Adaptive Stepsize Numerical Methods for Solving Ordinary Differential Equations

Oleg Golberg

May 19, 2007

1 Introduction

Consider an initial value problem

$$y'(x) = f(x, y(x)), \quad y(0) = y_0 \tag{1}$$

To approximate the value $y(t)$ many numerical algorithms such Runge-Kutta methods make computations for a set of points chosen on the interval $[0, t]$. Usually the chosen points form an arithmetic series with stepsize h. As h decreases, the algorithms yield more precise results. However, in practice it is often needed to find the approximation of $y(t)$ with a given precision and then the problem of choosing an adequate value for h arises. A naïve approach when h is divided by 2 if the iteration of the algorithm produces an error larger than allowed usually yields satisfactory results. With this approach, all information from the previous iteration is discarded. It is possible to use that information to predict the value of h which produces an error of an allowed magnitude. Using an adaptive iteration for h it is possible to make the use of the algorithm more computationally effective. For the sake of simplicity, we will show how to apply the adaptive stepsize technique to Euler integration, even though similar methods exist for Runge-Kutta and other numerical algorithms. We will start with deriving an adaptive recurrence relation for h from estimating the error Euler integration produces, and them compare the computational efficiency of the adaptive and naïve approaches.

2 Standard Euler Integration

Consider an initial value problem of the same form as (1). Suppose we are interested in approximating y at a point $x = t$. For this purpose, split the interval $[0, t]$ into n intervals of length $h = \frac{t}{n}$. Due to the intermediate value theorem for every x we have

$$y(x + h) = y(x) + hf(x, y(x)) + ch^2 \tag{2}$$

where for some $\xi \in (x, x + h)$ we have

$$c = \frac{f'(\xi, y(\xi))}{2} \tag{3}$$

This observation leads to Euler integration, a simple numerical method of solving ordinary differential equations. Let $t_k = kh$ for $k = 0, 1, \ldots, n$. Then define a recursive sequence as follows

$$y_{k+1} = y_k + hf(t_k, y_k), \quad k \geq 0 \tag{4}$$

Due to (2) the error introduced on each step of (4) is ch^2 where c is proportional to $y''(\xi)$. In practice, we most often need to approximate the value of $y(t)$ with a given precision. If we use the standard Euler integration method with a fixed stepsize h, we cannot deduce anything about the error factor c. Therefore, we need to iterate the algorithm for larger values of n until the resulting y_n approximates $y(t)$ with the desired precision. In other words we need to devise a sequence h_i and compute the value of $y_n^{(i)}$ using the relations $y_0^{(i)} = y_0$ and

$$y_{k+1}^{(i)} = y_k^{(i)} + h_i f(t_k, y_k^{(i)}) \tag{5}$$

for $i = 1, 2, \ldots$ until $|y_{n_{i+1}}^{(i+1)} - y_{n_i}^{(i)}|$ becomes small enough. If the implementation of the algorithm uses fixed-precision representation of real numbers, which is often the case in practice, the complexity of each step of (5) is constant. Therefore, the complexity of computing $y_n^{(i)}(t)$ with an $n-$step integration is linear in n. Let n_{min} be the smallest number of steps necessary to achieve the desired precision. Then our goal is to reduce the total number of iterations of (5) and avoid values of n that are much larger than n_{min}. A natural way is to double the number of steps n on every iteration. In other words, h_i is given by a simple recursion $h_{i+1} = \frac{h_i}{2}$. This way only $\log_2 n_{min}$ iterations of the algorithm are needed and all n that we use do not exceed $2n$.

3 Adaptive Stepsize Method

It is easy to notice that every iteration of the Euler integration algorithm provides much information that the simple approach when n is doubled on every step does not use. Moreover, the desired precision level is not used when calculating the stepsize on the next iteration. One solution to this problem which allows to optimize the iteration algorithm is to look at the error each step produces. We already found that each step of the iteration (5) produces an error of ch^2 where the rate of change of c is proportional to $f'''(\xi)$. Let us define the error of each step

$$\epsilon_k^{(i)} = y(t) - y_k^{(i)} \tag{6}$$

Then we are looking for $|\epsilon_{n_i}^{(i)}| < \epsilon$ to hold. If we assume that the error factor c in (2) is constant, it will enable us to devise a sequence h_i that will lead to a more efficient approximation of $y(t)$. If c is constant, then the accumulated error when calculating $y_n^{(i)}$ is

$$\epsilon_{n_i}^{(i)} = n_i c h_i^2 = \frac{t}{h_i} c h_i^2 = tch_i = ah_i \tag{7}$$

where a is a constant. Consider two iterations with stepsizes h_1 and h_2. Recalling the definition of the error of each iteration, we have

$$y(t) - y_{n_1}^{(1)} = \epsilon_{n_1}^{(1)} = ah_1 \tag{8}$$

$$y(t) - y_{n_2}^{(2)} = \epsilon_{n_2}^{(2)} = ah_2 \tag{9}$$

Subtracting, we have

$$y_{n_2}^{(2)} - y_{n_1}^{(1)} = a(h_1 - h_2) \tag{10}$$

$$a = \frac{y_{n_2}^{(2)} - y_{n_1}^{(1)}}{h_1 - h_2} \tag{11}$$

Let us find what is the condition for h_3 needed for the next iteration to produce an error within the desired range.

$$\epsilon > |\epsilon_{n_3}^{(3)}| = |ah_3| = \frac{h_3|y_{n_2}^{(2)} - y_{n_1}^{(1)}|}{|h_1 - h_2|} \tag{12}$$

$$h_3 > \frac{(h_1 - h_2)\epsilon}{|y_{n_2}^{(2)} - y_{n_1}^{(1)}|} \tag{13}$$

This leads to a simple adaptive stepsize algorithm determined by the sequence h_i which is defined as

$$h_{i+2} = q\frac{(h_i - h_{i+1})\epsilon}{|y_{n_{i+1}}^{(i+1)} - y_{n_i}^{(i)}|} \tag{14}$$

for some coefficient $q < 1$.

4 Comparison

Let us compare the efficiency of Euler integration using the naïve doubling iteration and the adaptive stepsize method shown by applying them to the following initial value problem

$$y'(x) = 1 - 4x + y(x), \quad y(0) = 1 \tag{15}$$

The goal is to approximate the value $y(1)$ with error less than 10^{-3}. We will use the following simple implementation of Euler integration in Common Lisp using 8-byte floating point numbers.

```
(defun euler (f tn n y0)
  "Approximate y(tn) with n-step Euler method"
  (declare (type double-float x y0)
           (type integer n))
  (let ((h (/ tn n)) (y y0))
    (progn
      (loop for i from 0 to (- n 1)
         do (incf y (* h (funcall f (* i h) y))))
      y)))
```

The exact solution of the initial value problem (15) is easy to find to be

$$y(x) = -\frac{3}{16} + \frac{x}{4} + \frac{19}{16}e^{4t} \tag{16}$$

Then $y(1) \approx 64.897$. Let us choose the initial stepsize $h_1 = 0.1$ and find the values of $y_{n_i}^{(i)}$ for the sequence h_i defined by $h_{i+1} = 2h_i$ first.

$$
\begin{aligned}
y_{10}^{(1)} &\approx 34.411 & y_{5120}^{(10)} &\approx 64.796 \\
y_{20}^{(2)} &\approx 45.588 & y_{10240}^{(11)} &\approx 64.847 \\
y_{40}^{(3)} &\approx 53.807 & y_{20480}^{(12)} &\approx 64.872 \\
y_{80}^{(4)} &\approx 58.916 & y_{40960}^{(13)} &\approx 64.885 \\
y_{160}^{(5)} &\approx 61.786 & y_{81920}^{(14)} &\approx 64.891 \\
y_{320}^{(6)} &\approx 63.310 & y_{163840}^{(15)} &\approx 64.894 \\
y_{640}^{(7)} &\approx 64.095 & y_{327680}^{(16)} &\approx 64.896 \\
y_{1280}^{(8)} &\approx 64.494 & y_{655360}^{(17)} &\approx 64.897 \\
y_{2560}^{(9)} &\approx 64.695 &
\end{aligned}
$$

Therefore, 17 iterations are needed and the cost of computing the approximation is

$$10m + 20m + \cdots + 655360m = 10(2^{16} - 1)m = 1310710m \tag{17}$$

where m is the computational cost of one step of (5) which is constant for our implementation. Now let us keep the initial stepsizes $h_1 = 0.1$, $h_2 = 0.05$ but use the adaptive recurrence (14) with the coefficient $q = 0.9$. The first two approximations to $y(t)$ are the same.

$$y_{10}^{(1)} \approx 34.411 \quad y_{20}^{(2)} \approx 45.588$$

Because the allowed error $\epsilon = 0.001$, we have

$$h_3 = 0.9\frac{(0.1 - 0.05) \times 0.001}{|45.588 - 34.411|} \approx 4.026 \times 10^{-6} \tag{18}$$

Then $n_3 = \lceil \frac{1}{h_3} \rceil = 248378$. As expected, the iteration yields error close to the allowed limit.

$$y_{248378}^{(3)} \approx 64.896 \tag{19}$$

For the next iteration we have

$$h_4 = 0.9\frac{h_2 - h_3}{|64.896 - 45.588|} \approx 2.331 \times 10^{-6} \tag{20}$$

Then $n_4 = \lceil \frac{1}{h_4} \rceil = 429086$. The fourth iteration yields an approximation with the desired precision.

$$y_{429086}^{(4)} \approx 64.897 \tag{21}$$

Thus, when a simple adaptive stepsize method is used, the number of iterations needed is only 4 with the total cost of computation

$$10m + 20m + 248378m + 429086m = 677494m \tag{22}$$

which is around half of the cost of computing the approximation with doubling the number of steps.

5 Conclusion

We have seen how to successfully apply the adaptive stepsize methods to Euler integration making it more computationally effective. Similar but more advanced techniques can be applied to more efficient numerical methods such as Runge-Kutta to develop adaptive stepsize algorithms such as Runge-Kutta-Fehlberg and Dormand-Prince methods which are used in practice. For example, Dormand-Prince method is used in one of the Matlab ordinary differential equation solvers.