

Error Control Matters

G.D. Byrne

Departments of Applied Mathematics and
Chemical & Environmental Engineering
Illinois Institute of Technology
Chicago, IL 60616, U.S.A.
gbyrne@wi.rr.com

S. Thompson*

Department of Mathematics and Statistics
Radford University
Radford, VA 24142, U.S.A.
thompson@radford.edu

January 31, 2013

Abstract

Sometimes an ordinary differential equation (ODE) solver gives the results it should, even if they are unexpected or undesirable. To obtain correct, expected numerical results, both relative and absolute error control tolerances must be chosen judiciously. Here we attempt to clear up some possible misunderstandings regarding the performance of a popular ODE solver and error control.

1 Introduction

A recent article [11] discusses the failure of a Runge-Kutta code `ode45` in MATLAB [7] and analyzes the causes of the failure to correctly solve an ODE describing a coupled oscillator. That paper argues that stiffness causes `ode45` to use a step size that is too large at times and that `ode45` does not recognize this quickly enough to yield an accurate solution.

In this paper we offer another explanation for the failure. Although possible stiffness is always a concern, we argue that the cause for failure for this particular problem is related to the choice of error tolerances for an unstable problem and that basically the use of tolerances that are too large leads to an incorrect solution. We will see that it is relatively straightforward to obtain the analytic

*Corresponding author.

general solution for this problem and see there are multiple equilibrium solutions for $\theta_2 - \theta_1$, and that using large error tolerances can cause numerical solutions to jump between these equilibria leading to the inaccuracies observed in [11].

Although our comments regarding the judicious choice of error tolerances are given in the context of the problem considered in [11], they apply equally well to other problems. In the authors' experience, rarely does a user select error tolerances that are too stringent. Moreover, we have made the seemingly counter-intuitive observation that very often more stringent error tolerances provide an efficient, correct solution, whereas lax error tolerances frequently provide a solution that has neither of these characteristics. Finally, we will attempt to help our audience to more effectively use high quality ODE software and to clear up some issues that may arise from attempting to solve an ODE. It is hoped that our comments regarding error tolerances and local error control will entice the reader to explore these issues beyond the context of this particular problem.

2 Stiffness Issues

Stiffness is generally considered to be a phenomenon exhibited when certain stable systems of ODEs that include components with differing time scales are solved. Understanding and dealing with stiffness is complicated by the fact that it really depends on several issues, specifically, the mathematical stability of the system, the length of the integration interval, the method being used, and the manner in which the method is implemented in an ODE solver [6].

The common approach to understanding stiffness is motivated by the behavior of fixed step size solutions for systems of linear ODEs with constant coefficients since the eigenvalues of the coefficient matrix completely characterize the solution of the system in this case; and they determine the behavior of numerical methods applied to the system. Generally, stiffness is considered to be important for systems in which all of the eigenvalues λ_i of the coefficient matrix have negative real parts, that is, to systems that are mathematically stable. If a system has exponentially growing solutions, the best we can generally hope to do is track the solution for such a system with a quality ODE solver by using appropriate relative error tolerances; explicit methods are generally preferable in this case.

In the event a system is stable, we can obtain an indication of stiffness by the size of $L(b - a)$ where $L = \max |Re(\lambda_i)|$ and $[a, b]$ is the integration interval of interest. We extend this measure to nonconstant coefficient linear ODEs and to nonlinear ODEs by using the eigenvalues of the local Jacobian matrix. We include the factor $b - a$ because an excessive amount of work may be required even if L is of moderate size if $b - a$ is large enough. Similarly, even if L is very large, an excessive amount of work is not necessarily required if $b - a$ is small enough. In short, the size of $L(b - a)$ is what determines the amount of work required and is therefore the commonly accepted indicator of stiffness.

When explicit methods such as those in `ode45` are used, stiffness imposes

restrictions on the step size required to obtain a stable numerical approximation. It is easy to see how such restrictions arise by considering what happens when the Forward Euler method

$$y_{n+1} = y_n + hf(t, y_n) \quad (1)$$

(where h is the step size and $t_{n+1} = t_n + h$ is a sequence of times at which the approximate solution y_{n+1} is desired) is applied to the linear test equation

$$y'(t) = \lambda_i y(t), \quad y(0) = 1. \quad (2)$$

An easy calculation shows that the numerical solution is given by

$$y_n = (1 + h\lambda_i)^n. \quad (3)$$

The numerical solution decays to 0 as t_n increases only if $|1 + h\lambda_i| < 1$. The complex values for which this is the case form the *interior* of a unit circle centered at the point $(-1, 0)$. Since this restriction must hold for each of the eigenvalues, it imposes a severe restriction on the step size when L is large. Any explicit method has a finite stability region determined by the above condition.

By way of contrast, some implicit methods have infinite stability regions. A first example is the Backward Euler method

$$y_{n+1} = y_n + hf(t, y_{n+1}) \quad (4)$$

which generates the approximations

$$y_n = \frac{1}{(1 - h\lambda_i)^n}. \quad (5)$$

These approximations decay to 0 for points *exterior* to the unit circle centered at the point $(1, 0)$. In particular, the resulting stability region contains the entire left half-plane. If the ODE system is stable, no numerical stability restriction is thus placed on the step size which is then controlled by keeping local errors sufficiently small. The MATLAB stiff ODE solver `ode15s` implements this method and similar higher order “stiffly stable” methods whose stability regions contain a significant portion of the left half-plane.

ODE solvers such as `ode45` and `ode15s` vary the step size based on local error estimates. (The exact details of how they do this are not needed for this discussion; they may be found in [6] and [9].) Imagine that an explicit method such as `ode45` is being used. Step sizes will be used for which hL is near the stability boundary for the method; and for some step sizes hL will fall outside the stability region. Here is where the stability of the ODE system comes into play. If appropriate error tolerances are used the numerical solution will not stray too far from the desired stable solution. On the other hand, if the system is not stable (for example, if multiple equilibria are present) and large error tolerances allow steps that are too large, the numerical solution may very well migrate repeatedly to other equilibria with surprising results. In the following sections we will explain how this occurs for the problem considered in [11].

Note that on the surface it might appear that the phenomenon just described may not be as relevant for stiff solvers such as `ode15s` (or other stiff solvers) since they have infinite stability regions. We might then be tempted to conclude that only stiff solvers should be used for problems such as the present one. However, there are two crucial issues to consider. We certainly would not want to use a stiff solver needlessly for a large system since the cost of the associated linear algebra required to solve the necessary nonlinear equations might very well be prohibitive. More germane to the present problem, there is no guarantee that large error tolerances will not also allow the numerical solution produced by *any* ODE solver to move away from the desired solution if the ODE is not stable. In particular, depending on the initial conditions used, the stiff solver `ode15s` can behave in the same manner as `ode45` for (6) if the error tolerances are too loose.

3 The Problem

The failure reported in [11] arose from the use of the MATLAB ODE solver `ode45`, with default error tolerances, to solve the coupled oscillator model

$$\begin{aligned}\dot{\theta}_1 &= 1 + \sin(\theta_2 - \theta_1) \\ \dot{\theta}_2 &= 1.5 + \sin(\theta_1 - \theta_2)\end{aligned}\tag{6}$$

with the initial conditions

$$\theta_1(0) = 3, \quad \theta_2(0) = 0\tag{7}$$

and with

$$0 \leq t \leq 200,\tag{8}$$

the time interval of integration. It should be noted that `ode45` is an implementation of the explicit Runge-Kutta (4,5) pair of Dormand and Prince and is sometimes called DOPRI 5 [5], [9], and [10].

The eigenvalues for system (6) are

$$\lambda_1 = -2 \cos(\theta_2 - \theta_1)\tag{9}$$

and

$$\lambda_2 = 0.\tag{10}$$

Near steady state,

$$\lambda_1 \approx -2.\tag{11}$$

Following [11], we solved the oscillator problem with `ode45` in MATLAB and with the default error tolerances. We obtained the results reproduced in our Figure 1, below, which is a reproduction of [11, Fig. 1]. We note that the default error tolerances for all of the MATLAB ODE solvers are `AbsTol = 1e-6` and `RelTol = 1e-3`. By the way, large default error tolerances are used in MATLAB

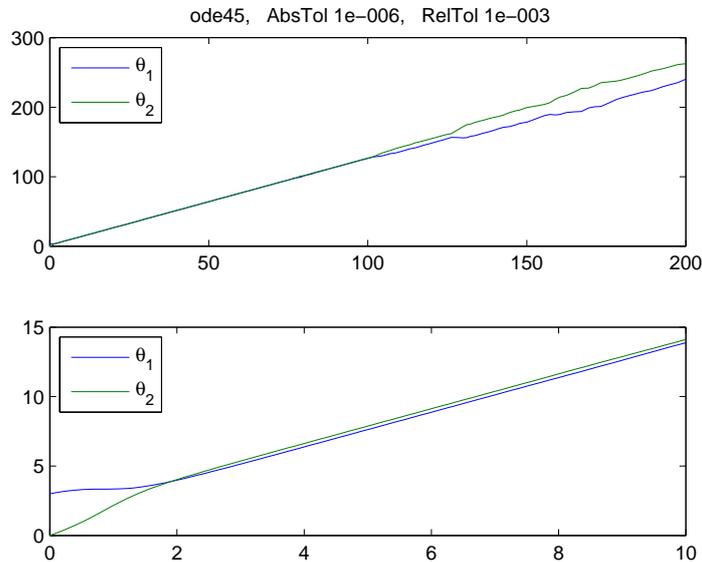


Figure 1: Solution of (6), (7), (8) with `ode45`, default tolerances

because solutions primarily are desired to within the plotting resolution, but smaller error tolerances should be used if unexpected plots are obtained.

Figure 1 clearly shows that the graphs of the solutions θ_1 and θ_2 begin to separate at about $t = 100$ which is not correct. Can `ode45` or any other non-stiff ODE solver provide a correct solution for the coupled oscillator problem? We now turn to answering this question.

4 Another Experiment with `ode45`

We felt the error tolerances in §3 were too loose and that the problem could be easily solved with `ode45`. Consequently we integrated (6), (7) over the interval

$$0 \leq t \leq 2 \times 10^4 \quad (12)$$

with `AbsTol = 1e-9` and `RelTol = 1e-6`. (We will refer to these as tight tolerances, even though they are not particularly tight.) This solution is displayed graphically in Figure 2.

Figure 2 displays the correct solution of the oscillator problem (6), (7) on (12) which is 100 times as long as the interval in [11]. Furthermore, this solution is exactly what one would expect. We have also correctly solved this problem with all of the other non-stiff and stiff MATLAB ODE solvers with tight tolerances and others, as well. The performance of the non-stiff solvers is not significantly

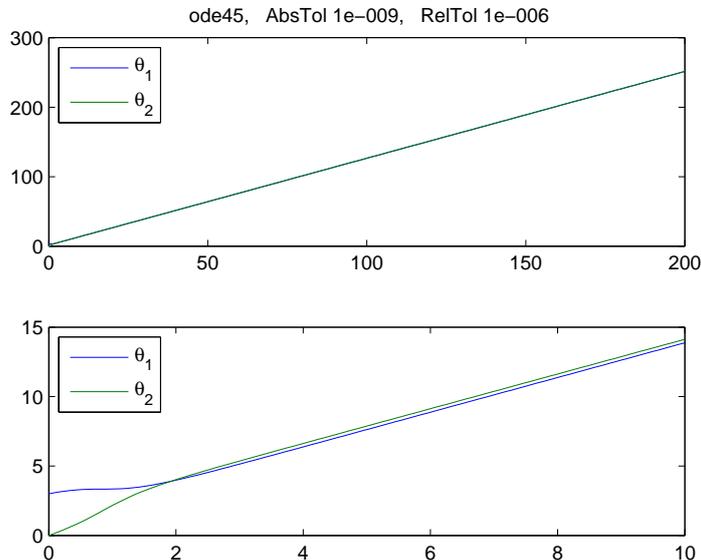


Figure 2: Solution of (6). (7), (12) with `ode45`, tight tolerances

worse than the performance of the stiff solvers in MATLAB, which indicates that the oscillator problem is not stiff.

5 Error Control

We have seen that the correct solution of (6), (7), (12) can be obtained using tight error tolerances with `ode45`. On the other hand, with the default or loose error tolerances, the correct solution was not obtained. In this section, we will attempt to explain how error control works.

During the course of the numerical solution of an ODE, a modern ODE solver computes a local error estimate for each component of the solution at each time step. If we denote this estimate for the error in y_i (component i of the computed solution y) by e_i , then the weighted error estimate for this component can be represented by

$$E_i = \frac{e_i}{\text{RelTol}_i \times |y_i| + \text{AbsTol}_i} \quad (13)$$

where we are now taking the absolute and relative error tolerance to be vectors. The subscripts i denoted vector components and the vectors are of length N , with N the number of ODEs in the system being solved. The solver strives to maintain the inequality

$$\|E\| < 1 \quad (14)$$

where $E = [E_1, E_2, \dots, E_N]^T$, and where $\|\cdot\|$ is some norm ([9, § 1.4], [4]). (For example, in many ODE solvers the root mean square or RMS norm is used [2].) When we say that the estimated error is local, we mean that it is not the global error for the entire interval of integration. It is local in the sense that it is the error for the current time step only, even though there may be extrapolation involved in the estimate. In (13), notice that if we set the error tolerances to be the scalars `AbsTol = 1e-6` and `RelTol = 1e-3` and take $y_i = 100$ then we have

$$E_i \approx \frac{e_i}{10^{-1}}. \quad (15)$$

The reader should recall that this is the weighted error estimate for one component of the *local* error. Indeed, as we have seen with the example at hand, the global error accumulates and allows an incorrect numerical solution.

The tried and true strategies used in different solvers to estimate local errors and use them to control step size (and integration order for linear multistep methods) are rather different and are somewhat complicated. [6] and [9] describe the strategies used for the Runge-Kutta and linear multistep methods used in MATLAB. For the purposes of this article, it suffices to point out that the estimates are local and that the quality of local error estimates depends on the use of appropriate error tolerances and step sizes.

6 How to Correctly Set Error Tolerances

By now, we would hope that the reader has reached the conclusion that blindly using default error tolerances with *any* ODE solver can sometimes lead to incorrect numerical results. Then how does a user correctly set error tolerances?

To answer the last question, we now turn to a classic reference [1, p. 131] where we find:

We cannot emphasize strongly enough the importance of carefully selecting these tolerances to accurately reflect the scale of the problem. In particular, for problems whose solution components are scaled very differently from each other, it is advisable to provide the code with vector valued tolerances. For users who are not sure how to set the tolerances RTOL and ATOL, we recommend starting with the following rules of thumb. Let m be the number of significant digits required for solution component y_i . Set $\text{RTOL}_i = 10^{-(m+1)}$. Set ATOL_i to the value at which $|y_i|$ is essentially insignificant.

To clarify matters, this quotation is concerned with vector-valued error tolerances, which are allowed in most modern ODE solvers. All of our remarks to this point can be easily generalized by the reader from scalar to vector error tolerances. Moreover, in the quotation, their `RTOL` is our (or the MATLAB) `RelTol` and their `ATOL` is our `AbsTol`.

In the realm of most scientific computations, we need to understand when the value of a component of the solution is insignificant or equivalent to 0 and

to use that value. It may not be very easy to decide and may well take a few computer runs and some staring at numerical results or graphical results to arrive at reasonable choices. It might also depend upon an understanding of the phenomenon being modeled. Even so, it is an important decision that can and will impact the quality of the solution and the performance of the ODE solver.

While the above advice is sound, determining appropriate error tolerances involves art as well as science. Users of a modern ODE solver can resort to a variety of actions. Solution plots often suggest incorrect results. If the behavior of available methods is considerably different as in the present problem, users of a quality solver should be alerted to the possibility something is awry. In addition, inspection of integration statistics can be very helpful. A significant increase in the number of failed steps for tighter error tolerances can be informative. For example, if derivative discontinuities are present, a solver typically will encounter many failed steps; and the step sizes it uses may be quite small near discontinuities. Further, if discontinuities are present, using smaller error tolerances typically exacerbates the problem with failed steps and small step sizes. For the present problem, however, the number of failed steps does not increase dramatically when the error tolerances are reduced; and the step sizes used are not extremely small. This is a strong indication that an explicit solver is operating near the boundary of its stability region and that the problem is not stable.

So, in general, how do we know when the numerical solution of an ODE is correct? As the author of [11] suggests, one way to make sure the answer is correct is to do a little analysis, as we too have done here. However, as we have seen, if we make a false assumption, misinterpreted results can lead us to an incorrect conclusion. Another way suggested above is to try solving the problem at hand with different error tolerances, but there is no money back guarantee that this will always work even though it often is very helpful. A third way is to look at the physical model and to see if the numerical results match the model. The authors happen to think that the very best way of all is to set error tolerances judiciously, and to use common sense and whichever technique or techniques are available.

7 Analytical Considerations

As a matter of interest, we can appeal to the analytic solution for this problem and verify our previous explanations. If we let $\theta = \theta_2 - \theta_1$ and solve the equation $\dot{\theta} = \dot{\theta}_2 - \dot{\theta}_1$, we obtain the general solution for this equation given by

$$\theta(t) = -2 \arctan \left(-4 + \sqrt{15} \tanh \left(\frac{\sqrt{15}}{4} t + C \frac{\sqrt{15}}{4} \right) \right). \quad (16)$$

The particular solution $\theta(t)$ for this problem may be expressed as

$$\arctan \left(4 - \frac{1}{15} \tanh \left(\frac{\sqrt{15}}{4} t + \operatorname{arctanh} \left(\frac{\sqrt{15}}{15} \left(\tan \left(\frac{3}{2} \right) + 4 \right) \right) \right) \right). \quad (17)$$

Figure 3 depicts a partial direction field for (16). The particular solution (17) is the one that begins at $\theta = -3$. Figure 3 shows a few of the equilibrium solutions. Note that some of the equilibria are not asymptotically stable. Also, for solutions starting near a stable equilibria if error tolerances that are too loose are used, the solution can jump from one of the stable equilibria to the vicinity one of the unstable equilibria. It is a simple matter to monitor the `ode45` local solutions and see this is precisely what happens for this problem when the error tolerances used are too large.

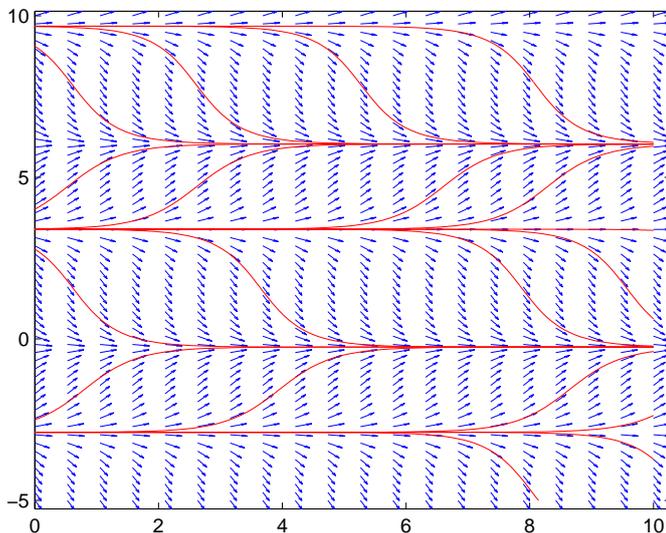


Figure 3: Direction Field for (16)

8 Conclusions

In this paper we have worked through [11] and have attempted to explain some aspects of solving ODEs, particularly with MATLAB, which might be troubling to the reader. We have examined the oscillator example of [11] and solved it with `ode45`. We have shown that the oscillator problem equations (6) and (7) can be successfully solved numerically on the intervals (8) and (12), which is 100 times as long as the interval used in [11], by tightening the error tolerances.

Further, incorrect numerical solutions *can* be obtained with the MATLAB solvers when default tolerances are used. Most importantly, perhaps, we have reviewed how correct error tolerances should be chosen for solving ODEs. Finally, for all of the problems discussed here, the MATLAB ODE solver `ode45` gives the results it should, even if they are not what we might like, when lax error tolerances are used.

To emphasize the selection of error tolerances for the MATLAB solvers, we strongly urge the reader to review §1.4, Control of the Error [9, pp. 27–34]. The authors believe that the selection of correct error tolerances is widely misunderstood and as a consequence can lead to some serious errors, misunderstandings, and misinformation. Finally, [10] contains very useful descriptions of all of the solvers used in the MATLAB suite of ODE solvers.

References

- [1] K.E. Brenan, S.L. Campbell, and L.R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations* (SIAM Classics in Applied Mathematics, 14) SIAM, Philadelphia, 1996.
- [2] P.N. Brown, G.D. Byrne, and A.C. Hindmarsh, VODE, a variable-coefficient ODE solver, SIAM J. Sci. Stat. Comput., 10 (1989) 1038–1051, American Institute of Chemical Engineers, New York, 1977.
- [3] G.D. Byrne and A.C. Hindmarsh, *Numerical Solution of Stiff Ordinary Differential Equations* AIChE Today Series, 1977.
- [4] G.D. Byrne and A.C. Hindmarsh, Stiff ODE solvers: a review of current and coming attractions, J. Comp. Phys., 70 (1986) 1–62.
- [5] J.R. Dormand and P.J. Prince, A family of embedded Runge-Kutta formulae, J. Comput. Appl. Math., 27 (1980) 19–26.
- [6] L.F. Shampine, *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall, New York, NY, 1994.
- [7] MATLAB, The MathWorks, Inc., 3 Apple Hill Dr., Natick, MA 01760, 2004.
- [8] L.F. Shampine and C.W. Gear, A user’s view of solving stiff ordinary differential equations, SIAM Rev., 21 (1979), 1–17.
- [9] L.F. Shampine, I. Gladwell, and S. Thompson, *Solving ODEs with MATLAB*, Cambridge Univ. Press, Cambridge, 2003.
- [10] L.F. Shampine and M.W. Reichelt, The MATLAB ODE Suite, SIAM J. Sci. Comput., 18 (1997) 1–22.
- [11] J.D. Skufca, Analysis still matters: a surprising instance of failure of Runge-Kutta-Felberg [sic] ODE solvers, SIAM Rev. 46 (2004) 729–737.