Cambridge International
Science Publishing
Cambridge CB1 6AZ
Great Britain

Journal of Computational Methods in Sciences and Engineering
vol. 2, no. 3, 2, pp. 1–3
ISSN 1472–7978

Adaptive Time-Stepping and Computational Stability

Gustaf Söderlind[1,2]

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

Lina Wang[3]

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

Received 30 March, 2003; accepted in revised form 10 December, 2003

Abstract: We investigate the effects of adaptive time-stepping and other algorithmic strategies on the computational stability of ODE codes. We show that carefully designed adaptive algorithms have a most significant impact on computational stability and reliability. A series of computational experiments with the standard implementation of DASSL and a modified version, including stepsize control based on digital filters, is used to demonstrate that relatively small algorithmic changes are able to extract a vastly better computational stability at no extra expense. The inherent performance and stability of DASSL are therefore much greater than the standard implementation seems to suggest.

Keywords: Computational stability, stepsize control, adaptive time-stepping, PI control, digital filters, DASSL, mathematical software, algorithm analysis

PACS: PACS numbers of the paper

Mathematics Subject Classification: 65L05 Numerical Analysis, Ordinary Differential Equations—initial value problems; multistep methods

1 Introduction: Computational stability

The objective of this paper is to investigate the effect of adaptive time-stepping and other algorithmic strategies on what we refer to as the *computational stability* of an ODE solver. All stability notions are concerned with a continuous data dependence. As for computational stability, we ask that the complete algorithm, as well as its implementation (including all internal decision making of the code), is a "continuous" map from data to computed results, provided that the given ODE being solved is smooth enough. In other words, *a small change of parameters should only have a small effect on the computed output*. Alternatively, this could be expressed by saying that the computational process is well-conditioned.

Ideally, this should hold for any parameter, regardless of whether it is a parameter of the ODE itself, a parameter in the algorithmic specification of the code, or a parameter in the calling sequence of the software. As it concerns *software*, the notion of computational stability goes beyond the classical well-conditioning requirement of an *algorithm*.

[1]Gustaf.Soderlind@na.lu.se
[2]Corresponding author
[3]Lina.Wang@na.lu.se

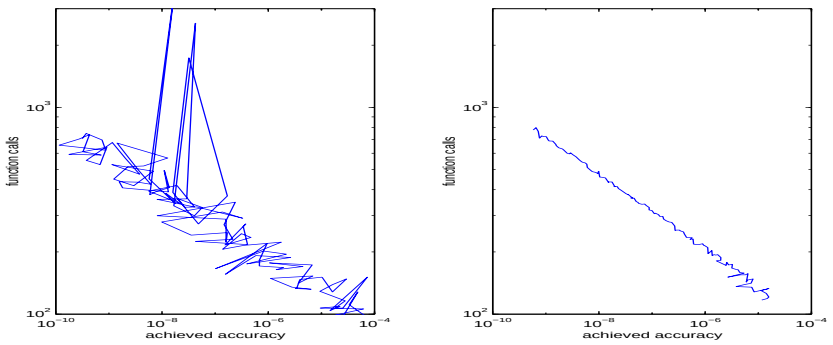

Figure 1: Work–precision diagrams. As TOL varies from 10^{-4} to 10^{-10} through 119 intermediate values, work (function evaluations) is plotted vs achieved accuracy (number of significant correct digits at the final solution point) for the Chemakzo test problem. DASSL with original adaptivity heuristics (left) and with modified strategies based on digital filters (right). The low regularity (left) corresponds to an uncertainty of some 1.5 orders of magnitude in precision and 50% in work, even if the three outliers at the top of the diagram are disregarded; the lack of computational stability is mainly due to the zero order adaptivity in the stepsize control. Modified strategies (right) keep both uncertainties below 10%, leading to good computational stability—performance is now predictable as a function of TOL, which may be used for extrapolation.

In theory, an algorithm refers to a complete description of how data are transformed into output. However, numerical "algorithms" are usually idealized and incomplete; e.g., round-off, necessary protection against divide-by-zero and other exception handling are often omitted, [13, Ch. 6]. Moreover, there may be unspecified sub-algorithms such as computing finite difference approximations of Jacobians, iteration termination criteria, scaling and factorization, etc. Some algorithmic elements are adaptive and depend on problem properties, others are left open as "user-defined" options, e.g. the choice of norm for error measurement. Therefore a "well-conditioned algorithm" is an idealized mathematical notion, and its actual implementation might show a much inferior computational stability.

As we shall see in some examples, see e.g. Figure 1, this does happen in ODE software. Thus, if adaptive strategies are based on heuristics with a lot of logic and intermittent decision making, the algorithm's potentially continuous data dependence may never be reached and is sometimes highly degraded. It is therefore important to develop strategies that are amenable to and supported by mathematical analysis, and provide a smoother behavior.

To make the notion of computational stability more precise, we shall define some fundamental accuracy criteria. Accuracy is measured in terms of the global error, which can be affected by an external accuracy control parameter, TOL.

1. **Convergence.** For each sufficiently smooth problem and any prescribed $\varepsilon > 0$ the algorithm/code should be able to produce an approximation with a global error $\|e\| \leq \varepsilon$, given sufficient CPU time and a sufficiently accurate arithmetic.

2. **Continuity.** For each sufficiently smooth problem and for any $\varepsilon > 0$ there should be a $\delta > 0$ such that $TOL \leq \delta \;\Rightarrow\; \|e\| \leq \varepsilon$.

3. **Tolerance proportionality.** For each sufficiently smooth problem there should be constants c and C such that $c \cdot TOL \leq \|e\| \leq C \cdot TOL$.

These requirements represent progressively stringent conditions on the behavior of the code.

Convergence implies that the code potentially can obtain arbitrarily small errors. Continuity requires that TOL is a means to achieve that goal. However, for a regular behavior we need some form of one-to-one correspondence between the required and the achieved accuracy. This is covered by the notion of tolerance proportionality. As real computations invariably incur some numerical "noise," e.g. caused by roundoff or prematurely terminated Newton iterations, we allow a slack in the proportionality; the smaller we can take C/c, the better.

Computational stability requires that C/c is small. Typically, $C/c < 2$ would correspond to a fairly good computational stability. But computational stability does not require tolerance *proportionality*—we shall see that one often encounters a computational behavior of the type

$$c \cdot TOL^\alpha \le \|e\| \le C \cdot TOL^\alpha \tag{1}$$

for some positive α. As long as C/c is small, however, this is perfectly acceptable, as it implies that in a log–log diagram of achieved error vs. TOL, we nearly obtain a straight line. All computed results are then found within a narrow band, of width $\log_{10} C - \log_{10} c$. From examples, such as the one in Figure 1, we shall see that the same numerical method, applied to the same ODE, but employing different control strategies, may require $\log_{10} C - \log_{10} c \ge 1.5$ for one choice of strategies while $\log_{10} C - \log_{10} c \le 0.05$ can be reached for another, although total computational costs are the same in both cases. Thus the design of adaptive strategies has a strong impact on computational stability and the conditioning of the software. It is therefore a nontrivial task to construct a good code.

A code should produce an error $\|e\| \le \varepsilon$ while trying to minimize the total computational effort. Computational stability will also require that work is a sufficiently regular function of TOL. In smooth problems, $\log_{10} WORK$ can be expected to be very nearly an affine function of $\log_{10} TOL$. Work-precision diagrams therefore become regular as well. Just like with accuracy, however, the smoothness of work as a function of TOL varies significantly with the control strategies.

Performance is, at large, the solution *quality* produced by the code, in terms of stability and accuracy, *per unit computational cost*. It can be increased in two different ways: either one reduces the cost for a given delivered quality, or one increases the quality for a given cost. Now, as it is easier to increase quality than to reduce costs, we will develop adaptive algorithms that do not change either the achieved accuracy or the computational cost (hence not affecting wall-clock time), but significantly improve computational stability, quality and reliability.

We will focus on computational stability in the sense that a small change of the tolerance parameter TOL should only have a small effect on the accuracy delivered by the code, as well as on the work required to achieve that accuracy. The new adaptive strategies will be based on the recent approach of using digital filters for stepsize selection [12]. The objective is to develop that theoretical framework into a full set of strategies and verify them in a real code applied to approved test problems. Thus we choose to study these phenomena and algorithms in the well-established and widely used ODE code DASSL, [2], and take test problems from the CWI test set, [10]. Such tests, like in Figure 1, indicate that DASSL is not a good code. This conclusion is wrong, however. Thus we will propose some minor algorithmic changes that are able to extract a vastly better computational stability from DASSL at no extra expense. DASSL's potential performance and stability are therefore far greater than Figure 1 suggests.

We first investigate the effects of first order adaptive time-stepping on computational stability, and then proceed to investigate how to choose an appropriate termination criterion for Newton iterations. These two aspects are tested extensively using a single test problem, the Chemical Akzo Nobel ("Chemakzo") ODE system [10], and to simplify the testing, we choose the DASSL parameters $ATOL = RTOL = TOL$ in all test runs, and thus have a single scalar tolerance parameter TOL in our graphs. This setting corresponds to a realistic scaling of the test problem. In this way we arrive at new strategies and parameterizations of DASSL, and the modified code is finally tested on a few more problems and compared to the standard implementation.

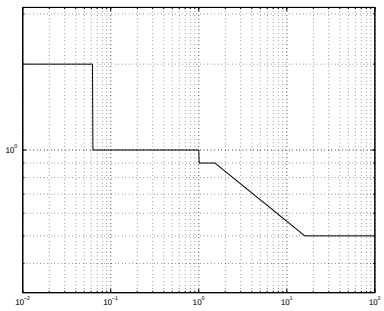

Figure 2: Stepsize change ratio $\log(h_{n+1}/h_n)$ as a function of the error excess $\log(\hat{r}_n/\varepsilon)$ in DASSL; the graph captures all essential features of the implemented stepsize selection strategy, which has *limiters* (saturation) to prevent too large stepsize increases/decreases, and *cut-outs* (dead-zones) at the origin to prevent small stepsize changes. The only permissible stepsize increase is a step doubling; the elementary stepsize change rule (2) is used only for stepsize reductions after a rejected step. The strategy is nonlinear, discontinuous and unsymmetric.

2 Common stepsize strategies and their effects

The most common way to adapt the stepsize to local variations in the solution is the *elementary controller*

$$h_{n+1} = \left(\frac{\varepsilon}{\hat{r}_n}\right)^{1/k} h_n, \tag{2}$$

see e.g. [3, 11]. Here ε is a suitable fraction of TOL and k is p or $p+1$ (depending on the control objective), where p is the order of the local error estimator $\hat{r}_n$. By taking logarithms, we obtain

$$\log(h_{n+1}/h_n) = -\frac{1}{k}\,\log(\hat{r}_n/\varepsilon). \tag{3}$$

Hence the stepsize change ratio $\log(h_{n+1}/h_n)$ is negatively proportional to the error excess $\log(\hat{r}_n/\varepsilon)$; this is an example of linear negative feedback control. It may be regarded as the simplest possible control for the asymptotic error model.

The elementary controller is however rarely if ever implemented in this way; it is augmented with limiters and other types of exception handling. An example representative of today's codes is given in Figure 2, which shows the error–stepsize map of DASSL. As (3) shows, this map should ideally be a straight line of slope $-1/k$, but in Figure 2 logic and decision-making override the controller (2), completely preventing its use in normal situations. Instead, step doubling is the only option for increasing the stepsize; this means that a *cut-out* or *dead-zone* in the error–stepsize map is used to prevent small stepsize increases, and a *limiter* or *saturation* is used to prevent increases larger than a factor of 2. Likewise, if the error is a little bit too large, the stepsize is reduced by a factor of 0.9, i.e., there is a cut-out also to the right of the origin. Should this not suffice, the step is rejected; this is the only situation when (2) is used, although there is a limiter to prevent the stepsize from being reduced by more than a factor of 2.

Lack of computational stability, see Figure 1, is typically caused by such a highly discontinuous and nonlinear stepsize control algorithm. Consider the major cut-out in Figure 2, which is plotted for $k = 4$. The local error estimate $\hat{r}_n$ may then be anywhere in the interval $[\varepsilon/2^k, \varepsilon]$, *without invoking any control action*. Thus, the higher the order, the larger is the dead-zone. As a consequence the interval of "uncertainty" (i.e., what the actually produced accuracy might be) grows. At $k = 5$, the interval corresponds to a factor of 32, or 1.5 orders of magnitude, in good agreement

with observations in Figure 1. As no control action is taken in this interval, the controller (2) is *disengaged*. It is therefore unable to force the local error to ε. This implies that the adaptivity order is $p_A = 0$, [11, 12]. The adaptivity order p_A measures the contoller's ability to eliminate a deviation between $\hat{r}_n$ and ε. Thus if the principal error function is a constant, then a $p_A = 1$ controller changes the stepsize until $\hat{r}_n \equiv \epsilon$, after which h_n as well becomes constant; for $p_A = 2$ and higher, the controller must also be able to completely eliminate the deviation when the logarithm of the prinicipal error is a polynomial of degree $p_A - 1$ in n.

With the low computational stability of the left plot of Figure 1, even very small changes in the computational setup have visible effects. For example, rewriting numerical constants in the code in such a way that only roundoff is different, or exchanging compilers (which may have a similar effect) can be sufficient to change the computational history and produce a distinctly different work-precision diagram. In fact, the exact details of the left plot of Figure 1 are nearly irreproducible.

There are various motivations for discontinuous strategies of the type shown in Figure 2. *Limiters* are needed as there might occur error estimates which are far too large or small, and such exceptional events should not be allowed to affect the stepsize. Another reason to include limiters is that it is known for multistep methods that the stepsize variations must be limited or the method may become unstable; this argument is however much overemphasized, since *closed-loop* control is used, i.e., if the error estimator shows signs of instability, any properly chosen feedback controller will automatically detect and counteract such effects. In fact, the stepsize doubling allowed in DASSL violates the maximum stepsize increase for high order BDF methods, but to our knowledge it has never been observed that the implemented methods go unstable.

As for the *cut-outs*, it has been argued that small stepsize changes are not worthwhile, as they require recomputations of method coefficients and they may also force refactorizations of Jacobians. The former argument never holds except possibly for ODEs with only a very few equations; the latter is not very significant either, as a small stepsize variation can easily be accommodated in the Newton iterative process, see [6] or the actual strategy used in DASSL [2], which is based on an over/under-relaxation, compensating for the changed stepsize. The only more significant argument for a cut-out seems to be that there are some multistep method implementations whose coefficients have singularities for certain small stepsize ratios, [1, 9]. This is however not the case for BDFs.

Linked to the cut-outs is the choice of *termination criterion* for the Newton iteration. In DASSL the iteration is terminated when the estimated remaining iteration error is less than $TOL/3$. Consider Figure 2 again. Immediately before a stepsize increase by a factor of 2, the (truncation) error must be on the order of $\varepsilon/2^k$, which is typically much smaller than the remaining Newton error. It is therefore unlikely that the stepsize increase takes place at all, unless the problem is linear or the Jacobian matrix has just been re-computed. Otherwise the error estimate will be dominated by a large and irregular iteration error; this can only be reduced by using a much tighter termination criterion. Alternatively, one should increase the stepsize much sooner. Below we will find that a combination of these two changes has a very positive effect.

Varying the *order* is a further complication. This takes place only intermittently, and is therefore discontinuous by nature. Most multistep codes do not allow the order to be changed too frequently, and in particular DASSL requires the order to be kept the same for several steps following a stepsize change. We shall see that such a strategy has to be abandoned in order to obtain a smooth work-precision diagram and a good performance. In general, the intermittent decision-making ought to be kept to a minimum.

Last, but not least, it should be recognized that discontinuous stategies such as the one depicted in Figure 2 have worked surprisingly well since they became "standard practice" around 1970, and it has only recently been noticed that they create computational instability. While limiters are benign (and should be included as safety nets), the problems are primarily caused by cut-outs. By removing them we will be able to use first order adaptive control.

3 First order adaptive stepsize control

We shall investigate how computational stability is affected by adaptive stepsize selection. The adaptivity will be based on control theory, [4, 5, 11] (see also [7]), including the recent approach of using digital filters, [12]. These are all covered by the generic stepsize recursion

$$h_{n+1} = \left(\frac{\varepsilon}{\hat{r}_n}\right)^{\beta_1} \left(\frac{\varepsilon}{\hat{r}_{n-1}}\right)^{\beta_2} \left(\frac{h_n}{h_{n-1}}\right)^{-\alpha_2} h_n, \tag{4}$$

if the dynamics is limited to at most second order, [12]. The parameters $k\beta_1$, $k\beta_2$ and α_2 are selected in accordance with *order conditions* for adaptivity and low-pass filtering, [12].

$k\beta_1$	$k\beta_2$	α_2	Name
1	-	-	elementary controller, [3]; used in DASSL
3/5	$-1/5$	-	PI.4.2 controller, [11]
1/4	1/4	1/4	$H211b$ digital filter ($b = 4$), [12]

Table 1: Three first order adaptive controllers

We shall here work with three different controllers whose parameterizations are found above in Table 1. All three are first order adaptive [11, 12] if properly implemented, i.e., *the use of cut-outs is precluded*. An adaptivity order $p_A \geq 1$ is necessary for the stepsize control algorithm to be able to adapt the error (stepsize) to the required precision as given by TOL and avoid the loss of computational stability seen in Figure 1.

A major cause of irregularities in the TOL-precision graphs is the combination of a cut-out and an order strategy that prevents order increase after a stepsize change. Decreasing the size of the cut-out aggravates the problem. For example, for some values of TOL (even when small!) the code only uses the implicit Euler method (Figure 3). However, if the cut-out is removed, the adaptivity order becomes $p_A = 1$, and the stepsize changes continually. This yields a regular TOL–precision graph (this is so for all fixed order methods), as the code cannot increase the method order at all and the implicit Euler method is the only one used. The interaction between stepsize and order controls must therefore be reconsidered. We shall therefore remove DASSL's restriction on order change: a single line of code is removed to allow the code full freedom in changing the order. The result is seen in Figure 4.

Computational stability increases significantly. The modified strategy produces a TOL-precision graph regularity better than $\log_{10} C - \log_{10} c < 0.1$. Consequently, the achieved precision is not only predictable: as $\|e\| = (1 \pm 0.1) \cdot TOL^{\alpha}$, it even allows extrapolation on TOL in order to produce a one-digit global error estimate.

When the method order is allowed to change freely, it is also of interest to find out what order the code is using, on average. We define the *mean order*

$$\bar{p} = \frac{\Sigma_p \, p \cdot n_p}{\Sigma_p \, n_p}, \tag{5}$$

where n_p is the number of steps taken with method order p. For each run, $\bar{p}$ will vary, and in Figure 4 we have plotted how $\bar{p}$ varies with TOL. Except for loose tolerances, we find that a mean order between 4 and 5, showing that the combined stepsize and order control works fine. We observe *no instability tendencies at all* due to frequent stepsize/order change.

The removal of cut-outs and order change prohibitions from the original code replaces the intermittent character of the adaptive strategies by smooth, gentle actions. There is no loss of efficiency in doing so, but a large gain in computational stability and regularity.

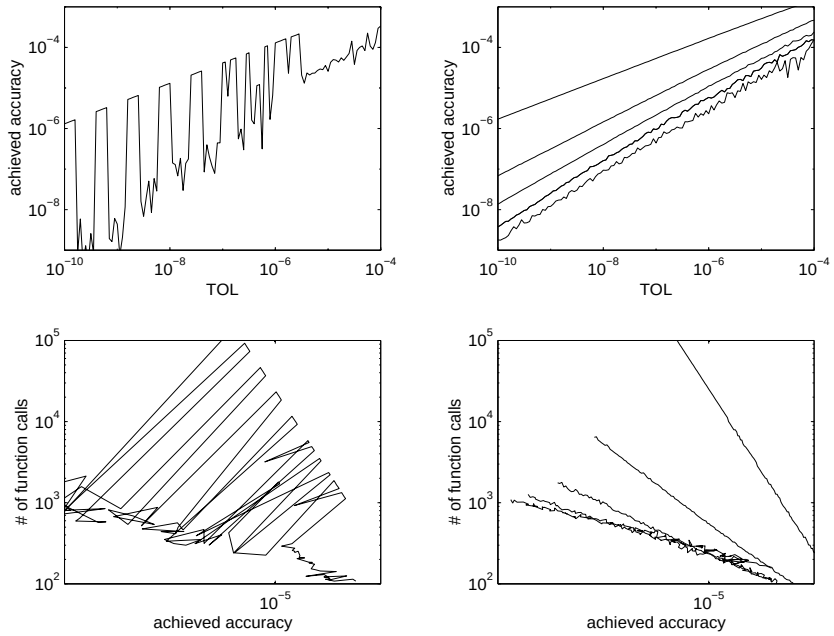

Figure 3: The Chemakzo test problem is solved when TOL varies from 10^{-4} to 10^{-10} through 119 intermediate values. The plots on the left show DASSL with a small cut-out of size 1.01 (stepsize increases below 1% are prohibited) but otherwise unchanged. Achieved accuracy varies significantly. As a result, the work-precision diagram (lower left) is highly erratic. Removing the cut-out completely results in the upper graph in the top right diagram where the method order is constant at $p = 1$. The order of adaptivity in this regular graph is now $p_A = 1$. The lower graphs in this plot correspond to letting DASSL run with fixed method orders, no cut-outs and the elementary controller. From top to bottom $p = 1, 2, 3, 4, 5$. Finally, the work-precision diagram (lower right) is regular and shows how efficiency varies with order.

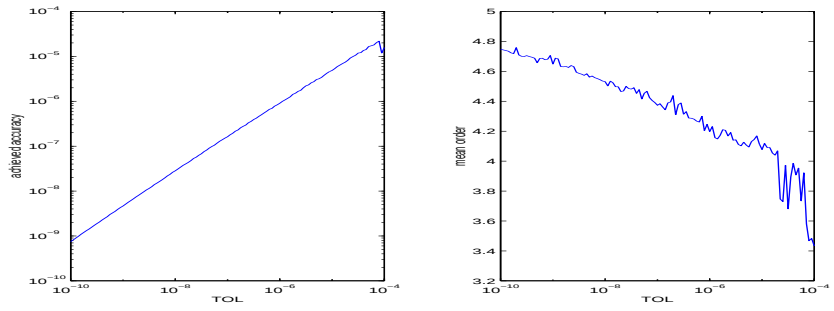

Figure 4: The left plot shows achieved precision vs. TOL (Chemakzo problem) for DASSL with stepsize controller $H211b$ ($b = 4$) without cut-out and with free order change as TOL varies from 10^{-4} to 10^{-10}. Once the order of adaptivity is $p_A = 1$, the TOL-precision diagrams are regular. The right plot shows the mean order $\bar{p}$ vs. TOL for the same test.

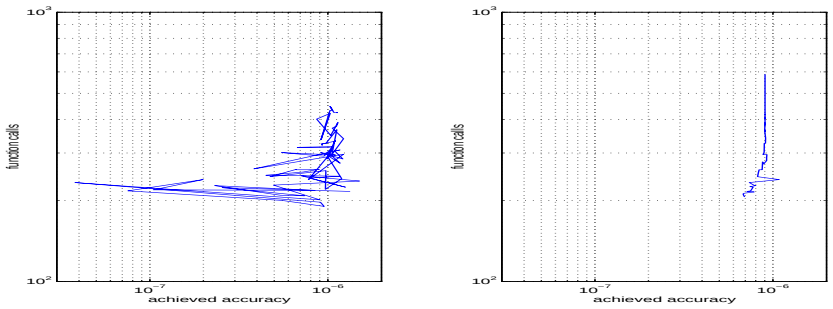

Figure 5: Work–precision diagrams. The Chemakzo problem is solved with DASSL at $TOL = 10^{-6}$ when the Newton termination criterion varies; θ runs from $10^{-0.5}$ to 10^{-8} through 74 intermediate values. For values $\theta > 10^{-2}$, the total work remains largely unaffected in the original DASSL (left) while the achieved accuracy at the endpoint varies by up to 1.5 orders of magnitude. As θ decreases, however, the achieved accuracy stabilizes near $9 \cdot 10^{-7}$, and the required work grows. With modified stepsize selection using the digital filter $H211b$ (right), the achieved accuracy is virtually independent of θ; decreasing θ only increases total number of function calls. The endpoint accuracy of $9 \cdot 10^{-7}$ is completely stable, and the modified code can employ a relatively loose termination criterion without loss of computational stability.

4 Termination criterion for Newton iterations

When an implicit method is applied to an ODE, a nonlinear equation

$$y = \gamma h f(y) + \psi \tag{6}$$

must be solved on each step. Apart from choosing between Newton and fixed point iterations, there are basically two different approaches: using a prescribed number of iterations or "iteration until convergence." The latter approach is usually preferred, using a termination criterion related to the required accuracy as specified by TOL. (In addition there is a maximum permissible number of iterations.) In most cases the termination criterion is formulated as

$$\|y^k - y^*\| \leq \theta \cdot TOL, \tag{7}$$

where y^k is the kth iterate, y^* is the solution to (6), and θ is a suitable fraction. As $y^k - y^*$ is not computable, it is estimated by a quantity Δy^k, which is required to be less than $\theta \cdot TOL$ in magnitude. The question we shall deal with here is how to choose θ, and to what extent this choice affects computational stability—Newton errors are typically less regular than the truncation errors of the time-stepping method itself. In order not to upset the intrinsic computational stability of the method, one might therefore expect that θ must be kept fairly small. A common choice is to use $\theta = 1/10$, but there are many variants. A few codes adapt the value of θ to the method and the requested accuracy.

In DASSL, the choice is $\theta = 1/3$. As Figure 5 shows, this is too loose a termination criterion, and there is a loss of computational stability. The latter can be improved by reducing θ. However, it must be reduced significantly as the achieved accuracy is not stable for $\theta > 10^{-4}$. But when the stepsize selection is replaced by a digital filter, it is possible to obtain very stable results. The achieved accuracy then depends only weakly on the choice of θ.

For a fixed, sharp termination criterion of $\theta = 10^{-7}$, a TOL-precision diagram is plotted in Figure 6. Even at this level, the original DASSL code typically displays a 100% variation (a factor of 2) in achieved accuracy for small changes in TOL. However, when this termination criterion

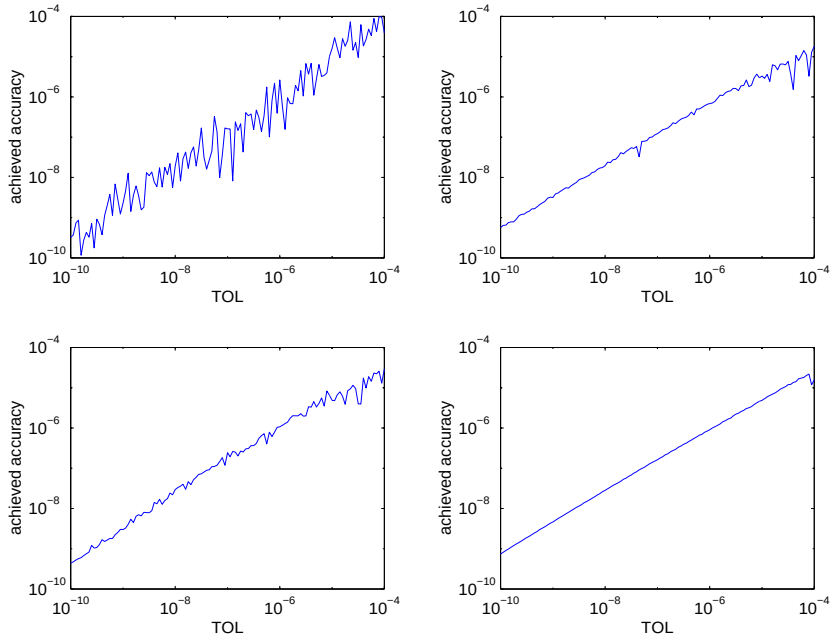

Figure 6: Achieved accuracy vs tolerance in the Chemakzo problem. DASSL with original stepsize strategy (left) and modified DASSL with $H211b$ stepsize controller (right). In the top graphs the original Newton termination criterion of $\theta = 1/3$ was used; in the lower graphs it was sharpened to $\theta = 10^{-7}$. The tighter termination criterion improves regularity in both cases, but as regularity is better in the top right graph than in the lower left, it is seen that the stepsize controller is the primary means of improving computational stability.

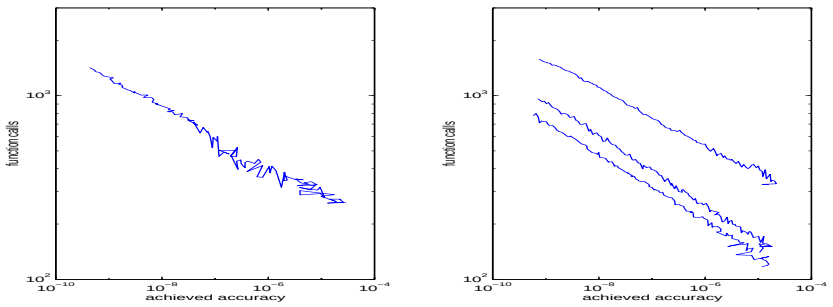

Figure 7: Work–precision diagrams for the Chemakzo problem. Original DASSL with Newton termination criterion sharpened to $\theta = 10^{-7}$ (left) and modified DASSL with $H211b$ stepsize controller (right). For the modified version the three graphs correspond to $\theta = 10^{-7}$, $\theta = 1/300$ and $\theta = 1/30$, from top to bottom. Computational stability is largely the same in this range, although total work increases as the termination criterion is successively sharpened.

is combined with first order adaptive time-stepping based on the digital filter $H211b$, variations become completely negligible.

In the conventional work–precision diagram Figure 7, a significant improvement of computational stability is observed for the original DASSL (cp. Figure 1), at the price of an increased overall work. It is however not possible to reach the same computational stability as one does with the modified version using $H211b$. Thus, a sharper termination criterion alone cannot save the original code.

Further tests with first order adaptive time-stepping based on digital filters indicate that θ should be in the range $1/30$ — $1/100$ (corresponding to $\theta \in [10^{-2}, 10^{-1.5}]$) for DASSL to perform well. In this range, computational stability is close to the limit of what the code can achieve: sharper termination criteria do not pay off.

We conclude that, while Newton errors are less regular than truncation errors, the Newton termination criterion must be selected properly in order to have good computational stability. But as the stepsize controller has a greater impact, it is necessary to use a first order adaptive stepsize strategy and eliminate cut-outs and intermittent strategies first. For DASSL we suggest using $\theta = 1/30$ in conjunction with any first order adaptive stepsize controller combined with free order change.

5 Tolerance proportionality

Tolerance proportionality implies that if TOL changes by one order of magnitude, then the actual error should also change by one order of magnitude. But as we have seen in several tests, (1) holds with a value of α less than 1. In Figure 3 we see that with cut-outs, it is in principle impossible to obtain a value of α at all; without cut-outs but constant method order, on the other hand, one may find a value for α, but it varies with the method order.

It is therefore rather surprising that with first order adaptive stepsize control and free order control, Figure 4 seems to reveal that a single, rather well defined, value for α can be found. There is also a possible problem dependency, but this appears to be rather weak. If TOL changes by 4 (or 5) orders of magnitude, the achieved accuracy changes by about 3 (or 4) orders of magnitude, indicating that α is to be found in the interval $[0.75, 0.8]$. Needless to say, this is an experimental result and not theoretically proved.

We shall use the value $\alpha = 7/9$, and introduce the logarithmically affine tolerance rescaling

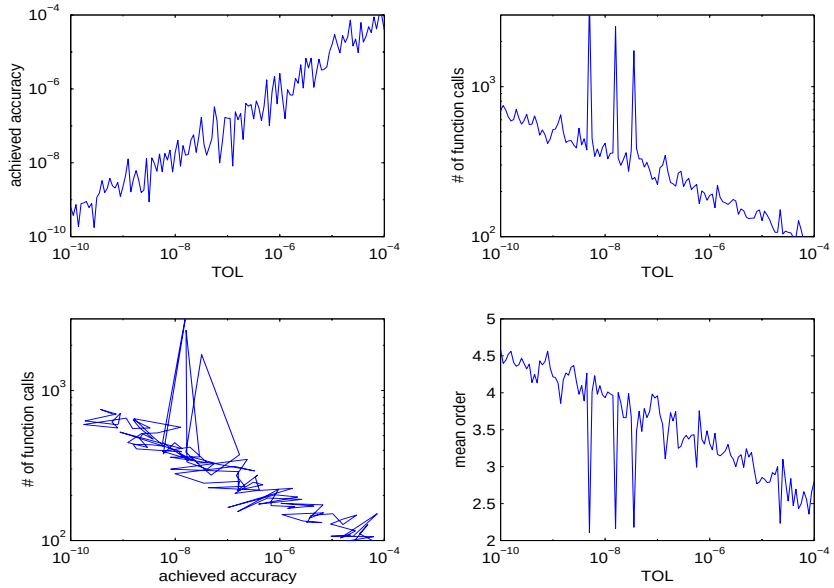

Figure 8: Original DASSL and the Chemakzo problem. Work shows 50% variations, even if outliers are neglected (top right). Mean method order shows abrupt variations and three spurious runs, where the order rarely if ever exceeds $p = 2$ (bottom right).

$TOL' = \tau \cdot TOL^{1/\alpha}$. We choose τ to make $TOL' = TOL = TOL_0$ at a suitable equivalence point TOL_0. Hence

$$TOL' = TOL_0^{(\alpha-1)/\alpha} \cdot TOL^{1/\alpha}. \tag{8}$$

Although the equivalence point is arbitrary, it may be reasonable to choose the value $TOL_0 = 10^{-5}$ as this is a fairly typical tolerance used in practice. With $\alpha = 7/9$ this leads to the approximate tolerance rescaling $TOL' = 27 \cdot TOL^{9/7}$.

The parameter sent to the code is still TOL, but internally the code calculates TOL', which is used to control the local error. The achieved global error will then be proportional to $(TOL')^\alpha = \tau^\alpha \cdot TOL$, i.e., the global error is now, as desired, proportional to the user-supplied TOL, see Figure 9. In this particular case, we obtain more than expected: there is a complete agreement between global error and tolerance ($\|e\| = TOL$) in Figure 9, but this is coincidental. There are also some wiggles for loose tolerances; they correspond to the fact that after tolerance rescaling the problem is effectively run over the interval $TOL' \in [4 \cdot 10^{-12}, 2 \cdot 10^{-4}]$, which is more than one and a half order of magnitude wider than the interval used before rescaling. At $TOL' = 2 \cdot 10^{-4}$ the integration is completed very quickly (about 50 steps or less), and even minor differences in stepsize selection and Newton iterations will have a visible effect on diagrams.

Performance in terms of work required to produce a certain accuracy remains unchanged. Total work remains unaffected at the equivalence point TOL_0. However, for $TOL' = \epsilon < TOL_0$, more work is required than if we had chosen $TOL = \epsilon$, as the tolerance proportional code will produce a higher accuracy at this setting. Conversely, for $TOL' = \epsilon > TOL_0$, comparatively less work is required.

The choice of the equivalence point TOL_0 could be done in several different ways. For example, a tolerance calibration could be based on the scalar linear test equation $\dot{y} = \lambda y$ with initial condition $y(0) = 1$ and to be solved on the interval $[0, 1]$. This approach would have important advantages as it would make it possible to calibrate *different codes* and make them not only tolerance proportional

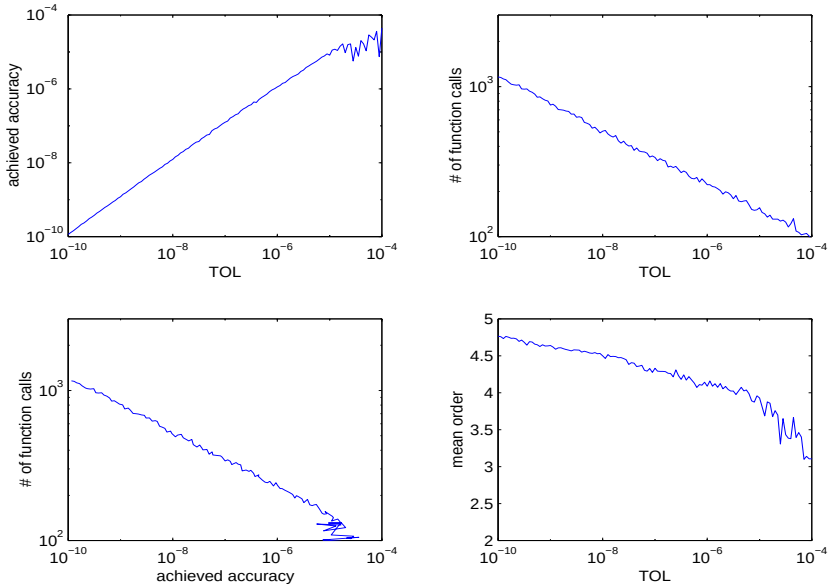

Figure 9: After tolerance rescaling, DASSL runs in tolerance proportional mode (top left). $H211b$ stepsize control, free order control and Newton termination criterion $\theta = 1/30$ are combined with a smooth limiter instead of sharp cut-offs. Work-precision diagram (bottom left) is largely unaffected by the tolerance rescaling, see Figure 7.

but also deliver the same accuracy in a reference case.

6 Limiters and anti-windup

As we saw in Figure 2, discontinuities are also introduced by limiting the maximum stepsize increase and decrease, respectively. Unlike the cut-out, however, such discontinuous limiters are relatively benign, as they come into effect only on rare occasions. Nevertheless, we prefer to introduce a smooth limiter.

In the present work, we implement the filter-based controller (4) using *control error filtering*, see [12]. The control is written $h_{n+1} = \rho_n h_n$ with

$$\rho_n = (\varepsilon/\hat{r}_n)^{\beta_1}\ (\varepsilon/\hat{r}_{n-1})^{\beta_2}\ \rho_{n-1}^{-\alpha_2}. \tag{9}$$

Although (9) is equivalent to (4), stepsize rejections may be reduced by basing the test on the requested change ρ rather than on the error itself; the sequence $\{\rho_n\}$ is smoother than the error sequence.

Moreover, by introducing a smooth nonlinear transformation, we can employ a smooth limiter that eliminates the previous discontinuities. First ρ_n is computed from (9). Then this value is limited so that it is bounded away from 0 and ∞ by a transformation $\hat{\rho}_n = \omega(\rho_n)$. Apart from being smooth and monotonically increasing, the function ω is required to have the following properties:

1. max stepsize reduction $= \omega(0) < 1$
2. $1 < \omega(\infty) =$ max stepsize increase
3. $\omega(1) = 1$ and $\omega'(1) = 1$.

The third condition ensures that during the controller's normal action, when $\rho_n \approx 1$, small variations in ρ_n are unaffected by ω. Hence in normal circumstances $\hat{\rho}_n \approx \rho_n$, i.e., the limiter has not effect there. We choose the limiter ω as

$$\hat{\rho}_n = 1 + \kappa \cdot \arctan[(\rho_n - 1)/\kappa], \tag{10}$$

and update the stepsize according to $h_{n+1} = \hat{\rho}_n h_n$. The parameter κ determines the max stepsize increase/reduction. The limiter's effect is reduced by increasing κ. Thus, the larger the value of κ, the wider is the interval where $\hat{\rho}_n \approx \rho_n$. The value of κ could be chosen to depend on the method order, if a smaller maximum increase is desired for higher order methods. Choosing $\kappa \in [0.7, 2]$ implies that stepsize increase is approximately limited to a step doubling, and that a reduction is limited to a factor of 3 — 5. In our tests we have settled for $\kappa = 1$. The actual value of κ does not appear to be crucial. However, it is to be noted that a larger κ may allow the stepsize to ramp up quicker after transients have decayed.

The introduction of nonlinear limiters can lead to problems for the controller, in particular if the limiter acts on the stepsize h_n rather than on the control error ρ_n. Such a limiter overrides the controller's action, which may lead to the controller trying harder and harder to change the stepsize without being allowed by the limiter to do so. This phenomenon, known as *windup*, must then be compensated by *anti-windup*, [14]. However, as we have chosen to limit ρ_n rather than the stepsize, there is no need for anti-windup in the present implementation as this choice does not restrict the controller's action.

There are also alternative ways of implementing the limiter. The most attractive appears to be to limit the error output $\{\varepsilon/\hat{r}_n\}$ of the computational process, before the digital filter is applied. This could even be done with a rational (second order) limiter, that would offer the added benefit of eliminating the need for protection against division by zero. However, this requires a major modification of DASSL, as the code unfortunately builds the tolerances ($ATOL$ and $RTOL$, and hence ε) into the norm definition. Modifying this could increase robustness further.

7 Summary of strategy modifications

The following sums up the strategy modifications of DASSL.

Order control. Remove the single line of code that prevents order change following a stepsize change. This is necessary as we employ at least first order adaptive digital filter stepsize controllers; these change the stepsize on every step, albeit by small factors.

Stepsize control. The following is a pseudo-code implementation of the stepsize controller used in the modified version of DASSL. It shows how to use the $H211b$ digital filter, in control error filtering mode [12], using the arctangent error limiter, with details of supplementary code in separate functions.

```
cerr = tol/r;
rho = filter('h211b', cerr, cerrold, rho, k);
ratio = limit(rho);
h = ratio * h;
cerrold = cerr
IF ratio < 0.9 THEN reject_step_and_recompute_with_new_stepsize
```

This requires the following functions, which offer three different stepsize controllers, $H211b$ digital filter, a PI controller, and finally the elementary controller, to be used during startup, when sufficient data for running controllers of second order dynamics are not yet available:

```
function limit(u)
kappa = 1;
limit = 1 + kappa * arctan( (u-1)/kappa );

function filter(filter_type, c1, c0, rho, k)
filter = filter_type(c1, c0, rho, k);

function h211b(c1, c0, rho, k)
% H211b digital filter, with standard parameter setting b=4
b = 4;
h211b = c1^(1/b/k) * c0^(1/b/k) * rho^(-1/b);

function pi42(c1, c0, rho, k)
% PI.4.2 controller
pi42 = c1^(3/5/k) * c0^(-1/5/k);

function elementary(c1, c0, rho, k)
% Elementary integrating controller for startup
elementary = c1^(1/k);
```

Note that by changing the powers in the function `h211b` we change the filter coefficients. The order dynamics is denoted by p_D, and $p_D = 1$ corresponds to a one-step contoller, such as the elementary controller. The $H211b$ is a two-step controller, hence $p_D = 2$. For higher order dynamics than $p_D = 2$, the previous value of `rho` must be saved. Likewise, the limiter can be modified by changing `kappa`. The step rejection criterion is based on the filtered control error and limited stepsize ratio, by not accepting major stepsize reductions without recomputing the step.

The control structures can in principle be implemented from this pseudo code, with one exception. When the order changes, the value of k changes. This implies that consecutive control errors (`c0` and `c1`) are derived from error estimates of different orders. The problem then arises of how to choose k in the filters with $p_D \geq 2$. We have chosen to always use the same k for both factors in $H211b$, in spite of `c0` and `c1` coming from methods of different orders. Experiments with other possibilities seem to indicate that this choice has little influence, probably because control errors are typically very small.

Newton termination criterion. The only change proposed here is to sharpen the termination criterion from $TOL/3$ to $TOL/30$, or alternatively, $TOL/100$.

Tolerance rescaling. We propose to include an option of using tolerance rescaling according to

$$TOL' = TOL_0^{-2/7} \cdot TOL^{9/7}, \tag{11}$$

using the equivalence point $TOL_0 = 10^{-4}$.

8 Additional tests and further work

The complete set of modifications have been implemented in Dassl and Daspk3.0. We here report a few more tests, first with the elementary and the PI.4.2 controllers replacing $H211b$ on the Chemakzo problem, to demonstrate that the code performs consistently with different types of first order adaptive controllers. Then, returning to $H211b$, we show results for two other problems, the Pleiades and Hires problems, both from [10]. For the latter two, the modified code is compared to the original. In all cases, the tests where run in a fixed scaling mode, corresponding to $ATOL = RTOL$, combined with tolerance rescaling.

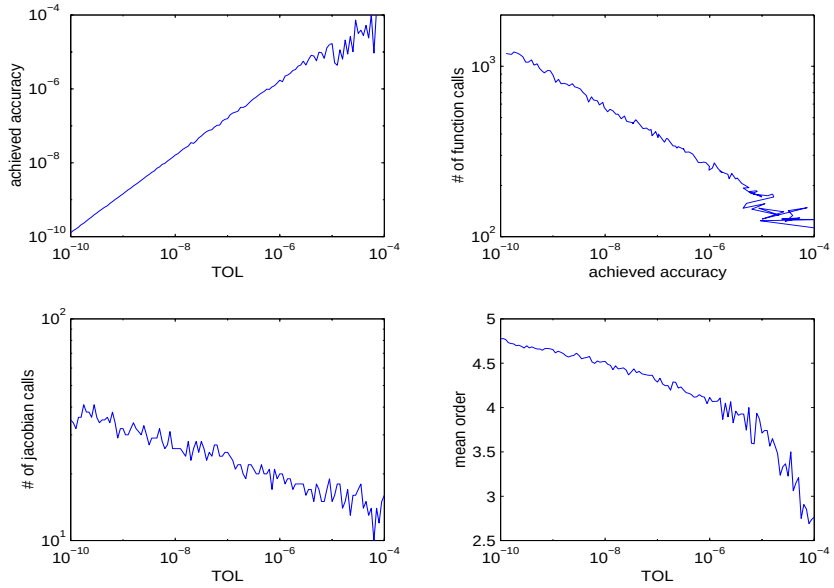

Figure 10: Modified code applied to the Chemakzo problem when first order adaptive elementary integrating controller is used. With this simple controller, no special startup procedure is required.

The algorithmic changes proposed in this paper could be further extended. It appears, however, that the most significant changes for improving computational stability have been included. Some of the points that will need further examination are: the startup procedure for two-step controllers; the interaction of order change and controller (i.e., how to choose k in the controller when the error estimates were obtained by methods of different orders); and possibly also employing an integral controller to select method order instead of a mere inspection of finite differences. Preliminary experiments with the first two of these issues have not indicated that the actual choice is crucial to computational stability, efficiency or accuracy. We therefore believe that the proposed changes are the most essential to implement in a state-of-the-art ODE code. As for DAE problems, some new control issues arise, but the basic techniques should remain the same.

9 Acknowledgements

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

References

[1] C. Arévalo, C. Führer and M. Selva. A collocation formulation of multistep methods for variable stepsize extensions, *APNUM* 42:5–16, 2002.

[2] 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 1996.

[3] C.W. Gear. Numerical Initial Value Problems in Ordinary Differential Equations. Prentice–Hall, Englewood Cliffs 1971.

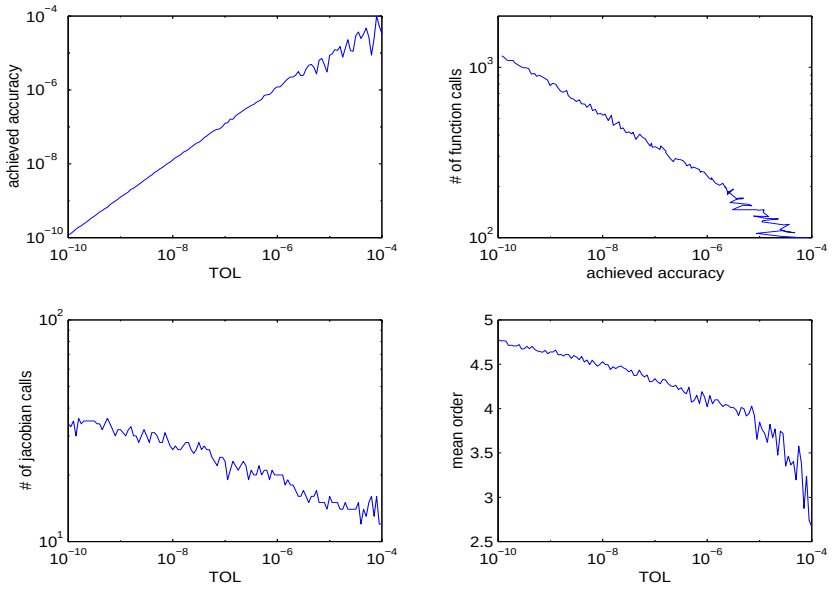

Figure 11: Modified code applied to the Chemakzo problem when first order adaptive PI controller PI.4.2 is used. Compared to the elementary controller, there is a marginal improvement of efficiency. Computational stability is similar to that of $H211b$.

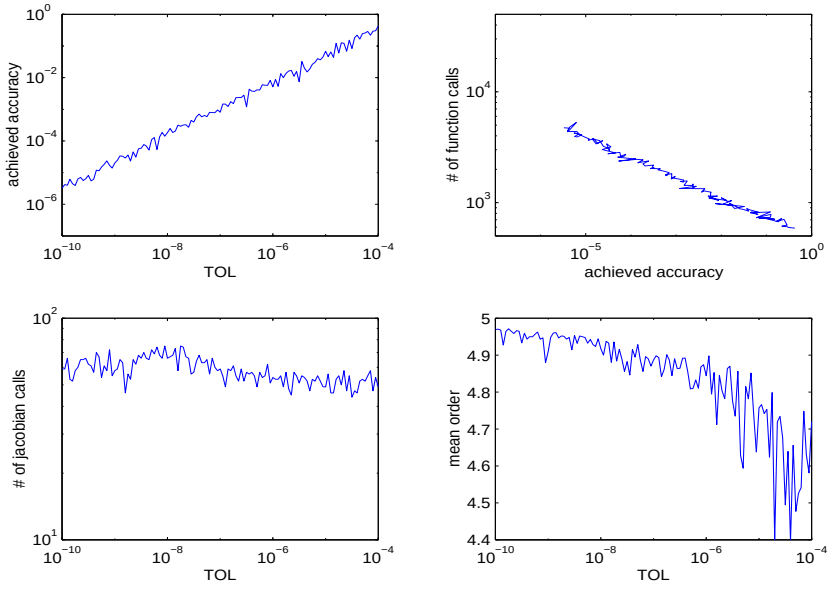

Figure 12: The original DASPK3.0 code is applied to the Pleiades test problem. Computational stability is quite good, but for loose tolerances, the order control shows considerable instability (bottom right).

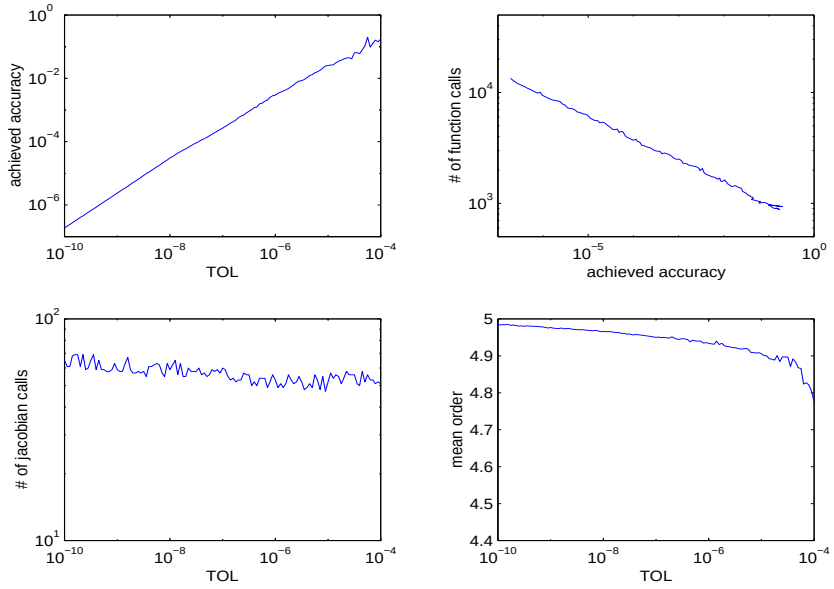

Figure 13: Modified code applied to the Pleiades test, using $H211b$. Computational stability is very high and order control smooth. The modified code uses about 20 – 25% more function calls for a given accuracy (top right).

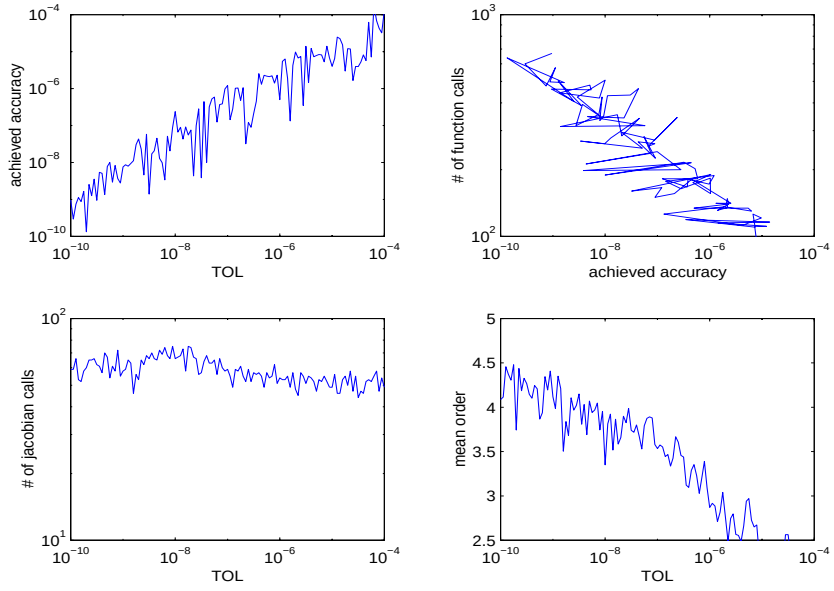

Figure 14: Original code applied to the pollution test problem. Computational stability is low, work–precision is highly irregular and the order control also displays significant instability.

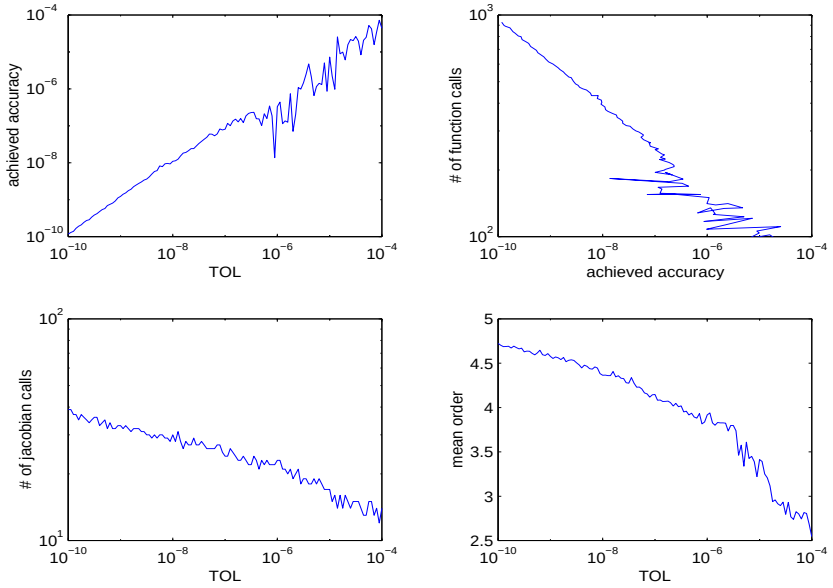

Figure 15: Modified code applied to the pollution test problem. Computational stability is high for sharp tolerances; for loose tolerances the integration requires rather few steps and some computational instability remains, also in the work-precision diagram. Order control is stable, and there is a significant reduction of Jacobians.

[4] K. Gustafsson. Control theoretic techniques for stepsize selection in explicit Runge–Kutta methods. *ACM TOMS* 17:533–554, 1991.

[5] K. Gustafsson. Control theoretic techniques for stepsize selection in implicit Runge–Kutta methods. *ACM TOMS* 20:496–517, 1994.

[6] K. Gustafsson and G. Söderlind. Control strategies for the iterative solution of nonlinear equations in ODE solvers. *SIAM J. Sci. Comp.* 18:23–40, 1997.

[7] E. Hairer, S.P. Nørsett and G. Wanner. Solving Ordinary Differential Equations I: Nonstiff Problems. Springer-Verlag, 2nd revised edition, Berlin 1993.

[8] E. Hairer and G. Wanner. Solving Ordinary Differential Equations II: Stiff and Differential-algebraic Problems. Springer-Verlag, 2nd revised edition, Berlin 1996.

[9] R.D. Skeel. Construction of variable stepsize multistep formulas, *Math. Comp.* 47:503–510, S45–S52, 1986.

[10] J.J.B. de Swart. Test set for IVP solvers. CWI, Amsterdam 1995. `http://www.cwi.nl/cwi/projects/IVPtestset.html`

[11] G. Söderlind. Automatic control and adaptive time–stepping. *Numerical Algorithms* 31:281–310, 2002.

[12] G. Söderlind. Digital filters in adaptive time–stepping. *ACM Trans. Math. Software*, 29:1–26, 2003.

[13] Ch. W. Ueberhuber. Numerical Computation 1. Methods, Software, and Analysis. Springer, Berlin 1997.

[14] K.J. Åström and B. Wittenmark. Computer-controlled systems. Theory and design. (2nd ed.) Prentice–Hall, Englewood Cliffs 1990.