FAST STABLE DEFERRED CORRECTION METHODS FOR TWO-POINT BOUNDARY VALUE PROBLEMS[*]

JOHN STRAIN[†]

Abstract. Simple, high-order accurate, adaptive methods for linear two-point boundary value problems are generated by iterated deferred correction of the second-order midpoint rule with high-order uncentered difference formulas on an adaptive mesh. A fast structured QR factorization produces robust solutions of the ensuing linear systems. The methods solve problems with general nonseparated boundary conditions as efficiently as separated ones. Numerical experiments on well-resolved problems show that the methods achieve high-order accuracy despite some previously conjectured order barriers. Efficiency and accuracy compare favorably with several standard codes on a gallery of difficult test problems.

Key words. deferred correction, two-point boundary value problems, adaptive mesh, midpoint method, QR-factorization, Airy functions, Bessel functions

AMS subject classifications. 65L10, 65L12, 65L50, 65F05, 65F50

1. Introduction.

Numerical methods for solving two-point boundary value problems in ordinary differential equations have been extensively developed: classical methods such as shooting [26], collocation [2, 3, 28], finite differences and deferred correction [7, 25] are analyzed in [4, 32]. More recent developments include spectral integral [22, 24, 31], rational pseudospectral [6], Lobatto deferred correction [5, 12] and mono-implicit Runge-Kutta deferred correction [9, 14, 13, 15, 34, 36, 35] methods. Several widely available codes based on these approaches can solve routine problems accurately and efficiently. However, the solution of extremely difficult problems with widely separated scales, high-frequency oscillations, and boundary and internal layers, is still an area of active research.

In this paper, we develop a simple, fast, and effective numerical method for high-accuracy solution of difficult linear boundary value problems. The method combines iterated deferred correction (§2.4) of the simple midpoint rule (§2.1), high-order uncentered difference formulas (§2.3), a robust adaptive mesh refinement strategy (§2.5), and a fast stable structured QR factorization (§2.2) for the solution of linear systems. We focus on linear problems because appropriate continuation strategies for Newton's method effectively reduce nonlinear problems to linear ones. While most codes for two-point boundary value problems require special separated boundary conditions, our method solves general separated and nonseparated problems with equal efficiency. It computes complex high-accuracy solutions to difficult stiff and oscillatory problems (§3.1) on both equidistant (§3.2) and adaptive (§3.3) meshes, in CPU time comparable to several popular public domain codes (§3.5). Experiments on a well-resolved problem demonstrate optimal orders of accuracy (§3.4), higher than predicted by classical convergence theory [16, 29].

2. The numerical method.

Our high-order adaptive deferred correction method is constructed in the following way. First, we define boundary value problems (BVPs) and the midpoint rule, and formulate them as linear operator equations. Second, we develop a fast stable solution method for the operator version of the midpoint

[*] Research supported by Air Force Office of Scientific Research Grant number FDF49620-02-1-0160-STRAIN-3/03.

[†] Department of Mathematics, University of California, 970 Evans Hall #3840, Berkeley, California 94720-3840 (strain@math.berkeley.edu)

rule. Third, we derive high-order discretizations which are accurate but costly to solve. Fourth, we employ iterated deferred correction as a fast approximate method for solving high-order discretizations. Fifth, we use error estimates produced in the course of iterated deferred correction to devise a robust strategy for adaptively refining the mesh to solve difficult problems efficiently. Finally, we combine these five ingredients into a complete algorithm.

2.1. The midpoint rule. Consider a linear two-point boundary value problem (BVP)

$$\begin{aligned} y' - C(t)y &= f(t), \qquad a < t < b \\ Ay(a) + By(b) &= g \end{aligned} \tag{1}$$

for a vector-valued function $y : [a,b] \to \mathbf{R}^q$. Here $C(t)$ is a continuous $q \times q$ matrix-valued function on the interval $I = [a,b]$ and A, B are fixed $q \times q$ matrices. We will often find it convenient to treat the BVP as a single operator equation

$$Ly = \begin{bmatrix} y' - C(t)y \\ Ay(a) + By(b) \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix} = F, \tag{2}$$

where L maps $C^1(I;\mathbf{R}^q)$ continuously to the Cartesian product space $C^0(I;\mathbf{R}^q) \times \mathbf{R}^q$.

Given an n-point mesh $a = t_1 < t_2 < \cdots < t_n = b$, a stable second-order accurate discretization of Eq. (1) is provided by the midpoint rule

$$\begin{aligned} \frac{u_{j+1} - u_j}{h_j} - C_j \frac{u_{j+1} + u_j}{2} &= f_j \qquad 1 \le j < n, \\ Au_1 + Bu_n &= g. \end{aligned} \tag{3}$$

Here u_j approximates $y(t_j)$, each mesh size $h_j = t_{j+1} - t_j$ is bounded by $h = \max_j h_j$, the midpoint $t_{j+1/2} = t_j + h_j/2$, and the midpoint data is $f_j = f(t_{j+1/2})$ and $C_j = C(t_{j+1/2})$. We often treat the numerical method as a $N \times N$ matrix-vector equation $M_h u = F_h$ where $N = nq$,

$$\begin{aligned} u &= (u_1(t_1), u_2(t_1), \ldots, u_q(t_1), u_1(t_2), \ldots, u_q(t_n))^T, \\ F_h &= (f_1(t_{3/2}), \ldots, f_q(t_{3/2}), \ldots, f_q(t_{n-1/2}), g_1, \ldots, g_q)^T, \\ M_h &= \frac{1}{2} \begin{bmatrix} \frac{-2}{h_1} - C_1 & \frac{2}{h_1} - C_1 & 0 & 0 & \cdots & 0 \\ 0 & \frac{-2}{h_2} - C_2 & \frac{2}{h_2} - C_2 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \frac{-2}{h_{n-1}} - C_{n-1} & \frac{2}{h_{n-1}} - C_{n-1} \\ A & 0 & \cdots & 0 & 0 & B \end{bmatrix}. \end{aligned}$$

Under reasonable assumptions on the operator L and mesh sizes h_j [4, 32], the midpoint rule is stable (the matrix norm $\|M_h^{-1}\|$ is bounded independently of the maximum mesh size $h \to 0$) and second-order accurate (the error is bounded by $\max_j \|u_j - y(t_j)\| = O(h^2)$). These basic properties ensure that the solution can be computed to arbitrary accuracy ϵ with $n = O(1/h) = O(\epsilon^{-1/2})$ mesh points t_j on $[a,b]$ as $\epsilon \to 0$. The standard Gaussian elimination approach used in most BVP solvers [8] requires $O(N^3) = O(\epsilon^{-3/2})$ work and storage to compute the solution u, and lacks stability when the ill-conditioning of the BVP is inherited by the midpoint rule matrix M_h. Thus achieving high accuracy in reasonable CPU times will require stable solution methods which respect the sparsity structure of M_h, which has at most $2q$ nonzero elements among the nq elements in each row.

2.2. Stable efficient solution methods. The midpoint discretization matrix M_h has the block "half-arrowhead" sparsity structure

$$M_h = \begin{bmatrix} D_1 & S_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & D_2 & S_2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & D_3 & S_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & D_{n-1} & S_{n-1} \\ B_1 & 0 & 0 & \cdots & 0 & 0 & B_n \end{bmatrix}$$

where D_i, S_i and B_i are $q \times q$ blocks.

An implementation of Gaussian elimination for the solution of $M_h u = F_h$ should ideally be both efficient and stable. Efficiency requires that the sparsity structure of M_h be respected as in [8], but stability usually requires pivoting, which destroys the sparsity structure.

The requirements of efficiency and stability are compatible if M_h is block bidiagonal ($B_1 = 0$), or when the rows and columns of M_h can be rearranged into block bidiagonal form. Such a rearrangement is possible iff the boundary conditions are *separated*: for each index $p = 1$ to q either row p of $A = B_1$ or row p of $B = B_n$ is zero. Intuitively, each boundary condition restricts the components of $y(a)$ or $y(b)$ but does not intertwine them. Numbering the boundary conditions on $y(a)$ first and $y(b)$ last among the N equations makes M_h block tridiagonal, so Gaussian elimination can be both efficient and stable [25].

However, many practical problems present nonseparated boundary conditions for which Gaussian elimination becomes extremely inefficient. Thus we use a fast solution technique which handles general separated and nonseparated boundary conditions with equal efficiency. We use Householder transformations [18] to compute the QR factorization $QR = M_h$, with Q orthogonal and R upper triangular, whereupon u is easily computed as $R^{-1}Q^T F_h$. Orthogonal factorization is more attractive than Gaussian elimination for solving the possibly ill-conditioned sparse linear systems produced by discretization of difficult BVPs, because it provides superior stability properties while eliminating pivoting and excessive fill-in. The resulting factors Q and R can be stored in the special arrowhead matrix

$$M_h = \begin{bmatrix} D_1 & S_1 & 0 & 0 & \cdots & 0 & R_1 \\ 0 & D_2 & S_2 & 0 & \cdots & 0 & R_2 \\ 0 & 0 & D_3 & S_3 & \cdots & 0 & R_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & D_{n-2} & S_{n-2} & R_{n-2} \\ 0 & 0 & \cdots & 0 & 0 & D_{n-1} & S_{n-1} \\ B_1 & B_2 & B_3 & B_4 & \cdots & B_{n-1} & B_n \end{bmatrix},$$

where the nonzero components of the Householder vectors are stored in the B_j blocks, the subdiagonal parts of the D_j blocks and an extra vector b containing the diagonal components. The QR factorization requires $O(Nq^2)$ time and $4Nq$ storage, making it essentially optimal with respect to the problem size N. Once the matrix M_h is factorized, subsequent linear systems with different right-hand sides can be solved at only $O(Nq)$ cost per solve. Fast updating of the factorization is also possible [19], facilitating continuation methods as in [12].

Accuracy ϵ can now be obtained in $O(Nq) = O(\epsilon^{-1/2})$ work and storage, corresponding to roughly three times the work for each additional digit of accuracy. This cost/benefit ratio is acceptable for low accuracy, but high accuracy requires discretization methods with higher orders of accuracy than the second-order midpoint rule.

2.3. Higher-order discretizations. The midpoint rule can be improved by using better approximations of the midpoint derivatives $y'(t_{j+1/2})$ and values $y(t_{j+1/2})$. Such approximations can be generated by uncentered higher-order polynomial interpolation

$$y'(t_{j+1/2}) \approx P_j'(t_{j+1/2}), \qquad y(t_{j+1/2}) \approx P_j(t_{j+1/2}).$$

Here P_j is a (q-vector valued) polynomial of degree at least $2p-1$ which interpolates the values u_k at $2p$ points t_k, centered at $t_{j+1/2}$ to the extent possible without crossing the endpoints of the interval:

$$P_j(t_k) = u_k \qquad l_j \leq k \leq r_j = l_j + 2p - 1$$

where

$$l_j = \begin{cases} 1 & 1 \leq j < p, \\ j-p+1 & p \leq j \leq n-p, \\ n-2p+1 & n-p < j < n. \end{cases}$$

While $P_j(t)$ and $P_j'(t)$ can be evaluated by standard techniques such as Lagrange or Newton interpolation [33], their values at $t_{j+1/2}$ can be evaluated by convenient difference formulas such as

$$y'(t_{j+1/2}) \approx \sum_{k=l_j}^{r_j} \alpha_{jk} u_k, \qquad y(t_{j+1/2}) \approx \sum_{k=l_j}^{r_j} \beta_{jk} u_k.$$

Stable efficient routines for the computation of the coefficients α_{jk} and β_{jk} have been provided by Fornberg [21]. The resulting higher-order discretizations of the BVP (1) have the form

$$L_h u = \left[\begin{array}{c} (\sum_{k=l_j}^{r_j} \alpha_{jk} u_k - C_j \sum_{k=l_j}^{r_j} \beta_{jk} u_k) \\ Au_1 + Bu_n \end{array} \right] = \left[\begin{array}{c} (f_j) \\ g \end{array} \right] = F_h.$$

L_h is formally order-$2p$ accurate, and therefore asymptotically more efficient than low-order discretizations for computing highly accurate solutions to well-conditioned smooth problems. Obtaining accuracy ϵ with a method of order $2p$ requires on the order of $n = O(\epsilon^{-1/2p})$ mesh points, which is extremely efficient whenever the error tolerance ϵ is sufficiently stringent and the constants implied by the $O()$ symbol are not too large. However, the matrix L_h has a bandwidth proportional to p and therefore requires $O(np^3) = O(Np^2)$ CPU time and storage. For a 20th order method, a factor of order 100 is often acceptable in CPU time but not in storage. Deferred correction, like Krylov subspace methods [27], greatly reduces the storage requirements for solution of high-order discretizations.

2.4. Iterated deferred correction. Deferred correction iterates with a low-order preconditioner to solve a high-order discretization at low cost. The approach comes in many variants [7, 5, 9, 12, 14, 13, 15, 25, 34, 36, 35], some based on complicated asymptotic error expansions [4]. The simplest one can be summarized as approximately solving for the error in a computed solution from a residual equation, and then subtracting the error to get an improved solution.

Consider a numerical solution u which is interpolated by a piecewise polynomial $P_j(t)$ on the interval $[t_j, t_{j+1}]$ to give a residual

$$\rho(t) = F - LP = \begin{bmatrix} C(t)P_j(t) + f(t) - P_j'(t) \\ g - AP(a) - BP(b) \end{bmatrix}.$$

At mesh midpoints $t_{j+1/2}$, the residual components are

$$\begin{aligned} \rho_j &= C_j P_j(t_{j+1/2}) + f_j - P_j'(t_{j+1/2}) \\ &= C_j \sum_{k=l_j}^{r_j} \beta_{jk} u_k + f_j - \sum_{k=l_j}^{r_j} \alpha_{jk} u_k, \qquad 1 \leq j < n, \\ \rho_n &= g - Au_1 - Bu_n. \end{aligned}$$

Since the residual satisfies $LP = F - \rho$ by definition and the exact solution satisfies $Ly = F$, subtraction gives $L(y - P) = \rho$. Therefore the correction $c = y - P$ which makes P into the exact solution $y = P + c$ satisfies the correction equation

$$Lc = \rho.$$

The correction equation can be solved efficiently by the midpoint rule $M_h c_1 = \rho$ with the factorized matrix M_h, yielding a relatively second-order accurate approximation c_1 to the correction c. The corrected solution $u + c_1$ is formally two orders more accurate than u, if the residual has been computed with sufficient accuracy.

The correction procedure can be repeated, formally gaining two additional orders of accuracy per repetition, as long as the error is sufficiently smooth and the residual has been computed with sufficient accuracy. However, several theoretical analyses [4, 16, 29] have shown that when uncentered differences are used near the ends of the interval, m corrections may not yield the expected order of accuracy $2(m + 1)$ for $m > 2$. Instead, the order of accuracy increases like the sequence 2, 4, 6, 7, 8, 9, 10, 11, 12, ... This result assumes the mesh is smooth in the strong sense that

$$\max_j \left| \frac{h_j}{h_{j-1}} - 1 \right| \leq O(h),$$

and polynomial interpolation of variable degree $2k + 1$ is used for each correction number $k = 1$ to m. Our numerical experiments (§3.4), by contrast, demonstrate that the use of a fixed degree $2m + 1$ of polynomial interpolation at each correction $k = 1$ to m yields the full order $2(m + 1)$ on an equidistant mesh. This does not contradict the analyses of [4, 16, 29] since we are using a fixed high interpolation degree $2m + 1$ at every correction step, while the analyses of [4, 16, 29] address the use of the minimum possible degree $2k + 1$ at correction step k.

Deferred correction also provides a sequence of approximate corrections which allow us iteratively to generate a suitable adaptive mesh for resolving the solution with maximal efficiency.

2.5. Adaptive mesh refinement. Many BVPs have coefficients or solutions which change rapidly over small fractions of the computational domain. Such rapid concentrated variations cannot be efficiently resolved by global equidistant meshes; instead, a nonequidistant mesh tailored to these variations must be built adaptively during the solution process. We employ a natural strategy to build this adaptive mesh: given an error estimate $e(t) = O(h^p)$ for a solution u computed with mesh sizes $h(t) \leq h$, and a desired error tolerance ϵ, we choose new mesh sizes

$$\tilde{h}(t) = \left(\frac{\gamma\epsilon}{\max\{e(t), \gamma\epsilon\}} \right)^{1/p} h(t)$$

at each point $t \in [a, b]$. Here $\gamma = 10^{-1}$ is a fudge factor designed to avoid cycling. The new mesh size is chosen to make the new error at each point less than $\gamma\epsilon$, assuming that the error estimate decreases like $O(h^p)$ as $h \to 0$. In deferred correction, the final computed correction norm $\|c_m\|$ is a natural error estimate $e(t)$. Since it decreases like $O(h^{2m})$, we put $p = 2m$ and $e(t) = \|c_m(t)\|$. Setting $e(t) = \|c_m(t)\|$ vastly overestimates the actual $O(h^4)$ error if we use only the first correction norm $\|c_1\|$ of size $O(h^2)$; but the deferred correction approach is most useful when high accuracy is required, and then our computational experience indicates that the $O(h^{2m})$ correction norm $\|c_m\|$ is an reliably conservative estimate of the actual $O(h^{2m+2})$ error.

Numerical experiments have suggested several improvements of this basic strategy. First, we require that $\tilde{h}(t) \leq 0.8h(t)$ at every t, to ensure that the mesh generation process terminates rapidly rather than cycling near its end. Second, we discourage extreme gradients in the mesh by requiring $\tilde{h}(t) \geq h(t)/10$, so mesh sizes cannot decrease too suddenly. Finally, we have observed that deferred correction rarely can improve a second-order solution which is completely inaccurate. Therefore we construct an initial mesh (for higher-order deferred correction method) on which the first $O(h^2)$ correction c_1 is no larger than 10% of the solution maximum. This amounts to determining a preliminary mesh by the fourth-order method with $\epsilon = 10^{-1}$, which costs less than using a high-order method on a preliminary mesh. Given one-digit accuracy on the initial correction with a preliminary mesh, deferred correction usually does a good job of the subsequent correction and mesh refinement steps.

2.6. Adaptive deferred correction algorithm. Our algorithm can be summarized as follows:

```
**Initialize** mesh of intervals
**Do** until error estimate is below tolerance or workspace is exhausted
    **Initialize** $u = 0$
    **Build** and factorize $M_h = QR$ for the current mesh
    **Do** $j = 0, 1, \ldots, m$ corrections to order $p = 2m + 2$
        **Compute** residual $\rho = F - LP$ from interpolant $P$ to $u$
        **Solve** correction equation $M_h c_j = \rho$ with saved QR factorization of $M_h$
        **Update** $u \leftarrow u + c_j$
    **End do**
    **Refine** mesh by shrinking intervals where $\|c_m\|$ is too large
**End do**
```

The initial computation of the basic second-order solution by the midpoint rule has been eliminated from the algorithm by viewing it as a correction $u = c_0$ to the zero solution.

3. Implementation and numerical results. We implemented our deferred correction approach in a double precision Fortran 77 code `bvpdc`. The code is designed for easy adaptability to a wide variety of boundary value problems: the user provides subroutines for evaluating the right-hand side and boundary conditions, and specifies a few numerical parameters such as the error tolerance ϵ, the number of corrections m, and the type of mesh. The code was compiled with the GNU Fortran 77 compiler `g77`, using the `-O` optimization flag, and run on one 450MHz CPU of a Sun Ultra 60 under Solaris 8. Special QR factorization routines were coded to take advantage of the block arrowhead structure of the matrix M_h.

We tested `bvpdc` on a gallery of highly challenging test problems (§3.1) taken from articles [11, 22, 24, 31] on other methods for solving two-point boundary value problems. Results are reported in §3.2 and §3.3.

We verify high-order accuracy of the method on a straightforward but nontrivial beam problem in §3.4. As discussed in §2.4, our results do not contradict order barriers proved in [16, 29], because we use high-order uncentered differences throughout the correction process.

For comparison, the results of solving the same test problems with four standard BVP codes from Netlib (`www.netlib.org`) are presented in §3.5.

3.1. Test problems. We study the performance of `bvpdc` and the four Netlib codes on the following collection of challenging test problems for two-point BVPs. Some of these test problems have been studied by sophisticated spectral integral methods with excellent results [22, 24, 30, 31]. Comparisons of earlier numerical methods for two-point BVPs are reported in [11].

3.1.1. Beam equation. A standard elasticity problem [31] can be transformed into the first-order system

$$y' - \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-K}{EI} & 0 & 0 & 0 \end{bmatrix} y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{q}{EI} \end{bmatrix}, \qquad a = 0 < t < b = L,$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} y(a) + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} y(b) = 0,$$

where $L = 1.2 \times 10^2$, $k = 2.604 \times 10^3$, $E = 3 \times 10^7$, $q = 4.34 \times 10^4$ and $I = 3 \times 10^3$ are material constants. The exact solution is a combination of exponential functions varying on scales of order $(K/EI)^{1/4} \approx 10^{-2}$.

3.1.2. Stiff equation. The model stiff boundary value problem of [31] is given by

$$y' - \begin{bmatrix} 998 & 1998 \\ -999 & -1999 \end{bmatrix} y = f(t), \qquad a = 0 < t < b = 1,$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} y(a) + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} y(b) = g.$$

Here $f(t)$ and g are determined so that the exact solution is given by

$$y(t) = \begin{bmatrix} 2v_1 - v_2 \\ -v_1 + v_2 \end{bmatrix}$$

where

$$v(t) = \left(3e^{-t} + 3(e^{-t} - 1 + t), 5e^{-1000t} + 4 \times 10^{-6}(e^{-1000t} - 1 + 1000t)\right)^T.$$

3.1.3. Boundary layer. A boundary layer of width $\epsilon = 10^{-4}$ is present in the boundary value problem [22]

$$y' - \begin{bmatrix} 0 & 1 \\ \frac{1}{\epsilon^2} & 0 \end{bmatrix} y = f(t), \qquad a = -1 < t < b = 1$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} y(a) + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} y(b) = g.$$

Here $f(t)$ and g are determined so that the exact solution is given by $y(t) = (u(t), u'(t))^T$ where

$$u(t) = \cos(\pi t) + \frac{e^{-(1+t)/\epsilon}}{1 + e^{-2/\epsilon}} + e^{-(1-t)/\epsilon}.$$

The first component of the solution is shown in Fig. 1(a).

3.1.4. Bessel system. A vector of six Bessel functions and derivatives $J_\nu(t)$ with orders ν near $\nu = 10$ satisfies the boundary value problem [31] on $[0, 600]$

$$y' - \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ \frac{(\nu-t)(\nu+t)+\nu}{t^2} & 0 & \frac{-1}{t} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{(\nu-t)(\nu+t)-\nu}{t^2} & 0 & \frac{-1}{t} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{1}{t} & 0 & \frac{(\nu-t)(\nu+t)-5\nu+6}{t^2} & 0 \end{bmatrix} y = f(t),$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} y(a) + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} y(b) = g.$$

Here $f(t)$ and g have been determined so that the exact solution is given by

$$y(t) = \left(J_\nu(t), J'_\nu(t), J_{\nu-1}(t), J'_{\nu-1}(t), J_{\nu-2}(t), J'_{\nu-2}(t)\right)^T.$$

The first component of the solution is shown in Fig. 1(b).

3.1.5. Airy turning point. An appropriate combination of Airy functions [1] satisfies the quantum-mechanical "turning point problem" [23, 11, 24]

$$y' - \begin{bmatrix} 0 & 1 \\ \frac{t}{\epsilon} & 0 \end{bmatrix} y = 0, \qquad a = -1 < t < b = 1$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} y(a) + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} y(b) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

with $\epsilon = 10^{-6}$. Here the exact solution is given by $y(t) = (u(t), u'(t))$ where

$$u(t) = C_1 \mathrm{Ai}(\delta t) + C_2 \mathrm{Bi}(\delta t)$$

$\delta = \epsilon^{-1/3} = 100$, $C_1 = 5.6576000136230$ and $C_2 = 1.6552936963622 \times 10^{-289}$ are constants. This problem contains almost every possible difficulty: dense oscillations, stiffness, a turning point where the solution changes character, and a sharp boundary layer at the right edge. The first component of the solution is shown in Fig. 1(c).

3.1.6. Parabolic cylinder functions. An appropriate parabolic cylinder function satisfies the highly ill-conditioned boundary value problem [17, 24]

$$y' - \begin{bmatrix} 0 & 1 \\ \frac{-1}{\epsilon} & \frac{t}{\epsilon} \end{bmatrix} y = 0, \qquad a = -1 < t < b = 1$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} y(a) + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} y(b) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

with $\epsilon = 1/70$. Here the exact solution is given by $y(t) = (u(t), u'(t))^T$ where

$$u(t) = \frac{t}{2} + \frac{3}{2M\left(\frac{1}{2\epsilon}\right)} M\left(\frac{t^2}{2\epsilon}\right)$$

with M the parabolic cylinder function [1]

$$M(x) = \sum_{n=0}^{\infty} \frac{-1}{2n-1} \frac{x^n}{n!}.$$

Since the condition number of this test problem is about $\kappa = 10^{15}$ and the best accuracy expected from any backward stable numerical method in double precision is of order $O(\kappa \times 10^{-14}) = O(1)$, our code does very well to obtain six-digit accuracy in less than 1 sec of CPU time.

3.2. Equidistant mesh results. We verify the high-order accuracy of `bvpdc` by solving the six test problems of §3.1 on equidistant n-point meshes. We used orders $p = 2$ through 20 and $n = 16$ through 65536 to solve each test problem and plotted the error vs. CPU time in Fig. 2. The plots exhibit the expected order of convergence to 12-digit accuracy, except for the ill-conditioned parabolic cylinder problem (3.1.6).

Our high-order methods obtain much better accuracy per unit CPU time than low-order methods, up to about order 12 or so. For example, 4 corrections (10th order) gives 12-digit accuracy on the Airy problem (3.1.5), in the 10 CPU sec that the uncorrected second-order midpoint rule requires to get three-digit accuracy. Two factors contribute to this highly stable behavior: the stability of QR factorization, and the similarity between deferred correction and iterative improvement for the solution of ill-conditioned linear systems [18].

The inevitable roundoff error begins to reduce accuracy for very large N, because the linear system $M_h c = \rho$ has condition number $\kappa = O(N) \approx 10^5$. Since the backward error is on the order of κ times the residual, we cannot expect better than 12-digit accuracy with $N = 65536$ even if the residual ρ approaches machine precision.

3.3. Adaptive mesh results. We verify the reliability of the adaptive mesh strategy in `bvpdc` by solving each test problem in §3.1 on a sequence of adaptively generated meshes with tolerances $\epsilon = 10^{-3}$ through 10^{-10}, using orders $p = 4$ through 20. The resulting errors vs. CPU times are plotted in Fig. 3. Table 1 presents adaptive mesh statistics at orders $p = 4$, 8, 12 and 16, which clearly demonstrate the superior efficiency of high-order methods with $p \geq 8$ even for rather coarse tolerances such as $\epsilon = 10^{-6}$.

TABLE 1

Statistics for the adaptive method with tolerance $\epsilon = 10^{-6}$ *and orders* $p = 4$, 8, 12 *and* 16*: number of mesh points* N*, mesh ratio* $R = \max_j h_j / \min h_j$*, accumulated CPU time* $T = T_1 + \cdots + T_S$*, maximum error* $E = max_j \|u(t_j) - y(t_j)\|$ *and error estimate* $C = max_t \|c(t)\|$ *at each step* 0, 1, ..., *of the adaptive refinement procedure. (High-order results agree with low-order results for the first few steps, and are not repeated in the tables. Asterisks denote cases where the fourth-order method terminated early due to lack of space.)*

Example	p	Step	N	R	T	E	C
3.1.1		0	491	1	.06	.16-11	.38-5
	4	1	4709	1.41	.57	.16-13	.43-7
	8	1	1163	1.34	.23	.16-13	.62-13
	12	1	823	1.19	.22	.51-14	.32-14
	16	1	710	1.13	.28	.16-13	.19-14
3.1.2		0	491	1	.01	.45-1	.14-0
	4	1	1493	1.14	.05	.44-3	.81-2
		2	13822	1.23	.36	.24-7	.77-4
		3	127979	1.33	3.23	.57-11	.77-6
	8	2	3209	5.23	.19	.79-10	.36-9
	12	2	2355	2.84	.27	.77-9	.26-8
	16	2	2066	2.18	.41	.39-8	.13-7
3.1.3		0	491	1	.02	.38+1	.11+1
	4	1	1617	1.86	.08	.89-0	.14+1
		2	5133	3.49	.28	.97-2	.58-1
		3	51319	3.49	2.2	.54-6	.42-3
		4	500000	3.49	22	.12-8	.20-4*
	8	3	11159	16.2	.94	.40-10	.69-9
	12	3	8171	10.5	1.14	.34-8	.21-7
	16	3	7152	7.65	1.63	.87-9	.55-8
3.1.4		0	491	1	.15	.15+1	.58-0
	4	1	2196	1.74	.83	.44-0	.63-0
		2	7769	3.14	3.20	.45-1	.78-1
		3	77655	4.44	27.1	.34-5	.11-3
		4	500000	6.28	29.5	.35-5	.11-3*
	8	3	77501	16.4	32.1	.89-10	.22-8
	12	3	31780	9.82	18.1	.16-5	.17-5
		4	46995	9.82	40.1	.31-6	.34-6
	16	3	21246	7.09	15.9	.14+1	.98-0
		4	29245	6.89	33.6	.86-4	.92-4
		5	38913	6.88	56.8	.65-6	.71-6
3.1.5		0	491	1	.01	.73+1	.10+1
	4	1	1745	1.71	.07	.66+1	.98+0
		2	6926	2.99	.30	.11+1	.11+1
		3	29091	5.26	1.23	.16-2	.64-1
		4	290866	5.26	10.6	.15-6	.62-3
		5	500000	2.29	26.7	.13-6	.55-4*
	8	4	221324	24.4	13.4	.27-9	.26-8
	12	4	100594	15.7	11.4	.56-9	.22-8
	16	4	69594	11.5	14.0	.39-9	.16-8

Our strategy obtains high accuracy in CPU times comparable to the nonadaptive code for all test problems. It also compares favorably with the best Netlib codes tested (§3.5). For example, both with and without an adaptive mesh, `bvpdc` obtains 12-digit accuracy on a boundary layer of width 10^{-4} (3.1.3) in CPU time comparable to `mirkdc` (order 2, 4 or 6) or `twpbvp`.

3.4. High-order convergence on the beam problem. We demonstrate high-order convergence on the beam problem (3.1.1) with 0 through 9 deferred correction steps on an equidistant n-point mesh, in quadruple-precision arithmetic with machine precision about 10^{-28}. For this high-precision convergence study, we compiled `bvpdc` with the Sun Fortran compiler `f77` and compiler flags `-xtypemap=real:64,double:128`. The quadruple precision executable runs orders of magnitude slower than double precision, but allows us to verify higher orders of convergence.

Table 2 clearly indicates that orders $p = 2$ through 20 are achieved after $m = p/2-1$ correction steps, and suggests that the classical convergence analysis of deferred correction with uncentered difference formulas [16, 29] may be incomplete.

Table 2
Order-p errors in the beam problem (3.1.1) on a uniform n-point mesh.

p	$n = 512$	1024	2048	4096	8192	16384	32768	65536
2	6.6-1	1.2-0	4.3-0	5.8-1	3.1-1	1.0-1	2.9-2	7.3-3
4	3.9-2	3.6-3	1.9-4	8.3-6	4.7-7	2.8-8	1.7-9	1.1-10
6	2.4-2	1.1-4	6.1-6	6.9-8	3.7-10	8.1-12	1.5-13	2.4-15
8	2.1-2	1.2-4	1.3-6	4.4-9	8.4-12	1.2-14	2.0-17	5.4-20
10	1.4-2	4.3-5	1.7-7	1.8-10	9.5-14	3.4-17	1.0-20	2.7-24
12	1.1-2	1.2-5	2.3-8	7.8-12	1.2-15	1.1-19	8.5-24	5.8-28
14	1.6-2	5.6-6	3.2-9	3.4-13	1.4-17	3.6-22	7.1-27	3.0-29
16	4.4-2	1.8-6	4.4-10	1.5-14	1.8-19	1.2-24	5.8-30	5.0-30
18	6.3-2	7.1-7	6.3-11	6.6-16	2.2-21	3.9-27	9.4-30	1.8-29
20	3.2-2	2.6-7	8.9-12	3.0-17	2.7-23	1.5-29	1.4-29	2.2-29

3.5. Comparison codes. We compared `bvpdc` to four freely-available Fortran 77 codes from Netlib—`colnew`, `mirkdc`, `musl`, and `twpbvp`. Such comparisons are intended to evaluate the potential usefulness of our simple technique for solving difficult problems. Fig. 4 exhibits base-10 log-log plots of the maximum error vs. CPU milliseconds required by each Netlib code to solve each test problem with tolerances $\epsilon = 10^{-2}, 10^{-4}, \ldots, 10^{-10}$.

`colnew` [3], an updated version of `colsys` [2], uses collocation at Gaussian points [28] with an improved basis replacing the B-splines of `colsys`. It solves extremely general linear and nonlinear multipoint BVPs of mixed orders up to four, requires separated boundary conditions, and automatically generates a mesh on which the error is approximately equidistributed. The results of `colnew` are indicated by a "`C`" in Fig. 4.

`mirkdc` [20] solves nonlinear first-order BVPs with separated boundary conditions by deferred correction of mono-implicit Runge-Kutta methods of orders 2, 4 or 6. The mesh is adapted to equidistribute an approximate residual or "defect" computed from a C^1 interpolant, so the "`dc`" in the name stands for "defect control" rather than deferred correction. The results of `mirkdc` with order 2, 4 or 6 are indicated by a "2", "4" or "6" respectively in Fig. 4. The occasional very short lines are due to the excessive accuracy obtained by `mirkdc` for some problems.

`musl` [26] uses an adaptive multiple shooting approach to solve nonstiff linear two-point BVPs, with results indicated by a "`M`" in Fig. 4.

`twpbvp` [10, 15, 12] implements two deferred correction steps based on 6th and 8th order MIRK formulas, yielding an 8th order solution. The mesh is successively doubled to achieve a user-specified accuracy tolerance and refined to equidistribute the last correction. Results are indicated with a "T" in Fig. 4.

Some of the Netlib comparison codes obtained suboptimal accuracy or failed to produce a solution at all. The collocation code `colnew` was not intended for stiff problems and failed to solve half of the test problems. The shooting method `musl` did well on the beam equation (3.1.1) and the stiff problem (3.1.2) but failed on all the other test problems. By contrast, the two codes `mirkdc` (operating in order-6 mode) and `twpbvp` were able to attain at least 6-digit accuracy in every test problem in our gallery.

3.6. Conclusions. We have developed and tested a simple, fast, stable deferred correction method for the numerical solution of two-point boundary value problems. The method handles nonseparated boundary conditions and ill-conditioned BVPs with a stable efficient sparse QR factorization. Numerical results show that the method can solve some extremely challenging problems with efficiency comparable to standard Netlib codes.

The deferred correction framework displays considerable promise as a high-order automatic solver. Useful extensions could well solve differential-algebraic problems and elliptic boundary value problems in higher dimensions.

Acknowledgments. The author thanks the referees for several helpful suggestions.

REFERENCES

[1] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*, Dover, NY, 1965.

[2] U. Ascher, J. Christiansen, and R. D. Russell, *A collocation software for mixed order systems of boundary-value problems*, Math. Comp., 33 (1979), pp. 659–679.

[3] ———, *Collocation software for boundary-value ODEs*, ACM Trans. Math. Softw., 7 (1981), pp. 209–222.

[4] U. M. Ascher, R. M. M. Mattheij, and R. D. Russell, *Numerical solution of boundary value problems for ordinary differential equations*, SIAM, 1995.

[5] Z. Bashir-Ali, J. R. Cash, and H. H. M. Silva, *Lobatto deferred correction for stiff two-point boundary value problems*, Comput. Math. Appl., 36 (1998), pp. 59–69.

[6] J.-P. Berrut and H. D. Mittelmann, *The linear rational pseudospectral method with iteratively optimized poles for two-point boundary value problems*, SIAM J. Sci. Comput., 23 (2001), pp. 961–975.

[7] K. Böhmer and H. J. Stetter, eds., *Defect correction methods: theory and applications*, vol. 5 of Computing Supplementum, Springer-Verlag, 1984.

[8] C. D. Boor and R. Weiss, *Solveblok: A package for solving almost block diagonal linear systems*, ACM Trans. Math Softw., 8 (1980), pp. 88–91.

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

[10] ———, *On the numerical integration nonlinear two-point boundary value problems using iterated deferred corrections. part 2: The development and anlaysis of highly stable deferred correction formulae*, SIAM J. Numer. Anal., 25 (1988), pp. 862–882.

[11] ———, *A comparison of some global methods for solving two-point boundary value problems*, Appl. Math. Comput., 31 (1990), pp. 449–462.

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

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

[14] J. R. Cash and A. Singhal, *Mono-implicit Runge-Kutta formulae for the numerical integration of stiff differential equations*, IMA J. Numer. Anal., 2 (1982), pp. 221–227.

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

[16] J. Christiansen and R. D. Russell, *Deferred corrections using uncentered end formulas*, Numer. Math., 35 (1980), pp. 21–33.

[17] P. P. N. de Groen, *The nature of resonance in a singular perturbation problem of turning point type*, SIAM J. Math. Analysis, 11 (1980), pp. 1–22.

[18] J. W. Demmel, *Applied numerical linear algebra*, SIAM, 1997.

[19] O. Edlund, *A software package for sparse orthogonal factorization and updating*, ACM Trans. Math Softw., 28 (2002), pp. 448–482.

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

[21] B. Fornberg, *Generation of finite difference formulas on arbitrarily spaced grids*, Math. Comput., 51 (1988), pp. 699–706.

[22] L. Greengard and V. Rokhlin, *On the numerical solution of two-point boundary value problems*, Comm. Pure Appl. Math., XLIV (1991), pp. 419–452.

[23] P. W. Hemker, *A numerical study of stiff two-point boundary value problems*, Mathematisch Centrum, Amsterdam, 1977.

[24] J.-Y. Lee and L. Greengard, *A fast adaptive numerical method for stiff two-point boundary value problems*, SIAM J. Sci. Comput., 18 (1997), pp. 403–429.

[25] M. Lentini and V. Pereyra, *An adaptive finite difference solver for nonlinear two-point boundary value problems with mild boundary layers*, SIAM J. Numer. Analysis, 14 (1977), pp. 91–111.

[26] R. M. M. Mattheij and G. V. M. Staarink, *An efficient algorithm for solving general linear two-point BVP*, SIAM J. Sci. Comput., 5 (1984), pp. 745–763.

[27] Y. Saad, *Iterative methods for sparse linear systems*, PWS Pub. Co., Boston, 1996.

[28] K. H. Schild, *Gaussian collocation via defect correction*, Numer. Math., 58 (1990), pp. 369–386.

[29] R. D. Skeel, *The order of accuracy for deferred correction using uncentered end formulas*, SIAM J. Num. Analysis, 23 (1986), pp. 393–402.

[30] H. P. Starr, *Rapid solution of one-dimensional integral and differential equations*, PhD thesis, Yale University Department of Computer Science, 1993.

[31] H. P. Starr and V. Rokhlin, *On the numerical solution of two-point boundary value problems II*, Research Report YALEU/DCS/RR-802, Yale University Department of Computer Science, 1990.

[32] H. J. Stetter, *Analysis of discretization methods for ordinary differential equations*, Springer, 1973.

[33] J. Stoer and R. Bulirsch, *Introduction to numerical analysis*, Springer-Verlag, 1980.

[34] W. M. G. Van Bokhoven, *Implicit end-point quadrature formulae*, BIT, 20 (1980), pp. 87–99.

[35] M. Van Daele and J. R. Cash, *Superconvergent deferred correction methods for first order systems of nonlinear two-point boundary value problems*, SIAM J. Sci. Comput., 22 (2000), pp. 1697–1716.

[36] M. Van Daele, T. Van Hecke, G. Vanden Berghe, and H. De Meyer, *Deferred correction with mono-implicit Runge-Kutta methods for first order IVPs*, J. Comput. Appl. Math., 111 (1999), pp. 37–47.

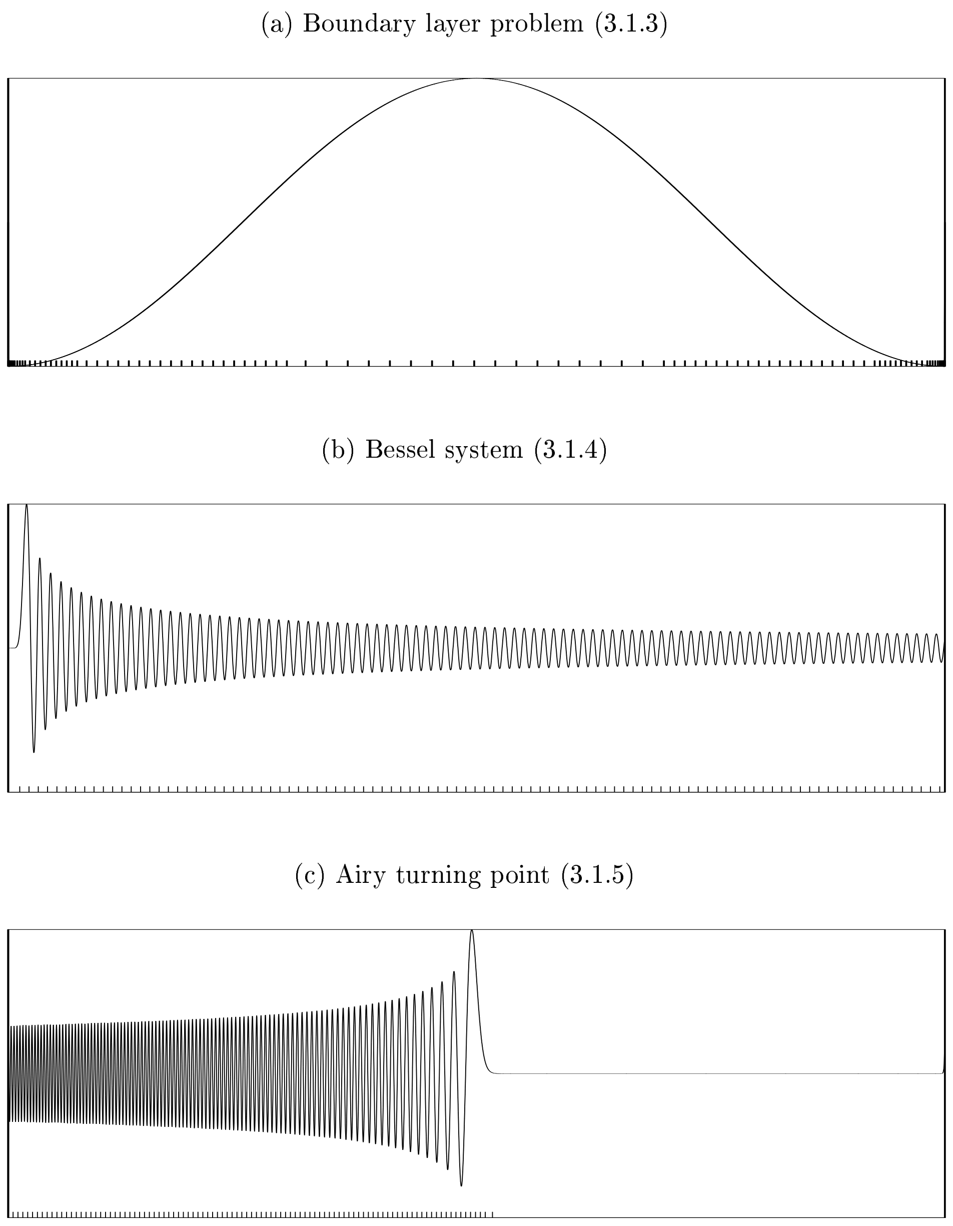

FIG. 1. *The first component of the solutions of the boundary layer problem (3.1.3), the Bessel system (3.1.4), and Airy turning point problem (3.1.5), with the relative density of the adaptive mesh produced by* `bvpdc` *for error tolerance* $\epsilon = 10^{-3}$.

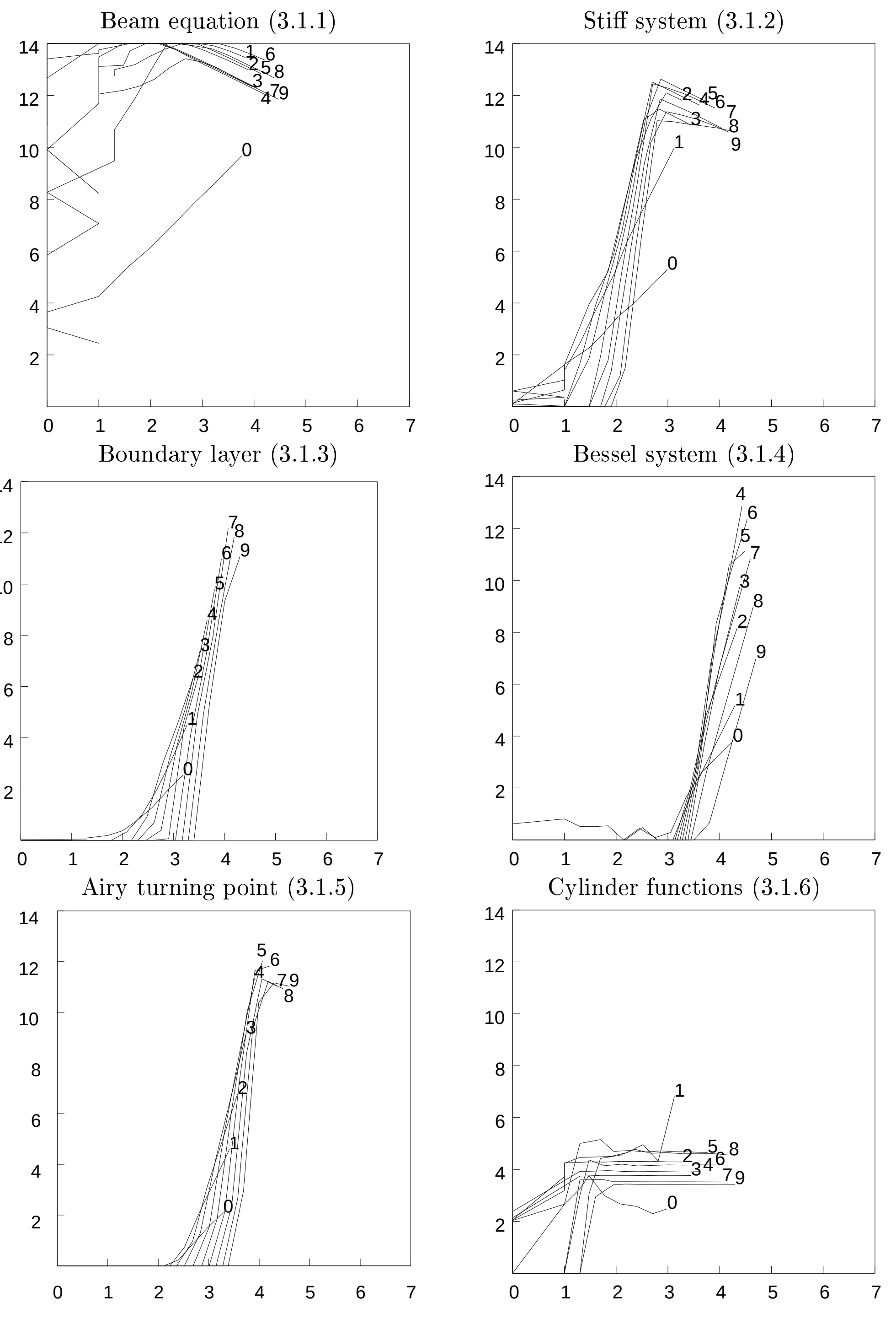

FIG. 2. *Base-10 logarithm of error vs. logarithm of CPU milliseconds with 0 to 9 steps of deferred correction yielding schemes of orders 2 through 20 on uniform meshes.*

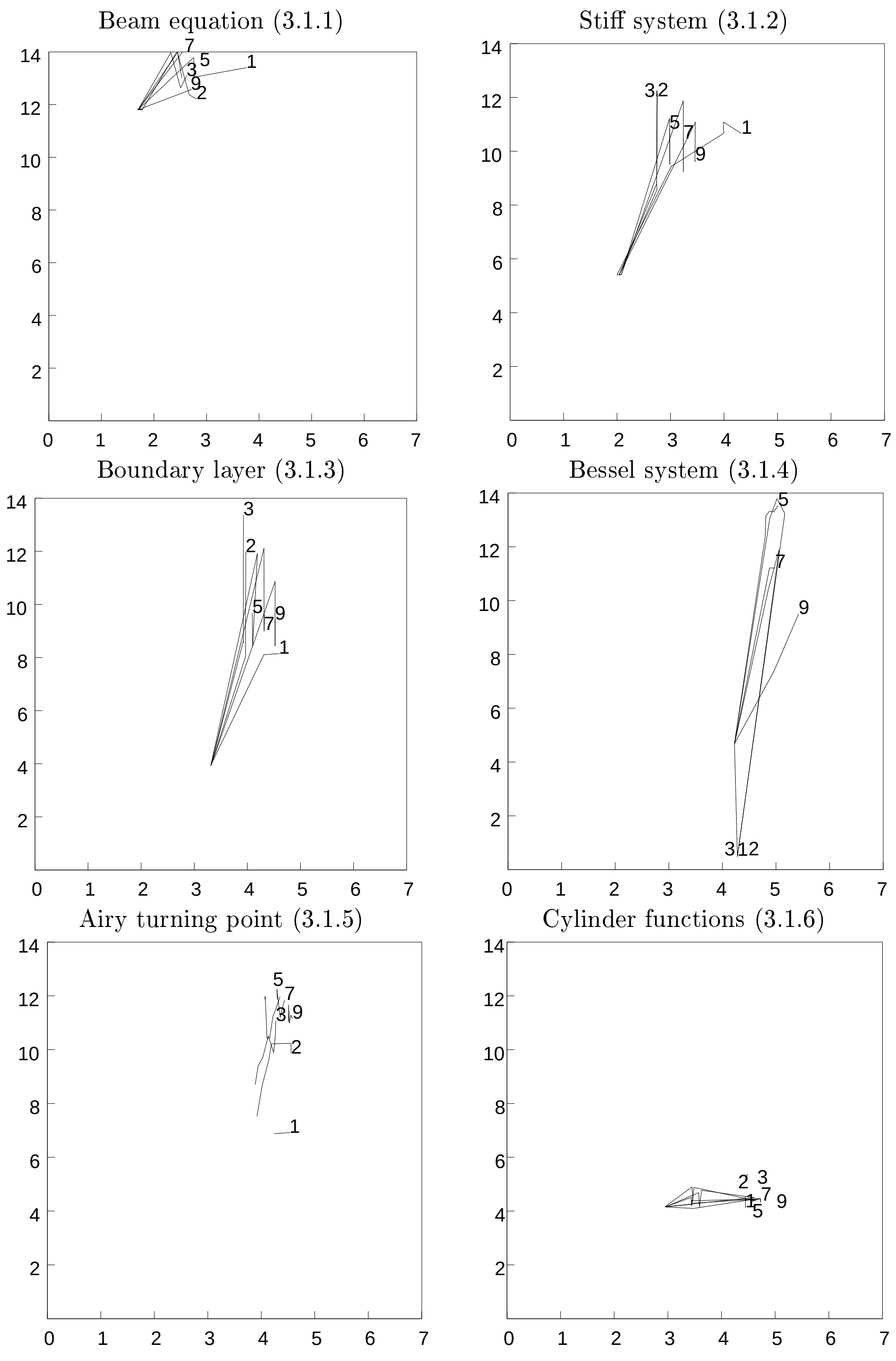

FIG. 3. *Base-10 log-log plot of error vs. CPU milliseconds with 1 to 9 steps of deferred correction yielding schemes of orders 4 through 20 on adaptive meshes.*

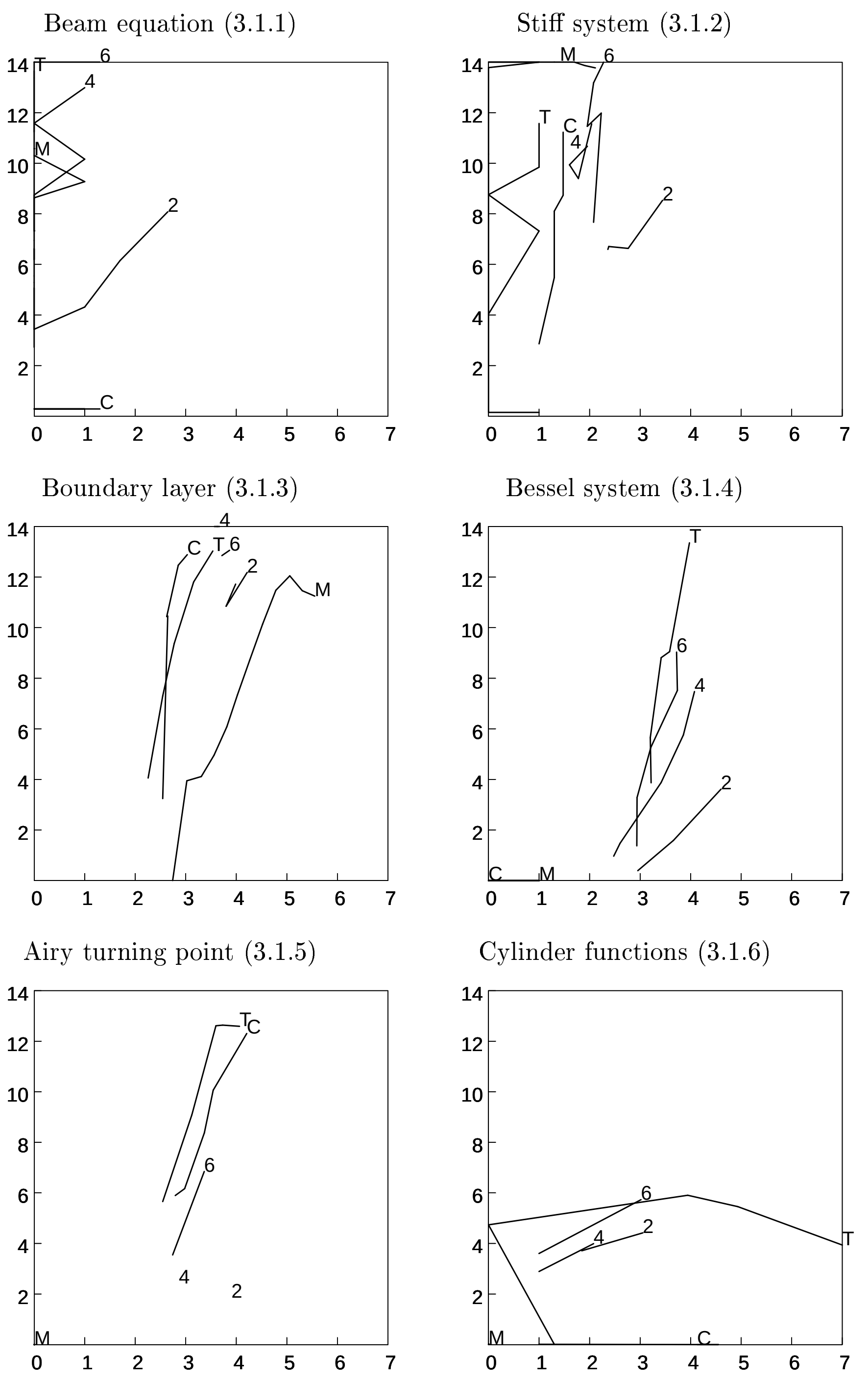

FIG. 4. *Base-10 logarithm of error vs. logarithm of CPU milliseconds with three standard adaptive codes from Netlib: "C" indicates results for* `colnew`*, "M" for* `musl`*, "T" for* `twpbvp`*, and numerals 2 through 6 indicate* `mirkdc` *results of the corresponding order.*