Second-Order Runge-Kutta — Theory

January 29, 2025

Velocity from Acceleration — Recap

On January 24, we settled on and used the following update strategy:

$$t_{i+1} = t_i + \Delta t$$

$$x(t_{i+1}) = x(t_i) + \frac{v(t_i)+v(t_{i+1})}{2} \cdot \Delta t$$

$$v(t_{i+1}) = v(t_i) + a\left(t_i + \frac{\Delta t}{2}\right) \cdot \Delta t$$

I warned you that there was going to be a complication.

The Complication — Recap

In the formula

$$v(t_{i+1}) \approx v(t_i) + a\left(t_i + \frac{\Delta t}{2}\right) \cdot \Delta t$$

or with Newton's 2nd Law substituted in,

$$v(t_{i+1}) \approx v(t_i) + \frac{F\left(t_i + \frac{\Delta t}{2}\right)}{m} \cdot \Delta t$$

the complication is that in most physical systems

$$a(t) = \frac{F(t)}{m}$$

is not directly a given function of time. Far more commonly it is indirectly a function of time via the position and velocity:

$$a(t) = \frac{F(x(t), v(t))}{m}$$

Sometimes the force is both directly and indirectly a function of time, in which case we would write:

$$a(t) = \frac{F(t, x(t), v(t))}{m}$$

A Simplification

So that we don't overdo the complexity right off the bat, let's focus on a simpler and quite common situation,

$$a(t) = \frac{F(x(t))}{m}$$

In other words, let's leave out the possibility that F depends directly on the time, and also leave out the possibility that F depends on the velocity. We are only allowing F to depend indirectly on the time via the position.

Leaving out velocity-dependent forces means that systems with friction, drag, or magnetism aren't ones we can yet work on. Leaving out time-dependence means that we can't work on systems that have externally-controlled forces, like the force of a person periodically pushing on the back of a kid on a swing-set.

However, a lot of interesting systems can be idealized as frictionless — for example, the motion of a mass on a spring, or the motion of a planet around the Sun — and many, if not most, interesting systems do not have externally applied and controlled forces.

A Chicken-and-Egg Problem

Here are our three update equations, but rewritten to emphasize that the only way that the acceleration a depends on t is via $x(t)$:

$$t_{i+1} = t_i + \Delta t$$

$$x(t_{i+1}) = x(t_i) + \frac{v(t_i)+v(t_{i+1})}{2} \cdot \Delta t$$

$$v(t_{i+1}) = v(t_i) + a\left(x\left(t_i + \frac{\Delta t}{2}\right)\right) \cdot \Delta t$$

Notice that I am still using the trapezoid approximation to update x and I am still using the midpoint time in $a(x(t))$ to update v, exactly as we did above.

Do you see the chicken-and-egg problem!?

We can't compute the updated $x(t_{i+1})$ without knowing $v(t_{i+1})$

We can't compute the updated $v(t_{i+1})$ without knowing a at the midpoint time, which in turn requires knowing x at the midpoint time, $x\left(t_i + \frac{\Delta t}{2}\right)$.

A Worse Chicken-and-Egg Problem

We might try a trapezoid approximation for the calculation of $v(t_{i+1})$, but that makes the chicken-and-egg problem if anything a little more poignant:

$$t_{i+1} = t_i + \Delta t$$

$$x(t_{i+1}) = x(t_i) + \frac{v(t_i)+v(t_{i+1})}{2} \cdot \Delta t$$

$$v(t_{i+1}) = v(t_i) + \frac{a(x(t_i))+a(x(t_{i+1}))}{2} \cdot \Delta t$$

Good luck telling Mathematica how to do that pile of circular reasoning.

A Way Out

Imagine that instead of using midpoint or trapezoid, we begin by making the simple left-hand approximation for $x(t_{i+1})$, and we'll even give it its own symbol to emphasize that it is the dirt-simple left-hand approximation:

$$x^*(t_{i+1}) \approx x(t_i) + v(t_i)\,\Delta t$$

Definitely we can do that without encountering any circular reasoning because the right-hand side only contains $x(t_i)$ and $v(t_i)$. Then we make a version of the trapezoid approximation to $a_{i\to i+1,\text{avg}}$, by trying

$$a_{t_i\to t_{i+1},\text{avg}} \approx \frac{a(x(t_i))+a(x^*(t_{i+1}))}{2}$$

Then we put that version of the trapezoid approximation into the update equation for $v(t_{i+1})$:

$$v(t_{i+1}) = v(t_i) + \frac{a(x(t_i))+a(x^*(t_{i+1}))}{2} \cdot \Delta t$$

Still no circular reasoning! Finally, we use the trapezoid approximation again to get a more sophisticated approximation to $x(t_{i+1})$:

$$x(t_{i+1}) = x(t_i) + \frac{v(t_i)+v(t_{i+1})}{2} \cdot \Delta t$$

You might have a feeling that this is *ad hoc,* but at least the chicken-and-egg problem has been evaded.

This procedure and generalizations of it have stood the test of time. They were studied by Carl Runge and Martin Wilhelm Kutta around 1900 and this version and other Runge-Kutta procedures that we will soon get to are still widely used.

Summary — Second-Order Runge-Kutta

We were motivated to try this update procedure when F or a depended on time indirectly via the position x:

$$t_{i+1} = t_i + \Delta t$$

$$x(t_{i+1}) = x(t_i) + \frac{v(t_i)+v(t_{i+1})}{2} \cdot \Delta t$$

$$v(t_{i+1}) = v(t_i) + \frac{a(x(t_i))+a(x(t_{i+1}))}{2} \cdot \Delta t$$

We realized that this system suffered a chicken-and-egg problem.

We decided to try this update procedure instead:

$$t_{i+1} = t_i + \Delta t$$

$$x^*(t_{i+1}) = x(t_i) + v(t_i)\,\Delta t$$

$$v(t_{i+1}) = v(t_i) + (a(x(t_i)) + a(x^*(t_{i+1}))) \cdot \frac{\Delta t}{2}$$

$$x(t_{i+1}) = x(t_i) + (v(t_i) + v(t_{i+1})) \cdot \frac{\Delta t}{2}$$

This update procedure is known as Second-Order Runge-Kutta — although I have seen variations of this procedure given the same name. It is what we are going to use shortly to get the motion of a mass on a spring.

A Note on Sources

At some point in the future, you may want a reference that gets to a sophisticated punch-line much more quickly. I am deliberately building up a sophisticated procedure one modest step at a time. More advanced references cut straight to the chase. For example, what I used to double-check that I was actually presenting Second-Order Runge-Kutta — and not some similar but not identical procedure — was Section 3.1 of Gregory Fasshauer's course notes for IIT Math 472, which are freely-available on-line at https://www.math.iit.edu/~fass/472Notes.pdf.

A more introductory book that I first learned Runge-Kutta from was Alejandro L. Garcia, *Numerical Methods for Physics,* Prentice-Hall, 1994. The code in it — as in many other numerical methods books — is written in MATLAB, which is still popular among engineers. Nicholas J. Giordano, *Computational Physics,* Prentice-Hall, 1997, is another popular textbook. The code listings in it are in BASIC.