

10.3: Backward Euler Method

In the **Backward Euler Method**, we take

$$\vec{y}_{n+1} = \vec{y}_n + h\vec{F}(\vec{y}_{n+1}, t_{n+1}). \quad (10.3.1)$$

Comparing this to the formula for the Forward Euler Method, we see that the inputs to the derivative function involve the solution at step $n+1$, rather than the solution at step n . As $h \rightarrow 0$, both methods clearly reach the same limit. Similar to the Forward Euler Method, the local truncation error is $O(h^2)$.

Because the quantity \vec{y}_{n+1} appears in both the left- and right-hand sides of the above equation, the Backward Euler Method is said to be an **implicit method** (as opposed to the Forward Euler Method, which is an explicit method). For general derivative functions F , the solution for \vec{y}_{n+1} cannot be found directly, but has to be obtained *iteratively*, using a numerical approximation technique such as [Newton's method](#). This makes the Backward Euler Method substantially more complicated to implement, and slower to run.

However, implicit methods like the Backward Euler Method have a powerful advantage: it turns out that they are generally stable regardless of step size. By contrast, explicit methods—even explicit methods that are much more sophisticated than the Forward Euler Method, like the Runge-Kutta methods discussed below—are unstable when applied to stiff problems, if the step size is too large. To illustrate this, let us apply the Backward Euler Method to the same ODE, $dy/dt = -\kappa y(t)$, discussed previously. For this particular ODE, the implicit equation can be solved exactly, without having to use an iterative solver:

$$y_{n+1} = y_n - h\kappa y_{n+1} \Rightarrow y_{n+1} = \frac{y_n}{1 + h\kappa} = \frac{y_0}{(1 + h\kappa)^n}. \quad (10.3.2)$$

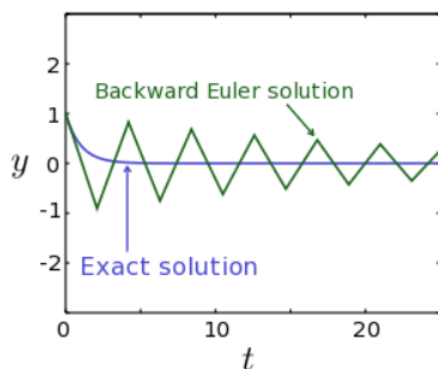


Figure 10.3.1: The exact solution (blue) and Backward Euler solution (green) for the same problem as Fig. 10.2.1. The numerical solution is stable.

From this result, we can see that the numerical solution does not blow up for large values of h , as shown for example in Fig. 10.3.1. Even though the numerical solution in this example isn't accurate (because of the large value of h), the key point is that the error does not accumulate and cause a blow-up at large times.

This page titled [10.3: Backward Euler Method](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Y. D. Chong](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.