SIAM J. SCI. COMPUT.
Vol. 18, No. 2, pp. 403–429, March 1997

004

A FAST ADAPTIVE NUMERICAL METHOD FOR STIFF TWO-POINT BOUNDARY VALUE PROBLEMS*

JUNE-YUB LEE† AND LESLIE GREENGARD‡

Abstract. We describe a robust, adaptive algorithm for the solution of singularly perturbed two-point boundary value problems. Many different phenomena can arise in such problems, including boundary layers, dense oscillations, and complicated or ill-conditioned internal transition regions. Working with an integral equation reformulation of the original differential equation, we introduce a method for error analysis which can be used for mesh refinement even when the solution computed on the current mesh is underresolved. Based on this method, we have constructed a black-box code for stiff problems which automatically generates an adaptive mesh resolving all features of the solution. The solver is direct and of arbitrarily high-order accuracy and requires an amount of time proportional to the number of grid points.

Key words. mesh refinement, integral equations, singular perturbations problems

AMS subject classifications. 34B05, 34B27, 34A50, 65L10, 41A25

PII. S1064827594272797

1. Introduction.

In this paper we describe a robust, automatic, adaptive algorithm for the solution of two-point boundary value problems of the form

$$u''(x) + p(x)u'(x) + q(x)u(x) = f(x) \tag{1}$$

or, more suggestively,

$$\epsilon u''(x) + p(x)u'(x) + q(x)u(x) = f(x), \tag{2}$$

where ϵ is a small parameter. Many different phenomena can arise in such problems, including boundary layers, dense oscillations, and complicated or ill-conditioned internal transition regions. We are primarily interested in the singularly perturbed or "stiff" case, usually written in the form (2), but we will take (1) to be the standard form of an equation.

The last few decades have seen substantial progress in the development of numerical methods for the solution of such problems and several excellent textbooks, such as [2] and [21], and software packages, such as COLSYS [3, 5], PASVAR [27], and MUS [29] are presently available. We do not seek to review the subject here, but we will briefly list some of the currently available strategies for mesh selection.

1. **Transformation methods** assume that one knows a priori where complicated features of the solution are to be found and introduce a change of variables. The solution of the transformed problem in the new variable is smooth and can be resolved on a simple, uniform mesh. A good reference is [25]. When applicable, such an approach provides rapid and accurate answers, but it is not an automatic strategy.

*Received by the editors August 12, 1994; accepted for publication (in revised form) July 24, 1995. This work was supported by the Applied Mathematical Sciences Program of the U.S. Department of Energy under contract DEFGO288ER25053, by an NSF Presidential Young Investigator Award, and by a Packard Foundation Fellowship.

http://www.siam.org/journals/sisc/18-2/27279.html

†Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (jylee@mathl.kaist.ac.kr). Current address: Department of Mathematics, Ewha Womans University, Seoul, 120-750, Rep. of Korea (jylee@math1.ewha.ac.kr).

‡Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (greengard@cims.nyu.edu).

2. **Multiple shooting and Ricatti methods** are based on the use of initial value problem solvers and have achieved some success for stiff problems. It is difficult, however, to ensure that one has resolved all features of the solution unless their location is known in advance. For a thorough discussion, see [2].

3. **Finite difference and collocation methods** are probably the most popular and best-developed general purpose solvers available [3, 5, 24, 27, 30]. There are two possible modes for mesh selection: the user can specify in advance a nonuniform mesh which resolves complicated features or the user can request that the solver construct a sequence of adaptively refined meshes as part of the solution process. In the latter case, the refinement strategy is based on a posteriori error estimation with varying levels of sophistication. In brief, once an approximate solution has been obtained, the code attempts to ascertain which subintervals incur the maximum error and subdivides those selectively. The problem with this approach is that it is unreliable until some, perhaps crude, resolution has been achieved. From that point on, a posteriori error analysis is quite robust.

Remark. For stiff problems, the goal of all the above strategies as well as our own) is the same: the construction of a mesh on which all features of the solution are locally smooth.

Our goal, however, is to be able to determine which subintervals require further refinement before any resolution has been achieved. For this, we work with an integral equation reformulation of the original two-point boundary value problem. Such an approach has generally been avoided due to the excessive computational cost associated with solving dense linear systems. A finite difference or collocation method requires only $O(N)$ arithmetic operations to solve a discretized problem with N mesh points. A straightforward integral equation method requires $O(N^3)$ work for the same size problem. In the last few years, however, several fast algorithms have been introduced which allow for the direct solution of the integral equation in only $O(N)$ or $O(N \log N)$ operations [4, 7, 8, 11, 12, 19, 20, 28, 31]. Of particular relevance are the papers [20], which considers a single scalar equation of the form (1), and [31], which extends this method to first-order systems. These schemes are well conditioned and of arbitrary (but fixed) order accuracy. They perform well even for ill-behaved equations such as high-order Bessel equations and problems with internal or boundary layers. They do not, however, address the question of mesh selection.

In this paper, we introduce a method for error analysis which can be used even when the solution computed on the current mesh yields no accuracy. We also show how to incorporate this method into an adaptive version of the fast integral equation solver mentioned above. The result is a "black box" code for stiff equations whose performance is demonstrated on a suite of test problems in section 4.

2. Mathematical preliminaries.

In this section, we begin by summarizing the standard integral equation approach to the solution of two-point boundary value problems. We then consider the mathematical framework of a recursive solver, based on the algorithm of [20].

2.1. Green's functions for second-order ordinary differential equations.

Consider the problem of determining a function u in $C^2[a, c]$ which satisfies the second-order differential equation

$$u''(x) + p(x)u'(x) + q(x)u(x) = f(x) \tag{3}$$

on the interval $[a, c] \subset \mathbf{R}$, where p, q are continuous functions on (a, c), subject to the linear boundary conditions

$$\zeta_{l0} \cdot u(a) + \zeta_{l1} \cdot u'(a) = \Gamma_l, \tag{4}$$

$$\zeta_{r0} \cdot u(c) + \zeta_{r1} \cdot u'(c) = \Gamma_r. \tag{5}$$

It is well known that the solution of the original equation (3) can be decomposed into two terms, $u = u_h + u_i$, where u_i is a linear (or for pure Neumann conditions a quadratic) function which satisfies the inhomogeneous boundary conditions (4) and (5) and u_h solves the equation

$$u_h''(x) + p(x)u_h'(x) + q(x)u_h(x) = \tilde{f}(x), \tag{6}$$

where

$$\tilde{f}(x) \equiv f(x) - (u_i''(x) + p(x)u_i'(x) + q(x)u_i(x)),$$

with homogeneous boundary conditions

$$\zeta_{l0} \cdot u_h(a) + \zeta_{l1} \cdot u_h'(a) = 0, \tag{7}$$

$$\zeta_{r0} \cdot u_h(c) + \zeta_{r1} \cdot u_h'(c) = 0. \tag{8}$$

We will represent the solution to equations (6), (7), and (8) in terms of Green's function for a simpler problem by making use of the following lemma [10].

LEMMA 2.1. *Let $q_0 \in C^1(a, c)$ and suppose that the equation*

$$\phi''(x) + q_0(x)\phi(x) = 0 \tag{9}$$

subject to the homogeneous boundary conditions (7) *and* (8) *has only the trivial solution. Then there exist two linearly independent functions $g_l(x), g_r(x)$ which satisfy equation* (9) *and the boundary conditions* (7) *and* (8)*, respectively. Green's function for this equation, denoted by G_0, can be constructed as follows:*

$$G_0(x,t) = \begin{cases} g_l(x)g_r(t)/s & \text{if } x \le t, \\ g_l(t)g_r(x)/s & \text{if } x \ge t, \end{cases} \tag{10}$$

where s is a constant given by $s = g_l(x)g_r'(x) - g_l'(x)g_r(x)$.

Given Green's function G_0, any twice differentiable function satisfying the boundary conditions (7) and (8) can be uniquely represented in the form

$$\phi(x) = \int_a^c G_0(x,t) \cdot \sigma(t)dt, \tag{11}$$

where $\sigma(x)$ is an unknown density function. In order for the function ϕ, represented by (11), to satisfy the ordinary differential equation (6), $\sigma(x)$ must simply satisfy the second kind of integral equation

$$\sigma(x) + \tilde{p}(x) \int_a^c G_1(x,t)\sigma(t)dt + \tilde{q}(x) \int_a^c G_0(x,t)\sigma(t)dt = \tilde{f}(x), \tag{12}$$

where $\tilde{p}(x) = p(x)$ and $\tilde{q}(x) = q(x) - q_0(x)$, and

$$G_1(x,t) = \frac{d}{dx} G_0(x,t). \tag{13}$$

Once $\sigma(x)$ is known, then $u_h(x) = \phi(x)$ can be added to $u_i(x)$ to provide the solution of the original problem (3), (4), and (5). Similarly, $u'(x) = u'_h(x) + u'_i(x)$, where

$$u'_h(x) = \int_a^c G_1(x,t)\sigma(t)dt. \tag{14}$$

Of course, if $p(x) = 0$ and $q_0(x) = q(x)$, then the solution to equation (12) is just $\sigma = \tilde{f}$. In practice, however, we choose q_0 to be a nonpositive constant so that Green's function G_0 is readily available, while Green's function for the original system is not. In particular, it is easy to verify the following result.

LEMMA 2.2. *If $|\zeta_{l0}| \geq |\zeta_{l1}|$ or $|\zeta_{r0}| \geq |\zeta_{r1}|$, then Green's function corresponding to $q_0(x) = 0$ in equation* (9) *can be constructed from*

$$g_l(x) = \zeta_{l0}(x-a) - \zeta_{l1} \tag{15}$$

$$g_r(x) = \zeta_{r0}(x-c) - \zeta_{r1}. \tag{16}$$

If both $|\zeta_{l0}| < |\zeta_{l1}|$ and $|\zeta_{r0}| < |\zeta_{r1}|$, then Green's function corresponding to $q_0(x) = -1$ in equation (9) *can be constructed from*

$$g_l(x) = \zeta_{l1}\cosh(x-a) - \zeta_{l0}\sinh(x-a), \tag{17}$$

$$g_r(x) = \zeta_{r1}\cosh(x-c) - \zeta_{r0}\sinh(x-c). \tag{18}$$

2.2. Notation. We define the operator $P : L^2[a,c] \to L^2[a,c]$ as follows:

$$P\eta(x) = \eta(x) + \tilde{p}(x)\int_a^c G_1(x,t)\eta(t)dt + \tilde{q}(x)\int_a^c G_0(x,t)\eta(t)dt. \tag{19}$$

The integral equation (12) can then be written in the form

$$P\sigma = \tilde{f}. \tag{20}$$

In terms of the component functions g_l and g_r which define Green's function G_0, we have

$$P\eta(x) = \eta(x) + \psi_l(x)\int_a^x g_l(t)\eta(t)dt + \psi_r(x)\int_x^c g_r(t)\eta(t)dt, \tag{21}$$

where

$$\psi_l(x) = (\tilde{p}(x)g'_r(x) + \tilde{q}(x)g_r(x))/s, \tag{22}$$

$$\psi_r(x) = (\tilde{p}(x)g'_l(x) + \tilde{q}(x)g_l(x))/s. \tag{23}$$

Consider now a subinterval $B = [b_l, b_r] \subset [a,c]$ and let us denote the restriction of η to B by η_B and the restriction of P to B by P_B. That is $P_B : L^2(B) \to L^2(B)$ and for $x \in B$,

$$\begin{aligned} P_B\eta_B(x) &= \eta_B(x) + \tilde{p}(x)\int_{b_l}^{b_r} G_1(x,t)\eta_B(t)dt + \tilde{q}(x)\int_{b_l}^{b_r} G_0(x,t)\eta_B(t)dt \\ &= \eta_B(x) + \psi_l(x)\int_{b_l}^{x} g_l(t)\eta_B(t)dt + \psi_r(x)\int_x^{b_r} g_r(t)\eta_B(t)dt. \end{aligned} \tag{24}$$

Note that we would like to find $\sigma(x) = P^{-1}\tilde{f}(x)$ but that the linear system obtained from discretization of P is dense and computationally unattractive. There is, however, a remarkable relationship between σ_B, the restriction of the solution of the full

integral equation to a subinterval B, and the function $P_B^{-1}\tilde{f}(x)$. To understand this relationship, we divide $[a,c]$ into three pieces, namely $A=[a,b_l], B=[b_l,b_r]$, and $C=[b_r,c]$. Then for $x\in B$,

$$\begin{aligned} P\sigma(x) &= P_B\sigma_B(x) + \psi_l(x)\int_a^{b_l} g_l(t)\sigma_A(t)dt + \psi_r(x)\int_{b_r}^{c} g_r(t)\sigma_C(t)dt \\ &= P_B\sigma_B(x) - \psi_l(x)\lambda_l^B - \psi_r(x)\lambda_r^B = \tilde{f}(x), \end{aligned} \tag{25}$$

where λ_l^B and λ_r^B are given by

$$\begin{aligned} \lambda_l^B &= -\int_a^{b_l} g_l(t)\sigma(t)dt, \\ \lambda_r^B &= -\int_{b_r}^{c} g_r(t)\sigma(t)dt. \end{aligned} \tag{26}$$

DEFINITION 2.1. *The constants λ_l^B and λ_r^B will be referred to as coupling coefficients.*

An immediate consequence of equation (25) is the following fundamental formula.

LEMMA 2.3. *Given the coupling coefficients λ_l^B and λ_r^B, the restriction of the solution of the global integral equation* (20) *to a subinterval B can be expressed as a linear combination of the solution of purely local problems:*

$$\sigma_B(x) = P_B^{-1}\tilde{f}(x) + \lambda_l^B P_B^{-1}\psi_l(x) + \lambda_r^B P_B^{-1}\psi_r(x). \tag{27}$$

This leads to an efficient method for solving the original equation (20), for we can subdivide $[a,c]$ into a large number of subintervals $B_1,\ldots,B_M$ and apply this observation on each one. The local problems are much less expensive to solve than (20) and it remains only to somehow determine the coupling coefficients $\lambda_l^{B_i}$ and $\lambda_r^{B_i}$ for each B_i.

2.3. A divide and conquer procedure. By Lemma 2.3, the solution of the integral equation (20) on a fixed subinterval B can be constructed as a linear combination of $P_B^{-1}\psi_l(x), P_B^{-1}\psi_r(x)$, and $P_B^{-1}\tilde{f}(x)$. In order to efficiently compute the coupling coefficients λ_l^B and λ_r^B, however, we will need to develop a number of analytic relations which they satisfy. We will return to the integral equation itself in section 2.4. For the moment then let us suppose that η_B satisfies

$$P_B\,\eta_B = \mu_l^B\psi_l + \mu_r^B\psi_r + \mu^B\tilde{f}, \tag{28}$$

where μ_l^B, μ_r^B, and μ^B are *given* constants. We subdivide B into a left and a right subinterval, denoted by D and E, respectively, and refer to D and E as B's children. The restriction of η_B to D and E will be denoted by η_D and η_E. From the discussion in section 2.2, it is easy to see that there exist coefficients $\mu_l^D, \mu_r^D, \mu^D, \mu_l^E, \mu_r^E$, and μ^E, such that

$$P_D\,\eta_D = \mu_l^D\psi_l + \mu_r^D\psi_r + \mu^D\tilde{f}, \tag{29}$$

$$P_E\,\eta_E = \mu_l^E\psi_l + \mu_r^E\psi_r + \mu^E\tilde{f}. \tag{30}$$

DEFINITION 2.2. *The coefficients μ_l^D, μ_r^D, μ^D, μ_l^E, μ_r^E, and μ^E, which define the right-hand sides in equations* (29) *and* (30)*, will be referred to as the refinement of the coefficients μ_l^B, μ_r^B, and μ^B, which define the right-hand side in equation* (28)*.*

We will require a number of inner products given by the following.

DEFINITION 2.3. *Let X denote a subinterval of $[a,c]$. Then*

$$\alpha_l^X \equiv \int_X g_l(t) P_X^{-1} \psi_l(t) dt, \qquad \alpha_r^X \equiv \int_X g_r(t) P_X^{-1} \psi_l(t) dt, \tag{31}$$

$$\beta_l^X \equiv \int_X g_l(t) P_X^{-1} \psi_r(t) dt, \qquad \beta_r^X \equiv \int_X g_r(t) P_X^{-1} \psi_r(t) dt, \tag{32}$$

$$\delta_l^X \equiv \int_X g_l(t) P_X^{-1} \tilde{f}(t) dt, \qquad \delta_r^X \equiv \int_X g_r(t) P_X^{-1} \tilde{f}(t) dt. \tag{33}$$

LEMMA 2.4. *Suppose that we are given the coefficients μ_l^B, μ_r^B, μ^B in equation (28). Then their refinements are given by*

$$\begin{aligned} \mu^D &= \mu^B, \\ \mu^E &= \mu^B, \\ \mu_l^D &= \mu_l^B, \\ \mu_r^E &= \mu_r^B, \\ \begin{pmatrix} \mu_r^D \\ \mu_l^E \end{pmatrix} &= \begin{pmatrix} 1 & \alpha_r^E \\ \beta_l^D & 1 \end{pmatrix}^{-1} \begin{pmatrix} \mu_r^B(1-\beta_r^E) - \mu^B \delta_r^E \\ \mu_l^B(1-\alpha_l^D) - \mu^B \delta_l^D \end{pmatrix}. \end{aligned} \tag{34}$$

Proof. We first expand $P_B\, \eta_B$ in the form

$$\begin{aligned} P_B\, \eta_B &= \eta_B(x) + \psi_l(x) \int_{b_l}^{x} g_l(t) \eta_B(t) dt + \psi_r(x) \int_x^{b_r} g_r(t) \eta_B(t) dt \\ &= \begin{cases} P_D\, \eta_D + \psi_r(x)(g_r, \eta_E) & \text{if } x \in D, \\ P_E\, \eta_E + \psi_l(x)(g_l, \eta_D) & \text{if } x \in E, \end{cases} \end{aligned} \tag{35}$$

where

$$\begin{aligned} (g_r, \eta_E) &= \int_E g_r(t) \eta_E(t) dt, \\ (g_l, \eta_D) &= \int_D g_l(t) \eta_D(t) dt. \end{aligned}$$

Using equations (28), (29), (30), and (35), it is easy to see that

$$P_B\, \eta_B = \mu_l^B \psi_l + \mu_r^B \psi_r + \mu^B \tilde{f} \tag{36}$$

$$= \begin{cases} \mu_l^D \psi_l + (\mu_r^D + (g_r, \eta_E)) \psi_r + \mu^D \tilde{f} & \text{if } x \in D, \\ (\mu_l^E + (g_l, \eta_D)) \psi_l + \mu_r^E \psi_r + \mu^E \tilde{f} & \text{if } x \in E. \end{cases} \tag{37}$$

The first four relations in (34) can now be obtained by comparing the coefficients of ψ_l, ψ_r, and $\tilde{f}$. For μ_l^E and μ_r^D, we have the more complicated conditions

$$\mu_l^B = \mu_l^E + (g_l, \eta_D) = \mu_l^E + \mu_l^D \alpha_l^D + \mu_r^D \beta_l^D + \mu^D \delta_l^D \tag{38}$$

$$\mu_r^B = \mu_r^D + (g_r, \eta_E) = \mu_r^D + \mu_l^E \alpha_r^E + \mu_r^E \beta_r^E + \mu^E \delta_r^E. \tag{39}$$

The remaining relations in (34) follow in a straightforward manner. ∎

If we were only interested in one level of subdivision, then we would allow B to play the role of the full interval $[a, c]$ and we would set $\mu_l^B = \mu_r^B = 0$, $\mu^B = 1$, and $\eta_B = \sigma_B$. We would obtain the solution to the global equation by first computing $P_D^{-1}\psi_l, P_D^{-1}\psi_r, P_D^{-1}\tilde{f}, P_E^{-1}\psi_l, P_E^{-1}\psi_r$, and $P_E^{-1}\tilde{f}$ and subsequently computing the inner products

$$\alpha_l^D, \alpha_r^D, \beta_l^D, \beta_r^D, \delta_l^D, \delta_r^D, \alpha_l^E, \alpha_r^E, \beta_l^E, \beta_r^E, \delta_l^E, \delta_r^E.$$

Lemma 2.4 then provides the refined coefficients needed on the two subintervals D and E. It is easy to see that by virtue of the specific choice $\mu_l^B = \mu_r^B = 0$, $\mu^B = 1$, the refined coefficients *are* the coupling coefficients defined in (26). Thus, we can evaluate σ_D and σ_E via Lemma 2.3. We have allowed a more general definition of the coefficients μ_l^B, μ_r^B, and μ^B in order to simplify the proof of the next lemma. It describes how to compute the coefficients α, β, and δ for a parent interval given the corresponding coefficients for its children.

LEMMA 2.5. *Suppose that B is a subinterval with children D and E. Then*

$$\alpha_l^B = \frac{(1-\alpha_l^E)\cdot(\alpha_l^D - \beta_l^D\cdot\alpha_r^E)}{\Delta} + \alpha_l^E, \tag{40}$$

$$\alpha_r^B = \frac{\alpha_r^E\cdot(1-\beta_r^D)\cdot(1-\alpha_l^D)}{\Delta} + \alpha_r^D. \tag{41}$$

$$\beta_l^B = \frac{\beta_l^D\cdot(1-\beta_r^E)\cdot(1-\alpha_l^E)}{\Delta} + \beta_l^E, \tag{42}$$

$$\beta_r^B = \frac{(1-\beta_r^D)\cdot(\beta_r^E - \beta_l^D\cdot\alpha_r^E)}{\Delta} + \beta_r^D. \tag{43}$$

$$\delta_l^B = \frac{1-\alpha_l^E}{\Delta}\cdot\delta_l^D + \delta_l^E + \frac{(\alpha_l^E-1)\cdot\alpha_l^D}{\Delta}\cdot\delta_r^E, \tag{44}$$

$$\delta_r^B = \frac{1-\beta_r^D}{\Delta}\cdot\delta_r^E + \delta_r^D + \frac{(\beta_r^D-1)\cdot\alpha_r^E}{\Delta}\cdot\delta_l^D\ , \tag{45}$$

where $\Delta = 1 - \alpha_r^E\beta_l^D$.

Proof. Observe that by choosing $\mu_l^B = 1$, $\mu_r^B = 0$, and $\mu^B = 0$, equations (28), (29), and (30) and Lemma 2.4 together yield

$$P_B^{-1}\psi_l(x) = \begin{cases} P_D^{-1}\psi_l(x) - \frac{\alpha_r^E(1-\alpha_l^D)}{\Delta}P_D^{-1}\psi_r(x) & \text{for } x\in D, \\ \frac{1-\alpha_l^D}{\Delta}P_E^{-1}\psi_l(x) & \text{for } x\in E. \end{cases} \tag{46}$$

Similarly, choosing $\mu_l^B = 0$, $\mu_r^B = 1$, and $\mu^B = 0$ yields

$$P_B^{-1}\psi_r(x) = \begin{cases} \frac{1-\beta_r^E}{\Delta}P_D^{-1}\psi_r(x) & \text{for } x\in D, \\ P_E^{-1}\psi_r(x) - \frac{\beta_l^D(1-\beta_r^E)}{\Delta}P_E^{-1}\psi_l(x) & \text{for } x\in E \end{cases} \tag{47}$$

and choosing $\mu_l^B = 0$, $\mu_r^B = 0$, and $\mu^B = 1$ yields

$$P_B^{-1}\tilde{f}(x) = \begin{cases} P_D^{-1}\tilde{f}(x) + \frac{\alpha_r^E\delta_l^D - \delta_r^E}{\Delta}P_D^{-1}\psi_l(x) & \text{for } x\in D, \\ P_E^{-1}\tilde{f}(x) + \frac{\beta_l^D\delta_r^E - \delta_l^D}{\Delta}P_E^{-1}\psi_l(x) & \text{for } x\in E. \end{cases} \tag{48}$$

A small amount of algebra provides the desired results. ∎

2.4. Informal description of a recursive solver. The tools developed in the previous subsection suggest the following algorithm:

1. **Generate subinterval tree.**

Starting from the root interval $[a, c]$, recursively subdivide each interval into two smaller intervals until the intervals at the finest level are sufficiently small that the restricted integral equations can be solved directly and inexpensively. This procedure generates a *binary tree* with each *node* corresponding to a subinterval. The finest level intervals will be referred to as *leaf nodes*. Nodes which are not leaves will be referred to as *internal nodes*.

2. **Solve restricted integral equations on leaf nodes.**

For each leaf node B_i, compute $P_{B_i}^{-1}\psi_l$, $P_{B_i}^{-1}\psi_r$, and $P_{B_i}^{-1}\tilde{f}$. Compute the inner products α, β, and δ on each leaf directly from the definitions (31), (32), and (33).

3. **Sweep upward to compute $\boldsymbol{\alpha}, \boldsymbol{\beta}$, and $\boldsymbol{\delta}$.**

Use Lemma 2.5 to compute α, β, and δ at each internal node.

4. **Sweep downward to compute $\boldsymbol{\lambda}$.**

Once the coefficients α, β, and δ are available at all nodes, Lemma 2.4 can be used to generate the coupling coefficients λ_l and λ_r at all nodes. The process is initialized by setting $\mu_l^B = \mu_r^B = 0$ and $\mu^B = 1$ at the root node. The refined coefficients at the nodes B_i are then the necessary coupling coefficients.

5. **Construct solution to integral equation.**

Evaluate σ on each leaf node as a linear combination of local solutions via Lemma 2.3.

6. **Construct solution to boundary value problem.**

The solution of the original ordinary differential equation can be constructed from the representation (11).

3. An adaptive algorithm. The recursive procedure described in the previous section is exact; in other words, the solution of the global integral equation is constructed analytically from the solutions of local equations on subintervals. In this section, we discuss the discretization process. Once a fully discrete algorithm is available, we will describe our mesh selection strategy and give a detailed description of an adaptive solver.

3.1. Chebyshev discretization. For a nonnegative integer k, the Chebyshev polynomial $T_k(x)$ is defined on $[-1, 1]$ by the formula

$$T_k(\cos\theta) = \cos(k\theta).$$

Consider now a fixed positive integer K. The roots of T_K are real, located on $[-1, 1]$, and given by

$$\tau_j = \cos\frac{(2K - 2j + 1)\pi}{2K} \qquad \text{for } j = 1, \ldots, K.$$

If the values of a function $\phi(x)$ are specified at the nodes τ_j, the Chebyshev interpolant is defined on $[-1, 1]$ by

$$\overline{\phi}(x) = \sum_{k=0}^{K-1} \alpha_k T_k(x),$$

where the Chebyshev coefficients α_k are given by

$$\alpha_0 = \frac{1}{K}\sum_{j=1}^{K} \phi(\tau_j),$$

$$\alpha_k = \frac{2}{K}\sum_{j=1}^{K} \phi(\tau_j) T_k(\tau_j) \quad \text{for } k \geq 1. \tag{49}$$

For a thorough discussion of Chebyshev approximation, we refer the reader to [15] or [17].

DEFINITION 3.1. *Let* $\mathbf{S}$ *denote the space of continuous functions on* $[-1,1]$. *The operator* $\mathcal{C}_K^f : \mathbf{S} \to \mathbf{R}^K$ *which maps a function to the vector of Chebyshev coefficients* $(\alpha_0, \alpha_1, \ldots, \alpha_{K-1})$ *will be referred to as the forward Chebyshev transform. The operator* $\mathcal{C}_K^b : \mathbf{R}^K \to \mathbf{S}$ *which maps the vector* $(\alpha_0, \alpha_1, \ldots, \alpha_{K-1})$ *to the corresponding* K*-term Chebyshev expansion will be referred to as the backward Chebyshev transform. Note that*

$$\mathcal{C}_K^b \mathcal{C}_K^f(\phi) = \overline{\phi}.$$

One of the reasons we use Chebyshev discretization is that the integration of functions expanded in Chebyshev series is particularly simple (see, for example, [9, 19]).

LEMMA 3.1. *Suppose that* $f(x)$ *is a continuous function defined on* $[-1,1]$ *with Chebyshev series*

$$f(x) = \sum_{k=0}^{\infty} f_k T_k(x).$$

Then

$$F_l(x) = \int_{-1}^{x} f(t)\,dt = \sum_{k=0}^{\infty} a_k T_k(x), \tag{50}$$

where

$$\begin{aligned} a_k &= \tfrac{1}{2k}\,(f_{k-1} - f_{k+1}) \quad \text{for } k \geq 2, \\ a_1 &= \tfrac{1}{2}\,(2f_0 - f_2), \\ a_0 &= \sum_{k=1}^{\infty} (-1)^{k-1}\, a_k. \end{aligned} \tag{51}$$

Similarly,

$$F_r(x) = \int_{x}^{1} f(t)\,dt = \sum_{k=0}^{\infty} b_k T_k(x), \tag{52}$$

where

$$\begin{aligned} b_k &= \tfrac{1}{2k}\,(f_{k+1} - f_{k-1}) \quad \text{for } k \geq 2, \\ b_1 &= \tfrac{1}{2}\,(f_2 - 2f_0), \\ b_0 &= -\sum_{k=1}^{\infty} b_k. \end{aligned} \tag{53}$$

Proof. For proof, see [9], [15], or [17]. ∎

DEFINITION 3.2. *Let* $\mathbf{f}$ *denote the vector of Chebyshev coefficients* $(f_0, f_1, \ldots, f_{K-1})$, *let* $\mathbf{a}$ *denote the vector of coefficients* $(a_0, a_1, \ldots, a_{K-1})$ *defined in equation* (51), *and let* $\mathbf{b}$ *denote the vector of coefficients* $(b_0, b_1, \ldots, b_{K-1})$ *defined in equation* (53). *The mapping*

$$\mathcal{I}_l^s : \mathbf{f} \to \mathbf{a}$$

will be referred to as the left spectral integration matrix and the mapping

$$\mathcal{I}_r^s : \mathbf{f} \to \mathbf{b}$$

will be referred to as the right spectral integration matrix.

DEFINITION 3.3. *The left integration operator $\mathcal{I}_l$ is given by*

$$\mathcal{I}_l \equiv \mathcal{C}_K^b \mathcal{I}_l^s \mathcal{C}_K^f.$$

The right integration operator $\mathcal{I}_r$ is given by

$$\mathcal{I}_r \equiv \mathcal{C}_K^b \mathcal{I}_r^s \mathcal{C}_K^f.$$

It is easy to verify that $\mathcal{I}_l$ and $\mathcal{I}_r$ are exact for polynomials of degree $K-2$ for all $x \in [-1,1]$. It is also easy to see that these integration operators are exact for polynomials of degree $K-1$ at the Chebyshev nodes themselves since $T_K(x) = 0$ there.

When the interval of integration is of the form $B_i = [b_i, b_{i+1}]$ rather than $[-1,1]$, we can define analogous operators using scaled Chebyshev polynomials and the scaled Chebyshev nodes

$$\tau_j^i = \frac{b_{i+1}-b_i}{2}\tau_j + \frac{b_{i+1}+b_i}{2} = \frac{b_{i+1}-b_i}{2}\cos\frac{(2K-2j+1)\pi}{2K} + \frac{b_{i+1}+b_i}{2}, \tag{54}$$

for $j = 1, \ldots, K$.

DEFINITION 3.4. *The left and right integration operators for the interval $B_i = [b_i, b_{i+1}]$ will be referred to as $\mathcal{I}_l^i$ and $\mathcal{I}_r^i$, respectively.*

3.2. Discretization of the integral operator. We are now in a position to discretize the local integral operator defined in equation (24). For the subinterval $B_i = [b_i, b_{i+1}]$, we denote this operator by P_i. Letting $\overline{\sigma}_i(x) : B_i \to \mathbf{R}$ be a function of the form

$$\overline{\sigma}_i(x) = \sum_{k=0}^{K-1} \sigma_i^k T_k\left(\tfrac{2(x-b_i)}{b_{i+1}-b_i} - 1\right),$$

we approximate

$$P_i\overline{\sigma}_i(x) = \overline{\sigma}_i(x) + \psi_l(x)\int_{b_i}^{x} g_l(t)\overline{\sigma}_i(t)dt + \psi_r(x)\int_{x}^{b_{i+1}} g_r(t)\overline{\sigma}_i(t)dt$$

by

$$\overline{P}_i\overline{\sigma}_i(x) = \overline{\sigma}_i(x) + \psi_l(x)\cdot\mathcal{I}_l^i(g_l\overline{\sigma}_i)(x) + \psi_r(x)\cdot\mathcal{I}_r^i(g_r\overline{\sigma}_i)(x). \tag{55}$$

The fully discrete local integral equation then takes the form

$$\begin{aligned}\overline{P}_i\overline{\sigma}_i(\tau_j^i) &= \overline{\sigma}_i(\tau_j^i) + \psi_l(\tau_j^i)\cdot\mathcal{I}_l^i(g_l\overline{\sigma}_i)(\tau_j^i) + \psi_r(\tau_j^i)\cdot\mathcal{I}_r^i(g_r\overline{\sigma}_i)(\tau_j^i)\\ &= \tilde{f}(\tau_j^i), \quad j = 1, \ldots, K.\end{aligned} \tag{56}$$

Analogous linear systems are used to approximate $\overline{P}_i^{-1}\psi_l$ and $\overline{P}_i^{-1}\psi_r$. We will use the notation $\overline{P}_i$ to denote both the operator in equation (55) and the $K \times K$ matrix in equation (56). The meaning should be clear from the context. Once the local

integral operator is discretized in the form (56), it remains only to discretize the inner products α, β, and δ. This is done through spectral integration.

Suppose now that $[a, c]$ has been subdivided into M subintervals $B_1, \ldots, B_M$. Then we can denote by $\overline{f}$ the vector consisting of the values of $\tilde{f}$ at the scaled Chebyshev nodes of each interval $B_1, \ldots, B_M$ in turn. The global integral equation $P\sigma = \tilde{f}$ is then replaced by a (rather complicated) dense linear system $\overline{P}\overline{\sigma} = \overline{f}$. Note that we never form this matrix explicitly; we simply invert the system via the recursive solver.

3.3. Mesh refinement. Suppose now that we have solved the discrete system $\overline{P}\overline{\sigma} = \overline{f}$ on the mesh defined by the subintervals $B_1, \ldots, B_M$. We would then like to be able to estimate the error in the computed solution. More critically, when the computational mesh does not properly resolve the true solution, we would like to be able to detect which subintervals require further refinement. For this, we begin with the following lemma.

LEMMA 3.2. *Suppose that σ is a solution of the original integral equation $P\sigma = \tilde{f}$ and that $\overline{\sigma}$ is a solution of the discretized integral equation $\overline{P}\overline{\sigma} = \overline{f}$. Then the relative error of $\overline{\sigma}$ satisfies the following bound:*

$$\frac{\|\overline{\sigma} - \sigma\|_2}{\|\sigma\|_2} \leq \kappa(P) \left(\frac{\|(P - \overline{P})\overline{\sigma}\|_2}{\|\tilde{f}\|_2} + \frac{\|\overline{f} - \tilde{f}\|_2}{\|\tilde{f}\|_2} \right), \tag{57}$$

where $\kappa(P)$ is the condition number of the operator P.

Proof. The result follows from the following two estimates:

$$\begin{aligned} \|\overline{\sigma} - \sigma\|_2 &= \|P^{-1}(P\overline{\sigma} - P\sigma)\|_2 \\ &\leq \|P^{-1}\|_2 \ \left(\|P\overline{\sigma} - \overline{P}\overline{\sigma}\|_2 + \|\overline{P}\overline{\sigma} - P\sigma\|_2 \right) \end{aligned}$$

and

$$\|P\|_2 \, \|\sigma\|_2 \geq \|\tilde{f}\|_2. \qquad \square$$

Remark. The source term $\tilde{f}$ combines information from the original right-hand side f as well as the functions p and q. Thus, all three need to be resolved in order for the term $\|\overline{f} - \tilde{f}\|_2$ to be small.

Remark. An important feature of Lemma 3.2 is that it provides a sufficient condition for convergence based only on the computed solution. Such an estimate cannot easily be obtained for finite difference or finite element discretizations of the original equation (3), since the ordinary differential operator is unbounded in L^2. (One can obtain finite element estimates using the Sobolev space H^1, but numerical differentiation is then required.)

We can make the error analysis even more precise by studying the last two coefficients of the Chebyshev expansions of the solutions $\overline{\sigma}_i$ defined on the subintervals B_i.

LEMMA 3.3. *Suppose that we are representing the solution in terms of a piecewise linear Green's function. If the last two Chebyshev coefficients of $\overline{\sigma}_i(x)$ satisfy $\sigma_i^{K-1} = \sigma_i^{K-2} = 0$, then*

$$P_i\overline{\sigma}_i = \overline{P}_i\overline{\sigma}_i.$$

Proof. The result follows immediately from the fact that spectral integration is exact for polynomials of degree less than $p-1$. ∎

Lemma 3.3 assures us that if the last two coefficients σ_i^{K-1} and σ_i^{K-2} are zero on every interval, then $P\overline{\sigma} = \overline{P}\overline{\sigma}$. Combining Lemmas 3.2 and 3.3, we have the following.

THEOREM 3.4. *Suppose that we are representing the solution in terms of a piecewise linear Green's function and that the following conditions hold:*

1. $\sigma_i^{K-1} < (\epsilon/2)\,\|\tilde{f}\|$ *for each subinterval* B_i,
2. $\sigma_i^{K-2} < (\epsilon/2)\,\|\tilde{f}\|$ *for each subinterval* B_i,
3. $\|\overline{f} - \tilde{f}\| < \epsilon\,\|\tilde{f}\|$.

Then

$$\frac{\|\overline{\sigma} - \sigma\|}{\|\sigma\|} \leq \kappa(P)\,(\|P - \overline{P}\| + 1)\,\epsilon.$$

In other words, if $\tilde{f}$ is properly resolved, then a sufficient condition for the error to be of the order $O(\epsilon)$ is that the last two coefficients of all local expansions be of the order $O(\epsilon)$. This has, of course, only been demonstrated rigorously when using a Green function G_0 which is piecewise linear, such as the one constructed from (15) and (16). Corresponding results for other Green's functions, such as those based on (17) and (18), can be obtained but are significantly more involved. In any case, the analysis of this section suggests that an adaptive refinement strategy can be based on the examination of the tails of the local Chebyshev expansions. We have chosen one such strategy based on the last three coefficients.

REFINEMENT ALGORITHM.

• On each subinterval B_i, compute the *monitor function*

$$S_i = |\sigma_i^{K-2}| + |\sigma_i^{K-1} - \sigma_i^{K-3}|. \tag{58}$$

• Compute $S_{div} = \max_{i=1}^{M} S_i/2^C$, where the constant $C > 1$ is provided by the user.

• Subdivide B_i if $S_i \geq S_{div}$.

• If B_i and B_{i+1} are children of the same node and $(S_i + S_{i+1}) < S_{div}/2^K$, then replace them by their parent. This step merges subintervals which are determined to be overresolved.

Remark. We recommend setting $C = 4.0$. This works well for all problems we have investigated. It is possible to tune this parameter to the type of problem being solved. For example, if there are only sharp transition regions, then a larger value of C might be slightly better. If there are dense oscillations, a smaller value of C might be more efficient.

Remark. The reason we have used the expression $S_i = |\sigma_i^{K-2}| + |\sigma_i^{K-1} - \sigma_i^{K-3}|$ as a monitor function rather than $|\sigma_i^{K-2}| + |\sigma_i^{K-1}|$ is somewhat complicated but stems from the fact that the left and right spectral integration matrices of order K are singular. In particular, when K is even, the function $f(x) = T_{K-1}(x) + T_{K-3}(x) + \cdots + T_1(x)$ happens to be a null vector. When the solution is underresolved, spurious values for σ_i^{K-1} can be introduced by projections in this direction. By evaluating the difference between σ_i^{K-1} and σ_i^{K-3}, we ignore this projection. A similar problem arises when K is odd.

When we create a new mesh by subdividing selected subintervals, we define a new discretized integral operator $\overline{P}$. Most of the subintervals remain unchanged, however, so only a few new operators $\overline{P}_i$ need to be inverted locally. This is computationally advantageous since, as we will see below, the local solvers dominate the computation cost.

3.4. Termination condition. Our refinement strategy is applied iteratively until a convergence criterion has been satisfied. There are several options for determining when the computed solution is sufficiently resolved. One can look at the tails of the Chebyshev expansions and apply Theorem 3.4, but such global error bounds are difficult to use since the condition number of the integral operator P and bounds on $\|P - \overline{P}\|$ are not known. A more standard approach is to look at changes in the computed solution $\overline{\sigma}(x)$ or the corresponding approximation to the solution $u(x)$ of the original equation. Letting u_r be the approximation to $u(x)$ at refinement stage r, we choose to stop when the following condition is satisfied:

$$\frac{\|u_r - u_{r-1}\|}{\|u_r + u_{r-1}\|} < \text{TOL},$$

where TOL is the desired accuracy.

Suppose that, with the preceding stopping criterion, there have been R stages of refinement and that the last solution computed is u_R. As a further check, we can then double the final mesh and solve the global integral equation using twice as many points as deemed necessary. We refer to the corresponding solution as u_{DBL}. We can estimate the error in u_R as

$$\|u_R - u\| \approx \|u_R - u_{DBL}\|. \tag{59}$$

In our implementation, we also check, after mesh doubling, that the termination condition given above is still satisfied.

3.5. Description of the algorithm. We now present a more detailed description of the numerical method. The user must provide

1. subroutines which evaluate the functions $p(x), q(x)$, and $f(x)$,
2. the desired order of accuracy of the method (K),
3. the initial mesh, defined by a sequence of distinct points $a = b_1, b_2, \ldots, b_M, b_{M+1} = c$, with $M \geq 1$,
4. the refinement parameter C,
5. the desired tolerance TOL.

ALGORITHM.

Initialization.

Comment [Define initial parameters and create tree data structure]

(1) Create M subintervals $B_i = [b_i, b_{i+1}]$ on $[a, c]$ so that $[a, c] = \cup_{i=1}^{M} B_i$.
(2) Construct a binary tree with M leaves corresponding to subintervals B_i.
(3) Create the *update* list, consisting of all leaf nodes.
(4) Set DblFlag = FALSE.

Remark. The number of nodes in the update list at refinement stage r will be denoted by U_r. The total number of subintervals in the discretization will be denoted by M_r. On initialization, $r = 1$, $M_1 = M$, and $U_1 = M_1$.

Step 1 (Local solver).

Comment [Solve the local integral equations and compute the inner products α, β, and δ on each leaf node in *update* list]

for (B_i **in** *update* **list)**

(1) Determine the locations of the scaled Chebyshev nodes $\tau_1^i, \tau_2^i, \ldots, \tau_K^i$ on B_i using (54).
(2) Evaluate $\tilde{p}(x), \tilde{q}(x)$, and $\tilde{f}(x)$ at the scaled Chebyshev nodes.

(3) Construct the $K \times K$ linear system $\overline{P}_i$ on B_i via equation (56).

(4) Evaluate $\overline{P}_i^{-1}\tilde{f}, \overline{P}_i^{-1}\psi_l, \overline{P}_i^{-1}\psi_r$ on B_i via Gaussian elimination.

(5) Evaluate the coefficients $\alpha^i_{[l,r]}, \beta^i_{[l,r]}$, and $\delta^i_{[l,r]}$ on B_i using Chebyshev integration and the formulas (31), (32), and (33).

(6) Remove B_i from *update* list.

end for

Remark. The amount of work required to factor the local linear systems $\overline{P}_i$ for U_r intervals in the *update* list is of the order $O(U_r \cdot K^3)$. This is the most expensive part of the algorithm. Once the factorizations of all the $\overline{P}_i$ are obtained, it is possible to apply $\overline{P}_i^{-1}$ to ψ_l, ψ_r and $\tilde{f}$ at a total cost of approximately $3U_r \cdot K^2$ operations. The evaluation of α, β, and δ requires only $3U_r \cdot K$ operations.

Step 2.A (Upward sweep).

Comment [Compute $\alpha^k_{[l,r]}, \beta^k_{[l,r]}$, and $\delta^k_{[l,r]}$ for all internal nodes k]

for each internal node k in the binary tree [from the finest level to the coarsest]

(1) Find its two children nodes k_l and k_r.

(2) Evaluate $\alpha^k_{[l,r]}, \beta^k_{[l,r]}$, and $\delta^k_{[l,r]}$ from $\alpha^{k_l,k_r}_{[l,r]}, \beta^{k_l,k_r}_{[l,r]}$, and $\delta^{k_l,k_r}_{[l,r]}$ using formulas (40)–(45).

end for

Step 2.B (Downward sweep).

Comment [Construct the coupling coefficients λ^k_l, λ^k_r for all nodes k]

Set $\lambda^{root}_l = \lambda^{root}_r = 0$ and $\lambda^{root} = 1$.

for each internal node *k* in the binary tree [from the coarsest level to the finest]

(1) Find its two children nodes k_l, k_r.

(2) Evaluate $\lambda^{k_l,k_r}_{[l,r]}$ from $\alpha^{k_l,k_r}_{[l,r]}, \beta^{k_l,k_r}_{[l,r]}$, $\delta^{k_l,k_r}_{[l,r]}$, and $\lambda^k_{[l,r]}$ using Lemma 2.4.

end for

Step 2.C (Evaluation of solution of global integral equation).

Comment [Compute the approximate solution $\overline{\sigma}$ of equation (20) at the nodes $\tau^i_1, \tau^i_2, \ldots, \tau^i_K$ on each B_i via Lemma 2.3]

for (each leaf node B_i)

(1) Determine the values of the solution $\overline{\sigma}$ at the node $\tau^i_j, j = 1, ..., K$ from $P^{-1}_{B_i}\psi_l$, $P^{-1}_{B_i}\psi_r$, $P^{-1}_{B_i}\tilde{f}$, and $\lambda^i_{[l,r]}$ via formula (27).

end for

Remark. Steps (2.A) and (2.B) require an amount of work proportional to the number of subintervals, while (2.C) requires $O(K)$ operations for each subinterval. Thus, the total cost for step 2 is $O(M) + O(MK)$.

Step 3.A (Evaluation of solution).

Comment [Precompute the definite integrals $J^i_l = \int_a^{b_i} g_l \cdot \sigma$ and $J^i_r = \int_{b_{i+1}}^c g_r \cdot \sigma$]

$J^1_l = 0, J^M_r = 0$

do i = 1, M-1

$J^{i+1}_l = J^i_l + \int_{b_i}^{b_{i+1}} g_l(t)\sigma(t)\,dt = J^i_l + \delta^i_l + \lambda^i_l\alpha^i_l + \lambda^i_r\beta^i_l$

end do

do i = M, 2, -1

$J^{i-1}_r = J^i_r + \int_{b_i}^{b_{i+1}} g_r(t)\sigma(t)\,dt = J^i_r + \delta^i_r + \lambda^i_l\alpha^i_r + \lambda^i_r\beta^i_r$

end do

Step 3.B (Evaluation of solution).

Comment [compute $u(x)$ and $u'(x)$ at each node in discretization]

for (each leaf node B_i)
do j=1,K

$$u(\tau_j^i) = u_i(\tau_j^i) + \frac{g_r(\tau_j^i)}{s} \cdot \left[J_l^i + \int_{b_i}^{\tau_j^i} g_l(t) \cdot \sigma(t)\, dt \right] + \frac{g_l(\tau_j^i)}{s} \cdot \left[\int_{\tau_j^i}^{b_{i+1}} g_r(t) \cdot \sigma(t)\, dt + J_r^i \right]$$

$$u'(\tau_j^i) = u_i'(\tau_j^i) + \frac{g_r'(\tau_j^i)}{s} \cdot \left[J_l^i + \int_{b_i}^{\tau_j^i} g_l(t) \cdot \sigma(t)\, dt \right] + \frac{g_l'(\tau_j^i)}{s} \cdot \left[\int_{\tau_j^i}^{b_{i+1}} g_r(t) \cdot \sigma(t)\, dt + J_r^i \right]$$

end do
end for

Remark. Approximately $2M$ operations are required in Step (3.A). The integrals required on each subinterval in Step (3.B) can be computed in $O(K \log K)$ work using the fast cosine transform and spectral integration. Since K is typically less than 20, however, we assume that the integrals are computed directly at a cost of K^2 operations. The total cost for evaluation of the solution is of the order $O(M) + O(MK^2)$.

Step 4 (Mesh refinement).

Comment [Refine mesh until computed solution converges within TOL]

Compute TEST $= \|u_r - u_{r-1}\| / \|u_r + u_{r-1}\|$.
if (TEST $>$ TOL)
Evaluate monitor function S_i on each interval B_i via equation (58).
Compute $S_{div} = \max_{i=1}^{M} S_i / 2^C$.
for (each interval B_i)
if $S_i \geq S_{div}$ then
(1) divide B_i into left and right subintervals
(2) add new intervals to *update* list.
else if (B_i and B_{i+1} are children of the same node) and $(S_i + S_{i+1}) < S_{div}/2^K$ **then**
merge B_i and B_{i+1}
end if
end for
DblFlag = FALSE
else if (DblFlag = FALSE) then
Divide all subintervals into halves and add them to *update* list
DblFlag = TRUE
else
Exit the loop (Convergence test is satisfied after doubling the mesh).
end if
Go to Step (1)

4. Numerical results. The algorithm of section 3 has been implemented in double precision in FORTRAN. To study its performance, we have chosen a representative collection of stiff two-point boundary value problems and have solved them using our algorithm on a SUN SPARCstation IPX. In each case, the order of the method is fixed to be 16 and the initial mesh is just a single interval.

Example 1 (Viscous shock). We first consider the steady advection–diffusion problem

$$\epsilon u''(x) + 2xu'(x) = 0, \tag{60}$$

with boundary conditions

$$u(-1) = -1; \qquad u(1) = 1,$$

whose exact solution is given by

$$u(x) = erf^{-1}\left(\frac{1}{\sqrt{\epsilon}}\right)\ erf\left(\frac{x}{\sqrt{\epsilon}}\right) = erf^{-1}\left(\frac{1}{\sqrt{\epsilon}}\right)\ \frac{2}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{\epsilon}}} e^{-t^2}\, dt.$$

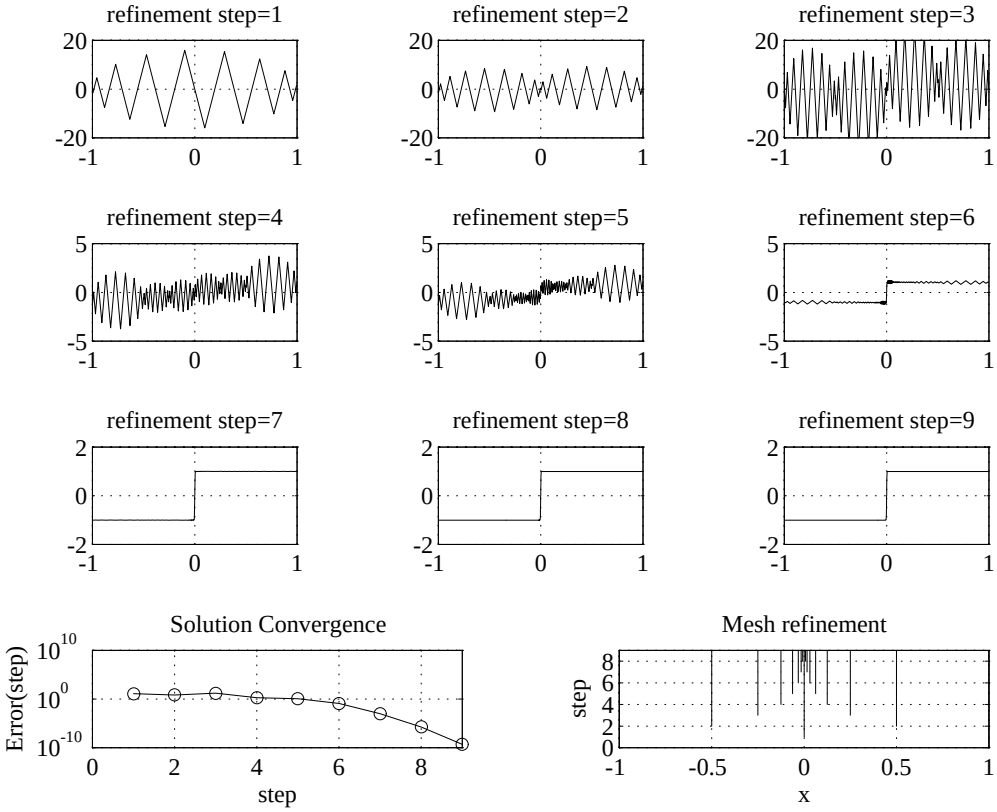


FIG. 1. *A summary of the adaptive solution of the viscous shock problem. The top nine graphs show the solutions at successive refinement steps. The bottom left graph is a plot of the relative L^2 norm of the error as a function of the refinement step. The bottom right graph is a plot of the subinterval boundaries constructed by the adaptive algorithm at each step of the refinement process.*

For $\epsilon = 10^{-5}$, the adaptive algorithm requires nine levels of mesh refinement. For illustration, we plot the computed solution at each step in the refinement process in Figure 1. We also plot the relative error of the computed solution in the L^2 norm and the subinterval boundaries constructed during the execution of the program. Note that refinement takes place only in the vicinity of the internal layer, despite the fact that the solution is underresolved for the first four or five steps.

We have examined the behavior of the algorithm over a wide range of ϵ, from 10^{-4} to 10^{-14}. Figure 2 plots the relative error against the number of subintervals constructed. The dashed line at the end of each curve shows how the error behaves after doubling the final mesh produced by the adaptive code. The number of accurate digits obtained fits the formula $10^{-16}/\sqrt{\epsilon}$ very closely. This is consistent with the fact that the condition number of the problem is of the order $O(1/\sqrt{\epsilon})$. The amount of time required for these cases is summarized in Table 1.

Example 2 (Bessel equation). Our second example is the Bessel equation

$$u''(x) + \frac{1}{x}u'(x) + \frac{x^2 - \nu^2}{x^2}u(x) = 0,$$

with boundary conditions

$$u(0) = 0; \qquad u(600) = 1,$$

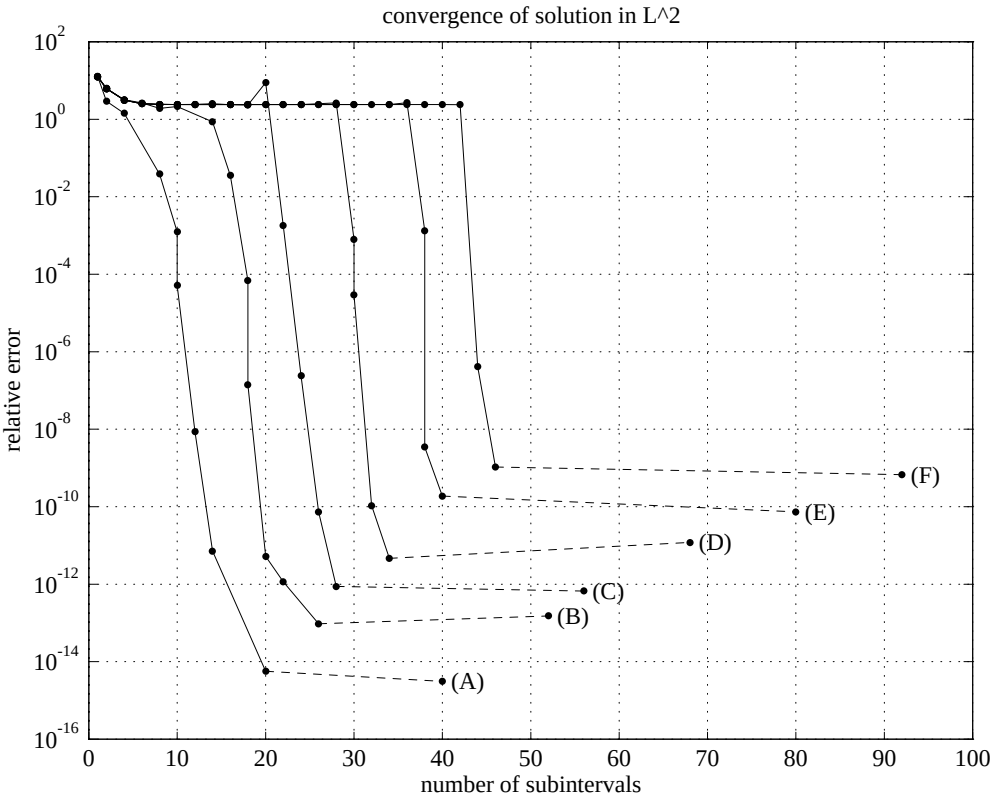


FIG. 2. *A study of the viscous shock problem over a wide range of viscosity ϵ. We plot the relative L^2 norm of the error against the number of subintervals constructed. Curves (A) through (F) correspond to values of ϵ from 10^{-4} to 10^{-14}. In each successive curve, the value of ϵ is reduced by a factor of* 100.

TABLE 1

Performance of the adaptive algorithm on the viscous shock problem (Example 1*) with varying values of the viscosity ϵ. R denotes the number of refinements, M_R denotes the number of subintervals in the final discretization, and times are measured in seconds.*

Legend	(A)	(B)	(C)	(D)	(E)	(F)
ϵ	10^{-4}	10^{-6}	10^{-8}	10^{-10}	10^{-12}	10^{-14}
R	9	13	15	18	21	24
M_R	20	26	28	34	40	46
Error	$5.63 \cdot 10^{-15}$	$9.50 \cdot 10^{-14}$	$8.75 \cdot 10^{-13}$	$4.66 \cdot 10^{-12}$	$1.88 \cdot 10^{-10}$	$1.05 \cdot 10^{-9}$
CPU Time	0.4	0.6	0.7	1.0	1.2	1.4

with $\nu = 100$, for which the exact solution is

$$u(x) = \frac{J_{100}(x)}{J_{100}(600)}.$$

The solution to this equation is smooth on $[0, 100]$, at which point it becomes highly oscillatory. There are approximately 80 full oscillations on $[100, 600]$, and the adaptive algorithm used 106 subintervals (or approximately 20 points per wavelength) to achieve an L^2 error of 4.6×10^{-10}. Although 6 points per wavelength are sufficient

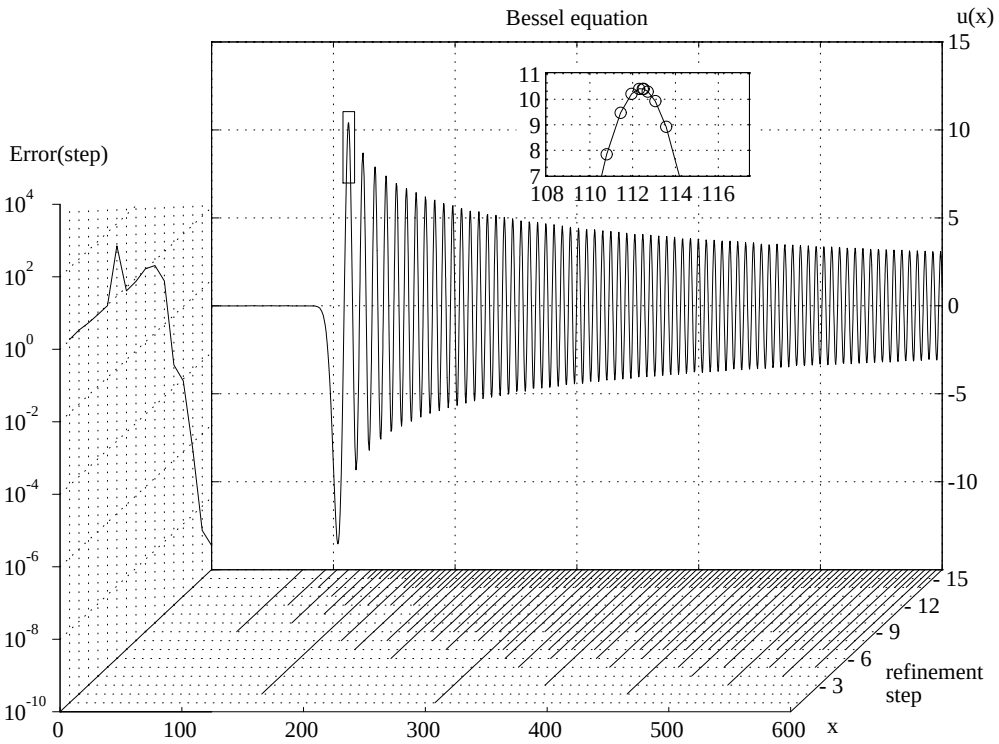


FIG. 3. *The adaptive solution of the Bessel equation (Example 2). In one coordinate plane, we plot the computed solution and a magnification of one small interval. In the magnified window, the circles indicate the location of discretization points. The leftmost coordinate plane plots the relative L^2 norm of the error as a function of the refinement step and the third coordinate plane plots the boundaries of the subintervals created by the adaptive algorithm at each refinement step.*

to resolve the problem, two more doublings are required to achieve full accuracy. A summary of the adaptive calculation is presented in Figure 3.

Example 3 (Turning point). The turning point problem

$$\epsilon u''(x) - xu(x) = 0,$$

with boundary conditions

$$u(-1) = 1; \qquad u(1) = 1,$$

has smooth regions, boundary layers, internal layers, and regions with dense oscillations. The exact solution is a linear combination of Airy functions

$$u(x) = c_1 * Ai\left(\frac{x}{\sqrt[3]{\epsilon}}\right) + c_2 * Bi\left(\frac{x}{\sqrt[3]{\epsilon}}\right).$$

A summary of the adaptive calculation is presented in Figure 4 for $\epsilon = 10^{-6}$.

Example 4 (Potential barrier). A boundary value problem typical of those which arise in quantum mechanics is

$$\epsilon u''(x) + (x^2 - w^2)u(x) = 0,$$

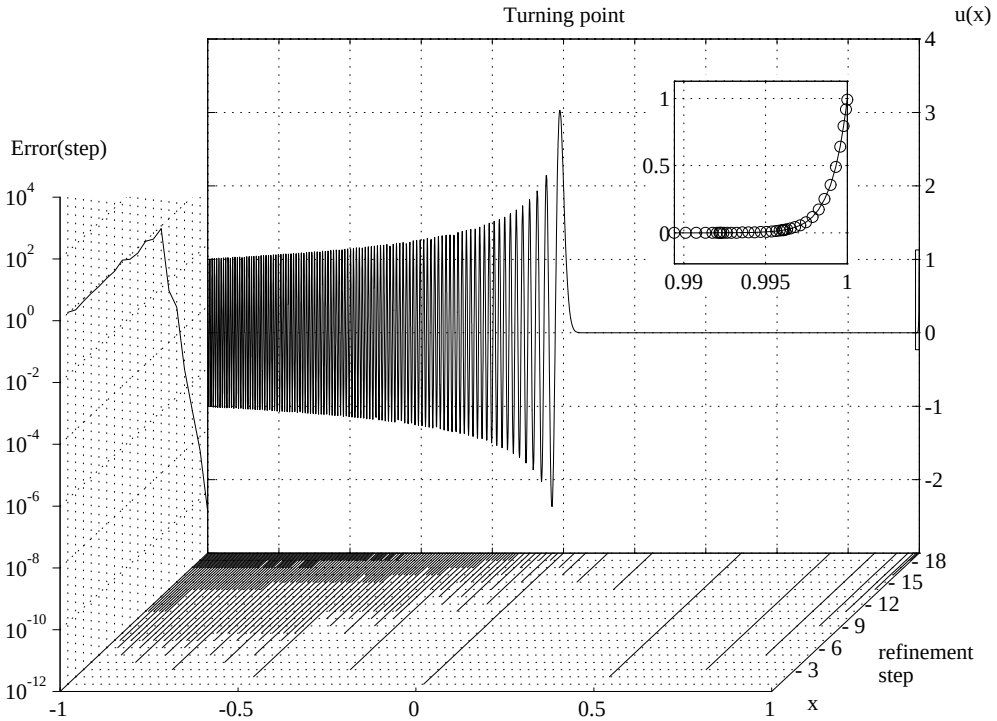


FIG. 4. *The adaptive solution of a turning point problem (Example 3). In this case, the magnified window examines the boundary layer at* $x = 1$.

with boundary conditions

$$u(-1) = 1; \qquad u(1) = 2,$$

with $w = 0.5$. Figure 5 summarizes the adaptive calculation for $\epsilon = 10^{-6}$. To measure accuracy, we have chosen as an "exact" solution the one obtained by doubling the last mesh produced by the algorithm.

Example 5 (Cusp). The problem

$$\epsilon u''(x) + xu'(x) - \frac{1}{2}u(x) = 0,$$

with

$$u(-1) = 1; \qquad u(1) = 2,$$

has a cusplike structure at the origin. The exact solution is

$$u(x) = \frac{3}{2}\frac{M(-\frac{1}{4}, \frac{1}{2}, -\frac{x^2}{2\epsilon})}{M(-\frac{1}{4}, \frac{1}{2}, -\frac{1}{2\epsilon})} + \frac{1}{2}x\frac{M(\frac{1}{4}, \frac{3}{2}, -\frac{x^2}{2\epsilon})}{M(\frac{1}{4}, \frac{3}{2}, -\frac{1}{2\epsilon})},$$

where M is a parabolic cylinder function [1]. Figure 6 summarizes the adaptive calculation for $\epsilon = 10^{-10}$. Since the exact solution is difficult to evaluate directly, we proceed as in Example 4. In other words, we choose as an "exact" solution the one obtained by doubling the last mesh produced by the algorithm.

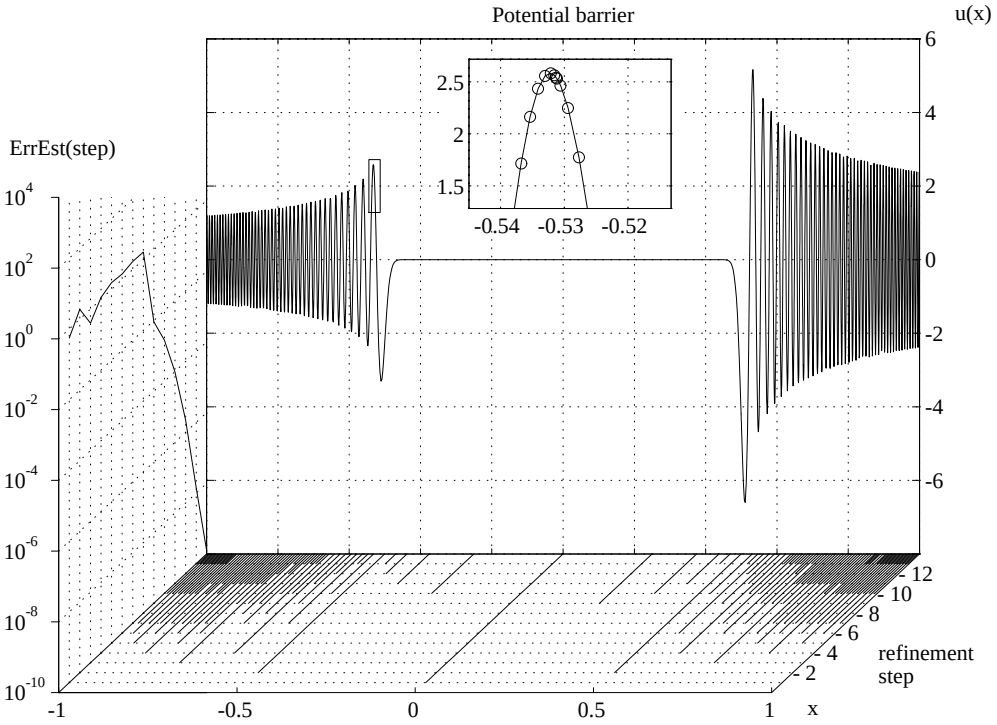


FIG. 5. *The adaptive solution of the potential barrier problem (Example* 4*).*

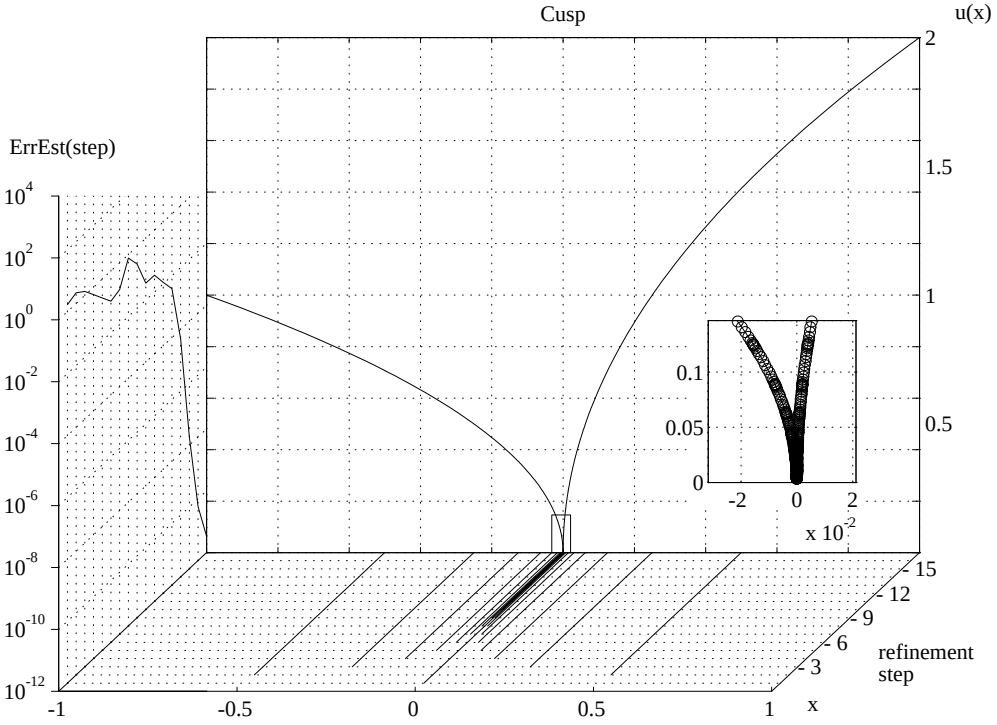


FIG. 6. *The adaptive solution of the cusp problem (Example* 5*).*

Example 6 (Exponential ill conditioning). Our final example is the equation

$$\epsilon u''(x) - xu'(x) + u(x) = 0,$$

with

$$u(-1) = 1; \qquad u(1) = 2,$$

whose exact solution is

$$u(x) = \frac{1}{2}x + \frac{3}{2}\frac{M(-\frac{1}{2}, \frac{1}{2}, \frac{x^2}{2\epsilon})}{M(-\frac{1}{2}, \frac{1}{2}, \frac{1}{2\epsilon})},$$

where the parabolic cylinder function now has the simple series expansion

$$M\left(-\frac{1}{2}, \frac{1}{2}, z\right) = \sum_{n=0}^{\infty} \frac{-1}{2n-1}\frac{z^n}{n!}.$$

This problem is exponentially ill conditioned; there is an eigenvalue of the order $e^{-1/2\epsilon}$. The difficulty is that, although the structure of the two boundary layers at ± 1 is easily obtained, the structure of the linear transition region joining them together is not. This equation is discussed in [18] and [22] from the point of view of asymptotics. The subsequent paper [14] presents an analysis based on eigenvalues and explains the ill-conditioned nature of the problem. Figure 7 summarizes the adaptive calculation for $\epsilon = 1/70$, at which point the condition number is approximately 10^{15}. For the sake of illustration, we have forced the adaptive algorithm to continue the refinement process beyond the obtainable accuracy in double precision (which is approximately one digit).

Summary. The performance of the adaptive algorithm on the preceding examples is presented in Table 2. A more detailed breakdown of the algorithm in two extreme cases is presented in Figure 8. The viscous shock problem (Example 1) has a single complicated feature while the Bessel equation (Example 2) is highly oscillatory. In both cases, it can be seen that the majority of the time is spent in solving local problems. For problems with layers, cusps, etc., but no dense oscillations, the algorithm performs as it does for the viscous shock. In the presence of dense oscillations (even over a relatively small subinterval), the algorithm performs as it does for the Bessel equation.

The following two observations can be made on the basis of the examples in this section.

1. With the exception of the ill-conditioned problem of Example 6, where we asked for more accuracy than could be achieved, the adaptive mesh constructed has never been determined to be unnecessarily refined.

2. The execution time of the adaptive algorithm is approximately twice that of the nonadaptive algorithm, had the resolving grid been known a priori.

5. Eigenvalue problems. We briefly consider the Sturm–Liouville eigenvalue problem

$$Lu + \lambda wu = (pu')' + qu + \lambda wu = 0.$$

Popular methods for such problems include shooting methods based on a Ricatti or Prüfer transformation [2, 6] and linear algebraic techniques based on finite difference

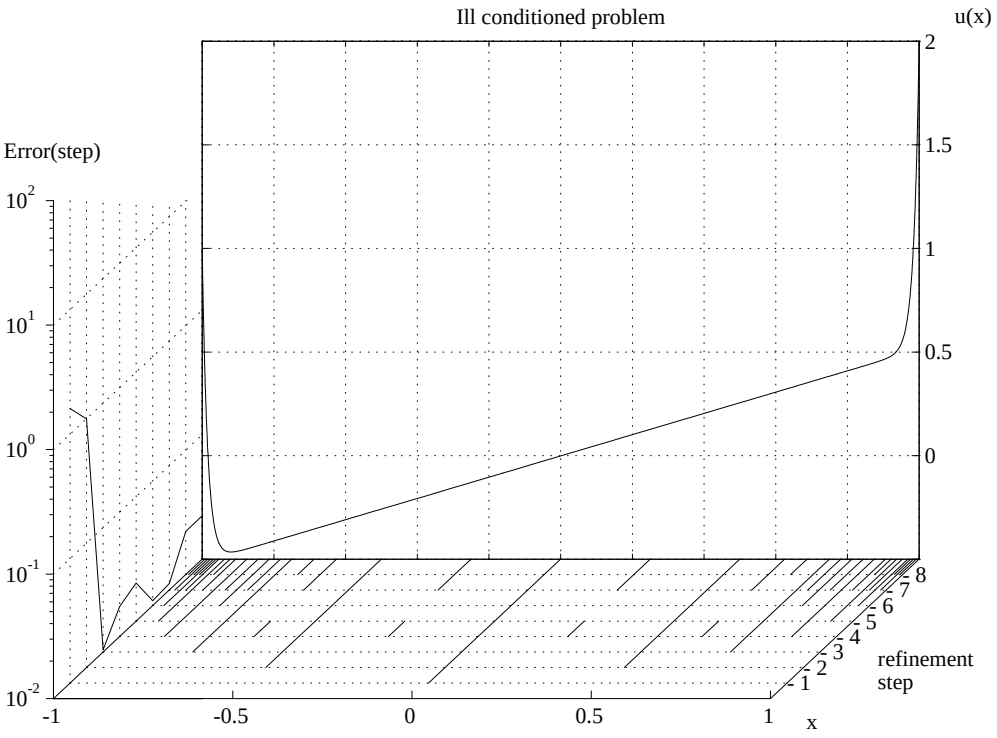


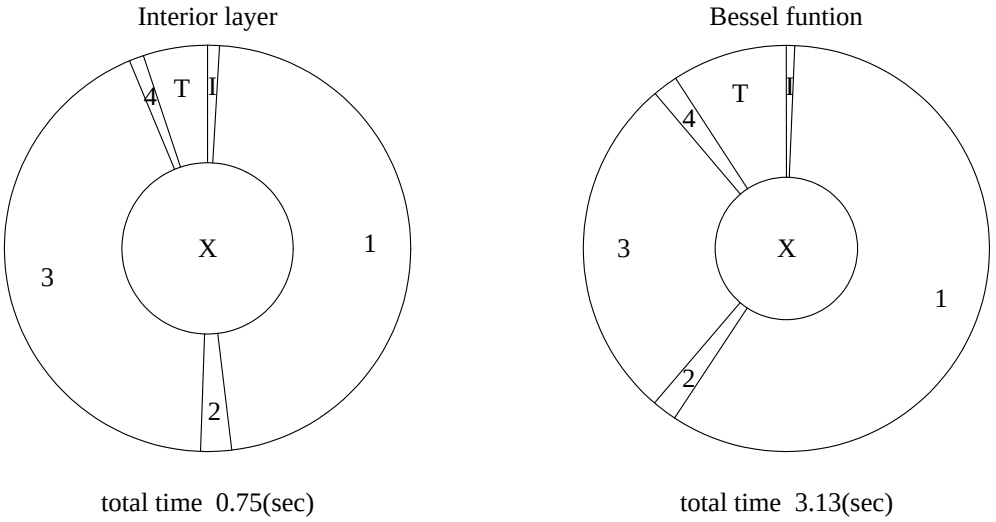
FIG. 7. *The adaptive solution of the problem of Example* 6. *For one digit of accuracy, the algorithm's automatic strategy suggested halting at step* 4. *We then asked for more digits of accuracy, which the code could not provide due to the extreme ill conditioning of the problem. In this case, at refinement step* 5, *the mesh was doubled but no improvement was made. At step* 6, *it was noted that a few subintervals could be merged without any loss of precision.*

TABLE 2

Performance of the adaptive algorithm on Examples 1–6. *The final error refers to the relative error in the* L^2 *norm and* R *denotes the number of refinement steps.* M_R *denotes the number of subintervals in the final discretization,* $M_{tot} = \sum_{r=1}^{R} M_r$, *and* $U_{tot} = \sum_{r=1}^{R} U_r s$. *Total time and doubled mesh time are actually measured in seconds.*

Example	1	2	3	4	5	6
Example Type	Shock	Bessel	Turn. Pt.	Barrier	Cusp	Ill Cond.
Parameter ϵ, ν	$\epsilon = 10^{-8}$	$\nu = 100$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-10}$	$\epsilon = 1/70$
R	15	16	19	14	17	9
Final error	$8.7 \cdot 10^{-13}$	$4.6 \cdot 10^{-10}$	$2.0 \cdot 10^{-11}$	$1.2 \cdot 10^{-10}$	$3.2 \cdot 10^{-12}$	$2.2 \cdot 10^{-2}$
R	15	16	19	14	17	9
M_R	28	106	200	142	32	37
U_{tot}	55	211	399	283	63	85
M_{tot}	211	587	1061	622	273	114
Total Time	0.7	3.1	6.2	2.7	1.1	1.0
Doubled mesh time	1.2	5.8	10.3	4.8	2.0	1.8

or finite element discretizations [13, 23]. We propose a somewhat different method, based on inverse orthogonal iteration with individual shifts, which is quadratically convergent [16]. The computational requirements of the algorithm are proportional to the number of nodes and the number of iterations but grow quadratically with the number of eigenvalues. Further discussion of the method with applications can be found in [26].



time(ms)	called	Legend	time(ms)	called
5.84	1	I: Initialization	18.02	1
290.8	55	1: Local Solver	1610	211
15.18	15	2: Up/down sweep	54.06	8
266.3	211	3: Function Evaluation	756.8	587
7.154	14	4: Mesh Refinement	54.81	15
31.54	14	T: Termination Test	252.3	15
133.2	15	X: Miscellaneous	384.4	16

FIG. 8. *A detailed breakdown of the CPU time requirements of the adaptive algorithm for Examples* 1 *and* 2. *Both the time spent in each step and the number of procedure calls are given. The steps correspond to those described in section* 3.5. *Note that there are U_{tot} calls to the local solver and M_{tot} calls for function evaluation. The row marked "Miscellaneous" includes the time for I/O and the evaluation of the exact solution at every refinement step.*

A brief description of the algorithm follows.

ALGORITHM.

Comment [This algorithm computes the J eigenvalues closest to a user-specified value λ_g. If the smallest eigenvalues are desired, set $\lambda_g = 0$]

Initialization.

Comment [Define initial guess for eigenvalue/eigenvector pairs]

(1) Let $\{q_1, \ldots, q_J\}$ be a random set of vectors.
(2) Construct the matrix $Q^{(0)} = \{q_1^{(0)} | \ldots | q_J^{(0)}\}$ by orthonormalization of the set $\{q_1, \ldots, q_J\}$.
(3) Define $\Lambda_1 = \cdots = \Lambda_J = \lambda_g$.
(4) Set $n = 0$.

Iteration.

do

(1) For $j = 1, \ldots, J$, solve $(L - \Lambda_j I) z_j^{(n+1)} = q_j^{(n)}$.
(2) Construct the matrix $Z^{(n+1)} = \{z_1^{(n+1)} | \ldots | z_J^{(n+1)}\}$.
(3) Compute QR factorization of $Z^{(n+1)}$: $Z^{(n+1)} = Q^{(n+1)} R^{(n+1)}$.

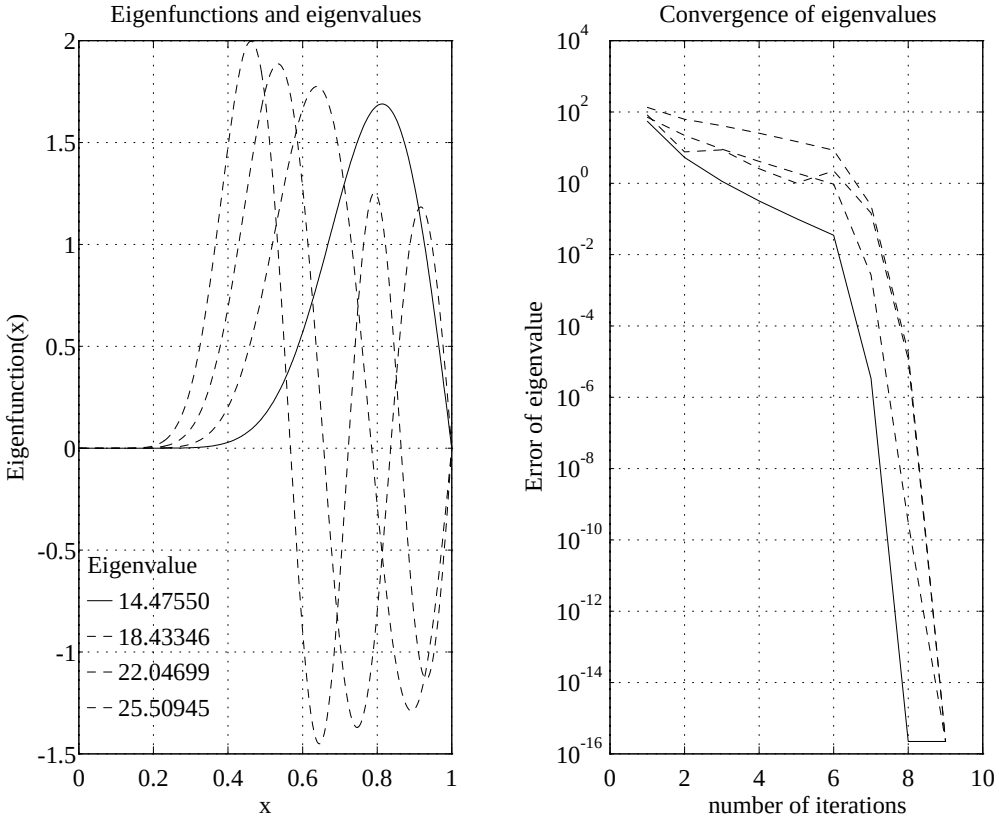


FIG. 9. *The first four eigenvalues and eigenfunctions of the Bessel function $J_{10}(x)$. The errors of the eigenvalues are plotted as a function of the number of iterations on the right.*

(4) For $j = 1, \ldots, J$, set $\lambda_j^{(n+1)} = 1/R_{jj}^{(n+1)}$.

(5) **if** $|\lambda_j^{(n+1)} - \lambda_j^{(n)}| < \frac{MIN\left(\left|\lambda_{j+1}^{(n+1)} - \lambda_j^{(n+1)}\right|, \left|\lambda_j^{(n+1)} - \lambda_{j-1}^{(n+1)}\right|\right)}{4}$ for all j
then $\Lambda_j = \lambda_j^{n+1}$ for all j.

(6) $n = n + 1$.

until all of the sequences λ_j converge.

Note that if $J = 1$, the preceding algorithm is just the inverse power method with shifts. The only unusual feature of the algorithm is step (5), which suggests that new shifts be created only when the indicated criterion is satisfied, rather than at every step. Since our two-point boundary value problem solver is particularly efficient for multiple right-hand sides, we would like to change the differential operator in step (1) as infrequently as possible.

Example 7 (Eigenvalues of the Bessel equation). For illustration, we study the singular Sturm–Liouville problem

$$Lu(x) + \lambda^2 w(x)u(x) = (xu'(x))' - \frac{1}{x}n^2 u + \lambda^2 xu(x) = 0, \tag{61}$$

with boundary conditions $u(0) < \infty$ and $u(1) = 0$ for $n > 0$. The corresponding eigenfunctions and eigenvalues are $u_j(x) = J_n(\lambda_j x)$, where λ_j is jth zero of the Bessel function J_n (Figure 9).

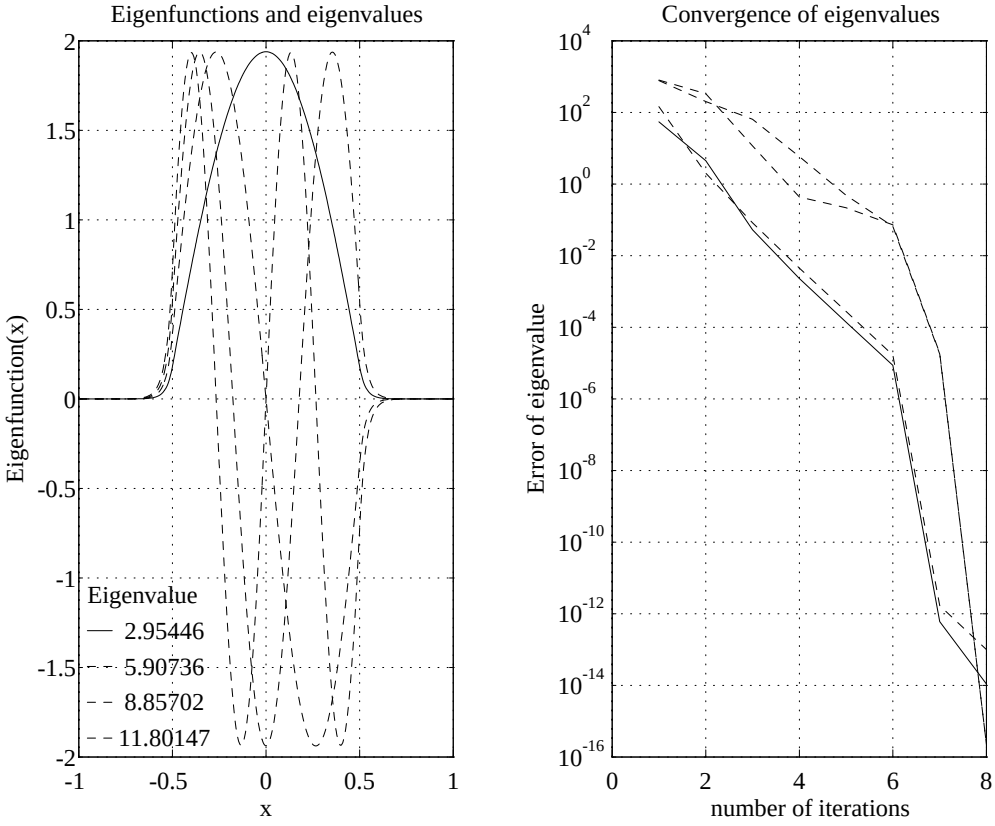


FIG. 10. *The first four eigenvalues and eigenfunctions for the square well potential of Example* 8. *The errors of the eigenvalues are plotted as a function of the number of iterations on the right.*

Example 8 (Square well potential). Typical of the problems which arise in quantum mechanics is the radial Schrödinger equation with a square well potential with discontinuities at $x = -\frac{1}{2}$ and at $x = \frac{1}{2}$. In Figure 10, we plot the first four eigenmodes computed by the algorithm outlined above.

$$Lu(x) + \lambda^2 w(x)u(x) = u''(x) - V(x)u(x) + \lambda^2 u(x) = 0, \tag{62}$$

where

$$V(x) = \begin{cases} 0 & \text{if } |x| \le \frac{1}{2}, \\ 1000 & \text{if } |x| > \frac{1}{2}. \end{cases}$$

6. Conclusions. We have developed a robust, adaptive solver for stiff two-point boundary value problems, with mesh selection based on a sequence of computed solutions. Without a priori information about the location of complicated features, the final mesh constructed is fine in regions which require it and coarse in regions which do not. Perhaps more remarkable, the method requires about twice as much work as a nonadaptive code which is simply given the resolving mesh structure on input.

We have described preliminary applications of the method to eigenvalue problems and are currently extending the scheme to time-dependent problems and first-order

systems. The algorithm of [31] provides a suitable integral equation framework into which our adaptive refinement strategy can be incorporated. While the present algorithm is inherently linear, it can, of course, be used to solve nonlinear problems by coupling it with an outer iteration such as Newton's method.

Acknowledgments. We would like to thank Professors Michael Ward and Vladimir Rokhlin for several useful discussions. In particular, we would like to thank Michael Ward for suggesting that we look at the ill-conditioned problem of Example 6.

REFERENCES

[1] M. Abramowitz and I. Stegun, eds. (1965), *Handbook of Mathematical Functions*, Dover, New York.

[2] U. Ascher, R. M. M. Mattheij, and R. D. Russell (1988), *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ.

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

[4] B. Alpert, G. Beylkin, R. Coifman, and V. Rokhlin (1993), *Wavelet-like bases for the fast solution of second kind integral equations*, SIAM J. Sci. Comput., 14, pp. 159–184.

[5] G. Bader and U. Ascher (1987), *A new basis implementation for a mixed-order boundary value ODE solver*, SIAM J. Sci. Statist. Comput., 8, pp. 483–500.

[6] P. B. Bailey, M. K. Gordon, and L. F. Shampine (1978), *Automatic solution of the Sturm–Liouville problem*, ACM Trans. Math. Software, 4, pp. 193–208.

[7] A. Brandt and A. A. Lubrecht (1990), *Multilevel matrix multiplication and fast solution of integral equations*, J. Comput. Phys., 90, pp. 348–370.

[8] G. Beylkin, R. Coifman, and V. Rokhlin (1991), *Fast wavelet transforms and numerical algorithms* I, Comm. Pure Appl. Math., 44, pp. 141–183.

[9] C. W. Clenshaw and A. R. Curtis (1960), *A method for numerical integration on an automatic computer*, Numer. Math., 2, pp. 197–205.

[10] R. Courant and D. Hilbert (1953), *Methods of Mathematical Physics, Vol.* I, Wiley Interscience, New York.

[11] E. A. Coutsias, T. Hagstrom, J. S. Hesthaven, and D. Torres (1995), *Integration preconditioners for differential operators in spectral τ-methods*, Houston J. Math., to appear.

[12] E. A. Coutsias, T. Hagstrom, and D. Torres (1996), *An efficient spectral method for ordinary differential equations with rational coefficients*, Math. Comp., 65, pp. 611–635.

[13] C. DeBoor and B. Swartz (1980), *Collocation approximation to eigenvalues of an ODE: The principle of the thing*, Math. Comp., 35, pp. 679–694.

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

[15] L. Fox and I. B. Parker (1968), *Chebyshev Polynomials in Numerical Analysis*, Oxford University Press, London.

[16] G. H. Golub and C. F. Van Loan (1983), *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD.

[17] D. Gottlieb and S. Orszag (1977), *Numerical Analysis of Spectral Methods*, SIAM, Philadelphia, PA.

[18] J. Grassman and B. J. Matkowsky (1977), *A variational approach to singularly perturbed boundary value problems for ordinary and partial differential equations with turning points*, SIAM J. Appl Math., 32, pp. 588–597.

[19] L. Greengard (1991), *Spectral integration and two-point boundary value problems*, SIAM J. Numer. Anal., 28, pp. 1071–1080.

[20] L. Greengard and V. Rokhlin (1991), *On the numerical solution of two-point boundary value problems*, Comm. Pure Appl. Math., 44, pp. 419–452.

[21] H. Keller (1968), *Numerical Methods for Two-Point Boundary Value Problems*, Blaisdell, New York, Waltham, MA.

[22] J. Kevorkian and J. D. Cole (1981), *Perturbation Methods in Applied Mathematics*, Springer-Verlag, Berlin, New York.

[23] H.-O. Kreiss (1972), *Difference approximations for boundary and eigenvalue problems for ordinary differential equations*, Math. Comp., 26, pp. 605–624.

[24] B. Kreiss and H.-O. Kreiss (1981), *Numerical methods for singular perturbation problems*, SIAM J. Numer. Anal., 18, pp. 262–276.

[25] H.-O. Kreiss, N. K. Nichols, and D. Brown (1986), *Numerical methods for stiff two-point boundary value problems*, SIAM J. Numer. Anal., 23, pp. 325–368.

[26] J.-Y. Lee and M. Ward (1995), *On the asymptotic and numerical analyses of exponentially ill-conditioned singularly perturbed boundary value problems*, Stud. Appl. Math., 94, pp. 271–326.

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

[28] J. H. Ma (1992), *The Rapid Solution of the Laplace Equation on Regions with Fractal Boundaries*, Tech. report 927, Department of Computer Science, Yale University, New Haven, CT.

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

[30] C. E. Pearson (1968), *On a differential equation of boundary layer type*, J. Math. Phys., 47, pp. 134–154.

[31] P. Starr and V. Rokhlin (1990), *On the Numerical Solution of Two-Point Boundary Value Problems* II, Tech. report 802, Department of Computer Science, Yale University, New Haven, CT.