Opuscula Mathematica • Vol. 26 • No. 2 • 2006

Jeff R. Cash

THE NUMERICAL SOLUTION OF NONLINEAR TWO-POINT BOUNDARY VALUE PROBLEMS USING ITERATED DEFERRED CORRECTION – A SURVEY

Abstract. The use of iterated deferred correction has proved to be a very efficient approach to the numerical solution of general first order systems of nonlinear two-point boundary value problems. In particular the two high order codes TWPBVP.f, based on mono-implicit Runge–Kutta (MIRK) formulae, and TWPBVPL.f based on Lobatto Runge–Kutta formulae as well as the continuation codes ACDC.f and COLMOD.f are now widely used. In this survey we describe some of the problems involved in the derivation of efficient deferred correction schemes. In particular we consider the construction of high order methods which preserve the stability of the underlying formulae, the choice of the mesh choosing algorithm which is based both on local accuracy and conditioning, and the computation of continuous solutions.

Keywords: Deferred Correction, boundary value problems, conditioning, mesh selection.

Mathematics Subject Classification: 65L05, 65L06, 65L20.

1. INTRODUCTION

In recent years some powerful codes have been developed for the numerical solution of the general first order system of nonlinear two-point boundary value problems

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

There are of course many important types of boundary value problems which do not fall into this class, such as eigenvalue problems, problems with integral constraints, problems posed on semi-infinite intervals, singular problems etc. but for the purpose of this paper we will confine our attention to (1.1). However there has been a lot of

work carried out on singular problems where special attention has to be paid to the singular point. In particular Weinmuller *et al.* ([31]) have considered the solution of boundary value problems with a singularity of the first kind. They consider a deferred correction based on the implicit Euler method and show that their scheme retains the classical order of convergence. The interested reader is referred to ([31]) and related work. In what follows we consider some boundary value codes. In particular we mention the collocation codes COLSYS ([1]) and COLNEW ([12]), the defect correction code MIRKDC of Enright and Muir ([2]), the top order methods of Trigiante and his co-workers ([20, 24, 5]) and the deferred correction codes of Cash ([3, 4]). The striking thing about these codes is that the way in which they attempt to find the solution of (1.1) is, in each case, very different as we will show.

In this survey we consider the solution of (1.1) using iterated deferred correction. We will explain some of the important components which go to make up a deferred correction scheme and we will point out the major differences between the deferred correction approach and others that have been proposed. One of the codes that has been popular for some time for the numerical solution of boundary value problems is COLSYS ([1]). This code is based on the widely used technique of polynomial collocation. In this approach the solution of (1.1) is approximated by a piecewise polynomial function $P(x)$ defined on a discrete mesh π. The coefficients of this approximating polynomial are uniquely determined by requiring P to satisfy the boundary conditions and also to satisfy the differential equation at certain points in each subinterval of the mesh (the collocation points). The link between collocation and implicit Runge–Kutta methods for initial value problems is well known ([13]) and so it is not surprising that the solution of (1.1) in the initial value case using collocation on Gauss points (as used by COLSYS) is equivalent to solving (1.1) using Gauss Runge–Kutta methods. It is natural to ask, in view of what has been done for initial value problems, whether it would be more efficient to solve (1.1) using a different class of Runge–Kutta methods since Gauss methods although having excellent accuracy and stability properties are expensive to implement. One class of Runge–Kutta methods which has been found to be particularly efficient for the numerical solution of boundary value problems, are mono-implicit Runge–Kutta (MIRK) [6] methods. The important property of this particular class of methods is that for initial value problems they are implicit only in the single unknown y_{n+1} while for boundary value problems they are implicit only in y_n and y_{n+1}. Although MIRK formulae are standard Runge–Kutta methods, and so can be expressed using the well known Butcher tableau notation, it is much more illuminating to write them in the special form

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i \, f(x_n + c_i \, h, \, Y_i) \tag{1.2}$$

$$Y_i \;\; = (1 - v_i) \, y_n + v_i \, y_{n+1} + h \sum_{j=1}^{s} x_{ij} \, f(x_n + c_j \, h, \, Y_j), \quad i = 1, \ldots, s.$$

The formula (1.2) can now be conveniently written in the following tableau form

$$\begin{array}{c|c|cccc} c_1 & v_1 & x_{11} & x_{12} & \dots & x_{1s} \\ c_2 & v_2 & x_{21} & x_{22} & \dots & x_{2s} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ c_s & v_s & x_{s1} & x_{s2} & \dots & x_{ss} \\ \hline & & b_1 & b_2 & \dots & b_s \end{array} \tag{1.3}$$

This characterisation of MIRK formulae in terms of the matrix X helps considerably in their derivation and shows clearly their special structure. We illustrate this by considering the well known Clippinger–Dimsdale formula (this is also the Lobatto IIIA formula of order 4) written using the well known Butcher notation [19]:

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{5}{24} & \frac{1}{3} & \frac{-1}{24} \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \tag{1.4}$$

It is clear that this formula can be rewritten in the form

$$y_{n+1} - y_n = \frac{h}{6}(k_1 + 4k_2 + k_3) \tag{1.5}$$

where

$$\begin{aligned} k_1 &= f(x_n, y_n), \\ k_2 &= f\left(x_{n+\frac{1}{2}}, \frac{y_n + y_{n+1}}{2} - \frac{h}{8}(k_3 - k_1)\right), \\ k_3 &= f(x_{n+1}, y_{n+1}). \end{aligned} \tag{1.6}$$

If we now introduce the notation used in (1.2) we can rewrite this equation as

$$\begin{array}{c|c|ccc} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{8} & \frac{-1}{8} & 0 \\ \hline & & \frac{1}{6} & \frac{1}{6} & \frac{2}{3} \end{array} \tag{1.7}$$

We note that the matrix X is lower triangular and it is this which gives MIRK formulae their desirable property of being implicit in the single unknown y_{n+1} for initial value problems. We also note that the link between the modified notation which uses the matrix X and the standard Butcher notation is given by the identity

$$A = X + vb^T. \tag{1.8}$$

In what follows we will consider a particularly efficient approach to the implementation of Runge–Kutta methods in a deferred correction framework. In this approach, which was originally proposed by Fox [14], we need to define two operators which will be used to characterise our deferred correction method. The first operator, which we will denote by ϕ, computes a cheap low order numerical approximation to the solution of (1.1) while the second, denoted by ψ computes an estimate of the local error in ϕ. Using these two operators our deferred correction formula can be written in the form

$$\phi(\eta) = 0, \tag{1.9}$$

$$\phi(\bar{\eta}) = \psi(\eta). \tag{1.10}$$

A general framework for proving accuracy results for deferred correction schemes of the form (1.9), (1.10) was given in an influential paper by Skeel [15]. In what follows we present his main theorem.

Consider the approximate numerical solution of (1.1) on a mesh

$$\pi\colon\ a = x_1 < x_2 < \ldots < x_{N+1} = b.$$

Denote by Δy the restriction of the continuous solution $y(x)$ to the finite grid π. Then we have the following theorem:

Theorem 1.1. *Let ϕ be a stable numerical method and assume that the following conditions hold for the deferred correction* (1.9), (1.10)*:*

(i) $\|\eta - \Delta y\| = O(h^p)$,

(ii) $\|\psi(\Delta y) - \phi(\Delta y)\| = O(h^{r+p})$,

(iii) $\psi(\Delta w) = O(h^r)$,

for arbitrary functions w having at least r continuous derivatives. Here $\|.\|$ is a suitable finite norm defined in [15]*. If $\phi(\bar{\eta}) = \psi(\eta)$ then*

$$\|\bar{\eta} - \Delta y\| = O(h^{r+p})\,.$$

The question now is how to define the two operators ϕ and ψ. In particular we want the scheme (1.9), (1.10) to be of high order while at the same time being cheap to implement. In what follows we will adopt a particular form of deferred correction which was first proposed by Fox [14] and later refined by Lindberg [16]. The proposal was as follows: Let ϕ_i, ϕ_j be two Runge–Kutta formulae of order i and j respectively where $i < j$. Consider the algorithm defined by

$$\phi_i(\eta) = 0, \tag{1.11}$$

$$\phi_i(\bar{\eta}) = -\phi_j(\eta), \tag{1.12}$$

which does of course lie in the general framework defined by (1.9), (1.10). It is immediately clear that if we make these choices for ϕ and ψ then the first two

conditions of Skeel's theorem are trivially satisfied. It is the third condition of Skeel's theorem that it is not straightforward to satisfy. To explain how to satisfy this third equation we consider the case $p = 4$, $r = 2$ in Theorem 1 and let ϕ_4, ϕ_6 be two Runge–Kutta formulae of order 4 and 6 respectively. Consider the algorithm defined by

$$\phi_4(\eta) = 0, \tag{1.13}$$

$$\phi_4(\bar{\eta}) = -\phi_6(\eta). \tag{1.14}$$

If we now appeal to Skeel's theorem it follows that $\bar{\eta}$ is an order 6 approximation to Δy proving that

$$\phi_4(\Delta w) - \phi_6(\Delta w) = O(h^2) \tag{1.15}$$

for arbitrary functions Δw having the necessary smoothness properties. An analysis of condition (1.15) was carried out by Cash and Silva [17] in the case where ϕ_4 and ϕ_6 are Lobatto formulae. They showed that when this choice of Runge–Kutta methods was made (1.15) was not satisfied and an explicit computation showed that the deferred correction scheme (1.13), (1.14) only raised the order of the overall scheme from 4 to 5. This may seem a little surprising since both ϕ_4 and ϕ_6 are symmetric Runge–Kutta formulae. However a closer examination reveals that the usual definition of symmetry is inappropriate if we wish to ensure that (1.15) holds. What we need to do is to redefine our Runge–Kutta formulae so that they are specially tuned to boundary value problems. The key to this is to note that it is not

$$\phi_4(\bar{\eta}) = -\phi_6(\bar{\eta}), \tag{1.16}$$

which we require to be symmetric in the scheme (1.13),(1.14) but instead we require

$$\phi_4(\bar{\eta}) = -\phi_6(\eta) \tag{1.17}$$

to be symmetric. This means that we need to modify our formulae so that they are appropriate for solving boundary value problems and this we now do.

To motivate our new definition of symmetry we consider the standard s-stage Runge–Kutta formula

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i \tag{1.18}$$

$$k_i = f\left(x_n + c_i h,\, y_n + h \sum_{j=1}^{s} a_{ij} k_j\right), \quad 1 \le i \le s\,.$$

We rewrite this in the modified form

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i \tag{1.19}$$

$$k_i = f\left(x_n + c_i h,\ \frac{y_n + y_{n+1}}{2} + h\sum_{j=1}^{s}(a_{ij} - b_j/2)k_j\right).$$

The Runge–Kutta formula (1.19) is defined to be BV-symmetric if

$$c_i = 1 - c_{s+1-i}, \quad b_i = b_{s+1-i}, \quad 1 \le i \le s \tag{1.20}$$

$$a_{ij} - \frac{1}{2}b_j = \frac{1}{2}b_{s+1-j} - a_{s+1-i,s+1-j}, 1 \le i, j \le s. \tag{1.21}$$

This definition is more transparent if we express it in terms of the coefficients of the modified Runge–Kutta formula (1.19). Thus if we rewrite (1.19) in the form

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i \tag{1.22}$$

$$k_i = f\left(x_n + c_i h,\ \frac{y_n + y_{n+1}}{2} + h\sum_{j=1}^{s}\hat{a}_{ij}k_j\right)$$

then the conditions for BV-symmetry are (1.20) and

$$\hat{a}_{ij} = -\hat{a}_{s+1-i,s+1-j}, \quad 1 \le i, j \le s.$$

In what follows will will assume that the ϕ_i are expressed in the form (1.19). We can now give the following lemma:

Lemma 1.1.

(i) *The deferred correction scheme* (1.13), (1.14) *is symmetric if* ϕ_4 *and* ϕ_6 *are BV-symmetric.*

(ii) *A symmetric deferred correction scheme satisfies condition* (1.15).

(iii) *The Runge–Kutta scheme* (1.19) *is BV-symmetric if and only if the underlying formula* (1.18) *is symmetric.*

The important practical implication of this lemma is that if we take two standard symmetric Runge–Kutta formulae ϕ_4 and ϕ_6, of order 4 and 6 respectively, and rewrite them in the modified form (1.19), then the deferred correction scheme (1.13), (1.14) based on these modified formulae is of order 6 and this has important implications for the construction of our deferred correction schemes.

There are several ways in which the approach defined by (1.13), (1.14) can be extended to derive higher order deferred correction algorithms. One approach would be to add on extra deferred correction stages. If, for example, we add on one extra stage to (1.13), (1.14) then we obtain a three stage algorithm of the form

$$\begin{aligned}
&\phi_4(\eta) = 0\\
&\phi_4(\bar{\eta}) = -\phi_6(\eta)\\
&\phi_4(\bar{\bar{\eta}}) = -\phi_6(\eta) - \phi_8(\bar{\eta}).
\end{aligned} \tag{1.23}$$

The two deferred correction codes TWPBVP.f and TWPBVPL.f are based on deferred correction formulae of precisely this form. In both cases the final solution is of order 8. It is of course possible to use even more stages in our deferred correction schemes. When the ϕ appearing in (1.23) are Lobatto formulae this is straightforward since high order Lobatto formulae are relatively easy to construct. For MIRK formulae, however, this is more difficult since high order MIRK formulae are not so straightforward to derive (see however [3]). However a tenth order MIRK formula has been derived by Capper ([25]) and a tenth order Lobatto formula has been analysed by Bashir-Ali ([26]). It would be of interest to see how these high order codes perform when compared with existing ones especially when a high degree of accuracy is required. Another important point to note concerning our deferred correction schemes is that we always start with order four. However this is a heuristic choice which can easily be changed. If, instead, we were to start with order 6, and then our eighth order deferred correction would have 2 stages, we would expect the new 2-stage formula to be more accurate and stable but more expensive to implement than conventional 3-stage formulae which start at order 4. Here the term more accurate means smaller error constants — the order of the two approaches would both be exactly the same. The main computational cost of starting at order 6 would be in solving the nonlinear algebraic equation $\phi_6(\eta) = 0$. However it would be of interest to investigate which of these two approaches is superior and this is an area for future research.

All of the deferred correction schemes that we have considered so far raise the order of accuracy by 2 for each deferred correction. An obvious question to ask is whether we can get more rapid convergence i.e. can we find schemes of the form described in Theorem 1 with $r > 2$? This question was answered in the affirmative by Van Daele and Cash [18]. They showed that it is possible to construct a deferred correction scheme of the general form described in Theorem 1 with $r = p$ and in particular they derived a two-stage "superconvergent" deferred correction scheme of the form

$$\phi_4(\eta) = 0, \tag{1.24}$$

$$\phi_4(\bar{\eta}) = -\phi_8(\eta), \tag{1.25}$$

where $\bar{\eta}$ is of order 8. Although this marked a big theoretical breakthrough in the analysis of deferred correction schemes these superconvergent methods do have some computational drawbacks compared with the more conventional formula (1.11), (1.12) with $i = 4$, $j = 6$. The particular problem experienced by (1.24), (1,25) concerns local error estimation. For deferred correction schemes of the form (1.23) we can control the error in

$$\|\bar{\eta} - \bar{\bar{\eta}}\| \tag{1.26}$$

whereas for (1.24), (1.25) there is no obvious local error estimate apart from

$$\|\bar{\eta} - \eta\|. \tag{1.27}$$

This error estimate has proved to be unsatisfactory because it estimates the error in a fourth order, rather than a sixth order, solution. Mainly for this reason it is still not clear whether it is more efficient to use a standard deferred correction formula such as (1.23) or a superconvergent scheme of the form (1.24), (1.25) and this is an area for future research.

We mentioned earlier that a particularly attractive class of formulae to use in our deferred correction schemes are MIRK formulae since the deferred corrections are explicit and therefore cheap to compute. However practical experience has shown that iterated deferred correction based on MIRK formulae does not perform very well on very stiff problems. This is of course to be expected since the MIRK deferred corrections are explicit and so there is a loss of stability when they are applied. The way to deal with this, if we are to handle stiff equations in an efficient and reliable manner, is to use a completely different class of Runge–Kutta methods for the ϕ_i and to choose these methods so that they have the excellent stability required to deal with stiff equations. In what follows we will discuss the stability of various Runge–Kutta formulae used in a deferred correction framework. A stability analysis for MIRK formulae has been carried out by Bashir-Ali ([26]). Applying the deferred correction scheme (1.11)(1.12) with $i = 4, j = 6$, and where the ϕ_i are MIRK methods, to the scalar test equation

$$y' = \lambda y \tag{1.28}$$

we find that we have the asymptotic relationship

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

and so it does not have the stability necessary to deal efficiently with stiff problems. We note in particular that this stability function is unbounded as $h\lambda \to \infty$ and this is entirely due to the explicitness of the deferred corrections. In order to maintain highly stable deferred correction schemes it is necessary to make the deferred corrections implicit and to achieve this we must define the ϕ_i appearing in (1.13), (1.14) to be a different class of implicit Runge–Kutta formulae. Various classes of implicit Runge–Kutta formulae have been investigated in Bashir-Ali ([26]) and one of the most efficient classes of formulae, based on accuracy, stability and ease of implementation, are the Lobatto formulae. The stability of Lobatto formulae in a deferred correction framework was investigated in some detail by Cash and Silva ([17]). One of their main findings was that when we use the deferred correction scheme (1.11), (1.12) with $i = 4$, $j = 6$, and where the ϕ_i are now Lobatto Runge–Kutta formulae of order i, and we apply this to the scaler test equation (1.28), we find

$$\frac{y_{n+1}}{y_n} \sim 1 \quad \text{as} \quad h\lambda \to \infty. \tag{1.29}$$

In view of this our expectation is that the code TWPBVPL.f, based on Lobatto formulae, will be much better than TWPBVP.f, which is based on MIRK formulae for dealing with stiff problems and this expectation has been borne out in practice.

In view of these findings, for non-stiff and mildly stiff problems we use the code TWPBVP.f while for stiff problems we use TWPBVPL.f. Both of these codes are available from the authors' web page ([28]).

2. MESH CHOOSING ALGORITHMS

Having chosen the classes of formulae that we will use in our deferred correction codes, the next important problem we need to consider is that of local error estimation and mesh refinement. One of the main reasons why deferred correction schemes can be implemented so efficiently is that they have a local error estimate already available. If, for example, we consider the deferred correction scheme (1.23) then the error estimate

$$\|\bar{\eta} - \bar{\bar{\eta}}\| \tag{2.1}$$

is an asymptotically (as $h \to 0$) correct error estimate of the local error in $\bar{\eta}$. Since we know that $\bar{\eta}$ is a $O(h^6)$ approximation to the true solution we can use this error estimate to refine the grid in the usual way. However practical experience on very stiff problems has shown that the approach does not always provide a good mesh refinement algorithm. In particular it has often been observed that a mesh selection algorithm subtracts out points in one mesh refinement only to bring these points back into the mesh during the next refinement. Subsequent analysis has shown that a robust mesh refinement algorithm must take account of the conditioning of the discrete problem as well as on the magnitude of the local error estimate. In particular it is important to choose the mesh so that the problem remains well conditioned throughout the computation. Some interesting numerical results which demonstrate this fact very clearly have been given by Mazzia and Trigiante [20]. In particular they show that a hybrid mesh selection algorithm, which depends in some sense on a combination of local error estimation and conditioning, can be much more effective than one which relies solely on local error estimation. The reason for this is clear as we now explain. Numerical algorithms compute an estimate of the local error on the assumption that the problem being solved is well conditioned. In particular if, for a well conditioned problem, the local error is in some sense small then the global error will also be correspondingly small. Conversely if the problem being solved is not well conditioned then a uniformly small local error can lead to a large global error. In view of this it can be argued that when solving a two-point boundary value problem it is vital to be able to obtain an estimate of the condition number of the discrete problem if we are to have any confidence in the accuracy of the solution that is obtained. An extreme example of the dangers of dealing with a badly conditioned problem has been given by Shampine and Muir [23]. They give an example where the MATLAB code bvp4c is used to solve Bratu's problem (see later in this section for a definition of this problem) for a value of the parameter, $\lambda = 3.55$, for which there is no solution. Rather unexpectedly the MATLAB code produces a perfectly reasonable looking solution which is of course totally spurious. Shampine and Muir argue, however, that this could have been anticipated by looking at the behaviour

of the conditioning of the problem. Their proposal is to compute the condition number so as to obtain an estimate of the global error after a discrete solution has been computed. If this condition number is large then they issue a warning to the user. However in the present paper we have a rather more ambitious goal in that we incorporate our conditioning estimate into the mesh refinement algorithm and choose the mesh using an error estimate which is a combination of accuracy and conditioning.

In what follows we will describe our mesh selection strategy in some detail. This will depend on a combination of the local error estimate (2.1) and on an estimate of three parameters which characterise the conditioning of the continuous problem. The approach which we will adopt follows very closely that given by Mazzia and Trigiante for the TOM code [20]. To explain this approach we consider the linear system of two-point boundary value problems

$$\begin{aligned} y' &= A(x)y + q(x), \qquad y \in R^m, \\ B_a y(a) &+ B_b y(b) = \eta, \end{aligned} \tag{2.2}$$

where $A(x)$, B_a, $B_b \in R^{m\times m}$. We assume that the BVP (2.2) has a unique solution $y(x)$. Following [21] this solution can be expressed as

$$y(x) = Y(x)Q^{-1}\eta + \int_a^b G(x,s)q(s)ds,$$

where $Y(x)$ is a fundamental solution, $G(x,s)$ is the Green's function and $Q = B_aY(a) + B_bY(b)$ is assumed to be nonsingular.

We now consider a perturbation of this problem which will in turn produce a perturbed solution $\hat{y}$ which satisfies

$$\begin{aligned} \hat{y}' &= A(x)\hat{y} + q(x) + \delta q(x) \qquad \hat{y} \in R^m, \\ B_a\hat{y}(a) &+ B_b\hat{y}(b) = \eta + \delta\eta, \end{aligned} \tag{2.3}$$

where small changes have been made to the boundary conditions as well as to the differential equation to produce a perturbed solution $\hat{y}(x)$. It can be shown [20] that we can bound the perturbation $\|\delta y(x)\|_\infty = \|\hat{y}(x) - y(x)\|_\infty$ in the solution in the following way

$$\max_{a\le x\le b} \|\hat{y}(x) - y(x)\|_\infty \le \kappa_1 \|\delta\eta\|_\infty + \kappa_2 \max_{a\le x\le b} \|\delta q(x)\|_\infty .$$

We can now define the conditioning constants κ_1 and κ_2 appearing in the above expression as

$$\kappa_1 = \max_{a\le x\le b} \left\|Y(x)Q^{-1}\right\|_\infty , \tag{2.4}$$

$$\kappa_2 = \sup_x \int_a^b \|G(x,s)\|_\infty \, ds.$$

The two constants κ_1 and κ_2 are related to the perturbation in the boundary conditions and in the differential equation respectively. A single conditioning constant can be defined by:

$$\kappa = \max_{a\le x\le b}(\left\|Y(x)Q^{-1}\right\|_\infty + \int_a^b \|G(x,s)\|_\infty \, ds), \tag{2.5}$$

since

$$\max_{a\le x\le b} \|\hat{y}(x) - y(x)\|_\infty \le \kappa \max(\|\delta\eta\|_\infty, \max_{a\le x\le b} \|\delta q(x)\|_\infty).$$

If κ is large the problem is usually considered ill conditioned, otherwise the problem is considered well conditioned. It is the parameter κ that gives complete information about the conditioning. However, when we deal with boundary value problems where the boundary conditions are appropriate for handling the decreasing and the increasing modes, that is there is a dichotomy present, the information provided by κ_1 is sufficient to classify the problem. In fact, in this case, it is easy to bound the conditioning constant κ_2, and therefore κ, in terms of κ_1(see [21] sec. 3.4.2 and 3.4.3 for a definition of dichotomy and its relation to well conditioning and also for its relation to the concept of absolute stability for initial value problems). With this in mind, we focus our attention on the perturbation $\delta y(x)$ in the solution resulting from a perturbation $\delta\eta$ of the boundary conditions. From our previous analysis it follows immediately that if $\delta q(x) = 0$

$$\|\delta y(x)\|_\infty \le \|Y(x)Q^{-1}\|_\infty \|\delta\eta\|_\infty.$$

We now define the 1-norm of a vector δy in $C([a,b])$ as

$$\|\delta y\|_1 = \frac{1}{b-a} \int_a^b \|\delta y(x)\|_\infty dx.$$

Using our definitions of the 1- and ∞-norms we obtain the two-upper bounds

$$\|\delta y\|_\infty \le \kappa_1 \|\delta\eta\|_\infty, \qquad \|\delta y\|_1 \le \gamma_1 \|\delta\eta\|_\infty$$

where κ_1 is defined in (2.4) and

$$\gamma_1 = \frac{1}{b-a} \int_a^b \|Y(x)Q^{-1}\|_\infty dx. \tag{2.6}$$

The difficulty which now faces us is how to compute an estimate of the condition number of the discrete problem. When solving for the discrete solution it is necessary to solve a linear system of algebraic equatons of the form

$$My = b. \tag{2.7}$$

Intuitively we would expect that the conditioning of the discrete problem will depend in some way on M^{-1} and indeed that is true. We can compute an estimate of the norm of M^{-1} using an algorithm due to Higham and Tisseur [22]. An algorithm for examining the conditioning of the discrete problem, which depends on three of the conditioning constants described earlier, is given in [29]. Using this we are able to define an explicit monitor function which fulfils our aim of depending both on local error estimates and on the conditioning of the problem. This analysis is too lengthy to give here and the interested reader is referred to [29]. Based on this analysis we have developed an eighth order deferred correction scheme with an improved mesh choosing algorithm based on accuracy and conditioning. This code is called TWPBVPC.f and is available from the authors' web page [28]. Numerical results given in [29] clearly show the improved performance of the code when conditioning is included and we end this section by giving a set of numerical results which demonstrates this.

An interesting nonlinear equation which we will discuss is Bratu's problem

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

that arises in a model of spontaneous combustion. This problem was included in [23] as an example for which the computation of the conditioning constant κ could give information about the quality of a solution. In fact this problem has two solutions for $0 \leq \lambda < \lambda^* = 3.51383\ldots$, one solution when $\lambda = \lambda^*$ and no solution when $\lambda > \lambda^*$. In [23] it was found that the MATLAB solver bvp4c has no problem in solving the BVP when $\lambda = 3.45$; TWPBVPC.f has a similar behaviour. If we solve this problem using TWPBVPC.f with conditioning, and with a tolerance of 10^{-3} imposed on both y and y', with an initial mesh of 16 equally spaced mesh points and initial guesses of zero, we obtain a solution without changing the mesh, the conditioning constants are $\kappa_1(\pi) = 1.6e1$, $\gamma_1(\pi) = 1.2e1$, $\kappa(\pi) = 2.4e1$. These conditioning constants are discrete approximations to the conditioning constants of the continuous problem defined earlier. In particular an approximation of the conditioning parameter κ is obtained by computing $\kappa(\pi) = \|M^{-1}\|_\infty$ which is estimated using the Higham and Tisseur algorithm. For an explanation of the other condition numbers together with an algortihm for computing them the reader is referred to [29]. In contrast to the behaviour of bvp4c which gives a 'solution' for $\lambda = 3.55$ when none exists the code TWPBVPC.f fails to give a solution for this value of λ and this in agreement with the theory. In fact TWPBVPC.f fails to give a solution for all $\lambda > \lambda^*$ and remarkably this allows λ^* to be found very quickly. To see how the conditioning parameters change when λ approaches λ^*, we solve the problem with TWPBVPC.f using different values of λ (see Tab. 1). All the three conditioning parameters grow as we get closer to λ^*

and this is a sign that something important is happening at this point. The approach of Shampine and Muir is to issue a warning to the user that the condition number is increasing. However in the approach of [29] this condition number is used in choosing the mesh. Further numerical results presented in [29] show the excellent performance of this new mesh selection strategy. As a final note we repeat our observation that if we are solving a two-point boundary value problem then a small local error in the solution does not necessarily lead to a small global error. To assess the quality of the final solution we must have an estimate of the condition number and we feel that the next generation of codes will follow our approach and routinely provide a condition number estimate along with the final solution.

Table 1. Conditioning parameters and final mesh for Bratu's roblem, $tol(ncomp_1) = tol(ncomp_2) = 10^{-3}$

	TWPBVPC with cond			
λ	$\kappa(\pi)$	$\kappa_1(\pi)$	$\gamma_1(\pi)$	N+1
3.5	5.4e1	3.7e1	2.8e1	16
3.51	1.0e2	7.1e1	5.4e1	16
3.513	2.2e2	1.5e2	1.2e2	16
3.5138	9.5e2	6.5e2	5.0e2	16
3.51383	4.9e3	3.3e3	2.5e3	31
3.513831	7.0e3	4.8e3	3.6e3	31
3.5138317	1.9e4	1.3e4	1.0e4	31
3.51383179	9.6e4	6.5e4	4.9e4	31

3. COMPUTATION OF CONTINUOUS SOLUTIONS

Finally in this section we consider the problem of deriving continuous solutions to two-point boundary value problems. It has been pointed out by Pruess [8] that the problem of defining an interpolant is much more difficult than at first seems to be the case. In what follows we wish to define an interpolant which is continuous for both y and y' at all mesh points, one which has an accuracy comparable to that of the discrete method, one that is efficient in the sense that it does not require the computation of too many extra derivatives and one such that the coefficients of the interpolant remain suitably bounded over the whole range of the integration. As already explained, the two codes COLSYS and MIRKDC automatically provide continuous solutions and the user has no choice as to whether he is given a continuous or discrete solution. In the case of COLSYS a continuous solution is provided because the solution is approximated by a piecewise continuous polynomial and in the case of MIRKDC a continuous solution is required because the code seeks to control the residual

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

However the extra cost of providing a continuous solution is relatively high and it may be that the user does not require a continuous solution at all. It would be better computationally to give the user the option of whether or not he wants a continuous solution as is done in the case of initial value problems and to compute this solution a posteriori after the discrete solution has been computed. One possibility is that the user may wish to compute the solution only at isolated 'off step' points. The most efficient way of doing this is to choose meshes so that these isolated points are contained in all meshes that are chosen. This is trivial to do since all codes provide the option of inserting fixed points in all meshes 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 where the function $y(x)$ or its derivative passes through zero. A big advantage of the deferred correction codes (and also of the TOM code) is that they do compute the continuous solution a posteriori i.e. after a discrete solution has been computed to the required degree of precision, if indeed a continuous solution is wanted at all.

The question now is how to derive a visually pleasing, high quality interpolant in an efficient way without the need to compute too many extra function evaluations. One approach would be to compute function evaluations over several adjacent grid points. However numerical experience has shown that symmetry is a very important property for interpolants to possess and they lose this near the end of the range of integration and on highly non-uniform meshes. In view of this we seek to make our interpolants symmetric, each interpolant is defined by data evaluated over just one mesh interval and this is consistent with the idea of a boundary value problem having no direction of integration.

The first problem we consider is that of deriving interpolants for use with MIRK formulae and in particular for use with the code TWPBVP.f. There are two sets of data available to define an interpolant. The first is data that has been used to compute the discrete solution. However this data often has low stage order and so is difficult to use in an interpolant. The other data available is the extra function evaluations that are computed, after the discrete solution has been defined, for the purpose of constructing the interpolant. It has proved to be difficult to define an interpolant for the original eighth order MIRK formula developed by Cash and Singhal [6] because many of the stages appearing in that formula have low stage order and so can not be easily used in the interpolant. The problem of defining an efficient high quality interpolant for MIRK formulae was solved by Cash and Moore [9]. By computing just three extra derivatives per mesh interval they were able to obtain an eighth order interpolant and extensive numerical testing has shown that this interpolant performs very well on a wide class of non stiff problems. In fact we feel that these interpolants perform so well that the problem of computing interpolants for non-stiff problems is now just about solved. To illustrate this we consider the numerical solution of the following singular perturbation problem, which we will refer to as Problem 2,

$$\epsilon y'' - xy = 0, \qquad y(-1) = y(1) = 1 \tag{3.2}$$

for a range of values of ϵ using the algorithm of ([9]). To obtain our numerical results we put an error tolerance of 10^{-8} on y and we then computed a sixth order and an eighth order interpolant as defined in ([9]). The results obtained are given in Table 2.

Table 2. The interpolation using MIRK8, $TOL = 10^{-8}$

ϵ	Err Sol	Nint	Err $y66$	Err $y88$
	Problem 2			
1	7.320d-14	17	5.104d-10	1.028d-13
10^{-1}	1.392d-11	33	1.000d-08	1.433d-11
10^{-2}	6.011d-11	101	2.721d-08	6.011d-11
10^{-3}	1.260d-11	409	5.155d-09	1.260d-11
10^{-4}	1.469d-11	697	2.784d-09	1.469d-11

For each value of ϵ the heading *ErrSol* gives the maximum error in the discrete problem at any grid point while *Nint* gives the number of points in the final mesh. To compute the error, $Erry66$, in the sixth order interpolant and $Erry88$ in the eighth order interpolant we computed the error between the interpolants and the true solution at the points x_i, $x_{i+\frac{h}{4}}$, $x_{i+\frac{h}{2}}$, $x_{i+\frac{3h}{4}}$, x_{i+1} and computed the maximum over all such points. Note that the sixth order interpolant gives satisfactory results in that the maximum error is everywhere less than Tol as required. Furthermore we see that the eighth order interpolant gives a very accurate solution. This set of results is typical of those obtained for other problems as well. Finally we note that the difference $\|Erry66 - Erry88\|$ gives a good estimate of the maximum error in the sixth order interpolant and this can be used to gauge the quality of this interpolant.

Although the code TWPBVP.f is often very efficient on mildly stiff and non-stiff problems it can perform very poorly when presented with stiff problems. This is due to the inferior stability properties of MIRK formulae and this in turn is directly related to the fact that the deferred corrections are explicit. To enable deferred correction codes to handle stiff problems efficiently, a new deferred correction code TWPBVPL.f based on Lobatto formulae was developed. This code computes implicit deferred corrections and consequently the stability is much better than for MIRK formulae. Extensive numerical testing with explicit interpolants has shown that they are of little use for dealing with stiff problems. If a highly stable scheme is used to compute the discrete solution then this stability is lost, and poor results are obtained, if an explicit interpolant is used. The problem of deriving implicit interpolants has been considered in [30] and the interested reader is referred to that paper. Numerical testing has shown that the interpolants derived in [30] are very accurate, in fact often much more accurate than is required, in the vast majority of cases and these new interpolants are very satisfactory for stiff problems. If we wish to find a sixth order interpolant we need to compute two extra function evaluations per mesh interval while for order eight we need to compute 4 extra functions. There are lots of savings of computational effort to be made because, since the interpolant

is computed *a posteriori*, we already have very good initial approximations to the extra functions that need to be computed. The conclusion is that we have now developed two codes which provide a continuous solution for two point boundary value problems. The first is TWPBVP.f which is based on MIRK methods and is suitable for non-stiff and mildly stiff problems. The other is TWPBVPL.f which is based on Lobatto formulae and is suitable for stiff problems. Both of these codes are available on the authors web page. Given in [30] are numerous examples showing the excellent performance of these interpolants and the interested reader is referred to this article.

Finally we mention that some problems are extremely difficult to solve without using continuation. Typical problems would be of the form

$$\epsilon y'' = f(x, y, y') \tag{3.3}$$

where ϵ is extremely small. For such problems it is necessary to use continuation since the layer regions defined by (3.3) often tend to be extremely narrow and so cannnot normally be detected by standard means. Using continuation codes we are able to successfully solve extremely difficult problems which we are unable to solve in any other way. The two codes ACDC.f and COLMOD.f, which are based on Lobatto deferred correction and collocation respectively, are both continuation codes and are available from the author's web page [28].

At the start of this paper we said that we would be mainly interested in the numerical solution of the general first order system of two-point boundary value problems of the form (1.1). However many two-point boundary value problems occur in a special form and there are considerable savings in computational effort to be made if we take account of this. The most commonly occuring special form is

$$y'' = f(x, y, y'), \tag{3.4}$$

or

$$y'' = f(x, y). \tag{3.5}$$

In particular after we have computed the discrete solution we have (y_n, y'_n and y''_n) available at each mesh point. This means that we can immediately define a sixth order interpolant based on this data. If we require an eighth order interpolant then we have to compute just two extra function evaluations per mesh interval and this is less than half the work compared with interpolants defined for (1.1). We notice that the special form of the equation not only offers cheap interpolants due to the fact that y'' is computed but we are also able to derive much cheaper formulae for computing the discrete solution. For a summary of how to derive efficient MIRK methods for the solution of (3.4) and (3.5) the interested reader is referred to [9, 27]. If attention is paid to the special structure of these problems we can often get much more efficient methods, and certainly more efficient interpolants, than if the problem is considered in standard form.

REFERENCES

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

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

[3] Cash J. R., *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), 129–150.

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

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

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

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

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

[9] Cash J. R., Moore D. R., *High-order interpolants for solutions of Two-Point boundary value problems using MIRK methods*, Computers and Mathematics with Applications 48 (2004), 1749–1763.

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

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

[12] Bader G., Ascher U., *A new basis implementation for a mixed order boundary value ODE solver*, SIAM J. Scient Stat Comput 8 (1987), 483–580.

[13] Wright K., *Some relationships between implicit Runge–Kutta, Collocation and Lanczos Tau methods and their stability properties*, BIT 20 (1970), 217–227.

[14] Fox L., *The Numerical Solution of Two-Point Boundary Value Problems in Ordinary Differential Equations*, Oxford University Press, 1957.

[15] Skeel R. D., *A theoretical framework for proving accuracy results for deferred corrections*, SIAM J. Numer. Anal. 19 (1982), 171–196.

[16] Lindberg B., *Error estimation and iterative improvement for discretization algorithms*, BIT 20 (1980), 486–500.

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

[18] Van Daele M., Cash J. R., *Superconvergent deferred correction methods for first order systems of nonlinear Two-Point boundary value problems*, SIAM J. Sci Comput. 22 (2001), 1697–1716.

[19] Butcher J. C., *The Numerical Analysis of Ordinary Differential Equations*, J. Wiley, 1987.

[20] Mazzia F., Trigiante D., *A hybrid mesh selection strategy based on conditioning for boundary value ODEs*, Numerical Algorithms 36, (2004), 169–187.

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

[22] Higham N. J., Tisseur F., *A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra*, SIAM J. Matrix Anal. 21 (2000), 1185–2001.

[23] Shampine L. F., Muir P. M., *Estimating conditioning for BVPs of ODEs*, Math. Comput. Modelling 40 (2004), 1309–1321.

[24] Brugnano L., Trigiante D., *Solving Differential Problems by Multistep Initial and Boundary Value Methods*, Gordon and Breach, Amsterdam, 1998.

[25] Capper S., *An order* 10 *MIRK formula*, to appear.

[26] Bashir-Ali Z., *Numerical Solution of Parameter Dependent Twopoint Boundary Value Problems using Iterated Deferred Correction*, Ph. D. University of London, (1998).

[27] Cash J. R., Garcia M. P., Moore D. R., *Mono-implicit Runge–Kutta formulae for the numerical solution of second order nonlinear two-point boundary value problems*, JACM, 143 (2002), 275–289.

[28] `http://www.ma.ic.ac.uk/~jcash/BVP_software/readme.html`

[29] Cash J. R., Mazzia F., *A new mesh selection algorithm, based on conditioning, for two-point boundary value codes*, JCAM, to appear.

[30] Cash J. R., Surmati N., Abdulla T. J., Vieira I., *The derivation of interpolants for nonlinear two-point boundary value problems*, to appear.

[31] Koch O., Weinmuller E., *Iterated defect correction for the solution of singular initial value problems* SIAM J. Numer. Anal., 38 (2001), 1784–1799.

J. R. Cash
j.cash@imperial.ac.uk

Imperial College, South Kensington
Department of Mathematics
London SW7, England.

Received: October 21, 2005