Numerically solving a system of ordinary differential equations

Edward Brown

September 20, 2022

Background

A common numerical task is to integrate a system of first-order ordinary differential equations (ODEs). That is, given a system of equations

$$\frac{\mathrm{d}z}{\mathrm{d}t} = f(t, z) \tag{1}$$

with specified initial conditions $z(t = t_0)$, find $z(t)$ Here z is shorthand for an *array* of variables: $z(t) = \{z_0(t), z_1(t), z_2(t), \ldots\}$. Likewise, $f(t, z)$ is an array of known, specified functions: $f(t, z) = \{f_0(t, z), f_1(t, z), f_2(t, z), \ldots\}$.

An example is Newton's equation of motion,

$$\frac{\mathrm{d}^2\boldsymbol{r}}{\mathrm{d}t^2} = \frac{\boldsymbol{F}}{m}. \tag{2}$$

You may object that this is a second-order differential equation; but notice that if we define

$$\boldsymbol{v} = \frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}t}$$

then we can recast this single second-order differential equation into a system of two[1] first-order differential equations of the form (1):

$$\frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}t} = \boldsymbol{v} \tag{3}$$

$$\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \frac{\boldsymbol{F}}{m}. \tag{4}$$

[1] Strictly speaking, this is a system of *six* ODEs: three components of $\boldsymbol{r}$ and three components of $\boldsymbol{v}$.

An additional example is the system of first-order ordinary differential equations

$$\frac{\mathrm{d}z_0}{\mathrm{d}t} = 2\pi z_1 \tag{5}$$

$$\frac{\mathrm{d}z_1}{\mathrm{d}t} = -2\pi z_0 \tag{6}$$

with boundary conditions

$$z_0(t = 0) = 0, \qquad z_1(t = 0) = 1. \tag{7}$$

By taking the derivative of eq. (5) w.r.t. t,

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\mathrm{d}z_0}{\mathrm{d}t} = 2\pi\frac{\mathrm{d}z_1}{\mathrm{d}t} = -4\pi^2 z_0,$$

and applying the boundary conditions (7), we find the solution to equations (5)–(6):

$$z_0 = \sin(2\pi t), \tag{8}$$
$$z_1 = \cos(2\pi t). \tag{9}$$

We'll now show how to obtain an approximate numerical solution for this system of ODEs.

Methods

Suppose we know $z(t)$ at some point t and we wish to make a numerical estimate of z at a nearby point $t + h$. We have the values of z and we can compute

$$\left.\frac{\mathrm{d}z}{\mathrm{d}t}\right|_t = f(t, z).$$

The following Python function implements $f(t, z)$ for equations (5)–(6),

```
12 def f(t,z):
13     """
14     RHS of equation dz/dt = f(t,z)
15     """
16     # this makes an array of length 2, each element of which is zero
17     dzdt = np.zeros(2)
18 
19     dzdt[0] = 2.0*np.pi*z[1]
20     dzdt[1] = -2.0*np.pi*z[0]
21     return dzdt
```

Box 1 Functions

What is a function in the context of a program? Basically, a function is a self-contained group of statements that you name. In the listing above, for example, we gave our function the uninspired name `f`. A function may receive information, which is listed in the parentheses after the function name: `(t,z)`. Thus, the first line

```
def f(t,z):
```

says "bundle the following list of statements together and call it `f`. The statements expect as input two variables, called ARGUMENTS, which will be referred to in the function as `t` and `z`." This function then does three things: it creates an array `dzdt` of length `2`, sets the first element of this ar-

Box 1 continued

ray to `2.0*np.pi*z[1]`, and sets the second element to `-2.0*np.pi*z[0]`. The final statement

```
return dzdt
```

says "finish, and provide the value of `dzdt`" to whatever CALLED the function. Thus, for example, after defining the function, you could write

```
k = f(x,y)
```

where `x` and `y` are variables you had already defined. Python would interpret this as: "Execute the statements in the function `f`. In those statements, set the value of `t` to be that of `x` and the value of `z` to be that of `y`. After executing those statements, store the value of the function variable `dzdt` in `k` and carry on."

With the definition of a function f, we can compute the right-hand side of equation (1). Our problem can thus be stated as follows: given $z(t)$, construct an estimate for $z(t+h)$. If we can do this, then we can advance the solution stepwise from the initial condition $z(t = t_0)$. We'll now present three algorithms for doing so, starting with the least accurate. Our example will use $t = h = 0.12$; Fig. 1 shows the solution for $z_0(t)$ over this interval.

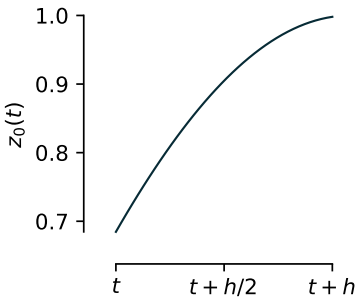

Figure 1: Solution (8) for z_0 in the system of equations (5)–(6) from $t = 0.12$ to $t + h = 0.24$.

Forward Euler

The first, and simplest, method goes back to Euler. Suppose at time t we know the solution $z(t)$ to eq. (1). We can expand $z(t)$ as a Taylor series about this point to obtain

$$z(t+h) = z(t) + h \left.\frac{\mathrm{d}z}{\mathrm{d}t}\right|_t + \frac{h^2}{2!} \left.\frac{\mathrm{d}^2 z}{\mathrm{d}t^2}\right|_t + \ldots$$

But $\mathrm{d}z/\mathrm{d}t = f(t,z)$ is a known function. So to $\mathcal{O}(h^2)$,

$$\begin{aligned} z(t+h) &\approx z(t) + h \left.\frac{\mathrm{d}z}{\mathrm{d}t}\right|_t . \\ &= z(t) + h\, f(t,z)|_t . \end{aligned} \tag{10}$$

Figure 2 displays a schematic of a forward Euler step. We first compute the slope $f(t,z)$ and then use this to extend the solution to a point $t+h$. By repeating this step over and over, we can march our solution along.

How accurate is this FORWARD EULER algorithm? From its definition, the error on each step comes from truncating the Taylor series

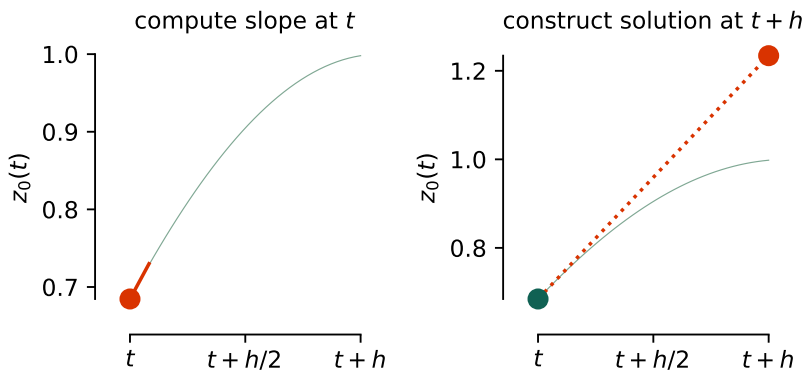

Figure 2: Schematic of a single forward Euler step.

and is $\mathcal{O}(h^2)$. What do we mean by this? For sufficiently small h, reducing the step by a factor of 2 should reduce the error in a single step by a factor of 4. Another way to put this is that the forward Euler reproduces $z(t)$ exactly if z is a linear function of t. Unfortunately, errors tend to compound with each step, and the smaller the stepsize, the more steps are required. To integrate over a fixed interval T takes T/h steps; if the error on a given step is $\mathcal{E} \sim \mathcal{O}(h^2)$, then the total integration error will be something like $T/h \times \mathcal{E} \sim \mathcal{O}(h)$. We therefore call this forward Euler method a **first-order** method. Reducing the stepsize by a factor of 2 only reduces the integration error by a factor of 2.

Second-order Runge-Kutta

As you can see from Fig. 2, the forward Euler algorithm is not very accurate. For the forward Euler method to be accurate, we must use a small step h and therefore we must take many steps. We could be more efficient if our step agreed with the Taylor series to a higher order in h.

An improved method is to take, using forward Euler, a step to the midpoint of the interval,

$$z_{\mathrm{P}}\left(t+\frac{h}{2}\right) = z(t) + \frac{h}{2} f[t, z(t)]. \tag{11}$$

Here z_{P} is our estimate of the solution at $t + h/2$.

We then use z_{P} to get a better estimate of f and use this corrected value of f for a step across the entire interval:

$$z(t+h) = z(t) + hf\left(t+\frac{h}{2}, z_{\mathrm{P}}\right). \tag{12}$$

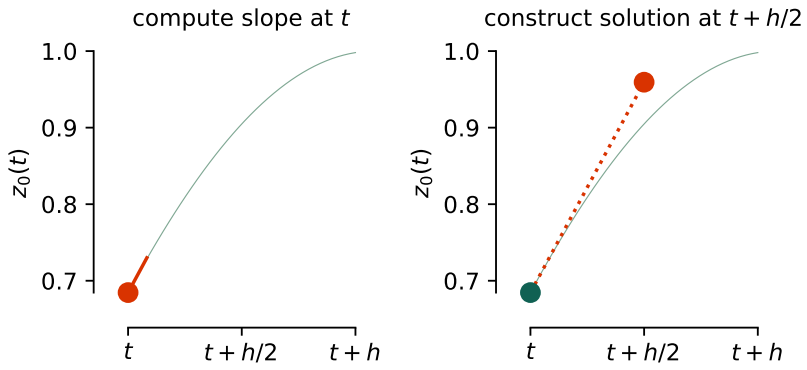

Figure 3: First stage of a second-order Runge-Kutta step.

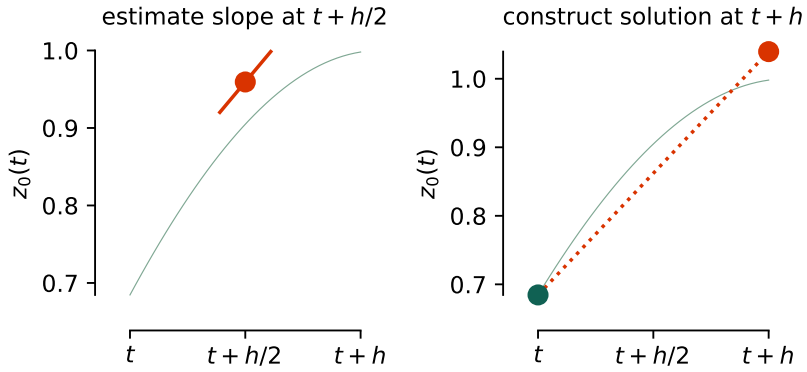

Figure 4: Second stage of a second-order Runge-Kutta step.

One can show that $z(t+h)$ agrees with the actual solution to $\mathcal{O}(h^3)$. When integrating over an interval T and taking T/h steps, the solution then has a global error $\sim \mathcal{O}(h^2)$. Equations (11) and (12) are known as the SECOND-ORDER RUNGE-KUTTA method. Note that if we reduce the stepsize by a factor of two, we expect the global truncation error to decrease by a factor of 4.

Fourth-order Runge-Kutta

An even better method is the classic FOURTH-ORDER RUNGE-KUTTA algorithm. In this method, the integration of z from t to $t+h$ is done in four steps.

First, a trial forward Euler step is taken to the midpoint $t+h/2$ and the solution estimated there (Fig. 5), just as in the second-order method.

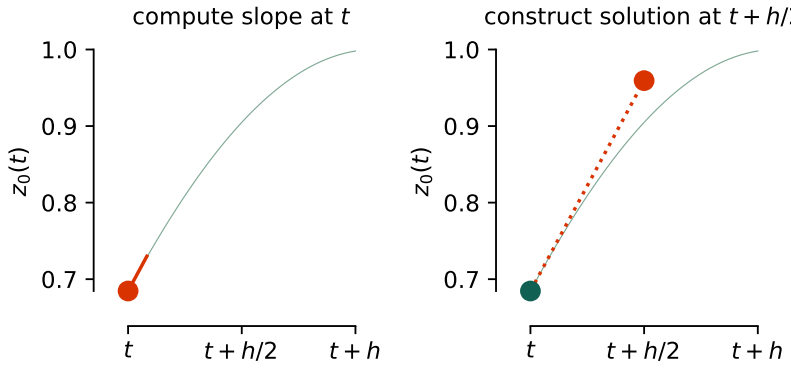

Figure 5: First stage of a fourth-order Runge-Kutta step.

This solution at the midpoint is used to make a better estimate of f, and a new step is taken, not to $t+h$, but rather from t just to the midpoint $t+h/2$ again (Fig. 6).

A new value of the slope f at the midpoint is then computed, and this value of f is then used to step across the full interval (Fig. 7).

The slope at the endpoint $t+h$ is then computed. At this point, we have computed the following four estimates for the value of $\mathrm{d}z/\mathrm{d}t$:

$$k_1 = f(t, z(t)) \tag{13}$$

$$k_2 = f\left(t + \frac{h}{2}, z(t) + \frac{h}{2}k_1\right) \tag{14}$$

$$k_3 = f\left(t + \frac{h}{2}, z(t) + \frac{h}{2}k_2\right) \tag{15}$$

$$k_4 = f\left(t + h, z(t) + hk_3\right). \tag{16}$$

The solution $z(t+h)$ is then constructed from a weighted sum of the

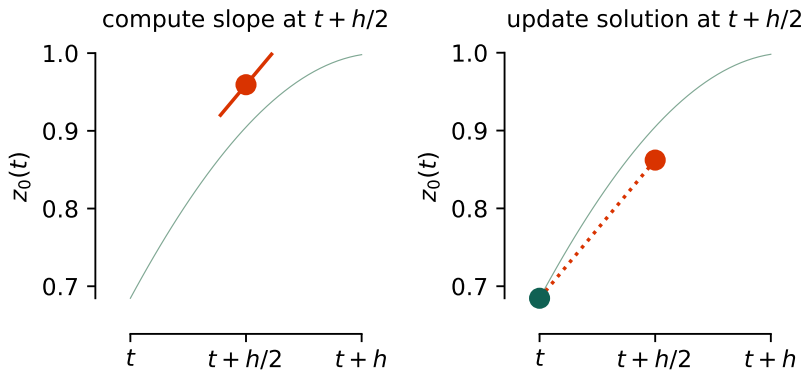

Figure 6: Second stage of a fourth-order Runge-Kutta step.

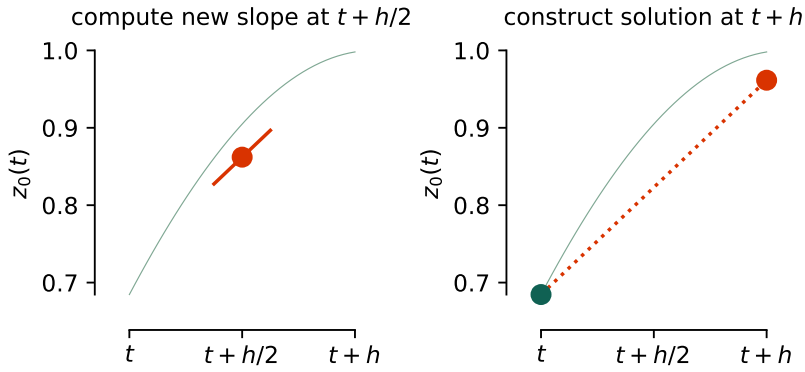

Figure 7: Third stage of a fourth-order Runge-Kutta step.

k_i (Fig. 8),

$$z(t+h) \approx z(t) + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right). \tag{17}$$

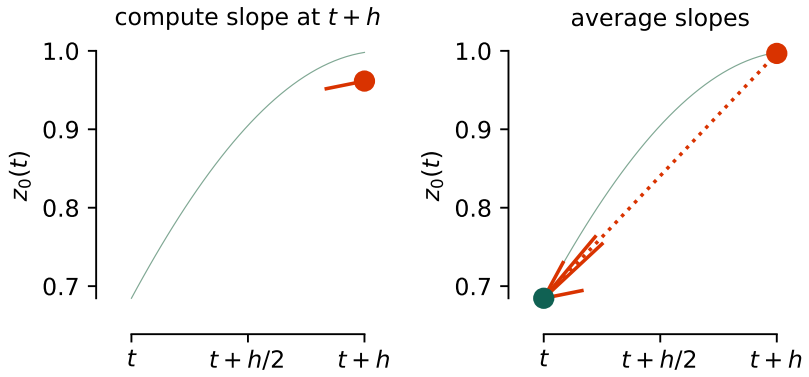

Figure 8: Fourth and final stage of a fourth-order Runge-Kutta step.

Notice that the k_i are not independent of one another: each one depends on the intermediate value of z computed in the previous step. The fourth-order Runge-Kutta scheme "sniffs" the behavior of $f(t,z)$ over the interval $(t, t+h)$ and constructs a weighted approximation for $\mathrm{d}z/\mathrm{d}t$. One can show that this method produces solutions with a global truncation error $\sim \mathcal{O}(h^4)$. That is, reducing the stepsize by a factor of 2 should reduce the error by a factor $2^4 = 16$.

References