Efficient Global Methods for the Numerical Solution of Nonlinear Systems of Two point Boundary Value Problems.[*]

Jeff R. Cash[†]   Francesca Mazzia[‡]

Abstract

In this paper we will be concerned with numerical methods for the solution of nonlinear systems of two point boundary value problems in ordinary differential equations. In particular we will consider the question "which codes are currently available for solving these problems and which of these codes might we consider as being state of the art". In answering these questions we impose the restrictions that the codes we consider should be widely available (preferably written in MATLAB and/or FORTRAN) they should have reached a fairly steady state in that they are seldom, if ever, updated, they try to achieve broadly the same aims and, of course, it is relatively inexpensive to purchase the site licence. In addition we will be concerned exclusively with so called boundary value (or global) methods so that, in particular, we will not include shooting codes or Shishkin mesh methods in our survey. Having identified such codes we go on to discuss the possibility of comparing the performance of these codes on a standard test set. Of course we recognise that the comparison of different codes can be a contentious and difficult task. However the aim of carrying out a comparison is to eliminate bad methods from consideration and to guide a potential user who has a boundary value problem to solve to the most effective way of achieving his aim. We feel that this is a very worthwhile objective to pursue. Finally we note that in this paper we include some new codes for BVP's which are written in MATLAB. These have not been available before and allow for the first time the possibility of comparing some powerful MATLAB codes for solving boundary value problems. The introduction of these new codes is an important feature of the present paper.

1   Introduction.

An important task in many areas of Numerical Analysis is to carry out meaningful comparisons of numerical algorithms which attempt to achieve roughly the same well defined objectives. This is of particular interest to a user who wishes to find an appropriate code to solve his problem as well as to someone who has developed a new algorithm and wishes to compare it with existing state of the art codes. An obvious example of a particular case where such comparisons have

---

been carried out leading to the development of codes which are very much more efficient is in the numerical solution of initial value problems of the form

$$\frac{dy}{dx} = f(x, y), \ \ x \geq a, \ \ y(a) = y_a. \tag{1.1}$$

The normal course of events which was followed by, for example, algorithms for initial value problems (and also by most linear algebra routines) is that first the mathematical theory behind the problems to be solved is well understood, then the mathematical theory behind the numerical algorithms is established, then the problems involved in the writing of high quality, robust codes are identified and overcome and finally some sort of numerical comparison is carried out to highlight the strengths and weaknesses of the codes.

It is particularly relevant to us in this paper to focus our attention for the present on what has been done for initial value problems of the general form (1.1) and this we now do. Traditionally when a new code was proposed for the numerical solution of (1.1) the code was illustrated by comparing it, on a small set of test problems, with other codes also designed to solve (1.1). This proved to be less than satisfactory since there is the danger that potential disadvantages of the new code are not discovered if such limited testing is done. The first widely used test set for general initial value problems was DETEST which was proposed by Enright, Hull and Lindberg [19]. After this test set first appeared, users proposing a new method were often expected to run their code on this test set (which contains 30 problems) and this proved very useful in eliminating poor codes. However as codes became even more powerful they tended to find the problems in DETEST to be too easy. A particular cause for concern was that many of the problems in DETEST were of very small dimension and so they were often solved extremely quickly even by relatively poor codes. Following an appraisal by Shampine [35] this test set was modified to make it much more challenging but the major disadvantage still was the small dimension of the problems. A big step forward in the quality of numerical testing came with the book of Hairer and Wanner [20]. They proposed a test set which was considerably more challenging than DETEST and, in particular, contained some problems of very large dimension (for example, the Brusselator problem on page 151 of [20] is of dimension 32768). They also considerably advanced the methodology of testing and set new standards in attempting to make the tests as fair as possible. Based on the work of Hairer and Wanner an even more demanding test set was derived in Amsterdam by Lioen and de Swart [23]. They considerably broadened the aims and scope of the test set by adding differential algebraic equations with index ≤ 3. More recently this test set was taken over by Francesca Mazzia and her co-workers at the University of Bari [25] and it now plays a central role in providing realistic test problems for IVP solvers. It is important to realise, however, that this test set now has several additional facilities which can be extremely useful to users and which considerably strengthens the case for carrying out sophisticated numerical testing. For example, the user is easily able to run several state of the art codes either on his own test problems or on problems that appear in the test set. It also allows the user to test his own code against those appearing in the test set on really challenging problems and as output it produces highly relevant, and easy to understand, statistics. For these reasons the Bari test set now plays an important role in the development of powerful codes for the numerical solution of initial value problems of the form (1.1) and it also has several other important features available, which are much more important than was anticipated when this project was first begun. To see exactly what is available (and to note the very high level of use of this facility) the reader is urged to log into the Bari web page [25].

2 Boundary Value Problems.

Having explained what has been done in developing test sets for initial value problems of the form (1.1), and witnessing the central role that is taken by test sets in deriving efficient codes for initial value problems, it clearly would be a worthwhile aim to extend some of these ideas to codes for the numerical solution of two-point boundary value problems. Ideally we would like to be able to answer the question "What are the best codes currently available for solving a large class of nonlinear two-point boundary value problems and what properties of the boundary value problem need to be taken into account when deriving efficient numerical methods for its solution". Here the situation is very different than it is for initial value problems simply because boundary value problems tend to be much harder to solve and much more diverse than initial value problems. However it could be argued that in these circumstances a user is in even more need of guidance. Due to the many forms that can be taken by two-point boundary value problems, it is clear that we will need to impose some important restrictions: firstly, on which classes of problems we will attempt to solve; secondly, on which classes of numerical methods we will consider. We need to decide for example whether or not we will consider non-separated boundary conditions, eigenvalues and other parameter dependent problems, singular problems, problems of the special form $y'' = f(x, y)$ where there is no first derivative present) etc., etc. What we will in fact consider in this paper is global methods, i.e. not shooting methods or Shishkin methods, for first order systems of nonlinear two-point boundary value problems with separated boundary conditions. This means that we will restrict the class of problems that we are interested in to

$$\frac{dy}{dx} = f(x, y), \ \ a \le x \le b, \ \ \ g(y(a), y(b)) = 0. \tag{2.1}$$

In this paper our aim will be to identify which codes are at present suitable to be used for the numerical solution of (2.1). In order to do this, we need to reconsider the codes that were presented in a previous article [5], where one of the present authors sought to identify those codes designed for the numerical solution of two point boundary value problems of the form (2.1), which were in some sense efficient for the solution of this class of problems. This has naturally involved some repetition of what was considered in [5] but we feel that this is justified since it is appropriate, for the sake of completeness, for us to collect together in a single article those codes which we feel are efficient and are suitable for inclusion in a numerical comparison. Note however that all of these codes in [5] are written in FORTRAN. In due course we will hope to provide a comparison of the performance of various codes on (2.1) but in the present paper we will be mainly interested in identifying 'state of the art' codes which will be suitable for inclusion in a comparison. Some interesting thoughts concerning this can be found in [2, p.515]. In [5] three codes were identified as having the possibility of being considered state of the art codes and these were COLNEW.f/COLSYS.f, MIRKDC.f, and TWPBVP.f/TWPBVPL.f. Also identified in [5] as being powerful continuation codes were COLMOD.f and ACDC.f [9]. The first of these two codes is based on COLSYS.f and the second is an adaptation of TWPBVPL.f. These codes allow COLSYS.f and TWPBVPL.f to be run in a continuation framework. This makes these codes much more able to deal with really difficult singular perturbation problems than is the case when continuation is not used. The important questions we need to consider in this section are: whether these codes have stabilised in the time since [5] was written; whether these new codes can still be considered as state of the art; whether new codes which are competitive with these three codes have been developed in the interim; whether more recent codes written in MATLAB are now suitable. It is important to be aware of the fact that the development

3

of a test set for initial value problems took a lot of time, and considerable effort, and many researchers contributed to this project. We would expect the same to be the case for BVPs and so the present paper, which aims to identify which codes should have a place in a numerical comparison, should be regarded as the first step in an on going project.

The first code we consider is COLSYS.f/COLNEW.f [1, 2]. This code is based on the approximation of the solution of the differential equation (2.1) by a piecewise polynomial and it uses collocation at Gauss points to define this polynomial uniquely. In fact, the COLSYS.f codes are applicable directly to mixed order systems (i.e., there is no need to reduce the problem to a first order system before attempting to solve it) and in what follows we describe how COLSYS.f can deal with such systems. This code was considered in [5] but for completeness we include it here. However, there is a major difference in that in [5] we considered a single high order equation whereas here we consider a mixed order system. We consider the mixed order system of ODEs with separated boundary conditions

$$u_i^{(m_i)} = f_i(x, u_1, ..., u_1^{(m_1-1)}, u_2, .., u_d^{(m_d-1)}) \tag{2.2}$$

$$= f_i(x, z(u)), \quad 1 \le i \le d, \quad a \le x \le b. \tag{2.3}$$

The boundary conditions are

$$g_j(z(u(\eta_j))) = 0, \quad 1 \le j \le m^*, \tag{2.4}$$

where

$$u(x) = [u_1(x), u_2(x), ..., u_d(x)]^T, \tag{2.5}$$

$$m^* = \sum_{i=1}^{d} m_i, \quad a = \eta_1 \le \eta_2 \le .... \le \eta_{m^*} = b, \tag{2.6}$$

and

$$z(u(x)) = (u_1(x), u_1'(x), ..., u_1^{(m_1-1)}(x), u_2(x), ..., u_2^{(m_2-1)}(x), ..., u_d^{(m_d-1)}(x))^T. \tag{2.7}$$

There are some important points to be noted about COLSYS.f. The first is that multipoint boundary conditions are allowed but all boundary conditions must be separated. The second point is that, as mentioned earlier, collocation codes such as COLSYS.f do not require the problem to be reduced to first order form before it can be solved. The codes COLSYS.f and COLNEW.f approximate the solution by using collocation at Gauss points. This requires

$$u(x) \in C^{m_i-1}[a, b], \text{ for } i = 1, 2, ..., d. \tag{2.8}$$

In this approach, an approximate solution of the form

$$u_\pi(x) = \sum_{j=1}^{M} \alpha_j \phi_j(x), \quad a \le x \le b, \tag{2.9}$$

is sought. Here the $\phi_j(x)$ are known linearly independent functions, defined on the range $[a, b]$, and the α_j are parameters that remain to be chosen. The M parameters are determined by the requirement that $u_\pi(x)$ should satisfy the following M conditions: It should satisfy the m boundary conditions and it must also satisfy the ODE at $M - m$ points in $[a, b]$. These $M - m$ points are called the collocation points. A popular choice for the $\phi_j(x)$ is to let them be

piecewise polynomial functions and this is exactly what is done in COLSYS.f. It follows that the M conditions imposed on $u_\pi(x)$, to allow (2.9) to be uniquely defined are

1) u_π satisfies the m boundary conditions (2.4)

2) u_π satisfies the ODE at k points in each of the N mesh subintervals and this defines $Nk + m$ unknowns.

In order that the solution should satisfy the differential equation at Nk points we require that

$$0 = u_\pi^{(m)}(x_{ij}) - f(x_{ij}, u_\pi(x_{ij})), \quad 1 \le j \le k, 1 \le i \le N. \tag{2.10}$$

If we define the mesh with maximum mesh size h, and we derive our collocation method so that it is based on Gauss points with s collocation points per subinterval, then the global error is uniformly $O(h^s)$ while at the mesh points we have superconvergence and the error is $O(h^{2s})$. It is important to realise that COLSYS.f derives a continuous solution in the form of a piecewise polynomial and it also attempts to compute the error in this continuous solution. Of course, computing a continuous solution is a much more difficult task than just computing the solution at a discrete set of points as is done by several other codes. On the other hand, COLSYS.f performs the expensive task of computing a continuous solution even if the user only requires the solution to be computed at a few points. Thus, in any numerical comparison, we have to consider if the user will be provided with a continuous solution and if that is what he requires. As mentioned previously, COLSYS.f attempts to provide an estimate of the error in the continuous solution and it does this using equi-distribution. The idea is that mesh points are distributed so that an equal error is made in each mesh interval. A description of this technique can be found in [2, p.62].

The second code highlighted in [5] was MIRKDC.f. This code is based on Mono Implicit Runge-Kutta methods ([4, 11, 16]) which have been widely used for the numerical solution of two point boundary value problems. These methods have the special property that they can be implemented very efficiently for boundary value problems, due to a certain explicitness appearing in the MIRK equations. In what follows, we will describe exactly what MIRK formulae attempt to achieve and we show how this is done. Following Enright and Muir [17], we rewrite MIRK schemes in the slightly different form

$$\begin{aligned} y_{n+1} &= y_n + h \sum_{i=1}^{s} b_i \, f(x_n + c_i \, h, Y_i) \\ Y_i &= (1 - v_i) \, y_n + v_i \, y_{n+1} + h \sum_{j=1}^{s} z_{ij} \, f(x_n + c_j \, h, \, Y_j), \qquad i = 1, \dots, s \,. \end{aligned} \tag{2.11}$$

This is of course just a standard implicit Runge-Kutta method which can be expressed using the celebrated Butcher notation. A convenient way of expressing this formula as an array, is to write it in the form

$$\begin{array}{c|c|cccc} c_1 & v_1 & z_{11} & z_{12} & \dots & z_{1s} \\ c_2 & v_2 & z_{21} & z_{22} & \dots & z_{2s} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ c_s & v_s & z_{s1} & z_{s2} & \dots & z_{ss} \\ \hline & & b_1 & b_2 & \dots & b_s \end{array} \tag{2.12}$$

This is written to emphasise the fact that both y_n and y_{n+1} appear in the Runge-Kutta equations written in the form (2.11). The link between (2.12) and the standard Butcher notation is

$$A = Z + vb^T. \tag{2.13}$$

Having defined the Runge-Kutta method in this particular way, Enright and Muir then compute a solution of the given two point boundary value problem on a discrete set of points. Once

convergence has been obtained to a discrete solution, Enright and Muir consider how this can be transformed to a continuous solution by the computation of additional function evaluations. Muir and Owren consider schemes of the form

$$u(x) = u(x_i + \theta h_i) = y_i + h \sum_{r=1}^{s^*} b_r(\theta) k_r \tag{2.14}$$

where

$$\theta = (x - x_i)/h_i, \ 0 \le \theta \le 1, \ x_i \le x \le x_{i+1}, \tag{2.15}$$

which they call CMIRK schemes [33]. Note that this defines a continuous solution on the *ith* subinterval. The number of stages required to compute this continuous solution is s^*, with $s < s^*$, and the first s stages of the interpolating polynomial (2.14) are the same as the first s stages of the discrete solution. The main difference between discrete MIRK schemes and continuous CMIRK schemes is that the weight coefficients of the MIRK scheme are replaced by weight polynomials in (2.14). Having derived a continuous solution $u(x)$ by using the CMIRK framework (2.14), (2.15), Enright and Muir use defect control both for mesh selection and accuracy control, in order to define a continuous defect. The (continuous) defect is defined as

$$\delta(x) = u'(x) - f(x, u(x)) \tag{2.16}$$

and Enright and Muir estimate the maximum value of this defect on each subinterval. For further discussion of this defect correction approach, the reader is referred to [5, p.13]. In summary, we should point out that COLSYS.f controls the global error while MIRKDC.f controls the defect and these two tasks are not equivalent. The theory underlying the code MIRKDC.f [17] has remained largely unchanged since the MIRKDC.f code was written. What has changed, since [17] was written, is that there has been a major update to the code rather than to the theory. Concerned by the long calling sequence of many boundary value codes (in particular calling sequences for boundary value problems tend to be much longer than for initial value problems) and worried that many potential users may be put off using standard codes because of this, Shampine and Muir wrote a MIRKDC code which they called 'user friendly' [37]. This new code is written in FORTRAN 90/95, it cuts down drastically on the number of calling parameters required, it extends the capabilities of the MIRKDC code to the solution of problems with unknown parameters, it is able to deal with eigenvalue problems and some singular problems, and it allows the required Jacobian matrices to be computed using numerical differences. There is also a discussion explaining why it is desirable for the code to be written in FORTRAN 90/95. Also the possibility of using the code in a continuation framework is discussed. For the purpose of this paper we will consider this code to be state of the art and, for more details, the reader is referred to [37].

3 Deferred Correction Codes

The most convenient way to describe our deferred correction approach is to write our methods in a Runge-Kutta framework. As mentioned earlier, if we are using MIRK formulae we can write our Runge-Kutta formulae in the form (2.11). In what follows we consider the deferred correction framework originally proposed by Fox [18]. In this approach we first need to define

two Runge-Kutta methods which will be used to define our basic algorithm. The first Runge-Kutta method, which we will denote by ϕ, computes a cheap low order approximation to the solution of (2.1) while the second, denoted by ψ, computes an estimate of the local error in ϕ. This then allows us to define the deferred correction scheme in the basic form

$$
\begin{aligned}
\phi(\eta) &= 0, \\
\phi(\overline{\eta}) &= \psi(\eta).
\end{aligned}
\tag{3.1}
$$

For a long time the question concerning the order of accuracy of (3.1) was unresolved. This problem was solved by Skeel [38] and in what follows we give his theorem. In the theorem below the first two conditions have been known for some time, it was the third condition that was elusive. However the theorem is quite intuitive if we regard the true solution as being the sum of the numerical solution and the global error. To state Skeel's theorem consider the deferred correction scheme (3.1) defined on the grid

$$
\pi : a = x_0 < x_1 < ..... < x_N = b.
\tag{3.2}
$$

Denote by Δy the restriction of the continuous solution $y(x)$ to the final grid π. Then Skeel's theorem says the following:

Theorem Let ϕ be a stable numerical method and assume that the following conditions hold for the deferred correction scheme (3.1):

$$
\begin{aligned}
||\eta - \Delta y|| &= O(h^p), \\
||\psi(\Delta y) - \phi(\Delta y)|| &= O(h^{r+p}), \\
\psi(\Delta w) &= O(h^r),
\end{aligned}
\tag{3.3}
$$

for arbitrary functions w having at least r continuous derivatives. Then if $\phi(\overline{\eta}) = \psi(\eta)$ it follows that

$$
||\overline{\eta} - \Delta y|| = O(h^{r+p}).
\tag{3.4}
$$

The main problem was how to define the two operators ϕ and ψ. There are many ways in which the deferred correction schemes can be refined depending on the choices of ϕ and ψ and in what follows we use a particular form of deferred correction which was proposed by Fox [18] and later refined by Lindberg [22]. Their proposal was to consider two Runge-Kutta formulae of order i and j, respectively, where $i < j$. Having defined these formulae (and of course there is an extremely wide choice that we have), we consider the algorithm defined by

$$
\begin{aligned}
\phi_i(\eta) &= 0 \\
\phi_i(\overline{\eta}) &= -\phi_j(\eta).
\end{aligned}
$$

It is clear that the first two conditions of Skeel's theorem are trivially satisfied if we use this deferred correction approach, with $p = i$ and $r + p = j$. It is much more complicated to verify that the final condition is satisfied and this requires us to present our Runge-Kutta methods in a special way and to explain this the reader is referred to [10]. There are many ways in which these deferred correction methods can be refined. For example we could consider the deferred correction scheme

$$
\begin{aligned}
\phi_i(\eta) &= 0 \\
\phi_i(\overline{\eta}) &= -\phi_j(\eta) \\
\phi_i(\overline{\overline{\eta}}) &= -\phi_j(\eta) - \phi_k(\overline{\eta})
\end{aligned}
\tag{3.5}
$$

where an extra symmetric Runge-Kutta formula is introduced in an attempt to define a scheme of order k where $k > j$. Again the order of accuracy of this scheme can be determined by Skeel's theorem. A popular choice is to take $i = 4, j = 6, k = 8$. It is easy to show that if the ϕ are either MIRK formulae or Lobatto formulae then η , $\overline{\eta}$, and $\overline{\overline{\eta}}$ are of order $4, 6$ and 8 respectively. If we use the deferred correction scheme based on MIRK methods then this defines the code TWPBVP.f and if we base our codes on Lobatto formulae then we have the code TWPBVPL.f. This leads to two widely used codes which use the basic framework (3.5). For an illustration of how these formulae perform on a wide range of test problems the reader is referred to the web page of one of the authors [13]. We emphasise that the schemes we have described are just some of many that can be defined in the Runge-Kutta framework. Another possibility which certainly seems worth investigating is to use methods of different orders in different parts of the mesh. Another possibility is to use fixed order but include methods of order 10 rather than stopping at order 8. An important point to note is that these schemes, which are based on Runge -Kutta methods, have the usual problem that there is no convenient error estimate available and, also, the obtained solution is a discrete one. In order to define a continuous solution, if it is needed, we have to derive a suitable interpolant and as an estimate of the error we compute

$$\phi_i(\overline{\overline{\eta}}) - \phi_i(\overline{\eta}). \tag{3.6}$$

This gives us an error estimate in the jth order solution $\overline{\eta}$ and this is used for grid refinement [39]. There is also an important difference in the way in which the mesh is formed when using iterated deferred correction methods. As mentioned earlier, COLSYS.f uses equi-distribution and this results in mesh points 'sliding about'. The deferred correction methods discussed in [5], namely TWPBVPL.f which is based on Lobatto formulae and TWPBVP.f which is based on MIRK formulae, either add in or take out mesh points depending on the error estimate and this results in a lot of the mesh points not moving during a grid refinement. These codes are on the authors web page and can be considered as being state of the art codes. In particular, the code TWPBVP.f tends to be very effective for non-stiff problems while TWPBVPL.f is particularly efficient for stiff problems. The main reason for this behaviour is that the Lobatto codes, which use implicit deferred corrections, are much more reliable than the codes based on MIRK formulae which use explicit deferred corrections [12].

An important way in which deferred correction codes have changed since TWPBVP and TWPBVPL were written is that they now have the option of taking into account the conditioning of the problem. The importance of having this facility was demonstrated in a paper of Shampine and Muir [36] who wanted to test out the diagnostic capabilities of the two-point boundary value code BVP4C. They did this by considering Bratu's problem:

$$y'' = \lambda \exp(y), \quad y(0) = y(1) = 0. \tag{3.7}$$

Davis [14] has shown that if $0 \leq \lambda < \lambda^* = 3.51383..$ then there are 2 solutions to this problem and both are parabolic and are concave down in nature. If $\lambda = \lambda^*$ then there is just one solution and for $\lambda > \lambda^*$ there are no solutions. In their numerical experiments Shampine and Muir first described the performance of the code for $\lambda = 3.45$ and plotted one of the solutions which they obtained in a perfectly satisfactory way. However they did note that their estimate of the conditioning constant was quite high (the error tolerance imposed was 10^{-3}) and the estimate of the conditioning constant was $3.14 \cdot 10^3$. The code found this problem so easy that the solution was obtained on the initial mesh of 10 points. Having done this, Shampine and Muir solved Bratu's problem with $\lambda = 3.55$ for which the problem has no solution. The expectation was that BVP4C would fail to find a solution and would send this message back

to the user. However the code did not do this, instead it returned a solution which had all the characteristics of being parabolic in nature and concave downwards. The solver provided no warning message. However the code did need 179 points to compute the solution and the estimate of the conditioning constant was 10^6 which can be considered as being very large given the precision that is required. The reason for this poor performance of BVP4C is not hard to understand. The problem arises from the fact that the solution is obtained, and the mesh is refined, using a local error estimate. We do this on the assumption that if the local error estimate is sufficiently small, then the global error will also be small. However a backward error analysis shows that this may not be the case if the problem is ill conditioned. All of this makes a very powerful case, when solving two point boundary value problems, for computing the conditioning constant of the problem as well as the solution. Without a conditioning estimate we are not able to have any confidence in a solution that we compute based on local errors and we feel that this is a really important problem which requires considerable further investigation.

Early work on the estimation of conditioning for boundary value problems was carried out by Trigiante, Brugnano and Mazzia [3, 32]. The approach of these researchers was rather different from that of Shampine and Muir. The Shampine Muir approach on the detection of poor conditioning was to warn the user of this and to warn that there may be a severe loss of correct digits in the solution computed. In contrast, in a recent series of papers [6, 7] Cash, Mazzia and their co-workers derived efficient algorithms for estimating the condition numbers of BVPs and they developed a mesh choosing algorithm which takes into account the conditioning of the problem. In this approach the aim is to choose the mesh so that a local error tolerance is satisfied and also so that the problem remains well conditioned. This led to the derivation of codes TWPBVPLC.f and TWPBVPC.f which are based on Lobatto and MIRK methods respectively and which give an estimate of the conditioning of the problem along with the numerical solution computed by the code. The existence of these new codes gives the user the option of taking into account the conditioning of the problem and, if this option is used, the original codes TWPBVP.f and TWPBVPL.f require the change of just one input parameter. Extensive numerical testing on these four deferred correction codes appear on the web page of one of the authors [13]. What is found, for most problems where conditioning is not an issue, is that there is little difference between the performance of TWPBVP.f/ TWPBVPC.f and TWPBVPL.f/TWPBVPLC.f. However for problems where conditioning is important there is often a considerable gain in efficiency if the codes TWPBVPC.f and TWPBVPLC.f are used. Finally, another important feature, is that the estimate of the conditioning constant can be used in a backward error analysis to ensure that the final solution, which is based on local error estimation, gives a solution with a satisfactory global error. Although considerable advances have been made in the computation of conditioning constants we feel that there is still more work to be done. However the approach of Trigiante, Brugnano, Cash and Mazzia which chooses the mesh so that a local error tolerance is satisfied and so that the conditioning of the continuous and discrete problems remains roughly the same is a very powerful one.

4   Boundary Value Methods

In this section we describe a class of methods known as Boundary Value Methods (BVMs) [3]. These are linear multistep methods used in a special way that allows us to generate stable discrete boundary values schemes. In the case of symmetric schemes, given the grid π defined by (3.2) with $h_i = x_i - x_{i-1}, 1 \leq i \leq N$, the numerical scheme generated by a k-step BVM is

defined by the following equations:

$$\begin{cases} g(y_0, y_N) &= 0, \\ \sum_{j=-i}^{k-i} \alpha_{j+i}^{(i)} y_{i+j} &= h_i \sum_{j=-i}^{k-i} \beta_{j+i}^{(i)} f_{i+j}, \\ & \qquad i = 1, \dots, k_1 - 1, \\ & \text{(additional initial methods)} \\ \sum_{j=-k_1}^{k_2} \alpha_{j+k_1}^{(i)} y_{i+j} &= h_i \sum_{j=-k_1}^{k_2} \beta_{j+k_1}^{(i)} f_{i+j} \\ & \qquad i = k_1, \dots, N - k_2, \\ & \text{(main methods)} \\ \sum_{j=N-i-k}^{N-i} \alpha_{j-N+i+k}^{(i)} y_{i+j} &= h_i \sum_{j=N-i-k}^{N-i} \beta_{j-N+i+k}^{(i)} f_{i+j}, \\ & \qquad i = N + 1 - k_2, \dots, N, \\ & \text{(additional final methods)} \end{cases} \qquad (4.8)$$

where y_i is the approximation of $y(x_i)$, $f_i = f(x_i, y_i)$, k is odd, $k_1 = (k+1)/2$, $k_2 = k - k_1$ and $\boldsymbol{\alpha}^{(i)} = (\alpha_0^{(i)}, \dots, \alpha_k^{(i)})^T$ and $\boldsymbol{\beta}^{(i)} = (\beta_0^{(i)}, \dots, \beta_k^{(i)})^T$ are the coefficient vectors characterizing the method. The so called "additional initial methods" and "additional final methods" i.e. $k_1 - 1$ initial and k_2 final equations, are derived by using appropriate discretization schemes.

The main code currently available, which implements these methods, is known as TOM. This is a general purpose code for the solution of BVPs and is rather different from other codes that have been derived. The first release of the code was written in MATLAB in 2003 [24], and was based on a class of symmetric Boundary Value Methods (BVMs) [3]. The Top Order Methods (TOM) are k-step linear multistep methods with the highest possible order $2k$ and in the code TOM we use $k = 3$ to give a sixth order method. A complete description of these methods, along with their stability properties, can be found in [3, sec. 7.4]. Recently the code has been updated by implementing another class of BVMs, namely the BS linear multistep methods [27, 28, 29]. The new version of the code is available in MATLAB.

The BS methods are derived by imposing the restriction that the numerical solution of the general k-step linear multistep formula is the same as is given by the collocation procedure using the B-spline basis. The coefficients are computed by solving special linear systems. The k-step BS scheme provides a C^k continuous solution that is kth-order accurate uniformly in $[a, b]$ and collocating the differential equation at the mesh points [29]. In order to introduce the continuous extension, some further notation is needed. First, we represent any spline function s of degree $d = (k+1)$ and knot $x_i, i = 0, \dots, N$ using the associated (dth-degree) B–spline basis $B_j(x), j = -d, \dots, N-1$. This can be defined after prescribing two additional sets of d knots, $\{x_i, i = -d, \dots, -1\}$ (*left auxiliary knots*), with $x_{-d} \le \dots \le x_0$, and $\{x_i, i = N+1, \dots, N+d\}$ (*right auxiliary knots*), with $x_N \le x_{N+1} \le \dots \le x_{N+d}$ [15]. Using this notation, the spline collocating the differential equation can be represented as follows,

$$Q_d^{(BS)}(y) = \sum_{j=-d}^{N-1} \mu_j^{(BS)}(y) B_j, \qquad (4.9)$$

where $\mu_j^{(BS)}(y)$ are linear combinations of the values of y_i and f_i, $i = 0, \dots, N$ in π. We note that, if the values y_i and f_i have been computed with a different scheme, the continuous approximation is the quasi-interpolation spline described in [26]. This means that this continuous extension could safely be used for any discretization method.

The main feature of the code is that it implements a hybrid mesh selection strategy based on both the conditioning parameters of the problem and on a suitable approximation of the

local error. This strategy was introduced in [32] and was first implemented in the code TOM. Subsequently it was modified in [6, 7] for use in TWPBVPC and TWPBVPLC. As is the case for the codes based on deferred correction, by changing just one input parameter, it is possible for the mesh selection to use a standard strategy based only on the local error for the mesh selection. Instead of using a damped Newton method for the solution of the nonlinear equations, the code TOM implements a quasi-linearization technique [29, 30, 31]. This means that a sequence of continuous linear BVPs is solved to a suitable tolerance. This allows us to use very efficiently the mesh selection based on conditioning for non linear problems as well as linear ones.

We note that the conditioning parameters estimated by the code can be defined both for the continuous problem and for the discrete one giving the possibility to *measure* the reliability of the discrete problem with respect to the continuous one. In other words, we must be suspicious if the discrete problem provides parameters which are very different from the continuous ones, and this could be checked if the conditioning parameters do not converge. Since such parameters, for the discrete problem, depend on the chosen mesh, it is possible, for a fixed method, to vary the latter in order to be sure that the discrete parameters converge. This allows us, for example, to recognize if the solution has two or more time scales associated with it. This is the idea on which the mesh strategy is based.

Since the BS are collocation methods, the code provides as output a continuous solution, which could be used to estimate the error at any chosen point, or a set of discrete points, if, for example, the user requires only the solution at the mesh points. A continuous solution is also computed, if needed, when using the TOM code, by the quasi-interpolation technique based on the BS methods described in [26].

5   Interpolation

As mentioned earlier, the deferred correction codes TWPBVP.f and TWPBVPL.f are the only ones described in this paper which do not seek to provide a continuous solution. The obvious way of getting around this problem is to derive an a posteriori interpolant which can be computed using just a few extra function evaluations. This is rather similar to what the code MIRKDC does, although this code needs a continuous solution from the start so that it can estimate the defect. Here a discrete solution is computed first of all and this is accepted once the defect is sufficiently small. After the discrete solution has been accepted a continuous MIRK scheme is formed using an approach of Muir and Owren [33]. Note that, if a continuous solution is not required, for example the solution may be required only at a few mesh points, then this interpolant is not needed for error estimation. Numerical experience has shown that it is advisable to compute an interpolant using data from only one sub-interval, and this avoids problems at the end of the mesh. If the interpolant is a local one, that is it is defined over a single interval, then the interpolant is identical over each sub-interval and it also has the desirable property that it is symmetric. The problem of deriving high order interpolants for MIRK formulae has recently been considered in some detail. It has been shown in [8] that a sixth order MIRK interpolant can be computed using just one extra function evaluation. To compute an eighth order MIRK interpolant, we need four extra function evaluations and the way in which this is done is described in [8]. By deriving an a posteriori interpolant, we are effectively introducing a continuous solution and this allows a fair comparison with other codes discussed in this paper to be carried out.

6 Conclusion.

The purpose of the present paper was to answer the question 'Which codes for the numerical solution of two point boundary value problems of the form (2.1) can be considered as being state of the art'. The codes identified as falling into this class were based on: collocation methods (COLSYS/COLNEW); defect control methods based on MIRK formulae (MIRKDC); deferred correction methods (TWPBVP/TWPBVPL) and boundary value methods (TOM). There is considerable numerical evidence to suggest that these codes are amongst the most efficient global methods currently available. In addition these codes have reached what we have called a "steady state" and this makes them ideal for use in a numerical comparison. However there are two important points that we need to bear in mind.

Firstly, these codes attempt to solve slightly different problems. In particular COLSYS/COLNEW and TOM attempt to define a continuous solution by computing the error in a polynomial approximation to the solution. MIRKDC also seeks to provide a continuous solution but controls the error in the defect. In contrast the deferred correction codes compute a discrete solution initialy and the continuous solution is obtained using an a posteriori interpolant.

The second point to note is that it is important to take into account the conditioning of the problem when solving a problem of the form (2.1). A standard backward error analysis which links the global error to the local one relies on the problem being well conditioned and, if it is not, a solution which has not got the required accuracy may be accepted. A considerable amount of effort has been applied to the estimation of the conditioning of a problem and the reader is referred to [3, 6, 7, 32]. The boundary vale methods and deferred correction codes allow conditioning estimation and an estimate of the conditioning can be obtained by changing just one input parameter for these codes. We expect the estimation of the conditioning of a problem to become a very important topic in the future. In addition we feel that the four FORTRAN codes we have discussed are a very firm basis for carrying out numerical comparisons of different methods and we hope to be able to investigate this in the near future. Finally, we list those codes in FORTRAN and MATLAB which would be candidates for use in a numerical comparison. We feel it appropriate to divide the codes into two catagories, namely those which are written in FORTRAN and those which are written in MATLAB. Generally speaking, if run time is an issue then the user should probably be using a FORTRAN code. If it is not, then the user may find a MATLAB code to be more convenient. The codes that we feel either are or will become state of the art codes are

- **FORTRAN codes**: TWPBVPC, TWPBVPLC, ACDC, COLNEW, COLMOD, MIRKDC, BVP_SOLVER.

- **MATLAB codes**: BVP4C, BVP5C ([21]), TOM, TWPBVPC, TWPBVPLC.

Note that the MATLAB codes TWPBVPC/TWPBVPLC have only just been released but are based on widely used codes. We finish this section with the observation that codes written in languages other than FORTRAN or MATLAB are starting to appear. As an example we mention the code TWPBVP which is now available in R (package `bvpSolve`) [34].

References

[1] U. Ascher, J. Christiansen, and R. D. Russell. Collocation software for boundary-value odes. *ACM Trans. Math. Softw.*, 7(2):209–222, 1981.

[2] U. M. Ascher, R. M. M. Mattheij, and R. D. Russell. *Numerical solution of boundary value problems for ordinary differential equations*, volume 13 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1995. Corrected reprint of the 1988 original.

[3] L. Brugnano and D. Trigiante. *Solving differential problems by multistep initial and boundary value methods*, volume 6 of *Stability and Control: Theory, Methods and Applications*. Gordon and Breach Science Publishers, Amsterdam, 1998.

[4] J. R. Cash. A class of implicit Runge-Kutta methods for the numerical integration of stiff ordinary differential equations. *J. ACM*, 22(4):504–511, 1975.

[5] J. R. Cash. A survey of some global methods for solving two-point BVPs. *Appl. Numer. Anal. Comput. Math.*, 1(1-2):7–17, 2004.

[6] J. R. Cash and F. Mazzia. A new mesh selection algorithm, based on conditioning, for two-point boundary value codes. *J. Comput. Appl. Math.*, 184(2):362–381, 2005.

[7] J. R. Cash and F. Mazzia. Hybrid mesh selection algorithms based on conditioning for two-point boundary value problems. *JNAIAM J. Numer. Anal. Ind. Appl. Math.*, 1(1):81–90, 2006.

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

[9] 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(2):266–279, 1995.

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

[11] J. R. Cash and A. Singhal. High order methods for the numerical solution of two-point boundary value problems. *BIT*, 22(2):184–199, 1982.

[12] J. R. a Cash and M. H. Wright. A deferred correction method for nonlinear two-point boundary value problems: implementation and numerical evaluation. *SIAM J. Sci. Statist. Comput.*, 12(4):971–989, 1991.

[13] J.R. Cash and F. Mazzia. Algorithms for the solution of two-point boundary value problems. `http://www.ma.ic.ac.uk/~jcash/BVP_software/twpbvp.php`.

[14] Harold T. Davis. *Introduction to nonlinear differential and integral equations*. Dover Publications Inc., New York, 1962.

[15] C. de Boor. *A practical guide to splines*, volume 27 of *Applied Mathematical Sciences*. Springer-Verlag, New York, revised edition, 2001.

[16] W. H. Enright and P. H. Muir. Efficient classes of Runge-Kutta methods for two-point boundary value problems. *Computing*, 37(4):315–334, 1986.

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

[18] L. Fox. *The numerical solution of two-point boundary problems in ordinary differential equations.* Oxford University Press, New York, 1957.

[19] Enright W. H., Hull T. E., and Lindberg B. Comparing numerical methods for stiff systems of O.D.Es. *BIT*, 15(2):10–48, 1975.

[20] E. Hairer and G. Wanner. *Solving ordinary differential equations. II*, volume 14 of *Springer Series in Computational Mathematics.* Springer-Verlag, Berlin, 1991. Stiff and differential-algebraic problems.

[21] J. Kierzenka and L. F. Shampine. A BVP solver that controls residual and error. *JNAIAM J. Numer. Anal. Ind. Appl. Math.*, 3(1-2):27–41, 2008.

[22] B. Lindberg. Error estimation and iterative improvement for discretization algorithms. *BIT*, 20(4):486–500, 1980.

[23] W. M. Lioen and J.J.B. de Swart. Test set for IVP solvers. Technical Report MAS-R9832, CWI, Amsterdam, 1998.

[24] F. Mazzia. *Software for Boundary value Problems.* Department of Mathematics, University of Bari and INdAM, Research Unit of Bari, February 2003. Available at `http://www.dm.uniba.it/~mazzia/bvp/index.html`.

[25] F. Mazzia and C. Magherini. *Test Set for Initial Value Problem Solvers, release 2.4.* Department of Mathematics, University of Bari and INdAM, Research Unit of Bari, February 2008. Available at `http://www.dm.uniba.it/~testset`.

[26] F. Mazzia and A. Sestini. The BS class of hermite spline quasi-interpolants on nonuniform knot distributions. *BIT*, 49(3):611–628, 2009.

[27] F. Mazzia, A. Sestini, and D. Trigiante. B-spline linear multistep methods and their continuous extensions. *SIAM J. Numer. Anal.*, 44(5):1954–1973 (electronic), 2006.

[28] F. Mazzia, A. Sestini, and D. Trigiante. BS linear multistep methods on non-uniform meshes. *JNAIAM J. Numer. Anal. Ind. Appl. Math.*, 1(1):131–144, 2006.

[29] F. Mazzia, A. Sestini, and D. Trigiante. The continous extension of the B-spline linear multistep metods for BVPs on non-uniform meshes. *Appl. Numer. Math.*, 59(3-4):723–738, 2009.

[30] F. Mazzia and I. Sgura. Numerical approximation of nonlinear BVPs by means of BVMs. *Appl. Numer. Math.*, 42(1-3):337–352, 2002. Ninth Seminar on Numerical Solution of Differential and Differential-Algebraic Equations (Halle, 2000).

[31] F Mazzia and D. Trigiante. Efficient strategies for solving nonlinear problems in bvps codes. *Nonlinear Studies.* in press.

[32] F. Mazzia and D. Trigiante. A hybrid mesh selection strategy based on conditioning for boundary value ODE problems. *Numer. Algorithms*, 36(2):169–187, 2004.

[33] P. Muir and B. Owren. Order barriers and characterizations for continuous mono-implicit Runge-Kutta schemes. *Math. Comp.*, 61(204):675–699, 1993.

[34] R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

[35] L. F. Shampine. Evaluation of a test set for stiff ode solvers. *ACM Trans. Math. Softw.*, 7(4):409–420, 1981.

[36] L. F. Shampine and P. H. Muir. Estimating conditioning of BVPs for ODEs. *Math. Comput. Modelling*, 40(11-12):1309–1321, 2004.

[37] L. F. Shampine, P. H. Muir, and H. Xu. A user-friendly Fortran BVP solver. *JNAIAM J. Numer. Anal. Ind. Appl. Math.*, 1(2):201–217, 2006.

[38] R. D. Skeel. A theoretical framework for proving accuracy results for deferred corrections. *SIAM J. Numer. Anal.*, 19(1):171–196, 1982.

[39] R. Wright, J. Cash, and G. Moore. Mesh selection for stiff two-point boundary value problems. *Numer. Algorithms*, 7(2-4):205–224, 1994.