theoretical accuracy, they can differ significantly in stability, computational cost, and practical performance for particular differential equations.

Let us examine the performance of these methods using Maple. The figures 73 and 74 show a comparison between the exact solution and the numerical approximations obtained using the Euler, implicit Euler, midpoint (RK2) and Heun methods. As expected, the second–order methods (midpoint and Heun) follow the exact solution much more closely, while the Euler method exhibits a larger error and systematically underestimates the growth of the solution. The implicit Euler method, on the other hand, tends to overestimate the solution in this example, illustrating how different numerical schemes of similar step size can display noticeably different behaviors. Furthermore, this illustrates an important principle in numerical analysis: by evaluating the derivative at more than one point within each interval, higher–order methods can capture the local behavior of the solution much more effectively.

```
> hstep:=1.0:
  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):
  gH:=ODENumeric(ODEe,yinit=ystarte,x=ae..be,method=heunform,stepsize=hstep,style=point,plotops=[symbol=box,color=blue,symbolsize=10,
  legend="Heun method (RK2)"],silent=on):
  gRK:=ODENumeric(ODEe,yinit=ystarte,x=ae..be,method=midpoint,stepsize=hstep,style=point,plotops=[symbol=cross,color=magenta,symbolsize=
  18,legend="midpoint (RK2)"],silent=on):
  plots[display]({gEf,gEb,gH,gRK,gsole},view=[ae..be,0..yviewe]);
```

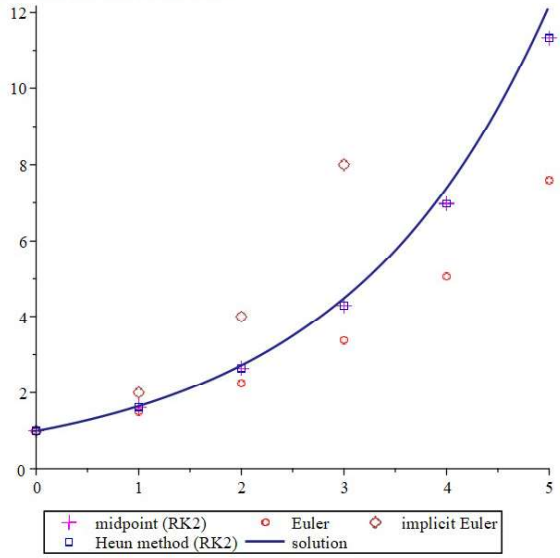


Figure 73: The midpoint, Euler, implicit Euler and Heun methods together with the real solution.

```
> 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):
  gH:=ODENumeric(ODEe,yinit=ystarte,x=ae..be,method=heunform,stepsize=hstep,style=point,plotops=[symbol=box,color=blue,symbolsize=10,
  legend="Heun method (RK2)"],silent=on):
  gRK:=ODENumeric(ODEe,yinit=ystarte,x=ae..be,method=midpoint,stepsize=hstep,style=point,plotops=[symbol=cross,color=magenta,symbolsize=
  18,legend="midpoint (RK2)"],silent=on):
  plots[display]({gEf,gEb,gH,gRK,gsole},view=[ae..be,0..yviewe]);
```

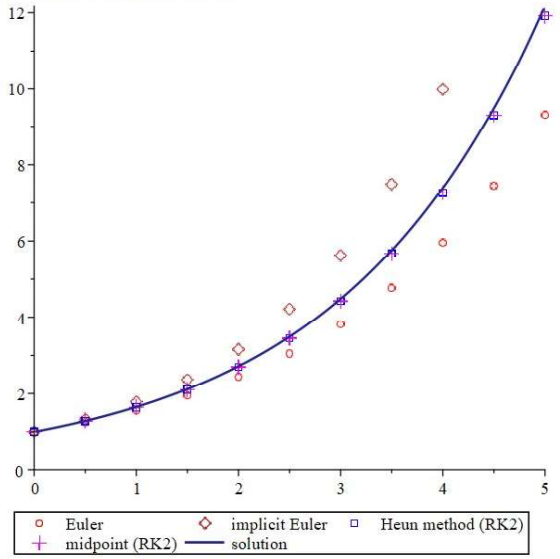


Figure 74: Now with smaller step size.

## 11.3 Runge-Kutta methods

Runge-Kutta methods are numerical methods for solving initial value problems of the form $y' = f(x, y)$ that improve the accuracy of Euler's method by evaluating the slope $f(x, y)$ at several points within each step and combining these slopes in a weighted average. Instead of using only the slope at the beginning of the interval, as in Euler's method, Runge–Kutta methods compute intermediate slopes (such as at midpoints or predicted points) to better approximate the integral of $f(x, y)$ over the step. In this way, they achieve higher-order accuracy while still remaining single-step methods, meaning that each new approximation depends only on the previous point and not on earlier values.

Let us now examine what we mean by a Runge-Kutta method and see how the Heun and midpoint methods arise as particular cases.

**Definition 11.5.** An explicit **Runge-Kutta method** for solving the IVP $y' = f(x, y)$ is given by fixing parameters

$$\begin{array}{c|ccccc} 0 & & & & & \\ c_2 & a_{21} & & & & \\ c_3 & a_{31} & a_{32} & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ c_N & a_{N1} & a_{N2} & \cdots & a_{N,N-1} & \\ \hline & w_1 & w_2 & \cdots & w_{N-1} & w_N \end{array}$$

(this is known as a **Butcher tableau**). Here $N$ is the number of steps, $c_j$ defines nodes, $a_{jl}$ forms the matrix of the method and $w_j$ are weights. When determining $y_{i+1}$ using $y_i$ we first estimate slopes at various points,

$$\begin{aligned} k_1 &= f(x_i, y_i), \\ k_2 &= f(x_i + c_2 h,\, y_i + a_{21} k_1 h), \\ k_3 &= f(x_i + c_3 h,\, y_i + (a_{31} k_1 + a_{32} k_2) h), \\ &\vdots \\ k_N &= f(x_i + c_N h,\, y_i + (a_{N1} k_1 + a_{N2} k_2 + \cdots + a_{N,N-1} k_{N-1}) h)\,, \end{aligned}$$

these estimates are averaged to get the best slope

$$k = \sum_{j=1}^{N} w_j k_j$$

and then we set

$$y_{i+1} = y_i + k\, h.$$

As we have mentioned before, the Heun method and the midpoint method are second–order Runge-Kutta methods (and then sometimes called RK2). In the Runge-Kutta framework, the coefficients of the method are summarized using a *Butcher tableau* as above, which specifies where the slopes are evaluated and how they are combined.

The Heun method evaluates the slope at the beginning of the interval and then at a predicted endpoint. These two slopes are then averaged.

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 1/2 & 1/2 \end{array}$$

From this tableau we obtain

$$k_1 = f(x_i, y_i), \qquad k_2 = f(x_i + h,\, y_i + hk_1),$$

and the update formula

$$y_{i+1} = y_i + \frac{h}{2}(k_1 + k_2).$$

On the other hand, the midpoint method evaluates the slope at the beginning of the interval and then at the midpoint using a provisional Euler step.

$$\begin{array}{c|cc} 0 & & \\ 1/2 & 1/2 & \\ \hline & 0 & 1 \end{array}$$

From this tableau we obtain

$$k_1 = f(x_i, y_i), \qquad k_2 = f\left(x_i + \tfrac{1}{2}h,\; y_i + \tfrac{1}{2}hk_1\right),$$

and the update formula

$$y_{i+1} = y_i + h\, k_2.$$

The classical fourth–order Runge–Kutta method (RK4, for short) is the most widely used Runge–Kutta method because it provides a very good balance between accuracy and computational cost. By evaluating the slope four times within each step and combining these values in a weighted average, RK4 achieves fourth–order accuracy, which significantly reduces the numerical error compared with lower–order methods such as Euler or second–order Runge–Kutta methods. At the same time, the method remains relatively simple to implement and does not require solving implicit equations. For many practical problems, this combination of reliability, accuracy, and simplicity makes RK4 a standard choice in scientific computing.

**Algorithm 11.6** (RK4 for IVP $y' = f(x, y)$)**.** Given: ODE $y' = f(x, y)$ on $[x_0, x_0 + T]$, initial value $y_0$, and $n \in \mathbb{N}$.

**0.** Set $h = \dfrac{T}{n}$.

**1.** $x_0$ and $y_0$ are given.

**2.** For $i = 0, \ldots, n-1$ set $x_{i+1} = x_i + h$ and:

(a) Estimate the slope $y'(x_i)$:

$$k_1 = f(x_i, y_i).$$

(b) Estimate $y(x_i + \frac{1}{2}h)$:

$$y^*_{i+1/2} = y_i + \tfrac{1}{2}k_1 h$$

and then estimate the slope $y'(x_i + \frac{1}{2}h)$:

$$k_2 = f(x_i + \tfrac{1}{2}h, y^*_{i+1/2}).$$

(c) Again (to improve?) estimate $y(x_i + \frac{1}{2}h)$:

$$y^{**}_{i+1/2} = y_i + \tfrac{1}{2}k_2 h$$

and then estimate the slope $y'(x_i + \frac{1}{2}h)$:

$$k_3 = f(x_i + \tfrac{1}{2}h, y^{**}_{i+1/2}).$$

(d) Estimate $y(x_i + h)$:

$$y^{*}_{i+1} = y_i + k_3 h$$

and then estimate the slope $y'(x_{i+1})$:

$$k_4 = f(x_{i+1}, y^{*}_{i+1}).$$

(e) Set

$$y_{i+1} = y_i + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]\, h.$$

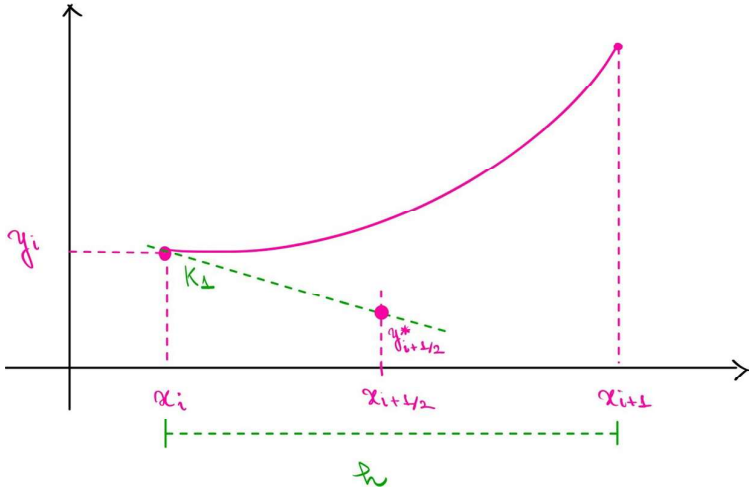


Figure 75: Steps (a) and (b) from Algorithm 11.6.

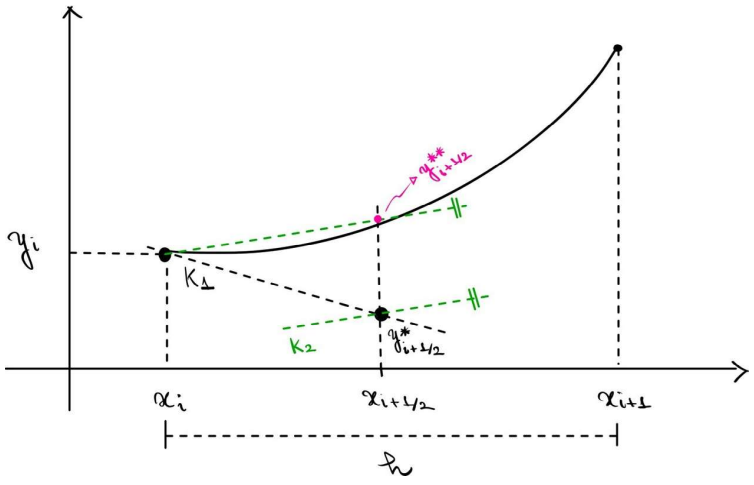


Figure 76: Step (c) from Algorithm 11.6.

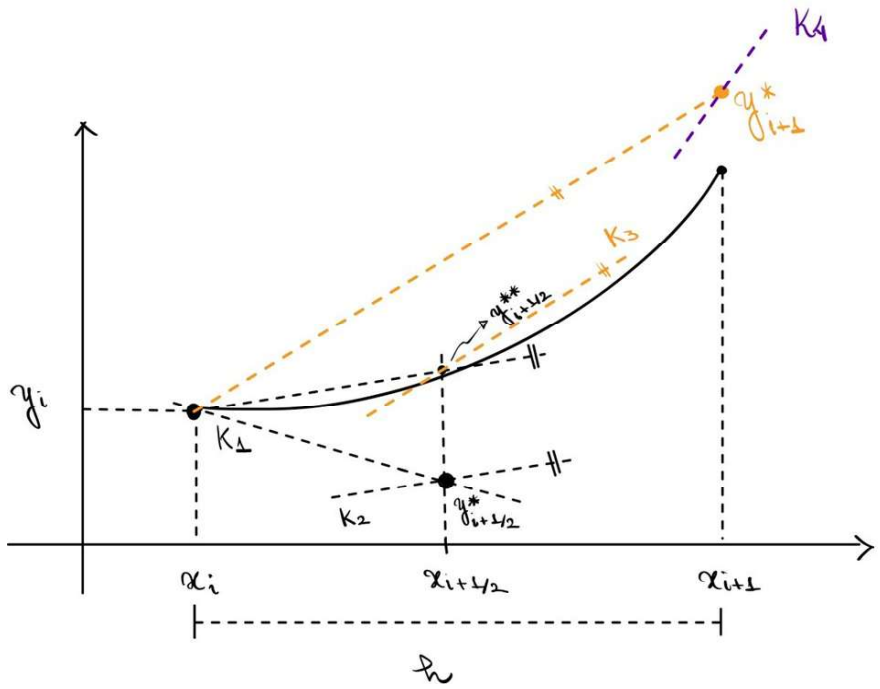


Figure 77: Step (d) from Algorithm 11.6.

The geometric idea behind the classical RK4 method is to approximate the average slope of the true solution over the interval $[x_i, x_{i+1}]$ by sampling the vector field $f(x, y)$ at several carefully chosen points within the step. Instead of relying on a single slope (as in Euler's method), RK4 evaluates slopes at the beginning of the interval, twice near the midpoint, and once near the end. Geometrically, these slopes describe several possible directions that the solution curve could take as it moves through the vector field. By combining them in the weighted average

$$k = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

the method constructs a slope that closely represents the true average direction of the solution across the step. The new point $y_{i+1}$ is then obtained by moving from $(x_i, y_i)$ along this averaged direction for a distance $h$. Notice that the slope taken as $k = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$, where the midpoint slopes $k_2$ and $k_3$ are given greater importance because they receive weight 2 each, making them heavier in the average than the endpoint slopes $k_1$ and $k_4$, which reflects the fact that the intermediate behavior of the solution is especially informative for achieving higher accuracy.

This process produces a much better approximation of the actual trajectory of the solution curve than using a single local slope. Indeed, we have the following result.

**Fact 11.7.** The RK4 method is of order 4.

We can see some performance of this method in Figure 78. The graph compares the numerical approximations produced by several methods, Euler, implicit Euler, midpoint (RK2), Heun (RK2) and now RK4 as well, with the exact solution of the differential equation. The exact solution is represented by the smooth curve, while the discrete markers show the approximations at the grid points. Notice once again that the Euler method underestimates the solution as the error grows with each step, while the

implicit Euler method tends to overestimate it; again the second–order methods, midpoint and Heun, follow the exact curve much more closely because they incorporate additional slope information within each step. Now, the new RK4 method provides the best approximation among the plotted methods: its points lie almost exactly on the true solution, illustrating how higher–order Runge–Kutta methods significantly improve accuracy for the same step size.

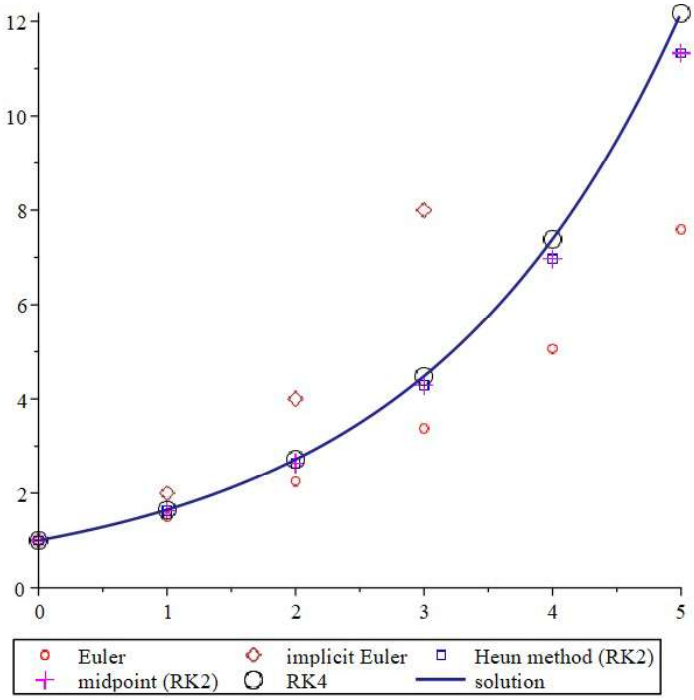


Figure 78: Comparison of the methods we have seen so far together with RK4.

## 11.4 Estimating error via control method

Normally, the exact solution of an IVP is not known. Therefore, a natural question arises: how can we assess the quality of the numerical methods we have introduced so far? To illustrate this issue, consider the differential equation $y' = \cos(x)\, y^2$ with initial condition $y(6\pi) = 2/3$. Let us first examine the performance of the Euler method, shown in Figure 79.

At first glance, it may not be clear whether these approximations are accurate. However, once we compare them with the true solution, we see that the Euler method performs very poorly (Figure 80).

When the exact solution of an IVP is unknown, a common strategy to assess the accuracy of a numerical method is to estimate the error by comparing solutions obtained with different step sizes. This approach is often called *error control* or *step–size control*. The general idea is to compute the numerical solution using a step size $h$, and then compute another approximation using a smaller step size, typically $h/2$. If the method is convergent, the two approximations should be close to each other, and the difference between them provides an estimate of the numerical error. If this estimated error is larger than a prescribed tolerance, the step size is reduced; if it is sufficiently small, the step size may be increased to improve efficiency. In this way, the algorithm dynamically adjusts the step size to maintain a desired level of accuracy even when the exact solution is not available.

```
> ODENumeric(ODEc,yinit=ystartc,x=ac..bc,method=foreuler,numsteps=40,output=plot,style=point,solution=ysolc,
  plotops=[symbol=circle,color=red,symbolsize=10,view=[ac..bc,0..2.0],legend="Euler"]);
Using step size 0.471239.
```

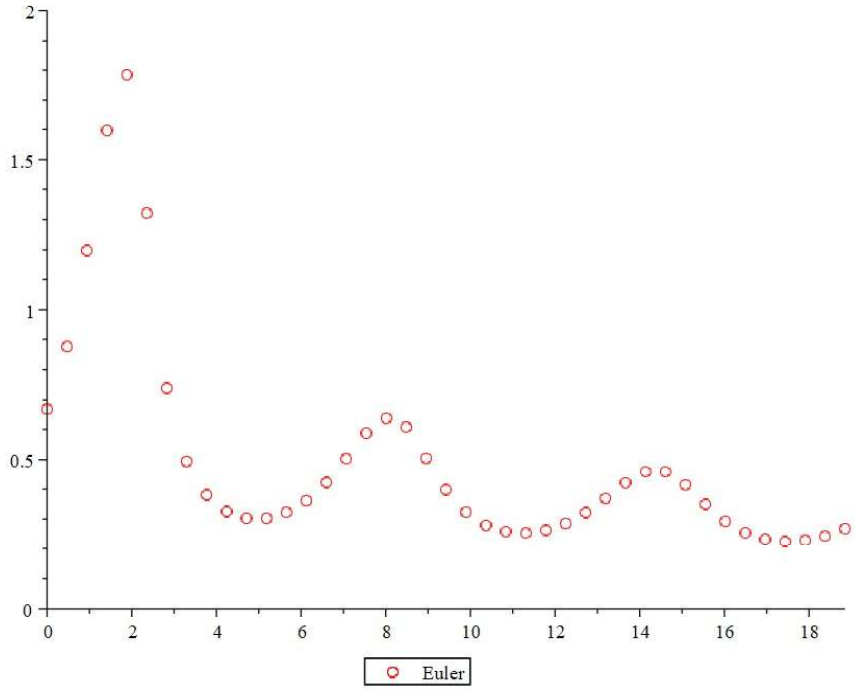


Figure 79: The Euler method applied to the equation $y' = \cos(x)y^2$ with $y(6\pi) = \frac{2}{3}$.

```
> ODENumeric(ODEc,yinit=ystartc,x=ac..bc,method=foreuler,numsteps=40,output=graph,style=point,solution=ysolc,plotops=[symbol=circle,color=
  red,symbolsize=10,view=[ac..bc,0..2.0],legend="Euler"]);
Using step size 0.471239.
```

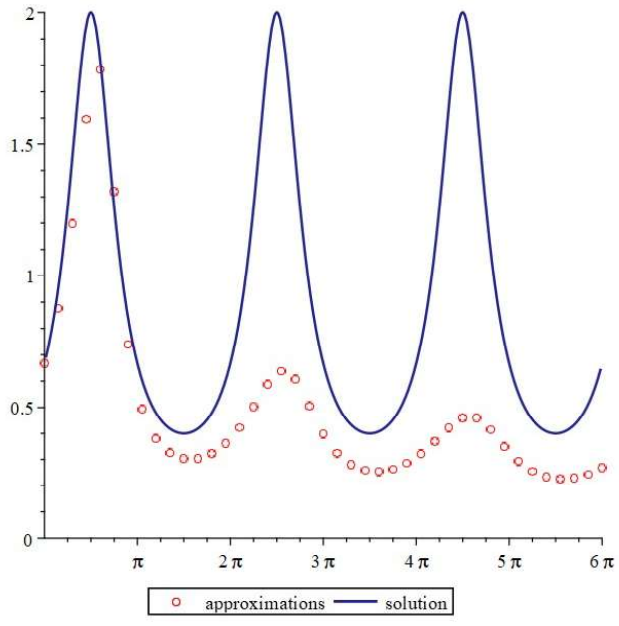


Figure 80: The Euler method applied to $y' = \cos(x)y^2$ with $y(6\pi) = \frac{2}{3}$, together with the exact solution.

Assume a numerical method has order $p$. This means that, for sufficiently small step size $h$, the error can be modeled as

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

where $c > 0$ is a constant that depends on the IVP and the method we are considering but not on $h$. If we now reduce the step size by a factor $a > 0$, i.e., we use $h/a$, then

$$E_{h/a} \approx c \left(\frac{h}{a}\right)^p = c\, h^p \, \frac{1}{a^p} = \frac{1}{a^p}\,(c\, h^p) \approx \frac{1}{a^p}\, E_h.$$

Hence, we have that

$$E_{h/a} \approx \frac{1}{a^p}\, E_h.$$

**Example 11.8.** Suppose that $y(x)$ is a solution (that we do not know how to find) of a certain IVP. As we wanted to approximate $y(5)$, we suppose that we used Euler's method (recall that it is of order $p = 1$) with step size $h = 0.2$, obtaining an estimated error $E_{0.2} \approx 0.030$.

a) If we use the same method with step size $h/4 = 0.05$, then

$$E_{0.05} \approx E_{0.2} \left(\frac{1}{4}\right)^1 = 0.030 \cdot \frac{1}{4} = 0.0075.$$

b) To achieve a target error $E = 0.002$, we solve

$$0.002 \approx 0.030 \left(\frac{h}{0.2}\right)^1 \quad \Longrightarrow \quad h \approx 0.2 \cdot \frac{0.002}{0.030} \approx 0.0133.$$

Thus, a recommended step size is $h \approx 0.0133$.

**Example 11.9.** Suppose once again that $y(x)$ is a solution (unknown in closed form) of a certain IVP. As before, we suppose that we wanted to approximate $y(2)$ and we used a second-order Runge-Kutta method (for instance, Heun or midpoint, so it is of order $p = 2$) with step size $h = 0.1$, obtaining an estimated error $E_{0.1} \approx 0.012$.

a) If we use the same method with step size $h/4 = 0.025$, then

$$E_{0.025} \approx E_{0.1} \left(\frac{1}{4}\right)^2 = 0.012 \cdot \frac{1}{16} = 0.00075.$$

b) To achieve a target error $E = 0.001$, we solve

$$0.001 \approx 0.012 \left(\frac{h}{0.1}\right)^2 \quad \Longrightarrow \quad \left(\frac{h}{0.1}\right)^2 = \frac{1}{12} \quad \Longrightarrow \quad h = 0.1\sqrt{\frac{1}{12}} \approx 0.0289.$$

Thus, a recommended step size is $h \approx 0.0289$.

**Example 11.10.** Suppose that $y(x)$ is a solution (unknown in closed form) of a certain IVP. We want to approximate $y(10)$ and we used RK4 (order $p = 4$) with step size $h = 0.05$, obtaining an estimated error $E_{0.05} \approx 0.0048$.

a) If we use the same method with step size $h/4 = 0.0125$, then

$$E_{0.0125} \approx E_{0.05} \left(\frac{1}{4}\right)^4 = 0.0048 \cdot \frac{1}{256} = 0.00001875.$$

b) To achieve a target error $E = 0.001$, we solve

$$0.001 \approx 0.0048 \left(\frac{h}{0.05}\right)^4 \quad \Longrightarrow \quad \left(\frac{h}{0.05}\right)^4 = \frac{0.001}{0.0048} = \frac{5}{24} \quad \Longrightarrow \quad h = 0.05 \left(\frac{5}{24}\right)^{1/4} \approx 0.0338.$$

Thus, a recommended step size is $h \approx 0.0338$.