Report no. 08/04

A Sixth-Order Extension to the MATLAB Package bvp4c of J. Kierzenka and L. Shampine

by
NICHOLAS HALE
Oxford University Computing Laboratory
and
DANIEL R. MOORE
Dept. of Mathematics, Imperial College London

A new two-point boundary value problem algorithm based upon the MATLAB bvp4c package of Kierzenka and Shampine is described. The algorithm, implemented in a new package bvp6c, uses the residual control framework of bvp4c (suitably modified for a more accurate finite difference approximation) to maintain a user specified accuracy. The new package is demonstrated to be as robust as the existing software, but more efficient for most problems, requiring fewer internal mesh points and evaluations to achieve the required accuracy.

Oxford University Computing Laboratory
Numerical Analysis Group
Wolfson Building
Parks Road
Oxford, England OX1 3QD
E-mail: nick.hale@comlab.oxford.ac.uk

April, 2008

1 Introduction

In [KS01] Kierzenka and Shampine describe a software package, bvp4c, to solve a large class of boundary value problems (BVPs) for ordinary differential equations (ODEs) in MATLAB. Specifically, they consider first order ODE systems of the form;

$$y' \;=\; f(x,y;p), \quad a \;\leq\; x \;\leq b \tag{1.1a}$$

subject to two-point boundary conditions (BCs)

$$g(y(a),\, y(b);p) \;=\; 0. \tag{1.1b}$$

They, as shall we, assume that $f(x,y;p)$ and $g(x,y;p)$ are as smooth as necessary, and in particular that f is continuous and satisfies a Lipschitz condition in y. The argument p is a vector of unknown parameters and is usually suppressed for clarity of exposition. The view of K&S is that in general, a user solving a BVP of the form (1.1) in MATLAB is interested in only a graphical representation of a solution. As such, a modest order solver such as the MIRK4 based Simpson method is appropriate for graphical accuracy.

It is our opinion that whilst a fourth-order solver is reasonable, recent developments mean that a sixth-order solver will supply not only greater accuracy, but also perform more efficiently. We thus develop a new piece of software, bvp6c, intended to solve the same class of problems efficiently, whilst maintaining the accuracy and robustness of the original software. Importantly we maintain other valuable attributes, such as treating non-separated BCs and unknown parameters without requiring the user to reformulate these as higher order problems in the canonical form (1.1).

Although MIRK (mono-implicit Runge-Kutta) formulae become more complex as order increases [CS82, DC01] - needing more function evaluations, more interior points and more arithmetic operations at each mesh point - we believe a higher order method will give comparable accuracy on a sufficiently coarser mesh (or alternatively, greater accuracy on a comparable mesh) and thus prove more efficient in the majority of cases.

2 Collocation Method

A number of existing methods for solving problems of the form (1.1) provide solutions only at mesh points, whereas others provide data everywhere but without uniform accuracy [CS82, CW91a, EM86, EM96]. Our method, like bvp4c, will provide a uniform prescribed accuracy throughout the computational interval. To achieve this we fit the MIRK6 data of [CS82] with the sixth-order interpolant described in [CM04], giving an accurate representation of the solution throughout the computational interval at little extra cost. If $h_i = x_{i+1} - x_i$, the MIRK6 formula given by [CS82] is

$$\begin{aligned} y_{i+1} \;&=\; y_i + \frac{h_i}{90}\left[7f(x_i,y_i) + 32f(x_{i+1/4},y_{i+1/4}) + 12f(x_{i+1/2},y_{i+1/2}) + \dots\right. \\ &\qquad\qquad \left. 32\,f(x_{i+3/4},y_{i+3/4}) + 7f(x_{i+1},y_{i+1})\right], \qquad (2.1) \\ &=\; y_i + \frac{h_i}{90}\left[7y'_i + 32y'_{i+1/4} + 12y'_{i+1/2} + 32y'_{i+3/4} + 7y'_{i+1}\right] \end{aligned}$$

where

$$\begin{aligned} y_{i+1/4} &= \frac{1}{64}\left(54y_i + 10y_{i+1} + h_i\left[9y'_i - 3y'_{i+1}\right]\right), & (2.2a)\\ y_{i+3/4} &= \frac{1}{64}\left(10y_i + 54y_{i+1} + h_i\left[3y'_i - 9y'_{i+1}\right]\right), & (2.2b)\\ y_{i+1/2} &= \frac{1}{2}\left(y_i + y_{i+1}\right) - \frac{h_i}{24}\left[5y'_i - 16y'_{i+1/4} + 16y'_{i+3/4} - 5y'_{i+1}\right]. & (2.2c) \end{aligned}$$

The algorithm used in bvp4c can be viewed as a collocation with a piecewise cubic polynomial function used to fit the data $\{y_i,\ y'_i,\ y_{i+1},\ y'_{i+1}\}$ from a three-point Lobatto IIIA approximation. This polynomial collocates, satisfies the boundary conditions and is continuous at the ends of each subinterval $[x_i, x_{i+1}]$, making it C^1 in the whole of $[a, b]$. In our algorithm we use the quintic interpolant of [CM04] to fit the MIRK6 $\{y_i,\ y'_i,\ y'_{i+1/2},\ y'_{i+3/4} - y'_{i+1/4},\ y_{i+1},\ y'_{i+1}\}$ in (2.1,2.2). If $\omega = (x - x_i)/h_i \in [0, 1]$, this interpolant $S(x)$ is defined locally for $x \in [x_i, x_{i+1}]$ by

$$\begin{aligned} S(x_i + \omega h_i) = {} & A_{66}(\omega)y_{i+1} + A_{66}(1-\omega)y_i + \\ & h_i\left[B_{66}(\omega)y'_{i+1} - B_{66}(1-\omega)y'_i + \right.\\ & \left. C_{66}(\omega)\{y'_{i+3/4} - y'_{i+1/4}\} + D_{66}(\omega)\overline{y}'_{i+1/2}\right], \end{aligned} \tag{2.3}$$

where

$$\begin{aligned} A_{66}(\omega) &= \omega^2\left(15 - 50\omega + 60\omega^2 - 24\omega^3\right),\\ B_{66}(\omega) &= \omega^2\left(\omega - 1\right)\left(12\omega^2 - 14\omega + 5\right)/3,\\ C_{66}(\omega) &= -8\omega^2\left(1-\omega\right)^2/3,\\ D_{66}(\omega) &= 8\omega^2\left(\omega - 1\right)^2\left(2\omega - 1\right) \end{aligned} \tag{2.4}$$

and

$$\overline{y}_{i+1/2} = \frac{1}{2}\left(y_{i+1} + y_i\right) - \frac{h_i}{24}\left[y'_{i+1} - y'_i + 4(y'_{i+3/4} - y'_{i+1/4})\right]. \tag{2.5}$$

Here (2.5) is a more accurate estimate of $y_{i+1/2}$ such that $\overline{y}'_{i+1/2} - y'(x_{i+1/2}) = O(h^6)$, and an additional function evaluation is required to find the updated $\overline{y}'_{i+1/2} = f(x, \overline{y}_{i+1/2})$. As with the fourth-order interpolant of bvp4c, the continuity and collocation properties imply $S(x)$ is C^1 in $[a, b]$. Furthermore, the accuracy of $\overline{y}'_{i+1/2}$ and the other MIRK6 data arising from (2.1,2.2) ensures that $S(x)$ is sixth-order accurate across the mesh interval [CM04]. Hence for $x \in [x_i, x_{i+1}]$ and each i

$$S(x) - y(x) = O(h_i^6), \tag{2.6}$$

and with the assumed Lipschitz condition on f, this implies

$$f(x, S(x)) = f(x, y(x)) + O(h_i^6). \tag{2.7}$$

3 Residual

As with bvp4c, the cornerstone of bvp6c is that both error estimation and mesh selection are based on the residual of $S(x)$, defined by

$$\begin{aligned} r(x) &= S'(x) - f(x, S(x)), \\ r_{BCs}(x) &= g(S(a), S(b)) \end{aligned} \tag{3.1}$$

for the ODEs and for the BCs respectively. Applying (2.7) to the ODE residual gives

$$\begin{aligned} r(x) &= (S'(x) - f(x, S(x))) - (y'(x) - f(x, y(x))), \\ &= S'(x) - y'(x) + O(h_i^6), \end{aligned} \tag{3.2}$$

for x in each $[x_i, x_{i+1}]$. We now seek to establish the asymptotic behaviour of this residual. Let $q(x)$ be the interpolant (2.3,2.4), but of the true solution $y(x)$. Subtracting from S and differentiating leads to

$$\begin{aligned} S'(x) - q'(x) &= S'(x_i + \omega h_i) - q'(x_i + \omega h_i), \\ &= \frac{1}{h_i}\left[A'_{66}(w)\,(y_{i+1} - y(x_{i+1})) - A'_{66}(1-w)\,(y_i - y(x_i))\right] + \\ &\quad B'_{66}(w)\left(y'_{i+1} - y'(x_{i+1})\right) + B'_{66}(1-w)\,(y'_i - y'(x_i)) + \\ &\quad C'_{66}(w)\left(\{y'_{i+3/4} - y'_{i+1/4}\} - \{y'(x_{i+3/4}) - y'(x_{i+1/4})\}\right) + \\ &\quad D'_{66}(w)\left(\overline{y}'_{i+1/2} - y'(x_{i+1/2})\right). \end{aligned} \tag{3.3}$$

The errors in $y'_i - y'(x_i)$, $\overline{y}'_{i+1/2} - y'(x_{i+1/2})$ and $y'_{i+1} - y'(x_{i+1})$ are $O(h_i^6)$ [CM04], and B'_{66} and D'_{66} are $O(1)$, hence

$$\begin{aligned} S'(x) - q'(x) &= \frac{1}{h_i}\left[A'_{66}(w)\,(y_{i+1} - y(x_{i+1})) - A'_{66}(1-w)\,(y_i - y(x_i))\right] \\ &\quad C'_{66}(w)\left(\{y'_{i+3/4} - y'_{i+1/4}\} - \{y'(x_{i+3/4}) - y'(x_{i+1/4})\}\right) + O(h_i^6). \end{aligned} \tag{3.4}$$

The sixth-order MIRK6 formula (2.1) is satisfied by $y(x)$ with an local truncation error $O(h_i^7)$, i.e. $y_{i+1} - y(x_{i+1}) = y_i - y(x_i) + O(h_i^7)$ [CS82]. Moreover,

$$\begin{aligned} \{y'_{i+\frac{3}{4}} - y'_{i+\frac{1}{4}}\} - \{y'(x_{i+\frac{3}{4}}) - y'(x_{i+\frac{1}{4}})\} &= \frac{3h_i^5}{1024}\left[\frac{1}{5}\frac{\partial f}{\partial y}y^{v} - \frac{1}{4}\frac{d}{dx}\left(\frac{\partial f}{\partial y}y^{iv}\right)\right] + O(h_i^6) \\ &=: err_{C66}h_i^5 + O(h_i^6). \end{aligned} \tag{3.5}$$

This leads to

$$S'(x) - q'(x) = \frac{1}{h_i}\left(A'_{66}(w) - A'_{66}(1-w)\right)(y_i - y(x_i)) + C'_{66}(w)err_{C66}h_i^5 + O(h_i^6), \tag{3.6}$$

but noting $A'_{66}(w) - A'_{66}(1-w) = 0$ reduces this to

$$S'(x) - q'(x) = C'_{66}(w)err_{C66}h_i^5 + O(h_i^6). \tag{3.7}$$

Substituting (3.7) to the expression for the residual in (3.1), we have

$$\begin{aligned} r(x) &= S'(x) - f(x, S(x)), \\ &= S'(x) - y'(x) + O(h_i^6), \\ &= q'(x) - y'(x) + C'_{66}(w) err_{C_{66}} h_i^5 + O(h_i^6), \end{aligned} \tag{3.8}$$

and [CM04] gives the interpolation error of $q(x)$ to $y(x)$ in $[x_i, x_{i+1}]$ as

$$\frac{h_i^6}{720} w^2 (1-w)^2 \left(w^2 - w + \frac{9}{32} \right) y^{vi} \big|_{\xi \in [0,1]} . \tag{3.9}$$

Thus, differentiating (3.9) with respect to x and substituting to (3.8) gives

$$\begin{aligned} r(x) &= \frac{h^5}{3840} w(4w-1)(2w-1)(4w-3)(w-1) y^{vi}(\xi) + h^5 C'_{66}(w) err_{C_{66}} + O(h^6) \\ &= h^5 \left[\frac{1}{3840} w(4w-1)(2w-1)(4w-3)(w-1) y^{vi}(\xi) - \dots \right. \\ &\qquad \left. \frac{16}{3} w(2w-1)(w-1) err_{C_{66}} \right] + O(h^6). \end{aligned} \tag{3.10}$$

The behaviour of the residual then is local to each subinterval and of order h_i^6 at the two end-points and the midpoint. To leading order the residual is a polynomial of degree 5, with dependence on y^{iv}, y^v and y^{vi}. This extra dependence not present in the bvp4c residual is complex, and in particular leaves us unable to determine local extrema or an asymptotically correct estimate of the L_∞ norm. However, in [KS01] the use of the L_∞ norm is argued against, and we instead focus on the L_2 norm. On each subinterval we compute

$$\|r(x)\|_i = \left(\int_{x_i}^{x_{i+1}} r^2(x) dx \right)^{1/2}, \tag{3.11}$$

and use a suitable quadrature method to integrate the now leading order degree ten polynomial $r^2(x)$ accurately. An n-point Lobatto quadrature rule is exact for polynomials of degree $2n-3$, so for an asymptotically correct estimate of our residual we need at least a 7-point procedure, requiring four extra function evaluations per subinterval, five including the evaluation to compute the more accurate $\overline{y}'_{i+1/2}$.

4 Implementation

The new software bvp6c is intended to be a direct extension to bvp4c, and as such implementation is almost identical, with support routines changed only where necessary to maintain sixth-order accuracy. We also wish to retain other functionality of the original package, such as solving problems with unknown parameters, generalised two-point BCs, not requiring (although allowing) user entered partial derivatives and the vectorisation of $f(x, y)$. Fortunately the change from fourth- to sixth-order has only a small impact on these areas, and little alteration is necessary. We discuss here some of the larger impacts the higher order extension has on implementation of the method.

4.1 Collocation Equations and Jacobians

The collocation method of bvp6c is applied to (1.1) on a mesh $a = x_0 \leq x_1 \leq \ldots \leq x_N = b$ by solving the algebraic equations

$$\Phi(X, Y) = 0, \tag{4.1}$$

where

$$\begin{aligned} X &= [x_0, x_1, \ldots, x_N]^T, \\ Y &= [y_0, y_1, \ldots, y_N, p]^T, \end{aligned} \tag{4.2}$$

and

$$\begin{aligned} \Phi_0(X, Y) &= g(y_0, y_N; p), & \text{(4.3a)} \\ \Phi_{i+1}(X, Y) &= y_{i+1} - y_i - \frac{h_i}{90}\left[7y'_i + 32y'_{i+1/4} + 12y'_{i+1/2} + 32y'_{i+3/4} + 7y'_{i+1}\right], \\ & \qquad i = 0, 1, \ldots, N-1. & \text{(4.3b)} \end{aligned}$$

The system (4.1) is solved using a simplified Newton (chord) method, which requires the global Jacobian $\partial\Phi/\partial Y$. The form of the Jacobian here is more complex than for the fourth-order Simpson method, and computationally more expensive. If one calculates the Jacobians as set out below, three matrix multiplications per Jacobian are required. In contrast, the fourth-order method of [KS01] required only one matrix multiplication per Jacobian. The global Jacobians here are given by

$$\begin{aligned} \frac{\partial \Phi_{i+1}}{\partial y_i} = I \; + \; & \frac{h}{90}\Big[7J_i + 27J_{i+1/4} + 6J_{i+1/2} + 5J_{i+3/4} & \text{(4.4a)} \\ + \; & \frac{h}{8}J_{i+1/2}\left\{54J_{i+1/4} - 10J_{i+3/4} + 3\,h\left(3J_{i+1/4} - J_{i+3/4}\right)J_i\right\} \\ + \; & \frac{h}{2}\left\{9J_{i+1/4} - 5J_{i+1/2} + 3J_{i+3/4}\right\}J_i\Big], \\ \frac{\partial \Phi_{i+1}}{\partial y_{i+1}} = -I \; + \; & \frac{h}{90}\Big[5J_{i+1/4} + 6J_{i+1/2} + 27J_{i+3/4} + 7J_{i+1} & \text{(4.4b)} \\ + \; & \frac{h}{8}J_{i+1/2}\left\{10J_{i+1/4} - 54J_{i+3/4} - 3\,h\left(J_{i+1/4} - 3J_{i+3/4}\right)J_{i+1}\right\} \\ - \; & \frac{h}{2}\left\{3J_{i+1/4} - 5J_{i+1/2} + 9J_{i+3/4}\right\}J_{i+1}\Big], \end{aligned}$$

where $J_l = \partial f(x_l, y_l)/\partial y$. The global Jacobian of Φ with respect to the parameters p is sufficiently complicated to merit setting out explicitly as well:

$$\begin{aligned}\frac{\partial\Phi_{i+1}}{\partial p} &= \frac{h}{90}\left[7\left(K_i+K_{i+1}\right)+32\left(K_{i+1/4}+K_{i+3/4}\right)+12K_{i+1/2}\right. \\ &\quad +\frac{h}{2}J_{i+1/2}\left\{5\left(K_{i+1}-K_i\right)+16\left(K_{i+1/4}-K_{i+3/4}\right)+\dots\right. \\ &\qquad\qquad \left.\frac{3h}{4}\left(J_{i+1/4}\left(3K_i-K_{i+1}\right)-J_{i+3/4}\left(K_i-3K_{i+1}\right)\right)\right\} \\ &\quad \left.+\frac{3h}{2}\left\{J_{i+1/4}\left(3K_i-K_{i+1}\right)+J_{i+3/4}\left(K_i-3K_{i+1}\right)\right\}\right],\end{aligned} \tag{4.4c}$$

where here $K_l = \partial f(x_l, y_l)/\partial p$.

In bvp4c the local internal Jacobian $J_{i+1/2}$ is approximated by an average when not changing rapidly. Here we have two additional internal Jacobians to calculate ($J_{i+1/4}$ and $J_{i+3/4}$), making such an approximation even more attractive. In [CS82] further simplifications for these Jacobians are set out that might be used to reduce overall computation time, but we have not applied these in current implementation. Our trials (see §5.3) suggest that although a user supplied Jacobian is not necessary, it can reduce computation speed compared with calculating these Jacobian by numerical differences. This result for the MATLAB package differs from Fortran implementations of the MIRK6 algorithm for two-point BVPs, where numerical difference calculation of the Jacobians is substantially faster than calculating the analytic matrix multiplication forms for all but the most complicated right hand sides [CGM02].

4.2 Residual Norm

Having found an approximation to the solution by solving the (4.1), we must now compute the norm of the residual (3.11) on each subinterval. As discussed in the previous chapter, we choose the 7-point Lobatto quadrature method, the abscissa and weights of which can be found in [Mic63]. This method will integrate a polynomial of degree 10 exactly, and the quadrature points include the end and the midpoints of the interval. Suppose we evaluate the interpolant S and its derivative at $x_{i+1/2}$,

$$\begin{aligned} S(x_{i+1/2}) &= \frac{1}{2}(y_{i+1}+y_i)-\frac{h_i}{24}\left[y'_{i+1}-y'_i+4(y'_{i+3/4}-y'_{i+1/4})\right] \qquad (4.5)\\ &= \overline{y}_{i+1/2}, \\ S'(x_{i+1/2}) &= \overline{y}'_{i+1/2}. \qquad (4.6)\end{aligned}$$

We see that this is exact for both $\overline{y}$ and $\overline{y}'$ (the improved approximations) at the midpoint, and as such

$$\begin{aligned} r(x_{i+1/2}) &= S'(x_{i+1/2})-f(x_{i+1/2}, S(x_{i+1/2})) \\ &= \overline{y}'_{i+1/2}-\overline{y}'_{i+1/2}=0\end{aligned} \tag{4.7}$$

needs no calculation. Similarly the residual gives zero contributions at the two end points, but the values of the residual at the four remaining quadrature points must be computed, requiring four evaluations of the right hand side functional f.

In [KS01] a similar analysis is used to show how the residual at the midpoint in bvp4c gives an approximation to how well the collocation equations (2.1) are satisfied. Such a result using values at the quarter-points $x_{i+1/4}$, $x_{i+3/4}$ is not available here because our interpolant $S(x)$ uses the altered value $\overline{y}_{i+1/2}$.

4.3 Mesh Selection

The norm of the residual computed as above is then used to update the mesh. We maintain the style of mesh selection used in bvp4c, but note that since bvp6c is a higher order method, introducing/removing points will have a greater effect on the size of the residual. Points are removed if the residual on an subinterval (and an estimate of the new residual on the coarser mesh) is deemed small enough, and added if the residual is too large. The policy in bvp4c is to remove mesh points if the predicted residual is less than half of *reltol*, the residual tolerance. We find with bvp6c this can cause trouble, with the algorithm repeatedly removing and replacing the same point on alternate iterations. This is solved by only removing mesh points when the predicted residual is less than one tenth of *reltol*. We do however follow bvp4c in introducing no more than two new mesh points at a time, and the experiments set out below suggest that this slight variation of the mesh refinement strategy proposed by [KS01] is robust enough to ensure user mandated accuracy for almost all test problems.

5 Results & Examples

We demonstrate the accuracy and efficiency of our new method, in particular comparing the results with those computed using the existing software bvp4c. We reconsider some of those problems discussed in [KS01] as well as applying a new set of problems from as the BVP problem test suite [CW91a, CW91b].

For the first family of problems we consider explicit solutions are not available, and we resort to comparing the computed solutions between the two methods. We also compare other important factors, such as solution time, number of mesh points and number of function evaluations. By defining 'reltol' and 'abstol' to some tolerance, we expect the solution of both methods, and hence their difference, to be approximately of this order.

5.1 Measles

The first example [AMR88, 1.10] models the spread of measles via the differential equations

$$y_1' = \mu - \beta(t)y_1y_3 \tag{5.1a}$$

$$y_2' = \beta(t)y_1y_3 - \frac{y_2}{\lambda} \tag{5.1b}$$

$$y_3' = \frac{y_2}{\lambda} - \frac{y_3}{\nu} \tag{5.1c}$$

where $\beta = 1575(1+\cos(2\pi t))$ and $\mu = 0.02, \lambda = 0.0279$ and $\nu = 0.01$ are given. The solution must also satisfy the (non-separated) periodicity conditions

$$y(0)=y(1). \tag{5.1d}$$

Since for the end user the use of bvp4c and 6c is identical, it is straightforward to adapt the code given in [KS01] to test both methods, and doing so results in Table 5.1.

	bvp4c			bvp6c			
tol	time	mesh	maxres	time	mesh	maxres	discrepancy
1e-3	0.384	26	2.0e-4	0.387	20	1.1e-5	1.1e-2
1e-6	0.821	63	7.0e-7	0.597	69	1.0e-7	2.5e-4
1e-9	1.503	344	9.6e-10	0.759	120	1.6e-10	1.2e-7
1e-12	7.164	2397	9.9e-13	1.429	239	9.7e-13	7.9e-11

Table 1: Comparison of the time taken, the total number of mesh points used and the maximum residual for bvp4c and bvp6c for a range of user specified tolerances for equations (5.1), the measles epidemic model of [AMR88].

What we see here shall become a consistent pattern: both methods perform similarly when only a small degree of accuracy is required, but as the tolerance is made stricter, bvp6c will find a solution using far fewer mesh points. This reduces the overall time to find a solution substantially. Quantitatively, bvp6c uses factor ten fewer mesh points for this problem, and finds a solution in less than one fifth of the time for the strictest tolerance setting.

We cannot draw any explicit conclusions as to the accuracy of the solutions here as the analytic solution is not available. Having said this, we expect the 'discrepancy' (the difference between interpolated solutions at a uniform set of points) given above to be no more than a factor of ten larger than the tolerance. By comparing with solutions obtained using a stricter tolerance, we believe bvp4c is returning the less accurate solution.

5.2 Injection

Our second example is also taken from [AMR88, 1.4] and describes flow in a long vertical channel with fluid injection:

$$f''' - R\left[(f')^2 - ff'' - A\right] = 0, \tag{5.2a}$$

$$h'' + Rfh' + 1 = ,0 \tag{5.2b}$$

$$\theta'' + Pf\theta = 0, \tag{5.2c}$$

where R is a Reynolds number and $P = 0.7R$. The parameter A is unknown, and it implemented in the call to the software as such. The eight boundary conditions are:

$$f(0) = f'(0) = 0,\ f(1) = 1,\ f'(1) = 1, \tag{5.2d}$$
$$h(0) = h(1) = 0,\ \ \theta(0) = 0,\ \ \theta(1) = 1.$$

We take $R = 100$ and obtain the results seen in Figure 1. In Table 5.2 we see that the discrepancy between the two methods behaves as expected, and for the most stringent tolerances the efficiency in the new method is even more pronounced.

	bvp4c			bvp6c			
tol	time	mesh	maxres	time	mesh	maxres	discrepancy
1e-3	0.401	25	5.8e-4	0.448	24	2.6e-5	3.0e-5
1e-6	1.284	152	9.9e-7	0.623	37	8.9e-7	3.0e-8
1e-9	7.756	910	9.9e-10	1.449	113	8.2e-10	8.0e-12
1e-12	99.37	6878	1.0e-13	3.876	358	9.9e-13	5.2e-15

Table 2: As for Table 5.1, for (5.2), the fluid injection problem of [AMR88].

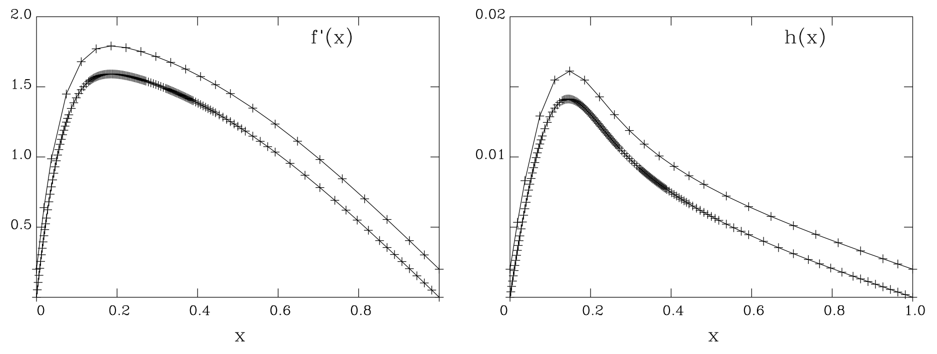

Figure 1: Solutions for (5.2), the fluid injection problem [AMR88, 1.4] solved for a tolerance of 10^{-6}. The bvp6c solutions for f' and h have been shifted upwards to aid comparison of the final solution meshes for the two algorithms.

We include the graphs of f' and h (note that one set of solutions has been shifted upwards) in Figure 1 to show that the bvp6c solution is, at least graphically, the same as that computed in [KS01], and to demonstrate the distribution of the mesh points for both methods. In this example too we see that bvp6c requires a far sparser mesh to compute the solution to the mandated accuracy. For six digits of accuracy bvp6c uses only 37 points, while bvp4c needs a mesh of 152 points. For nine digits this becomes 113 and 910 respectively, and for twelve digits of accuracy the new algorithm is a little over 25 times faster.

5.3 Shockbvp

The final BVP in this section [AMR88, 9.2] solves the singular problem (as $\varepsilon \to 0$),

$$\varepsilon y'' + xy' = -\varepsilon\pi^2 cos(\pi x) - (\pi x) sin(\pi x) \tag{5.3a}$$

with boundary conditions

$$y(-1) = -2,\ y(1) = 0. \tag{5.3b}$$

We use this example to demonstrate a number of key points, particularly the effects of

	tol=1e-3				tol=1e-6				tol=1e-9		
Final ε	1e-3	1e-4	1e-5		1e-3	1e-4	1e-5		1e-3	1e-4	1e-5
bvp4c	0.46	0.98	2.79		2.50	4.40	7.45		13.33	24.42	41.34
bvp6c	0.93	1.57	4.75		0.98	1.73	3.67		2.36	5.41	22.57
vbvp4c	0.58	1.16	3.21		2.63	4.72	8.05		14.47	26.19	42.83
vbvp6c	0.49	1.17	3.98		0.96	1.69	3.34		2.13	4.73	19.04
ajbvp4c	0.23	0.47	1.27		1.14	2.00	3.39		5.84	10.69	17.60
ajbvp6c	0.24	0.56	1.38		0.54	0.96	2.83		1.44	3.09	13.86
vjbvp4c	0.18	0.33	0.81		0.76	1.29	2.11		3.59	6.57	10.84
vjbvp6c	0.16	0.37	0.87		0.32	0.54	1.51		0.77	1.67	7.28

Table 3: Time (secs) to solve (5.3), the shock BVP [AMR88, 9.2] using continuation for various MATLAB implementations of the bvpc algorithms.

analytic Jacobians and vectorisation of the problem. Following the steps of the bvp4c analysis in [KS01], the shock problem is solved using continuation and the times shown are not the intermediate results, but the total time taken to reach a solution for the given value of ε. vbvp represents the results from a vectorised version of the problem, ajbvp from using an analytic Jacobian and vjbvp from combining the two. As a general observation on both solvers, it can be seen that vectorisation has a smaller effect on the computational time than that of analytic Jacobians, although it is when the two options are used in tandem that the time is most significantly reduced. For the weakest tolerance setting the new algorithms are the slightly slower of the two, but as the tolerance is made stricter, the improvement in bvp6c is significant. For the smallest value of ε and the strictest tolerance the gain in speed is perhaps not as large as we might expect, and we believe this to be caused by a cycle of adding and removing points unnecessarily on successive iterations as discussed in §4.3. This could perhaps be improved by tweaking the mesh adaption procedure further, but we make no attempt to do so here.

This problem demonstrates that if analytic Jacobians are available then both bvp4c and bvp6c significantly improve their performance, which we again note is in contrast to Fortran implementations of MIRK algorithms.

5.4 BVP ODE Suite

The bvp6c package has been tested extensively using the test suite [CW91a, CW91b]; a collection of linear and nonlinear BVPs designed to test the performance and robustness of a numerical BVP solver. Since a number of the problems posed do not have explicit analytic solutions, numerical solutions for these are computed using a recently developed twelfth-order MIRK method [CM06a, CM06b] to a high degree of precision and assumed sufficiently accurate for error analysis. Solutions for each of the 32 problems are then computed using both bvp4c and bvp6c with a range of specified tolerances. For simplicity we choose an initial guess of zero (where possible) on a grid of 33 equally spaced points. We do not make any attempt to vectorise the right hand side, nor do we supply an analytic Jacobian.

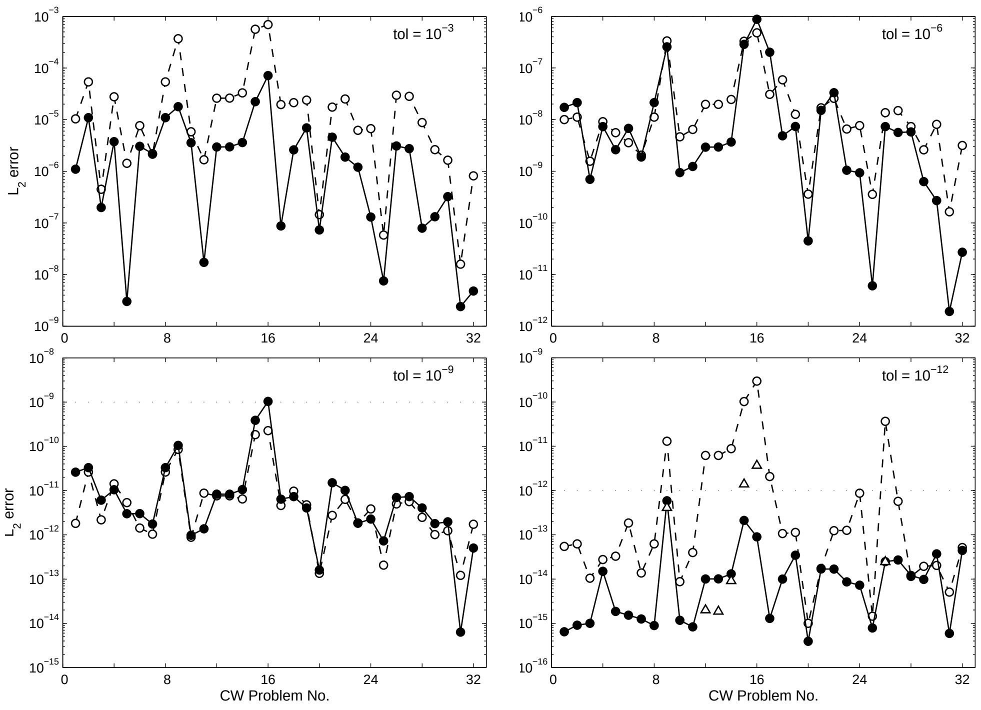

Figure 2: L_2 errors on the 32 BVP test problems [CW91b] for bvp4c (white & dashed line) and bvp6c (black & solid line) for user tolerances 10^{-3}, 10^{-6}, 10^{-9} & 10^{-12}. White triangles in Figure 2d represent bvp4c solutions with the maximum mesh size doubled.

While the 32 problems are largely unrelated, we connect the data in Figure 2 for the same method by a line to facilitate the reader's eye in seeing a consistent trend between the two sets of results, and to emphasise those few results that go against this trend. The error, defined in (5.4), is an averaged L_2 norm over each of the mesh points returned by the solver, rather than an interpolation at fixed set values.

$$\text{error} := \left(\frac{1}{N} \sum_{i=1}^{N} \left[y_i - y(x_i) \right]^2 \right)^{1/2} . \tag{5.4}$$

For all but one of the problems (and this exception being forgivably close), we see that bvp6c obtains a solution to the specified accuracy in this averaged norm. In contrast, bvp4c struggles with a number of the problems at the strictest tolerance level in Figure 2d. The triangles in this figure also represent bvp4c solutions, but where the maximum allowable mesh size has been doubled to 10,000. It is interesting to note that some problems are clearly more challenging to solve than others, and in particular two of the linear, two dimensional problems #15 and #16 appear the most troublesome.

Having demonstrated the new solver is at least comparable in terms of accuracy, we must investigate its performance in computational time. It is clear that for the lower tolerance (Figure 3a) the times for each method are similar, but for the midrange tolerances (Figures 3b & 3c) the new software is consistently around twice as fast. For the highest tested tolerance (Figure 3d) bvp6c is often faster by at least a factor of five.

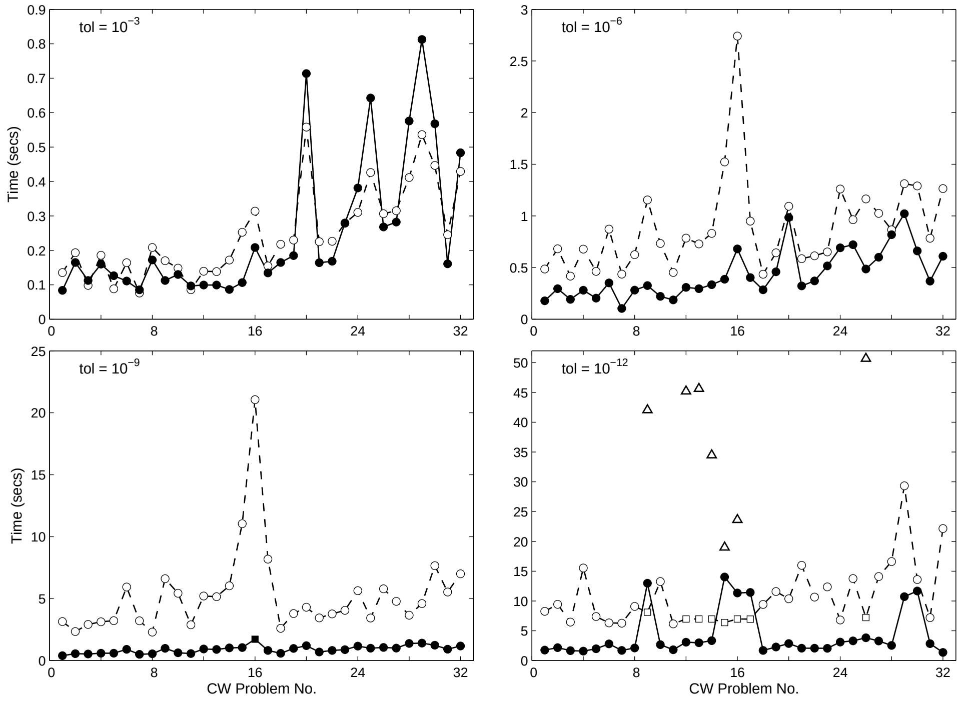

Figure 3: Times on the 32 BVP test problems [CW91b] for bvp4c (white & dashed line) and bvp6c (black & solid line) for user tolerances 10^{-3}, 10^{-6}, 10^{-9} & 10^{-12}. Squares indicate the accuracy is less than the prescribed tolerance for this problem, and triangles show bvp4c repeated with the maximum allowable mesh size doubled.

In those few problems in which bvp6c appears the the slower solver, returning to Figure 2d (and indicated by squares) one sees that these are problems for which bvp4c fails to obtain a solution of sufficient accuracy within the default maximum mesh size (squares). Further tests (triangles in Figure 3d) show that allowing a mesh large enough to reach a solution of the required accuracy with bvp4c takes significantly longer. To achieve the specified tolerance on problems #15 and #16, bvp4c requires 14,648 and 15,919 mesh points respectively, and a computational time of around two minutes.

Further experiments have shown that at strict tolerances there are few examples where the solvers supply a result of the required accuracy, but the residual is not reduced enough to terminate the algorithm. This suggests the residual is sometimes too overestimated, and a significantly example of this is solving problem #32 with bvp6c. Using default settings, the residual is not reduced below the tolerance before the maximum number of mesh points (500) is exceeded, which takes around 30 seconds. By reducing this maximum to only 200 points, a solution accurate to 13 digits is obtained in less than one second. This is the result given in Figures 2 and 3.

6 Conclusions

Extensive testing, particularly in §5.4, has shown that whilst not always providing the more accurate of the solutions, bvp6c consistently obtained the accuracy specified by the user. A meaningful comparison of accuracy between the two bvpc packages is difficult to quantify, since the design of the framework means that both solvers continue to improve their solutions until the residual is sufficiently reduced.

We have shown that locally the bvp6c residual is asymptotically sixth- and the bvp4c fourth-order accurate, but this type of analysis is relevant only when considering solutions obtained on the same mesh. However, since the residual determines the adaptive mesh strategy, we expect and demonstrate that the higher order finite difference formula of [CS82] and interpolation formula of [CM04] obtain a comparable reduction in the global residual with fewer internal mesh points, and correspondingly takes less time to compute. We conclude that whilst both methods generally give solutions satisfying the required tolerance, the new method does so more efficiently.

This can be further seen in a comparison of solution times for the bvp4c solver with a tolerance of 10^{-6} with those from bvp6c at 10^{-9}. Figure 4 show these are almost identical; meaning that in the same time it takes bvp4c to solve to an accuracy of 10^{-6}, bvp6c can solve the same problem to an accuracy of 10^{-9}. Such a result holds to an even greater extent for tolerances of 10^{-9} and 10^{-12}.

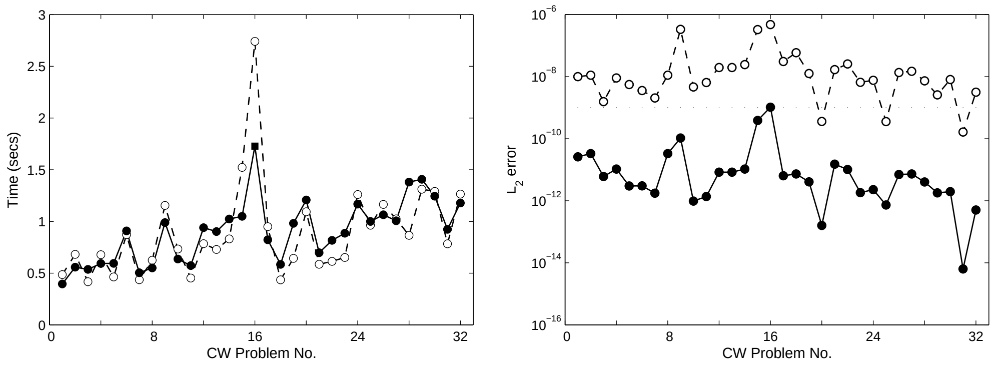

Figure 4: Solution times and L_2 errors for bvp4c with a tolerance of 10^{-6} (white & dashed line) and bvp6c with a tolerance of 10^{-9} (black & solid line).

In view of the apparent success of the bvp6c extension to the existing MATLAB software, it is tempting to consider a further development using eighth [CS82], tenth and even twelfth-order [CM06b] difference methods. However, the eighth-order scheme requires a total of seven interior points per interval, and the form of the Jacobians in §4.1 becomes increasingly complex. It seems likely the improvement in local accuracy of an even higher order scheme may come at the expense of an unjustifiable increase in the time needed to find a global solution of the required accuracy.

An area where we do see potential for the enhancement of the bvpc framework is in solving more efficiently problems of the form $y''(x) = f(x, y)$ and $y'' = f(x, y, y')$. Again in [CM04] there is a sixth-order Hermite-Birkhoff interpolant to fit data produced by a finite difference scheme corresponding to these forms of problems, and in [CCM06] eighth-order accurate algorithms for problems of this form are set out. It remains only to determine a suitable representation of the residual computable on each interval for this important class of boundary value problems.

7 Additional Comments

Since the development of bvp6c, the BVP solving capabilities of MATLAB have been augmented with a new package bvp5c, also written by Kierzenka and Shampine [KS08]. Whereas the error control in bvp4c and bvp6c is based on the residual $r(x) = S'(x) - f(x, S(x))$, bvp5c uses a fifth-order, four-point Lobatto IIIA formula with an error estimate combining both $r(x)$ and the true error $e(x) = S(x) - y(x)$. K&S report that in numerical experiments the speed of bvp5c is comparable to the speed of bvp4c for low-to-medium tolerance, and for stringent tolerances the advantage of bvp5c is significant. We forgo a detailed comparison of bvp6c with bvp5c, but offer in Figure 5 a variant of Figures 2c, 2d, 3c & 3d, replacing bvp4c with bvp5c.

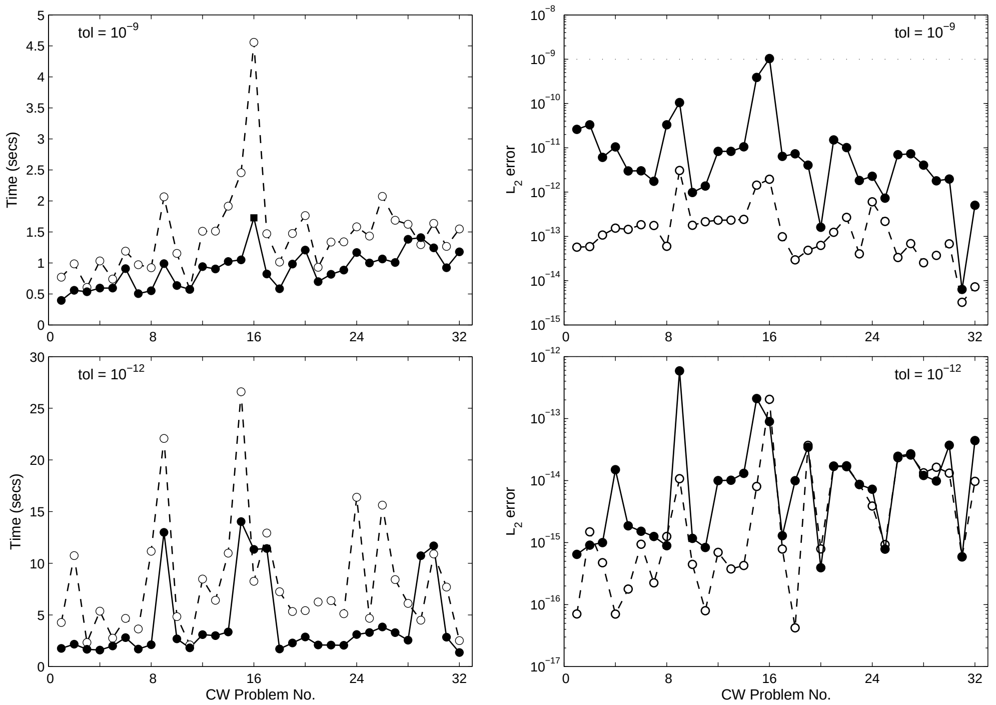

Figure 5: Solution times (left) and L$_2$ errors (right) from solving the 32 problems of §5.4 with bvp5c (white & dashed line) and bvp6c (black & solid line) using tolerances of 10^{-9} (top) and 10^{-12} (bottom).

All results in this paper were computed using MATLAB 7.5 on a dual core 1.7Ghz laptop. The parameters used for the 32 problems in §5.4 are 0.001, 0.01, 0.05, 0.025, 0.01, 0.022, 0.025, 0.01, 0.055, 0.022, 0.001, 0.0025, 0.0025, 0.0025, 0.005, 1/19, 0.0005, 0.013, 0.03, 0.05, 0.0008, 0.025, 5, 0.03, 0.0025, 0.02, 0.02, 0.03, 0.015, 0.042, 0.025 and 100 respectively.

The bvp6c package and the examples of §5 are freely available online on both the MATLAB Central file exchange at http://www.mathworks.com, and on NH's website http://www.comlab.ox.ac.uk/nick.hale/bvp6c.

Finally, we wish to thank Steven Capper for pointing out the more economical form for the Jacobians in the [CS82] MIRK6 scheme, and NH wishes to thank the Nuffield Foundation for an Undergraduate Research Bursary for 2005.

References

[AMR88] U. M. Ascher, R. M. R. Mattheij, and R. D. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations.* Prentice–Hall, Englewood Cliffs, NJ, USA, 1988.

[CCM06] J. R. Cash, S. D. Capper, and D. R. Moore. Lobatto-Obrechkoff formulae for 2nd order two-point boundary value problems. *Journal of Numerical Analysis, Industrial and Applied Mathematics*, 1:13–25, 2006.

[CGM02] J. R. Cash, M. P. Garcia, and D. R. Moore. Mono-implicit runge-kutta formulae for the numerical solution of second order nonlinear two-point boundary value problems. *Journal of Computational and Applied Mathematics*, 143(2):275–289, 2002.

[CM04] J. R. Cash and D. R. Moore. High-order interpolants for solutions of two-point boundary value problems using MIRK methods. *Computers and Mathematics with Applications*, 48(10–11):1749–1763, 2004.

[CM06a] S. D. Capper and D. R. Moore. NewNRK a Fortran 90 code for solving two-point boundary value problems, 2006.

[CM06b] S. D. Capper and D. R. Moore. On high order MIRK schemes and Hermite-Birkhoff interpolants. *Journal of Numerical Analysis, Industrial and Applied Mathematics*, 1:27–47, 2006.

[CS82] J. R. Cash and A. Singhal. High order methods for the numerical solution of two-point boundary value problems. *Behaviour & Information Technology*, 22:184–199, 1982.

[CW91a] J. R. Cash and M. H. Wright. A deferred correction method for nonlinear two-point boundary value problems: implementation and numerical evaluation. *SIAM Journal of Scientific and Statistical Computing*, 12(4):971–989, 1991.

[CW91b] J. R. Cash and M. H. Wright. Thirty–two test problems for analysis of two-point boundary value problem solvers, 1991. http://www.ma.ic.ac.uk/~jcash/BVP_software/PROBLEMS.PDF.

[DC01] M. Van Daele and J. R. Cash. Superconvergent deferred correction methods for first order systems of nonlinear two–point boundary value problems. *SIAM Journal of Scientific Computing*, 22(5):1697–1716, 2001.

[EM86] W. H. Enright and P. H. Muir. Efficient classes of Runge–Kutta methods for two–point boundary value problems. *Computing*, 37:315–334, 1986.

[EM96] W. H. Enright and P. H. Muir. Runge–Kutta software with defect control for boundary value ODEs. *SISSC*, 2:479–497, 1996.

[KS01] J. Kierzenka and L. F. Shampine. A BVP solver based on residual control and the MATLAB pse. *ACM Transactions on Mathematical Software*, 27(3):299–316, 2001.

[KS08] J. Kierzenka and L. F. Shampine. A BVP solver that controls residual and error. http://faculty.smu.edu/shampine/finalbvp5c.pdf, 2008.

[Mic63] H. H. Michels. Abscissas and weight coefficients for Lobatto quadrature. *Mathematics of Computation*, 17(83):237–244, 1963.