Numerical methods for solving differential equations

The general form of a first-order differential equation (DE) is $y' = f(x, y)$,

Comment. Recall that higher-order differential equations can be written as systems of first-order differential equations: $\boldsymbol{y}' = f(x, \boldsymbol{y})$ in terms of $\boldsymbol{y} = (y_1, y_2, y_3, \ldots)$ where we set $y_1 = y$, $y_2 = y'$, $y_3 = y''$, ...

It therefore is no loss of generality to develop methods for first-order differential equation. While we will focus on the case of a single function $y(x)$, the methods we discuss extend naturally to the case of several functions $\boldsymbol{y}(x) = (y_1(x), y_2(x), \ldots)$.

In order to have a unique solution $y(x)$ that we can numerically approximate, we will add an initial condition. As such, we discuss methods for solving first-order initial value problems (IVPs)

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

Comment. Recall from Differential Equations class that such an IVP is guaranteed to have a unique solution under mild assumptions on $f(x, y)$ (for instance, that $f(x, y)$ is smooth around (x_0, y_0)).

Comment. There would be no loss of generality in only considering only initial conditions of the form $y(0) = y_0$. Indeed, suppose the initial condition is $y(x_0) = y_0$. Then, by replacing x by $x + x_0$ in the DE and rewriting the DE in terms of $\tilde{y}(x) = y(x + x_0)$, we obtain an IVP with initial condition $\tilde{y}(0) = y_0$.

Review of the simplest differential equations

Let's start with one of the simplest (and most fundamental) differential equation (DE). It is **first-order** (only a first derivative) and **linear** (with constant coefficients).

Example 130. Solve $y' = 3y$.

Solution. $y(x) = Ce^{3x}$

Check. Indeed, if $y(x) = Ce^{3x}$, then $y'(x) = 3Ce^{3x} = 3y(x)$.

Comment. Recall we can always easily check whether a function solves a differential equation. This means that (although you might be unfamiliar with certain techniques for solving) you can use computer algebra systems to solve differential equations without trust issues.

To describe a unique solution, additional constraints need to be imposed.

Example 131. Solve the **initial value problem** (IVP) $y' = 3y$, $y(0) = 5$.

Solution. This has the unique solution $y(x) = 5e^{3x}$.

The following is a **non-linear** differential equation. In general, such equations are much more complicated than linear ones. We can solve this particular one because it is **separable**.

Example 132. Solve $y' = xy^2$.

Solution. This DE is separable: $\frac{1}{y^2}\mathrm{d}y = x\,\mathrm{d}x$. Integrating, we find $-\frac{1}{y} = \frac{1}{2}x^2 + C$.

Hence, $y = -\frac{1}{\frac{1}{2}x^2 + C} = \frac{2}{D - x^2}$. [Here, $D = -2C$ but that relationship doesn't matter.]

Comment. Note that we did not find the singular solution $y = 0$ (lost when dividing by y^2). We can obtain it from the general solution by letting $D \to \infty$.

Euler's method

Euler's method is a numerical way of approximating the (unique) solution $y(x)$ to the IVP

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

It follows from Taylor's theorem (Theorem 48) that

$$y(x + h) = y(x) + y'(x)h + \frac{1}{2}y''(\xi)h^2.$$

Choose a step size $h > 0$. Write $x_n = x_0 + nh$. Our goal is to provide approximations y_n of $y(x_n)$ for $n = 1, 2, \ldots$.

Since we know $y(x_0) = y_0$, we approximate

$$\begin{aligned}
y(x_0 + h) &\approx y(x_0) + y'(x_0)h \overset{\text{DE}}{=} y(x_0) + f(x_0, y(x_0))h \\
y(x_0 + 2h) &\approx y(x_0 + h) + y'(x_0 + h)h \overset{\text{DE}}{=} y(x_0 + h) + f(x_0 + h, y(x_0 + h))h \\
y(x_0 + 3h) &\approx y(x_0 + 2h) + y'(x_0 + 2h)h \overset{\text{DE}}{=} y(x_0 + 2h) + f(x_0 + 2h, y(x_0 + 2h))h \\
&\vdots
\end{aligned}$$

Comments.

- Here we use $y(x + h) \approx y(x) + y'(x)h$ first with $x = x_0$, then with $x = x_0 + h$ and so on.
- Note how, when approximating $y(x_0 + mh)$, we use the previous approximation $y(x_0 + (m - 1)h)$. All other quantities on the right-hand side are known to us.
- Clearly, the error in these approximations will accumulate and the approximations are likely worse as we continue (in other words, our approximations of $y(x)$ will be worse as x gets further away from x_0).

Write $x_n = x_0 + nh$. Our goal is to provide approximations y_n of $y(x_n)$ for $n = 1, 2, \ldots$.

Note that we start with x_0 and y_0 from the initial condition.

In terms of x_n and y_n our above approximations become:

$$y(x_n + h) \approx y(x_n) + \underbrace{y'(x_n)}_{f(x_n, y(x_n))} h \approx y_n + f(x_n, y_n)h =: y_n$$

Two kinds of errors. There are two different errors involved here: in the first approximation, the error is from truncating the Taylor expansion and we know that this **local truncation error** is $O(h^2)$. On the other hand, in the second approximation, we introduce an error because we use the previous approximation y_n instead of $y(x_n)$.

Suppose that we approximate $y(x)$ on some interval $[x_0, x_{\max}]$ using n steps (so that $x_n = x_{\max}$).

Then the step size is $h = \frac{x_{\max} - x_0}{n}$. We therefore have $n = \frac{x_{\max} - x_0}{h}$ many local truncation errors of size $O(h^2)$. It is therefore natural to expect that the **global error** is $O(nh^2) = O(h)$.

(Euler's method) The following is an order 1 method for solving IVPs:

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

Comment. As explained above, being an order 1 method means that Euler's method has a global error that is $O(h)$ (while the local truncation error is $O(h^2)$).

Comment. While Euler's method is rarely used in practice, it serves as the foundation for more powerful extensions such as the Runge–Kutta methods.

Armin Straub
straub@southalabama.edu

Example 133. Consider the IVP $y' = y$, $y(0) = 1$. Approximate the solution $y(x)$ for $x \in [0, 1]$ using Euler's method with 4 steps. In particular, what is the approximation for $y(1)$?

Comment. Of course, the real solution is $y(x) = e^x$. In particular, $y(1) = e \approx 2.71828$.

Solution. The step size is $h = \frac{1-0}{4} = \frac{1}{4}$. We apply Euler's method with $f(x, y) = y$:

$$\begin{array}{ll}
x_0 = 0 & y_0 = 1 \\
x_1 = \frac{1}{4} & y_1 = y_0 + f(x_0, y_0)h = 1 + \frac{1}{4} = \frac{5}{4} = 1.25 \\
x_2 = \frac{1}{2} & y_2 = y_1 + f(x_1, y_1)h = \frac{5}{4} + \frac{5}{4} \cdot \frac{1}{4} = \frac{5^2}{4^2} = 1.5625 \\
x_3 = \frac{3}{4} & y_3 = y_2 + f(x_2, y_2)h = \frac{5^2}{4^2} + \frac{5^2}{4^2} \cdot \frac{1}{4} = \frac{5^3}{4^3} \approx 1.9531 \\
x_4 = 1 & y_4 = y_3 + f(x_3, y_3)h = \frac{5^3}{4^3} + \frac{5^3}{4^3} \cdot \frac{1}{4} = \frac{5^4}{4^4} \approx 2.4414
\end{array}$$

In particular, the approximation for $y(1)$ is $y_4 \approx 2.4414$.

Comment. Can you see that, if instead we start with $h = \frac{1}{n}$, then we similarly get $x_i = \frac{(n+1)^i}{n^i}$ for $i = 0, 1, \ldots, n$. In particular, $y(1) \approx y_n = \frac{(n+1)^n}{n^n} = \left(1 + \frac{1}{n}\right)^n \to e$ as $n \to \infty$. Do you recall how to derive this final limit?

Example 134. Python Let us implement Euler's method to redo and extend Example 133.

```
>>> def euler(f, x0, y0, xmax, n):
        h = (xmax - x0) / n
        ypoints = [y0]
        for i in range(n):
            y0 = y0 + f(x0,y0)*h
            x0 = x0 + h
            ypoints.append(y0)
        return ypoints
```

```
>>> def f_y(x, y):
        return y
```

If we choose the number of steps `n` to be 4 and `xmax` to be 1 (because we want $x_n = 1$), then the following matches exactly our computation in Example 133:

```
>>> euler(f_y, 0, 1, 1, 4)
```

```
[1, 1.25, 1.5625, 1.953125, 2.44140625]
```

As expected, increasing the number of steps provides better approximations to the exact solution $y(x) = e^x$ with $y(1) = e \approx 2.718$.

```
>>> euler(f_y, 0, 1, 1, 10)
```

```
[1, 1.1, 1.2100000000000002, 1.3310000000000002, 1.4641000000000002, 1.61051,
1.7715610000000002, 1.9487171, 2.1435888100000002, 2.357947691, 2.5937424601]
```

```
>>> euler(f_y, 0, 1, 1, 100)[-1]
```

```
2.704813829421526
```

If `ypoints` is a list, then its elements can be accessed as `ypoints[0]`, `ypoints[1]`, ... Moreover, we can access the last element as `ypoints[-1]`. For instance, above, we used `euler_e(f, 0, 1, 1, 100)[-1]` to get the last element of the 101 approximations $y_0, y_1, \ldots, y_{100}$. That last element is the approximation of $y(1) = e$.

The following convincingly illustrates that the error in Euler's method is $O(h)$.

```
>>> from math import e
>>> [euler(f_y, 0, 1, 1, 10**n)[-1] - e for n in range(6)]
```

```
[-0.7182818284590451, -0.124539368359045, -0.013467999037519274, -
0.0013578962231490799, -0.00013590163381849152, -1.3591284549807625e-05]
```

However, note that our computer had to work pretty hard to get the final approximation, because that entailed computing 10^5 values. We clearly need a higher order method in order to compute to higher accuracy.

Taylor methods

(Taylor method of order k) The following is an order k method for solving IVPs:

$$y_{n+1} = y_n + f(x_n, y_n)h + \frac{1}{2}f'(x_n, y_n)h^2 + \cdots + \frac{1}{k!}f^{(k-1)}(x_n, y_n)h^k$$

where $f^{(n)}(x, y)$ is short for $\frac{\mathrm{d}^n}{\mathrm{d}x^n}f(x, y(x))$ (expressed in terms of f and its partial derivatives).

For instance. $f'(x, y) = \frac{\mathrm{d}}{\mathrm{d}x}f(x, y(x)) = f_x(x, y) + f_y(x, y)y'(x) = f_x(x, y) + f_y(x, y)f(x, y)$

Especially for higher derivatives, it is easier to compute these for specific f. See next example.

Comment. As for Euler's method, being an order k method means that the method has a global error that is $O(h^k)$ (while the local truncation error is $O(h^{k+1})$; note that we can see this because we truncate the Taylor expansion of $y(x)$ after h^k so that the next term is $O(h^{k+1})$).

Example 135. Spell out the Taylor method of order 2 for numerically solving the IVP

$$y' = \cos(x)y, \quad y(0) = 1.$$

Solution. The Taylor method of order 2 is based on the Taylor expansion

$$y(x + h) = y(x) + y'(x)h + \frac{1}{2}y''(x)h^2 + O(h^3),$$

where we have a local truncation error of $O(h^3)$ so that the global error will be $O(h^2)$.

From the DE we know that $y'(x) = \cos(x)y$, which is $f(x, y)$. We differentiate this to obtain

$$\begin{aligned} y''(x) &= \frac{\mathrm{d}}{\mathrm{d}x}\cos(x)y = -\sin(x)y + \cos(x)y' = -\sin(x)y + \cos^2(x)\, y \\ &= (-\sin(x) + \cos^2(x))y, \end{aligned}$$

which is $f'(x, y)$. Hence, the Taylor method of order 2 takes the form:

$$\begin{aligned} y_{n+1} &= y_n + f(x_n, y_n)h + \frac{1}{2}f'(x_n, y_n)h^2 \\ &= y_n + \cos(x_n)y_n\, h + \frac{1}{2}((-\sin(x_n) + \cos^2(x_n))y_n)h^2 \end{aligned}$$

For any choice of h, we can therefore compute $(x_1, y_1), (x_2, y_2), \ldots$ starting with (x_0, y_0) by the above recursive formula combined with $x_{n+1} = x_n + h$.

Armin Straub
straub@southalabama.edu