Euler's Method with Adaptive Step Size

Jihong Gan

April 2020

1 Introduction

In generic Euler's method, we have to manually pick a fixed step size for all iterations. When the target function changes rapidly on some intervals, we usually choose a small step size to guarantee the convergence and reduce the error. However, a small fixed step size slows down the computation. One way to solve this is to make step size small when needed, and larger when not.

This project implements Euler's method with adaptive step size. In one iteration, we compute an approximation of the target function with a certain step size and estimates the local error of our approximation. If the error is below a certain tolerance, the algorithm accepts the approximation and increases the step size in the next iteration. If above, decreases it and redo the current step. A few tests of our algorithm will be included.

2 Local Truncation Error Estimation

The core idea of most numerical methods with adaptive step size is to adjust the step size according to the local error in each iteration. The way of estimating the local error varies from algorithm to algorithm. In this project, we estimates the the local truncation error (LTE) of what we call the two-step Euler by computing the difference of one-step and two-step Euler. We then standardize the LTE into the error per unit step, which will be used to compare with our tolerance. This section explains and justifies such error estimation. The work here is based on [1], [2], and [3].

Consider the initial value problem (IVP)

$$y' = f(t, y), y(t_0) = y_0$$

Here, y has continuous first, second, and third derivative. f has continuous first and second partial derivatives in some rectangular region R containing the true and approximate solutions.

Denote the exact solution of the IVP by $\phi(t)$, such that

$$\phi'(t) = f(t, \phi(t)) \text{ for all } t$$
$$\phi(t_n) = y_n \text{ for each } n$$

We differentiate ϕ from y notation-wise, where the latter will often be used to denote the approximations made by Euler's method in this project.

Recall that the LTE, by definition, is the of truncation error generated in one step of a numerical approximation. In other words, it is the error in step $n+1$ if there was no error in step n. Let h be the step size in step $n+1$. In one-step Euler,

$$y_{n+1} = y_n + hf(t_n, y_n)$$

In two-step Euler, we compute y_{n+1} by doing one-step Euler twice, each time with step size $\frac{h}{2}$.

$$y_{n+1} = y_n + \frac{h}{2}f(t_n, y_n) + \frac{h}{2}f(t_n + \frac{h}{2}, y_n + \frac{h}{2}f(t_n, y_n))$$

Hence, the LTE of one-step Euler is

$$E_1(h) = \phi(t_n + h) - y_n - hf(t_n, y_n)$$

The LTE of two-step Euler is

$$E_2(h) = \phi(t_n + h) - y_n - \frac{h}{2}f(t_n, y_n) - \frac{h}{2}f(t_n + \frac{h}{2}, y_n + \frac{h}{2}f(t_n, y_n))$$

Apply Taylor's Theorem on $E_1 : [0, h] \to \mathbb{R}$. There exists a $\xi \in (0, h)$ such that,

$$\begin{aligned} E_1(h) &= E_1(0) + E_1'(0) + \frac{h^2}{2}E_1''(0) + \frac{h^3}{3!}E_1'''(\xi) \\ &= \frac{h^2}{2}\phi''(t_n) + \frac{h^3}{3!}E_1'''(\xi) \end{aligned}$$

Recall that ϕ has continuous third derivative on the interval $[t_n, t_n + h]$. This suggests that its third derivative must be bounded. It follows that there exists a real M such that $|E_1'''(\xi)| \leq M$. Let $K = \frac{1}{2}\phi''(t_n)$. Then,

$$E_1(h) = Kh^2 + O(h^3)$$

Similarly,

$$\begin{aligned} E_2(h) &= E_2(0) + E_2'(0) + \frac{h^2}{2}E_1''(0) + \frac{h^3}{3!}E_1'''(\xi) \\ &= \frac{1}{4}\phi''(t_n)h^2 + \frac{h^3}{3!}E_1'''(\xi) \\ &= \frac{1}{2}Kh^2 + O(h^3) \end{aligned}$$

We do not include the full symbolic computation process due to technical and time constraints. The readers may see [1] for more details.

Let $A_1 = \phi(t_n + h) + E_1(h)$, $A_2 = \phi(t_n + h) + E_2(h)$. It follows that

$$A_1 - A_2 = E_1(h) - E_2(h) = \frac{1}{2}Kh^2 + O(h^3) \approx E_2(h)$$

It is worth noting that we use $A_1 - A_2$, instead of $|A_1 - A_2|$. If $\phi(t)$ concaves up at t_n, $E_2 < 0$ and $A_1 - A_2 < 0$, but $|A_1 - A_2| > 0$. So $A_1 - A_2$ always has the same sign with E_2, but $|A_1 - A_2|$ does not.

We have thus proved that the difference between the approximation made by one-step and two-step Euler, $A_1 - A_2$, is indeed a good estimation of the LTE of the two-step Euler, E_2. We will investigate how this helps to implement our algorithm in the next section.

3 The Algorithm

In this section we describe the implementation of the algorithm of Euler's method with adaptive step size. Following from the previous section, in the step $n + 1$, let the approximation made by one-step Euler be

$$A_1 = y_n + hf(t_n, y_n) = \phi(t_n + h) + E_1(h)$$

Let the approximation made by two-step Euler be

$$A_2 = y_n + \frac{h}{2}f(t_n + \frac{h}{2}, y_n + \frac{h}{2}f(t_n, y_n)) = \phi(t_n + h) + E_2(h)$$

In the previous section, we have shown that the LTE of two-step Euler as a function of h is

$$E_2(h) \approx A_1 - A_2 \approx \frac{1}{2}Kh^2$$

Since h is varying through iterations, we standardize the LTE into the error per unit step

$$\epsilon = \frac{A_1 - A_2}{h} \approx \frac{1}{2}Kh$$

Let $\tau > 0$ be the manually picked tolerance of error.

If $|\epsilon| > \tau$, we reject A_2, update to a smaller step size h', and redo the current iteration. We pick h' such that

$$\frac{1}{2}|K|h' \approx \frac{|\epsilon|}{h}h' < \tau$$

To give us some safety margin, we let

$$h' = 0.9\frac{\tau}{|\epsilon|}h$$

Multiplying $\frac{\tau}{|\epsilon|}h$ by a constant less than 1 helps to prevent the round-off error. It does indeed improve the performance as shown by experiments. Then, the error per unit step under the updated step size is

$$|\epsilon'| \approx \frac{1}{2}|K|h' = 0.9\tau < \tau$$

If $|\epsilon| < \tau$, we accept the current approximation and initiate the next iteration with a step size larger than the current one. Naively, we may take A_2 as the approximation in the current iteration. However, note that

$$\begin{aligned}\phi(t_n + h) &= A_2 - \frac{1}{2}Kh^2 + O(h^3)\\ &= 2A_2 - A_1 + O(h^3)\end{aligned}$$

Hence, for better accuracy, we should set the approximation to be

$$y_{n+1} = 2A_2 - A_1$$

As the step size works well to control the error in the current step, we can reasonably adopt a larger step size h_n in the next iteration to speed up the computation with minor loss of accuracy. To give us some safety margin like before, we initiate the step size in step $n + 2$ as

$$h_{next} = 0.9\frac{\tau}{|\epsilon|}h$$

Note that it is possible that $t_n + h_{next} > t_f$, where t_f is the final time index. This can be avoided in actual implementation by setting

$$h_{next} = min(0.9\frac{\tau}{|\epsilon|}h_n, t_f - t_n)$$

We have now walked through one full iteration of the algorithm. In the next section, we present some tests of the algorithm.

4 Tests and Comparisons

In this section, we compare the performance of Euler's Method with adaptive step size and the generic Euler's method, concerning both run time and accuracy. The software used for testing is coded in Python 3.7.4. It is hosted on
`https://github.com/JihongGan/Smart-Euler-Method`

We use the time function in Python to measure the run time. Its output may vary slightly from time to time with the same test due to hardware conditions. The total number of iterations is also taken into account. We use the global truncation error to measure the accuracy of the algorithms. It is defined to be the difference between the exact solution value and the numerical approximation at certain t. In plain English, the closer our approximation is to the exact value of the solution at t, the higher the accuracy.

We first present an example where the solution function changes rapidly on some intervals. Consider the IVP

$$y'(t) = e^t sin(y), y(0) = 5$$

Let the final time index $t_f = 12.0$, and N be the number of iterations taken. From the analytic solution of the IVP, we know that $y(t_f)$ converges to $\pi \approx 3.1592654$.

For Euler's method with adaptive size:

τ	y_f	run time (in seconds)	N
4.0	3.141634207591565604605214411	1.0965139865875244	121131
3.0	3.141569658714168366534954769	1.1032710075378418	121152
0.5	3.141592653589792952476723979	1.1016321182250977	121062
0.1	3.141593081494632223223560815	1.1047279834747314	121073

For generic Euler's method:

h	y_f	run time (in seconds)	N
0.001	-597.8737006941127751286539	0.024658918380737305	$1.2e4$
0.0001	10716.00211531244980759546	0.24464917182922363	$1.2e5$
0.000013	2.652807008972126391984167	1.8145179748535156	*
0.000012	3.141592653589792952476723	2.0351157188415527	$1e6$
0.00001	3.141592653589793254342770	2.374187707901001	$1.2e6$

Remark: In this case, the performance of our algorithm is significantly better. For the generic Euler's method to converge, it has to spend nearly double amount of run time and 10 times more iterations.

Note that the performance of our algorithm does not change significantly with large or small tolerance. From the slope field, we intuitively see that the solution of this IVP changes drastically when t is small, but turns flat after t gets large enough. In other words, the global truncation error is not introduced evenly through propagations. $\tau = 4.0$ seems to be quite a large tolerance, but it is sufficient to bound the drastic changes at the beginning. Once the curve is flattened, little error will be introduced.

In the second example, we show that the benefit of adaptive step size applies to more evenly changing solution functions as well. Consider the IVP

$$y'(t) = 2y - 1, y(0) = 1$$

Adopting the same notation with the first example, let $t_f = 1.0$. From the analytic solution, $y(t_f) \approx 4.195$.

For Euler's method with adaptive size:

τ	y_f	run time (in seconds)	N
0.5	3.49409369154249	0.000120162963867	4
0.4	3.72928110680392	0.000136852264404	6
0.3	3.92868496613858	0.000164031982421	10
0.2	4.07621276815640	0.000180006027221	16
0.1	4.16255392476716	0.000220775604248	24
0.001	4.19452411099042	0.00838017463684082	1773

For generic Euler's method:

h	y_f	run time (in seconds)	N
0.1	3.5958682112	2.5033950805664062e-05	10
0.01	4.1223230593	0.0001010894775390625	100
0.001	4.1871561951	0.0009081363677978516	1000
0.0001	4.1937893162	0.008636951446533203	10000
0.00001	4.194454160	0.07775688171386719	100000

Remarks: The analytic solution of this IVP is $y(t) = \frac{e^{2t}+1}{2}$, which is a moderately changing function. In other words, the global truncation error is more evenly introduced comparing with the last example. We still see our algorithm performs well here, with 10 times less of run time, and 100 times less of iterations at convergence. The tolerance does have a pronounced influence upon the global truncation error in this case.

However, note that our algorithm does not necessarily outperform the generic Euler in situations other than convergence. When $\tau = 0.4$ and $h = 0.01$ in the second example, both algorithms have about the same run time, but the generic Euler has better accuracy. This is because our algorithm is not a computationally cheap one - it performs way more computations in each iteration than the generic Euler does. Its edge is only pronounced after the difference of total iterations out-weights the one of computation per iteration.

5 Conclusions

We have shown that the Euler's method with adaptive step size both theoretically and practically works. In the last section, we generally refer to the global truncation error as dependent on the tolerance. A natural question to ask here is how does the tolerance affect the global truncation error and the convergence of numerical approximation. How small does the tolerance have to be for the approximation to be convergent? How is that related to the behavior of specific functions? We have intuitively touched on the skin of these questions in the last section. Due to time and technical constraints, we must leave them for further investigations.

References

[1] Joel Feldman, *Variable Step Size Methods*, University of British Columbia, 1999.
`https://www.math.ubc.ca/~feldman/math/vble.pdf`

[2] Glenn Ledder, *Error in Euler's Method*, University of Nebraska - Lincoln.
`http://www.math.unl.edu/~gledder1/Math447/EulerError`

[3] Robert B. Israel, *Error Analysis of the Euler Method*, University of British Columbia, 2002.
`http://www.math.ubc.ca/~israel/m215/euler2/euler2.html`