European Society of Computational Methods in Sciences and Engineering (ESCMSE)

Journal of Numerical Analysis, Industrial and Applied Mathematics (JNAIAM)
vol. 1, no. 1, 2006, pp. 49-58
ISSN 1790–8140

The Derivation of Interpolants for Nonlinear Two-Point Boundary Value Problems

J. R. Cash* [1], N. Sumarti*, T. J. Abdulla† and I. Vieira‡

* *Dept. of Mathematics, Imperial College, South Kensington, London SW7 2AZ*
† *Dept. of Mathematics, College of Science, University of Bahrain*
‡ *ISCAP, Rua Jaime Lopes de Amorim, 4465-111 S. Mamede Infesta, Portugal*

Received 10 November, 2005; accepted in revised form 4 January, 2006

Abstract: There now exist several powerful codes for the numerical integration of first order systems of nonlinear two-point boundary value problems. All of these codes aim to provide a continuous solution to the problem being solved but the way in which each code achieves this aim is quite different. In this paper we survey some of the approaches that are at present being used to derive interpolants and we provide a new interpolant for use with TWPBVPL.f which is a highly stable deferred correction code based on Lobatto formulae.

Keywords: Interpolant, Boundary value problems

Mathematics Subject Classification: 65D05, 65L10

1 Introduction

In the present paper we will be concerned with the numerical integration of the first order system of nonlinear two-point boundary value problems

$$\frac{dy}{dx} = f(x, y), \quad a \leq x \leq b, \; g(y(a), y(b)) = 0. \tag{1.1}$$

Several efficient codes now exist for dealing with this class of problems and, in particular, we mention [1, 2, 3, 4, 5]. Modern codes seek to provide a continuous solution to (1.1) but, as we will explain, the way in which they go about this task differs considerably from code to code.

The first code we consider is COLSYS [1] which is based on collocation. The aim of COLSYS is to compute an approximation to the solution of (1.1) using a piecewise continuous polynomial which satisfies a prescribed accuracy requirement over the whole range of integration. This is done using collocation at Gauss points. Although we will be mainly interested in first order systems of

[1] Corresponding author : E-mail : j.cash@imperial.ac.uk

the form (1.1), we can most conveniently explain the COLSYS approach by considering the single second order equation

$$y'' = f(x, y, y'),\ a \le x \le b,\quad g(y(a), y'(a), y(b), y'(b)) = 0. \tag{1.2}$$

We will assume that $y(x) \in C[a, b]$ and we seek an approximate solution of (1.2) having the general form

$$v_\pi(x) = \sum_{j=1}^{M} \alpha_j \phi_j(x),\quad a \le x \le b. \tag{1.3}$$

Here the $\phi_j(x)$ are known linearly independent functions defined on the range $[a, b]$ and α_j are constants to be chosen. The M free parameters α_j appearing in (1.3) are determined by the requirement that $v_\pi(x)$ should satisfy the (two) boundary conditions at $x = a$ and $x = b$ and also that $v_\pi(x)$ should satisfy the ODE exactly at $M - 2$ points in $[a, b]$. The particular choices made in deriving COLSYS are that the $M - 2$ points (known as the collocation points) are Gauss points and the $\phi_j(x)$ are piecewise polynomials. If the collocation is carried out using s collocation points per subinterval, and the maximum size of any subinterval is h, then the global error in COLSYS is uniformly $O(h^s)$. It is important to appreciate the fact that COLSYS provides a continuous solution (which is normally much more expensive to compute than a discrete solution) and users are given no choice concerning whether they want a discrete or a continuous solution.
The second code we consider is MIRKDC which is due to Enright and Muir [2]. This code also computes a continuous solution but the way in which it is done, and the reason why a continuous solution has to be computed, is very different from what is the case with COLSYS. The MIRKDC code is based on the well known class of mono-implicit Runge-Kutta formulae [6]. The approach adopted by Enright and Muir is to add extra stages to the standard MIRK formulae so that a continuous solution can be computed. The problem of deriving continuous MIRK formulae has been considered in detail in [7] and, in that paper, continuous MIRK schemes having the minimum number of stages are derived for order 1 - 6. One of the things they discover is that computing higher order continuous MIRK schemes is very challenging. The approach now adopted by Enright and Muir is as follows. They first compute a discrete solution using a standard MIRK formula of the form

$$\begin{aligned} y_{n+1} &= y_n + h \sum_{i=1}^{s} b_i f(x_n + c_i h, Y_i), \\ Y_i &= (1 - v_i) y_n + v_i y_{n+1} + h \sum_{j=1}^{s} x_{ij} f(x_n + c_j h, Y_j), i = 1, \ldots, s. \end{aligned} \tag{1.4}$$

Having done this, they compute the extra stages necessary to define a continuous solution

$$U(x) = U(x_i + \theta h_i) = y_i + h_i \sum_{r=1}^{s^*} b_r(\theta) k_r, \tag{1.5}$$

where

$$\theta = \frac{(x - x_i)}{h_i},\quad 0 \le \theta \le 1,\quad x_i \le x \le x_{i+1}.$$

This formula is embedded in the sense that the first s stages of (1.5) are the same as the s stages used in (1.4).

Having defined a continuous solution $U(x)$, Enright and Muir seek to control the defect

$$r(x) = U'(x) - f(x, U(x)). \tag{1.6}$$

They derive a mesh choosing algorithm which equidistributes this defect and this allows them to solve difficult problems for which a highly non-uniform mesh is needed. Thus in this approach the interpolant $U(x)$ is computed in order to define the defect and, once a final solution has been computed, the continuous MIRK formula is used as the interpolant. Again the user has no control over whether a discrete or continuous solution is computed since the code has to compute a continuous solution (which is much more expensive than a discrete one to compute) in order to define the defect.

We now consider the problem of computing an interpolating polynomial when using algorithms such as [3, 4, 5] which compute the final numerical solution only at a discrete set of points. For algorithms such as these the problem of computing high quality interpolants in an efficient manner is more difficult than at first seems the case [8]. However, for algorithms of this particular class the users have the option of deciding whether or not they require a continuous solution to be computed. One possibility is that the user may require the solution to be computed only on a set of isolated points. The most efficient way of doing this is to choose all meshes so that all of these isolated points are included in every mesh. This is very straightforward to do and all codes considered in this paper have this facility. Alternatively the user may require a continuous solution over just a few mesh intervals. This is typically the case for event location where, for example, the user may wish to determine whether a solution $y(x)$, or its derivative $y'(x)$, passes through zero. Having to provide a continuous solution over just a few mesh intervals is likely to be much less expensive than providing a solution which is continuous everywhere. We will describe how to compute these 'a posteriori' interpolants in the next section but before doing this we will summarise some of the qualities we expect an interpolant to have.

1. We want the interpolant to be a piecewise continuous polynomial with global continuity of the solution and its first derivative at the mesh points.

2. We need to construct an interpolating polynomial which, for the sake of efficiency, uses at most a few additional function evaluations in each mesh interval.

3. We need to determine a measure of quality to ensure that the interpolated solutions are in some sense of comparable accuracy to those obtained at the mesh points.

4. We want the coefficients of the interpolation polynomial to remain in some sense 'nicely bounded' throughout the interval.

2 High Order Interpolants

In this section we will consider the problem of defining interpolants for the two deferred correction codes TWPBVP.f and TWPBVPL.f which are based on MIRK formulae and Lobatto formulae respectively. Very similar techniques can also be applied to the BVMs derived in [5]. Several approaches to the problem of deriving interpolants for finite difference methods which provide a discrete solution have been considered in the literature. One approach is to compute interpolants using data defined over several adjacent mesh intervals. Such interpolants become non-symmetric for a variable mesh and near the end of the range of integration. It is well known that interpolants of this form can perform very poorly on highly non-uniform meshes. For this reason we will define our interpolant to be symmetric and we use data defined over just a single mesh interval. Symmetric interpolation formulae are consistent with the fact that for boundary value problems there is no direction of integration and practical experience has shown that symmetric formulae can give very satisfactory results.

The first problem we consider is that of deriving interpolants for use with MIRK formulae and in particular for TWPBVP.f. There are two types of data that we can use to define an interpolant. The first is the data that has been used to compute the discrete solution. However unless this data is very carefully constructed, it often does not have sufficient accuracy to allow it to be used in the interpolant. The other type of data is the extra function evaluations that are computed after the discrete solution has been defined. It is relatively difficult to define an interpolant for the original eighth order formula developed by Cash and Singhal [6] because many of the stages in this formula have low order so they cannot be used in the interpolant.
The problem facing us can be summarised as follows: How can we derive an efficient discrete MIRK formula where some of the stages have high order and so can be used in the construction of the interpolant. This has been done by Cash [3] who derived a discrete MIRK formula of order 8 with 9 stages where some of these stages have high order. This formula, which is called MIRK8, was then investigated by Cash and Moore [9] who considered the problem of how to define a continuous solution using as few extra function evaluations as possible and they derived an order 8 interpolant requiring just 3 extra function evaluations per subinterval. Extensive numerical testing has shown that this interpolant performs extremely well on a wide class of problems and we feel that the problem of deriving quality interpolants for MIRK formulae is now just about solved.

In Table 1 we give some typical results obtained by the interpolant. The problem we consider is Problem 1 of the well known test set, which can be obtained on www.ma.imperial.ac.uk/∽ns1, run with an accuracy tolerance of $Tol = 10^{-8}$. This problem is

$$\epsilon y'' - y = 0, \quad y(0) = 1, \ y(1) = 0. \tag{2.1}$$

For a given value of ϵ we first computed the discrete solution, the maximum error at any grid point appears under the heading *Err Sol*. We then computed the order 6 and order 8 interpolants defined in the paper of Cash and Moore [9]. We computed the error in the interpolant at all mesh points $x_i, x_{i+h/4}, x_{i+h/2}, x_{i+3h/4}, x_{i+1}$ and computed the maximum error at any of these test points. The maximum error in the 6th order interpolant is given under *Err y6* and the maximum error in the 8th order interpolant is given under *Err y8*. As can be seen the eighth order interpolant is extremely accurate, the sixth order is less so but still gives excellent results. An added bonus is that the difference between *Err y6* and *Err y8* gives an excellent estimate of the error in the 6th order solution.

Table 1: The interpolation using MIRK8, $TOL = 10^{-8}$

ϵ	Err Sol	Err $y6$	Err $y8$
1	1.015d-17	1.241d-12	8.694d-17
10^{-1}	3.337d-13	1.172d-09	7.558d-13
10^{-2}	1.385d-11	1.731d-08	2.618d-11
10^{-3}	2.112d-12	4.365d-09	4.274d-12
10^{-4}	3.637d-12	2.794d-09	6.227d-12
10^{-5}	1.258d-11	1.615d-08	2.391d-11

It is well known that, although the code TWPBVP.f which is based on MIRK formulae is often very efficient for non-stiff and mildly stiff problems, it can perform very poorly on excessively stiff problems. This is due to the weak stability properties of MIRK formulae, which is in turn due to the fact that the deferred corrections performed in TWPBVP.f are explicit. The stability properties of some iterated deferred correction codes have been considered in detail in [11]. One of the most illuminating ways to examine the stability of different formulae is to apply the codes to

the initial value problem $y' = \lambda y$ and to monitor the behaviour of y as $\lambda \to \infty$. For the deferred correction scheme implemented in TWPBVP.f it was found that

$$\frac{y_{n+1}}{y_n} \sim (h\lambda)^4 \text{ as } h\lambda \to \infty,$$

and so it does not have the stability necessary to deal with stiff problems. To overcome this stability problem , a deferred correction code TWPBVPL.f based on Lobatto formulae was developed [4]. The crucial thing about this formula is that the deferred corrections are implicit and so we are able to obtain the excellent stability necessary to solve stiff systems of two-point boundary value problems. In fact for the code TWPBVPL.f we are able to prove that

$$\frac{y_{n+1}}{y_n} \sim 1 \text{ as } h\lambda \to \infty.$$

The obvious question that now faces us is whether we can easily modify the MIRK interpolants derived by Cash and Moore [9] so that they perform well with stiff problems. Put another way we need to know whether the explicitness of the Cash - Moore interpolants will cause problems when solving stiff equations even though the interpolants are computed "a posteriori" and the discrete solution is computed using a very stable formula. Extensive numerical testing with an explicit interpolant has shown that, as is to be expected, the explicit interpolants do perform very poorly when applied to stiff problems. In particular it is often the case that as the problem becomes more stiff so the performance of the interpolant becomes gradually worse.

In Table 2 we give some typical results that were obtained when an explicit interpolant was used with TWPBVPL.f. As can be seen the discrete algorithm is very efficient but as the problem becomes more and more stiff the performance of the interpolant deteriorates seriously. Note that in column *Nmsh* we put the number of final mesh points of the obtained discrete solution.

Table 2: The explicit interpolation using Lobatto, $TOL = 10^{-8}$

ϵ	Nmsh	Err Sol	Err $y6$	Err $y8$
1	10	0.463E-14	0.320E-09	0.637E-12
10^{-1}	19	0.663E-12	0.488E-08	0.242E-10
10^{-2}	55	0.250E-12	0.123E-08	0.574E-11
10^{-3}	83	0.283E-12	0.679E-09	0.361E-11
10^{-4}	97	0.109E-12	0.123E-08	0.383E-11
10^{-5}	109	0.842E-13	0.575E-09	0.262E-11
10^{-6}	115	0.151E-12	0.157E-08	0.528E-11
10^{-7}	131	0.231E-13	0.393E-09	0.467E-11
10^{-8}	140	0.352E-13	0.537E-09	0.128E-08
10^{-9}	162	0.138E-12	0.146E-08	0.389E-09
10^{-10}	163	0.903E-13	0.107E-08	0.159E-05
10^{-11}	146	0.239E-12	0.219E-08	0.214E-08
10^{-12}	151	0.365E-13	0.551E-09	0.471E-05
10^{-13}	221	0.821E-13	0.100E-08	0.318E-04
10^{-14}	197	0.747E-13	0.933E-09	0.785E-05
10^{-15}	249	0.332E-13	0.514E-09	0.218E-02

What our numerical results have shown is that in order to get a good interpolant for use with the implicit deferred correction code TWPBVPL.f it is necessary to make the interpolant implicit.

This problem has been considered in some detail in [10]. That paper contains a lot of important information about implicit interpolants which also applies to the interpolants considered in this section. However, the approach adopted in the present paper is different from that considered in [10]. In [10] the idea was to compute a discrete solution of order $p+2$ based on Lobatto formulae and then to use this data to compute an interpolant of order p. So, for example, in order to get an eighth order implicit interpolant it is necessary to have first computed the discrete solution using a tenth order Lobatto formula.

In this section we use a rather different and more straightforward approach. If we consider the case where the discrete solution is computed using a sixth order Lobatto formula we have at our disposal values of $[y_n, y'_n, y_{n+1}, y'_{n+1}]$ at the ends of each sub interval as well as some internal data which is not of sufficient accuracy for use in an interpolant. To define an order 6 interpolant we proceed as follows. We first compute some additional quantities $y_{n+\alpha}$ and $y_{n+\beta}$ as the solution of

$$\begin{aligned} y_{n+\alpha} &= A_{I\alpha}y_n + B_{I\alpha}y_{n+1} + h_n\left(C_{I\alpha}y'_n + D_{I\alpha}y'_{n+1} + \right. \\ &\quad \left. E_{I\alpha}y'_{n+\alpha} + F_{I\alpha}y'_{n+\beta}\right), \\ y_{n+\beta} &= A_{I\beta}y_n + B_{I\beta}y_{n+1} + h_n\left(C_{I\beta}y'_n + D_{I\beta}y'_{n+1} + \right. \\ &\quad \left. E_{I\beta}y'_{n+\alpha} + F_{I\beta}y'_{n+\beta}\right). \end{aligned}$$

Having computed these additional quantities we now define our interpolant as

$$\begin{aligned} y^{I6}_{n+\omega} = \; & A_{I6}(\omega)y_n + B_{I6}(\omega)y_{n+1} + h_n(C_{I6}(\omega)y'_n + D_{I6}(\omega)y'_{n+1} \\ & + E_{I6}(\omega)y'_{n+\alpha} + F_{I6}(\omega)y'_{n+\beta}). \end{aligned} \tag{2.2}$$

Note that these two quantities $y_{n+\alpha}$ and $y_{n+\beta}$ are defined implicitly so to compute them we need to solve a system of algebraic equations. However this system is much smaller than the system required to compute the discrete solution and also we should expect rapid convergence of the iteration scheme because we have very good initial approximations to the extra quantites we compute. The local truncation error associated with the formula (2.2) is given in the Appendix. We experimented with various choices of α and β, which are free parameters, but the code did not seem to be sensitive to different choices. We therefore chose these quantities as being symmetric but otherwise arbitrary as $\alpha = 1/3,\ \beta = 2/3$.

Finally we consider the problem of deriving an eighth order interpolant. To do this we need to define 4 extra pieces of data to go with $[y_n, y'_n, y_{n+1}, y'_{n+1}]$ which have been computed by the order 8 discrete deferred correction scheme. We first define $y_{n+\alpha}, y_{n+\beta}, y_{n+\delta}, y_{n+\eta}$ as the solution of the system of four equations

$$\begin{aligned} y_{n+\alpha} &= A_{I\alpha}y_n + B_{I\alpha}y_{n+1} + h_n\left(C_{I\alpha}y'_n + D_{I\alpha}y'_{n+1} + \right. \\ &\quad \left. E_{I\alpha}y'_{n+\alpha} + F_{I\alpha}y'_{n+\beta} + G_{I\alpha}y'_{n+\delta} + H_{I\alpha}y'_{n+\eta}\right), \\ y_{n+\beta} &= A_{I\beta}y_n + B_{I\beta}y_{n+1} + h_n\left(C_{I\beta}y'_n + D_{I\beta}y'_{n+1} + \right. \\ &\quad \left. E_{I\beta}y'_{n+\alpha} + F_{I\beta}y'_{n+\beta} + G_{I\beta}y'_{n+\delta} + H_{I\beta}y'_{n+\eta}\right), \\ y_{n+\delta} &= A_{I\delta}y_n + B_{I\delta}y_{n+1} + h_n\left(C_{I\delta}y'_n + D_{I\delta}y'_{n+1} + \right. \\ &\quad \left. E_{I\delta}y'_{n+\alpha} + F_{I\delta}y'_{n+\beta} + G_{I\delta}y'_{n+\delta} + H_{I\delta}y'_{n+\eta}\right), \\ y_{n+\eta} &= A_{I\eta}y_n + B_{I\eta}y_{n+1} + h_n\left(C_{I\eta}y'_n + D_{I\eta}y'_{n+1} + \right. \\ &\quad \left. E_{I\eta}y'_{n+\alpha} + F_{I\eta}y'_{n+\beta} + G_{I\eta}y'_{n+\delta} + H_{I\eta}y'_{n+\eta}\right). \end{aligned}$$

Table 3: The implicit interpolation using Lobatto, $TOL = 10^{-8}$

ϵ	Nmsh	Err Sol	Err $y6$ (2.2)	Err $y8$ (2.3)	Err $y8$ (3.3)
1	10	0.463E-14	0.131E-10	0.150E-12	0.878E-13
10^{-1}	19	0.663E-12	0.203E-09	0.964E-12	0.105E-11
10^{-2}	54	0.707E-12	0.500E-10	0.657E-12	0.132E-11
10^{-3}	81	0.512E-12	0.273E-10	0.301E-12	0.540E-12
10^{-4}	93	0.582E-12	0.500E-10	0.233E-12	0.595E-12
10^{-5}	106	0.842E-13	0.230E-10	0.872E-13	0.134E-12
10^{-6}	107	0.151E-12	0.638E-10	0.232E-12	0.223E-12
10^{-7}	125	0.231E-13	0.157E-10	0.755E-13	0.777E-13
10^{-8}	133	0.352E-13	0.215E-10	0.839E-13	0.123E-12
10^{-9}	152	0.138E-12	0.594E-10	0.213E-12	0.202E-12
10^{-10}	156	0.903E-13	0.434E-10	0.193E-12	0.283E-12
10^{-11}	132	0.239E-12	0.897E-10	0.332E-12	0.355E-12
10^{-12}	144	0.365E-13	0.220E-10	0.850E-13	0.799E-13
10^{-13}	206	0.821E-13	0.404E-10	0.188E-12	0.119E-12
10^{-14}	185	0.748E-13	0.376E-10	0.186E-12	0.108E-12
10^{-15}	244	0.332E-13	0.205E-10	0.818E-13	0.797E-13

Once this additional data has been computed we define the interpolant as

$$\begin{aligned} y^{I8}_{n+\omega} = \quad & A_{I8}(\omega)y_n + B_{I8}(\omega)y_{n+1} + h_n(C_{I8}(\omega)y'_n + D_{I8}(\omega)y'_{n+1} + \\ & E_{I8}(\omega)y'_{n+\alpha} + F_{I8}(\omega)y'_{n+\beta} + G_{I8}(\omega)y'_{n+\delta} + H_{I8}(\omega)y'_{n+\eta}). \end{aligned} \tag{2.3}$$

The system of equations to be solved is nonsingular if

$$\begin{aligned} &(((70\alpha - 35)\eta - 35\alpha + 21)\delta + (-35\alpha + 21)\eta + 21\alpha - 14)\beta + \\ &((-35\alpha + 21)\eta + 21\alpha - 14)\delta + (21\alpha - 14)\eta - 14\alpha + 10 \neq 0. \end{aligned}$$

The coefficients of this interpolant together with the local truncation error are given in the Appendix. Again the performance of the interpolant does not depend significantly on the choices of $\alpha, \beta, \gamma, \eta$ so we can choose $\alpha = 1/5, \beta = 2/5, \gamma = 3/5, \eta = 4/5$ which is a symmetric but otherwise arbitrary choice.

We now consider the performance of this interpolant on Problem (2.1) with $Tol = 10^{-8}$. As can be seen from Table 3 column 3, the Lobatto order 8 deferred correction code produces excellent results throughout. From columns 4 and 5, we can see that the sixth order interpolant produces very accurate results while the eighth order interpolant produces exceptionally accurate results. We also have the very important property that the difference between the sixth order and the eighth order interpolated solutions gives a very good estimate of the error in the order 6 interpolant. Extensive numerical testing on a test set of 32 problems [12] showed these implicit interpolants to generally perform extremely well in defining a continuous solution.

3 Special Problems

Finally in this section we consider the development of high quality interpolants for special problems. The particular class of problems we are interested in is

$$y'' = f(x, y, y'). \tag{3.1}$$

Normally we reduce the second order equation (3.1) to first order form

$$\begin{aligned} y' &= z \\ z' &= f(x, y, z), \end{aligned} \tag{3.2}$$

and derive methods for dealing with this first order system. However if the problem is of the special form (3.1) we have at the end of each subinterval the data $[y_n, y'_n, y''_n, y_{n+1}, y'_{n+1}, y''_{n+1}]$. This immediately allows us to derive a sixth order interpolation polynomial for both the code TWPBVP.f and TWPBVPL.f. To get an order 8 interpolant for use with TWPBVP.f we need to derive two extra pieces of data and this is described in [9]. An alternative way of doing this, if the differential equation being solved is relatively straightforward, is to differentiate both sides of (3.1) so that we have y''' available. This allows us to derive an order 8 Hermite interpolating polynomial. This works very well for non stiff problems but is poor for stiff problems. To derive an order 8 interpolating polynomial for stiff problems we need to define two extra pieces of implicit data. We do this by defining

$$\begin{aligned} y_{n+\alpha} &= A_\alpha y_n + B_\alpha y_{n+1} + h\left(C_\alpha y'_n + D_\alpha y'_{n+1}\right) + \\ &\quad h^2\left(E_\alpha y''_n + F_\alpha y''_{n+1}\right) + h\left(G_\alpha y'_{n+\alpha} + H_\alpha y'_{n+\beta}\right), \\ y_{n+\beta} &= A_\beta y_n + B_\beta y_{n+1} + h\left(C_\beta y'_n + D_\beta y'_{n+1}\right) + \\ &\quad h^2\left(E_\beta y''_n + F_\beta y''_{n+1}\right) + h\left(G_\beta y'_{n+\alpha} + H_\beta y'_{n+\beta}\right). \end{aligned}$$

Having computed this extra data, we define an interpolant as

$$\begin{aligned} y^{S8}_{n+\omega} = A_{S8}(\omega) y_n + B_{S8}(\omega) y_{n+1} + h\left(C_{S8}(\omega) y'_n + D_{S8}(\omega) y'_{n+1}\right) + \\ h^2\left(E_{S8}(\omega) y''_n + F_{S8}(\omega) y''_{n+1}\right) + h\left(G_{S8}(\omega) y'_{n+\alpha} + H_{S8}(\omega) y'_{n+\beta}\right). \end{aligned} \tag{3.3}$$

The values of these parameters and the local truncation error of this formula are given in the Appendix. Here the choice of α and β is arbitrary (but we should of course maintain symmetry) and we choose $\alpha = 1/3$, $\beta = 2/3$.

We complete this section by giving some numerical results to highlight the performance of this algorithm. We consider again Problem (2.1) with $TOL = 10^{-8}$. We found that the performance of the interpolant defined by (3.3) was very good (see the last column in Table 3) and the eighth order solution is as highly accurate as it is in the first order case. We feel that this implicit interpolant solves the problem of deriving continuous solutions for special problems of the form (3.1) and that the algorithms described in this paper are extremely efficient and reliable for defining continuous solutions.

References

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

[2] W.H. Enright and P.H. Muir, *Runge-Kutta software with defect control for boundary value ODEs*, SIAM J. Sci.Comput., 17 (1996), pp. 479-497.

[3] J.R. Cash, *On the derivation of high order symmetric MIRK formulae with interpolants for solving two-point boundary value problems*, New Zealand Journal of Mathematics, 29 (2000), pp. 129-150.

[4] J.R. Cash, G. Moore, and R.W. Wright, *An automatic continuation strategy for the solution of singularly perturbed linear two-point boundary value problems*, J. Comput. Phys. 122 (1995), pp. 266-279.

[5] F. Mazzia and I. Sgura, *Numerical Approximation of Nonlinear BVPs by means of BVMs*, Appl. Numer. Math., 42 (2002), pp. 337-352.

[6] J.R. Cash and A. Singhal, *High order methods for the numerical solution of two-point boundary value problems*, BIT, 22 (1982), pp. 184-199.

[7] P.H. Muir and B. Owren, *Order barriers and characterisations for continuous mono-implicit Runge-Kutta schemes*, M. Comp., 61 (1993), pp. 675-691.

[8] S. Pruess, *Interpolation schemes for collocation solutions of two-point boundary value problems*, SIAM J. Sci. Comput., 7 (1986), pp. 322-333.

[9] 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 (2004), pp. 1749-1763.

[10] J.R. Cash and R.W. Wright, *Continuous extensions of deferred correction schemes for the numerical solution of nonlinear two point boundary value problems*, Appl. Numer. Math., 28 (1998), pp. 227-244.

[11] J.R. Cash and H.H.M. Silva, *Iterated deferred correction for linear two-point boundary value problems*, Comp. Appl. Math.,Vol 15, 1 (1996), pp. 55-75.

[12] J. R. Cash and M. H. Wright. Bvp software. `http://www.ma.ic.ac.uk/~jcash/BVP_software/readme.html`.

4 Acknowledgement

TJA would like to acknowledge financial support from the Kuwait Foundation for the Advancement of Sciences (KFAS) and also from the Mathematics Department of the University of Cambridge.

Appendix

For $\alpha = \frac{1}{3}, \beta = \frac{2}{3}$, the coefficients in (2.2) are

$$\begin{aligned}
A_{I6}(\omega) &= (54\omega^3 - 27\omega^2 + 2\omega + 1)(\omega - 1)^2, \\
B_{I6}(\omega) &= -\omega^2(-30 + 110\omega - 135\omega^2 + 54\omega^3), \\
C_{I6}(\omega) &= \frac{\omega}{4}(27\omega^2 - 18\omega + 4)(\omega - 1)^2, \\
D_{I6}(\omega) &= \frac{\omega^2}{4}(\omega - 1)(27\omega^2 - 36\omega + 13), \\
E_{I6}(\omega) &= \frac{27}{4}\omega^2(3\omega - 1)(\omega - 1)^2, \\
F_{I6}(\omega) &= \frac{27}{4}\omega^2(3\omega - 2)(\omega - 1)^2,
\end{aligned}$$

The local truncation error of the interpolant (2.2) is

$$\frac{\omega^2}{2160}(3\omega^2 - 3\omega + 1)(\omega - 1)^2 y_n^{(vi)}(x)|_{x=\epsilon}, \; x_n \leq \epsilon \leq x_{n+1}.$$

The coefficients in (2.3) for $\alpha = 1/5, \beta = 2/5, \delta = 3/5, \eta = 4/5$ are

$$\begin{aligned}
A_{I8}(\omega) &= \frac{(\omega-1)^2}{11}(7500\omega^5 - 11250\omega^4 + 5700\omega^3 - 975\omega^2 + 22\omega + 11)\\
B_{I8}(\omega) &= \frac{-\omega^2}{11}(-1008 + 7672\omega - 23625\omega^2 + 35700\omega^3 - 26250\omega^4 + 7500\omega^5)\\
C_{I8}(\omega) &= \frac{\omega}{792}(\omega-1)^2(35625\omega^4 - 56875\omega^3 + 32575\omega^2 - 7725\omega + 792)\\
D_{I8}(\omega) &= \frac{\omega^2}{792}(\omega-1)(35625\omega^4 - 85625\omega^3 + 75700\omega^2 - 29300\omega + 4392)\\
E_{I8}(\omega) &= \frac{125}{792}\omega^2(\omega-1)^2(1125\omega^3 - 1550\omega^2 + 668\omega - 72)\\
F_{I8}(\omega) &= \frac{125}{396}\omega^2(\omega-1)^2(375\omega^3 - 700\omega^2 + 439\omega - 90)\\
G_{I8}(\omega) &= \frac{125}{396}\omega^2(\omega-1)^2(375\omega^3 - 425\omega^2 + 164\omega - 24)\\
H_{I8}(\omega) &= \frac{125}{792}\omega^2(\omega-1)^2(1125\omega^3 - 1825\omega^2 + 943\omega - 171)
\end{aligned}$$

The local truncation error of the interpolant (2.3) is

$$\frac{\omega^2}{75600000}(1875\omega^4 - 3750\omega^3 + 2875\omega^2 - 1000\omega + 144)(\omega-1)^2 y_n^{(viii)}(x)|_{x=\epsilon},$$
$$x_n \le \epsilon \le x_{n+1}.$$

If we choose $\alpha = 1/3, \beta = 2/3$, the coefficients in (3.3) are

$$\begin{aligned}
A_{S8}(\omega) &= -(540\omega^4 - 270\omega^3 + 6\omega^2 + 3\omega + 1)(\omega-1)^3\\
B_{S8}(\omega) &= \omega^3(280 - 1365\omega + 2436\omega^2 - 1890\omega^3 + 540\omega^4)\\
C_{S8}(\omega) &= -\frac{1}{4}\omega(351\omega^3 - 195\omega^2 + 12\omega + 4)(\omega-1)^3\\
D_{S8}(\omega) &= -\frac{1}{4}\omega^3(\omega-1)(351\omega^3 - 858\omega^2 + 675\omega - 172)\\
E_{S8}(\omega) &= -\frac{1}{2}\omega^2(3\omega-1)^2(\omega-1)^3\\
F_{S8}(\omega) &= \frac{1}{2}\omega^3(3\omega-2)^2(\omega-1)^2\\
G_{S8}(\omega) &= -\frac{81}{4}\omega^3(9\omega-4)(\omega-1)^3\\
H_{S8}(\omega) &= -\frac{81}{4}\omega^3(9\omega-5)(\omega-1)^3
\end{aligned}$$

The local truncation error of the interpolant (3.3) is

$$\frac{1}{1088640}\omega^3(27\omega^2 - 27\omega + 8)(\omega-1)^3 y_n^{(viii)}(x)|_{x=\epsilon}\ x_n \le \epsilon \le x_{n+1}.$$