Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Computational and Applied Mathematics 185 (2006) 244–260

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

www.elsevier.com/locate/cam

Evaluating numerical ODE/DAE methods, algorithms and software

Gustaf Söderlind*, Lina Wang

Numerical Analysis, Centre for Mathematical Sciences, Lund University, Box 118, SE-221 00 Lund, Sweden

Received 28 March 2003

Abstract

Until recently, the testing of ODE/DAE software has been limited to simple comparisons and benchmarking. The process of developing software from a mathematically specified method is complex: it entails constructing control structures and objectives, selecting iterative methods and termination criteria, choosing norms and many more decisions. Most software constructors have taken a heuristic approach to these design choices, and as a consequence two different implementations of the same method may show significant differences in performance. Yet it is common to try to deduce from software comparisons that one *method* is better than another. Such conclusions are not warranted, however, unless the testing is carried out under true *ceteris paribus* conditions. Moreover, testing is an empirical science and as such requires a formal *test protocol*; without it conclusions are questionable, invalid or even false.

We argue that ODE/DAE software can be constructed and analyzed by proven, "standard" scientific techniques instead of heuristics. The goals are computational stability, reproducibility, and improved software quality. We also focus on different error criteria and norms, and discuss modifications to DASPK and RADAU5. Finally, some basic principles of a test protocol are outlined and applied to testing these codes on a variety of problems.
© 2005 Published by Elsevier B.V.

MSC: 65L05-Numerical analysis; Ordinary differential equations—initial value problems; Software

Keywords: Software testing; Test protocols; Computational stability; ODE software; DAE software; Work/precision; Adaptive time-stepping; Mathematical software; Algorithm analysis

* Corresponding author.
E-mail addresses: Gustaf.Soderlind@na.lu.se (G. Söderlind), Lina.Wang@na.lu.se (L. Wang).

0377-0427/$ - see front matter © 2005 Published by Elsevier B.V.
doi:10.1016/j.cam.2005.03.009

1. Introduction

Initial value problems in ordinary differential equations (ODEs) and differential-algebraic equations (DAEs) are central to the mathematical modelling of dynamical processes in science and technology. Standard ODE software has been used routinely for over 30 years. DAE software emerged around 1980 and is beginning to reach maturity. Nevertheless, there is still progress in method construction and software design, and there is a potential for considerable improvement.

The issue of adaptivity has in the past decade received an increased attention, and adaptive strategies are now beginning to be based on researched, proven principles instead of earlier heuristics [5–7,13]. In particular, the use of a coherent strategy employing digital filters for the automatic control of stepsize and errors has been shown to have a major impact on computational stability [14,15]. Thus a small change of parameters, especially TOL, only leads to a small change in the results produced by the code. In contrast, many adaptive codes based on heuristic strategies are very sensitive to such perturbations. The new control strategies can therefore be said to improve the "condition number" of the software.

This improvement can typically be obtained without added computational expense [15]. As a result, there is a software performance increase, akin to replacing a poorly conditioned algorithm by a well-conditioned one that is equally fast. Conversely, the improved stability and quality *cannot* be traded for reduced wall-clock time—perhaps fortunately, it may be added, as such a trade would not be considered good practice in numerical analysis. To summarize: work on ODE/DAE software performance should not exclusively focus on speed-up but also on stability and well-conditioning. Reliability is more important than efficiency [17, Chapter 6].

The question then is: how do we test and evaluate software, and what practices should be observed in ODE/DAE software testing? The aim of this paper is to partially answer that question and lay down a few basic principles related to ODE/DAE software design and testing. Actual tests carried out in this paper will follow the principles outlined in the paper, and are performed with two modified versions of DASPK and RADAU5, implementing the adaptivity algorithms as specified in [15].

Starting with principles of methodology, we note that software testing is an *empirical science*. It therefore requires the standards and methodology of empirical sciences rather than the approach of "a mathematical demonstration," which is commonplace today. The latter approach is scientifically correct in the non-empirical context of logic and mathematics but is unsuitable, if not useless, in the complex situation presented by software evaluation. From the epistemological point of view this is well-known: *a single counterexample falsifies a theory*, and a single example can adequately demonstrate the proper interpretation of a theory; in contrast, *a large number of carefully planned and precisely controlled experiments are needed in order to corroborate even a modest empirical claim*.

In order to discern cause and effect and make it possible to draw conclusions about methods or adaptive strategies, control structures, implementation, performance or other similar aspects, *testing must be subject to a strict test protocol*. Given a problem, such a test protocol provides detailed instructions on how to run the problem, what to look for in it, what aspects to measure and how measure them. The testing conditions—such as choices of norm, error control criterion and controllers—under which the results were obtained must be completely specified to facilitate *reproducibility*. Moreover, comparative tests should be run under *ceteris paribus* conditions, i.e., tests must be "fair" in the sense that the tested aspect alone should differ, while everything else is kept the same for both test objects. Further, results should, if possible, be presented in a standardized form.

Testing and evaluation is further helped by existing collections of real-life test problems, e.g., the CWI test set [16], now maintained in Bari, Italy, or the test problems at [2,8], or the classical DETEST [11]. The latter was the pioneering effort that began to set standards for ODE software, based on early ideas on method comparison [12]. It was then followed by a "stiff" DETEST [3]. Many of the problems collected then are still of relevance today.

Although test sets can be used as performance benchmarks, other aspects may be more interesting. Trying to use a few test problems for code tuning in order to achieve "ultimate performance," is usually of little value as it often trades robustness and coherence for a marginal gain in a single aspect of performance, e.g., total number of steps. A more critical approach is necessary. Feedback from testing to the design and parameterization of adaptive strategies can be obtained. Quality aspects, such as producing a smooth solution, using smooth stepsize sequences or producing a more stable and regular tolerance proportionality, *while* meeting a specified accuracy criterion, are often overlooked. Nevertheless, as *performance is a measure of produced quality per unit cost*, they do belong in a serious performance evaluation.

2. Towards a test protocol

We shall define some elementary conditions on testing practices. The purpose of testing is always to try to infer some claim, or to evaluate some performance aspects. The strength of the claims depend on the scientific methodology used for testing; it could range from useless to affirmative [17, Chapter 6].

2.1. Ceteris paribus conditions

Ceteris paribus conditions imply that one aspect at a time is tested, *everything else being equal*. For example, if one wants to test a new stepsize controller, that algorithm should be tested against the reference standard technique within the same code. It is, as a matter of principle, not permissible to test the new feature in another code, as it would then not be clear exactly what factor that makes the difference. If it can be shown, under *ceteris paribus* conditions within a code, that the new feature "is superior," it is of course an added benefit if it can be shown to be superior, compared to the reference standard, also in another environment, i.e., in other codes. The testing is not between codes, but within the codes.

To maintain *ceteris paribus* conditions in code comparisons is much harder. It entails tolerance scaling (see below) as well as additional requirements on using the same types of norms, error test criteria, etc. It may still be that different codes use very different approaches to solve the same problem; any such major difference should then be clearly pointed out.

2.2. Reproducibility

For each test, data for the settings and modes used in the test must be reported. Details should be specified so that the test can be independently run and repeated, and parameters for the problem as well as the solver need to be recorded. Some environment parameters (e.g., compiler choice) may have an effect, but if a code is well designed and has high computational stability, such minor differences should not have any significant impact.

Test problem setup should be completely specified. With the test sets available today (see e.g., [16,2,8]) this is a fairly straightforward matter.

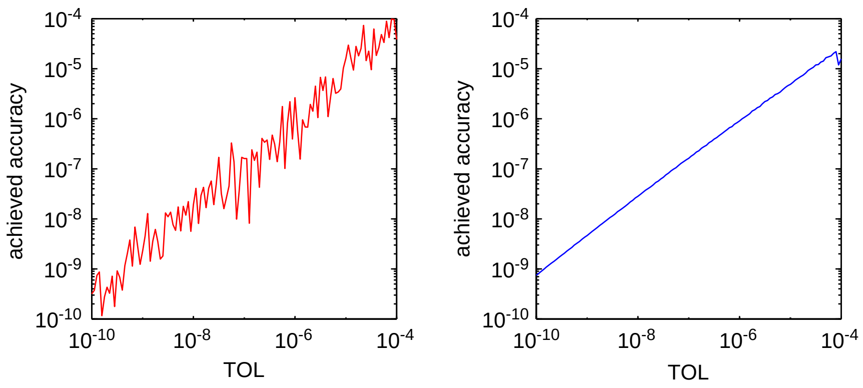

Fig. 1. Global error as a function of TOL when DASSL is applied to the Chemical Akzo Nobel problem from the CWI test set. Replacing the original heuristic stepsize controller (left) by a digital filter (right) has a strong impact on computational stability. Computational effort is similar in both graphs. Results are not tolerance proportional; that would require a tolerance rescaling.

2.3. *Computational stability*

In order to draw firm conclusions, results produced by the code must not vary strongly with small changes in the experimental setup. High computational stability implies improved reproducibility. Computational stability is defined in [15], as a qualification of *tolerance proportionality*: for each sufficiently smooth problem there should be constants c and C such that the global error e can be bounded above and below,

$$c\,\mathrm{TOL} \leqslant \|e\| \leqslant C\,\mathrm{TOL}, \tag{1}$$

and the smaller the ratio C/c, the higher is the computational stability. The first order adaptive time-step selection introduced in [14], in particular if based on digital filters, overcomes most of the instability problems associated with heuristic methods for time-step control, see Fig. 1, which is drawn from [15].

When evaluating method, algorithm or software characteristics we try to answer questions through various experiments, where the performance measurements play a key role. As a matter of principle, a scientific question can only be answered by empirical data provided that the observed results (measurements) are reasonably stable, consistent and robust under small variations in the experimental set-up. From this it follows that if the code has a high degree of computational stability, we can infer more about that code, and at a greater certainty. Questions having inherently unstable answers should be replaced by more focused questions. Finally, in some cases, stability can be improved by changing the measurement technique, rather than the code. Thus, e.g., it is sometimes of benefit to use a functional norm such as L^1 or L^2 over the whole range of integration while measuring the error, as such norms will have an averaging effect that reduces measurement fluctuations.

2.4. *Tolerance proportionality*

In many cases it appears to be possible to use tolerance scaling to make a code tolerance proportional [15]; if the error is proportional to TOL^α (see the right graph in Fig. 1), the code becomes tolerance proportional by internally using TOL′, defined by the scaling transformation

$$\mathrm{TOL}' = \mathrm{TOL}_0^{(\alpha-1)/\alpha}\,\mathrm{TOL}^{1/\alpha}. \tag{2}$$

We believe that every code should be run in a tolerance proportional mode: changing the tolerance by one order of magnitude should produce one order of magnitude difference in the error. It could be argued that this is of little importance as one eliminates the tolerance when plotting work/accuracy diagrams, but the important point is that in real computations, the error is unknown. The user only has access to TOL, and this parameter ought to have an influence on the error that is as predictable as possible. In particular, it is *impossible* for the user to assess the merits of two different codes if one is tolerance proportional and the other is not.

2.5. Tolerance calibration

In addition to scaling TOL according to (2), it is desirable to *calibrate* the tolerance: the code should produce a specified error on a reference problem. This is achieved by a suitable choice of the tolerance equivalence point TOL_0.

The important point is that TOL ought to have the *same meaning* in different codes. The reference problem could either be the linear test equation, say, or some other problem with interesting features. If TOL produces different results in two different codes on the same reference problem, more elaborate code comparisons are meaningless. We repeat that performance is subject to quality constraints; unless an effort is made to introduce *ceteris paribus* testing conditions, code comparison does not comply with elementary principles of empirical experiments.

Tolerance proportionality/calibration is therefore a requisite for scientific testing of ODE/DAE codes. We also believe that implementing such a standard would be of benefit to the user who makes production runs of real-life initial value problems. If two different codes deliver very different accuracy, for the same tolerance setting but at different costs, the user is likely to prefer the code which completes the integration faster. As the error is unknown to the user, the subjectively perceived performance is no longer subject to a proper quality constraint.

2.6. Graphical presentation

Graphs of results should be presented in a consistent manner. The “natural” way—not always used today—is to have small values to the left/down and large values to the right/up in the graphs. However, when plotting accuracy, it is not uncommon that “significant digits” are plotted; a *reduced* error that corresponds to an *increased* number of significant digits. We propose to always plot tolerances, error, work, etc. in accordance with the standard of right-hand oriented coordinate systems. This implies, in particular, that the slope of the graph in a “work/precision” (or rather “work/accuracy”) diagram should be *negative*, reflecting that there is a *trade-off* between work and accuracy. Reducing the error costs more work; reducing work leads to larger errors.

Standard measurements include plotting (i) achieved precision vs TOL (with a positive slope, see Fig. 1, as there should be a positive correlation—preferably proportionality—between the input parameter TOL and the error); (ii) plotting work (e.g., in terms of total number of steps, function evaluations, Jacobian evaluations, CPU time etc.) vs TOL, with negative slopes reflecting the trade-off; (iii) plotting work vs achieved precision, again with a negative slope. These plots should normally be presented in log–log diagrams.

3. From method to software: can heuristics be removed?

The algorithmic content of adaptive ODE/DAE software is not dominated by the discretization method but by a considerable amount of control structures, support algorithms and logic. Among the important elements are the choice of norm $\|\cdot\|$; the implementation of mixed relative–absolute error tests; aiming to keep the error per step (EPS) or per unit step (EPUS) below a tolerance; the actual stepsize controller; the construction of error estimators; the choice of predictors as well as termination criteria for fixed-point or Newton iteration; numerical or analytical Jacobians; refactorization strategies; convergence rate estimation and how to interpret "slow convergence"; maximum number if iterations per step; strategy for choosing method order; local extrapolation; recovery logic when some adaptive strategy fails, etc. The constructor of adaptive ODE/DAE software must make a choice on all of these issues, and they affect performance in various ways.

While the discretization method is thoroughly researched and fully specified, control structures and logic have often been designed using elementary *heuristics* without regard to rigorous analysis. The overall implementation is tuned using simple benchmark tests. When this process is complete, the user of the code still has a number of options with respect to the code elements mentioned above; some parameters may even be selected adaptively during the computation.

A main problem in obtaining reliable test results is that heuristic control structures often have a surprisingly degrading effect on computational stability and regularity. About 15 years ago, however, control theoretic methods started being used both for the analysis and design of adaptive algorithms, see the survey [13], and the subsequent developments [14,15]. This implies that it is now possible to analyze, with some rigor, the behaviour of central parts of the adaptive strategies. Their "mathematical status" is thus brought closer to that of the discretization method itself, and it is hoped that this will enable the method in its actual implementation to come closer to its theoretical properties and performance.

The strong need for adaptivity in ODE/DAE software leads to much more complex control structures than in e.g., linear algebra software. Because of this complexity, the quality and predictability of ODE/DAE software is still far from the modern computational standard set by linear algebra software. In order to bring ODE/DAE software closer to that standard, the same attention must be devoted to the control structures as to the method itself. The example of using control theory for the construction and analysis of adaptive time-stepping shows that significant improvements are within reach.

Last, but not least, this question is also a matter of software design. Traditionally not very emphasized in ODE software, this area deserves much more attention, for example along the lines indicated in [17, Chapter 6].

4. Norms, scaling and error criteria

We shall highlight one aspect of computational criteria that has a bearing on the design and use of ODE/DAE codes. Testing as well as production runs are concerned with computational efficiency, subject to accuracy. Reliable error measurements are needed in order to asses accuracy, but it turns out to be less than trivial to control and measure errors properly.

4.1. Significant correct digits

In the CWI test set, the end point value of an "exact" reference solution y^{ref} is provided. This is typically used to measure the number of *significant correct digits* of the solution. Let e be the global error and define

$$\mathrm{scd}(y(t_{\mathrm{end}})) := -\log_{10}\|e(t_{\mathrm{end}})./y^{\mathrm{ref}}(t_{\mathrm{end}})\|_{\infty}. \tag{3}$$

The component-wise division ./ in combination with the max norm implies that every component in the solution has an error not exceeding $10^{-\mathrm{scd}(y)}$.

The measure $\mathrm{scd}(y)$ has a number of drawbacks. First, it is not defined if a component of y^{ref} is zero; cases where some $y_i^{\mathrm{ref}}(t_{\mathrm{end}})$ is small must therefore also be excluded. Another drawback is the use of the max norm. This may not conform to the user's choice of norm and is often incompatible with the solver's internal norms, as well as with its accuracy objectives. In addition, $\|\cdot\|_{\infty}$ is non-differentiable, implying that $\mathrm{scd}(y)$ may have a less regular behaviour than desired. Finally, results will be difficult to interpret unless one measures a single solution component at a prescribed point in time, e.g., $\mathrm{scd}(y_i(t_{\mathrm{end}})) := -\log_{10}|e_i(t_{\mathrm{end}})/y_i^{\mathrm{ref}}(t_{\mathrm{end}})|$.

Precision in terms of significant digits calls for a criterion on the relative error, which must be combined with some absolute error for very small components. There are two different ways to specify such *mixed absolute–relative error* criteria corresponding to different invariance properties. We shall refer to these criteria as *fixed resolution* and *fixed scaling*, respectively. Variants of these measurements are always used in the software, and it is reasonable to try to measure the actual error in the same way.

4.2. Fixed resolution

The *fixed resolution* test defines a vector $d(y, \rho)$ by

$$d_i(y, \rho) = \mathrm{RTOL}|y_i| + \rho_i, \tag{4}$$

where RTOL is a user-prescribed relative tolerance parameter and the vector ρ specifies the *resolution level* of y_i. Given a local error estimate $\hat{l}$ and a norm $\|\cdot\|$, the error estimate is considered acceptable if

$$\|\hat{l}./d(y, \rho)\| \leqslant 1. \tag{5}$$

This construction is preferred when the user is interested in relative accuracy down to a certain level of resolution which is typically considered to be independent of the accuracy requirement. Thus the resolution limit ρ represents a "noise floor" which is characteristic of the *problem*. A concrete example of this error test is shown in the upper graph of Fig. 2.

4.3. Fixed scaling

In the *fixed scaling* test we define the norm to depend on the solution y and a *scaling* η_i of each solution component y_i. Putting

$$d_i(y, \eta) = |y_i| + \eta_i, \tag{6}$$

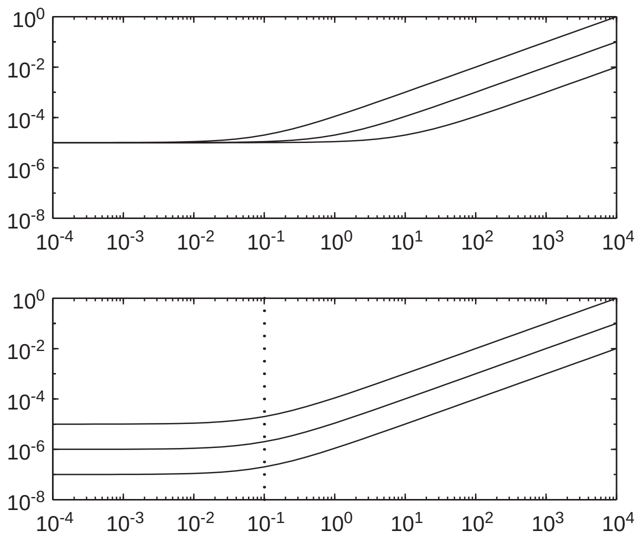

Fig. 2. Two modes to test for mixed absolute/relative accuracy. Error magnitude (vertical axis) is acceptable given the magnitude of the solution $|y|$ (horizontal axis) if the error is below the selected curve. Top: fixed resolution (10 μV) test; bottom: fixed scaling (100 mV) test. Resolution and scaling are shown by dotted lines. In both plots the tolerances are 10^{-4}, 10^{-5} and 10^{-6} from top to bottom; the curves for TOL $= 10^{-4}$ are the same for both tests. For details, see text.

we compute the norm of the error estimate by $\|\hat{l}./d(y,\eta)\|$. The error is acceptably small if

$$\|\hat{l}./d(y,\eta)\| \leqslant \mathrm{TOL}, \tag{7}$$

where TOL is a scalar tolerance parameter. This test measures relative accuracy for components with $|y_i| > \eta_i$ and absolute accuracy for components with $|y_i| < \eta_i$. The change from relative to absolute is gradual and takes place near the component scaling factor η_i. In a particular case the error criterion looks like in the lower graph of Fig. 2.

4.4. Fixed resolution vs fixed scaling—an example

Let us consider a problem of an electrical circuit, where we seek to compute nodal voltages, whose typical size does not exceed 5 V. We shall outline the differences between the two approaches above and illustrate their interpretations in terms of the error criteria as shown in Fig. 2.

Fixed resolution. Due to thermal noise, a resolution below say 10 μV is not of practical significance. We are therefore interested in *relative accuracy* down to this level, and take $\rho = 10$ μV. When RTOL is varied, the break-point between relative accuracy and noise floor shifts. Thus, by taking say RTOL $= 10^{-3}$, we impose a relative accuracy criterion for voltages $|y_i| > \rho_i/\mathrm{RTOL} = 10$ mV. However, for RTOL $= 10^{-6}$, the fixed resolution test no longer imposes a relative accuracy criterion, as this would require local errors to be below the noise floor for any voltage $|y_i| < 10$ V, see the upper graph of Fig. 2.

Fixed scaling. In the same electrical circuit, we may be interested in obtaining relative accuracy down to the fixed scale of 100 mV; this is specified by taking $\eta = 100$ mV. As TOL varies, this scaling remains fixed, implying that the accuracy criterion changes *both in the relative and the absolute domains*. Thus, while the break-point between absolute and relative accuracy is fixed at $\eta = 100\,\text{mV}$, the noise floor, equaling TOL η, shifts with TOL. For $\text{TOL} = 10^{-4}$ the noise floor becomes $10\,\mu\text{V}$, but for sharper tolerances the noise floor too is lowered, see the lower graph in Fig. 2.

4.5. Fixed resolution vs fixed scaling—practical remarks

In most ODE/DAE software the mixed absolute–relative criterion is implemented as a fixed resolution test using two different "tolerances," RTOL and ATOL, defining $d(y, \rho)$ by

$$d_i = \text{RTOL}|y_i| + \text{ATOL}_i, \tag{8}$$

and using the error test (5). This gives the impression that there are *two* tolerances to be set, affecting accuracy in distinct ways. This is true only in part; by taking $\text{RTOL} = 0$ we have a pure absolute error criterion, and by instead taking $\text{ATOL} = 0$ it becomes a pure relative error criterion. By combining nonzero values, however, the criterion is less clear, as the two parameters play different roles. The *quality* of the accuracy (absolute or relative, fixed resolution or fixed scaling) is distinctly different from the *quantity* of accuracy (determined by the tolerance).

In addition, (8) reveals that while RTOL is dimensionless, ATOL_i is not. Even if the problem has been recast in dimensionless form through scaling, both the resolution ρ and the scaling η may represent several different physical entities; ρ and η then tell what the user considers the units of the problem to be. Such problem properties should be clearly distinguished from the computation's requested numerical accuracy.

Both fixed resolution and fixed scaling can be used within the implementation (8). The choice

$$\text{RTOL} = \text{TOL}, \tag{9}$$

$$\text{ATOL}_i = \rho_i, \tag{10}$$

provides fixed resolution, while the choice

$$\text{RTOL} = \text{TOL}, \tag{11}$$

$$\text{ATOL}_i = \text{TOL}\,\eta_i, \tag{12}$$

provides a fixed scaling test criterion. Note that we propose using a scalar TOL parameter; in case different levels of relative accuracy are needed for different variables, a weighted norm is preferable (for reasons of invariance) to a vector valued RTOL. This is easily accomplished by multiplying the vector $\hat{l}./d$ by a diagonal matrix of weights W.

Only the fixed scaling can be used as a *global error measure*; no matter how TOL is varied, one simply measures e with the same norm by computing $\|e./d(y, \eta)\|$. In contrast, the fixed resolution test (5) is not a good measure of the actual error e as RTOL and ATOL enter the norm definition in $\|e./d(y, \rho)\|$. Suppose for instance that $\|e./d(y, \rho)\| = K$; this implies that the global error e is K times larger than a local error $\hat{l}$ at the same point, but it cannot be interpreted as the norm of the global error being K times the tolerance, since the norm is not a linear function of RTOL or ATOL unless one of these parameters is zero. For the same reason the fixed resolution mode has a nonlinear transfer function from error setpoint to error when the adaptivity is analyzed as a control system.

In production runs for real problems, the fixed resolution criterion is often an indispensable mode of operation. In particular, for problems modelling electrical circuits it is usually close to what the user needs and wants.

For mathematical software purposes such as testing and evaluation, however, where the tolerance parameter may vary over several orders of magnitude, it has considerable drawbacks. Thus one will typically need a covariation of RTOL and ATOL, and a common practice is to take RTOL = ATOL. However, this implies a particular *scaling* of the dependent variables, for (8) then becomes

$$d_i = \text{RTOL}\,(|y_i| + 1), \tag{13}$$

thus enforcing the scale 1 for every dependent variable, with RTOL as the noise floor. This *unintentional* fixed scaling is not necessarily a good one.

A seriously intended adaptive method needs to offer clear error tests of both types, not just one. Moreover, it would be an improvement to offer the user two distinct options of either fixed scaling or fixed resolution, rather than offering the ATOL–RTOL norm without careful instructions.

In the testing and evaluation of such a method the error criteria must be specified; the norm, the implementation and the actual *modus operandi* are but a few aspects of an automatic integration procedure which significantly affect its performance. The scientific value of test results depends crucially on whether sufficient information for reproducibility and a unique interpretation is given.

4.6. Fixed resolution vs fixed scaling—work/precision and code behaviour

We shall illustrate the effect of choosing either a fixed resolution or scaling error criterion. We test these notions by solving the differential equations describing a simple linear oscillatory circuit,

$$V'_{C_1} = -i_L/C_1, \tag{14}$$

$$V'_{C_2} = -i_L/C_2 - V_{C_2}/(RC_2), \tag{15}$$

$$i'_L = (V_{C_1} + V_{C_2})/L, \tag{16}$$

with initial conditions $V_{C_1}(0)=V_{C_2}(0)=1$ and $i_L(0)=0$, and component values $L=10\,\mu\text{H}$, $C_1=C_2=10\,\mu\text{F}$ and $R = 2\Omega$. The problem is solved over the time interval $[0, 100\,\mu\text{s}]$, and the error is evaluated by comparing with the exact solution at the endpoint of the integration.

The problem is solved with the modified version of DASPK [15] and by a new version of RADAU5, which has been modified by using the same $H211b$ digital filter for stepsize control as that employed by the modified DASPK.

The results are plotted in Figs. 3 and 4. . Both codes behave entirely in accordance with theory, but in each one, we observe widely differing qualitative behaviour depending on the user's choice of error criterion. The test demonstrates how differently a code can be made to work, and how important it is to understand the influence of code parameters in order to obtain an expected behaviour.

4.7. The need for other types of norms

Although the fixed resolution and fixed scaling criteria are suitable for many applications, there are some requiring other types of norms. For instance, in chemical reaction kinetics, the dependent variables

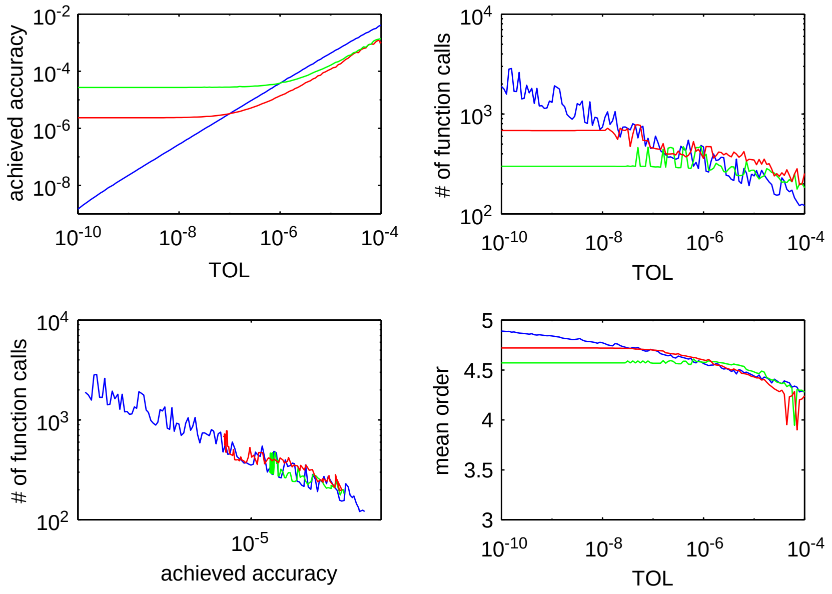

Fig. 3. Electrical circuit example is solved with Daspk using different error criteria. Only fixed scaling ($\eta = 1$) produces a tolerance proportional graph (top left); fixed resolution levels of $\rho = 10^{-6}$ and 10^{-7} produce graphs that flatten out for tight tolerances. Similar effects are seen in the work diagram (top right) and for the mean order (bottom right). Although less legible, the work precision graph (bottom left) shows that efficiency remains unaffected, but accuracy is limited to a narrow range.

typically vary over many orders of magnitude. It may then be necessary to scale the problem, or work with logarithms of the dependent variables, before either of the error criteria becomes suitable.

Worse still are problems in celestial mechanics. Systems such as the restricted three-body problem, especially Arenstorf orbits, [9, p. 130], are often solved using an absolute error criterion only, after scaling the problem so that it is of “unit size.” This criterion requires all local errors to be of the same size, yet near a singularity (a mass) the errors need to be smaller in order to accurately compute the “slingshot effect.” Alternatively, one could argue that the accuracy requirement could be relaxed a little away from singularities; hence an absolute error criterion is suboptimal. Moreover, the error criterion is typically very tight, calling for errors that are extremely minute compared to the problem size.

We therefore develop a new norm by defining

$$d_i(y, \eta) = \min_j \|y - y_j^*\| + \eta_i, \tag{17}$$

where y_j^* is the position of the jth singularity (or mass) and η_i is a scaling factor for the ith body. The norm of the local error is acceptably small if $\|\hat{l}./d(y, \eta)\| \leqslant \mathrm{TOL}$. This implies that near a singularity, the size of the denominator will force local errors to be smaller there, even though TOL is constant. Naturally, as computations involve two different types of variables, positions and velocities, these need to be treated differently.

The modified version of Daspk, [15], was tested together with a version of Radau5, both using the $H211b$ digital filter for stepsize generation, and with the additional control structure changes suggested in [15]. In both codes tolerance rescaling was used to make the codes tolerance proportional according

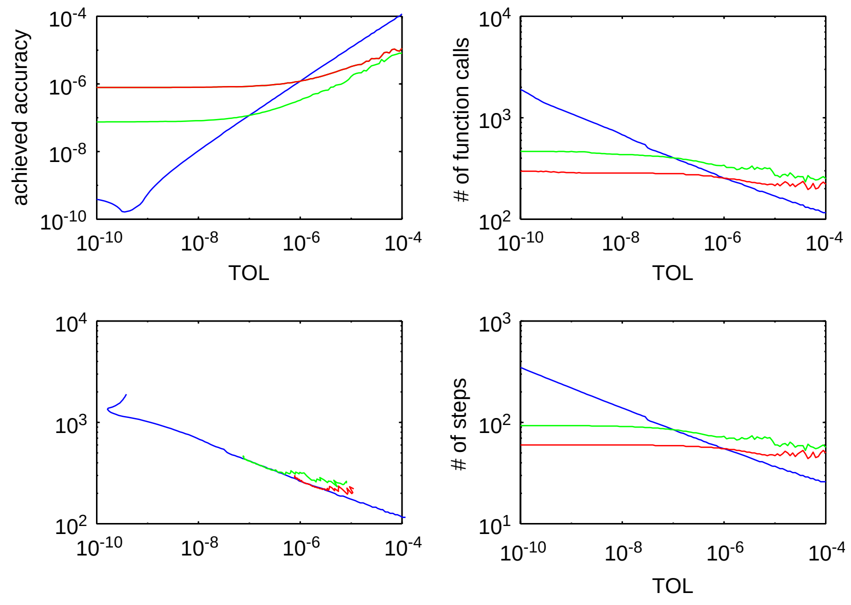

Fig. 4. The same electrical circuit example is solved using a modified version of RADAU5. The behaviour is similar to that of DASPK, but work precision graphs are smoother. The graph on the bottom right shows total number of steps instead of the mean order.

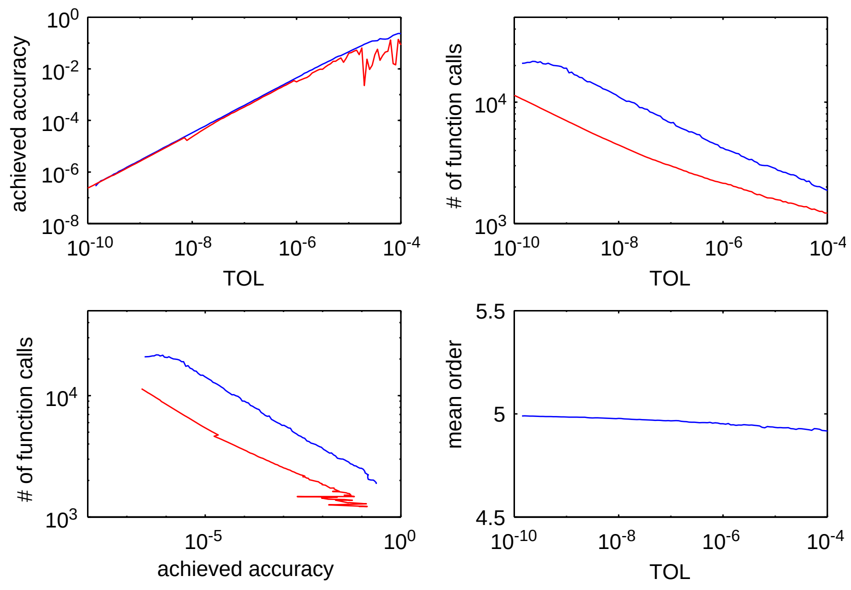

Fig. 5. The Arenstorf problem is used to compare modified versions of RADAU5 and DASPK, after tolerance calibration. Both achieve the same accuracy (top left) but DASPK is smoother. RADAU5 is more efficient as it uses about half the work to complete the integration (top right, bottom left). The mean order used by DASPK (bottom right) is very close to $p = 5$.

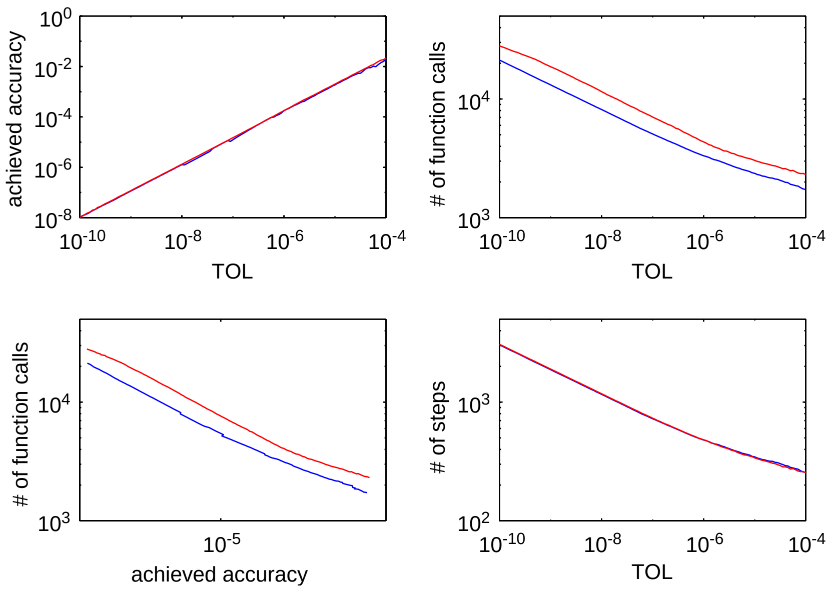

Fig. 6. The modified version of Radau5 is used to solve the Arenstorf problem with two different norms. In order to produce the same accuracy, individual tolerance calibrations are applied. The new norm constructed from (17) improves work–precision performance although the number of steps is the same.

to (4), using $\alpha = 6/5$ for Radau5. In addition, tolerance calibration was used, taking $\mathrm{TOL}_0 = 500$ for Daspk and $\mathrm{TOL}_0 = 1$ for Radau5, in order that the two codes produce approximately the same accuracy at equal tolerances. The codes were used to solve the Arenstorf problem [9, p. 130], using the new error norm with $\eta = 10^{-10}$. The comparison of the two codes is presented in Fig. 5

The next test concerns how the norm alone affects the code's behaviour. This is evaluated in the modified Radau5 code and plotted in Fig. 6 . As the norm affects the accumulated error, a tolerance calibration is necessary in order to conclude whether the norm has a beneficial influence on performance. For the standard norm, run in fixed scaling mode with $\eta = 1$, we use $\mathrm{TOL}_0 = 1700$. For the new norm, which has overflow protection at $\eta = 10^{-10}$, the value $\mathrm{TOL}_0 = 20$ produces the same global accuracy. Thus the tolerance calibration is set to produce higher accuracy than in the previous comparison with Daspk; the latter code is unable to produce this higher accuracy. Because of the different tolerance calibration, achieved accuracy is 30 times higher in Fig. 6.

5. Some experiments with Radau5

Continuing the above investigation on how to evaluate and compare ODE/DAE codes, we have followed the same procedure for modifying the Radau5 code of [10] as we did in [15] with Daspk. To follow the *ceteris paribus* approach, we use the same digital filter and limiter, but tolerance rescaling has to be adapted individually to method properties. Also, the Newton termination criterion varies slightly as Radau5 shows some sensitivity to this criterion.

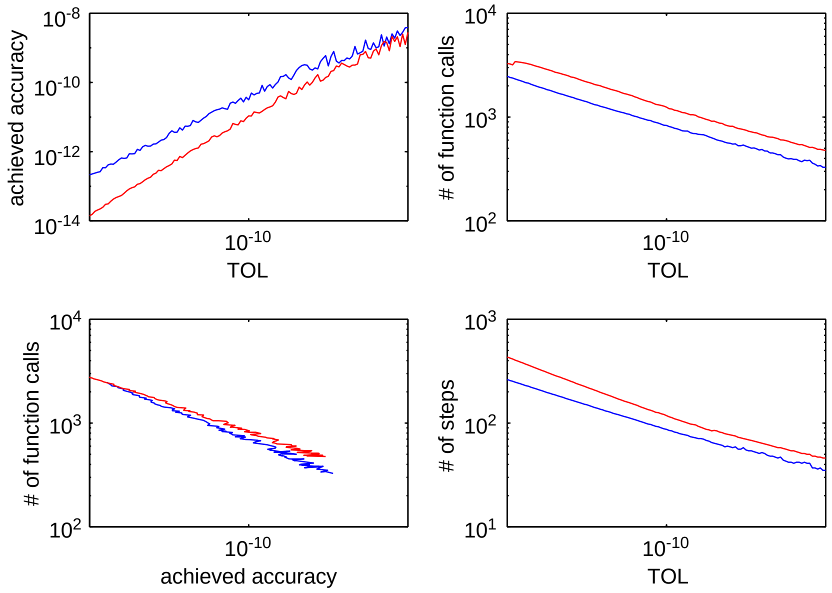

Fig. 7. The Chemakzo problem is solved using the original and the modified RADAU5 codes. The main difference is in tolerance proportionality. Both codes show a similar smoothness and computational stability, although the modified code is marginally less efficient for loose tolerances as more steps and more function evaluations are needed to complete the integration.

Our comparisons will focus on some qualitative differences in the behaviour of RADAU5 and DASPK. With respect to a few details, the former code displays some phenomena that are not observed in DASPK.

Our first test is concerned with RADAU5 alone. More precisely, we wish to evaluate the difference between the original code and the modified one. In the relatively simple Chemakzo problem, we see the effects of the modifications in Fig. 7. and 9 These tests were run with fixed scaling ($\eta = 1$). The goal is to establish that the new control structures can be implemented without loss of efficiency.

The objective of the second test is to determine whether stiffness has an effect on performance. We consider the linear equation

$$\dot{y} = A(y - \varphi(t)) + \dot{\varphi}(t), \tag{18}$$

where $\varphi(t) = (\sin t, \cos t)^{\mathrm{T}}$ and $y(0) = (0, 1)^{\mathrm{T}}$. The problem is solved on $[0, 2\pi]$, and because of the initial condition the exact solution equals $\varphi(t)$.

Here we use modified versions of both RADAU5 and DASPK. The problem is solved for two different matrices, corresponding to a nonstiff and a stiff case, with

$$A = \begin{pmatrix} -1 & 5 \\ 1 & -10 \end{pmatrix}, \quad A = \begin{pmatrix} -1 & 5 \cdot 10^3 \\ 1 & -10^4 \end{pmatrix}, \tag{19}$$

respectively. The accuracy criterion is fixed scaling, with $\eta = 1$, corresponding to running with RTOL = ATOL. The codes were tolerance calibrated for the nonstiff case by taking $\mathrm{TOL}_0 = 10^{-5}$ for DASPK and $\mathrm{TOL}_0 = 20$ for RADAU5; at this setting the same accuracy is produced by the two codes. That calibration is then kept for the stiff case. There, DASPK shows a behaviour unaffected by stiffness, while

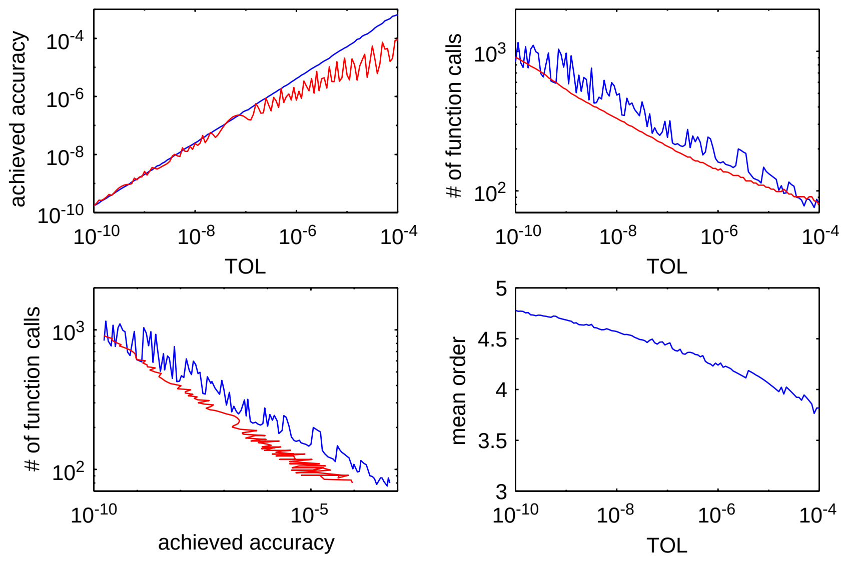

Fig. 8. Nonstiff linear problem solved with Daspk and Radau5. In the top left plot, the smooth graph was generated by Daspk. In the work-TOL and work-precision graphs, the smoother curves are those of Radau5. The lower right plot shows mean order for Daspk.

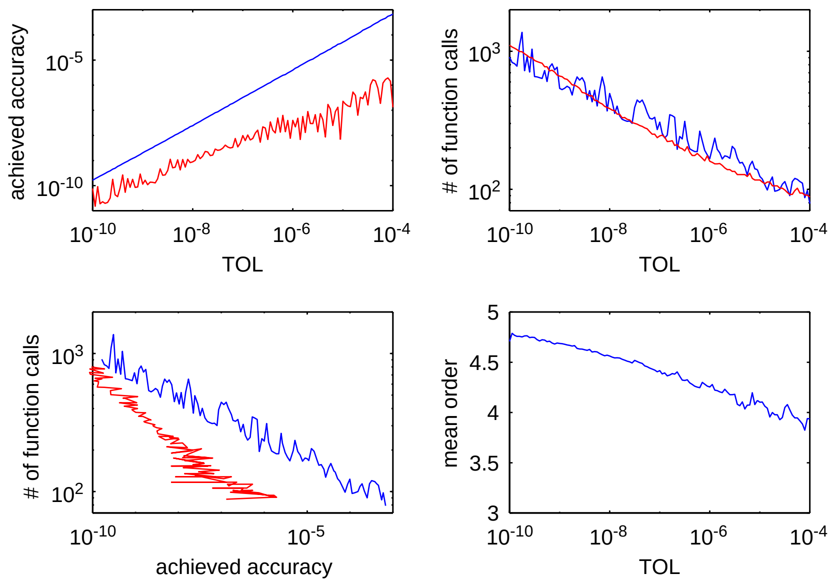

Fig. 9. In the stiff case, the two codes differ markedly. Daspk's behaviour is invariant with respect to stiffness, and its results can hardly be distinguished from those in the nonstiff case. Radau5, on the other hand, improves its accuracy, while cost is largely the same. Thus performance increases, but tolerance proportionality is also lost. Tolerance calibration is the same in both the nonstiff and the stiff tests.

RADAU5 improves both accuracy and efficiency, while at the same time losing tolerance proportionality, see Figs. 8.

Computational stability is high for DASPK in the TOL-precision graph, but only moderate for RADAU5, in spite of new stepsize algorithms. Even more surprising is that the converse holds for work, which is very regular for RADAU5, cf. the top right graphs of Figs. 8 and 9.

There are however some remaining problems in RADAU5. It appears to be difficult to obtain the same computational stability in RADAU5 as in DASPK. This could be caused by a lower intrinsic computational stability. On a few problems, such as the Pollution problem from [16], moreover, RADAU5 shows a high sensitivity to the Newton termination criterion and to the initial stepsize `h0`; changing these parameters may cause highly erratic TOL-precision graphs.

The accuracy may vary by several orders of magnitude for a small change in `h0` or the Newton termination criterion. As the variations do not appear to be stochastic but systematic, they indicate that the code's control structures have strong dependencies on these parameters. The spurious behaviour is not eliminated by replacing the controllers. One possible cause for this behaviour is that for loose tolerances, the integration can be completed in a small number of steps, typically in the range 15–30. In such a case it can make a major difference if the integration is carried out in 20 or 25 steps.

6. Conclusions

The tests carried out in this paper, with the exception of the left graph in Fig. 1, have been produced with two modified versions of DASPK and RADAU5. The codes implement the adaptivity algorithms as specified in [15]. More specifically, this entails using the $H211b$ digital filter for generating step-sizes, including the recommended control error limiter; an unconstrained order control in DASPK; a sharper Newton termination criterion in DASPK and a slightly modified one in RADAU5; tolerance rescaling and calibration in both codes; options to run in fixed scaling and fixed resolution modes. Technical details beyond the specifications in [15] are available in [18].

This paper has focused on the importance of a systematic approach to ODE/DAE testing practices, as well as the impact of properly designed control structures. We have attempted to run stringent tests of the modified DASPK and RADAU5 codes, also comparing the two. The tests have emphasized a *ceteris paribus* approach, where we have tried to isolate certain phenomena, or code changes, to see what impact they have. The interpretation and use of norms have been discussed and a new norm constructed for application in e.g., celestial mechanics.

Although it is possible to make significant improvements to existing codes, we believe that much remains to be done in order to improve the reliability of ODE/DAE software in general. This work rests on eliminating heuristic algorithmic elements. The evaluation of improved algorithms must follow a test protocol in order to draw firm conclusions. The tests presented here show that this is far from trivial, and that it also requires much more care than a simple benchmarking.

Acknowledgements

The research was in part funded by the Swedish Research Council under contract VR 2002-5370 and by the Swedish Research Council for Engineering Sciences under contract TFR 222/98-74.

References

[2] J. Cash, Software for Initial Value Problems. See website http://www.ma.ic.ac.uk/jcash/IVP_software/finaldae/readme.html
[3] W.H. Enright, T.E. Hull, B. Lindberg, Comparing numerical methods for stiff systems of ODEs, BIT 15 (1975) 10–48.
[5] K. Gustafsson, Control theoretic techniques for stepsize selection in explicit Runge–Kutta methods, ACM TOMS 17 (1991) 533–554.
[6] K. Gustafsson, Control theoretic techniques for stepsize selection in implicit Runge–Kutta methods, ACM TOMS 20 (1994) 496–517.
[7] K. Gustafsson, G. Söderlind, Control strategies for the iterative solution of nonlinear equations in ODE solvers, SIAM J. Sci. Comp. 18 (1997) 23–40.
[8] E. Hairer, Testset for Stiff ODEs. See website http://www.unige.ch/folks/hairer/testset/testset.html
[9] E. Hairer, S.P. Nørsett, G. Wanner, Solving Ordinary Differential Equations I: Nonstiff Problems, second revised ed., Springer, Berlin, 1993.
[10] E. Hairer, G. Wanner, Solving Ordinary Differential Equations II: Stiff and Differential-algebraic Problems, second revised ed., Springer, Berlin, 1996.
[11] G. Hall, W.H. Enright, T.E. Hull, A.E. Sedgwick, DETEST: a program for comparing numerical methods for ordinary differential equations, Report No. 60, Department of Computer Science and Technology, University of Toronto, Toronto, 1973.
[12] T.E. Hull, W.H. Enright, B.M. Fellen, A.E. Sedgwick, Comparing numerical methods for ordinary differential equations, SIAM J. Numer. Anal. 9 (1972) 603–637.
[13] G. Söderlind, Automatic control and adaptive time-stepping, Numer. Algorithms 31 (2002) 281–310.
[14] G. Söderlind, Digital filters in adaptive time-stepping, ACM Trans. Math. Software 29 (2003) 1–26.
[15] G. Söderlind, L. Wang, Adaptive time-stepping and computational stability, 2003, to appear.
[16] J.J.B. de Swart, Test set for IVP solvers, CWI, Amsterdam 1995. http://hilbert.dm.uniba.it/∼testset
[17] Ch.W. Ueberhuber, Numerical Computation 1, Methods, Software, and Analysis, Springer, Berlin, 1997.
[18] L. Wang, Computational Stability and Adaptive Strategies. An experimental study of ODE software, (Doctoral Theses in Mathematical Sciences 2003:3; ISBN 91-628-5318-X.) Numerical Analysis, Lund University, Sweden.

Further reading

[1] K.E. Brenan, S.L. Campbell, L.R. Petzold, Numerical Solution of Initial Value Problems in Differential-Algebraic Equations, SIAM Classics in Applied Mathematics 1996.
[4] C.W. Gear, Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, Englewood Cliffs, 1971.