Improved Implementation of Multiple Shooting for BVPs

Weidong Zhang
University of Toronto Computer Science Department

February 2, 2012

Abstract

Boundary value problems arise in many applications, and shooting methods are one approach to approximate the solution of such problems. A Shooting method transforms a boundary value problem into a sequence of initial value problems, and takes the advantage of the speed and adaptivity of initial value problem solvers. The implementation of continuous Runge-Kutta methods with defect control for initial value problems gives efficient and reliable solutions. In this paper, we design and implement a boundary value solver that is based on a shooting method using a continuous Runge-Kutta method to solve the associated initial value problems. Numerical tests on a selection of problems show that this approach achieves better performance than another widely used existing shooting method.

1 Introduction

Applications of boundary value problems (BVPs) arise in many different areas - see [1], section 1.2. Consider a boundary value problem defined by the system of ordinary differential equations (ODEs)

$$y' = f(t, y),\ g(y(a), y(b)) = 0 \tag{1}$$

where $t \in [a, b], y : R^n, f : R \times R^n \to R^n$, and $g : R^n \times R^n \to R^n$.

One approach for solving BVPs is to use a shooting method, which replaces a given BVP by one (simple shooting) or more (multiple shooting) initial value problems (IVPs). The idea of multiple shooting was first proposed by Morrison et al. [2], later popularized by Keller [3], who developed and analyzed both a simple shooting method (SSM) and a multiple shooting method (MSM). A more recent version of a multiple shooting method, MUSN, was developed by R. M. M. Matteij and G.W.M Staarink [4] (the latest version is available online through NETLIB [5]).

In this paper, we construct a new BVP solver which combines the shooting approach with a recently introduced class of continuous Runge-Kutta(CRK) IVP

solvers for the solution of BVPs. This new BVP solver generally converges faster, is more accurate and provides more robust solutions than the multiple shooting solver MUSN.

The existence and uniqueness theory for BVPs is considerably more difficult than it is for IVPs. Necessary and sufficient conditions for the existence and uniqueness of a solution of (1) can be found in [1], section 3.1. But the kind of conditions under which the BVP may have multiple solutions is far from clear, and such difficulties can arise in realistic problems in applications. So for a given BVP, there maybe multiple solutions. This is the case we will encounter in our test problems later in this paper. There are other general purpose methods for solving BVPs. One of the most popular methods, and the one that we will be using to benchmark the performance of our shooting method is a collocation method, COLNEW [6].

In section 2, we briefly review shooting methods. We introduce CRK IVP solvers in section 3, which we then use to implement a particular shooting method. We combine the shooting approach and a CRK method to create a prototype model of a new BVP solver. Some implementation issues that arise from this combination will be discussed in section 4. We report several test results in section 5. From the results of tests, we have some conclusions and observations which are presented in section 6.

2 Shooting Methods

Shooting methods transform a BVP to a sequence of IVPs by attempting to find the right initial conditions which lead to an approximate solution of the IVP that satisfies the boundary conditions. They take advantage of the speed and adaptivity of IVP methods. But they also inherit the stability (or instability) of the associated IVP, which may be unstable even if the BVP itself may be quite stable.

When applied to (1), shooting methods look for initial conditions $y(a) = s$, so that the solution $u(t)$ of the resulting IVP satisfies $g(s, u(b)) = 0$.

2.1 Simple Shooting

A simple shooting method applied to (1) introduces the associated IVP:

$$y' = f(t, y),\ y(a) = s,\ a \leq t \leq b \tag{2}$$

where s is a prescribed initial vector. If we denote $y(t; s)$ as the solution of (2), then problem (1) is reduced to finding a solution $s = s^*$ of the nonlinear system of equations,

$$g(s^*, y(b; s^*)) = 0. \tag{3}$$

This problem is solved iteratively, where on each iteration, we must evaluate $g(s, y(b; s))$ for some s and this involves the solution of IVP (2), integrated from $t = a$ to $t = b$ (as Figure 1 illustrates).

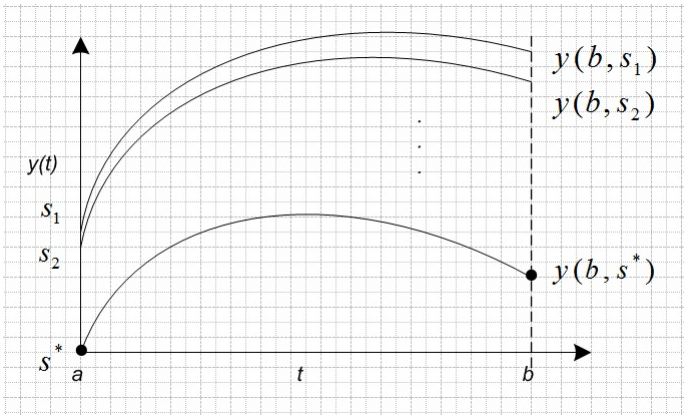


Figure 1: Representation of the iteration associated with simple shooting method

Although simple shooting is straightforward, there can be serious difficulties with the IVPs integrated in the method. One trouble is that the IVPs might be unstable, even when the BVP is well-conditioned. Another is that the solution of the IVP may not exist over the whole interval for a given s. These troubles can often be addressed by implementing multiple shooting.

2.2 Multiple Shooting

A multiple shooting method tries to resolve the difficulties arising with simple shooting methods by dividing the interval $[a, b]$ into a mesh of N subintervals

$$a = x_1 < x_2 < \cdots < x_N < x_{N+1} = b, \tag{4}$$

and replacing the unknown vector $s = y(a)$ by a set of unknown vectors $s_i \approx y(x_i)$, $1 \leq i \leq N$. The initial value integrations are performed (possibly in parallel) on each subinterval $[x_i, x_{i+1}]$ (as Figure 2 illustrates).

A multiple shooting method applied to (1) introduces N associated systems of IVPs for $1 \leq i \leq N$

$$y' = f(t, y),\ y(x_i) = s_i,\ x_i \leq t \leq x_{i+1}. \tag{5}$$

If we denote $y_i(t; s_i)$ as the solution of (5), then there are $n \times N$ unknown parameters

$$s^T = \{s_1^T, s_2^T, \ldots, s_N^T\} \tag{6}$$

to be determined, so that the solution is continuous over the entire interval $[a, b]$, and satisfies the boundary conditions of (1).

There are $N - 1$ additional matching conditions (the continuity constraints)

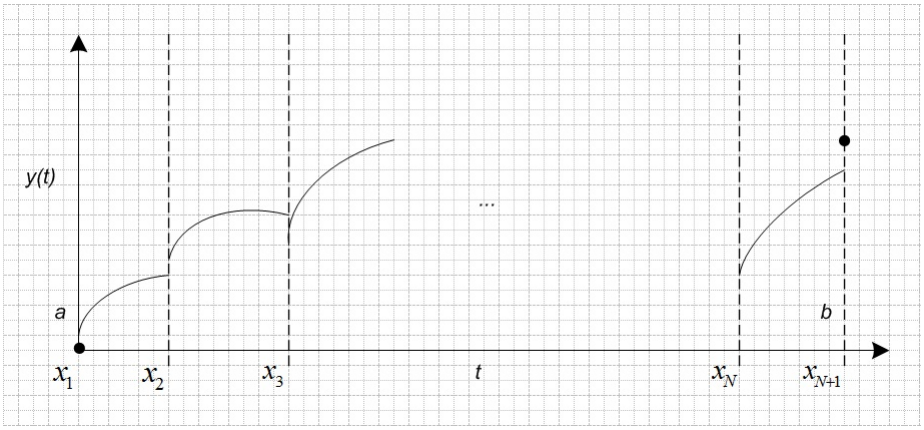


Figure 2: Visualization of one iteration of a multiple shooting method

added to the nonlinear equation (3) for finding a solution $s = s^*$ of,

$$F(s^*) = \begin{pmatrix} s_2^* - y_1(x_2; s_1^*) \\ s_3^* - y_2(x_3; s_2^*) \\ \vdots \\ s_N^* - y_{N-1}(x_N; s_{N-1}^*) \\ g(s_1^*, y_N(b; s_N^*)) \end{pmatrix} = 0. \tag{7}$$

Note that the dimension of $F(s)$ is nN.

When implementing a shooting method, the partitioning of the mesh (4) is often determined adaptively to cope with numerical instabilities that arise when solving the associated IVPs. These difficulties can arise when the associated IVPs are stiff. This implies that N is affected by the stiffness of the IVPs associated with the BVP. The quality of the integration on each subinterval $[x_i, x_{i+1}], 1 \leq i \leq N$ is determined by the chosen IVP solver. The IVP solver determines its own IVP mesh

$$x_i = t_1^i < t_2^i < \cdots < t_{M_i}^i < t_{M_{i+1}}^i = x_{i+1}. \tag{8}$$

The total number of mesh points associated with all the IVPs solved on each iteration is $M = \sum_{i=1}^{N} M_i + 1$.

3 Continuous Runge-Kutta Methods

Continuous Runge-Kutta methods were first introduced for practical problems such as graphic output, which require dense output to reveal details of an approximate solution - see [7], chapter II.6.

An s-stage, explicit pth-order discrete Runge-Kutta formula applied to (5) determines,

$$y_j^i = s_i, \;\; y_{j+1}^i = y_j^i + h_j^i \sum_{r=1}^{s} \omega_r k_r^i \tag{9}$$

where $h_j^i = t_{j+1}^i - t_j^i,\ 1 \le j \le M_i$, and

$$k_r^i = f\left(t_j^i + c_r h_j^i, y_j^i + h_j^i \sum_{q=1}^{r-1} a_{rq} k_q^i\right).$$

The standard error control mechanism of a discrete Runge-Kutta method applied to (5) can be described by introducing the local error associated with each step (9).

Let y_j^i be defined by (9) for $j = 1, 2, \ldots, M_i$, and let $z_{ij}(t)$ be the solution of the local IVP,

$$z'_{ij} = f(t, z_{ij}),\ z_{ij}(t_j^i) = y_j^i,\ t \in [t_j^i, t_{j+1}^i], \qquad (10)$$
$$1 \le j \le M_i,\ 1 \le i \le N.$$

The method attempts to ensure that the local error per unit step is bounded by a user specified tolerance TOL on each interval $[t_j^i, t_{j+1}^i]$. That is, the method attempts to ensure

$$\|z_{ij}(t_{j+1}^i) - y_{j+1}^i\| \le TOL(t_{j+1}^i - t_j^i).$$

To make a multiple shooting method efficient, both M and N should be kept as small as possible.

A continuous Runge-Kutta method extends the discrete formula (9) by adding $(\bar{s} - s)$ additional stages to obtain an accurate approximation for any $t \in [t_j^i, t_{j+1}^i]$ (see [8] for details),

$$u_{ij}(t) = y_j^i + h_j^i \sum_{q=1}^{\bar{s}} b_q(\tau) k_q^i = z_{ij}(t) + O((h_j^i)^{p+1}) \qquad (11)$$

where $\tau = \frac{t - t_j^i}{h_j^i}$, and $b_q(\tau)$ is a polynomial of degree at most $p + 1$,

$$b_q(\tau) = \sum_{r=0}^{p+1} \beta_{qr} \tau^r.$$

One can analyze the error in (11) by considering the local interpolant $u_{ij}(t)$ to be an approximation to the local solution $z_{ij}(t)$ for $t \in [t_j^i, t_{j+1}^i]$.

This polynomial interpolant can be written as

$$\begin{aligned} u_{ij}(t) \quad = \quad & d_0(\tau) y_j^i + h_j^i d_1(\tau) f(x_j^i, y_j^i) + d_2(\tau) y_{j+1}^i + \\ & h_j^i d_3(\tau) f(x_{j+1}^i, y_{j+1}^i) + h_j^i \textstyle\sum_{q=1}^{\bar{s}-s} d_{q+3}(\tau) k_{s+q}^i \end{aligned} \qquad (12)$$

where d_q is a polynomial degree $\le p$. The polynomial $u_{ij}(t)$ satisfies $u_{ij}(t) = z_{ij}(t) + O((h_j^i)^p)$ for $t \in (t_j^i, t_{j+1}^i)$. The polynomials $\{u_{ij}(t)\},\ i = 1, 2, \ldots, N,\ j = 1, 2, \ldots, M_i$ then define a vector of piecewise polynomials $u(t)$, which are continuous on $[a, b]$. And a simple set of constraints on $b_q(\tau)$ will ensure that $u_{ij}(t)$ interpolate y_j^i, y_{j+1}^i, so that the discrete Runge-Kutta method is embedded within the CRK.

This approach allows one to decompose the error in $u_{ij}(t)$ into two components: the error inherent in polynomial interpolation (the local interpolation error) and the error that arises as a consequence of "inexact" values being interpolated (the data error associated with the fact that we are interpolating approximate solution and derivative values).

The piecewise polynomial $u(t)$ allows an alternative error control mechanism for continuous Runge-Kutta methods, defect control, which is different from the local error control discussed earlier. The defect $\delta(t)$ associated with $u(t)$ is defined for $t \in [a, b]$ to be,

$$\delta(t) \equiv u'(t) - f(t, u(t)). \tag{13}$$

That is, $\delta(t)$ is the amount by which the associated piecewise polynomial fails to satisfy the differential equation. With an interpolation scheme defined by (11), one can show that the corresponding defect satisfies

$$\delta(t) = \psi_{p+1}(\tau)h^{p+1} + O(h^{p+2}) \tag{14}$$

with $h = \max_{i,j} h_j^i$ and $\psi_{p+1}(\tau)$, independent of h, satisfies

$$\psi_{p+1}(\tau) = q_1(\tau)F_1 + q_2(\tau)F_2 + \cdots + q_L(\tau)F_L.$$

The $F_j,\ 1 \le j \le L$ are elementary differentials evaluated at (x_i, y_i), and the q_j are polynomials of degree $\le p + 1$.

The additional $\bar{s} - s$ stages and the polynomial coefficients β_{qr} are not uniquely determined by the discrete formula (9), and different criteria can be used to identify promising interpolation schemes. The challenge of defect control is to find an efficient way to reliably estimate the maximum defect on each subinterval $t \in [t_j^i, t_{j+1}^i],\ 1 \le j \le M_i$.

There are two alternative promising defect estimation strategies, both of which apply to interpolation schemes satisfying (11). One non-asymptotically justified approach is to sample defects at one or more carefully selected points on each subinterval, and hope that the 'true' maximum value will not be much larger than the maximum sampled value. The number of samples has to be small in order to be efficient. And the points should be chosen in a careful way, not near the roots of the polynomials $q_j,\ 1 \le j \le L$ (see [8] for details). This defect control strategy is referred as relaxed defect control (RDC). RDC works well on most problems, but sometimes it can severely underestimate true maximum defect.

A more rigorous and asymptotically (as $h \to 0$) justified defect estimation strategy is referred as strict defect control (SDC) (the detailed process of deriving SDC CRKs can be found in [9]). SDC methods will usually require more additional stages (larger value for $\bar{s}$) and in our tests we have not found the RDC strategy to perform much worse than SDC, so we use RDC methods in this paper.

In implementation CRK formula, explicit CRKs are more straightforward than implicit CRKs, because there are no nonlinear equations to be solved and the cost of continuous extension is only the additional $\bar{s} - s$ function evaluations on each step. Several order p explicit CRK formulas have been investigated.

Formula	p	s	$\bar{s}$
CRK5	5	6	9
CRK6	6	7	11
CRK8	8	13	21

Table 1: Cost per step of the RDC explicit CRK formulas we have considered

The result of implementing and testing of some explicit CRKs with both RDC and SDC on 25 standard non-stiff problems of the DETEST package can be found in [10]. We have focused on RDC CRKs in our multiple shooting code as the cost per step for the CRK is roughly 75% of that for the SDC CRKs and there is little difference in the errors of the interpolants. Note that continuous Runge-Kutta methods can also be applied directly to a BVP in a different way (see [11] for details).

4 Implementation

Both simple shooting and multiple shooting methods need to solve nonlinear equations, either (3) or (7). We use a modified damped Newton method to solve these systems.

For simple shooting, the Jacobian matrix of this nonlinear system is an $n \times n$ matrix defined as,

$$\frac{\partial g}{\partial s} = \frac{g(s, y(b,s))}{\partial s} + \frac{g(s, y(b,s))}{\partial y(b,s)} Y(b), \tag{15}$$

where $Y(t) = \frac{\partial y(t,s)}{\partial s}$ is the $n \times n$ fundamental matrix which is the solution of the matrix IVP,

$$Y' = \frac{\partial f(t, y(t,s))}{\partial y} Y,\ Y(a) = I,\ a \le t \le b.$$

For multiple shooting, the Jacobian matrix of the associated nonlinear system is an $nN \times nN$ block bi-diagonal matrix,

$$\frac{\partial F}{\partial s} = \begin{pmatrix} -Y_1(x_2) & I & 0 & \cdots & & 0 \\ 0 & -Y_2(x_3) & I & 0 & \cdot & 0 \\ \vdots & 0 & & \cdots & & \vdots \\ & & & & -Y_{N-1}(x_N) & I \\ \frac{g(s,y(b,s))}{\partial s} & 0 & \cdots & & 0 & \frac{g(s,y(b,s))}{\partial y(b,s)} Y_N(b) \end{pmatrix} \tag{16}$$

where $Y_i(t) = \frac{\partial y_i(t,s_i)}{\partial s_i}$ is an $n \times n$ fundamental solution associated with the ith subinterval defined by the IVP,

$$Y_i' = \frac{\partial f(t, y(t,s_i))}{\partial y} Y_i,\ Y_i(x_i) = I,\ x_i \le t \le x_{i+1},\ 1 \le i \le N$$

The use of a damped Newton method in solving BVPs is discussed in [1], section 8.1, where it is shown that convergence of Newton's method can be improved. A damped Newton method uses a parameter λ to control the magnitude of step to be taken in the Newton direction,

$$s_{m+1} = s_m - \lambda \left(F'(s_m)\right)^{-1} F(s_m),\ 0 < \lambda \leq 1, \tag{17}$$

where $F'(s_m) = \frac{\partial F}{\partial s}\big|_{s=s_m}$. Newton's method corresponds to taking $\lambda = 1$.

Let $\Delta^m = -\left(F'(s_m)\right)^{-1} F(s_m)$, the Newton step on the $(m+1)^{\text{st}}$ iteration. For any $s \in R^{N\times m}$ define, on the $(m+1)^{\text{st}}$ iteration, an objective function

$$\hat{g}_m(s) = \frac{1}{2}\left\|\left(F'(s_m)\right)^{-1} F(s)\right\|_2. \tag{18}$$

We follow the algorithm introduced in [1] to determine the acceptable $\lambda \in [0.01, 1]$ on each iteration. An overview of the algorithm in presented in Figure 3 where λ^r is considered acceptable when

$$\hat{g}_m(s_m + \lambda^r \Delta^m) \leq (1 - 2\lambda^r\sigma)\hat{g}_m(s_m),\ \sigma = 0.01.$$

Note that $\hat{g}_m(s_m) = \frac{1}{2}\|\Delta^m\|_2$, and requires very little computation, whereas for $s \neq s_m$ an evaluation of $\hat{g}_m(s)$ requires the solution of a linear system plus the computation of $F(s)$. See [1] for a discussion and justification this technique for determining an acceptable value for λ.

$r = 1$

$$\lambda^r = \begin{cases} 1 & \text{for } m = 1 \\ \lambda_{m-1} & \text{if } \lambda_{m-1} < \lambda_{m-2}(1-\sigma) \\ \min(1, 2\lambda_{m-1}) & \text{otherwise} \end{cases}$$

do

until $\lambda^r < 0.01$ or $\hat{g}_m(s_m + \lambda^r\Delta^m) \leq (1 - 2\sigma\lambda^r)\hat{g}_m(s_m)$

$\lambda^{r+1} = \max\left(\tau\lambda, \frac{\lambda^2 \hat{g}_m(s_m)}{(2\lambda^r - 1)\hat{g}_m(s_m) + \hat{g}_m(s_m + \lambda^m\Delta^m)}\right)$

$r = r + 1$

end do

if $\lambda^r < 0.01$ then

signed no acceptable λ

else

$\lambda_m = \lambda^r$

end

Figure 3: An overview of the algorithm used to determine an acceptable $\lambda = \lambda_m$ on the each iteration

The objective function not only provides an indication of convergence, but also can be used to improve the initial guess and restart the iteration. From user provided initial guess s_0, for any iterate $s_i,\ i > 0$, with $\hat{g}(s_i) < \hat{g}(s_0)$, we assume this

indicates that s_i is closer to the solution than s_0. Since Newton's method is sensitive to the choice of initial guess, and converges to the solution very rapidly once the initial guess is close to a solution, by monitoring the value of $\hat{g}(s_i)$ on each iteration, we can replace the user provided initial guess s_0 by a better initial guess s_i when a restart is indicated. The modified Newton iteration needs to be restarted when certain events happen, such as when new mesh point(s) need to be inserted or divergence of $\hat{g}(s_m)$ is detected. Then the s_i corresponding to the residual solve of $\hat{g}(s_i)$ derived so far will be the initial guess for the restarted iteration. We hope with the better initial guess, we improve the chance of convergence.

The block matrix $Y_i(x_{i+1}), 1 \leq i \leq N$ in (16) can be interpreted as a local sensitivity matrix (see detail in [12]). We apply a QR-decomposition $Y_i(x_{i+1}) = Q_i R_i$. Let $r_{ii},\ 1 \leq i \leq n$ denote the diagonal entries of R_i, we use the ratio of

$$\frac{\max(|r_{ii}|)}{\min(|r_{ii}|)} \tag{19}$$

as a crude condition number estimator of the associated IVP (also see [13] for a similar scheme). This condition number estimator can indicate stiffness at $t \in [x_i, x_{i+1}]$. If the condition number is large, then we insert an additional mesh point in the middle of $[x_i, x_{i+1}]$. This mechanism can be used to determine the number of mesh points automatically without requiring a user to provide the initial mesh.

We set the maximum number of iteration to 30. If after 30 iterations the result is still not able to satisfy the given tolerance, then we double the number of mesh points and try again. And we also set the maximum number of mesh points to 1000. At any time, if the program tries to increase the number of mesh points greater than 1000, then we assume the given problem is too difficult for the multiple shooting method, and the method will exit and signal a failure.

5 Numerical Tests

We report on some numerical tests to illustrate the performance of our new BVP solver (denoted as MUSCRK). We use, as test problems, BVPs that depend on a single parameter. In most cases, as the parameter changes, the problems change from non-stiff to stiff. In this way, we can measure both performance and the range of stiffness where the solver can be effective.

We use RDC CRK78 as the CRK IVP solver [14]. For comparison purpose, we also report the performance of two other BVP solvers: COLNEW and MUSN. As noted earlier, COLNEW is a popular BVP solver based on collocation. It can be applied to a wide range of problems from non-stiff to stiff. In our testing, we set the order to 8, matching the order of CRK78. MUSN is a multiple shooting BVP solver discussed earlier. The IVP solver of MUSN is based on a Fehlberg 45 Runge-Kutta method, which is a lower order method than MUSCRK or COLNEW.

We consider several test problems and report results for MUSCRK, COLNEW, and MUSN. These problems are subjected to two different tolerance 10^{-3} and

10^{-6}, and we compare the results based on number of mesh point(s), the number of iteration(s) (which are reported as (N, m), where N is the number of mesh points, and m is the number of iterations), execution time, maximum defect (if applicable), and maximum error. If the reported number of mesh point(s), and number of iteration(s) appear as $(N, *)$, then this means there is no convergence with N mesh points. And if $(N_1, m_1), (N_2, m_2), \ldots$ appear, this indicate that although with N_1 mesh points after m_1 iterations, the method converges to a solution, the associated error estimation does not satisfy the given tolerance, and further smooth refinement is needed.

Here, the number of mesh points N has different meanings for BVP solvers based on shooting (MUSCRK and MUSN), and the collocation method (COLNEW). For BVP solvers based on shooting, N is intended to control the instability of the IVPs (as discussed in the end of section 2), and is normally insensitive to the value of TOL. For a BVP solver based on collocation, N determines the underlying discretization step, and must increases as TOL decreases.

The execution time measurement is measured on a dell studio 1558 laptop running Ubuntu 11.04. Each of the three methods solves a given test problem three times, and we report the average time. When determining the timing result, the program does not compute any of the other reported statistics.

MUSCRK uses a defect control mechanism on every step, so we can report the maximum defect on the whole interval for each test problem. But for the other two BVP solvers, defect control is not the default error control mechanism, they do not have the defect control feature, so we only report maximum defect for MUSCRK.

For each of our test problem, there is no known explicit analytic solution. We subjected these BVPs to a very severe tolerance (usually 10^{-10}), and use the result as the reference solution to the problem. Concerning the maximum error, MUSN only produces the approximate solution at mesh points, so for MUSN, maximum error is the maximum difference between the reference solution and computed solution at mesh points. Since both MUSCRK and COLNEW can produce approximate solution between mesh points, we divide the intervals of the test problems into 100 subintervals, and get the approximate solutions on these 100 points, and we use the maximum difference between the reference solution and approximate solution over these points as the maximum error for both MUSCRK and COLNEW.

We have chosen four test problems from the boundary value literature. Each depends on a parameter and we have generated results for four parameter values for each problem. If for a given parameter value, the BVP method cannot converge to a solution, we leave the column blank in the table associated with the method.

5.1 Plasma Confinement

$$y'' = \tau \sinh(\tau y), t \in [0, 1]$$

subject to boundary conditions

$$y(0) = 0,\ y(1) = 1$$

and parameter values $\tau = 1, 7, 10, 16$. The default initial guess is

$$y_1(t) = 0,\ y_2(t) = 0.$$

Refer to Figure 4 for a plot of the solutions and Tables 2 and 3 for a summary of the results for the 3 methods. This problem is also known as Troesch's equation. Increasing τ increases the stiffness of this ODE (taken from [13]).

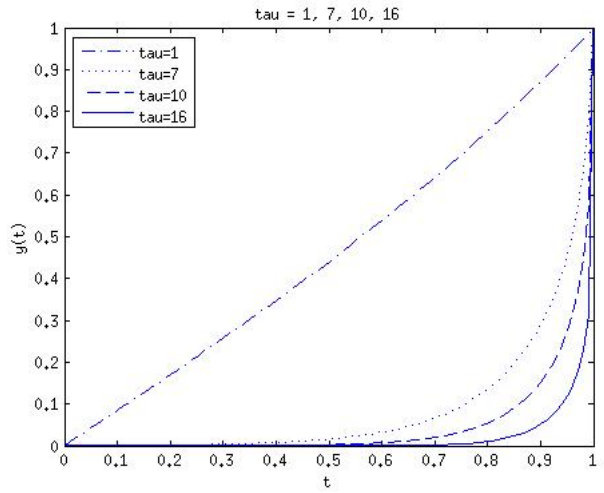


Figure 4: Solutions of plasma confinement problem for 4 values of τ

5.2 Swirling Flow III

$$\begin{aligned} \epsilon f'''' + f f''' + g g' &= 0 \\ \epsilon g'' + f g' - f' g &= 0 \end{aligned}$$

subject to boundary conditions

$$f(0) = 0, f'(0) = 0, g(0) = 1; f(1) = 0, f'(1) = 0, g(1) = -1$$

and parameter values $\epsilon = 1.0, 0.05, 0.005, 0.001$, and initial guess

$$y_1(t) = 0, y_2(t) = 0, y_3(t) = 0, y_4(t) = 0, y_5(t) = 2 * t - 1, y_6(t) = 2$$

This problem is taken from [1]. Refer to Figure 5 for a plot of the solutions and Tables 4 and 5 for a summary of the results for the 3 methods.

MUSCRK and COLNEW produce different solutions when $\varepsilon = 0.001$ as Figure 6 shows, MUSCRK produces a symmetric solution, while COLNEW produces a non-symmetric solution. Are both actual solutions to the problem? We used a reliable stiff IVP solver to verify both solutions. That is we applied the IVP solver to the differential equation with the initial condition associated with the value of

method	result				
MUSCRK	$\tau \Rightarrow$	1	7	10	16
	Profile	(1, 3)	(1, *) (2, *) (3, *) (4, *) (5, *) (6, *) (7, 11)	(1, *) (2, *) (3, *) (4, *) (5, *) (6, *) (8, *) (10, *) (13, *) (17, *) (20, 12)	(1, *) (2, *) (3, *) (4, *) (5, *) (6, *) (8, *) (11, *) (16, *) (23, *) (34, *)	(52, *) (78, *) (115, *) (161, *) (202, *) (212, *) (224, *) (240, *) (255, *) (259, *) (273, 20)
	Time(sec)	0.003	0.023	0.054	3.992	
	Error	2.01×10^{-4}	1.25×10^{-2}	1.41×10^{-2}	0.161	
	Defect	8.89×10^{-7}	6.53×10^{-4}	1.96×10^{-4}	2.01×10^{-4}	
COLNEW	$\tau \Rightarrow$	1	7	10	16	
	Profile	(5, 2) (10, 1)	(5, 9) (3, 1) (6, 1) (4, 1) (8, 1) (16, 1)	(5, 10) (3, 2) (6, 2) (4, 1) (8, 1) (5, 1) (10, 1) (5, 1) (10, 1)	(5, 14) (4, 5) (8, 5) (5, 5) (10, 3) (6, 2) (12, 2) (7, 1) (14, 1) (28, 1)	
	Time(sec)	0.004	0.007	0.009	0.016	
	Error	9.03×10^{-9}	4.73×10^{-4}	1.40×10^{-2}	0.166	
MUSN	$\tau \Rightarrow$	1	7	10	16	
	Profile	(5, 2)	(5, *) (10, *) (20, *) (40, 9)			
	Time(sec)	0.004	0.018			
	Error	1.33×10^{-6}	2.37×10^{-5}			

Table 2: Results for plasma confinement problem with $TOL = 10^{-3}$, and 4 values of τ

method	result				
MUSCRK	$\tau \Rightarrow$	1	7	10	16
	Profile	(1, 5)	(1, *) (2, *) (3, *) (4, *) (5, *) (6, *) (7, 11)	(1, *) (2, *) (3, *) (4, *) (5, *) (6, *) (8, *) (10, *) (13, *) (17, *) (20, 16)	(1, *) (2, *) (3, *) (4, *) (5, *) (6, *) (8, *) (11, *) (16, *) (23, *) (34, *) (35, *) (52, *) (78, *)	(115, *) (116, *) (161, *) (162, *) (202, *) (203, *) (204, *) (214, *) (226, *) (242, *) (257, *) (261, *) (275, 22)
	Time(sec)	0.003	0.037	0.073	4.397	
	Error	1.53×10^{-6}	3.03×10^{-6}	6.73×10^{-6}	3.78×10^{-5}	
	Defect	8.89×10^{-7}	9.24×10^{-7}	4.81×10^{-7}	1.76×10^{-7}	
COLNEW	$\tau \Rightarrow$	1	7	10	16	
	Profile	(5, 2) (10, 1)	(5, 10) (5, 2) (10, 1) (10, 1) (20, 1) (16, 1) (32, 1)	(5, 3) (5, 2) (10, 2) (10, 1) (20, 1) (20, 1) (40, 1) (80, 1)	(5, 15) (5, 6) (5, 4) (10, 4) (10, 3) (10, 2) (20, 2) (20, 1) (20, 1) (40, 1) (80, 1) (160, 1)	
	Time(sec)	0.004	0.004	0.007	0.016	
	Error	9.03×10^{-9}	1.82×10^{-6}	6.88×10^{-8}	4.61×10^{-8}	
MUSN	$\tau \Rightarrow$	1	7	10	16	
	Profile	(5, 3)	(5, *) (10, *) (20, *) (40, 10)			
	Time(sec)	0.004	0.023			
	Error	3.65×10^{-10}	4.41×10^{-6}			

Table 3: Results for plasma confinement problem with $TOL = 10^{-6}$ and 4 values of τ

method		result				
MUSCRK	$\varepsilon \Rightarrow$	1.0	0.05	0.005	0.001	
	Profile	(1, 5)	(1, *)	(4, *)	(1, *)	(149, *)
			(2, 6)	(7, *)	(2, *)	(168, *)
				(8, *)	(3, *)	(170, *)
				(9, *)	(4, *)	(171, *)
				(10, *)	(7, *)	(172, *)
				(12, 24)	(13, *)	(256, *)
					(22, *)	(257, *)
					(23, *)	(259, *)
					(24, *)	(260, *)
					(25, *)	(261, *)
					(27, *)	(262, *)
					(33, *)	(263, *)
					(41, *)	(265, *)
					(47, *)	(268, *)
					(51, *)	(270, *)
					(66, *)	(272, *)
					(82, *)	(303, *)
					(91, *)	(304, *)
					(94, *)	(305, *)
					(95, *)	(306, *)
					(97, *)	(307, *)
					(109, *)	(309, *)
					(120, *)	(311, 13)
	Time(sec)	0.005	0.029	0.155	12.135	
	Error	4.29×10^{-4}	2.02×10^{-4}	1.23×10^{-3}	5.32×10^{-4}	
	Defect	1.47×10^{-6}	8.76×10^{-6}	5.70×10^{-5}	8.17×10^{-5}	
COLNEW	$\varepsilon \Rightarrow$	1.0	0.05	0.005	0.001	
	Profile	(5, 3)	(5, *)	(5, *)	(5, *)	
		(10, 1)	(10, 4)	(10, 6)	(10, 10)	
			(20, 1)	(20, 1)	(7, 1)	
					(14, 1)	
	Time(sec)	0.010	0.015	0.019	0.025	
	Error	3.51×10^{-9}	1.56×10^{-7}	3.40×10^{-4}	*	
MUSN	$\varepsilon \Rightarrow$	1.0	0.05	0.005	0.001	
	Profile	(5, 3)				
	Time(sec)	0.006				
	Error	1.55×10^{-9}				

Table 4: Results for swirling flow III problem with $TOL = 10^{-3}$ and 4 values of ε
$*$ for severe $TOL = 10^{-10}$ COLNEW is not able to converge to a solution

method	result				
MUSCRK	$\varepsilon \Rightarrow$	1.0	0.05	0.005	0.001
	Profile	(1, 6)	(1, 13)	(5, *)	(1, *) (116, *)
				(9, *)	(2, *) (151, *)
				(10, *)	(3, *) (169, *)
				(11, *)	(4, *) (171, *)
				(13, *)	(7, *) (174, *)
				(14, 21)	(13, *) (182, *)
					(23, *) (183, *)
					(24, *) (184, *)
					(25, *) (185, *)
					(27, *) (187, *)
					(32, *) (190, *)
					(39, *) (191, *)
					(46, *) (194, *)
					(50, *) (213, *)
					(65, *) (214, *)
					(83, *) (237, *)
					(90, *) (238, *)
					(94, *) (239, *)
					(96, *) (240, *)
					(98, *) (241, *)
					(100, *) (242, *)
					(102, *) (244, *)
					(104, *) (246, *)
					(105, *) (247, 15)
	Time(sec)	0.009	0.031	0.157	11.439
	Error	1.38×10^{-6}	9.19×10^{-7}	1.76×10^{-6}	1.00×10^{-7}
	Defect	4.15×10^{-8}	3.54×10^{-7}	6.54×10^{-7}	8.20×10^{-7}
COLNEW	$\varepsilon \Rightarrow$	1.0	0.05	0.005	0.001
	Profile	(5, 3)	(5, *)	(5, *)	(5, *)
		(10, 1)	(10, 4)	(10, 7)	(10, 11)
			(20, 1)	(20, 1)	(20, 1)
				(40, 1)	(40, 1)
					(80, 1)
	Time(sec)	0.009	0.016	0.021	0.035
	Error	3.50×10^{-9}	1.55×10^{-7}	1.47×10^{-8}	*
MUSN	$\varepsilon \Rightarrow$	1.0	0.05	0.005	0.001
	Profile	(5, 3)			
	Time(sec)	0.006			
	Error	1.50×10^{-9}			

Table 5: Results for swirling flow III problem with $TOL = 10^{-6}$ and 4 values of ε
$*$ for severe $TOL = 10^{-10}$ COLNEW is not able to converge to a solution

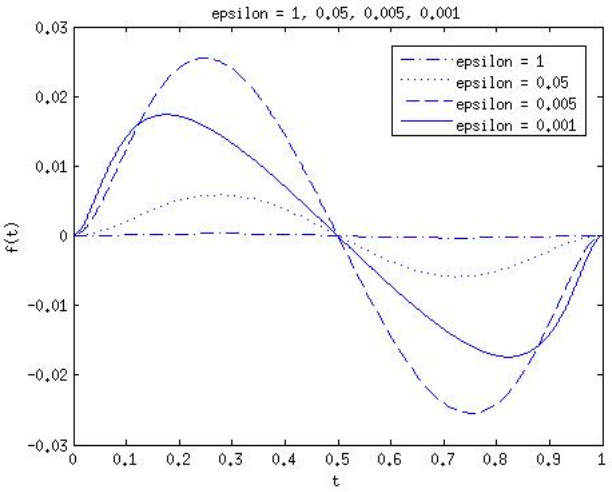
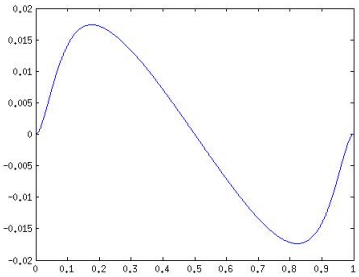


Figure 5: Solutions of swirling flow III problem for 4 values of ε

the converged BVP solution determined by the two BVP methods. In each case the approximate solution generated for the respective initial value problem satisfy the BVP boundary condition at the right endpoint. From the results of IVP approximations, we believe that both solutions are actual solutions to this problem.

result of MUSCRK



result of COLNEW

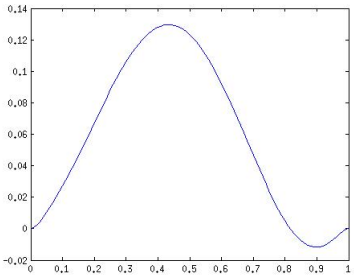


Figure 6: Solutions of swirling flow III problem for $\varepsilon = 0.001$, $f(t)$ versus t

We further test how different initial guesses can affect the solutions of BVP solvers for this problem. We used the non-symmetric solution of COLNEW ($\varepsilon = 0.001$, and $TOL = 10^{-6}$) to determine an initial guess for MUSCRK (at 50 equally distributed points). The result of MUSCRK shows that MUSCRK can be forced to converge to the non-symmetric solution. On the other hand, we use the symmetric solution of 50 equally distributed mesh points of MUSCRK ($\varepsilon = 0.001$, and $TOL = 10^{-6}$) to determine an initial guess for COLNEW. COLNEW didn't converge to a solution. When we increase the number to 100 equally distributed mesh points, COLNEW converges to a totally different result as Figure 7 shows.

This result is different from either solution in Figure 6. We cannot verify this result using a reliable stiff IVP solver. We conclude this result is not a solution of the problem.

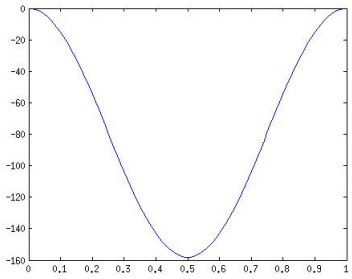


Figure 7: Result of CONEW to swirling flow III problem for $\varepsilon = 0.001$, $f(t)$ versus t by the solution of 100 equally distributed mesh points of MUSCRK as initial guess to COLNEW

5.3 Nonlinear Elastic Beams

$$\begin{array}{rcc} y' & = & \sin\theta \\ \theta' & = & M \\ M' & = & \frac{-Q}{\epsilon} \\ Q' & = & \frac{(y-1)\cos\theta - M(\sec\theta + \epsilon Q\tan\theta)}{\epsilon} \end{array}$$

subject to boundary conditions

$$y(0) = y(1) = 0, M(0) = M(1) = 0$$

and parameter values $\epsilon = 0.1, 0.05, 0.01, 0.005$ The default initial guess is

$$y_1(t) = 0, y_2(t) = -3 - t, y_3(t) = 0, y_4(t) = 1 + t$$

(taken from [15]). Refer to Figure 8 for a plot of the solutions and Tables 6 and 7 for a summary of the results for the 3 methods.

From Figure 8 it is clear that there are two types of solutions determined by the three methods. MUSCRK seems to produce one type (an oscillating solution with a frequency that increases as ε is decreased). The second type of solution is a "U-shaped" solution produced by COLNEW and MUSN with a boundary layer at both endpoints that becomes sharper as ε is decreased. The question is: are both of these actual solutions of the problem?

To investigate this question, we use the oscillating result ($\varepsilon = 0.05$, and $TOL = 10^{-6}$) to determine an initial guess supplied to COLNEW and MUSN. We use 20 equally distributed mesh points to capture the feature of the oscillating result for this initial guess, and plot the solution obtained in Figure 9. The result of MUSN seems to confirm that the oscillating result is an actual solutions of the problem.

method	result				
MUSCRK	$\varepsilon \Rightarrow$	0.1	0.05	0.01	0.005
	Profile	(1, 9)	(1, *) (2, 5)	(1, *) (2, *) (4, *) (8, *) (9, *) (10, *) (11, *) (13, *) (14, *) (28, *) (56, *) (112, *) (113, 9)	(1, *) (2, *) (4, *) (8, *) (16, *) (31, *) (32, 9)
	Time(sec)	0.018	0.021	0.707	0.272
	Error	2.05×10^{-3}	1.76×10^{-3}	8.87×10^{-4}	4.29×10^{-2}
	Defect	8.24×10^{-4}	2.91×10^{-4}	1.50×10^{-4}	7.88×10^{-4}
COLNEW	$\varepsilon \Rightarrow$	0.1	0.05	0.01	0.005
	Profile	(5, 2) (10, 1)	(5, 2) (10, 1)	(5, 3) (10, 1) (5, 1) (10, 1) (5, 1) (10, 1)	(5, 3) (10, 1) (20, 1) (10, 1) (20, 1)
	Time(sec)	0.006	0.006	0.011	0.012
	Error	1.90×10^{-5}	3.39×10^{-4}	2.88×10^{-3}	5.83×10^{-5}
MUSN	$\varepsilon \Rightarrow$	0.1	0.05	0.01	0.005
	Profile	(5, 3)	(5, 4)	(5, *) (10, 5)	(5, 10)
	Time(sec)	0.006	0.008	0.016	0.040
	Error	2.40×10^{-4}	7.03×10^{-3}	4.25×10^{-2}	3.92×10^{-2}

Table 6: Results for nonlinear elastic beams problem with $TOL = 10^{-3}$ and 4 values of ε

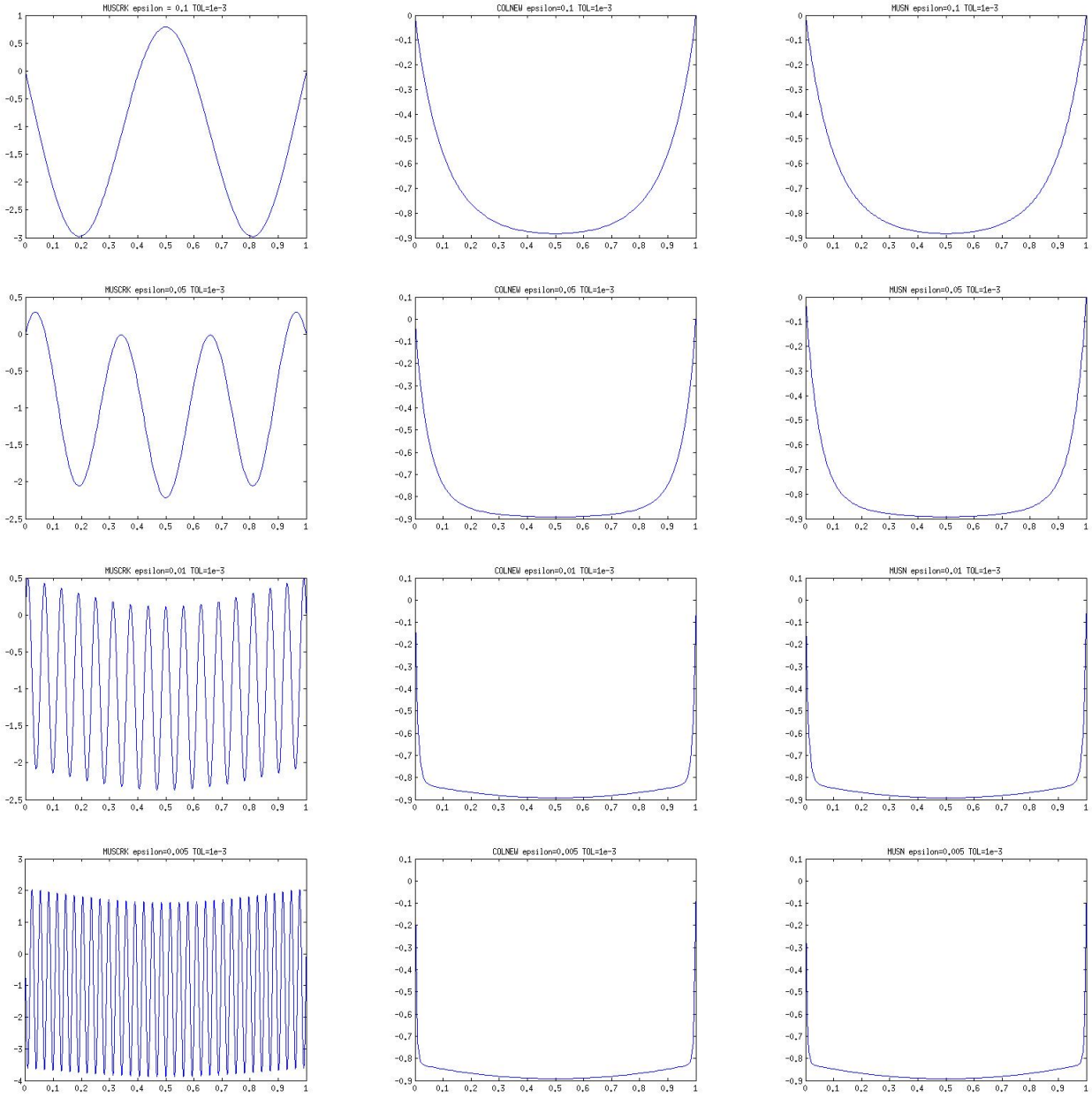


Figure 8: Solutions of the nonlinear elastic beams problem for 4 values of ε (left column solutions are computed by MUSCRK, middle column solutions are computed by COLNEW, and right column solutions are computed by MUSN)

method	result				
MUSCRK	$\varepsilon \Rightarrow$	0.1	0.05	0.01	0.005
	Profile	(1, 9)	(1, *) (2, 7)	(1, *) (2, *) (4, *) (8, *) (10, *) (11, *) (16, *) (32, 10)	(1, *) (2, *) (4, *) (8, *) (16, *) (19, *) (20, *) (31, *) (32, *) (64, *) (128, 12)
	Time(sec)	0.020	0.033	0.426	0.993
	Error	2.22×10^{-6}	1.15×10^{-5}	1.70×10^{-5}	1.38×10^{-5}
	Defect	9.99×10^{-7}	9.25×10^{-7}	8.72×10^{-7}	6.12×10^{-7}
COLNEW	$\varepsilon \Rightarrow$	0.1	0.05	0.01	0.005
	Profile	(5, 3) (10, 1) (20, 1)	(5, 3) (10, 1) (20, 1) (40, 1)	(5, 3) (10, 1) (10, 1) (20, 1) (40, 1)	(5, *) (10, 1) (20, 1) (20, 1) (16, 1) (32, 1)
	Time(sec)	0.009	0.013	0.014	0.015
	Error	6.50×10^{-7}	5.48×10^{-7}	6.70×10^{-6}	4.09×10^{-6}
MUSN	$\varepsilon \Rightarrow$	0.1	0.05	0.01	0.005
	Profile	(5, 5)	(5, 5)	(5, *) (10, 6)	(5, *) (10, *) (20, *) (40, 34)
	Time(sec)	0.010	0.014	0.027	0.066
	Error	3.24×10^{-6}	7.15×10^{-6}	3.69×10^{-5}	7.51×10^{-5}

Table 7: Results for nonlinear elastic beams problem with $TOL = 10^{-6}$ and 4 values of ε

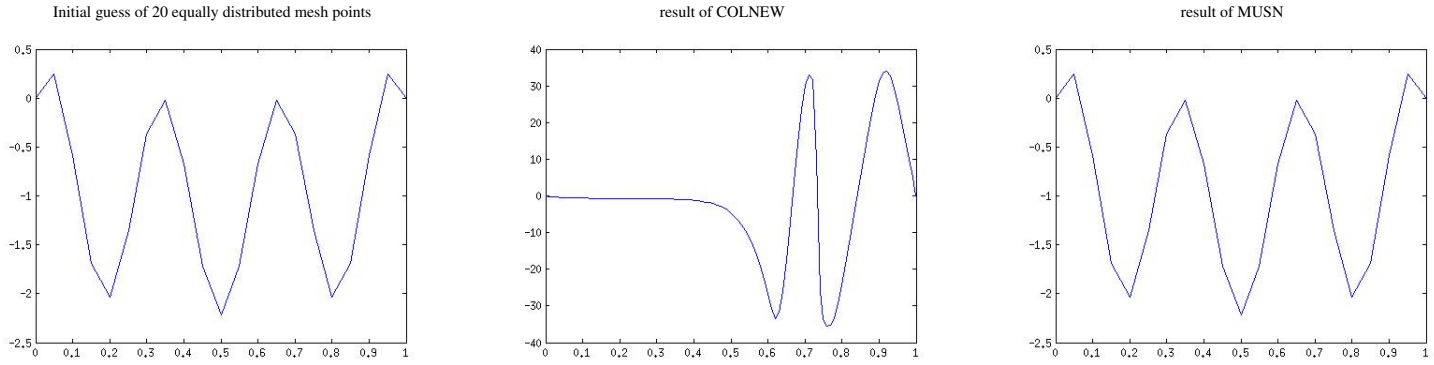


Figure 9: The result of nonlinear elastic beams problem using the oscillating solution ($\varepsilon = 0.05$, and $TOL = 10^{-6}$) as initial guess for COLNEW and MUSN, $M(t)$ versus t

Then, we use the "U-shaped" solution ($\varepsilon = 0.05$, and $TOL = 10^{-6}$) to produce an initial guess for MUSCRK. We use 16 unequally distributed mesh points to capture the feature of the result as initial guess for MUSCRK with the same values of τ and TOL. The result is shown in Figure 10. With the change of initial guess, MUSCRK is able to converge to the "U-shaped" solution.

As discussed above for the swirling flow III problem where multiple solutions were derived, we used a reliable IVP solver to verify the "U-shaped" solution produced by COLNEW and MUSN, and the oscillating solution produced by MUSCRK. From the results of our IVP approximation, we believe there are at least two different solutions to this problem "U-shaped" and oscillating solutions (there may be more). On the other hand, the approximate solution determined by COLNEW (in Figure 9) is not a solution.

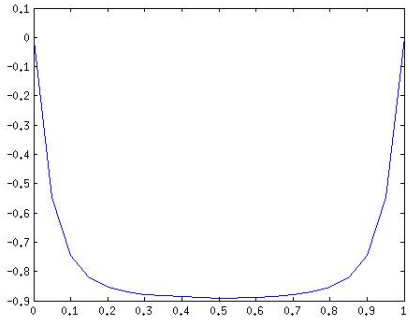
For this problem, MUSCRK produces oscillating solutions, while COLNEW and MUSN produce "U-shaped" solutions. We try to use the result of MUSCRK as initial guess supplied to COLNEW, but COLNEW cannot produce the oscillating solution (as illustrated in Figure 9). In Figure 9, we use 20 equally distributed initial guesses provided by MUSCRK, we further tried 50 and 100 equally distributed initial guesses provided by MUSCRK, and none of these resulted in convergence to the oscillating solution. The approximation result of 100 equally distributed initial guesses converges to "U-shaped" solution.

Of course one cannot compare performance when the methods are computing approximation to different solutions.

5.4 Artificial Boundary Layer

$$y'' = \frac{-3\tau y}{(\tau + t^2)^2}, t \in [-0.1, 0.1]$$

Initial guess of 16 unequally distributed mesh points



result of MUSCRK

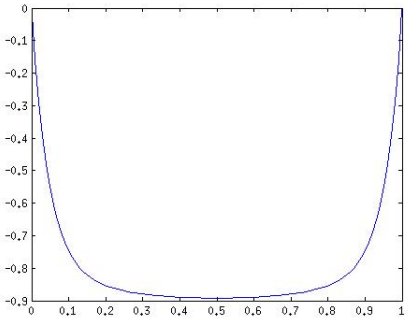


Figure 10: The result of nonlinear elastic beams problem using the "U-shaped" ($\tau = 10^{-5}$, and $TOL = 10^{-6}$) as initial guess for MUSCRK, $M(t)$ versus t

subject to boundary conditions

$$-y(-0.1) = y(0.1) = \frac{0.1}{\sqrt{\tau + 0.01}}$$

and parameter values $\tau = 0.01, 1e-3, 1e-4, 1e-5$. The default initial guess is

$$y_1(t) = 0, y_2(t) = 0,$$

(taken from [13]). It is easy to verify that the solution $y(t)$ is an odd function ($-f(x) = f(-x)$). Refer to Figure 11 for a plot of the solutions and Tables 8 and 9 for a summary of the results for the 3 methods.

From Figure 11 it is clear that there are two types of solutions determined by the three methods. MUSCRK and COLNEW seem to approximate one type (an "S-shaped" solution with a boundary layer in the middle that becomes sharp as τ is decreased). The second type is an oscillating solution with a frequency that increases as τ is decreased. We have the same question as we have in section 5.3: are both types actual solutions to the problem?

To investigate this question, we use "S-shaped" result ($\tau = 10^{-5}$, and $TOL = 10^{-6}$) to determine an initial guess for MUSN. We use 15 unequally distributed mesh points to capture the feature of the "S-shaped" result. The result is displayed in Figure 12, MUSN appears to converge to a solution from the supplied initial guess, but the approximation is not accurate near the boundary layer.

Then, we used the oscillating result ($\tau = 10^{-4}$, and $TOL = 10^{-6}$) as the initial guess for MUSCRK and COLNEW. We use 50 equally distributed mesh points to capture the feature of the result as initial guess with the same values of τ and TOL. The result is shown in Figure 13, it seems that at least for this particular problem, the change of initial guess is not able to alter the solution determined by MUSCRK and COLNEW.

We further use a reliable IVP solver to verify whether the "S-shaped" result produced by MUSCRK and COLNEW, and the oscillating solution produced by

method		result			
MUSCRK	$\tau \Rightarrow$	0.01	$1e-3$	$1e-4$	$1e-5$
	Profile	(1, 2)	(1, 2)	(1, 2)	(1, 2)
	Time(sec)	0.002	0.005	0.004	0.006
	Error	2.17×10^{-4}	6.06×10^{-4}	2.66×10^{-3}	2.71×10^{-3}
	Defect	1.79×10^{-4}	2.11×10^{-4}	4.05×10^{-4}	7.51×10^{-4}
COLNEW	$\tau \Rightarrow$	0.01	$1e-3$	$1e-4$	$1e-5$
	Profile	(5, 1) (10, 1)	(5, 1) (10, 1)	(5, 1) (10, 1) (20, *) (40, 2) (20, 1) (40, 1)	(5, 1) (10, 1) (7, 1) (14, 1) (10, 1) (20, 1) (40, 1)
	Time(sec)	0.004	0.004	0.011	0.009
	Error		1.31×10^{-2}	1.47×10^{-4}	6.20×10^{-3}
MUSN	$\tau \Rightarrow$	0.01	$1e-3$	$1e-4$	$1e-5$
	Profile	(5, 1)	(5, 2)	(5, 3)	(5, *) (10, 3)
	Time(sec)	0.003	0.003	0.004	0.008
	Error	6.28×10^{-6}	4.32×10^{-3}	1.04×10^{-4}	7.02×10^{-2}

Table 8: Results for artificial boundary layer problem with $TOL = 10^{-3}$ and 4 values of τ

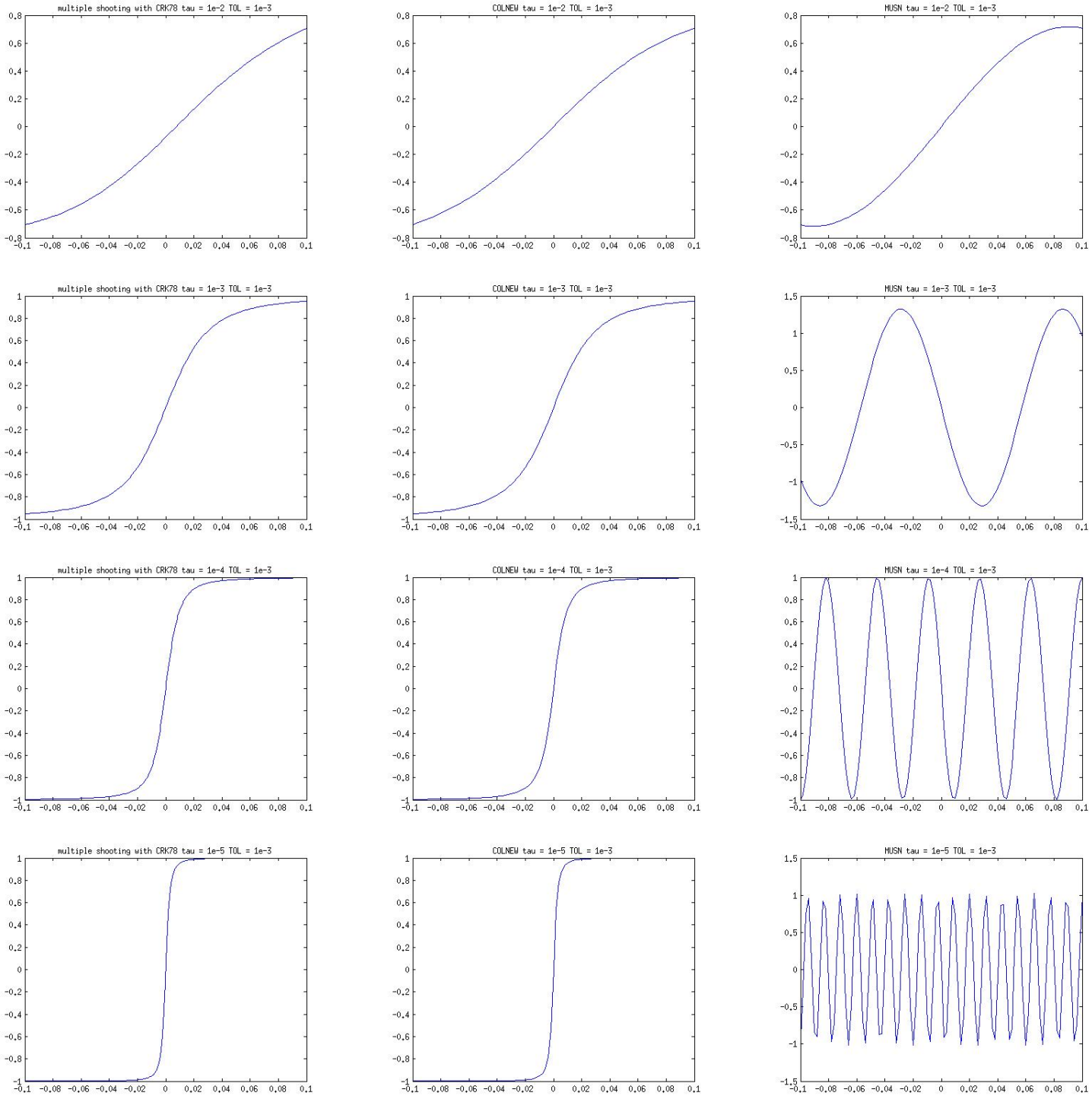
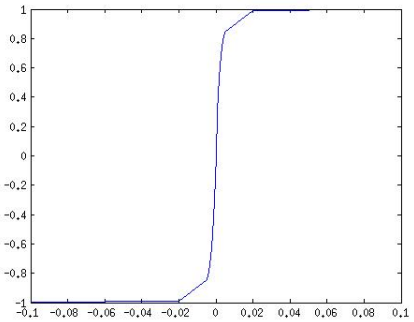


Figure 11: Solutions of artificial boundary layer problem for values of τ (left column solutions are computed by MUSCRK, middle column solutions are computed by COLNEW, and right column solutions are computed by MUSN)

method	result				
MUSCRK	$\tau \Rightarrow$	0.01	$1e-3$	$1e-4$	$1e-5$
	Profile	(1, 2)	(1, 4)	(1, 5)	(1, 9)
	Time(sec)	0.004	0.010	0.012	0.021
	Error	4.59×10^{-7}	1.04×10^{-6}	3.81×10^{-6}	4.63×10^{-6}
	Defect	6.41×10^{-7}	4.66×10^{-7}	7.85×10^{-7}	4.11×10^{-7}
COLNEW	$\tau \Rightarrow$	0.01	$1e-3$	$1e-4$	$1e-5$
	Profile	(5, 1) (10, 1) (20, 1) (40, 1) (80, 1) (160, 1) (320, 1) (640, *) (1280, *) (2560, *)	(5, 1) (10, 1) (20, 1) (40, 1)	(5, 1) (10, 1) (20, *) (40, 2) (29, 1) (58, 1)	(5, 1) (10, 1) (10, 1) (10, 1) (20, 1) (20, 1) (40, 1) (80, 1) (40, 1) (80, 1)
	Time(sec)		0.007	0.011	0.014
	Error		1.22×10^{-5}	4.56×10^{-5}	2.92×10^{-5}
MUSN	$\tau \Rightarrow$	0.01	$1e-3$	$1e-4$	$1e-5$
	Profile	(5, 2)	(5, 3)	(5, 4)	(5, *) (10, 5)
	Time(sec)	0.003	0.004	0.008	0.019
	Error	4.32×10^{-6}	6.73×10^{-6}	1.04×10^{-4}	2.36×10^{-4}

Table 9: Results for artificial boundary layer problem with $TOL = 10^{-6}$ and 4 values of τ

Initial guess of 15 unequally distributed mesh points



result of MUSN

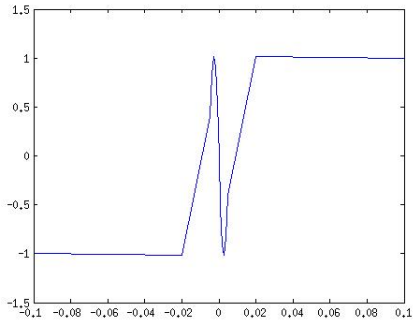


Figure 12: The result of artificial boundary layer using "S-shaped" solution ($\tau = 10^{-5}$, and $TOL = 10^{-6}$) as initial guess for MUSN, $y(t)$ versus t

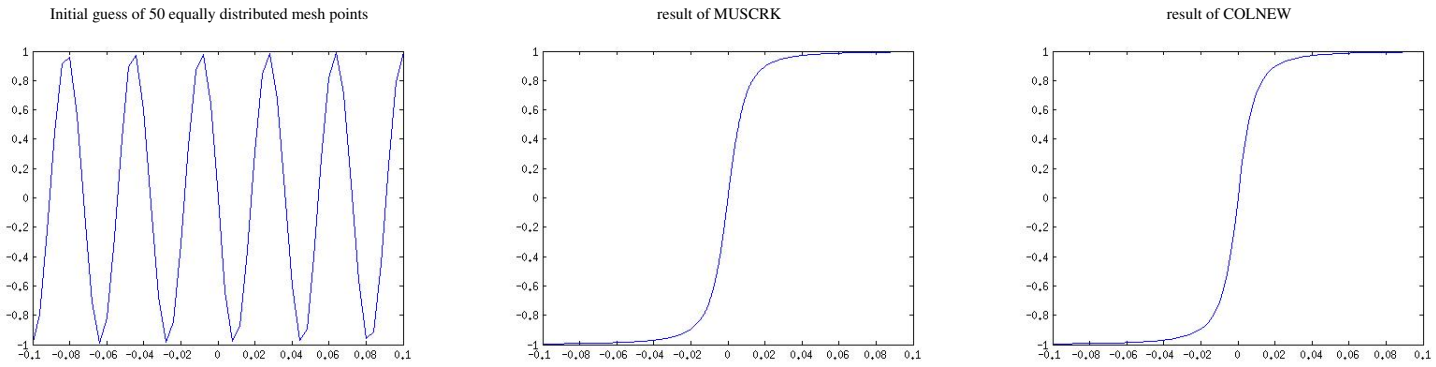


Figure 13: The result of artificial boundary layer using oscillating solution ($\tau = 10^{-4}$, and $TOL = 10^{-6}$) as initial guess for MUSCRK and COLNEW, $y(t)$ versus t

MUSN are solutions of the BVP. In addition, we investigate the inaccurate approximate solution in Figure 12 produced by MUSN (for all these IVP solutions, we use $\tau = 10^{-5}$, and $TOL = 10^{-3}, 10^{-6}, 10^{-8}$), to check whether they are solutions or not. From the results of the IVP approximations, we are able to confirm that "S-shaped" result is a solution, while all the other approximations produced by MUSN are not. From the oscillating solution of MUSN, we select 5, 10, and 50 equally distributed mesh points as initial values for the IVP solver, none of IVP approximations produces the oscillating result.

The approximation in Figure 12 produced by MUSN is produced by 15 unequally distributed mesh points from the solution of COLNEW. Using an IVP solver confirms that it is not a solution. When we use 33 unequally distributed mesh points of the solution of COLNEW as initial guesses, MUSN produces a similar result as the one in Figure 12, and IVP approximation confirms that it is not a solution either. So we can conclude both the oscillating results and the inaccurate approximation result in Figure 12 produced by MUSN are not solutions of the artificial boundary layer problem.

6 Conclusion and Observation

From the numerical tests we can see that MUSCRK can be applied to non-stiff and mildly stiff BVPs. For non-stiff BVPs, MUSCRK could sometimes outperform COLNEW and MUSN with lower execution time and fewer mesh points. As the BVPs become stiffer, for MUSCRK, the number of mesh points and the execution time increases rapidly. Generally, MUSCRK can be applied to wider range of stiff BVPs than MUSN; but is not as effective as COLNEW for problems with very sharp boundary layers.

MUSCRK generally does not need to be given mesh points as COLNEW and

MUSN do. MUSN cannot adjust the mesh points during the solution process. MUSN can only generate solution at mesh points if it converges to a solution, or display error message if it does not converge. So if dense output is required with MUSN, the only choice is to increase the number of mesh points. COLNEW can adjust mesh points during the process of solving BVPs. Generally for non-stiff BVPs, MUSCRK requires fewer mesh points to converge to a solution than does COLNEW and MUSN. For mildly stiff BVPs, MUSCRK requires fewer mesh points to converge than MUSN.

Also as the comparison tests above show, for a given BVP, multiple solutions may exist, and different initial guess can determine which solution is approximated. MUSCRK generally can be forced to converge to a different solution by changing the initial guess, while it is sometimes difficult for COLNEW to do so. Users do assume that the approximations returned by a BVP solver are approximations to a true solution. In our lengthy testing, we were able to find examples where this is not the case for the two widely used BVP solvers, COLNEW and MUSN.

One of the disadvantages of our implementation of MUSCRK is as the number of mesh points N becomes large, there are N IVPs to be integrated, and continuous Runge-Kutta methods require substantially more function evaluations per step than a discrete Runge-Kutta method of the same order. We made some time measurement of function evaluations on some of the test problems, and we find that function evaluations account for about 80% of the total computation time for a method. One of the features of a multiple shooting method is that the integration of N IVPs in (5) can be performed independently (mentioned in [1], section 4.3), If we export this observation in MUSCRK, it should be possible to improve the performance significantly.

Another disadvantage of MUSCRK compared with COLNEW is that as the problem becomes stiffer, generally MUSCRK fails to converge to solution earlier than COLNEW. This is because we use explicit continuous Runge-Kutta methods (such as CRK78 and CRK56). We could switch to implicit Runge-Kutta method when the problem becomes stiff.

7 Acknowledgments

I am grateful to my supervisor Prof. Wayne Enright for his efforts in conducting my research project and preparing this thesis. I would like to thank M. Shakourifar for his helpful discussions during the preparation of this paper. I would also like to thank Dr. Tom Fairgrieve for his valuable comments and suggestions after carefully reading this thesis.

References

[1] R.M.M. Mattheij U.M. Ascher and R.D. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. Classics in

Applied Mathematics Series, Society for Industrial and Applied Mathematics, Philadelphia, 1995.

[2] J. D. Riley D. D. Morrison and J. F. Zancanaro. Multiple shooting method for two-point boundary value problems. *Comm. AMC*, 1(5):613–614, 1962.

[3] Herbert B. Keller. *Numerical Solution of Two Point Boundary Value Problems*. SIAM, 1976.

[4] R.M.M. Mattheij and G.W.M. Staarink. An efficient algorithm for solving general linear two point bvp. Report 8220, Math. Inst. Catholic University, Nijmegen, 1982.

[5] R.M.M. Mattheij and G.W.M. Staarink. Musn. `http://www.netlib.org/ode/`, June 1992.

[6] U.M. Ascher and G. Bader. Colnew. `http://www.netlib.org/ode/`, June 1992.

[7] S. P. Nørsett E. Hairer and G. Wanner. *Solving Ordinary Differential Equations I*. Springer-Verlag, 1993.

[8] W.H. Enright. A new error-control for initial value solvers. *Applied Mathematics and Computation - AMC*, 31(3):288–301, 1989.

[9] W.H. Enright and Li Yan, 2009. The Reliability/Cost Trade-off for a Class of ODE solvers.

[10] W.H. Enright. Continuous numerical methods for odes with defect control. *Journal of Computational and Applied Mathematics*, 125(2000):159–170, 1999.

[11] W.H. Enright and P.H. Muir. New interpolants for asymptotically correct defect control of bvodes. *Numerical Algorithms*, 53(2):219238, 2010.

[12] U.M. Ascher and L. P. Petzold. *Computer methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, 1998.

[13] P. Deuflahrd. Nonlinear equation solvers in boundary value problem codes. *Proceedings of a Working Conference on Codes for Boundary-Value Problems in Ordinary Differential Equations*, 76(1979):40–66, 1978.

[14] W. H. Enright. The relative efficiency of alternative defect control schemes for high-order continuous runge-kutta formulas. *SIAM Journal on Numerical Analysis*, 30:1419–1445, 1993.

[15] J. Cash. 35 boundary value test problems. `http://www2.imperial.ac.uk/~jcash/BVP_software`, June 1992.