Solving Differential Equations Numerically

Differential equations are ubiquitous in Physics since the laws of nature often take on a simple form when expressed in terms of infinitesimal changes of the variables. Many differential equations do not have a "simple" analytic solution in terms of well known functions, so numerical methods need to be used to solve them. Here we present three similar methods: Euler's algorithm and two Runga-Kutta algorithms. These fall under the catagory of finite-difference methods for solving differential equations.

We first develop these algorithm's for a single function of one variable, $x(t)$, that satifies a first order differential equation. Let $x(t)$ satisfy the following differential equation:

$$\frac{dx}{dt} = f(t, x) \tag{1}$$

The approach in finite-difference methods is to "discretize" the t-variable. The easiest way to quantize t is to use equal spacing, δ, between the values. That is, the chosen t values are indexed by the integer i: $t_i = i * \delta$, where i goes from zero to some maximum value $imax$. The continuous function $x(t)$ becomes an array $x[i] = x(t_i)$. The continuous differential equation is transformed into an algebraic equation containing the $x(t_i)$.

Euler Algorithm

In the Euler Algorithm, the derivative is replaced by its finite difference approximation. Defining $x_i \equiv x(t_i)$, we have

$$\frac{dx}{dt} \approx \frac{x_{i+1} - x_i}{\delta} \tag{2}$$

Substituting into the differential equation gives:

$$\frac{x_{i+1} - x_i}{\delta} \approx f(t_i, x_i) \tag{3}$$

Solving for x_{i+1} yields

$$x_{i+1} \approx x_i + \delta f(t_i, x_i) \tag{4}$$

If the initial condition is such that x_0 is known, the above equation can be used to find x_1. Then the equation can be used again to find x_2, etc.

The Euler algorithm is equivalent to including only the first term in the Taylor expansion. The Taylor expansion for $x(t + \delta)$ about $x(t)$ is given by

$$\begin{aligned} x(t+\delta) &= x(t) + \delta x'(t) + \frac{\delta^2}{2} x''(t) + \cdots \\ x(t+\delta) &= x(t) + \delta f(t,x) + \frac{\delta^2}{2} f'(t,x) + \cdots \end{aligned}$$

Neglecting terms or order δ^2 or greater gives the approximation

$$x(t+\delta) \approx x(t) + \delta f(t,x) \tag{5}$$

which is the Euler algorithm.

The following is an example code for the Euler algorithm:

```
t = 0;
x1 = initial position;
for (i = 1; i < imax; i++)
{
x2 = x1 + del*f(t,x1);
t = t + del;
x1 = x2;
}
```

Note that in this code we don't save all the $x(t_i)$, but just the last ones. The example above is for a first-order differential equation. However, in physics we often encounter second-order differential equations. However, one can turn a second order differential equation into two first order ones. We demonstrate this with Newton's force law equation in one dimension. The equation

$$\frac{d^2x}{dt^2} = \frac{F(t,x,v_x)}{m} \tag{6}$$

where $F(t,x,v_x)$ is the force and $v_x = dx/dt$ is the velocity of the particle. If we consider $v_x(t)$ as a function of time, we have

$$\begin{aligned} \frac{dv_x}{dt} &= F(t,x,v_x) \\ \frac{dx}{dt} &= v_x \end{aligned}$$

We can now apply the Euler algorithm to each equation. Below is an example code to handle this situation:

```
t = 0;
x1 = initial position;
v1 = initial velocity;
for $(i = 1; i < imax; i++)$
{
v2 = v1 + del*F(t,x1,v1)/m;
x2 = x1 + del*v1;
t = t + del;
v1 = v2;
x1 = x2;
}
```

There are some variations to the Euler algorithm for second order differential equations. One variation is to use v2 instead of v1 in the equation for x2:

```
t = 0;
x1 = initial position;
v1 = initial velocity;
for $(i = 1; i < imax; i++)$
{
v2 = v1 + del*F(t,x1,v1)/m;
x2 = x1 + del*v2;
t = t + del;
v1 = v2;
x1 = x2;
}
```

This simple change yields more accurate results if the motion is oscillatory.

Another variation for a second order differential equation is to use the Euler approach for the second derivative. The second derivative at x_i can be approximated as:

$$\frac{d^2x}{dt^2} \approx \frac{(x_{i+1} - x_i)/\delta - (x_i - x_{i-1})/\delta}{\delta} \tag{7}$$

Collecting terms gives

$$\frac{d^2x}{dt^2} \approx \frac{x_{i+1} + x_{i-1} - 2x_i}{\delta^2} \tag{8}$$

Substituting this finite-difference formula for the second derivative into Newton's law of motion yields

$$\frac{x_{i+1} + x_{i-1} - 2x_i}{\delta^2} \approx \frac{F(t_i, x_i, v_i)}{m} \tag{9}$$

From this equation we can solve for x_{i+1}:

$$x_{i+1} \approx 2x_i - x_{i-1} + \delta^2 \frac{F(t_i, x_i, v_i)}{m} \tag{10}$$

If F does not depend on the velocity, v_i, then this form is particularly useful.

Runge-Kutta Methods

The Euler algorithm is conceptually easy to understand. To improve accuracy, one can decrease the step size and increase the number of iteration points. However, the computation time increases with the number of iteration points. As you might guess, there are a more efficient algorithms for increasing accuracy. We will describe one such method first, the Second-Order Runge-Kutta algorithm, then "prove" why it is more accurate.

We want the best estimate of $x(t+\delta)$, with the knowledge of $x(t)$ and $f(t,x)$. The Euler algorithm uses the first two terms of the Taylor expansion, with the derivative evaluated at x at time t. From the mean-value theorum, there is a value of $x = x'$ and $t = t'$ such that

$$x(t+\delta) = x(t) + \delta f(x', t') \tag{11}$$

A simple improvement over the Euler method is to choose values $x' = x_m$, where x_m is half way between $x(t)$ and $x(t+\delta)$. We don't know x_m exactly, but we can use the Euler algorithm to approximate it. The two step process would be as follows:

$$\begin{aligned}
x_m &\approx x(t) + (\delta/2) * f(x(t), t) \\
t_m &\approx t + \delta/2 \\
x(t+\delta) &\approx x(t) + \delta * f(x_m, t_m)
\end{aligned}$$

Note that x_m is determined via the Euler algorithm, but with a step size of $\delta/2$. The results are still approximate, but as we will show are accurate to order δ^2. The above method is a type of second-order Runge-Kutta algorithm. An example code for the algorithm is:

```
t = 0;
x1 = initial position;
for $(i = 1; i < imax; i++)$
{
xm = x1 + del*f(t,x1)/2;
tm = t + del/2;
x2 = x1 + del*f(tm,xm);
t = t + del;
x1 = x2;
}
```

For a second order differential equation, this algorithm will need to be applied to x and dx/dt. In the case of Newton's force law, the code would look like:

```
t = 0;
x1 = initial position;
v1 = initial velocity;
for $(i = 1; i < imax; i++)$
{
vm = v1 + del*F(t,x1,v1)/m/2;
xm = x1 + del*v1/2;
tm = t + del/2;
v2 = v1 + del*F(tm,xm,vm)/m;
x2 = x1 + del*vm;
t = t + del;
v1 = v2;
x1 = x2;
}
```

Although there are nearly twice as many lines of code for each step δ as the Euler algorithm, this Runge-Kutta algorithm can converge somewhat faster than the Euler algorithm. In the next section we derive the general form for the second-order Runge-Kutta algorithm.

Development of Runge-Kutta Forms

The approach in second-order Runge-Kutta methods is to use only first derivative information to best estimate the change in x(t) for one step size. A good method to accomplish this is to use a combination of two terms: one with derivative information at the start of the interval $f(t_0, x_0)$ and a second term which uses derivative information somewhere in the interval $f(t_0+\alpha\delta, x_0+\beta\delta f(t_0, x_0)$. Here α and β are constants, and the interval starts at time t_0 and $x_0 \equiv x(t_0)$. The ansatz for this approach is

$$x(t_0+\delta) = x(t_0) + w_1\delta f(t_0, x_0) + w_2\delta f(t_0+\alpha\delta, x_0+\beta\delta f(t_0, x_0)) \tag{12}$$

where w_1 is the weighting for the derivative at the start of the interval, and w_2 is the weighting for the derivative somewhere in the interval. The argument x_0 means $x(t_0)$. Note, that the Euler algorithm is obtained if $w_1 = 1$ and $w_2 = 0$.

To compare with the Taylor expansion, we can expand the right side of the equation about $x(t_0)$. Expanding $f(t_0+\alpha\delta, x_0+\beta\delta f(t_0, x_0)$ we have

$$f(t_0+\alpha\delta, x_0+\beta\delta f(t_0, x_0) = f(t_0, x_0) + \alpha\delta\frac{\partial f}{\partial t} + \beta\delta f\frac{\partial f}{\partial x} + \cdots \tag{13}$$

to order δ^2. Note that the derivative are evaluated at $x = x_0$ and $t = t_0$. Substituting into Eq. 12, yields

$$\begin{aligned} x(t_0+\delta) &= x(t_0) + w_1\delta f(t_0, x_0) + w_2\delta(f(t_0, x_0) + \alpha\delta\frac{\partial f}{\partial t} + \beta\delta f\frac{\partial f}{\partial x} + \cdots) && (14) \\ x(t_0+\delta) &= x(t_0) + (w_1+w_2)\delta f(t_0, x_0) + \alpha w_2\delta^2\frac{\partial f}{\partial t} + w_2\beta\delta^2 f\frac{\partial f}{\partial x} + \cdots && (15) \end{aligned}$$

to order δ^2. We need to compare this expression with that of the Taylor expansion of $x(t)$. The Taylor expansion of $x(t_0+\delta)$ about $x(t_0)$ is given by

$$x(t_0+\delta) = x(t_0) + \delta\frac{dx}{dt} + \frac{\delta^2}{2}\frac{d^2x}{dt^2} + \cdots \tag{16}$$

to order δ^2. The first derivative of $x(t)|_{t_0}$ equals $f(t_0, x_0)$. The second derivative of $x(t)$ with respect to t equals the first derivative of f with respect to t: $d^2x/dt^2 = df/dt$. f has an implicit and explicite time dependance:

$$\begin{aligned} \frac{df}{dt} &= \frac{\partial f}{\partial t} + \frac{\partial f}{\partial x}\frac{dx}{dt} \\ \frac{d^2x}{dt^2} &= \frac{\partial f}{\partial t} + f\frac{\partial f}{\partial x} \end{aligned}$$

Substituting this expression for the second derivative into the Taylor expansion yields:

$$x(t_0 + \delta) = x(t_0) + \delta f(x_0, t_0) + \frac{\delta^2}{2}\frac{\partial f}{\partial t} + \frac{\delta^2}{2} f \frac{\partial f}{\partial x} + \cdots \tag{17}$$

to order δ^2, where we have used $dx/dt|_{t_0} = f(x_0, t_0)$. Comparing Eq. 15 with Eq. 17, we see that if w_1, w_2, α and β satisfy the following conditions:

$$w_1 + w_2 = 1 \qquad \alpha w_2 = 1/2 \qquad \beta w_2 = 1/2$$

then the expansion agrees with the Taylor expansion to order δ^2. This is one order of δ better than the Euler algorithm.

There is not a unique solution to these three equations. They do constrain α to equal β, and $w_2 = 1 - w_1$. However, w_1 can take on any value between 0 and 1. One of the simplest choices is $w_1 = 0$. This choice gives $w_2 = 1$, and $\alpha = \beta = 1/2$, and is the choice we presented in these notes:

$$x(t_0 + h) = x(t_0) + \delta f(t_0 + \delta/2, x_0 + (\delta/2) f(t_0, x_0)) \tag{18}$$

This equation is more clearly expressed as we did before:

$$\begin{aligned} x_m &\approx x(t_0) + (\delta/2) * f(t_0, x_0)) \\ t_m &\approx t_0 + \delta/2 \\ x(t_0 + \delta) &\approx x(t_0) + \delta * f(t_m, x_m) \end{aligned}$$

The Taylor expansion analysis allows us to determine the accuracy of the algorithms. For the Euler algorithm, $x(t + \delta)$ is exact up to first order in δ. That is, the errors decrease with order δ^2. If the step size δ is divided by N, then the error for $x(t + \delta/N)$ is $1/N^2$ as large as the error for $x(t + \delta)$. However, with $1/N$ the step size, one needs N steps to reach $x(t + \delta)$, so the overall gain is a decrease of $1/N$ in the error.

For the second-order Runge-Kutta algorithm, the errors decrease with order δ^3. If δ is decreased by a factor of N, then the error for $x(t + \delta/N)$ decreases by a factor of $1/N^3$ from the error of $x(t + \delta)$. Since N steps are needed, the overall gain is a decrease of $1/N^2$ in the error. However, since the second-order Runge-Kutta requires 5/3 the computing time per δ/N step as the Euler algorithm, there is often only a marginal gain in accuracy per computing time.

Fourth-Order Runge-Kutta algorithm

The most common Runge-Kutta algorithm used in for "initial value" differential equations is the 4^{th}-order Runge-Kutta. It gives great accuracy per computing time. Below, we state the algorithm without derivation:

$$\begin{aligned} K_1 &= \delta f(t,x) \\ K_2 &= \delta f(t+\delta/2, x+K_1/2) \\ K_3 &= \delta f(t+\delta/2, x+K_2/2) \\ K_4 &= \delta f(t+\delta, x+K_3) \\ x(t+\delta) &= x(t) + (K_1+2K_2+2K_3+K_4)/6 \end{aligned}$$

For the fourth-order Runge-Kutta algorithm, the errors decrease with order δ^5, and there ends up being a gain in accuracy per computing time.