A BVP Solver that Controls Residual and Error

J.A. Kierzenka
The MathWorks, Inc.
3 Apple Hill
Natick, MA, 01760, U.S.A.
Jacek.Kierzenka@mathworks.com

L.F. Shampine
Mathematics Department
Southern Methodist University
Dallas, TX 75275, U.S.A.
lshampin@smu.edu

June 25, 2007

1 Introduction

We consider the solution of a first order system of ordinary differential equations

$$y'(x) = f(x, y(x), p), \quad a \le x \le b \tag{1}$$

subject to boundary conditions

$$g(y(a), y(b), p) = 0 \tag{2}$$

that may be nonlinear and may not be separated. Here p is an optional vector of unknown parameters. For methods that approximate the solution of a boundary value problem (BVP) with a piecewise-smooth function $S(x)$, a natural measure of error is the residual in the differential equations

$$r(x) = S'(x) - f(x, S(x), p) \tag{3}$$

and a corresponding expression for the residual in the boundary conditions. This can be interpreted as saying that $S(x)$ is the exact solution of the problem (1),(2) with its data $f(x, y, p)$ and $g(y(a), y(b), p)$ perturbed by residuals, e.g.,

$$S'(x) = f(x, S(x), p) + r(x)$$

and similarly for the boundary conditions. From the viewpoint of backward error analysis that is so important to linear algebra, a "good" solution is one

for which these perturbations are "small". Virtues of the approach are that it makes minimal demands on the smoothness of the problem data and the size of the residual is meaningful even when the mesh is too crude for limit arguments to be valid. This is the approach to error control taken by `bvp4c`, the BVP solver of MATLAB [12]. This solver has proved to be effective, but it is not all that we might want. For one thing, users are accustomed to controlling the true error $e(x) = S(x) - y(x)$. This is rather different from controlling the residual. For instance, if the BVP is well-conditioned, a small residual implies a small true error, but this need not be true if the BVP is ill-conditioned [3]. A practical distinction is that the methods we consider approximate $r(x)$ to a lower order than $e(x)$. Furthermore, the solution $y(x)$ is the natural weight when controlling a norm of $e(x)$ and $y'(x)$ is the natural weight when controlling a norm of $r(x)$.

In this paper we describe `bvp5c`, a new BVP solver that controls a residual *and* the true error. We showed [15] that something like this was possible with the `ddesd` program of MATLAB that solves delay differential equations (DDEs) with time- and state-dependent delays. For robustness the new solver is based on control of a residual. The residual is scaled so that it has the same order of convergence as the true error. For a large class of methods we prove that if this scaled residual is less than a given tolerance, then asymptotically the true error is also less than the tolerance. For the four-point Lobatto method implemented in `bvp5c`, we can use results of Russell and Christiansen [13] to determine how much less. In brief, control of the scaled residual in `bvp5c` is meaningful even at tolerances and mesh spacings so crude that asymptotic results are scarcely applicable. If asymptotic results are applicable, the true error is smaller than the error tolerance by a factor of about 0.14. Others have found Lobatto formulas to be attractive for solving BVPs. Indeed, Cash and Wright changed the MIRK formulas used in the well–known solver ACDC [1] to Lobatto formulas because of their superior stability properties [5].

Our implementation of the four-point formula in `bvp5c` resembles our implementation of the three-point formula in `bvp4c`, but differs in two ways that require discussion: The four-point formula is evaluated directly because the analytical condensation used in `bvp4c` to evaluate the three-point formula is not possible for this formula. The important capability of unknown parameters is realized with an entirely different technique. Several auxiliary functions for `bvp4c` had to be modified for `bvp5c` and, of course, a function for evaluating its continuous extension written. We also take this opportunity to document the `bvpxtend` function which is used for continuation in the length of the interval. The paper [14] explains how `bvp4c` was extended to solve a class of singular BVPs of the form

$$y'(x) = \frac{S}{x} + f(x, y(x), p), \quad 0 \le x \le b$$

with constant matrix S and $b > 0$. We have similarly extended `bvp5c`, but for simplicity we discuss here only the non-singular form (1).

From a user's point of view, `bvp5c` differs from `bvp4c` in only two ways, but both are important. One is that error tolerances are specified in the same

manner, but they do not mean the same. Because of this it is possible to compare the performance of the two solvers only in rough terms. The other is that `bvp4c` can solve directly multipoint BVPs and `bvp5c` is not (yet) able to do this. Numerical examples in §5.2 and §6 show that `bvp5c` is an effective BVP solver that controls well both the scaled residual and the true error.

2 Formula and Continuous Extension

The four-point Lobatto IIIA formula implemented in `bvp5c` is stated as the implicit Runge–Kutta formula (3.3.21) in [8] and Table 5.8 in [9]. The method can be viewed as collocation on a subinterval $[x_i, x_i+h_i]$ at points $t_{i,j} = x_i+c_j\,h_i$ with $c_1 = 0$, $c_2 = (5-\sqrt{5})/10$, $c_3 = (5+\sqrt{5})/10$, $c_4 = 1$. To define a continuous extension $q(x)$ of the Runge–Kutta formula on $[x_i, x_{i+1}]$, we write Y_j for the approximation to the solution at $t_{i,j}$ and $K_j = f(t_{i,j}, Y_j)$. The quartic polynomial $q(x)$ is defined by requiring that $q(x_i) = Y_1$ and $q'(t_{i,j}) = K_j$ for $j = 1, 2, 3, 4$. (Definition of $q(x)$ as a continuous extension does *not* include $q_i(x_{i+1}) = Y_4$.) Equivalently, $q(x)$ can be defined by collocation, meaning that it is a quartic polynomial that satisfies (1) at the $t_{i,j}$. That is, $q(x)$ satisfies the ODEs at four specific points in $[x_i, x_i + h_i]$.

Standard convergence results [3] for collocation apply to the function $s(x)$ composed of $q(x)$ on each $[x_i, x_{i+1}]$, but the continuous extension $S(x)$ that we use is a little different. In practice we do not evaluate the implicit formulas exactly, meaning that $q(x)$ defined as stated does not satisfy the collocation equations exactly at all the $x_i + c_j\,h_i$ in $[x_i, x_{i+1}]$, though the interpolation condition at x_i makes it satisfy the collocation equation at that end point by definition. A consequence of this is that $q'(x)$ is continuous from one subinterval to the next, but $q(x)$ generally is not. To avoid this disagreeable behavior, we define the continuous extension $S(x)$ used in `bvp5c` so that it interpolates value and slope at both ends of the subinterval and the value y_{mid} of $q(x)$ at the midpoint. It is straightforward to derive

$$y_{mid} = Y_1 + h\left[\frac{17}{192}\,K_1 + \frac{40+15\sqrt{5}}{192}\,K_2 + \frac{40-15\sqrt{5}}{192}\,K_3 - \frac{1}{192}\,K_4\right]$$

as well as the representation of $S(x)$ in terms of y_{mid} and the values and slopes at the end points. If the algebraic equations are satisfied exactly, the two continuous extensions are the same, but even when they are not satisfied exactly, $S(x) \in C^1[a, b]$ and collocates exactly at the mesh points. Naturally the numerical evaluation of the implicit formulas is exact as the tolerance goes to zero, so perturbing $s(x)$ in this way does not alter convergence results.

3 Error Estimation and Control

In this section we begin with some general observations about control of a scaled residual and control of the true error. This is followed by the development of

a more precise relationship between the scaled residual and the error for the Lobatto four-point scheme implemented in `bvp5c`. We consider methods that approximate the solution $y(x)$ of (1),(2) on a mesh $a = x_1 < x_2 < \ldots < x_{N+1} = b$ by a function $S(x)$ that is smooth on each subinterval $[x_i, x_{i+1}]$. The mesh spacing $h_i = x_{i+1} - x_i$ and convergence is considered as $h = \max_i h_i$ tends to zero. We assume that $e(x) = S(x) - y(x)$ is $O(h^p)$ and $S'(x) - y'(x)$ is $O(h^{p-1})$. All this is shown to hold for the popular collocation methods in the text [3]. It is further shown that implicit Runge–Kutta methods can be viewed as collocation methods when they are supplemented with a natural continuous extension and that many methods of this kind are superconvergent at mesh points, meaning that if the method is of order p, a norm of the error at mesh points is at least $O(h^{p+1})$.

3.1 Scaled Residual

In solving DDEs with time- and state-dependent delays [15], we found it valuable to control not the residual, but the residual scaled by the mesh spacing. The relationship between the two controls is reminiscent of the relationship between error per unit step and error per step when solving IVPs.

Some details about how `bvp4c` controls the size of the residual will be needed. The approximate solution $S(x)$ is smooth on subintervals $[x_i, x_{i+1}]$, so the size of the residual on this subinterval is measured by using a weighted norm $|r(x)|$ at each x and defining

$$\|r(x)\|_i = \max_{x_i \le x \le x_{i+1}} |r(x)|$$

For a given tolerance τ, `bvp4c` aims to produce a numerical solution for which $\max_i \|r(x)\|_i \le \tau$. In this it constructs a mesh that approximately equidistributes the residual.

It might seem that controlling the scaled residual, $h_i\|r(x)\|_i$, is obviously less demanding than controlling the residual, $\|r(x)\|_i$, because of the small factor h_i, but this neglects the role of the norm. When controlling the scaled residual as in `bvp5c` it is natural to use $y(x)$ as weight in the norm, but when controlling the residual itself as in `bvp4c`, it is natural to use $y'(x)$ as weight. Obviously the norms and therefore the controls will differ significantly near points where either $y(x)$ or $y'(x)$ vanish.

Unknown parameters p do not affect the arguments, so we leave them out of expressions for simplicity. Subtracting (1) from (3) leads to

$$r(x) = (S(x) - y(x))' - [f(x, S(x)) - f(x, y(x))] \tag{4}$$

With the usual assumption that f satisfies a Lipschitz condition,

$$|f(x, S(x)) - f(x, y(x))| \le L\,|S(x) - y(x)|$$

hence the last term in (4) is $O(h^p)$. We are assuming that $S'(x) - y'(x)$ is $O(h^{p-1})$, so to leading order the residual is equal to the error in the first derivative. This implies that the scaled residual is $O(h^p)$, the same as the true error.

We prove now that for an important class of methods, there is a more useful connection between scaled residual and true error.

To investigate the relationship between scaled residual and true error, we begin by integrating (4) over a subinterval of $[x_i, x_{i+1}]$,

$$\int_{x_i}^{\beta} r(x)\,dx = e(\beta) - e(x_i) - \int_{x_i}^{\beta} [f(x, S(x)) - f(x, y(x))]\,dx \tag{5}$$

Suppose now that the method of order p is superconvergent at mesh points so that $e(x_i)$ is $O(h^{p+1})$. We define β as a point in $(x_i, x_{i+1}]$ where $|e(\beta)| = \|e(x)\|_i$. As argued earlier, the integrand on the right hand side of (5) is $O(h^p)$ and the interval is of length no bigger than h_i, so

$$\| \int_{x_i}^{\beta} r(x)\,dx\| = \|e(x)\|_i + O(h^{p+1})$$

The inequality

$$\| \int_{x_i}^{\beta} r(x)\,dx\| \le h_i \|r(x)\|_i$$

then tells us that to leading order, the size of the scaled residual is an upper bound on the size of the true error. If we require that $\max_i h_i\|r(x)\|_i \le \tau$ for a tolerance τ, then to leading order we have $\max_i \|e(x)\|_i \le \tau$. For methods of the kind considered, this is a strong argument for controlling the size of the scaled residual rather than the residual itself. The four-point Lobatto scheme is of this kind, but we shall see that there is a more intimate relationship in this particular case. The three-point Lobatto scheme of `bvp4c` is not of this kind because it is not superconvergent at mesh points.

3.2 Error Estimates

Except for obvious references to equations in this paper, all the definitions, equations, and page numbers referenced in this subsection are those of Russell and Christiansen [13]. For the four-point Lobatto IIIA formula, the quantities $m = 1$ and $k = 5$, so according to (2.20) in [13], the residual satisfies

$$r(x) = \frac{y^{(5)}(x_{i+1/2})}{4!}\,p(x) + O(h^5) \tag{6}$$

for x in $[x_i, x_{i+1}]$. Here

$$p(x) = \prod_{j=1}^{4}(x - t_{i,j}) \tag{7}$$

and $t_{i,j}$ is defined in §2. For later use we note that with $x = x_i + s\,h_i$

$$p(x) = h_i^4 \prod_{j=1}^{4}(s - (t_{i,j} - x_i)/h_i) = h_i^4 \prod_{j=1}^{4}(s - c_j)$$

so that the extreme values of $p(s)$ on $[0,1]$ are multiplied by h_i^4 when the interval is $[x_i, x_{i+1}]$. The polynomial $p(s)$ vanishes at the collocation points c_j and it is straightforward to determine that the extreme values occur at $1/2 - \sqrt{15}/10$, $1/2$, $1/2 + \sqrt{15}/10$ where the values themselves are $-1/100$, $+1/80$, $-1/100$. Notice that the maximum of these extreme values occurs at the midpoint of the subinterval.

For this formula $d = 6$, so according to (2.11), $|e(x_i)| = O(h^6)$, which is a superconvergence result since a uniform bound on the error is only $O(h^5)$. Indeed, because $d > k$, the error expression (2.12) is valid,

$$\|e(x)\|_i = C\, h_i^5\, \|y^{(5)}\|_i + O(h^6) \tag{8}$$

The constant C here is given on p. 77 for $m = 1$ as

$$C = \frac{1}{2^5\, 4!} \max_{-1 \le x \le 1} \left| \int_{-1}^{x} \prod_{j=1}^{4} (\tau - \tau_j)\, d\tau \right| \tag{9}$$

for collocation points τ_j in $[-1, 1]$. This is equivalent to

$$C = \frac{1}{4!} \max_{0 \le \xi \le 1} \left| \int_0^{\xi} p(s)\, ds \right|$$

The extreme values of $Q(\xi) = \int_0^{\xi} p(s)\, ds$ occur where $Q'(\xi) = p(\xi) = 0$, namely at c_1, c_2, c_3, c_4. It is straightforward to determine that Q vanishes at c_1 and c_4 and has the same magnitude at c_2 and c_3, namely $\sqrt{5}/1250$.

In the manner of Russell and Christiansen, we see from (6) that for any ξ in $[x_i, x_{i+1}]$ different from the nodes $t_{i,j}$,

$$|r(\xi)| = \frac{1}{4!}\, \|y^{(5)}(x)\|_i\, |p(\xi)| + O(h^5)$$

We can solve this expression for $\|y^{(5)}(x)\|_i$ and substitute the result into (8) to get

$$\|e(x)\|_i = C\, h_i^5\, |r(\xi)| \frac{4!}{|p(\xi)|} + O(h^6)$$

If we write $p(\xi)$ in terms of a variable scaled to $[0,1]$ and use the value of C just derived, the equation becomes

$$\|e(x)\|_i = \frac{\sqrt{5}}{1250\, |p((\xi - x_i)/h_i)|}\, h_i\, |r(\xi)| + O(h^6)$$

A natural choice for ξ is where the residual has its maximum magnitude, namely the midpoint. This leads to

$$\|e(x)\|_i = \frac{8\sqrt{5}}{125}\; h_i\, |r(x_{i+1/2})| + O(h^6) = \frac{8\sqrt{5}}{125}\; h_i\, \|r(x)\|_i + O(h^6) \tag{10}$$

This is the strong connection between the size of the true error and the size of the scaled residual promised earlier. The constant here is about 0.1431, so if we require that $\max_i h_i \|r(x)\|_i \leq \tau$, the true error will be less than τ by a margin that should be sufficient to account for the possibility that limit results are poorly applicable.

It is remarkable that a single sample of the residual provides an asymptotically correct estimate of the size of the true error, but the result must be used with some caution in practice. Though formally correct, if the derivative in (6) is exceptionally small at the midpoint, the term $O(h^5)$ might dominate and the relationship between the residual and derivative that we use to estimate the true error might not be valid. Likewise, if the term involving the derivative in (8) is not dominant, the estimate might not be valid. We prefer to be cautious in assessing the error, so `bvp5c` controls the scaled residual rather than the true error. Of course we expect that this will usually control the true error as well. Indeed, equation (10) states that asymptotically the true error is substantially less than the scaled residual. We are also cautious in that we sample the residual at all three of the points where it has an extremum in the limit so as to obtain a more reliable approximation to its size. Our caution has been reinforced by examples we have encountered with residuals that were exceptionally small at the midpoint for a good many subintervals, even at stringent tolerances.

Russell and Christiansen point out in connection with (2.23) that equidistributing what amounts to the scaled residual will asymptotically pick a mesh that is optimal in a certain sense. They remark that methods of this kind provide both a mesh selection scheme and an error estimate. Although we focus on control of the size of the scaled residual rather than the true error, it is gratifying that it results in a good mesh.

4 Unknown Parameters

Most BVP solvers do not provide for unknown parameters nor non-separated boundary conditions because they can then exploit software for solving linear systems with banded or almost block diagonal (ABD) matrices. Because `bvp4c` uses a linear equation solver for general sparse matrices, it was straightforward to provide for these possibilities. Other solvers expect users to introduce new variables so as transform a problem with unknown parameters into a bigger problem without unknown parameters [4]. An additional ODE is needed for each unknown parameter, but generally there are few unknown parameters, so this increase in the size of the system is usually harmless. This is at best an annoyance for users, so to provide a user interface for unknown parameters similar to that of `bvp4c` and still take advantage of software for ABD systems, the BVP_SOLVER [17] program transforms the problem inside the solver.

We insisted that the user interface of `bvp5c` for unknown parameters be identical to that of `bvp4c`, but it was not clear how we ought to implement this capability. Certainly we could use the general sparse linear equation solver as in `bvp4c`, but it is based on Gaussian elimination with partial pivoting. Wright

[19] has given a class of BVPs that show this can be unstable for matrices that arise when using collocation methods. Wright states that instability is rarely encountered in practice, but codes do not commonly monitor stability. As it happens, `bvp4c` *does* monitor the condition of the matrix. An error message from `bvp4c` about a singular Jacobian is not unusual, but we have always taken this to mean that the guesses for the mesh and solution do not represent adequately the behavior of the solution. It is possible that instability of partial pivoting is responsible for such a message, but we do not know whether this has ever happened. It is important to appreciate that the difficulty pointed out by Wright arises only when the matrix has the form of a band plus a border due to non-separated boundary conditions. Non-separated boundary conditions are not common in our experience, but we cannot be complacent because unknown parameters lead to the same kind of bordering. Wright speculates that typical matrices are small enough that the growth factor is not unbearable. That is certainly relevant to BVPs solved to the modest accuracies typical of MATLAB. In experiment Wright found that scaling the portion of the matrix corresponding to the boundary conditions was quite beneficial; `bvp4c` does explicit row scaling. The matter is murky, but we recognize that the transformation approach to unknown parameters leads to matrices that do not have the dangerous form. Also, no method is universally successful in solving BVPs, so it is useful to have an alternative to `bvp4c` that takes a different approach. It was found in coding BVP_SOLVER [17] that a convenient way to implement the transformation approach was to define internal functions that substitute for the ones provided by the user. This technique is already used in `bvp4c` to deal with vectorization, so the transformation approach could be implemented in a natural way in `bvp5c`.

5 The Programs

In this section we discuss a few software issues. These include the implementation of basic algorithms in the solver and fitting the solver into the MATLAB ODE Suite.

5.1 `bvpinit`

A user must call `bvp4c` with a structure containing guesses for the mesh and the solution that reveal the behavior of the solution. An auxiliary function, `bvpinit`, makes it easy to form a guess structure. Obviously we have to modify this function to form guesses for `bvp5c`, but is it possible to do this in the same way for both solvers? The implicit Runge–Kutta formula of `bvp4c` is evaluated using analytical condensation, so it is not necessary to provide a guess for the solution at the one interior collocation point, but for `bvp5c` we must somehow provide guesses at two interior points. As we explain in §5.2, the `bvpinit` function has evolved since its introduction [10]. At present there are two ways to specify a guess. If it is reasonable to guess that the solution is a constant vector at all mesh points, this vector can be supplied. Of course, it is easy to

deal with this case for `bvp5c` because we can use this vector as guess at the interior collocation points. The other way is to supply a function that `bvpinit` evaluates to obtain guesses at the mesh points. If we were to ask users which solver they have in mind, we could use this function to form the additional values required by `bvp5c`. However, to make the user interface as simple as possible, we instead added the function to the guess structure. If the solver is `bvp5c`, it tests on entry whether this function handle is present in the guess structure and if it is, the guess is augmented at this time to include the interior collocation points by evaluating the function at those points. By proceeding in this way, the user sees nothing different about `bvpinit` modified to deal with `bvp5c`.

5.2 `bvpxtend`

Difficult BVPs are frequently solved by continuation. One form of continuation is to solve BVPs on a sequence of intervals that approach the one of interest. This form is especially common when solving problems on infinite intervals. When doing continuation in the interval, it is necessary somehow to use the solution on one interval to form a guess for the solution on a bigger interval. The original version of `bvpinit` had an option for this. A guess for the solution outside the interval was formed by extrapolation and more specifically in the case of `bvp4c`, cubic extrapolation. It is well-known that the greater the distance from the data and the higher the degree of the polynomial, the more likely that extrapolation will provide an unsatisfactory approximation. Extrapolation of the solution is even less appealing with the quartics of `bvp5c`. We simplified `bvpinit` by moving the option of extending a solution structure to a new function `bvpxtend`. By restricting the extension to only one end point at a time, we were able to simplify the user interface and offer new possibilities. The new function has an option for the order of extrapolation. It has three values, 'constant', 'linear', and 'solution', with 'constant' extrapolation as default. An important new possibility is to provide a guess for the solution at the end point rather than extrapolating. When some kind of analytical approximation to the solution at the end point is available, this is likely to be the most satisfactory choice. It is to be appreciated that analytical approximations may well depend on unknown parameters, but approximations to these parameters will be available in the solution structure. In our experience the lower orders of extrapolation have proved more robust than the scheme originally used in `bvpinit` and supplying a good analytical approximation has been even better.

Example 3.5.5 of [16] solves the Fisher BVP

$$U'' + cU' + U(1-U) = 0, \qquad U(-\infty) = 1, U(+\infty) = 0$$

with a program called `ch3ex5`. It solves a sequence of BVPs on intervals of increasing size to gain some confidence that an acceptable approximation to the solution on $(-\infty, +\infty)$ is achieved. The various problems are solved independently with `bvp4c` using values from an asymptotic approximation due

to Murray as initial guess for each BVP rather than extrapolating a previous solution. That was done because if we try to extend the interval too rapidly using the cubic extrapolation of the old version of `bvpinit`, the computation fails. The new options of linear and constant extrapolation allow a much faster extension of the interval and the new possibility of using asymptotic approximations is best of all.

5.3 `deval`

MATLAB uses a single function, `deval`, to evaluate the continuous extensions of all its differential equation solvers. The solution structure produced by `bvp5c` includes a field that identifies the solver as well as fields that hold the solution and its first derivative at mesh points and the solution at midpoints. When `deval` is called with a solution structure computed by `bvp5c`, it first identifies the solver and then calls an auxiliary function `ntrp4h` that uses the data stored in the solution structure to evaluate the continuous extension developed in §2. The new function `ntrp4h` is much like the corresponding function `ntrp3h` used with `bvp4c`. In particular, it is vectorized for the efficient evaluation of the continuous extension at many points that is commonly needed when plotting a solution and it returns the first derivative on demand.

5.4 `bvp5c`

The formula

$$Y_3 = Y_1 + h_i \left[\frac{11+\sqrt{5}}{120} K_1 + \frac{25+13\sqrt{5}}{120} K_2 + \frac{25+\sqrt{5}}{120} K_3 + \frac{1+\sqrt{5}}{120} K_4 \right]$$

illustrates the implicit Runge-Kutta formulas that are to be evaluated on each subinterval $[x_i, x_{i+1}]$. The form is natural when we solve for the Y_j, but other implementations of implicit Runge-Kutta methods solve equivalent formulas for the K_j. We evaluate the formulas in `bvp5c` in a way that is quite similar to that of `bvp4c` and in particular, we solve for the Y_j. A notable difference is that analytical condensation is not possible for the four-point Lobatto scheme, so we work with the form stated. Having computed iterates $Y_j^{[m]}$ for all the subintervals, we form $K_j^{[m]} = f(x_i + c_j\, h_i, Y_j^{[m]})$, linearize the formulas using an approximate Jacobian matrix, and compute corrections to get $Y_j^{[m+1]}$. As in `bvp4c` the size of the corrections is used to judge whether the current iterate satisfies the algebraic equations sufficiently well. If the equations were solved exactly, the continuous extension $s(x)$ would collocate at four-points in each subinterval. As explained in §2, this will not be the case in practice because the algebraic equations are not satisfied exactly. `bvp5c` controls the size of the residual of $S(x)$, so it is natural to require that it be relatively small at the collocation points. To monitor this, we invert the relationship between the Y_j

and K_j. For instance,

$$K_3 = \frac{\sqrt{5}}{5}K_1 + \frac{17\sqrt{5}-5}{10h_i}Y_1 - \frac{5+3\sqrt{5}}{2h_i}Y_2 + \frac{5-\sqrt{5}}{2h_i}Y_3 + \frac{5+3\sqrt{5}}{10h_i}Y_4$$

Using these expressions and the $Y_j^{[m]}$ on the right hand side, we form values $K_j^{[m+1]}$ that amount to derivatives of a continuous extension at the collocation points. The differences between these values and the $K_j^{[m]} = f(x_i + c_j\, h_i, Y_j^{[m]})$ are residuals at the collocation points. `bvp5c` supplements the test in `bvp4c` on the change in the iterate itself with this test on the residual at collocation points. Kierzenka and Shampine [10] point out that using the residual at the one interior collocation point in the error control of `bvp4c` provides an indirect control of the collocation error at that point. The residual of $S(x)$ vanishes at both ends of the subinterval by construction and we insist that the residual at the interior collocation points be no more than one tenth the maximum allowed. With this extra test it is natural to assume that the residual will be relatively small at the collocation points and have the qualitative behavior expected from equation (6). This is usually the case, but not always. Indeed, we have seen examples for which the maximum value of the residual occurred at a collocation point because the residual was uniformly small in the subinterval.

After evaluating the implicit Runge-Kutta formulas sufficiently well, we estimate the norm of the scaled residual by evaluating it at the midpoint. Recall that asymptotically the residual has its maximum magnitude at this point. If the scaled residual is too big, we refine the mesh appropriately and try again. If it is acceptable, we supplement the estimate by evaluating the residual at the other two points where it has a local extremum asymptotically and test again. Even if the expected asymptotic behavior is not evident, we are controlling the size of the scaled residual of $S(x)$ using seven samples in each subinterval. If the asymptotic behavior is evident, the analysis of §3.2 shows that the size of the true error is a fraction of the size of the scaled residual. Proceeding in this way we secure the virtues of residual control when asymptotic results are of dubious applicability and control of the true error when they are valid. Together with multiple assessments of error in both situations, the error control of `bvp5c` appears to be exceptionally robust.

Dynamic storage allocation and array operations make certain economies practical. One is in the reuse of function evaluations when the mesh is refined. At the time a procedure is called to refine the mesh we have values for the derivatives of the approximate solution at both mesh and interior collocation points. These values remain valid for subintervals that are not refined and the values at the end points remain valid for the ones that are. In the subintervals that are refined, the continuous extension is used to obtain approximations to the solution and the differential equations are evaluated to get approximations to the derivatives. Reusing all the function evaluations corresponding to subintervals that are not refined provides a useful reduction in the total number of function evaluations.

There are two ways that knowledge of the sparsity pattern of the Jacobian of a system of ODEs can be exploited. One is in the efficient approximation of the Jacobian by finite differences and the other is to reduce the costs of solving linear systems. This is an important option in the IVP solvers of MATLAB, but `bvp4c` does not provide for sparse Jacobians because it is expected that the number of ODEs will ordinarily be small enough that the benefit is not worth complicating the user interface. All quality BVP solvers exploit somehow the structure of the linear systems that arise in solving the algebraic equations for the discrete solution. Our solvers are exceptional in that they treat these systems as general sparse matrices. A distinction is seen with respect to the sparsity of the Jacobian itself: We compute the Jacobian as a full matrix, but when it is used in assembling the system matrix, the PSE automatically recognizes any zero entries and takes them into account in forming the sparse system matrix. We note that our scheme for approximating Jacobians at interior collocation points preserves the sparsity pattern of the Jacobian. `bvp5c` uses an unsymmetric multifrontal method [7] for solving sparse linear systems. It is significantly more efficient in both time and memory than the method originally used by `bvp4c`.

6 Numerical Experiments

Keeping in mind that error tolerances do not have the same meaning and that `bvp4c` has capabilities that `bvp5c` does not, it is easy to test `bvp5c` thoroughly with programs readily available that use `bvp4c`—just change the name of the solver (and use auxiliary functions modified so that they apply to the new solver). MATLAB itself includes a variety of examples. The documentation for `bvp4c` is linked to a tutorial [18] that explains how to solve BVPs that exhibit a wide range of difficulties and provides programs for the examples. Chapter 3 of [16] does the same thing. `bvp5c` performed very well on these examples, with just two exceptions, namely `ex4bvp` and `ex5bvp`. After changing the guess or continuation scheme, `bvp5c` was able to solve these problems. In connection with extending `bvp4c` to a class of singular BVPs [14], we assembled a set of test problems from the literature. `bvp5c` solved all these problems without difficulty..

Difficult problems involving a singular coefficient at an end point or an end point at infinity are typically solved by approximating the solution analytically near the end point. Often one or more of the coefficients in this approximation must be found as part of solving the BVP. A program like COLSYS [2] that is based on Gaussian formulas can be applied to problems with a singular coefficient at a finite end point because the solver does not evaluate the equation there. The Lobatto formulas of our solvers preclude this approach and anyway, we believe that it is imprudent to attempt the solution of BVPs with singularities without first sorting out the behavior of the solution near the singular points. Cash and Silva [6] make this point forcefully with the equation

$$y'' - \frac{y'}{x^2} + 100y = 1000x - \frac{10}{x^2} + \frac{10\cos(10x)}{x^2} \tag{11}$$

to be solved subject to boundary conditions

$$y'(0) = 0, \qquad y(1) = 10 - \sin(10) \tag{12}$$

The analytical solution $y(x) = 10x - \sin(10x)$ is available to compare with numerical solutions. Cash and Silva report that "...at all tolerances COLSYS accepted a solution which had no resemblance to the true solution." Assuming a Taylor series expansion for a solution that satisfies the boundary condition $y'(0) = 0$ and equating coefficients of like powers in the differential equation leads to

$$y(x) = P + \left(\frac{500 + 100P}{3}\right) x^3 + 50Px^4 + \left(120P - \frac{2500}{3}\right) x^5 + \ldots$$

In this series the parameter $P = y(0)$ is unknown. The idea is to use this analytical approximation on an interval $[0, d]$ and solve the differential equation numerically on $[d, 1]$. The solution on $[d, 1]$ is subject to the given boundary condition at $x = 1$ and continuity of the analytical approximation to the solution and its first derivative with the numerical solution and its first derivative at $x = d$. We need two continuity conditions at $x = d$ because we need three boundary conditions to determine P along with $y(x)$ and $y'(x)$. We solved the BVP in this way with `bvp5c` using default tolerances. In this we used the very poor guess of 5 for P and a constant guess for the solution consisting of the terms through x^5 for the series and its derivative evaluated at d and the guess for P. The initial mesh was taken to be 20 equally spaced points. As seen in Figure 1, the computation succeeds with $d = 0.01$ and provides a good solution. The unknown parameter was computed to be about 0.0048. Keeping in mind that the error controls are quite different, we found that if we changed only the name of the solver in this program, the computation was successful with `bvp4c`, but the result was unsatisfactory: The numerical solution had little resemblance to the true solution and the unknown parameter had the unacceptable value 1.20598. When solving a BVP with a singular coefficient, it is often not clear how to choose a suitable value for d. Here the value of 0.1 is large enough that the series is not very accurate at d. The resulting numerical solution with `bvp5c` has the correct shape, but it does not have even graphical accuracy. On the other hand, with $d = 0.001$, the solver returns an error message to the effect that it has encountered a singular Jacobian—an example of the condition monitor mentioned in §4. This is not surprising in light of how big coefficients become for d this small and the cancellation that takes place in evaluating the differential equation.

To test the effectiveness of the error control we solved the three "badly behaved" BVPs used by Russell and Christiansen [13] to study adaptive mesh selection algorithms. They are

A. $y''(x) + 2\gamma y'(x) + 2y(x) = 0, \quad y(0) = 1, \quad y(1) = e^{-\gamma}, \quad \gamma = 150;$

B. $\epsilon y''(x) - (2 - x^2)y(x) = f(x), \quad y'(0) = 1, \quad y(1) = \beta, \quad \epsilon = 10^{-4};$

C. $y''(x) + \dfrac{2}{x}y'(x) + \dfrac{1}{x^4}y(x) = 0, \quad y\left(\dfrac{1}{3\pi}\right) = 0, \quad y(1) = \sin(1).$

The solution of Example A is $e^{-\gamma x^2}$. Example B defines $f(x)$ and β so that the solution is

$$y(x) = \frac{1}{2-x^2} - e^{-(1-x)/\sqrt{\epsilon}} - e^{-(1+x)/\sqrt{\epsilon}}$$

The solution of Example C is $\sin(1/x)$. Assessment of the error is complicated by the two error tolerances of `bvp5c`. The solver has a scalar relative error tolerance `RelTol` and an absolute error tolerance `AbsTol` that can be a scalar or vector. For the tests we took `RelTol` and `AbsTol` to be the same and solved the examples for a range of this single tolerance τ. At the largest tolerance of $\tau = 10^{-1}$, we used a very crude guess consisting of 10 equally spaced points for the mesh and the constant vector `[1; 1]` for the solution. Subsequently the solution at tolerance τ was used as guess at tolerance $\tau/10$. The equations are linear, so naturally we provided the solver with analytical Jacobians. Using the analytical solution we computed the norm of the scaled residual and the norm of the true error by sampling at 20 equally spaced points in each subinterval of the mesh. The tables show the ratio of the actual error to the tolerance (the error overrun) for both kinds of error, the scaled residual and true error. `bvp5c` attempts to produce a solution for which the norm of the scaled residual is no bigger than the tolerance, hence a ratio no bigger than 1. If all goes well, the norm of the true error will also be no bigger than 1. As seen in Table 1, the solver did very well with Example A. When the tolerance is small, we expect that the norm of the true error will usually be smaller than the norm of the scaled residual by a factor of about 0.14. That appears to be the case in this table. The mesh used in solving Example B for $\tau = 10^{-1}$ was chosen so conservatively that the same mesh was used for $\tau = 10^{-2}$, which is why the error overruns are so small at the crudest tolerance in Table 2. We note that the true error is controlled very well at all tolerances in this table, but it is not as much smaller than the scaled residual as we expect from asymptotic results. Results for Example C are found in Table 3. The solver is relatively conservative at the crudest tolerances, but it is controlling the errors well at all tolerances and the true error behaves about as expected from limit arguments. In summary, `bvp5c` does very well at controlling the norm of the scaled residual to be smaller than the specified tolerance. For reasons of efficiency the scaled residual is generally close to the tolerance, but it may be rather smaller at crude tolerances where it is appropriate to be cautious. The solver also controls the true error very well. It is rather smaller than the tolerance in all cases by a factor that is typically comparable to the factor of 0.14 expected from asymptotic arguments.

As stated earlier, it is not possible to compare `bvp4c` and `bvp5c` in any precise way because the error controls are so different. Nevertheless, the Russell–Christiansen examples give us some feeling for the matter. When the programs for the computations just described were changed to use `bvp4c`, it was found that in nearly all cases, control of the size of the residual resulted in true errors that were not greatly larger than the tolerance. It would be natural to use the number of mesh points, *nmpt*, as a measure of difficulty for a solver, but clearly we ought to take into account the number of interior collocation points. Also, we should recognize that `bvp4c` uses analytical condensation and `bvp5c`

does not. With this in mind, we counted the number of unknowns in the final linear system solved. For *neqn* equations, this number is $neqn \times (3\,nmpt - 2)$ for `bvp5c` and $neqn \times nmpt$ for `bvp4c`. Because of the difference in orders and analytical condensation, we might expect that `bvp4c` solve smaller systems at crude tolerances and the reverse be true at stringent tolerances. Interestingly the crossover occurred at the same τ for all three examples. Specifically, `bvp4c` solved smaller systems for all $\tau \geq 10^{-6}$ and `bvp5c` solved smaller systems for all $\tau \leq 10^{-7}$. The difference was substantial at the more stringent tolerances. At the most stringent tolerance the number of unknowns for `bvp4c` for Examples A, B, C were 3280, 4822, 5690, respectively, and for `bvp5c`, they were 1724, 2210, 3006. It is difficult to obtain consistent measurements of run time in MATLAB. We used the standard approach of measuring the run time on the second invocation of a program. In solving examples A, B, C with `bvp4c`, the run times were 2.41s, 2.89s, 3.31s, respectively, and with `bvp5c` they were 1.48s, 2.80s, 2.31s. A considerable difference in the sizes of the linear systems does not make as big a difference as one might expect because built-in functions such as the linear equation solvers have the speed of compiled computation in an overall program that is interpreted.

With different error definitions used in the solvers, the tolerances do not mean the same in `bvp4c` and `bvp5c`. This makes the runtime-based efficiency measures very subjective, but our experiments across all the examples of [18] suggest that for low-to-medium tolerances, the speed of `bvp5c` is comparable to the speed of `bvp4c`. The results of this section show that for stringent tolerances, the advantage of `bvp5c` is significant.

7 Acknowledgements

We acknowledge with appreciation that the analytical results of this paper were derived with Maple [11], either directly or indirectly through the Symbolic Math Toolbox of MATLAB. I. Gladwell of Southern Methodist University (Dallas) and P. Muir of Saint Mary's University (Halifax) provided very helpful comments on the manuscript.

References

[1] ACDC, `http://www.ma.ic.ac.uk/~jcash/BVP_software/readme.php`

[2] U.M. Ascher, J. Christiansen, and R.D. Russell, Collocation software for boundary value ODEs, ACM Trans. Math. Softw., 7 (1981) 209–222.

[3] U.M. Ascher, R.M.M. Mattheij, and R.D. Russell, Numerical Solution of Boundary Value Problems for Ordinary Differential Equations, SIAM, Philadelphia, 1995.

[4] U.M. Ascher and R.D. Russell, Reformulation of boundary value problems into "standard" form, SIAM Rev., 23 (1981) 238–254.

[5] Z. Bashir-Ali, J.R. Cash, and H.H.M. Silva, Lobatto deferred correction for stiff two-point boundary value problems, Comp. & Maths. with Applics., 36 (1998) 59–70.

[6] J.R. Cash and H.H.M. Silva, On the numerical solution of a class of singular two-point BVPs, J. Comput. Appl. Math., 45 (1993) 91–102.

[7] T.A. Davis, UMFPACK Version 4.6 User Guide, Dept. Computer and Information Science and Engineering, Univ. of Florida, Gainesville, FL, 2002.

[8] K. Dekker and J.G. Verwer, Stability of Runge–Kutta Methods for Stiff Nonlinear Differential Equations, North-Holland, Amsterdam, 1984.

[9] E. Hairer and G. Wanner, Solving Ordinary Differential Equations II Stiff and Differential–Algebraic Problems, 2nd ed., Springer, New York, 1996.

[10] J. Kierzenka and L.F. Shampine, A BVP solver based on residual control and the MATLAB PSE, ACM Trans. Math. Softw., 27 (2001) 299–316.

[11] Maple, Maplesoft, 615 Kumpf Dr., Waterloo, CA, N2V 1K8

[12] MATLAB, The MathWorks, Inc., 3 Apple Hill Dr., Natick, MA 01760.

[13] R.D. Russell and J. Christiansen, Adaptive mesh selection strategies for solving boundary value problems, SIAM J. Numer. Anal., 14 (1978) 59–80.

[14] L.F. Shampine, Singular boundary value problems for ODEs, Appl. Math. and Comp., 138 (2003) 99–112.

[15] L.F. Shampine, Solving ODEs and DDEs with Residual Control, Appl. Numer. Math, 52 (2005) 113–127.

[16] L.F. Shampine, I. Gladwell, and S. Thompson, Solving ODEs with MATLAB, Cambridge Univ. Press, New York, 2003.

[17] L.F. Shampine, P.H. Muir, and H. Xu, A User-friendly Fortran BVP solver, JNAIAM, to appear. The paper and programs are available at `http://cs.stmarys.ca/~muir/BVP_SOLVER_Webpage.shtml`

[18] L.F. Shampine, M.W. Reichelt, and J. Kierzenka, Solving Boundary Value Problems for Ordinary Differential Equations in MATLAB with bvp4c. The tutorial and programs are available at `http://www.mathworks.com/bvp_tutorial`

[19] S.J. Wright, A collection of problems for which Gaussian elimination with partial pivoting is unstable, SIAM J. Sci. Comput., 14 (1993) 231–238.

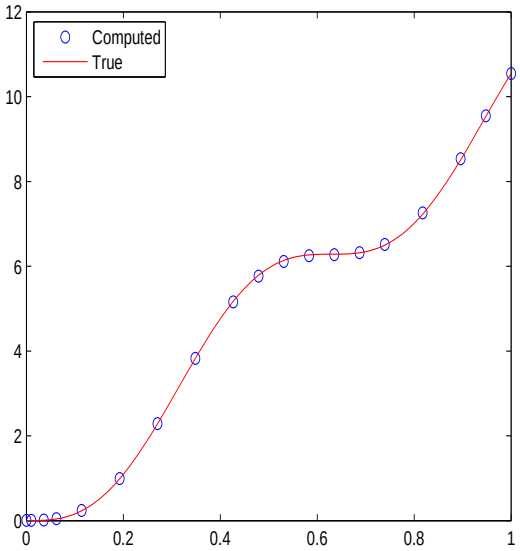

Figure 1: Solution $y(x)$ of (11),(12) with $d = 0.01$.

Table 1: Example A

tolerance τ	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
(scaled residual)/τ	0.69	0.76	0.76	0.72	0.96
(true error)/τ	0.18	0.34	0.23	0.12	0.15
tolerance τ	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
(scaled residual)/τ	0.96	0.90	0.95	0.98	0.98
(true error)/τ	0.14	0.14	0.14	0.15	0.15

Table 2: Example B

tolerance τ	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
(scaled residual)/τ	0.07	0.73	0.97	0.96	0.86
(true error)/τ	0.01	0.10	0.07	0.15	0.13
tolerance τ	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
(scaled residual)/τ	0.95	0.97	0.96	1.01	0.99
(true error)/τ	0.15	0.21	0.27	0.34	0.29

Table 3: Example C

tolerance τ	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
(scaled residual)/τ	0.69	0.76	0.76	0.72	0.96
(true error)/τ	0.18	0.34	0.23	0.12	0.15
tolerance τ	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
(scaled residual)/τ	0.96	0.90	0.95	0.98	0.98
(true error)/τ	0.14	0.13	0.14	0.15	0.15