



**A Spectral Integration Method for Linear
Two-Point Boundary Value Problems**

B. Garrett

SMU Math Report 2003-02

**DEPARTMENT OF MATHEMATICS
SOUTHERN METHODIST UNIVERSITY**

A Spectral Integration Method for Linear Two-Point Boundary Value Problems

Bentley T. Garrett
Department of Mathematics
Southern Methodist University
Dallas, TX 75275, USA

Abstract

In recent years, a robust method based on spectral integration with Chebyshev expansions has been applied to stiff linear two-point boundary value problems. Furthermore, the complexity of the solution process has been reduced by solving local integral equations over subintervals, resulting in $O(Np^2)$ instead of $O(N^3)$ arithmetic operations, where N is the number of nodes, and p is the order of the Chebyshev approximation on each subinterval. An adaptive version has also been devised. In this paper, a spectral analysis is used to demonstrate how the order affects the accuracy. Also the specialized method for the constant coefficient case, the general nonadaptive method, and the general adaptive method are compared against one another in terms of complexity. The complexity of all three methods are then compared to that of COLNEW, a collocation code that is currently one of the best established, robust codes for solving general boundary value problems.

Keywords: Spectral methods, boundary value problems, Chebyshev polynomials

1 Introduction

Given a general linear two-point boundary value problem (BVP) on an interval $[a, c]$,

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

$$\zeta_{l0}u(a) + \zeta_{l1}u'(a) = \Gamma_l \tag{2}$$

$$\zeta_{r0}u(c) + \zeta_{r1}u'(c) = \Gamma_r, \tag{3}$$

where $p(x)$, $q(x)$, and $f(x)$ are continuous, the solution u must be in $C^2[a, c]$. Thus, from approximation theory it is known that u may be written as a Chebyshev series that is uniformly convergent, and whose coefficients are absolutely convergent:

$$u(x) = \frac{a_0}{2}T_0(x) + a_1T_1(x) + a_2T_2(x) + a_3T_3(x) + \dots \tag{4}$$

[2, 9]. Truncating the expansion (4) allows one to apply simple spectral differentiation rules to the coefficients to generate a linear system for the coefficients; however, these systems are often ill-conditioned.

The differential equation can be reformulated as an integral equation via the use of Green's functions (for the general variable coefficient case). After discretizing with the Chebyshev polynomial basis, the resulting linear system is well-conditioned. In the case of constant coefficients p and q in the differential equation, the system is sparse, and can be solved in $O(N)$ operations. With variable coefficients, the system requires $O(N^3/3)$ operations. This difficulty motivates solving smaller dense linear systems on the subintervals of a mesh, which can then be combined into a global solution [6, 8], greatly reducing the number of operations. Here, systems need only be solved locally on p

nodes, so that the cost of solving each system is only $O(p^3/3)$ operations, with p being the order of the expansion. The result is a flop count of $O(Mp^3/3)$, or $O(Np^2/3)$, with M being the number of subintervals and N the total number of nodes on the mesh.

To subdivide the interval only at the rapidly changing areas of the solution, an adaptive code has been written, which uses a monitor function based on the size of the last two coefficients of the Chebyshev expansion on each subinterval [8]. The stopping criterion and error estimate are simply based on the relative change of the last two iterates of the solution. The user is also asked whether the difference between the final solution and that found after halving all subintervals should be used as a second check.

COLNEW, by Ascher and Bader, is a currently popular code that is also used to solve such problems. In general, it can solve an arbitrarily large system of differential equations, with each equation limited to order 4. It uses a spline collocation method with polynomial interpolants at Gaussian points on each subinterval. The number of Gaussian points, k , is limited to $2 \leq k \leq 7$. The bases used are the Lagrangian monomials, as opposed to Chebyshev polynomials. An outline of the procedure is given in [1].

To improve the efficiency of COLNEW's linear system solution, local elimination is performed to remove the values of the solution at the collocation points, so that the solution u_{i+1} at node $i+1$ is expressed only in terms of u_i and u_{i+2} ; the result is an almost block-diagonal (ABD) system. The global system for the entire interval is then solved. For the specific case of linear second-order problems, the first derivative of u at each node is forced to be continuous: the polynomials representing the solution must be equal at the endpoints of the intervals, and these polynomials are two-component vectors representing $[u, u']^T$. In general, COLNEW can take the quasi-linearization of a nonlinear problem and solve the resulting linear system by local elimination followed by an ABD solution.

In the first section of this paper, a complete derivation of the specialized algorithm for the constant coefficient case is supplied. This will aid in explaining the calculation of Chebyshev expansions, the conversion to integral equations, and the discretization for all algorithms discussed. Along the way, implementation issues are addressed. An analysis of the solution error and its relationship to the Chebyshev coefficients are given on four problems with the use of MATLAB code, which we have written. Also, a comparison of the number of function evaluations between this algorithm and COLNEW is given.

In the second section, the generalization of the integral forms to variable coefficient BVP's, using Green's functions, is explained. Next, a sketch of the method for combining local problems on subintervals into a global solution is discussed. Again, in terms of function evaluations, the complexity of the resulting algorithm is compared to that of the specialized constant coefficient algorithm on the same four problems. Complexity is also compared with COLNEW on various problems with boundary layers, internal layers, dense oscillations, and combinations of these phenomena.

In the last section, the strategy behind the adaptive scheme is given. Complexity comparisons are made with the constant coefficient algorithm, the subdivision algorithm with no adaptive strategy, and COLNEW.

The Chebyshev collocation code for the variable coefficient and adaptive cases has been written by J. Lee [7].

2 The constant coefficient case

2.1 Integral equation discretization

The following derivation is extracted from [5]. Given a problem of the form

$$\begin{aligned} u'' + \mu u' + \nu u &= f(x), \\ u(-1) = \alpha, u(1) &= \beta, \end{aligned} \tag{5}$$

if we set $u'' = \sigma$ and integrate twice, we obtain:

$$\sigma(x) + \mu \int_{-1}^x \sigma(t)dt + \mu C_1 + \nu \int_{-1}^x \int_{-1}^t \sigma(\tau)d\tau dt + \nu C_1 x + \nu C_0 = f(x), \quad (6)$$

where C_0 and C_1 are constants of integration.

Since the coefficients in the Chebyshev expansion of a function in $C^2[-1,1]$ are absolutely summable, it is reasonable to approximate the function with a truncated series. Letting $T_n(x)$ represent the n^{th} -order Chebyshev polynomial on $[-1,1]$, σ and f can be approximated

$$\sigma(x) = \sum_{k=0}^N a_k T_k(x) = \frac{a_0}{2} T_0(x) + a_1 T_1(x) + \dots + a_N T_N(x) \quad (7)$$

and

$$f(x) = \sum_{k=0}^N f_k T_k(x) = \frac{f_0}{2} T_0(x) + f_1 T_1(x) + \dots + f_N T_N(x). \quad (8)$$

It is well known that performing indefinite integrals, in the form given above, on Chebyshev expansions is equivalent to a matrix multiplication of $O(N)$ flops (by the spectral integration matrix). The integral of σ , for instance, is calculated by:

$$\int_{-1}^x \sigma(t)dt = \frac{d_0}{2} T_0(x) + d_1 T_1(x) + \dots + d_{N+1} T_{N+1}(x), \quad (9)$$

where

$$\begin{aligned} d_k &= \frac{1}{2k} (a_{k-1} - a_{k+1}), \quad 1 \leq k \leq N+1 \\ d_0 &= d_1 - d_2 + d_3 - d_4 + \dots \end{aligned} \quad (10)$$

Since the a_k are decaying, then the d_k must be decaying as well, resulting in a stable calculation. If spectral differentiation was used on the original equation, the resulting coefficients would be seen to grow in size. This is precisely the motivation for the integral reformulation.

Referring to equation (6), the first derivative is

$$u' = \int_{-1}^x \sigma(t)dt + C_1. \quad (11)$$

Performing spectral integration in (11) results in

$$u' = C_1 T_0 + \sum_{k=1}^{N+1} \frac{1}{2k} (a_{k-1} - a_{k+1}) T_k. \quad (12)$$

Here, since T_0 is unity, we have incorporated the resulting constant of integration into the arbitrary constant of integration C_1 .

In turn, u is represented as

$$u = \int_{-1}^x u'(t)dt + C_0, \quad (13)$$

with a new arbitrary constant of integration. By applying the integration matrix to (12) and incorporating the resulting constant (T_0) term into C_0 , we obtain:

$$\begin{aligned} u &= C_0 T_0 + (C_1 - \frac{1}{8}(a_1 - a_3)) T_1 \\ &\quad + \sum_{k=2}^{N+2} \frac{1}{2k} \left[\frac{1}{2(k-1)} (a_{k-2} - a_k) - \frac{1}{2(k+1)} (a_k - a_{k+2}) \right] T_k. \end{aligned} \quad (14)$$

Substituting (12) and (14) into the integral equation (6) yields

$$\begin{aligned} \sum_{k=0}^N a_k T_k + \mu \left[C_1 T_0 + \sum_{k=1}^{N+1} \frac{1}{2k} (a_{k-1} - a_{k+1}) T_k \right] + \nu C_0 T_0 \\ + \nu (C_1 - \frac{1}{8}(a_1 - a_2)) T_1 \\ + \nu \sum_{k=2}^{N+2} \frac{1}{2k} \left[\frac{1}{2(k-1)} (a_{k-2} - a_k) - \frac{1}{2(k+1)} (a_k - a_{k+2}) \right] T_k = \sum_{k=0}^N f_k T_k \end{aligned} \quad (15)$$

Since the Chebyshev polynomials are a linearly independent set, we can equate the coefficients of the left- and right-hand sides of (15) to generate $N + 1$ equations:

$$\frac{1}{2}a_0 + \mu(C_1) + \nu(C_0) = \frac{1}{2}f_0 \quad (16)$$

$$a_1 + \frac{1}{2}\mu(a_0 - a_2) + \nu(C_1 - \frac{1}{8}(a_1 - a_3)) = f_1 \quad (17)$$

$$a_k + \mu\frac{1}{2k}(a_{k-1} - a_{k+1}) + \nu\frac{1}{2k}\left[\frac{1}{2(k-1)}(a_{k-2} - a_k) - \frac{1}{2(k+1)}(a_k - a_{k+2})\right] = f_k, \quad (18)$$

for $k = 2 \dots N$.

The equations corresponding to the terms T_{N+1} and T_{N+2} are ignored in this approximation since the coefficients involved are assumed small. We also note that a few of the terms in (16) and (17) are slightly modified versions of those in [5].

The two remaining unknowns, C_0 and C_1 , are determined by the boundary conditions, yielding $N + 3$ equations for $N + 3$ unknowns. Observe that $T_k(-1) = (-1)^k$ and that $T_k(1) = 1$ for all k . Substituting into (14) yields:

$$u(1) = C_0 + C_1 - \frac{1}{8}(a_1 - a_3) + \sum_{k=2}^{N+2} \frac{1}{2k} \left[\frac{1}{2(k-1)}(a_{k-2} - a_k) - \frac{1}{2(k+1)}(a_k - a_{k+2}) \right] = \beta \quad (19)$$

$$u(-1) = C_0 - C_1 + \frac{1}{8}(a_1 - a_3) + \sum_{k=2}^{N+2} \frac{1}{2k} \left[\frac{1}{2(k-1)}(a_{k-2} - a_k) - \frac{1}{2(k+1)}(a_k - a_{k+2}) \right] (-1)^k = \alpha. \quad (20)$$

This linear system can now be solved to find the expansion for u in (14).

We mention, again, that the T_1 terms are different from those in [5] in order to keep our own derivation consistent. Also, terms for $k = N + 1$ and $k = N + 2$ have been retained in (19)-(20), whereas they have been excluded in [5] as a further approximation.

2.2 Implementation

To solve the system above, the coefficients f_k in the expansion for f

$$f(x) = \frac{f_0}{2}T_0 + f_1T_1 + f_2T_2 \dots \quad (21)$$

are needed. Recalling that

$$T_k(\cos(\theta)) = \cos(k\theta), \quad k = 0, 1, 2, \dots, \quad (22)$$

transforming by $x = \cos(\theta)$ for $0 \leq \theta \leq \pi$ yields

$$f(\cos \theta) = \frac{1}{2}f_0 + f_1 \cos \theta + f_2 \cos(2\theta) + \dots + f_{N-1} \cos((N-1)\theta), \quad (23)$$

if truncation is after term $N - 1$. The orthogonality of the functions $\cos(n\theta)$ for all n on this domain leads to:

$$f_n = \frac{2}{\pi} \int_0^\pi f(\cos \theta) \cos(n\theta) d\theta, \quad 0 \leq n \leq N - 1 \quad (24)$$

The coefficients are simply the cosine transforms of $f(\cos \theta)$.

To compute, we can approximate via the midpoint rule, which employs the points

$$\theta_i = \frac{(2i-1)\pi}{2N}, \quad i = 0, 1, 2, \dots, N-1 \quad (25)$$

to give

$$f_n \approx \frac{2}{N} \sum_{i=0}^{N-1} f\left(\cos\left(\frac{(2i+1)\pi}{2N}\right)\right) \cos\left(n\left(\frac{(2i+1)\pi}{2N}\right)\right). \quad (26)$$

(If $f \in C^k[-1, 1]$, then the error in these approximate coefficients is $O(1/N^k)$.) In other words, when f is evaluated at the zeros of the N^{th} Chebyshev polynomial, a discrete cosine transform can be performed. Fortunately, this can efficiently be implemented with an FFT subroutine, in $O(N \log N)$ operations. In MATLAB, the quadrature was performed by first evaluating f at the Chebyshev points, then expanding the array to one of size $2N$, with the values of f placed on the even numbered positions and zeros elsewhere. The resulting sequence was then padded with additional zeros to make the array size $4N$. (This preliminary step set up the sequence so that it may be used by any “black box” FFT subroutine.) Calling this expanded sequence ff , the sequence of coefficients f_n were then calculated by the MATLAB call

$$f = \frac{2}{N} \mathbf{real}(\mathbf{fft}(ff)), \quad (27)$$

where f , in this case, is the vector resulting from the cosine transform.

The linear system (16)-(20) has a pentadiagonal structure, with the exception of four rows. Thus, we should hope to achieve a solution in $O(N)$ flops. Even with row exchanges in Gaussian elimination, the wider band width is still limited to 7, so that the subsequent back-substitution involves only $O(N)$ calculations for large N . This elimination also was implemented in MATLAB, using the black-box `sparse()` function on the matrix.

The final step is to convert the coefficients of u back into values of u . Since the solution has $N + 2$ terms in this case,

$$u(\cos\theta) = \frac{1}{2}u_0 + u_1 \cos\theta + \dots + u_{N+1} \cos((N+1)\theta), \quad (28)$$

using the definition of the Chebyshev polynomials. Evaluating at the zeros of T_{N+2} yields

$$u\left(\cos\left(\frac{(2i+1)\pi}{2(N+2)}\right)\right) = \sum_{k=0}^{N+1} u_k \cos\left(k\left(\frac{(2i+1)\pi}{2(N+2)}\right)\right). \quad (29)$$

Thus, the inverse process is another discrete cosine transform, which was performed in MATLAB in a way similar to the forward transform.

2.3 Numerical examples

Four problems have been run in our MATLAB implementation to measure the relative L_2 error for varying values of order N . The solution to the first problem,

$$\begin{aligned} -u'' + 400u &= -400 \cos^2(\pi x) - 2\pi^2 \cos(2\pi x), \\ u(0) = u(1) &= 0, \end{aligned} \quad (30)$$

is oscillatory, with large derivatives near the endpoints. In the second problem,

$$\begin{aligned} \varepsilon u'' - u &= 0, \\ u(-1) = 1, u(1) &= 2, \varepsilon = 10^{-5}, \end{aligned} \quad (31)$$

sharp boundary layers occur at the endpoints. The other two examples are highly oscillatory:

$$\begin{aligned} u'' + 5u' + 10,000u &= -500 \cos(100x)e^{-5x}, \\ u(0) = 0, u(1) &= \sin(100)e^{-5} \end{aligned} \quad (32)$$

and

$$\begin{aligned} u'' + (k^2 + 5)u &= 5 \sin(kx), \\ u(-1) = \sin(-k), u(1) &= \sin(k), k = 630. \end{aligned} \quad (33)$$

Transformations of the domain to $[-1, 1]$ have been implemented as they are necessary for our code. The behavior of the first three solutions are displayed in Figures 1,2, and 3, along with the exact relative L_2 errors as a function of the order N and the Chebyshev coefficients. Due to the high frequency of the solution in the last problem, only the error and coefficients are given in Figure 4. To clarify, note that the coefficients are those obtained from the last (highest) order used for each problem, as indicated in the graphs for the L_2 error.

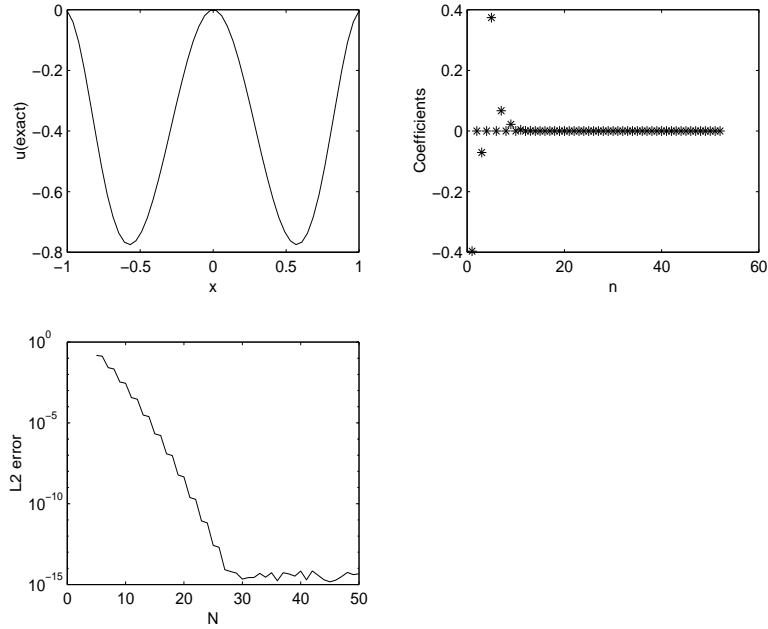


Figure 1: Problem (30)

For problem (30), the coefficients die off quickly, due to the slow oscillation in the solution. The forcing terms here indicate that a low order expansion will deliver reasonable accuracy. The number of cycles of u when x increases by 2π is roughly 6. Graphically, on our scale, the coefficients appear to vanish at around $n = 12$; the Chebyshev series is not quite the same as a Fourier cosine series, in which case we would expect full resolution at $n = 6$. Machine accuracy, however, is not achieved until the order of the Chebyshev expansion is about $N = 30$. Because the coefficients die off so quickly, the first coefficient truncated can be used as an error estimate for the solution.

In problem (31), there is slower decay of the coefficients and the error, as higher order terms are needed to produce large derivatives. Unlike (30), the first truncated term does not estimate the error well. By initial inspection of this problem, it is difficult to know which order is appropriate.

An interesting aspect of (32) is the alternating nature of the coefficients. In like manner, the error oscillates, implying that adding another term to the expansion can actually worsen the error. Also, the frequency of 100 on the right-hand-side leads one to believe that at least 100 nodes are necessary; however, the error reaches the “floor” of 10^{-12} after about 80 terms. For the final example, the right-hand-side implies a highest frequency component of 630: the error and coefficients decay accordingly.

It is obviously difficult to ascertain an appropriate order *a priori*, to mimic the solution behavior. Also, even though the size of the last coefficient may indicate the error in some cases, it is not always an adequate measure. We emphasize that our results are qualitatively similar to those in [5] even though the terms in our expansion have been modified.

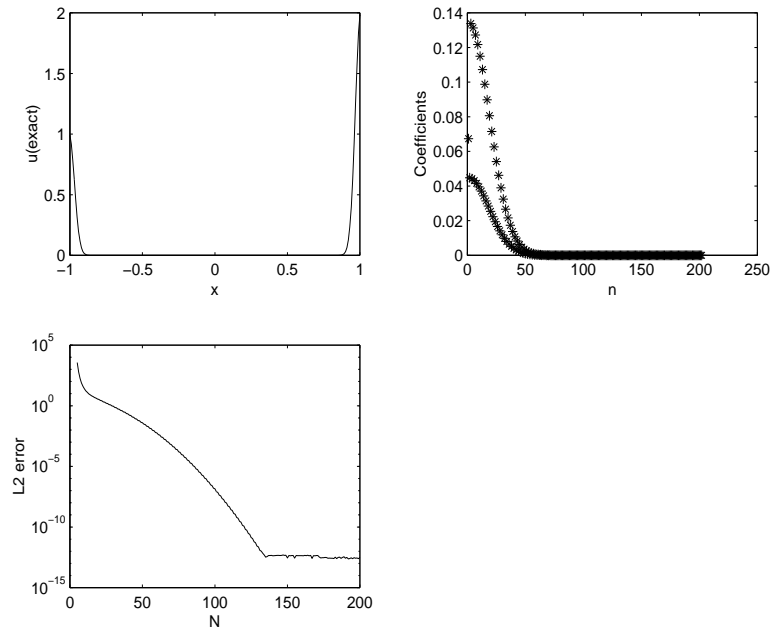


Figure 2: Problem (31)

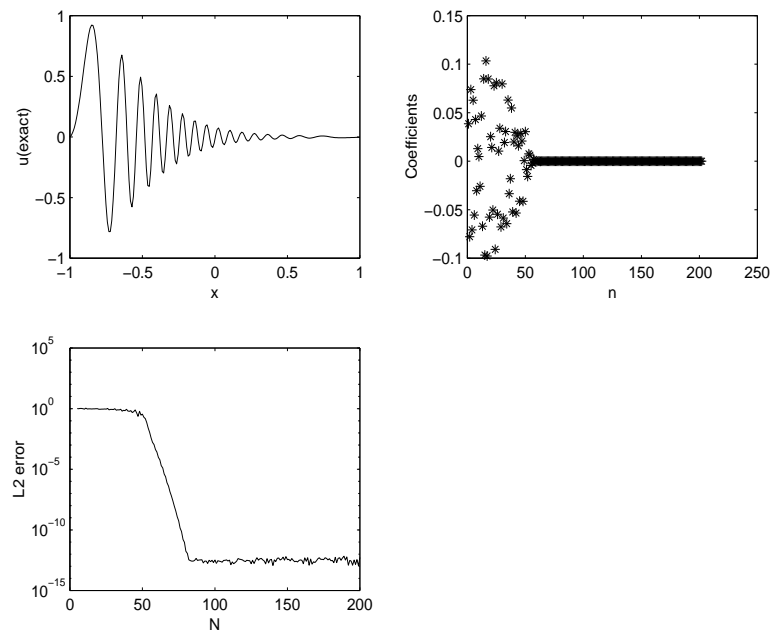


Figure 3: Problem (32)

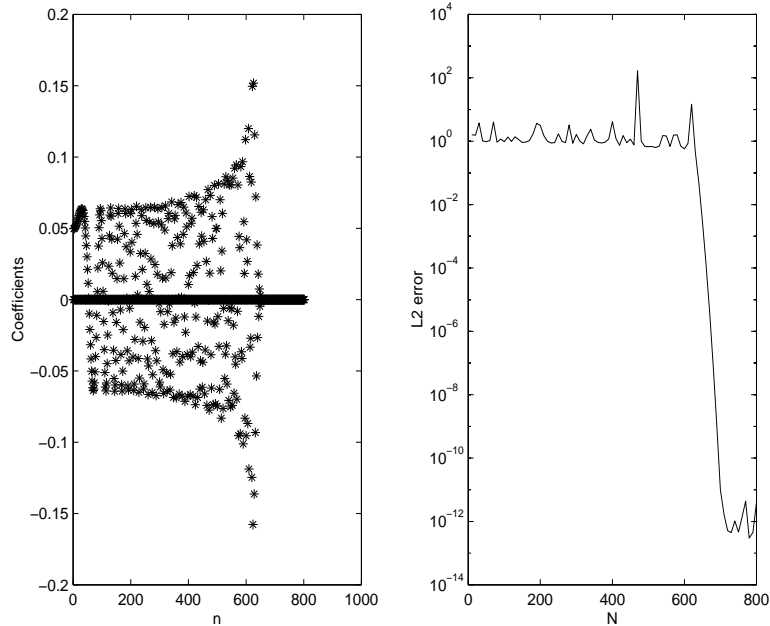


Figure 4: Problem (33)

2.4 Comparison with COLNEW

All four of these problems were run on COLNEW (in linear mode) for tolerances ranging from 10^{-1} to 10^{-15} for orders 3 and 7; an initial mesh of 5 equal intervals was used. The number of function evaluations was recorded for each tolerance. These results were compared to the number of evaluations needed for our constant coefficient code (CC). (In this case, the number of function evaluations is equal to the order.) Tables 1-2 show the results for (30) and (31), respectively. The tolerance was increased for COLNEW until the necessary number of subdivisions exceeded the limit imposed by the code; the order was increased for code CC until machine precision was attained.

TOL	COLNEW		CC	L_2 ERROR
	(3)	(7)		
10^{-1}	45	105	5	.1492
10^{-3}	117	105	10	.0028
10^{-5}	495	105	15	2.14×10^{-6}
10^{-7}	975	245	20	4.54×10^{-9}
10^{-9}	3855	525	25	2.68×10^{-13}
10^{-11}	15375	525	30	2.28×10^{-15}
10^{-13}	61455	1085	35	7.30×10^{-15}
10^{-15}		2205		

Table 1: COLNEW vs. CC, Problem (30)

Note that the higher order requires less work in COLNEW for our particular measure. These problems are representative of all four problems in that the specialized code is much more efficient. This evidence suggests that, at least for problems that are mainly oscillatory in nature, or that have boundary layers, using a higher order basis is much more efficient for either method.

For (31), the boundary layer problem, the choice of midpoint rule quadrature is particularly well-suited since this gives rise to nodes on the Chebyshev zeros, which are clustered near the endpoints.

TOL	COLNEW		CC	L_2 ERROR
	(3)	(7)		
10^{-1}	387	105	5	3.66×10^3
10^{-3}	1191	903	10	24.9
10^{-5}	1308	1505	20	3.31
10^{-7}	5841	2296	30	.913
10^{-9}	10590	1925	40	.215
10^{-11}	44805	2457	50	.0403
10^{-13}	128739	6545	60	.0058
10^{-15}		10745	70	6.28×10^{-4}
			80	5.09×10^{-5}
			100	1.39×10^{-7}
			120	1.18×10^{-10}
			130	2.35×10^{-12}
			150	7.59×10^{-13}
			160	7.46×10^{-13}
			200	6.00×10^{-13}

Table 2: COLNEW vs. CC, Problem (31)

3 The variable coefficient case

3.1 General integral formulation

For the general BVP given in (5), it is well-known that the solution u can be decomposed into $u = u_h + u_i$, where u_i is a linear function satisfying the given inhomogeneous BC's [8]. Since u_i can be found analytically, the problem is then reduced to finding u_h , which satisfies

$$u_h(x)'' + p(x)u_h'(x) + q(x)u_h(x) = f(x) - (u_i''(x) + p(x)u_i'(x) + q(x)u_i(x)) = \tilde{f}(x), \quad (34)$$

with homogeneous boundary conditions corresponding to the original boundary conditions in (5), i.e., $\Gamma_l = \Gamma_r = 0$. This solution can be expressed in terms of a Green's function:

$$u_h(x) = \int_a^c G_0(x, t)\sigma(t)dt, \quad (35)$$

$$G_0(x, t) = \begin{cases} \frac{g_l(x)g_r(t)}{s(t)}, & x \leq t \\ \frac{g_l(t)g_r(x)}{s(t)}, & x \geq t \end{cases}$$

$$s(t) = g_l(t)g_r'(t) - g_l'(t)g_r(t),$$

with g_r and g_l solving

$$g''(x) + q_0(x)g(x) = 0, q_0 \in C^1[a, c], \quad (36)$$

and where g_l satisfies the left homogeneous condition above, and g_r satisfies the right. Any q_0 can be chosen, as long as the only solution of (36) that satisfies both homogeneous BC's simultaneously is the trivial one.

Thus far, given a simple q_0 , G_0 can be determined analytically. As we show below, the problem reduces to solving for $\sigma(x)$ numerically. To force u_h to solve the original problem, its representation in (35) is substituted into (5), resulting in

$$\begin{aligned} \sigma(x) + \tilde{p}(x) \int_a^c \frac{d}{dx} G_0(x, t)\sigma(t)dt + \tilde{q}(x) \int_a^c G_0(x, t)\sigma(t)dt &= \tilde{f}(x) \\ \tilde{p}(x) &= p(x) \\ \tilde{q}(x) &= q(x) - q_0(x). \end{aligned} \quad (37)$$

At this point, Chebyshev discretization and spectral integration can be used to solve for σ , similarly to what we have described above. The result is a linear system as we have seen in the constant coefficient case. Next, with σ known at Chebyshev points, spectral integration can be applied to compute u_h . Moreover, u' is easily calculated since $u' = u'_h + u'_i$ so that

$$u'_h(x) = \int_a^c \frac{d}{dx} G_0(x, t) \sigma(t) dt. \quad (38)$$

In the code designed by Lee [7], q_0 is chosen to be either 0 or -1, depending on the boundary conditions; the Green's function is either linear or a combination of $\cosh(x)$ and $\sinh(x)$. A relatively simple G_0 is desired since, as Greengard and Rokhlin claim, the choice of q_0 has little impact on the efficiency and stability of the solution [6].

3.2 Interval subdivision

In contrast to the constant coefficient case, directly applying the spectral method to the integral equation for $\sigma(x)$ gives rise to a dense linear system, i.e., with a cost of $O(N^3/3)$ operations with Gaussian elimination instead of $O(N)$. Instead, the idea of Greengard and Lee is to solve smaller dense systems on subintervals, using a lower order expansion in each subinterval than would be required on the entire interval. These local solutions can then be combined efficiently to yield the solution on the entire interval in $O(Np^2)$ operations.

The global integral equation 37 can be expressed in terms of g_r and g_l :

$$\begin{aligned} P\sigma(x) &= \sigma(x) + \psi_l(x) \int_a^x \frac{g_l(t)\sigma(t)}{s(t)} dt + \psi_r(x) \int_x^c \frac{g_r(t)\sigma(t)}{s(t)} dt = \tilde{f}(x) \\ \psi_l(x) &= (\tilde{p}(x)g'_r(x) + \tilde{q}(x)g_r(x))/s \\ \psi_r(x) &= (\tilde{p}(x)g'_l(x) + \tilde{q}(x)g_l(x))/s \end{aligned}, \quad (39)$$

where we have factored out $1/s(t) = 1/s$ since $s(t)$ is constant when $q_0 = 0$ or $q_0 = -1$. In [6, 8], it has been shown that the same operator P restricted to a subinterval B ($P_B : L_2(B) \rightarrow L_2(B)$) can be related to P by:

$$P\sigma = P_B\sigma_B - \psi_l\lambda_l^B - \psi_r\lambda_r^B = \tilde{f}(x), \quad x \in B \quad (40)$$

with certain constants λ_l^B and λ_r^B dependent on the interval B . Thus, P_B can be used to solve for $\sigma(x)$ restricted to B (σ_B):

$$P_B\sigma_B(x) = \tilde{f}(x) + \lambda_l^B\psi_l(x) + \lambda_r^B\psi_r(x), \quad x \in B \quad (41)$$

In short, given the constants λ_l^B and λ_r^B , σ_B on B can be calculated the same way locally as σ is on $[a, c]$, but with the inverse integral operator taking on two additional terms. These constants can be computed efficiently for each subinterval.

In order to understand the calculation of the λ_l and λ_r of each subinterval, the chosen (*a priori*) subintervals must be considered part of a tree data structure. For simplicity, if the number of subintervals is assumed to be a power of 2, then these intervals can be considered the leaf (bottom) "nodes" of a binary tree. Each pair of intervals has one parent, i.e., the next-to-last layer of the tree is made up of intervals which are twice the length of the leaf intervals; each parent interval includes its two children. This pattern continues up the tree to the last parent, which is the entire interval $[a, c]$.

The calculation of the λ_l and λ_r for each leaf subinterval proceeds as follows. First, $P_B^{-1}\tilde{f}$, $P_B^{-1}\psi_l$, and $P_B^{-1}\psi_r$ can be solved on each subinterval via Chebyshev discretization and linear system solves. Inner products on each subinterval between each of these solutions, and both g_l and g_r , yield 6 more quantities, α_l , α_r , β_l , β_r , δ_l , and δ_r :

$$\begin{aligned} \alpha_l^B &\equiv \int_B g_l(t) P_B^{-1} \psi_l(t) dt, & \alpha_r^B &\equiv \int_B g_r(t) P_B^{-1} \psi_l(t) dt \\ \beta_l^B &\equiv \int_B g_l(t) P_B^{-1} \psi_r(t) dt, & \beta_r^B &\equiv \int_B g_r(t) P_B^{-1} \psi_r(t) dt \\ \delta_l^B &\equiv \int_B g_l(t) P_B^{-1} \tilde{f}(t) dt, & \delta_r^B &\equiv \int_B g_r(t) P_B^{-1} \tilde{f}(t) dt \end{aligned} \quad (42)$$

Again, these inner products are performed via spectral integration and are needed for every subinterval, or “node,” in the tree before the λ_l and λ_r can be efficiently calculated. It so happens that only algebraic expressions with the α_l , α_r , β_l , β_r , δ_l , and δ_r on two adjacent subintervals are needed to compute these inner products corresponding to the parent interval. In other words, $P^{-1}\psi_l(t)$, $P^{-1}\psi_r(t)$, and $P^{-1}\tilde{f}(t)$ are not needed on the parent interval to calculate these inner products. The process of computing these inner products for larger intervals is continued up the tree until α_l , α_r , β_l , β_r , δ_l , and δ_r are calculated for the last parent $[a, c]$. Essentially, a divide-and-conquer approach is done to calculate these inner products on $[a, c]$ instead of directly using the large inverse problem $P_{[a,c]}^{-1}\tilde{f}$.

The final process is to compute the constants λ_l^B and λ_r^B for each leaf subinterval. An approach similar to the calculation of the above inner products is used here, except that these constants defined on each subinterval are computed by moving from the top of the tree to the finest subintervals. It is shown that, for each child interval I , λ_l^I and λ_r^I can be computed from algebraic expressions with only the α_l , α_r , β_l , β_r , δ_l , δ_r , λ_l , and λ_r corresponding to the parent (α_l , α_r , β_l , β_r , δ_l , and δ_r are known from the previous tree traversal). Thus, given the $\lambda_l^{[a,c]}$ and $\lambda_r^{[a,c]}$ for the top “node” $[a,c]$, those for all children are computed, moving down the tree. Since $\sigma_{[a,c]} = P_{[a,c]}^{-1}\tilde{f}$ is the inverse problem on $[a, c]$ (the original problem), the initialization is $\lambda_l^{[a,c]} = 0$ and $\lambda_r^{[a,c]} = 0$. Now, according to (41), σ is calculated on every subinterval (the leaf nodes that were chosen in advance).

In essence, the λ_l and λ_r for each leaf “node” are calculated from the integrations on the other leaves, thus tying the local solution to the global solution.

Spectral Chebyshev integration of order p is implemented on all the finest intervals to find $P_B^{-1}\tilde{f}$, $P_B^{-1}\psi_l$, and $P_B^{-1}\psi_r$ on p Chebyshev points. This yields $p \times p$ matrices, and solving uses $O(p^3/3)$ flops. For all M intervals, the complexity becomes $O(Mp^3/3)$. This part requires most of the work in the algorithm. As we have stated before, spectral integration is used to perform the p -point inner products α_l , α_r , β_l , β_r , δ_l , and δ_r for every “node.” The result is that all solution values for σ are found at the p Chebyshev points on each subinterval.

Finally, spectral integration is used to calculate the inner product for u_h in (35).

3.3 Comparison with the code for constant coefficients

A comparison of the number of functions evaluations between the code with subdivisions (ADAP) and our MATLAB code (CC) was performed on the four problems given above. Tables 3-5 refer to three of the problems, (30),(31), and (33), respectively. For the constant coefficient code, a graduated set of orders was input until the highest precision was reached, whereas, for the code with subdivisions, four sets of equal subdivisions were input: 1, 10^1 , 10^2 , and 10^3 . To simulate the subdivided case without adaptivity ADAP was run with one iteration on the given mesh. We call this application of the code “SUB” in the tables. Because of a limit of 1024 subintervals in the code, the order of magnitude of the size of the sets could not be increased further. This explains the limits of 3000, 7000, and 16000. In the tables, the error for both methods is the order of magnitude of the L_2 error relative to the exact solution. For SUB, the number of function evaluations equals the order multiplied by the the number of subdivisions. This data is listed for orders 3, 7, and 16.

These tables help to confirm the implication of the results from the constant coefficient code: using a higher order on fewer subdivisions is often more efficient than using a lower order on many subdivisions. The extreme case of a single subinterval proves optimal. A higher order captures oscillations and layers very well. It must be understood, however, that the count of function evaluations indicates the total number of nodes used in these methods and only indirectly indicates the total work done in the program. For “complex” right-hand-sides, this portion of the algorithm would involve most of the work; in other cases, other portions of the algorithm, such as linear system solves, must be taken into account.

In Problem (31), when the order is increased, more interpolation points are automatically placed in the region of the boundary layers because of the Chebyshev zero distribution. Naturally, fewer subdivisions in the layers are needed in this case.

L_2	CC	SUB		
		(3)	(7)	(16)
$> 10^{-1}$	5	3		
10^{-1}			7	
10^{-3}	10	30		
10^{-6}	15	300		
10^{-7}			70	16
10^{-9}	20			
10^{-10}		3000		
10^{-13}	25			
10^{-14}			700	
10^{-15}	30,35		7000	160,1600

Table 3: SUB vs. CC, Problem (30)

L_2	CC	SUB		
		(3)	(7)	(16)
$> 10^{-1}$		3		
10^{-1}	40	30	7	
10^{-2}	50		70	16
10^{-3}	60	300		
10^{-4}	70			
10^{-5}	80			
10^{-6}		3000		160
10^{-7}	100		700	
10^{-10}	120			
10^{-12}	130			
10^{-13}	150			
10^{-14}			7000	1600,16000

Table 4: SUB vs. CC, Problem (31)

L_2	CC	SUB		
		(3)	(7)	(16)
$> 10^{-1}$		3000	700	16,160
10^{-1}	600	3,30,300	7,70	
10^{-3}	650			
10^{-4}	660			1600
10^{-6}	670		7000	
10^{-8}	680			
10^{-9}	690			
10^{-12}				16000
10^{-13}	750			

Table 5: SUB vs. CC, Problem (33)

It is interesting to note that, in Problem (33) with highly oscillatory behavior (Table 5), a relatively large number of subdivisions are needed before the error begins to decrease; in fact, the errors *increase* before they start to decrease. This occurs perhaps because a higher resolution is needed for the lower order solutions, i.e., a higher sampling rate of the oscillation is needed since 3rd and 7th order polynomials have slower oscillation. To capture the frequency in a subinterval, more 3rd order polynomials are needed than 16th order ones in the same interval. Once the proper frequency has been attained, the error drops off rapidly as the number of subdivisions is increased.

3.4 Comparison with COLNEW

The number of function evaluations necessary to achieve a given error were compared with those of COLNEW, using both order 3 and order 7 polynomials. We clarify, again, that the error used for the solutions from Lee's code (SUB) was an exact error, while that for COLNEW was an error estimate. An upper bound is given for both errors: we use the tolerance to represent the error in COLNEW, and the exact error for SUB is given as one order of magnitude greater than the actual value. Again, COLNEW was initialized with a regular 5-interval mesh. Seven problems were tested, including the four above and three additional ones: one with a shock at $x = 0$,

$$\begin{aligned} \varepsilon u'' + 2xu' &= 0, \\ u(-1) &= -1, u(1) = 1, \varepsilon = 10^{-5}; \end{aligned} \quad (43)$$

one with a Bessel function solution,

$$\begin{aligned} u'' + \frac{1}{x}u' + \frac{x^2 - \nu^2}{x^2}u &= 0, \\ u(0) &= 0, u(600) = 1, \nu = 100; \end{aligned} \quad (44)$$

and one with an exponentially small eigenvalue and boundary layers,

$$\begin{aligned} \varepsilon u'' - xu' + u &= 0, \\ u(-1) &= 1, u(1) = 2, \varepsilon = 1/70. \end{aligned} \quad (45)$$

The three solutions are depicted in Figure 5 below.

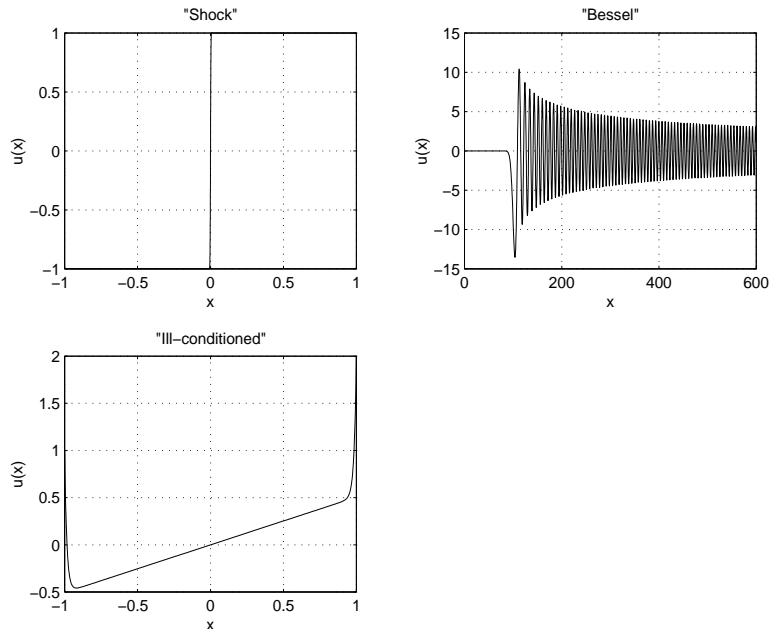


Figure 5: Problems (43) (“Shock”), (44) (“Bessel”), (45) (“Ill-conditioned”)

TOL	COLNEW		SUB			L_2
	(3)	(7)	(3)	(7)	(16)	
10^{-1}	45	105	3			10^0
10^{-3}	117	105		7		10^{-1}
10^{-5}	495	105	30			10^{-2}
10^{-7}	975	245	300			10^{-5}
10^{-9}	3855	525		70	16	10^{-6}
10^{-11}	15375	525	3000			10^{-9}
10^{-13}	61455	1085		700		10^{-13}
10^{-15}		2205		7000		10^{-14}
					160,1600,16000	10^{-15}

Table 6: SUB vs. COLNEW , Problem (30)

TOL	COLNEW		SUB			L_2
	(3)	(7)	(3)	(7)	(16)	
10^{-1}	42	98	3,30,300,3000	7,70,700	16,160	$> 10^{-1}$
10^{-3}	39720	9317		7000		10^{-7}
10^{-5}	58779	9863			1600	10^{-8}
10^{-7}	91695	14266			16000	10^{-10}
10^{-9}	178977	22743				

Table 7: SUB vs. COLNEW , Problem (44) (“Bessel”)

Table 6 shows statistics for solving problem (30) and is representative of the constant coefficient problems. Here, COLNEW attains its own limit in the number of subdivisions as the tolerance is decreased. The spectral code performs slightly better than COLNEW. With oscillations distributed throughout the interval, subdividing throughout the interval works well.

The code behaves similarly with Problem (33) and the “Bessel” problem (44) (Table 7). More nodes are needed for SUB for adequate resolution of the oscillations than in Problem (30); the number of subdivisions must increase considerably before the error begins to decrease. Still, when the error begins to decrease, the code needs fewer evaluations than COLNEW (for orders 7 and 16). For the 3rd order, if the subdivision were continued, it is possible that the complexity would be less than 39,720 for an error bound of 10^{-3} .

The spectral code does not seem to perform as well for sharp transitions, boundary layers, and other similar features. The most extreme example is the ill-conditioned problem (45) (Table 8). For these types of solutions, adaptive schemes are especially important because effort is not wasted in

TOL	COLNEW		SUB			L_2
	(3)	(7)	(3)	(7)	(16)	
10^{-1}	285	105	3,30,3000	7,70	16,1600,16000	$> 10^{-1}$
10^{-3}	537	501725	300	700,7000	160	10^{-1}
10^{-5}	570	814457				
10^{-7}		645085				
10^{-9}		645085				
10^{-11}		645085				
10^{-13}		860125				

Table 8: SUB vs. COLNEW, Problem(45) (“Ill-Conditioned”)

regions away from the high derivatives. Again, many points are needed to resolve the boundary layer since the Chebyshev polynomials of order 3 and 7 do not have large enough derivatives. In the case of orders 3 and 16, the error increases between 100 and 1000 subintervals. This same phenomenon was found to occur with COLNEW. When the density of the initial regular mesh was increased past a certain point, more work (nodes) was needed to achieve the same tolerance.

4 Adaptive strategy

4.1 Development

From observing the nonadaptive code's behavior when large-derivative regions are present, we realize that an adaptive approach is needed to improve efficiency.

One such technique has been developed and implemented in [8]. It is elegant in that the monitor function used to determine subdivisions is simply based on the last two Chebyshev coefficients in each subinterval for σ , the solution in (37) that we are actually computing. The motivating theorem essentially states the following: given a piecewise *linear* Green's function for the solution and three additional conditions, the relative error between the numerical solution $\bar{\sigma}$ and the exact solution σ is bounded as

$$\frac{\|\bar{\sigma} - \sigma\|}{\|\sigma\|} \leq \kappa(P)(\|P - \bar{P}\| + 1)\varepsilon, \quad (46)$$

where κ is the condition number of P , and \bar{P} is the discretized operator for P . The three other necessary conditions are:

$$\sigma_i^{K-1} < \left(\frac{\varepsilon}{2}\right) \|\tilde{f}\|, \quad \text{for all } B_i \quad (47)$$

$$\sigma_i^{K-2} < \left(\frac{\varepsilon}{2}\right) \|\tilde{f}\|, \quad \text{for all } B_i \quad (48)$$

$$\|\bar{f} - \tilde{f}\| < \varepsilon \|\tilde{f}\|, \quad (49)$$

where σ_i^{K-1} and σ_i^{K-2} are the last and penultimate coefficients, respectively, of the local solution in a subinterval B_i ; \tilde{f} is the discretized f evaluated at the Chebyshev points. Essentially, the error is of order ε if the last two Chebyshev coefficients are of order ε , given that \tilde{f} is sufficiently well discretized. The idea, then, is to subdivide those intervals whose coefficients are not small enough, and to merge two adjacent intervals whose coefficients are very small. Even though this theorem applies to linear Green's functions, similar requirements exist for the other Green's functions used in the code.

The monitor function for each subinterval i becomes

$$S_i = |\sigma_i^{K-2}| + |\sigma_i^{K-1}|. \quad (50)$$

Lee and Greengard observe in [8] that the left and right spectral integration matrices in the code are singular, such as those arising in the discretization of (39). In particular, if K is the order, then the function

$$T_{K-1}(x) + T_{K-3}(x) + T_{K-5}(x) + \dots + T_{0(1)}(x) \quad (51)$$

is a null vector. If K is even, this represents the odd bases; if K is odd, it represents the even ones. To account for this, Lee and Greengard modify (50) slightly to subtract off magnifications of roundoff errors that can appear in the direction of the null vector determined by K . The code ADAP actually uses this modified version.

Once the monitor function is calculated for each subinterval, then

$$S_{div} = \max_{i=1 \dots M} \frac{S_i}{2^C} \quad (52)$$

is calculated, with C being provided by the user. The subinterval i is divided if $S_i \geq S_{div}$. Furthermore, if subintervals i and $i + 1$ are children of the same parent, and $(S_i + S_{i+1}) < S_{div}$, then the two intervals are merged. In [8], it is recommended to set C to 4.0, and in our experiments this has been chosen. Whenever interval merging or splitting is executed in the code, only the local integral equations on the *new* subintervals must be solved. The tree is then traversed for the coefficients λ_l and λ_r , and σ is calculated as before.

Termination conditions are determined by the numerical solution, and not by the coefficients, however. Two error checks are made against the tolerance: one is the relative change between the last two iterates, and the other is the relative change between the last iterate and the solution obtained after every interval is halved (i.e., with twice the number of nodes). Expressly, for both errors we require

$$\left\| \frac{u_r - u_{r-1}}{u_r + u_{r-1}} \right\| < TOL, \quad (53)$$

where u_r is the last (doubled) iterate, and u_{r-1} is the next to last (last) iterate. The error estimates provided are the relative error above, between the last two iterates, and the absolute error between the last iterate and the solution on the doubled mesh. The user is given the option whether to check the latter error. In the few experiments we performed, we found that the former error estimate usually yielded an exact error well within the tolerance, especially for higher orders. The second error check yielded an exact error much smaller than requested, and proved to be very expensive: the number of function evaluations roughly doubled. Thus, our comparisons below are based on the first estimate only.

4.2 Comparison with the constant coefficient scheme

Tables 9 (for Problem (30)) and 10 (for Problem (31)) juxtapose the number of right-hand-side evaluations for the adaptive code (ADAP) and the constant coefficient code at three orders. Again, these problems represent oscillatory solutions and solutions with layers. As before, the number of function evaluations needed to attain exact L_2 errors are shown. The tolerances requested by the user are also provided; they are expressed in parentheses as a power of ten. As with COLNEW, increasing odd powers of the tolerance were passed to the adaptive code until the required mesh size exceeded the limit imposed by the code. Again, an initial regular mesh of 5 intervals was applied.

Again, in terms of the total number of nodes used, the constant coefficient code is much more efficient. Furthermore, most of the work is in the FFT, whereas, in the adaptive code, many linear systems must be solved. In most cases, the exact error proves to be much less than the required error (tolerance). This is probably indicative of a strict monitor function. For order 16, for example, a tolerance of 10^{-1} yields an exact error of 10^{-6} . We note again that a higher order requires fewer nodes (or subdivisions) in the adaptive case.

4.3 Comparison with the nonadaptive scheme

The number of function evaluations are shown for Problem (44) (“Bessel”) in Table 11 and for Problem (43) (“shock”) in Table 12. The power of 10 for the tolerance in the adaptive code is shown in parentheses.

For orders 3 and 16, the adaptive code does not perform as well on the “Bessel” problem, and it is difficult to compare the behaviors at order 7 because of the subdivision limit. We note that probably fewer than 16,000 nodes were needed for SUB(16) to achieve 10^{-12} . For the “shock,” the adaptive code is more efficient, as expected. When the “areas of difficulty” in the solution are widely distributed, the work in the adaptive code tends to be excessive, whereas, for local regions of high derivatives the adaptive scheme is useful.

$L_2(\text{exact})$	ADAP			CC
	(3)	(7)	(16)	
10^{-1}				5
10^{-3}	45(-1)			10
10^{-4}	201(-3)			
	2541(-5)			
	19917(-7)			
	158841(-9)			
10^{-6}		63(-1)		15
		63(-3)		
10^{-8}		147(-5)		
10^{-9}				20
10^{-11}		581(-7)		
10^{-13}				25
10^{-14}		1169(-9)		
10^{-15}		2289(-11)		30,35
		9121(-13)		
		29603(-15)		
10^{-16}			144(-1,-3,...,-11)	
			208(-13,-15)	

Table 9: ADAP vs. CC, Problem (30)

$L_2(\text{exact})$	ADAP			CC
	(3)	(7)	(16)	
10^{-1}				30,40
10^{-2}				50
10^{-3}	87(-1)	91(-1)		60
	315(-3)			
	1977(-5)			
	15183(-7)			
10^{-4}				70
10^{-5}				80
10^{-6}		147(-3)	114(-1,-3,-5)	
		231(-5)		
10^{-7}				100
10^{-9}		1001(-7)		
10^{-10}				120
10^{-12}				130
10^{-13}		5873(-9)	272(-7)	150,160,200
		9527(-11)	432(-9,-11)	
10^{-14}			784(-13)	
10^{-15}		51513(-13)		

Table 10: ADAP vs. CC , Problem (31)

$L_2(\text{exact})$	(3)		(7)		(16)	
	ADAP	SUB	ADAP	SUB	ADAP	SUB
$> 10^{-1}$	24378(-1)					16,160
	95622(-3)					
10^{-1}		3,30,300,		7,70,700,		
		3000		7000		
10^{-4}			6167(-1)			
10^{-5}			12229(-3)			
10^{-7}					2064(-1,-3,-5)	
10^{-8}			18193(-5)			
10^{-9}						1600
10^{-10}			33187(-7)		3952(-7)	
10^{-11}			91609(-9)			
10^{-12}					4176(-9)	16000
					4336(-11)	

Table 11: ADAP vs. SUB , Problem (44) (“Bessel”)

$L_2(\text{exact})$	(3)		(7)		(16)	
	ADAP	SUB	ADAP	SUB	ADAP	SUB
$> 10^{-1}$		30		70		16,160
10^{-1}				7		
10^{-2}	2157(-3,-5)	3,300		700		
	4467(-7)					
	25707(-9)					
10^{-3}	1167(-1)					
10^{-4}					560(-1)	
10^{-5}		3000	245(-1)			1600
10^{-7}			301(-3)		720(-3)	
10^{-8}			357(-5)			
			497(-7)			
			1365(-9)			
10^{-9}				7000		
10^{-11}					816(-5)	
10^{-12}					912(-7,-9,-11)	
10^{-14}			3689(-11)		1232(-13)	
			5033(-13)			
10^{-15}						16000

Table 12: ADAP vs. SUB , Problem (43) (“Shock”)

4.4 Comparison with COLNEW

Three additional problems were used to compare the adaptive code and COLNEW. Problem (54) has a cusp at the origin:

$$\begin{aligned} \varepsilon u'' + xu' - \frac{1}{2}u &= 0, \\ u(-1) = 1, u(1) = 2, \varepsilon &= 10^{-10}. \end{aligned} \tag{54}$$

Problem (55) models a subatomic particle with a potential barrier:

$$\begin{aligned} \varepsilon u'' + (x - .25)u &= 0, \\ u(-1) = 1, u(1) = 2, \varepsilon &= 10^{-6}; \end{aligned} \tag{55}$$

and problem (56) is essentially the same as (54), except with the cusp shifted to a location that corresponds to an irrational number, $\sqrt{2} - .8$:

$$\begin{aligned} \varepsilon u'' + (x - \sqrt{2} + .8)u' - \frac{1}{2}u &= 0, \\ u(-1) = 1, u(1) = 2, \varepsilon &= 10^{-10}. \end{aligned} \tag{56}$$

The solutions of the three problems are depicted in Figure 6.

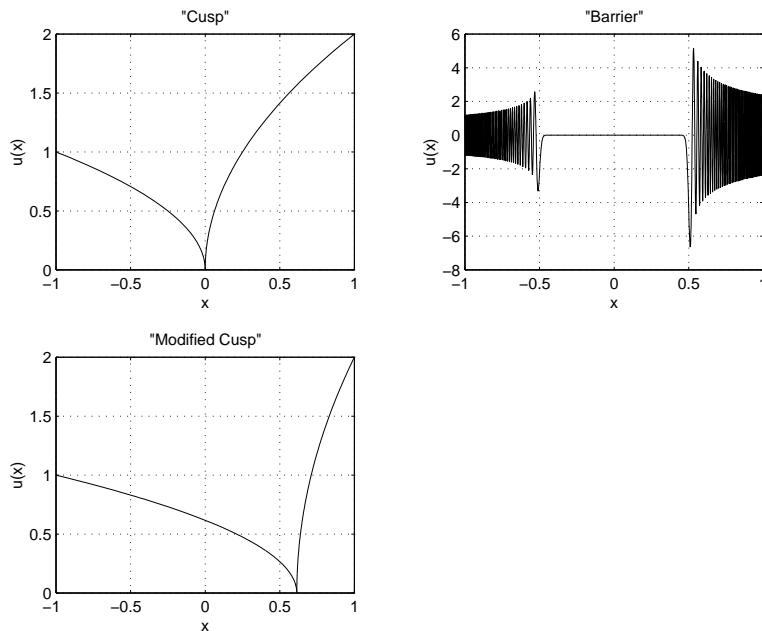


Figure 6: Problems (54) (“Cusp”), (55) (“Barrier”), (56) (“Modified Cusp”)

A representative selection of problems is (30), (45) (“Ill-conditioned”), (55) (“Barrier”), (54) (“Cusp”), and (56) (“Modified cusp”). Results are given in Tables 13-17. In cases where the exact errors are available from ADAP, the number of function evaluations is listed according to this error, with tolerances in parentheses. Where they are not available, results from both COLNEW and ADAP are listed with the tolerance. We again stress that in ADAP the tolerance is an upper bound on the relative change from the penultimate to the last iterate; we also reiterate that the tolerances were decreased for both codes until the maximum permissible number of subdivisions was attained. In all experiments, the initial mesh consisted of 5 regular subintervals for both COLNEW and ADAP.

In Table 13, we notice that for a lower order, COLNEW appears to be more efficient; for a higher order, ADAP is comparable. We might have expected ADAP to outperform COLNEW since the code updates the linear system solutions only locally, whereas COLNEW updates throughout the interval; however, in this problem, the oscillation is distributed so that updates will also be distributed.

TOL	COLNEW		ADAP			L_2
	(3)	(7)	(3)	(7)	(16)	
10^{-1}	45	105	45(-1)			10^{-3}
10^{-3}	117	105	201(-3)			10^{-4}
10^{-5}	495	105	2541(-5)			
10^{-7}	975	245	19917(-7)			
10^{-9}	3855	525	158841(-9)			
10^{-11}	15375	525		63(-1,-3)		10^{-6}
10^{-13}	61455	1085		147(-5)		10^{-8}
10^{-15}		2205		581(-7)		10^{-11}
				1169(-9)		10^{-14}
				2289(-11)		10^{-15}
				9121(-13)		
				29603(-15)		
					144(-1,-3,...-11)	10^{-16}
					208(-13,-15)	

Table 13: ADAP vs. COLNEW, Problem (30)

TOL	(3)		(7)		(16)
	COLNEW	ADAP	COLNEW	ADAP	ADAP
10^{-1}	21765	790000	105	10038	1840
10^{-3}	42645		6755	14560	2224
10^{-5}	142947		9821	22694	3056
10^{-7}	302184		29365	53816	4336
10^{-9}	886452		36505	159128	6032
10^{-11}			71162		7312

Table 14: ADAP vs. COLNEW, Problem (55) (“Barrier”)

Table 14 compares the codes on the “barrier” problem. Since the exact solution for this problem was not available, the comparison is based on the tolerance. Here, ADAP is inferior for orders 3 and 7, but improves at order 16. In fact, order 3 does not prove to be very useful in most of the problems investigated.

The “ill-conditioned” (Table 15) problem serves as a good example for the layer problems. As in previous tables, we see that the error can increase with a denser mesh, and that the mesh size exceeds the maximum before the error decreases again. This problem is a rare example that shows a poorer performance for 16th order. Problem (45) is highly unstable, with exponential modes. When we solve small dense linear systems in these boundary layers, we are not, in effect, decoupling the growing modes from the decaying modes. The growth in these modes is so rapid that instability can be problematic even in the solution of these small linear systems.

The “cusp” problems (Tables 16-17) serve as additional examples of layer problems. Again, we rely on the tolerances for the error. This data implies a superior performance by ADAP. The “modified cusp” problem was applied to test the code’s ability to “find” the layer at an irrational location, as opposed to a location of zero. (In the subdivision process, due to the symmetry of the interval about zero, a subinterval endpoint is expected to appear at zero.) Varying the location of the cusp appears to have made little difference.

Overall, for the layer problems, Lee and Greengard’s adaptive code competes well with COLNEW; for oscillatory problems, it does not compete as well. These results contradict our intuition: since Chebyshev polynomials are cosine functions under a transformation, it seems that ADAP would approximate well those solutions with sinusoidal components.

TOL	COLNEW		ADAP			L_2
	(3)	(7)	(3)	(7)	(16)	
10^{-1}	285	105		411649(-7,-9,-11,-13)	272(-3,-7,-9)	10^{-2}
10^{-3}	537	501725			464(-11,-13)	
10^{-5}	570	814457	105(-1)		144(-1)	10^{-3}
10^{-7}		645085	237(-3)			
10^{-9}		645085	1329(-5)			
10^{-11}		645085	10029(-7)			
10^{-13}		860125	159069(-9)			
				119(-1)		10^{-5}
				147(-3)		10^{-6}
				231(-5)		

Table 15: ADAP vs. COLNEW, Problem (45) (“Ill-conditioned”)

TOL	(3)		(7)		(16)
	COLNEW	ADAP	COLNEW	ADAP	ADAP
10^{-1}	105	414	105	324	368
10^{-3}	491505	1890	501725	420	1008
10^{-5}	833793	2076	814457	448	1008
10^{-7}	614385	2076	645085	504	1008
10^{-9}	614385	4494	645085	54124	1072
10^{-11}	614385	12600	645085	168294	1072
10^{-13}			860125		

Table 16: ADAP vs. COLNEW, Problem (54) (“Cusp”)

TOL	(3)		(7)		(16)
	COLNEW	ADAP	COLNEW	ADAP	ADAP
10^{-1}	105	1164	105	147	272
10^{-3}	614385	1611	501725	658	1232
10^{-5}	552945	1611	581000	770	1232
10^{-7}	552945	1611	645085	27580	1232
10^{-9}	552945	2559	645085	78778	1680
10^{-11}		11217		78778	2608

Table 17: ADAP vs. COLNEW, Problem (56) (“Modified cusp”)

5 Conclusions

Computing the complexity in terms of the number of function evaluations only indicates the effort used in refining the mesh, by showing the total number of nodes used. In general, ADAP appears to be competitive with COLNEW at higher orders. We have shown the dramatic improvement in efficiency as the order is increased from 3 to 16. However, in terms of the overall complexity of the algorithm, increasing the order indefinitely will not improve efficiency indefinitely: as the order N increases, less refinement is needed, but larger systems must be solved. Eventually, we cease to solve local problems and resume solving a large dense system.

An obvious advantage of Lee and Greengard's method is its ability to solve systems only locally for refinement. The bulk of the work is in the $O(N \log p)$ operations that follow, where the binary tree must be traversed. Maintenance of this tree data structure, however, is a complex procedure that may counteract the efficiency of the system solves. It will be necessary, then, to compare the actual CPU times of COLNEW and ADAP to make a clearer comparison. Along the same lines, it may be worth investigating the CPU time needed for the spectral code on problems with variable coefficients and without subdivisions. These results might reveal whether the tree structure provides any real savings.

References

- [1] 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
- [2] N.L. Carothers (1994), *A Short Course on Approximation Theory*, Bowling Green State University, Dept. of Mathematics, preprint
- [3] P. Davis and P. Rabinowitz (1984), *Methods of Numerical Integration, Second Edition*, Academic Press, Inc., Orlando, FL
- [4] L. Fox and I.B. Parker (1968), *Chebyshev Polynomials in Numerical Analysis*, Oxford University Press, London
- [5] L. Greengard (1991), *Spectral integration and two-point boundary value problems*, SIAM J. Numer. Anal., 28, pp. 1071-1080
- [6] L. Greengard and V. Rokhlin (1991), *On the numerical solution of two-point boundary value problems*, Comm. Pure Appl. Math., 44, pp. 419-452
- [7] J. Lee (1994), "Ode_adap" software package, Version 1.0, Dept. of Mathematics, Ewha Women's University, Seoul, South Korea
- [8] J. Lee and L. Greengard (1997), *A fast adaptive numerical method for stiff two-point boundary value problems*, SIAM J. Sci. Comput., 18, pp. 403-429
- [9] T.J. Rivlin (1969), *An Introduction to the Approximation of Functions*, Dover Publications, Inc., New York