Available online at www.sciencedirect.com
SCIENCE @ DIRECT®
ELSEVIER
Mathematical and Computer Modelling 40 (2004) 1309–1321
MATHEMATICAL AND COMPUTER MODELLING
www.elsevier.com/locate/mcm

Estimating Conditioning of BVPs for ODEs

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

P. H. MUIR*
Department of Mathematics and Computing Science, Saint Mary's University
Halifax, Nova Scotia B3H 3C3, Canada
Paul.Muir@StMarys.ca

Abstract—An alternative to control of the global error of a numerical solution to a boundary value problem (BVP) for ordinary differential equations (ODEs) is control of its residual, the amount by which it fails to satisfy the ODEs and boundary conditions. Among the methods used by codes that control residuals are collocation, Runge-Kutta methods with continuous extensions, and shooting. Specific codes that concern us are bvp4c of the MATLAB problem solving environment and the FORTRAN code MIRKDC for general scientific computation. The residual of a numerical solution is related to its global error by a conditioning constant. In this paper, we investigate a conditioning constant appropriate for BVP solvers that control residuals and show how to estimate it numerically at a modest cost.

Codes that control residuals can compute pseudosolutions, numerical solutions to BVPs that do not have solutions. That is, a "well-behaved" approximate solution is computed for an ill-posed mathematical problem. The estimate of conditioning is used to improve the robustness of bvp4c and MIRKDC and in particular, help users identify when a pseudosolution may have been computed. © 2005 Elsevier Ltd. All rights reserved.

Keywords—Condition, BVP, ODE, Residual, Pseudosolution.

1. INTRODUCTION

This investigation began with a striking example. To illustrate its error messages, the MATLAB function bvp4c [1] was given a two-point boundary value problem (BVP) for a system of first-order ordinary differential equations (ODEs) *that was known not to have a solution.* The solver promptly computed and returned a plausible numerical solution with no messages of any kind! This solver computes a piecewise-polynomial solution $S(x)$ using collocation. It controls the residual of this solution, the amount by which $S(x)$ fails to satisfy the ODEs and boundary conditions. Evidently, the code has found a numerical solution $S(x)$ that satisfies well the ODEs and boundary conditions of a BVP that does not have a solution. We call such a numerical solution a *pseudosolution.* The FORTRAN code MIRKDC [2] is based on Runge-Kutta methods

*Supported by the Natural Sciences and Engineering Research Council of Canada.
The authors would like to acknowledge the many helpful suggestions provided by the referees.

with continuous extensions and it also solves BVPs by controlling residuals. Experimentation showed that it can also produce pseudosolutions, suggesting that they can arise in any method that employs residual control.

For a BVP with solution $y(x)$, the size of the residual of a numerical solution $S(x)$ is related to the size of the error $y(x) - S(x)$ by a conditioning constant for the BVP. After some preliminaries, we begin with a discussion of residual control and pseudosolutions from the point of view of backward error analysis. We then investigate a conditioning constant that is appropriate to understanding codes that control residuals. Next, we show how to estimate this conditioning constant using quantities formed whilst solving the BVP. Finally, we show how to make this estimate practical in bvp4c and MIRKDC by carefully using software for estimating a matrix condition number developed by Higham and Tisseur [3,4]. An inexpensive estimate of this conditioning constant improves the robustness of these BVP solvers and, in particular, helps users recognize the possibility of a pseudosolution. Some numerical examples illustrate the theoretical developments.

2. PRELIMINARIES

We make heavy use in this paper of results from Ascher, Mattheij and Russell [5] and, in particular, all results that we describe as "standard" are found in this text. Our attention is sharply focused on the methods implemented in bvp4c and MIRKDC, namely implicit Runge-Kutta methods with continuous extensions. (bvp4c implements a collocation method, but for first-order ODEs, collocation methods are equivalent to a class of implicit Runge-Kutta methods with continuous extension [6].) When approximating the solution of a two-point BVP

$$y'(x) = f(x, y(x)), \tag{1}$$

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

on a mesh $a = x_1 < \cdots < x_{N+1} = b$, such methods generally involve approximations at intermediate points in addition to the accurate approximations at mesh points $y_n \approx y(x_n)$. By analytical condensation in the case of bvp4c and by the special form of the Runge-Kutta methods in the case of MIRKDC, these intermediate approximations are eliminated and a numerical solution $Y = [y_1, \ldots, y_{N+1}]$ of approximations at mesh points only is computed. The discrete problem then has the form of a set of algebraic equations

$$\Phi(Y) = 0. \tag{3}$$

After solving this set of nonlinear algebraic equations to obtain approximations at the mesh points, continuous extensions are used to obtain an approximate solution $S(x) \in C^1[a, b]$.

It is usual to discuss first BVPs with linear ODEs and linear boundary conditions and then to apply the results to nonlinear problems by linearizing about a solution of interest. The second part is standard and the first exposes the issues, so we give our attention to BVPs of the form

$$y'(x) = A(x)y(x) + q(x), \tag{4}$$

$$\beta = B_a y(a) + B_b y(b), \tag{5}$$

with smooth coefficients $A(x) \in R^{n\times n}$, $q(x) \in R^n$, $y(x) \in R^n$, and $\beta \in R^n$, $B_a, B_b \in R^{n\times n}$. Standard results state that if $Y(x)$ is the fundamental solution of the ODEs, then the BVP has a unique solution $y(x)$ if, and only if, the matrix $Q = B_a Y(a) + B_b Y(b)$ is nonsingular. When there is a solution, it can be expressed in terms of the Green's function $G(x, t)$ as

$$y(x) = Y(x)Q^{-1}\beta + \int_a^b G(x, t)q(t)\, dt. \tag{6}$$

3. BACKWARD ERROR ANALYSIS

Codes like MIRKDC and bvp4c that control the residual adopt the view of backward error analysis. In this view, a good numerical solution of a problem is one that is the exact solution of a problem with data close to the data of the given problem. Whether this solution is close to the exact solution of the given problem is then a matter of the conditioning of the problem. In principle, it is clear that there is a possibility of computing a numerical solution with a small residual to a problem that has no solution. As mentioned earlier, we call such numerical solutions pseudosolutions and show by example in Section 7 that they can occur in practice.

Though not usually described in this way, codes for solving numerically initial value problems (IVPs) for ODEs can be regarded as controlling residuals. In the first instance, the methods produce a numerical solution at mesh points only. The codes control local errors and it is fundamental that this controls the true (global) error only indirectly. More specifically, how accurate such a solution is depends on the stability (conditioning) of the IVP. Some methods have continuous extensions that provide a piecewise-polynomial solution $S(x)$ for the whole interval $[a, b]$. For theoretical purposes it is always possible to augment standard methods for IVPs with a piecewise-smooth continuous extension [7]. Moreover, the residual of $S(x)$ can have a size comparable to the local error controlled by the code. With this in mind, we can say that standard methods for solving IVPs control the size of the residual of a piecewise-smooth function $S(x)$. Whether $S(x)$ is close to $y(x)$ depends then on the conditioning of the IVP.

We can use the observation of the preceding paragraph to understand that shooting codes for BVPs can be interpreted as controlling residuals. A simple shooting code proceeds as follows. Some components of $y(a)$ are not known. With approximations for these components, the ODEs (1) are integrated from a to b with local error control that produces a numerical solution $S(x)$ with a small residual. The approximations at the two end points are substituted into the boundary conditions (2), and a residual $\delta = g(S(a), S(b))$ is found. A Newton iteration is used to find values of the unknown components that make $\delta = 0$. From this we see that the code controls the residual of $S(x)$ in the ODEs and its residual δ in the boundary conditions. A multiple shooting code partitions $[a, b]$ and solves a local IVP with approximate initial conditions over each subinterval. Matching conditions at the ends of subintervals and boundary conditions are then applied, yielding a large nonlinear algebraic system whose unknowns are the mesh point solution approximations which provide the initial conditions. This nonlinear system is solved by a Newton iteration and upon convergence this scheme produces a piecewise-smooth function $S(x)$ that satisfies the ODEs and boundary conditions with small residuals. We do not claim that residual control is a good description of what is done in practice when solving IVPs, hence when solving BVPs with a shooting method. Rather we claim that the residual control seen in MIRKDC and bvp4c is not so very different from what is done in the more familiar situation of a shooting code.

There is an obvious distinction between the error controls of codes for IVPs and BVPs. It is considered too expensive to control the global (true) error when solving IVPs, so there are very few codes that do. On the other hand, solving BVPs is more difficult and for popular methods it is possible to control an estimate of the global error at little additional expense. For this reason, BVP codes like MIRKDC and bvp4c that control residuals instead of estimates of the global error are unusual. On the other hand, residual control is a reasonable way to proceed according to backward error analysis, it has been very successful with IVPs, and it has important advantages with respect to robustness.

Experience with IVPs helps us appreciate the distinction between control of residuals and the true error, but the IVPs that arise in practice invariably have a solution, at least locally, and this is not true for BVPs. The solution of a system of linear algebraic equations

$$Ax = b, \tag{7}$$

provides a better analogy in this respect. The backward error analysis point of view is fundamental to understanding numerical linear algebra. In particular, we expect Gaussian elimination to produce a solution z that satisfies (7) with a small residual r

$$Az = b + r. \tag{8}$$

It is generally appreciated that this does not imply that z is close to x, that being a matter of the conditioning of the problem. Suppose now that A is singular and (7) is inconsistent but z is such that (8) is consistent for some r with $\|r\|$ small. We would call such a z a pseudosolution of (7). The given problem (7) has no solution, but z is a solution in the sense of backward error analysis—it is the exact solution of a problem with data close to that of the given problem. An important difference between solving systems of linear algebraic equations and BVPs is that (7) is usually solved as accurately as possible while BVPs are often solved to modest accuracy using crude meshes. Furthermore, one is more likely to compute a pseudosolution when the requested tolerance is coarse. For this reason, while pseudosolutions might arise in the more familiar context of solving a system of linear algebraic equations, they are much more likely to arise when solving BVPs with methods that control residuals.

From the point of view of backward error analysis, any numerical solution $S(x)$ of a BVP that has small residuals in the ODEs and boundary conditions is a good solution. Whether the global error $y(x) - S(x)$ is small depends on the conditioning of the BVP, so we investigate now a conditioning constant appropriate for BVP solvers that control residuals.

4. THE CONDITIONING CONSTANT κ

In this section, we investigate a conditioning constant for linear BVPs (4),(5). The analysis is closely related to that of [5, pp. 111–112], but the conditioning constant itself is tailored for methods that control the residual. Furthermore, we consider the role of weighted norms carefully because they are essential for application of the theory to the codes. Because we are interested in residual control, we consider only numerical methods that produce a piecewise-smooth solution $S(x) \in C^1[a, b]$. An approximate solution with this much smoothness satisfies the ODEs with a residual $r(x)$,

$$S'(x) = A(x)S(x) + q(x) + r(x),$$

and the boundary conditions with a residual δ,

$$\beta + \delta = B_a S(a) + B_b S(b).$$

With these definitions and representation (6), it is then immediate that

$$S(x) - y(x) = Y(x)Q^{-1}\delta + \int_a^b G(x,t)r(t)\,dt. \tag{9}$$

In defining a realistic conditioning constant, we have to account for the fact that the codes measure residuals in a relative sense. For instance, the codes measure the residual in the ODEs relative to $f(x, S(x))$. Accordingly, for the conditioning constant we assume that the residual $r(x)$ in the ODEs is measured relative to a weight function $w_1(x)$ given as a diagonal matrix with positive entries

$$\|r(x)\|_{w_1} = \max_{a \le x \le b} \left\| w_1^{-1}(x) r(x) \right\|_\infty .$$

Similarly, the residual δ in the boundary conditions is measured relative to a constant diagonal matrix w_2 with positive entries

$$\|\delta\|_{w_2} = \left\| w_2^{-1} \delta \right\|_\infty .$$

It is natural to measure the size of the error $y(x)-S(x)$ relative to a quantity that depends on $S(x)$, hence with a weight function different from that used to measure the residual in the ODEs. To account for this, we assume that the error is measured relative to a weight function $w_3(x)$ given as a diagonal matrix with positive entries

$$\|S(x)-y(x)\|_{w_3} = \max_{a\le x\le b} \left\|w_3^{-1}(x)(S(x)-y(x))\right\|_\infty .$$

Using the various weights, a little manipulation of (9) results in

$$w_3^{-1}(x)(S(x)-y(x)) = \left(w_3^{-1}(x)Y(x)Q^{-1}w_2\right)\left(w_2^{-1}\delta\right) + \int_a^b \left(w_3^{-1}(x)G(x,t)w_1(t)\right)\left(w_1^{-1}(t)r(t)\right)\,dt.$$

From this equation it is then easy to deduce the bound

$$\|y(x)-S(x)\|_{w_3} \le \kappa \max\left(\|r(x)\|_{w_1}, \|\delta\|_{w_2}\right). \tag{10}$$

In this

$$\kappa = \max_{a\le x\le b}\left(\int_a^b \left\|w_3^{-1}(x)G(x,t)w_1(t)\right\|_\infty dt + \left\|w_3^{-1}(x)Y(x)Q^{-1}w_2\right\|_\infty\right) \tag{11}$$

is a conditioning constant for the BVP because it relates the size of the error to the sizes of the residuals in the ODEs and boundary conditions.

Deuflhard [8] discusses the solution of BVPs by multiple shooting. In this he develops a measure of the sensitivity of a BVP and applies it to an interesting example that we examine in Section 7. In his approach to the solution of linear BVPs (4),(5), it is assumed that the fundamental solution $Y(x)$ satisfies $Y(a)=I$. A solution of the BVP is sought in the form

$$y(x) = Y(x)s + y_p(x),$$

involving a particular solution $y_p(x)$ of (4). Then, $y(x)$ is a solution of (4) for any s and it satisfies (5) if, and only if,

$$Qs = [B_a + B_bY(b)]s = \beta - [B_ay_p(a) + B_by_p(b)].$$

Deuflhard describes Q as a sensitivity matrix and computes it numerically. He advocates using a QR decomposition of the matrix for solving linear systems. With such a decomposition, he obtains a lower bound on the condition of Q in the two norm from the maximum ratio of the elements on the diagonal of R.

The Jacobian of $y(x)$ with respect to β satisfies

$$\frac{\partial y(x)}{\partial\beta} = Y(x)\frac{\partial s}{\partial\beta} = Y(x)Q^{-1}. \tag{12}$$

Recalling that $Y(a)=I$, we see that the sensitivity of the solution at $x=a$ to perturbations in the boundary values β can be measured by $\|Q^{-1}\|$, an interpretation of Deuflhard's measure of sensitivity that is more closely related to our approach. However, it is clear from (12) that it would be better to assess the sensitivity of the solution throughout $[a,b]$ with

$$\max_{a\le x\le b}\left(\|Y(x)Q^{-1}\|\right)$$

(which is at least as big as $\|Q^{-1}\|$ because $Y(a)=I$). Comparing this quantity to (11) in this case of no weights, we see that if Deuflhard's measure of sensitivity indicates ill-conditioning, then so will our conditioning constant κ.

5. ESTIMATING κ

Generally, we cannot guarantee mathematically that a BVP has a solution, so we solve problems numerically with the expectation on physical grounds that there is a solution. If the BVP does not have a solution or the matter is computationally unclear, we would like a BVP solver to provide some kind of warning. We have seen that the conditioning of the BVP is fundamental. In this section we investigate an estimate for the conditioning constant (11).

It is standard result that discretization of the ODEs with a one-step method and the boundary conditions result in a system of linear algebraic equations with matrix

$$A = \begin{pmatrix} S_1 & R_1 & & & & \\ & S_2 & R_2 & & & \\ & & \ddots & & & \\ & & & S_N & R_N \\ B_a & & & & B_b \end{pmatrix}. \tag{13}$$

To state a standard result about A, we define $h_i = x_{i+1} - x_i$, $h = \max h_i$, $D = \operatorname{diag}\{R_1^{-1}, \ldots, R_N^{-1}, I\}$, and

$$M^{-1} = \begin{pmatrix} G(x_1, x_2) & \cdots & G(x_1, x_{N+1}) & Y(x_1)Q^{-1} \\ \vdots & & \vdots & \vdots \\ G(x_{N+1}, x_2) & \cdots & G(x_{N+1}, x_{N+1}) & Y(x_{N+1})Q^{-1} \end{pmatrix}.$$

The quantities G, Y, and Q here were defined in Section 4 and $\{x_i\}_{i=1}^{N+1}$ are the meshpoints partitioning $[a, b]$, defined in Section 2. Theorem 5.38 of [5] states that

$$A^{-1} = M^{-1}D + O(h). \tag{14}$$

This implies that for a sufficiently fine mesh,

$$\|A^{-1}\| \approx \|M^{-1}D\|. \tag{15}$$

The way that we estimate the conditioning constant is clearer if we begin with the case of the unweighted maximum norm treated in [5], where it is shown in Theorem 5.38 that

$$\|M^{-1}D\|_\infty = \max_i \left(\sum_{j=1}^{N} h_j \|G(x_i, x_{j+1})\|_\infty + O(h) + \|Y(x_i)Q^{-1}\|_\infty \right). \tag{16}$$

Interpreting the sums as approximations to integrals and using (15), we find that

$$\|A^{-1}\|_\infty \approx \max_{a \le x \le b} \left(\int_a^b \|G(x,t)\|_\infty \, dt + \|Y(x)Q^{-1}\|_\infty \right). \tag{17}$$

The right-hand side of this expression is the conditioning constant (11) for this case. The BVP solvers actually form the matrix A, so if we can estimate the norm of A^{-1}, we can estimate κ. We remark that this is not the same as estimating the condition number of the matrix A, which involves $\|A\|$ too, a point emphasized by Ascher, Mattheij and Russell in their discussion of conditioning constants for BVPs.

Now, we modify (17) to account for the weights appearing in the definition (11) of κ. In view of the form (13) of the matrix A, we use the diagonal weight matrices $w_1(x)$, w_2, $w_3(x)$ of Section 4 to define two block diagonal matrices

$$W_{12} = \operatorname{diag}\{w_1(x_2), \ldots, w_1(x_{N+1}), w_2\},$$
$$W_3 = \operatorname{diag}\{w_3(x_1), \ldots, w_3(x_{N+1})\}.$$

Introducing these matrices into (16) results in

$$\left\|W_3^{-1}M^{-1}DW_{12}\right\|_\infty = \max_i \left(\sum_{j=1}^{N} h_j \left\|w_3^{-1}(x_i)G(x_i,x_{j+1})w_1(x_{j+1})\right\|_\infty \right. \\ \left. + O(h) + \left\|w_3^{-1}(x_i)Y(x_i)Q^{-1}w_2\right\|_\infty \right).$$

Approximating sums by integrals and using (14) leads to

$$\left\|W_3^{-1}A^{-1}W_{12}\right\|_\infty \approx \max_{a\le x\le b} \left(\int_a^b \left\|w_3^{-1}(x)G(x,t)w_1(t)\right\|_\infty dt + \left\|w_3^{-1}(x)Y(x)Q^{-1}w_2\right\|_\infty \right).$$

Comparing this expression to (11), we see that for a sufficiently fine mesh, we have an approximation to the conditioning constant

$$\left\|W_3^{-1}A^{-1}W_{12}\right\|_\infty \approx \kappa. \tag{18}$$

6. COMPUTING AN ESTIMATE FOR κ

The question we must answer now is whether we can estimate κ using (18) without significant additional computational expense. Higham and Tisseur [3,4] have developed an iterative procedure for estimating the one norm of a matrix C and implemented it in MATLAB as `normest1` and in FORTRAN as DLACON [9]. It provides a good estimate in just a few iterations, each requiring the evaluation of CX and $C^{\mathsf{T}}X$ for certain matrices X. The algorithm does not need access to the matrix C itself, just the result of multiplying X on the left by C or C^{T}. Within each iteration the algorithm provides X together with an indication of which product it requires; we compute this product in an auxiliary computation and return the result.

We now consider how to use this algorithm to estimate inexpensively the conditioning constant κ. We first observe that for any matrix C, $\|C\|_\infty = \|C^{\mathsf{T}}\|_1$, so to estimate conditioning constant by (18), we can apply the algorithm of Higham and Tisseur to

$$\left(W_3^{-1}A^{-1}W_{12}\right)^{\mathsf{T}}.$$

This requires us to compute the matrix products

$$V = \left(W_3^{-1}A^{-1}W_{12}\right)X \qquad \text{and} \qquad V = \left(W_3^{-1}A^{-1}W_{12}\right)^{\mathsf{T}}X.$$

The first kind of product is computed by solving

$$AU = W_{12}X, \qquad \text{and then solving } W_3V = U.$$

The second kind is computed by first solving

$$W_3Z = X, \qquad \text{then solving } A^{\mathsf{T}}U = Z, \qquad \text{and finally forming } V = W_{12}U.$$

The weights are diagonal matrices so the computations involving them are easy and inexpensive. The question, then, is how to solve efficiently linear systems involving A and its transpose. Here is where we must recognize that A is a large, sparse, highly-structured matrix. Because bvp4c and MIRKDC deal with this in different ways, the rest of the discussion is particular to the solver.

Sparse matrix technology is fully integrated into MATLAB, so bvp4c forms the matrix A of (13) and holds it as a general sparse matrix. An unusual aspect of this solver is that it does explicit row scaling by choosing a diagonal scaling matrix S and then forming $\hat{A} = SA$. A sparse LU decomposition results in $P\hat{A} = LU$. The solver forms the permuted diagonal matrix $\hat{S} = PS$ for other purposes, so we have available the decomposition $\hat{S}A = LU$. With this decomposition of A and the transpose and backslash operators of MATLAB, it is then both easy and efficient to solve the linear systems arising in the evaluation of the matrix products.

In MIRKDC the matrix A has a form different from (13) because this solver accepts only BVPs with separated boundary conditions. For this class of problems it is natural to order the equations differently so that A is an "almost block diagonal" matrix, cf. [10]. Taking advantage of this structure so as to store only the entries that might be nonzero makes it practical to work with large matrices. Equally important, linear systems involving such matrices can be solved efficiently and stably with no fill-in by elimination using both row and column operations. The matrix decomposition is different from the sparse LU decomposition of bvp4c, but it is used in the same way. To be specific, linear systems are solved in MIRKDC with the COLROW package [11,12]. The subroutine BSPCNDMAX [13,14] uses DLACON [9] and a modification of COLROW [15] to compute the condition number of an almost block diagonal matrix in the maximum norm. We have modified BSPCNDMAX and employed it within MIRKDC in order to estimate $\|W_3^{-1}A^{-1}W_{12}\|_\infty$.

As part of approximating the solution of the BVP, the matrix A (or a row-scaled version of it) is formed and decomposed. We see now that by solving a few more systems of linear equations using this matrix decomposition, we can estimate the conditioning constant κ. In this, we can take full advantage of the special structure of A. Indeed, it does not matter that bvp4c and MIRKDC handle storage and the efficient solution of linear systems in very different ways. To minimize the overhead, we estimate κ only when a discrete approximate solution that satisfies the error criterion has been computed.

7. NUMERICAL EXPERIMENTS

In this section, we solve three BVPs with bvp4c and MIRKDC to illustrate our theoretical developments. MIRKDC implements formulae of orders 2, 4, and 6, whereas bvp4c implements only a formula of order 4. To make the numerical results for the two codes more comparable, we present results here only for formulae of order 4. Results obtained with the other two formulae of MIRKDC were qualitatively the same. Though broadly similar from a user's point of view, the two codes differ notably in detail. In particular, the residual controls are different. bvp4c has a relative residual tolerance RelTol and an absolute residual tolerance AbsTol. The default values are RelTol $= 10^{-3}$ and AbsTol $= 10^{-6}$. This solver controls the residual in the ODEs in a relative sense, i.e., it controls

$$\frac{|S_j'(x) - f_j(x, S(x))|}{\max(|f_j(x, S(x))|, \text{AbsTol} / \text{RelTol})}, \qquad j = 1, 2, \dots, n,$$

and similarly, the residual in the boundary conditions. MIRKDC has one tolerance tol and controls

$$\frac{|S_j'(x) - f_j(x, S(x))|}{1 + |f_j(x, S(x))|}, \qquad j = 1, 2, \dots, n.$$

The residual controls of the two codes differ considerably when AbsTol is rather smaller than RelTol (as it is for the default values) and $|S'(x)|$ ($\approx |f(x, S(x))|$) is rather smaller than 1. The residual controls are roughly equivalent when tol = RelTol = AbsTol, but they still differ so much that direct comparisons are somewhat misleading.

The residual controls of the codes are a given, but in our study of conditioning constants, we must choose how to measure the error $y(x) - S(x)$. In bvp4c we measure

$$\frac{|y_j(x) - S_j(x)|}{\max(|S_j(x)|, \text{AbsTol} / \text{RelTol})}, \qquad j = 1, 2, \dots, n,$$

and in MIRKDC we measure

$$\frac{|y_j(x) - S_j(x)|}{1 + |S_j(x)|}, \qquad j = 1, 2, \ldots, n.$$

Again, these measures differ considerably when AbsTol is rather smaller than RelTol and $|S|$ is rather smaller than 1.

In either code, if the product of the estimated conditioning constant κ and the tolerance on the residual is so large that inequality (10) does not guarantee any correct digits in the numerical solution, it would be prudent to solve the problem again with a much more stringent tolerance (and a much finer mesh). In our limited experiments, when there is no solution or more than one, this has usually resulted in some kind of error message. If the solver did not actually fail, it returned a larger than expected estimate for κ, indicating that the computed solution should not be trusted.

EXAMPLE 1. Bratu's problem,

$$y'' + \lambda e^y = 0, \qquad y(0) = 0 = y(1),$$

arises in a model of spontaneous combustion. Davis [16] shows that for $0 \le \lambda < \lambda^* = 3.51383\ldots$, there are two solutions. Both have a parabolic shape and are concave down. They approach each other as $\lambda \to \lambda^*$ and merge when $\lambda = \lambda^*$. There is no solution for $\lambda > \lambda^*$.

Using bvp4c (with default tolerances), we solved the BVP with $\lambda = 3.45$ and plotted one of the solutions as the lower curve in Figure 1. This was done without any difficulty using default tolerances, an initial mesh of `linspace(0,1,10)`, initial guesses of zero for all mesh point solution values, and default finite difference approximations to partial derivatives. Indeed, solving the BVP was so easy that only the initial mesh was used. The conditioning constant was estimated with the scheme of Section 6 to be 3.4×10^3. The code had to work much harder to "solve" the BVP with $\lambda = 3.55$, but it produced a pseudosolution on a mesh of 179 points. The pseudosolution for $\lambda = 3.55$ plotted in Figure 1 looks much like the true solution for $\lambda = 3.45$, so it is not obvious that there is no true solution for this value of λ. The solver itself provided no warning message, but the conditioning constant was estimated to be 1.0×10^6. Considering inequality (10), this is so big that it raises the question as to whether a numerical solution computed with these tolerances has any correct digits.

To illustrate the effect of the accuracy requirement on the conditioning constant, we repeated these computations with the absolute residual tolerance increased so that AbsTol = RelTol = 10^{-3}. The solution and pseudosolution looked the same as those of Figure 1 and the numbers of mesh points were the same, but the conditioning constants were considerably smaller with the new set of weights. The estimated conditioning constant was 1.2×10^1 for the solution of $\lambda = 3.45$ and 8.0×10^3 for the pseudosolution of $\lambda = 3.55$. Though the constants were smaller in this experiment, the conditioning constant for the pseudosolution was again much larger than that for the solution and again it was so big that it is questionable whether the numerical solution has any correct digits. In our experiments, we also observed that when sufficiently sharp tolerances are employed, bvp4c will not compute a pseudosolution; rather the code will return with an indication that it was not able to compute a solution.

EXAMPLE 2. It is shown in [17] that the BVP

$$y'' + |y| = 0, \qquad y(0) = 0, \qquad y(\pi) = B,$$

has a unique solution for $B < 0$, infinitely many solutions for $B = 0$, and no solution for $B > 0$. We applied MIRKDC to this problem using tolerances of 10^{-3}, 10^{-5}, 10^{-7}, and a range of B values. In each case we used a uniform initial mesh of five subintervals and initial guesses of $y(x) = Bx/\pi$ and $y'(x) = B/\pi$. Specifically, we attempted to solve the BVP for $B = 0$ and

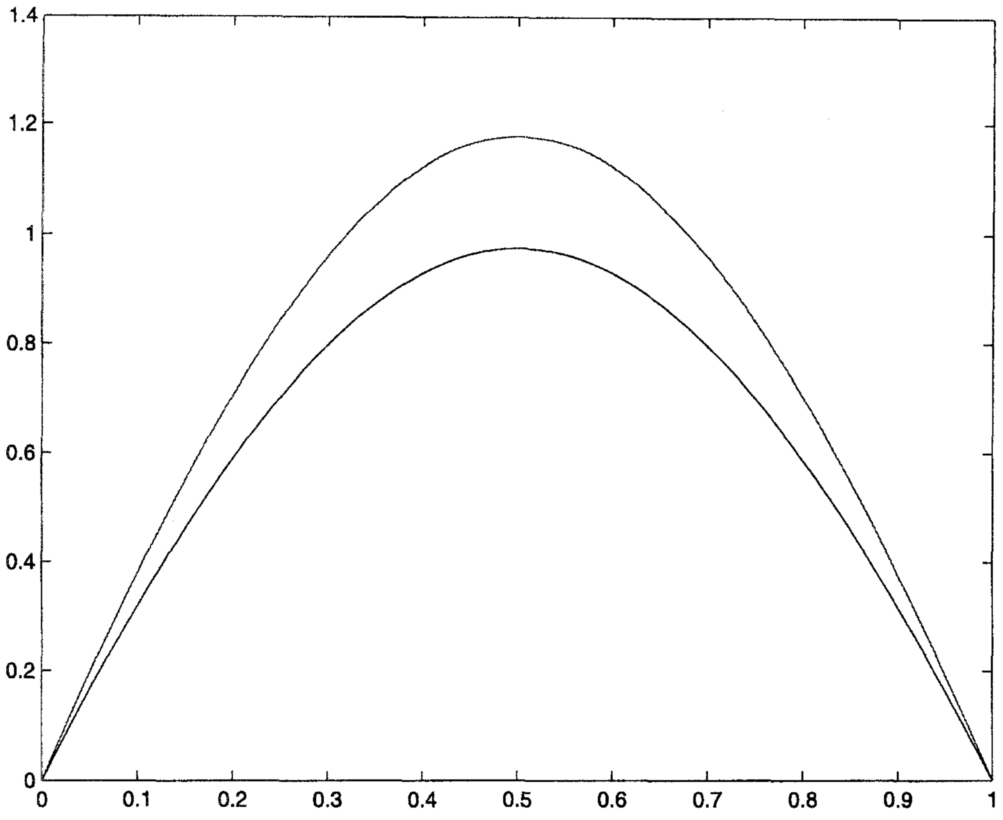

Figure 1. Numerical solutions of Example 1 computed with **bvp4c**. The lower curve is a genuine solution corresponding to the problem parameter $\lambda = 3.45$; the upper curve is a pseudosolution corresponding to $\lambda = 3.55$.

$B = \pm 10^{-q}$ for $q = 1, \ldots, 4$. For each value of B the solver returned a numerical solution, even in the cases for which the BVP has no solution or more than one. Figure 2 shows the estimated conditioning constants plotted on a logarithmic scale. (The data points are connected by straight lines to make them more visible.) For all three tolerances $\kappa(B)$ is small for the subcritical values $B < 0$. At the critical value $B = 0$ where multiple solutions exist, there is a dramatic increase in the estimated conditioning constant. The estimated conditioning constants remain large for $B > 0$ where there is no solution and the code returns pseudosolutions. The product of κ and the tolerance on the residual is so large when $B \geq 0$ that according to inequality (10), we cannot be sure that the solution has any correct digits. For this example a large conditioning constant provides a warning that there might be more than one solution or no solution at all.

Example 3. Deuflhard [8] considers the BVP

$$y'' + \frac{3\epsilon y}{(\epsilon + x^2)^2} = 0, \qquad y(\pm 0.1) = \frac{\pm 0.1}{\sqrt{\epsilon + 0.01}},$$

and notes its "sensitivity" when $\epsilon = 0.01$. Indeed, his shooting code finds a number of solutions of the BVP then, and he describes this as "a significant failure" of the code. We would not describe it that way because, with this value of ϵ, the BVP has a family of solutions. The shooting code has succeeded in producing a numerical solution with small residuals. On the other hand, we would like for a code to report that it suspects there are other solutions when $\epsilon = 0.01$.

We solved this problem with MIRKDC for a range of ϵ values and tolerances 10^{-3}, 10^{-5}, 10^{-7}. In each case, we used a uniform initial mesh of five subintervals and initial guesses of zero. The estimated conditioning constants are plotted on a logarithmic scale in Figure 3. (The data points are connected by straight lines to make them more visible.) There is indeed a "spike" in $\kappa(\epsilon)$

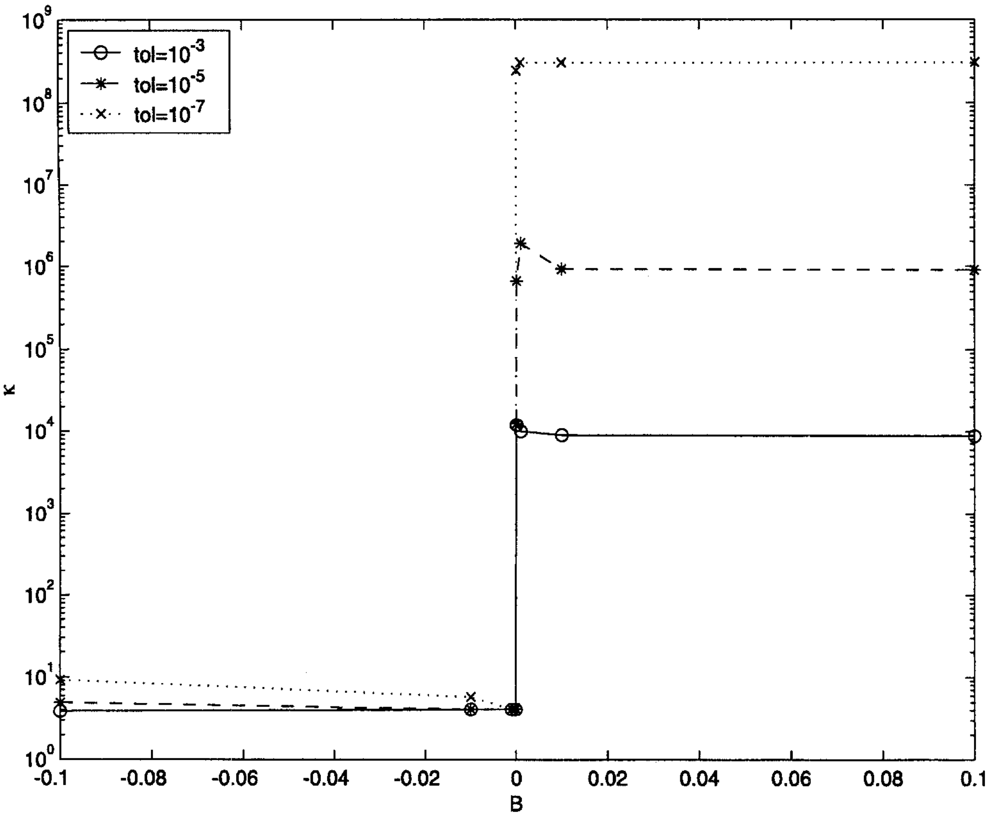

Figure 2. Estimated conditioning constant κ vs. problem parameter B of Example 2 computed with MIRKDC.

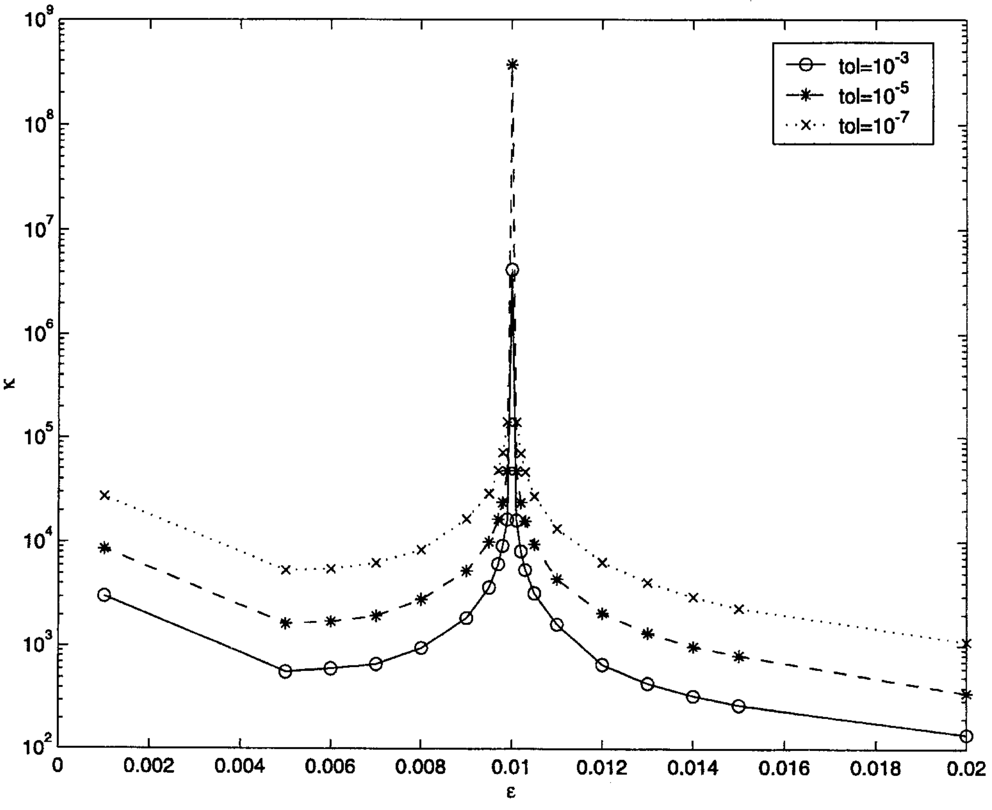

Figure 3. Estimated conditioning constant κ vs. problem parameter ϵ of Example 3 computed with MIRKDC.

at the value $\epsilon = 0.01$ corresponding to the existence of multiple solutions. (MIRKDC did not compute a solution for $\epsilon = 0.01$ when the tolerance was 10^{-7} because Newton's method failed to converge. The iteration matrices were so ill-conditioned that no correct digits were being computed.)

The estimates for κ associated with the three tolerances are in reasonable agreement when we consider that they are obtained from matrices based on different and relatively coarse meshes; our computations are based on estimates κ that are only $O(h)$ approximations—see (14) and (16)—and thus for coarse meshes with large mesh spacing h we expect only crude estimates of κ. Furthermore, we only *approximate* a matrix norm with the algorithm of Higham and Tisseur.

8. CONCLUSIONS

It is plausible that pseudosolutions are more likely when the tolerances are crude (and the mesh is coarse), and that is what we found in our experiments. With its emphasis on graphical interpretation of solutions, such tolerances are much more common when solving problems in MATLAB than in general scientific computation. By default bvp4c approximates partial derivatives internally with finite differences, though it has an option for analytical partial derivatives. MIRKDC requires analytical partial derivatives, but numerical partial derivatives are so convenient that they are to be an option in the next release of the solver. It is also plausible that pseudosolutions are more likely to be computed when partial derivatives are approximated numerically, and that has been our experience.

In this paper, we have investigated a conditioning constant appropriate to BVP solvers that control residuals and an inexpensive way to estimate this constant. In our experiments, this estimated conditioning constant κ has been quite helpful in recognizing that a BVP has no solution or more than one. In particular, it has been quite helpful in distinguishing pseudosolutions from true solutions.

REFERENCES

1. J. Kierzenka and L.F. Shampine, A BVP solver based on residual control and the MATLAB PSE, *ACM Trans. Math. Softw.* (to appear).
2. W.H. Enright and P.H. Muir, Runge-Kutta software with defect control for boundary value ODEs, *SIAM J. Sci. Comput.* **17**, 479–497, (1996).
3. N.J. Higham, FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation (Algorithm 674), *ACM Trans. Math. Softw.* **14**, 381–396, (1988).
4. N.J. Higham and F. Tisseur, A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra, *SIAM J. Matrix. Anal. Appl.* **21**, 1185–1201, (2000).
5. U.M. Ascher, R.M.M. Mattheij and R.D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, SIAM, Philadelphia, (1995).
6. R. Weiss, The application of implicit Runge-Kutta and collocation methods to boundary value problems, *Math. Comp.* **28**, 449–464, (1974).
7. L.F. Shampine, *Numerical Solution of Ordinary Differential Equations*, Chapman and Hall, New York, (1994).
8. P. Deuflhard, Nonlinear equation solvers in boundary value problems, In *Codes for Boundary-Value Problems in Ordinary Differential Equations*, (Edited by B. Childs *et al.*), pp. 40–66, Springer, New York, (1979).
9. E. Anderson, Z. Bai, C.H. Bischof, S. Blackford, J.W. Demmel, J.J. Dongarra, J.J. Du Croz, A. Greenbaum, S.J. Hammarling, A. McKenney and D. Sorensen, *LAPACK Users' Guide*, Third edition, SIAM, Philadelphia, (1999).
10. P. Amodia *et al.*, Almost block diagonal linear systems: Sequential and parallel solution techniques, and applications, *Numer. Linear Algebra Appl.* **7**, 275–317, (2000).
11. J.C. Diaz, G. Fairweather and P. Keast, FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination, *ACM Trans. Math. Softw.* **9**, 358–375, (1983).
12. J.C. Diaz, G. Fairweather and P. Keast, Algorithm 603. COLROW and ARCECO: FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination, *ACM Trans. Math. Softw.* **9**, 376–380, (1983).
13. J. Patterson, P.H. Muir and P. Keast, BSPCNDMAX, software for estimation of the max norm condition number of an almost block diagonal matrix, `http://www.mscs.dal.ca/~keast/`, (2001).
14. P. Keast and R. Affleck, BSPCND, software for estimation of the 1-norm condition number of an almost block diagonal matrix, `http://www.mscs.dal.ca/~keast/`, (1997).

15. P. Keast, COLROW, `http://www.mscs.dal.ca/~keast/`, (1992).
16. H.T. Davis, *Introduction to Nonlinear Differential and Integral Equations*, Dover, New York, (1962).
17. P.B. Bailey, L.F. Shampine and P.E. Waltman, *Nonlinear Two Point Boundary Value Problems*, Academic Press, New York, (1968).