Some Notes on Numerical Methods for Differential Equations

Yoichiro Mori

March 27, 2019

1 Accuracy

Consider the differential equation:

$$\frac{dx}{dt} = f(x), \; x(0) = x_0, \; x \in \mathbb{R}^M.$$  \hfill (1)

We consider numerical approximations to this differential equation. The simplest method is the following Forward Euler method (scheme):

$$\frac{x_{n+1} - x_n}{\Delta t} = f(x_n)$$  \hfill (2)

Here, $\Delta t$ is the time step and $x_n$ is the approximation to $x(n\Delta t)$, the true value of $x$ at time $n\Delta t$. We may solve the above for $x_{n+1}$ to find:

$$x_{n+1} = x_n + f(x_n)\Delta t.$$  \hfill (3)

We thus have a discrete dynamical system that gives us the value $x_{n+1}$ given $x_n$.

We are interested in knowing how well our Forward Euler numerical solution approximates the true solution. This amounts to estimating the difference between $x_n$ and $x(n\Delta t)$. Let us examine this difference using an example. Consider:

$$\frac{dx}{dt} = x, \; x(0) = 1.$$  \hfill (4)
The solution to this equation is, clearly,

$$x(t) = e^t.$$  \hfill (5)

Now, let us use the Forward Euler method to find a numerical solution. We solve up to $t = 1$ and increase $N$, the number of time steps taken to reach $t = 1$. This means that $\Delta t = 1/N$. Numerical experimentation suggests that, as $\Delta t$ is halved ($N$ is doubled), the error is reduced by a half. Thus,

$$|x(1) - x_N| \approx C\Delta t, \ N\Delta t = 1,$$  \hfill (6)

where $C$ is some number that does not depend on $N$ or $\Delta t$. In general, for the Forward Euler scheme, it can be shown that:

$$|x(T) - x_N| \leq C\Delta t, \ N\Delta t = T$$  \hfill (7)

where the constant $C$ does not depend on $N$ or $\Delta t$. The error, therefore, is roughly proportional to $\Delta t$.

More generally, if a numerical method satisfies:

$$|x(T) - x_N| \leq C\Delta t^k, \ N\Delta t = T$$  \hfill (8)

then, this numerical method is said to be $k$-th order accurate. The Forward Euler method is thus a 1st order accurate scheme.

If one can devise a numerical method of higher order, the error will decrease faster. Indeed, if we have a 2nd order scheme, the error will be $1/4$ as $\Delta t$ is halved, whereas in a 1st order scheme, the error was only $1/2$. How can we devise a second order scheme?

For inspiration, we look to numerical integration. Consider the approximate computation of the integral:

$$I = \int_0^a f(t)dt.$$  \hfill (9)

One way to approximate this integral is to consider:

$$I_L = \sum_{n=0}^{N-1} f(t_n)\Delta t, \ t_n = n\Delta t, \ \Delta t = \frac{a}{N}.$$  \hfill (10)

This is a sum of the area of thin rectangles. The height of the rectangles are given by evaluating $f$ at the left endpoint of the interval $t_k \leq t \leq t_{k+1}$, so
we may call this the left endpoint rule. How good is the left endpoint rule?
It turns out that:

$$|I - I_L| \leq C \Delta t$$  \hspace{1cm} (11)

where $C$ does not depend on $\Delta t$.

Many of you probably know that a better approximation to the integral is
to use the trapezoidal or midpoint rules:

$$I_M = \sum_{n=0}^{N-1} f(t_{n+1/2}) \Delta t, \quad t_{n+1/2} = (n + 1/2) \Delta t,$$
(midpoint rule)

$$I_T = \sum_{n=0}^{N-1} \frac{f(t_n) + f(t_{n+1})}{2} \Delta t,$$
(trapezoidal rule)

For $I_M$ and $I_T$ (assuming that $f$ is twice continuously differentiable) are
better approximations to $I$ in the sense that:

$$|I - I_T| \leq C \Delta t^2, \quad |I - I_M| \leq C \Delta t^2.$$  \hspace{1cm} (13)

We may say that the left endpoint rule is first order accurate whereas the
midpoint and trapezoidal rules are second order accurate.

The left endpoint rule is much like the Forward Euler method. This suggests
that we can adapt the midpoint and trapezoidal rules for numerical integra-
tion to devise a second order numerical scheme for differential equations.
The \textit{midpoint rule} and \textit{trapezoidal rule} for differential equations are given
by:

$$\frac{x_{n+1} - x_n}{\Delta t} = f \left( \frac{x_n + x_{n+1}}{2} \right), \quad \text{(midpoint rule)},$$

$$\frac{x_{n+1} - x_n}{\Delta t} = \frac{1}{2} (f(x_n) + f(x_{n+1})), \quad \text{(trapezoidal rule)}.$$  \hspace{1cm} (14)

It is known that the above schemes indeed give second order accuracy. Note
that the above equations must be solved for for $x_{n+1}$. This is in contrast to
the Forward Euler scheme, in which $x_{n+1}$ was written explicitly in terms of
$x_n$ (see (3)). The midpoint and trapezoidal rules are \textit{implicit} schemes where
as the Forward Euler method is \textit{explicit}.

Is it possible to devise a numerical method that is second order accurate and
explicit? The following \textit{2nd order Runge Kutta} scheme is an explicit second
order scheme:

\[
\begin{align*}
\frac{x_{n+1/2} - x_n}{\Delta t/2} &= f(x_n), \\
\frac{x_{n+1} - x_n}{\Delta t} &= f(x_{n+1/2}).
\end{align*}
\] (15)

Here, a half step is taken to obtain an approximation \(x_{n+1/2}\), which is then used to obtain \(x_{n+1}\).

It is possible to devise yet higher order methods for numerically solving differential equations. Much as we did above, one may develop such methods by adapting higher order methods for numerical integration.

2 Stability

Consider the equation:

\[\frac{dx}{dt} = -kx, \quad k > 0, \quad x(0) = 1.\] (16)

Clearly, the solution to this differential equation is given by:

\[x(t) = e^{-kt}.\] (17)

Now, let us solve the above equation using the Forward Euler method. We have:

\[\frac{x_{n+1} - x_n}{\Delta t} = -kx_n.\] (18)

We then have:

\[x_{n+1} = (1 - k\Delta t)x_n.\] (19)

Solving this equation, we have:

\[x_n = (1 - k\Delta t)^n.\] (20)

Before we go further, let us remark that, in this case, one can easily show that \(x_n\) indeed approaches \(e^{-kt}\) as \(N \to \infty\). To see this, let \(N\Delta t = T\), and let \(N \to \infty\) which keeping \(T\) fixed. We have:

\[x_N = (1 - k\Delta t)^N = \left(1 - \frac{kT}{N}\right)^N \to e^{-kT} \text{ as } N \to \infty.\] (21)
The last limit is something that you probably saw in a calculus class.

Now, suppose $k = 100$, and we took $\Delta t$ to be 0.1. We then have:

$$x_n = (1 - k\Delta t)^n = (1 - 10)^n = (-9)^n. \quad (22)$$

This would produce complete nonsense. Even though the solution $x(t)$ is decaying rapidly to 0, $x_n$ is oscillating very wildly. This type of behavior is called numerical instability.

Let us examine the conditions under which $|x_n|$ will not go to infinity as $n \to \infty$. Clearly, we need:

$$|1 - k\Delta t| \leq 1. \quad (23)$$

This then shows that we must have:

$$k\Delta t \leq 2, \text{ or equivalently, } \Delta t \leq \frac{2}{k}. \quad (24)$$

What this tells us is that, for large $k$, we must take $\Delta t$ to be quite small just to avoid numerical instabilities.

Now, consider the following Backward Euler scheme for (1):

$$\frac{x_{n+1} - x_n}{\Delta t} = f(x_{n+1}). \quad (25)$$

Here, we must solve an equation for $x_{n+1}$ for each time step, and we thus have an implicit scheme. In terms of accuracy, the Backward and Forward Euler scheme both turn out to be first order schemes.

Let us now apply the Backward Euler scheme to (16). We have:

$$\frac{x_{n+1} - x_n}{\Delta t} = -kx_{n+1}. \quad (26)$$

We thus have:

$$x_{n+1} = (1 + k\Delta t)^{-1}x_n. \quad (27)$$

Thus,

$$x_n = (1 + k\Delta t)^{-n}. \quad (28)$$

Here, we see that Backward Euler scheme does not suffer from numerical instability. However large I take my $\Delta t$, $x_n$ will always decay to 0 with increasing $n$. 

Yoichiro Mori
Let us now consider the system differential equation:

\[
\begin{align*}
\frac{dx_1}{dt} &= -\lambda_1 x_1, \\
\frac{dx_2}{dt} &= -\lambda_2 x_2, \\
&\vdots \\
\frac{dx_M}{dt} &= -\lambda_M x_M,
\end{align*}
\tag{29}
\]

where all the $\lambda_k$ are positive. We may apply the Forward Euler scheme to the above system of differential equations. It can be seen in exactly the same way as before that, in order to avoid numerical instability, we need:

\[
\Delta t \leq \min_{1 \leq k \leq M} \frac{2}{\lambda_k} = \frac{2}{\max_{1 \leq k \leq M} \lambda_k}. 
\tag{30}
\]

More generally, let us consider the differential equation:

\[
\frac{dx}{dt} = -Ax, \quad x \in \mathbb{R}^M 
\tag{31}
\]

where $A$ is a diagonalizable $M \times M$ matrix all of whose eigenvalues $\lambda_k, k = 1, \cdots, M$ are all positive. The case is a special case of this in which $A$ is just a diagonal matrix. It turns out that (30) is still the condition that assures numerical stability. Backward Euler, on the other hand, will remain numerically stable for any value of $\Delta t$.

The reciprocals of the eigenvalues $1/\lambda_k$ are the time scales inherent in the problem. Condition (30) essentially says that $\Delta t$ must be taken smaller than the smallest time scale in the problem. This observation applies more generally. Suppose we have a differential equation model that involves different time scales. A differential equation model for the Earth’s climate, for example, may involve vastly different time scales corresponding to daily effects as well as effects that manifest only across millenia. Suppose you want to understand the climate a million years form now. You want to take time steps that are sufficiently large so that you can perform your simulations in a reasonable amount of time, but, given that day-to-day effects are present in your differential equation, Forward Euler method will require that the $\Delta t$ be at least as small as a day or so, potentially making the computation prohibitively expensive. On the other hand, the Backward Euler method does not suffer from this difficulty.
In general, implicit methods are more numerically stable than explicit methods. Although implicit methods are generally more computationally expensive than explicit methods, implicit methods are often used for numerically stiff problems (problems that involve multiple disparate time scales) since larger time steps can be taken without running into stability issues.