Verifying A Runge-Kutta Solver Using Automatic Differentiation

Chantal Edwards¹
University of Maryland, Baltimore County, Baltimore, MD 21206
Dr. Michael Fagan, Advisor²
Rice University, Houston, TX 77005

June 2003

Abstract

This report describes our effort to verify differential equation solvers using automatic differentiation (AD). In particular, the report describes the AD verification technique in general, as well as the application of the technique to a 4th order Runge-Kutta solver using the ADOL-C tool.

1 Introduction

1.1 Verifying ODE Solvers

¹ Thanks to AGEP program for funding this work
² Thanks to LACSI for funding this work
Ordinary Differential Equations (ODEs) have numerous applications in a wide range of fields. They are used to model both biological and physical phenomena, and play an integral role in scientific computing. In using ODEs, it is often necessary to compute their solutions, which can be difficult given the fact that most do not have closed-form solutions. (e.g. solutions that can be computed by hand). Therefore, numerical ODE solvers (usually written as computer programs) are used to compute approximate solutions.

Verification refers to the process of determining the correctness of any computer program. ODE solvers can be difficult to verify because they can be quite long and difficult to understand. The goal is to use AD tools to create a new method by which one may verify these solvers. Since both linear ODEs and those with closed-form solutions can be easily solved, and since ODEs with formulaic solutions are a rarity, a non-linear ODE with neither of the aforementioned properties is chosen.

1.2 Automatic Differentiation and ADOL-C

Automatic Differentiation (AD) is a relatively new method for differentiating functions, but it is rapidly becoming a more mainstream tool. It is different from traditional methods of computing derivatives such as finite (or divided) differences and symbolic differentiation with respect to the way derivatives are computed. Using finite differences, rough approximations of the derivative are obtained by dividing the change in the dependent variable by the change in the independent variable. These approximations can return considerable error, depending on the function being evaluated. Symbolic differentiation consists of differentiating a function based on fundamental principles of derivative calculus such as the power rule, chain rule, and product and quotient rules. Since symbolic
differentiation computes formulae for derivatives, there is no error associated with it. However, programs that differentiate functions symbolically generally have a relatively high run-time.

Automatic Differentiation utilizes symbolic differentiation, but sometimes includes instructions for improving program efficiency in computing derivatives. To use ADOL-C, active variables are chosen. Active variables refer to independent and dependent variables involved with the function being differentiated. ADOL-C records all mathematical operations during which active variables are used into a separate file. Each mathematical operation called by the function is then differentiated symbolically with respect to the independent variable. The results are returned via a matrix provided to ADOL-C by the user. In contrast to most methods of differentiation, AD tools compute the derivative of functions defined by computer programs. Although any AD tool is acceptable for use, Automatic Differentiation by Overloading in C++ (ADOL-C v.1.8) is used for this research.

2 Methods

To begin this research, a non-linear ODE without closed-form solution is selected. Since single ODEs do not have properties such as stiffness, which characterize the difficulty involved with solving them. Thus, the ODE may be chosen arbitrarily, and is shown below:

\[ u' = \frac{t^2 + \sin(u)}{u^2 + 1}, u(t_0) = u_0 \]

A numerical method, in this case, the explicit Fourth-Order Runge-Kutta (RK-4) method, is then chosen to integrate the equation. The Runge-Kutta (RK) methods are effective ODE solvers, requiring relatively low computational effort, and delivering accurate approximations. It is called a “one-step” method, because it requires information from the
previous iteration to determine the value of the unknown function at the next step. The Fourth-Order RK method is found below for an ODE of the form $y'(t) = f(t,y)$ using a stepsize $h$:

$$
k_1 = hf(t_n, y_n)
k_2 = hf(t_n + \frac{h}{2}, y_n + \frac{k_1}{2})
k_3 = hf(t_n + \frac{h}{2}, y_n + \frac{k_2}{2})
k_4 = hf(t_n + h, y_n + k_3)
y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)
$$

Higher-order Runge-Kutta methods are widely used throughout industry. For these reasons, the Fourth-Order RK method was chosen for the purposes of this research.

The C Programming Language is used to write an ODE solver according to the RK-4 method. This program accepts arguments as follows: initial and final values of the independent variable $t$, initial conditions $u(t_0) = u_0$, and the number of steps at which $u(t)$ should be evaluated. Varying the arguments, output from the RK-4 program is checked against output from ‘ode45,’ MATLAB’s built-in Fourth-Order Runge-Kutta solver, to verify that it works properly.

Once confidence in the RK-4 program is established, it is necessary to differentiate the program using ADOL-C. ADOL-C is free to the public and can be downloaded from the internet (see Works Cited for website). The method for differentiation and verification is as follows:

Consider an ODE in the form $u'(t) = f(t,u)$ (or $u'(t) - f(t,u) = 0$) where $u$ is the dependent variable and $t$ is the independent variable. Also consider a numerical integrator written as a computer program, which shall be called P. The program P accepts a vector $t$ as the independent variable, and outputs approximations to $u(t)$, so that $P(t) \approx u(t)$. Using
ADOL-C, one can find $P'(t)$, which gives approximations to $u'(t)$. In order to verify $P$, one checks that $P'(t) = f(t,u) \approx 0$.

3 Results

Differentiating the program with $t$ as the independent variable yields incorrect results. The table below is a sample of output from the differentiated RK-4 program. The last column gives the magnitude of the absolute error between $u'(t)$ and RK-4$'$(t) as computed by ADOL-C. Given that ADOL-C computes derivatives without truncation error, and given that the Runge-Kutta methods are considered to be accurate methods of integration, the initial results seem suspect. Note, however, that RK-4 approximations to $u(t)$ match the output from the undifferentiated code exactly. This suggests that ADOL-C is not being used correctly.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$u(t)$</th>
<th>RK(t)</th>
<th>RK$'$ (t)</th>
<th>RK$'$ (t) - $u'(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5492021</td>
<td>9.484861</td>
<td>0.000091</td>
<td>-0.137646</td>
<td>-0.137731</td>
</tr>
<tr>
<td>3.5503803</td>
<td>9.485024</td>
<td>0.000091</td>
<td>-0.137731</td>
<td>-0.137817</td>
</tr>
<tr>
<td>3.5515586</td>
<td>9.485186</td>
<td>0.000091</td>
<td>-0.137817</td>
<td>-0.137902</td>
</tr>
<tr>
<td>3.5527368</td>
<td>9.485349</td>
<td>0.000092</td>
<td>-0.137902</td>
<td>-0.137988</td>
</tr>
<tr>
<td>3.5539150</td>
<td>9.485511</td>
<td>0.000092</td>
<td>-0.137988</td>
<td>-0.138073</td>
</tr>
<tr>
<td>3.5550932</td>
<td>9.485674</td>
<td>0.000092</td>
<td>-0.138073</td>
<td>-0.138159</td>
</tr>
<tr>
<td>3.5562715</td>
<td>9.485837</td>
<td>0.000092</td>
<td>-0.138159</td>
<td>-0.138244</td>
</tr>
<tr>
<td>3.5574497</td>
<td>9.486000</td>
<td>0.000092</td>
<td>-0.138244</td>
<td>-0.138330</td>
</tr>
</tbody>
</table>
Figure 3.1 (left) shows actual values of $u'(t)$ for $t = [1.37419, 6.200219]$ (denoted by the red line). Other lines represent ADOL-C's approximations to $u'(t)$ at 256, 1024, 2048, and 4096 timesteps. Figure 3.2 zooms in on the region $t = (3.535, 3.57)$. Notice the large difference between actual $u'(t)$ values and the values of RK-4$(t)$.

Therefore, RK-4 is modified to take $h$ as its independent variable, and ADOL-C is used to differentiate the new program. The output below displays little absolute error, tentatively showing that the program works accurately.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$u(t)$</th>
<th>RK$(t)$</th>
<th>RK$(t)$ - $u'(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5492021</td>
<td>9.484861</td>
<td>0.137731</td>
<td>-0.000006</td>
</tr>
<tr>
<td>3.5503803</td>
<td>9.485024</td>
<td>0.137817</td>
<td>-0.000006</td>
</tr>
<tr>
<td>3.5515586</td>
<td>9.485186</td>
<td>0.137902</td>
<td>-0.000006</td>
</tr>
<tr>
<td>3.5527368</td>
<td>9.485349</td>
<td>0.137988</td>
<td>-0.000006</td>
</tr>
<tr>
<td>3.5539150</td>
<td>9.485511</td>
<td>0.138073</td>
<td>-0.000006</td>
</tr>
<tr>
<td>3.5550932</td>
<td>9.485674</td>
<td>0.138159</td>
<td>-0.000006</td>
</tr>
<tr>
<td>3.5562715</td>
<td>9.485837</td>
<td>0.138244</td>
<td>-0.000006</td>
</tr>
<tr>
<td>3.5574497</td>
<td>9.486000</td>
<td>0.138330</td>
<td>-0.000006</td>
</tr>
</tbody>
</table>

Figure 3.3 (left) shows actual values of $u'(t)$ for $t = [1.37419, 6.200219]$ (denoted by the red line). Other lines represent ADOL-C's approximations to $u'(t)$ at 256, 1024, 2048, and 4096 timesteps. Figure 3.4 zooms in on the region $t = (3.535, 3.57)$. 
4 Future Work

Further goals involved with this research are:

1. Checking this method of verification to ensure that it returns correct results and

2. Applying this method to other numerical ODE solvers. (Contingent upon the results of checking the method).

It is necessary that the correctness of this method of verification be evaluated. Thus, output from an RK-4 program that works properly (such as the one used for this research) will be checked against output from an RK-4 program that returns “bad” approximations to u(t). Both programs will be differentiated via ADOL-C, and the results for dRK-4’(h) will be compared. One should expect that a properly working method of verification would return larger error for imperfect programs, and smaller error for properly working ones. Based upon an analysis of results from both differentiated programs, one can determine whether the verification method is able to detect “bad” ODE solvers.

Assuming that the method of verification is able to do so, it will be applied to other commonly used ODE integrators. It is anticipated that upon differentiation, more primitive integrators such as Euler’s Method return a larger error than more sophisticated ones such as the RK methods.

Bibliography

