Simple ODE Solvers — Derivation

These notes provide derivations of some simple algorithms for generating, numerically, approximate solutions to the initial value problem

\[ y'(t) = f(t, y(t)) \]
\[ y(t_0) = y_0 \]

Here \( f(t, y) \) is a given function, \( t_0 \) is a given initial time and \( y_0 \) is a given initial value for \( y \). The unknown in the problem is the function \( y(t) \). We start with

**Euler’s Method**

Our goal is to determine (approximately) the unknown function \( y(t) \) for \( t \geq t_0 \). We are told explicitly the value of \( y(t_0) \), namely \( y_0 \). Using the given differential equation, we can also determine exactly the instantaneous rate of change of \( y \) at time \( t_0 \).

\[ y'(t_0) = f(t_0, y(t_0)) = f(t_0, y_0) \]

If the rate of change of \( y(t) \) were to remain \( f(t_0, y_0) \) for all time, then \( y(t) \) would be exactly \( y_0 + f(t_0, y_0)(t - t_0) \). The rate of change of \( y(t) \) does not remain \( f(t_0, y_0) \) for all time, but it is reasonable to expect that it remains close to \( f(t_0, y_0) \) for \( t \) close to \( t_0 \). If this is the case, then the value of \( y(t) \) will remain close to \( y_0 + f(t_0, y_0)(t - t_0) \) for \( t \) close to \( t_0 \). So pick a small number \( h \) and define

\[ t_1 = t_0 + h \]
\[ y_1 = y_0 + f(t_0, y_0)(t_1 - t_0) = y_0 + f(t_0, y_0)h \]

By the above argument

\[ y(t_1) \approx y_1 \]

Now we start over. We now know the approximate value of \( y \) at time \( t_1 \). If \( y(t_1) \) were exactly \( y_1 \), then the instantaneous rate of change of \( y \) at time \( t_1 \) would be exactly \( f(t_1, y_1) \). If this rate of change were to persist for all future time, \( y(t) \) would be exactly \( y_1 + f(t_1, y_1)(t - t_1) \). As \( y(t_1) \) is only approximately \( y_1 \) and as the rate of change of \( y(t) \) varies with \( t \), the rate of change of \( y(t) \) is only approximately \( f(t_1, y_1) \) and only for \( t \) near \( t_1 \). So we approximate \( y(t) \) by \( y_1 + f(t_1, y_1)(t - t_1) \) for \( t \) bigger than, but close to, \( t_1 \). Defining

\[ t_2 = t_1 + h = t_0 + 2h \]
\[ y_2 = y_1 + f(t_1, y_1)(t_2 - t_1) = y_1 + f(t_1, y_1)h \]
we have

\[ y(t_2) \approx y_2 \]

We just repeat this argument ad infinitum. Define, for \( n = 0, 1, 2, 3, \ldots \)

\[ t_n = t_0 + nh \]

Suppose that, for some value of \( n \), we have already computed an approximate value \( y_n \) for \( y(t_n) \). Then the rate of change of \( y(t) \) for \( t \) close to \( t_n \) is \( f(t, y(t)) \approx f(t_n, y(t_n)) \approx f(t_n, y_n) \) and, again for \( t \) close to \( t_n \), \( y(t) \approx y_n + f(t_n, y_n)(t - t_n) \). Hence

\[ y(t_{n+1}) \approx y_{n+1} = y_n + f(t_n, y_n)h \quad \text{(Eul)} \]

This algorithm is called **Euler’s Method**. The parameter \( h \) is called the **step size**.

Here is a table applying a few steps of Euler’s method to the initial value problem

\[
\begin{align*}
y' &= -2t + y \\
y(0) &= 3
\end{align*}
\]

with step size \( h = 0.1 \). For this initial value problem

\[
\begin{align*}
f(t, y) &= -2t + y \\
t_0 &= 0 \\
y_0 &= 3
\end{align*}
\]

Of course this initial value problem has been chosen for illustrative purposes only. The exact solution is, easily, \( y(t) = 2 + 2t + e^t \).

<table>
<thead>
<tr>
<th>( n )</th>
<th>( t_n )</th>
<th>( y_n )</th>
<th>( f(t_n, y_n) = -2t_n + y_n )</th>
<th>( y_{n+1} = y_n + f(t_n, y_n) \ast h )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>3.000</td>
<td>(-2 \ast 0.0 + 3.000 = 3.000)</td>
<td>(3.000 + 3.000 \ast 0.1 = 3.300)</td>
</tr>
<tr>
<td>1</td>
<td>0.1</td>
<td>3.300</td>
<td>(-2 \ast 0.1 + 3.300 = 3.100)</td>
<td>(3.300 + 3.100 \ast 0.1 = 3.610)</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>3.610</td>
<td>(-2 \ast 0.2 + 3.610 = 3.210)</td>
<td>(3.610 + 3.210 \ast 0.1 = 3.931)</td>
</tr>
<tr>
<td>3</td>
<td>0.3</td>
<td>3.931</td>
<td>(-2 \ast 0.3 + 3.931 = 3.331)</td>
<td>(3.931 + 3.331 \ast 0.1 = 4.264)</td>
</tr>
<tr>
<td>4</td>
<td>0.4</td>
<td>4.264</td>
<td>(-2 \ast 0.4 + 4.264 = 3.464)</td>
<td>(4.264 + 3.464 \ast 0.1 = 4.611)</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>4.611</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Improved Euler’s Method

Euler’s method is one algorithm which generates approximate solutions to the initial value problem
\[ y'(t) = f(t, y(t)) \]
\[ y(t_0) = y_0 \]
In applications, \( f(t, y) \) is a given function and \( t_0 \) and \( y_0 \) are given numbers. The function \( y(t) \) is unknown. Denote by \( \varphi(t) \) the exact solution for this initial value problem. In other words \( \varphi(t) \) is the function that obeys
\[ \varphi'(t) = f(t, \varphi(t)) \]
\[ \varphi(t_0) = y_0 \]
exactly.

Fix a step size \( h \) and define \( t_n = t_0 + nh \). We now derive another algorithm that generates approximate values for \( \varphi \) at the sequence of equally spaced time values \( t_0, t_1, t_2, \cdots \). We shall denote the approximate values \( y_n \) with
\[ y_n \approx \varphi(t_n) \]
By the fundamental theorem of calculus and the differential equation, the exact solution obeys
\[ \varphi(t_{n+1}) = \varphi(t_n) + \int_{t_n}^{t_{n+1}} \varphi'(t) \, dt \]
\[ = \varphi(t_n) + \int_{t_n}^{t_{n+1}} f(t, \varphi(t)) \, dt \]
Fix any \( n \) and suppose that we have already found \( y_0, y_1, \cdots, y_n \). Our algorithm for computing \( y_{n+1} \) will be of the form
\[ y_{n+1} = y_n + \text{approximate value for } \int_{t_n}^{t_{n+1}} f(t, \varphi(t)) \, dt \]
In fact Euler’s method is of precisely this form. In Euler’s method, we approximate \( f(t, \varphi(t)) \) for \( t_n \leq t \leq t_{n+1} \) by the constant \( f(t_n, y_n) \). Thus
\[ \text{Euler’s approximate value for } \int_{t_n}^{t_{n+1}} f(t, \varphi(t)) \, dt = \int_{t_n}^{t_{n+1}} f(t_n, y_n) \, dt = f(t_n, y_n)h \]
The area of the complicated region \( 0 \leq y \leq f(t, \varphi(t)), t_n \leq t \leq t_{n+1} \) (represented by the shaded region under the parabola in the left half of the figure below) is approximated by
the area of the rectangle \( 0 \leq y \leq f(t_n, y_n), \ t_n \leq t \leq t_{n+1} \) (the shaded rectangle in the right half of the figure below).

Our second algorithm, the improved Euler’s method, gets a better approximation by attempting to approximate by the trapezoid on the right below rather than the rectangle on the right above. The exact area of this trapezoid is the length \( h \) of the base multiplied by the average, \( \frac{1}{2}[f(t_n, \varphi(t_n)) + f(t_{n+1}, \varphi(t_{n+1}))] \), of the heights of the two sides. Of course we do not know \( \varphi(t_n) \) or \( \varphi(t_{n+1}) \) exactly. Recall that we have already found \( y_0, \ldots, y_n \) and are in the process of finding \( y_{n+1} \). So we already have an approximation for \( \varphi(t_n) \), namely \( y_n \), but not for \( \varphi(t_{n+1}) \). Improved Euler uses

\[
\varphi(t_{n+1}) \approx \varphi(t_n) + \varphi'(t_n)h \approx y_n + f(t_n, y_n)h
\]

in approximating \( \frac{1}{2}[f(t_n, \varphi(t_n)) + f(t_{n+1}, \varphi(t_{n+1}))] \). Altogether

Improved Euler’s approximate value for

\[
\int_{t_n}^{t_{n+1}} f(t, \varphi(t)) \, dt
\]

is

\[
= \frac{1}{2} \left[ f(t_n, y_n) + f(t_{n+1}, y_n + f(t_n, y_n)h) \right] h
\]

so that the improved Euler’s method algorithm is

\[
y(t_{n+1}) \approx y_{n+1} = y_n + \frac{1}{2} \left[ f(t_n, y_n) + f(t_{n+1}, y_n + f(t_n, y_n)h) \right] h \quad \text{(ImpEul)}
\]

Here are the first two steps of the improved Euler’s method applied to

\[
y' = -2t + y \quad y(0) = 3
\]
with \( h = 0.1 \). In each step we compute \( f(t_n, y_n) \), followed by \( y_n + f(t_n, y_n)h \), which we denote \( \tilde{y}_{n+1} \), followed by \( f(t_{n+1}, \tilde{y}_{n+1}) \), followed by \( y_{n+1} = y_n + \frac{1}{2}[f(t_n, y_n) + f(t_{n+1}, \tilde{y}_{n+1})]h \).

\[
t_0 = 0 \quad y_0 = 3 \quad \Rightarrow \quad f(t_0, y_0) = -2 \times 0 + 3 = 3
\]
\[
\quad \Rightarrow \quad \tilde{y}_1 = 3 + 3 \times 0.1 = 3.3
\]
\[
\quad \Rightarrow \quad f(t_1, \tilde{y}_1) = -2 \times 0.1 + 3.3 = 3.1
\]
\[
\quad \Rightarrow \quad y_1 = 3 + \frac{1}{2}[3 + 3.1] \times 0.1 = 3.305
\]
\[
t_1 = 0.1 \quad y_1 = 3.305 \quad \Rightarrow \quad f(t_1, y_1) = -2 \times 0.1 + 3.305 = 3.105
\]
\[
\quad \Rightarrow \quad \tilde{y}_2 = 3.305 + 3.105 \times 0.1 = 3.6155
\]
\[
\quad \Rightarrow \quad f(t_2, \tilde{y}_2) = -2 \times 0.2 + 3.6155 = 3.2155
\]
\[
\quad \Rightarrow \quad y_2 = 3.305 + \frac{1}{2}[3.105 + 3.2155] \times 0.1 = 3.621025
\]

Here is a table which gives the first five steps.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( t_n )</th>
<th>( y_n )</th>
<th>( f(t_n, y_n) )</th>
<th>( \tilde{y}_{n+1} )</th>
<th>( f(t_{n+1}, \tilde{y}_{n+1}) )</th>
<th>( y_{n+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>3.000</td>
<td>3.000</td>
<td>3.300</td>
<td>3.100</td>
<td>3.305</td>
</tr>
<tr>
<td>1</td>
<td>0.1</td>
<td>3.305</td>
<td>3.105</td>
<td>3.616</td>
<td>3.216</td>
<td>3.621</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>3.621</td>
<td>3.221</td>
<td>3.943</td>
<td>3.343</td>
<td>3.949</td>
</tr>
<tr>
<td>3</td>
<td>0.3</td>
<td>3.949</td>
<td>3.349</td>
<td>4.284</td>
<td>3.484</td>
<td>4.291</td>
</tr>
<tr>
<td>4</td>
<td>0.4</td>
<td>4.291</td>
<td>3.491</td>
<td>4.640</td>
<td>3.640</td>
<td>4.647</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>4.647</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Runge-Kutta Method

The Runge-Kutta algorithm is similar to the Euler and improved Euler methods in that it also uses, in the notation of the last section,

\[
y_{n+1} = y_n + \text{approximate value for } \int_{t_n}^{t_{n+1}} f(t, \varphi(t)) \, dt
\]

But rather than approximating \( \int_{t_n}^{t_{n+1}} f(t, \varphi(t)) \, dt \) by the area of a rectangle, as does Euler, or by the area of a trapezoid, as does improved Euler, it approximates by the area under a parabola. That is, it uses Simpson’s rule. According to Simpson’s rule (if you don’t know Simpson’s rule, just take my word for it)

\[
\int_{t_n}^{t_n+h} f(t, \varphi(t)) \, dt \approx \frac{h}{6} \left[ f(t_n, \varphi(t_n)) + 4f(t_n + \frac{h}{2}, \varphi(t_n + \frac{h}{2})) + f(t_n + h, \varphi(t_n + h)) \right]
\]

As we don’t know \( \varphi(t_n) \), \( \varphi(t_n + \frac{h}{2}) \) or \( \varphi(t_n + h) \), we have to approximate them as well. The Runge-Kutta algorithm, incorporating all these approximations, is
Here are the first two steps of the Runge-Kutta algorithm applied to

\[ y' = -2t + y \quad y(0) = 3 \]

with \( h = 0.1 \).

\[
\begin{align*}
  t_0 &= 0 \quad y_0 = 3 \\
  \implies k_{0,1} &= f(0, 3) = -2 \times 0 + 3 = 3 \\
  \implies y_0 + \frac{h}{2} k_{0,1} &= 3 + 0.05 \times 3 = 3.15 \\
  \implies k_{0,2} &= f(0.05, 3.15) = -2 \times 0.05 + 3.15 = 3.05 \\
  \implies y_0 + \frac{h}{2} k_{0,2} &= 3 + 0.05 \times 3.05 = 3.1525 \\
  \implies k_{0,3} &= f(0.05, 3.1525) = -2 \times 0.05 + 3.1525 = 3.0525 \\
  \implies y_0 + hk_{0,3} &= 3 + 0.1 \times 3.0525 = 3.30525 \\
  \implies k_{0,4} &= f(0.1, 3.30525) = -2 \times 0.1 + 3.30525 = 3.10525 \\
  \implies y_1 &= 3 + \frac{0.1}{6} [3 + 2 \times 3.05 + 2 \times 3.0525 + 3.10525] = 3.3051708 \\
  t_1 &= 0.1 \quad y_1 = 3.3051708 \\
  \implies k_{1,1} &= f(0.1, 3.3051708) = -2 \times 0.1 + 3.3051708 = 3.1051708 \\
  \implies y_1 + \frac{h}{2} k_{1,1} &= 3.3051708 + 0.05 \times 3.1051708 = 3.4604293 \\
  \implies k_{1,2} &= f(0.15, 3.4604293) = -2 \times 0.15 + 3.4604293 = 3.1604293 \\
  \implies y_1 + \frac{h}{2} k_{1,2} &= 3.3051708 + 0.05 \times 3.1604293 = 3.4631923 \\
  \implies k_{1,3} &= f(0.15, 3.4631923) = -2 \times 0.15 + 3.4631923 = 3.1631923 \\
  \implies y_1 + hk_{1,3} &= 3.3051708 + 0.1 \times 3.4631923 = 3.62149 \\
  \implies k_{1,4} &= f(0.2, 3.62149) = -2 \times 0.2 + 3.62149 = 3.22149 \\
  \implies y_2 &= 3.3051708 + \frac{0.1}{6} [3.1051708 + 2 \times 3.1604293 + 2 \times 3.1631923 + 3.22149] = 3.6214025 \\
  t_2 &= 0.2 \quad y_2 = 3.6214025
\end{align*}
\]
and here is a table giving the first five steps. The intermediate data is only given to three decimal places even though the computation has been done to many more.

<table>
<thead>
<tr>
<th>n</th>
<th>t_n</th>
<th>y_n</th>
<th>k_{n1}</th>
<th>y_{n1}</th>
<th>k_{n2}</th>
<th>y_{n2}</th>
<th>k_{n3}</th>
<th>y_{n3}</th>
<th>k_{n4}</th>
<th>y_{n+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>3.000</td>
<td>3.000</td>
<td>3.150</td>
<td>3.050</td>
<td>3.153</td>
<td>3.053</td>
<td>3.305</td>
<td>3.105</td>
<td>3.305170833</td>
</tr>
<tr>
<td>1</td>
<td>0.1</td>
<td>3.305</td>
<td>3.105</td>
<td>3.460</td>
<td>3.160</td>
<td>3.463</td>
<td>3.163</td>
<td>3.621</td>
<td>3.221</td>
<td>3.621402571</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>3.621</td>
<td>3.221</td>
<td>3.782</td>
<td>3.282</td>
<td>3.786</td>
<td>3.286</td>
<td>3.950</td>
<td>3.350</td>
<td>3.949858497</td>
</tr>
<tr>
<td>3</td>
<td>0.3</td>
<td>3.950</td>
<td>3.350</td>
<td>4.117</td>
<td>3.417</td>
<td>4.121</td>
<td>3.421</td>
<td>4.292</td>
<td>3.492</td>
<td>4.291824240</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>4.648</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These notes have, hopefully, motivated the Euler, improved Euler and Runge-Kutta algorithms. So far we not attempted to see how efficient and how accurate the algorithms are. A first look at those questions is provided in the notes “Simple ODE Solvers – Error Behaviour”.

© Joel Feldman. 2013. All rights reserved. January 23, 2013 Simple ODE Solvers — Derivation 7