Adaptive techniques for spline collocation

Christina C. Christara and Kit Sun Ng
Department of Computer Science
University of Toronto
Toronto, Ontario M5S 3G4, Canada
{ccc,ngkit}@cs.utoronto.ca

July 18, 2005

Abstract

We integrate optimal quadratic and cubic spline collocation methods for second-order two-point boundary value problems with adaptive grid techniques, and grid size and error estimators. Some adaptive grid techniques are based on the construction of a mapping function that maps uniform to non-uniform points, placed appropriately to minimize a certain norm of the error. One adaptive grid technique for cubic spline collocation is mapping-free and resembles the technique used in COLSYS (COLNEW) [2, 4]. Numerical results on a variety of problems, including problems with boundary or interior layers, and singular perturbation problems indicate that, for most problems, the cubic spline collocation method requires less computational effort for the same error tolerance, and has equally reliable error estimators, when compared to Hermite piecewise cubic collocation. Comparison results with quadratic spline collocation are also presented.

AMS Subject Classification: 65L10, 65L20, 65L50, 65L60, 65L70, 65D05, 65D07.

Key words: spline collocation, second-order two-point boundary value problem, error bounds, optimal order of convergence, adaptive grid, grid size estimator, error estimator, spline interpolation.

1 Introduction

Optimal Quadratic Spline Collocation (QSC) and Cubic Spline Collocation (CSC) methods on non-uniform partitions have been recently developed [7] for the solution of linear two-point Boundary Value Problems (BVPs). The development and analysis of the methods in [7] are based on a function w, that maps uniform partition points to non-uniform ones. It was shown that, for a certain grid size N, when the mapping function is such that more points are placed in regions of large variation of the solution to the BVP and fewer in other regions, the observed errors are much smaller than when the same total number of equidistant points are used.

Under realistic situations, an appropriate mapping function for a given BVP is not known or given. In this paper, we present adaptive techniques for the construction of an appropriate mapping function for

a given BVP. We then introduce grid size and error estimators for QSC and CSC, and present adaptive techniques for solving a BVP within a certain error tolerance.

The outline of this paper is as follows. In Section 2, we describe an algorithm (PlaceMap), that, given a certain grid size N and a BVP, computes the best placement of the grid points to minimize a certain norm of the error. This algorithm can be integrated with QSC or CSC. We also describe a way to compute an (approximation to an) appropriate mapping function from uniform to non-uniform partitions, for the given problem. In Section 3, we present an algorithm (AdaptSolve1) for CSC, that, given a certain tolerance and a BVP, computes an approximation to the minimum grid size N and a respective CSC approximation to the BVP solution, so that the error in a certain norm is below the given tolerance. This technique is mapping-free and resembles the one in COLSYS (COLNEW) [2, 4], with a few differences which we elaborate. In the same section, we also present two alternative adaptive techniques for solving a BVP by QSC or CSC within a certain tolerance, that are based on the construction of a mapping function. In Section 4, we present numerical results that demonstrate the behaviour of the adaptive QSC and CSC methods on a variety of problems, including problems with boundary or interior layers, and singular perturbation problems. We compare the results from CSC and from Hermite piecewise cubic collocation (which we refer to as HPCC) as implemented in COLSYS.

Throughout this paper, we adopt the notation of [7]. Let $\Omega \equiv (0,1)$ be the domain of the BVP and $u(x)$ be the exact solution. Let also $\Delta \equiv \{x_0 = 0 < x_1 < \ldots < x_N = 1\}$ be a uniform partition of $\overline{\Omega}$ with stepsize $h \equiv 1/N$ and $T \equiv \{\tau_0 \equiv x_0, \tau_i \equiv (x_i + x_{i-1})/2; i = 1, \ldots, N, \tau_{N+1} \equiv x_N\}$ be a set of data points. Let $\Delta_w \equiv \{s_i, i = 0, \ldots, N\}$ be the partition of $\overline{\Omega}$ with respect to which the splines are defined, and $T_w \equiv \{w_i, i = 0, \ldots, N+1\}$ be the set of QSC points. The CSC points coincide with the points in Δ_w. The relation between the points in T_w and those in T and Δ_w is discussed in Section 2. Adopt also the notation $h_i \equiv w_{i+1} - w_i$, $i = 1, \ldots, N-1$, $h_i^a \equiv s_i - w_i$, $i = 1, \ldots, N$, $h_i^b \equiv w_{i+1} - s_i$ $i = 0, \ldots, N-1$, $H_i \equiv s_{i+1} - s_i$, $i = 0, \ldots, N-1$, and $H \equiv \max\{\max_{i=1,\ldots,N}\{h_i^a\}, \max_{i=0,\ldots,N-1}\{h_i^b\}\}$.

2 Construction of Mapping Function using Adaptive Techniques

In [7], we assumed that $s_i = w(x_i), i = 0, \ldots, N$, and $w_i = w(\tau_i), i = 0, \ldots, N+1$, for some predefined smooth enough and strictly monotone increasing bijective mapping function w. In this section, we relax the assumption that w is predefined, and discuss the use of adaptive techniques to generate mapping functions appropriate for the problem considered. We describe the techniques for QSC, but they can be easily adjusted for CSC.

The primary idea is to first construct a non-uniform grid s_i, $i = 0, \ldots, N$, using adaptive techniques, then construct a monotone Hermite piecewise cubic interpolant w such that $w(s_0) = s_0, w(s_N) = s_N$ and $w(\tau_i) = (s_{i-1} + s_i)/2$, $i = 1, \ldots, N$.

We use the adaptive techniques and the idea of *grading* functions presented in [5] to construct the non-uniform grid. According to these techniques, the partition points are distributed so that the error in some chosen norm (or semi-norm) is equidistributed among the subintervals of the partition. Depending on the norm chosen, a different grading function arises, based on which the partition points are constructed. A grading function is of the form $\xi(x) = \int_0^x \hat{u}dx / \int_0^1 \hat{u}dx$, for some appropriate *monitor* function $\hat{u}$. The value $\xi(x)$ of the grading function at x denotes the portion of the approximate error from the left endpoint

up to point x. All monitor functions involve high derivatives of u, which, under realistic situations, are not known. Therefore, the spline approximation u_Δ is substituted in place of u, to obtain the respective approximate grading functions.

According to [5], with u_Δ being a piecewise polynomial of degree 2, and the chosen norm being the H^1 semi-norm, the grading function is $\xi_{q5}(x) = \int_0^x (u^{(3)})^{2/5} dx / \int_0^1 (u^{(3)})^{2/5} dx$, while, if the chosen norm is the H^0 norm, the grading function is $\xi_{q7}(x) = \int_0^x (u^{(3)})^{2/7} dx / \int_0^1 (u^{(3)})^{2/7} dx$. De Boor [8] suggests that, for a method with error proportional to $h^p u^{(q)}$, a good grading function is $\int_0^x |u^{(q)}|^{1/p} dx / \int_0^1 |u^{(q)}|^{1/p} dx$. For QSC we take $q = 3$ (as the error formula for the interpolant suggests) and $p = 3$ (the global order), resulting in the grading function $\xi_{q6}(x) = \int_0^x (u^{(3)})^{2/6} dx / \int_0^1 (u^{(3)})^{2/6} dx$. An alternative is to equidistribute the arc length, as [1] suggests, resulting in the grading function $\xi_{q2}(x) = \int_0^x \sqrt{1 + (u')^2} dx / \int_0^1 \sqrt{1 + (u')^2} dx$.

The grading functions we considered for CSC are $\xi_{c7}(x) = \int_0^x (u^{(4)})^{2/7} dx / \int_0^1 (u^{(4)})^{2/7} dx$ and $\xi_{c9}(x) = \int_0^x (u^{(4)})^{2/9} dx / \int_0^1 (u^{(4)})^{2/9} dx$ (equidistributing the H^1 semi-norm and the H^0 norm of the error according to [5]), $\xi_{c8}(x) = \int_0^x (u^{(4)})^{2/8} dx / \int_0^1 (u^{(4)})^{2/8} dx$ (as [8] suggests) and $\xi_{c2}(x) = \xi_{q2}(x)$.

The following algorithm is taken from [5] and adjusted for the QSC method. Given a grading function $\xi(x)$ and a number of subintervals N, the algorithm computes points s_i, $i = 0, \ldots, N$, with $\xi(s_0) = \xi(0) = 0$ and $\xi(s_N) = \xi(1) = 1$, such that $\xi(s_i) - \xi(s_{i-1}) \approx 1/N$, $i = 1, \ldots, N$, or equivalently $\xi(s_i) \approx i/N$.

The algorithm works iteratively, with a stopping criterion suggested in [5]. That is, at each iteration, we calculate the "drift" $\max_i \{\int_{s_i}^{s_{i+1}} \hat{u} dx\} - \int_0^1 \hat{u} dx / N$ from the target placement of the points. The stopping criterion is then $drift < tol$, where tol is a user chosen tolerance. During the experiments, we noticed that sometimes the drift oscillates as the iterations proceed, resulting in the partition points vibrating with small amplitude. However, we noticed that the partition points were already distributed reasonably well. This phenomenon was also noticed in [5]. It may result in the adaptive algorithm ending without reaching the stopping criterion and leaving us with the last set of points, even if those were not the points of the smallest drift. To get the "most" from the adaptive method, at each iteration we save the set of partition points that gives the smallest drift, and use these to compute the interpolant.

Once we have the non-uniform grid, we use the algorithm for monotone piecewise cubic interpolation from [9] to generate the mapping function w. In this way, the mapping function is in $\mathbf{C}^1$, that is, it does not satisfy the assumptions of the theorems in [7]. However, it has worked well in all experiments, as will be shown in Section 4. It is worth noting that a piecewise linear $\mathbf{C}^0$ interpolant did not give satisfactory results.

For the approximation of all integrals arising in the algorithm we use the midpoint rule, since for QSC the midpoints are points of high accuracy and no discontinuities. Note that the approximation to u' is given directly by the derivative of the standard (first step) QSC approximation, while the approximation to $u^{(3)}$ is computed as shown in Section 2.2 of [7].

The outline of the algorithm, which we refer to as **PlaceMap**, is as follows:

Pick N (usually $N = 32$) and initial mesh s_i, $i = 0, \ldots, N$ (usually uniform)
Pick grading function $\xi(x)$
For $itadpt = 1, \ldots, maxstep$ do
    Use the first step of QSC on mesh s_i, $i = 0, \ldots, N$, with $\tau_i = (s_{i-1} + s_i)/2$, $i = 1, \ldots, N$,
        to approximate u

Approximate the appropriate derivatives of u at τ_i, $i = 1, \ldots, N$
Compute new points s_i, $i = 0, \ldots, N$, redistributed according to ξ
Calculate drift
If drift smaller than all previous iterations, save s_i, $i = 0, \ldots, N$
If $drift < tol$ then exit loop
End
Use s_i, $i = 0, \ldots, N$, to construct monotone Hermite interpolant w
Output N, s_i, $i = 0, \ldots, N$, w, and approximation to u

In the case that the data of the problem are given only on certain points, and we do not have a way to evaluate the source terms and the coefficient functions of the problem at the points chosen by the adaptive technique, the QSC method is still applicable. Given a set of points $T_w \equiv \{w_0 = 0 < w_1 < \ldots < w_{N+1} = 1\}$, we can construct a monotone Hermite piecewise cubic interpolant w such that $w(\tau_i) = w_i$, $i = 0, \ldots, N+1$. Once we have the mapping function w, we can define the nodes $s_i = w(x_i)$, $i = 0, \ldots, N$, of the partition, then setup the collocation equations, since all stepsizes we need are now defined.

It is important to note that the algorithm **PlaceMap** is usually applied to a relatively small grid size N, usually $N = 32$. Once w has been constructed, this same w is used to generate the nodes and the collocation points for other grid sizes, by mapping points of a uniform partition of any grid size to the respective points of a non-uniform partition. There are some alternatives to this procedure, which we briefly discuss.

(a) We can apply the adaptive algorithm for any chosen N, and construct a mapping function based on the computed grid points. Thus the mapping function may be different for each N. This procedure is more costly, since both the adaptive and the interpolation algorithms are applied for larger N's. We tested this procedure numerically for some problems, but since the error results were compatible with those using the same mapping computed with a small N, we present only the latter in the next section. It is worth pointing out that, if the mapping function is to be constructed only once, the size of the grid on which it is constructed is crucial for the success of the method. In general, the appropriate grid size is not known apriori, thus, in most realistic situations, the mapping needs to be constructed for any N, or for a large enough N, determined based on the problem.

(b) We can get rid of the mapping completely, by applying the adaptive algorithm for any chosen N, then computing the position of the non-uniform midpoints by Taylor expansions based at the grid points. More specifically, once the grid points are computed, we have H_i, $i = 0, \ldots, N-1$, and from these the stepsizes h_i^a, $i = 1, \ldots, N$, and h_i^b, $i = 0, \ldots, N-1$, can be computed using the expansions of Lemma 1 in [7] and similar expansions for H_i, $i = 0, \ldots, N-1$. We developed $O(h^3)$ and $O(h^4)$ approximations of h_i^a and h_i^b and tested them numerically on some problems. The order of convergence obtained was about 3 or a bit above 3, but below the optimal 4. Therefore, we do not present these results. From the above discussion, it becomes clear that it is difficult to get rid of the mapping function in non-uniform QSC, since the method requires both the non-uniform "midpoints" w_i, and the grid points s_i. However, as mentioned in [7], the implementation of CSC is essentially mapping-free.

3 Adaptive Mesh Generation

For practical purposes, a user needs to solve a problem within a certain error tolerance. The above procedure of adaptive placement of grid points and mapping function computation can be useful in this context, if it is combined with grid size and error estimators. Extrapolation through two runs, one of double size as the other can be used to estimate the error and the grid size needed to reach a given tolerance. However, for the grid size estimator to be reliable, a procedure is needed to gradually advance the grid size, until enough points are taken so that the solution behaviour is properly captured.

Given the above, we have implemented an adaptive mesh generation technique similar to the one in COLSYS, with a few differences. This technique can be used with CSC or QSC. It is important to note, though, that the CSC method only requires the grid points (and not both the midpoints and grid points), therefore, it does not necessarily need the computation of a mapping function. We also note that the computation of the mapping function for large grid sizes increases the computational cost of the method. We present here an overview of the adaptive technique in COLSYS and elaborate on the differences we applied for CSC.

We first select a grading function $\xi(x)$. For an mth order BVP, the grading function arising from the monitor function chosen in COLSYS is $\xi_G(x) = \int_0^x (u^{(k+m)})^{1/(k+m)} dx / \int_0^1 (u^{(k+m)})^{1/(k+m)} dx$, which, for Hermite piecewise cubic polynomials ($k = 2$) and second-order BVPs ($m = 2$), is the same as ξ_{c8}.

We now introduce grid size and error estimators. In the following discussion, for convenience, the subscripts $[N]$ and $[2N]$ denote number of subintervals used to compute piecewise polynomial or spline approximations (and *not* values of functions at the respective collocation points).

The grid size estimation in each iteration of the adaptive procedure in COLSYS is done according to a formula developed by theory. The error $\epsilon^G_{[N]}(x) \equiv u^G_{[N]}(x) - u(x)$ in the Gaussian spline collocation approximation $u^G_{[N]}$ with grid size N is proportional to $h^{k+m}u^{(k+m)}$ and to h^{2k} and satisfies $||\epsilon^G_{[N]}||_\infty \leq C(\frac{\theta}{N})^{k+m}(1 + O(h)) + O(h^{2k})$, where $\theta \approx \int_0^1 |u^{(k+m)}|^{1/(k+m)} dx$, and C is a constant that depends on m and k [3]. For Hermite piecewise cubic polynomials and second-order BVPs, $C = 1/384$. Given a tolerance TOL, COLSYS estimates that it will require a grid of size $NestC = \lceil \theta(\frac{C}{TOL})^{1/(k+m)} \rceil$ to obtain a spline approximation $u^G_{[N]}$ such that the error satisfies $||\epsilon^G_{[N]}(x)||_\infty \leq TOL$. However, for many problems the above grid size estimator often underestimates the required number of points, especially when θ is calculated by composite quadrature based on a small number of points.

The problem is partly overcome by toughening the user chosen tolerance by a factor of 10, a fact that we observed in the code of COLSYS. However, even with the tougher tolerance, the number of points is usually underestimated, and it takes several iterations of the adaptive procedure until a reasonable grid size estimation is made. That is, COLSYS gradually advances the mesh, and recomputes θ and $NestC$. Each time the grid size changes and new points are selected, the new points are checked for sufficient error equidistribution. Each time the grid size increases and the equidistribution of error improves, the grid size estimator becomes more reliable. The error estimation and the decision for termination of computation in COLSYS are done by extrapolation through two runs, one of double grid size as the other. Once $u^G_{[N]}$ and $u^G_{[2N]}$ have been computed, the error for any M is estimated by $\epsilon^G_M \approx \frac{u^G_{[2N]} - u^G_{[N]}}{2^\rho - 1}(\frac{2N}{M})^\rho$, where $\rho = k + m$ is the expected order of convergence of $u^G_{[N]}$. Thus, $\epsilon^G_{[2N]} \approx \frac{u^G_{[2N]} - u^G_{[N]}}{2^\rho - 1}$.

We have incorporated similar techniques for grid size and error estimators in our code. Let $\epsilon^3_{[N]} \equiv$

$u^3_\Delta(x) - u(x)$ be the error in the CSC approximation u^3_Δ with grid size N. If we follow COLSYS, the grid size estimator predicts that

$$NestC = \lceil \theta (\frac{C}{TOL})^{1/4} \rceil \tag{1}$$

subintervals are needed, to have $||\epsilon^3_{[NestC]}||_\infty \leq TOL$, where $\theta \approx \int_a^b |u^{(4)}|^{1/4} dx$. Note that we do not have a theoretical derivation for the value of C for CSC, but we calculated it experimentally based on the following arguments. We assume that the error behaves as $\epsilon^3_{[N]} \approx Ch^4 u^{(4)} +$ higher order terms. Consider the simple BVP $u'' = 12x^2$, with $u(0) = 0$ and $u(1) = 1$. This has the solution $u = x^4$, with $u^{(4)} = 24$, which implies that the error behaves as $\epsilon^3_{[N]} \approx 24Ch^4$. By applying CSC on this problem we found that $C = 1/384$. (By similar tests, for QSC, the midpoint error was found to behave as $h^4 u^{(4)}/128$.) However, as in COLSYS, we found that, for many problems, $NestC$ often underestimates the required number of points. (We did not use a factor of 10 to toughen the tolerance.) For the error estimation we use extrapolation as in COLSYS.
The outline of the algorithm, which we refer to as **AdaptSolve1**, is as follows:

Pick$N$ (usually $N = 32$) and initial mesh s_i, $i = 0, \ldots, N$ (usually uniform)
Pick grading function $\xi(x)$ and tolerance TOL
For $itadpt = 1, \ldots, maxstep$ do
    Use CSC on mesh s_i, $i = 0, \ldots, N$, to compute $u_{[N]}$
    Approximate the required derivatives of u on s_i, $i = 0, \ldots, N$
    Calculate $drift = \max_i \{\int_{s_i}^{s_{i+1}} \hat{u} dx\} / (\int_0^1 \hat{u} dx / N)$
    If just_doubled, apply err_est, if $err\_est < TOL$, exit loop, end, end
    Decide whether to redistribute, double, half, or start over
End
If $TOL/4 < err\_est < TOL$ and $drift > 2$, redistribute once, end
Output N, s_i, $i = 0, \ldots, N$, $u_{[N]}$ and err_est

It is important to note that algorithm **AdaptSolve1** includes the construction of the adaptive mesh and the approximate solution to the problem within tolerance TOL. It does not compute a mapping function. (A mapping function may be implicitly defined by the computed location of the grid points, but it is never explicitly computed or used.) Notice also that the drift in this algorithm is calculated in a relative way, as in COLSYS, and not in an absolute way, as in [5] and **PlaceMap**.

The decision whether to redistribute, double, half, or start over with a new set of grid points is taken according to similar criteria as in COLSYS, with the following differences:

(a) COLSYS doubles the grid size if the same N has been used 3 consecutive times, or if $N/2$, N have been used alternatively 3 consecutive times, or if $drift \leq 2$. We also do the same, but we allow one more (fourth) redistribution of the points for the same N before doubling if $1.1 < drift \leq 2$ and $err\_est1/err\_est > err\_est/TOL$, where err_est1 is the first error estimate, err_est the current error estimate. The reason for this difference from COLSYS is that through our experiments we noticed that having a better point distribution helps when we are close to the desired tolerance, while it may worsen the results if we are far from the desired tolerance. Being far from the desired tolerance is usually due to the fact that the grid size is too small, so a proper point distribution cannot occur. Notice that the relation $err\_est1/err\_est > err\_est/TOL$ can be interpreted as being less than half-way away from the

desired error tolerance. Thus, we allow a fourth redistribution if the mesh is fairly well but not very well distributed, and if we are "close" to the desired tolerance.

(b) COLSYS computes the error estimate only when the grid is doubled. We compute the error estimate when the grid is doubled, as well as when a redistribution has taken place as long as the data for the half grid size have been computed. Thus we have a more updated error estimate. In addition, COLSYS uses a factor of 10 to toughen the tolerance when $NestC$ is computed. We do not use a factor of 10 to toughen the tolerance, but we do one more redistribution if $TOL/4 < err\_est < TOL$ and $drift > 2$. Through our experiments we found that the (updated) error estimate is in general reliable, except if the points are not well distributed.

(c) COLSYS starts over (i.e. picks a completely new grid size $N = NestC/2$ and a uniform grid) if $N < NestC < 2N$. We start over if $N < NestC < 2N/1.1$. The reason for the 1.1 factor is that when $NestC$ is too close to $2N$, starting over wastes all the information already computed (i.e. the distribution of points), while at the same time the choice $N = NestC/2$ will only save a very small number of points.

As mentioned above, algorithm **AdaptSolve1** does not compute a mapping. Doubling the grid size means making the current midpoints and grid points new grid points and setting $N = 2N$, just as COLSYS does. A variation of the algorithm implements doubling by computing a mapping function for each new mesh, then computing a double size mesh by mapping the uniform double size mesh to a non-uniform one. We refer to this variation as **AdaptSolve2**. Thus, **AdaptSolve2** for CSC computes a mapping for each new mesh except the last (largest) one. We implemented this algorithm for CSC, but, since the results were very similar to the results of **AdaptSolve1**, and since **AdaptSolve2** involves extra computation, we do not present them.

It is worth mentioning that algorithm **AdaptSolve1** can be easily adjusted for QSC, if we also take into consideration that, for QSC, the location of the collocation points ("midpoints") needs to be computed for each new mesh. Thus, **AdaptSolve1** for QSC computes a mapping function each time a new mesh (including the last one) is computed either by redistribution or by doubling or by choosing a new mesh size. Moreover, since, in the QSC case, the computation of the mapping function is not avoided, the implementation of **AdaptSolve2** does not involve extra computation, compared to **AdaptSolve1**. We have implemented both algorithms for QSC and present some results for comparison.

3.1 An alternative adaptive mesh generation algorithm

Algorithm **AdaptSolve1** (or its variation **AdaptSolve2**) can be used in a more indirect context. More specifically, the algorithm as presented above is used to select a grid size and an appropriate placement of the points, so that the solution computed for that grid reaches a certain tolerance. But, the same algorithm can be used to select an appropriate grid size, on which an effective mapping function is computed. Then, the mapping function and extrapolation can be used to compute the final grid size and placement of points, on which the solution is computed so that it reaches a certain tolerance. This procedure gives rise to an alternative adaptive mesh generation algorithm, which computes an approximation to u within tolerance TOL. We refer to it as **AdaptSolve3** and summarize it as follows:

Apply algorithm **AdaptSolve1** with tolerance $\sqrt{TOL}$
Let $\hat{N}$ be the mesh size selected by **AdaptSolve1**

Apply algorithm **PlaceMap** with $N = \hat{N}$
Let $\hat{s}_i$, $i = 0, \ldots, \hat{N}$, be the grid points, w the mapping function
and $u_{[\hat{N}]}$ the solution computed by **PlaceMap**
Generate points s_i, $i = 0, \ldots, 2\hat{N}$, by the mapping function w
Compute $u_{[2\hat{N}]}$ on s_i, $i = 0, \ldots, 2\hat{N}$
Use extrapolation between $\hat{N}$ and $2\hat{N}$ to obtain err_est
If $err\_est < TOL$, exit algorithm, end
Use extrapolation between $\hat{N}$ and $2\hat{N}$ to predict N so that $u_{[N]}$ reaches TOL
Generate points s_i, $i = 0, \ldots, N$, by the mapping function w
Compute $u_{[N]}$ on s_i, $i = 0, \ldots, N$
Use extrapolation between $2\hat{N}$ and N to obtain err_est
Output N, s_i, $i = 0, \ldots, N$, $u_{[N]}$ and err_est

Note that **AdaptSolve3** generates a mapping at a relatively small, but problem and tolerance dependent grid size $\hat{N}$. The choice of tolerance $\sqrt{TOL}$ (when running **AdaptSolve1** in order to calculate $\hat{N}$) is supported by our experiments and the following arguments. Using tolerance $\sqrt{TOL}$, we try to balance the trade-off between using too few points to calculate the mapping (as in the case of using $N = 32$ for all problems and tolerances) and using too many points (as in the case of **AdaptSolve2**, where redistributions are applied to all but the last grid). When too few points are used to calculate the mapping the behaviour of the solution may not be captured, while when lots of points are used, the cost increases, and, in addition, our experiments indicate that more points do not necessarily improve the effectiveness of the mapping function, but sometimes degrade it, possibly due to increased computational errors.

Algorithm **AdaptSolve3** can be implemented for CSC and QSC. We present results from this algorithm in the next section. Here we note that, such an algorithm is particularly effective with QSC, since it avoids the computation of mapping functions for large grids.

4 Numerical Results

In this section, we first present numerical results to demonstrate the convergence of the QSC and CSC methods for BVPs with non-uniform grids, using the adaptively computed mapping functions with the procedure **PlaceMap**. We then present results to demonstrate the effectiveness of the adaptive mesh generation algorithms presented in Section 3, with CSC and QSC.

All computations in this section were carried out in double precision. The QSC and CSC methods were programmed in MATLAB by us. The linear systems arising were solved by Gauss elimination using the backslash operator or the lu function in MATLAB. We used the MATLAB functions $pchip$ and $ppval$ to construct and evaluate a monotone Hermite piecewise cubic interpolant.

In our implementation, as basis functions for the quadratic spline space S_{Δ_w} we choose the functions

$\phi_i(x)$, $i = 0, \ldots, N+1$, where

$$\phi_i(x) \equiv \begin{cases} \frac{(x-s_{i-2})^2}{(s_i-s_{i-2})(s_{i-1}-s_{i-2})} & \text{for } s_{i-2} \le x \le s_{i-1} \\ \frac{(x-s_{i-2})(s_i-x)}{(s_i-s_{i-2})(s_i-s_{i-1})} + \frac{(s_{i+1}-x)(x-s_{i-1})}{(s_{i+1}-s_{i-1})(s_i-s_{i-1})} & \text{for } s_{i-1} \le x \le s_i \\ \frac{(s_{i+1}-x)^2}{(s_{i+1}-s_{i-1})(s_{i+1}-s_i)} & \text{for } s_i \le x \le s_{i+1} \\ 0 & \text{elsewhere.} \end{cases} \tag{2}$$

Note that $\phi_i(x)$ and $\phi_i'(x)$ are well-defined at the nodes. Whenever we need $\phi_i''(s_j)$, we define it by right (without loss of generality) continuity for $s_i, i = 0, \ldots, N-1$, and by left continuity for s_N.

As basis functions for the cubic spline space $S^3_{\Delta_w}$ we choose the functions $\phi_i^3(x)$, $i = 0, \ldots, N+2$, where

$$\phi_i^3(x) \equiv \begin{cases} \frac{(x-s_{i-2})^3}{(s_{i+1}-s_{i-2})(s_i-s_{i-2})(s_{i-1}-s_{i-2})} & \text{for } s_{i-2} \le x \le s_{i-1} \\ \frac{(x-s_{i-2})}{(s_{i+1}-s_{i-2})}\left(\frac{(x-s_{i-2})(s_i-x)}{(s_i-s_{i-2})(s_i-s_{i-1})} + \frac{(s_{i+1}-x)(x-s_{i-1})}{(s_{i+1}-s_{i-1})(s_i-s_{i-1})}\right) & \text{for } s_{i-1} \le x \le s_i \\ + \frac{(s_{i+2}-x)(x-s_{i-1})^2}{(s_{i+2}-s_{i-1})(s_{i+1}-s_{i-1})(s_i-s_{i-1})} & \\ \frac{(x-s_{i-2})(s_{i+1}-x)^2}{(s_{i+1}-s_{i-2})(s_{i+1}-s_{i-1})(s_{i+1}-s_i)} & \text{for } s_i \le x \le s_{i+1} \\ + \frac{(s_{i+2}-x)}{(s_{i+2}-s_{i-1})}\left(\frac{(x-s_{i-1})(s_{i+1}-x)}{(s_{i+1}-s_{i-1})(s_{i+1}-s_i)} + \frac{(s_{i+2}-x)(x-s_i)}{(s_{i+2}-s_i)(s_{i+1}-s_i)}\right) & \\ \frac{(s_{i+2}-x)^3}{(s_{i+2}-s_{i-1})(s_{i+2}-s_i)(s_{i+2}-s_{i+1})} & \text{for } s_{i+1} \le x \le s_{i+2} \\ 0 & \text{elsewhere.} \end{cases} \tag{3}$$

In all tables, the notation $x.y \pm z$ means $x.y \times 10^{\pm z}$. The observed errors of QSC and CSC are denoted by ϵ and ϵ^3, respectively. The uniform norm $\|\cdot\|_\infty$ is approximated by the maximum absolute value on a constant grid of 2001 evaluation points, independently of the discretization grid.

We present results from experiments with the mapping functions constructed by the adaptive technique **PlaceMap** on the following three Problems.

PROBLEM 1 $\quad \{(1+\eta x)u'\}' = 0 \quad \text{in} \quad (0,1), \quad u(0) = 0, \quad u(1) = 1.$

The solution of this problem is $u(x) = \frac{\log(1+\eta x)}{\log(1+\eta)}$, and has a boundary layer at $x = 0$, the sharpness of which is controlled by the magnitude of η. Problem 1 was taken from [6] and was also used in [7].

PROBLEM 2 $\quad u'' + u' - u = g \quad \text{in} \quad (0,1), \quad u(0) = 0, \quad u(1) = 0.$

The function g is chosen so that $u(x) = \frac{\log(1+x\eta)\log(1+(1-x)\eta)}{\log(1+\eta)^2}$ is the solution to the problem. This function has boundary layers at both ends, and their sharpness is controlled by the magnitude of η.

PROBLEM 3 $\quad -(1/\nu + \nu(x-\mu)^2)u'' - (2\nu(x-\mu))u' = g \quad \text{in} \quad (0,1), \quad u(0) = 0, \quad u(1) = 0.$

The function g is chosen so that the exact solution to this problem is $u(x) = (1-x)(\arctan(\nu(x-\mu)) + \arctan(\nu\mu))$, which for large ν has an interior layer near μ. Problem 3 was taken from [5].

In **PlaceMap**, we used $tol = 10^{-2}$, $maxstep = 20$ and a grid size of $N = 32$ to construct w, for each problem, independently of the size of the discretization grid used to solve the problem. We test the performance of QSC on Problem 1 with mapping functions w_{q6} and w_{q7}, respectively, and compare it with the performance when the mapping function is $w_e(x) = ((1+\eta)^x - 1)/\eta$, which is the inverse of the exact

N	error	order	error	order	error	order	error	order	error	order	error	order
QSC	w_e				w_{q6}				w_{q7}			
	$\|\epsilon(x)\|_\infty$		$\|\epsilon(w_i)\|$		$\|\epsilon(x)\|_\infty$		$\|\epsilon(w_i)\|$		$\|\epsilon(x)\|_\infty$		$\|\epsilon(w_i)\|$	
32	1.62-3		1.64-3		1.70-3		1.69-3		6.32-3		6.32-3	
64	1.14-4	3.8	1.10-4	3.9	1.27-4	3.7	1.24-4	3.8	4.87-4	3.7	4.76-4	3.7
128	7.43-6	3.9	6.93-5	4.0	8.32-6	3.9	7.82-6	4.0	3.86-5	3.7	3.91-5	3.6
256	5.12-7	3.8	4.34-7	4.0	5.45-7	3.9	4.92-7	4.0	3.18-6	3.6	3.01-6	3.7
CSC	w_e				w_{c8}				w_{c9}			
	$\|\epsilon^3(x)\|_\infty$		$\|\epsilon^{3'}(s_i)\|$		$\|\epsilon^3(x)\|_\infty$		$\|\epsilon^{3'}(s_i)\|$		$\|\epsilon^3(x)\|_\infty$		$\|\epsilon^{3'}(s_i)\|$	
32	3.14-3		2.21+1		3.07-3		2.13+1		9.94-3		7.03+1	
64	2.10-4	3.9	1.43+0	3.9	2.04-4	3.9	1.39+0	3.9	5.67-4	4.1	4.45+0	4.0
128	1.34-6	4.0	9.39-2	3.9	1.30-5	4.0	9.32-2	3.9	3.53-5	4.0	3.57-1	3.6
256	8.49-7	4.0	6.12-3	3.9	8.27-7	4.0	6.17-3	3.9	2.22-6	4.0	2.56-2	3.8

Table 1: Observed errors and respective orders of convergence corresponding to Problem 1 with $\eta = 10,000$, solved by QSC and mapping functions w_e, w_{q6}, and w_{q7}, and by CSC and mapping functions w_e, w_{c8}, and w_{c9}.

solution u, and which, for this problem, is expected to produce good results. The adaptive method for w_{q6} and w_{q7} reached the tolerance in 7 and 6 iterations, respectively. Table 1 shows that the approximate QSC solutions arising from w_{q6}, w_{q7} and w_e have optimal global and local convergence for the points indicated. Similar results are shown for CSC and the mapping functions w_{c8}, w_{c9}, and w_e. The adaptive method for w_{c8} and w_{c9} reached the tolerance in 6 and 7 iterations, respectively. It should be noted that, for QSC, w_e produces errors very close to those of w_{q6}, and, for CSC, w_e produces errors very close to those of w_{c8}. Certainly, under realistic situations, it is not always possible to construct an analytic formula for a mapping function such as w_e. It should also be noted that the errors from the CSC method are, in general, slightly lower than those from the QSC method, something which is expected. In a few cases (see, for example, some errors in Table 1), the QSC errors are smaller.

We next present results from QSC and CSC applied to Problem 2. The adaptive method **PlaceMap** applied to QSC with either w_{q5} or w_{q6} required 9 iterations to reach the tolerance, while when applied to CSC with either w_{c7} or w_{c8} required 7 iterations to reach the tolerance. Table 2 shows results from the application of QSC and CSC to this problem. In the table, $\sigma_{ij} \equiv w(\delta_{ij})$, $j = 1, 2$, $i = 1, \ldots, N$, where $\delta_{ij} \equiv x_i - \lambda_j h$; $j = 1, 2$, $i = 1, \ldots, N$, and $\lambda_1 \equiv (3 - \sqrt{3})/6$ and $\lambda_2 \equiv (3 + \sqrt{3})/6$. Notice that the approximate QSC solution arising from w_{q5} has less derivative error than the one arising from w_{q6}, while the opposite is true if we look at the function error. Also, the approximate CSC solution arising from w_{c7} has less derivative error than the one arising from w_{c8}, while the opposite is true if we look at the function error. This is expected from the definitions of w_{q5}, w_{q6}, w_{c7} and w_{c8}.

We next consider Problem 3 and present the results in Table 3. The adaptive method **PlaceMap** applied to QSC with mapping functions w_{q5} and w_{q6} did not reach the tolerance in 20 iterations, but the minimum drift was obtained in 5 and 11 iterations, respectively. The adaptive method applied to w_{q7} and w_{q2} required 5 and 11 iterations, respectively to reach the tolerance. **PlaceMap** applied to CSC with w_{c7},

N	error	order	error	order	error	order	error	order
QSC	w_{q5}				w_{q6}			
	$\lvert\epsilon(w_i)\rvert$		$\lvert\epsilon'(\sigma_{ij})\rvert$		$\lvert\epsilon(w_i)\rvert$		$\lvert\epsilon'(\sigma_{ij})\rvert$	
32	6.13-2		1.50-0		4.28-3		1.22+1	
64	4.38-3	3.8	3.50-1	2.1	4.61-4	3.2	4.90+0	1.3
128	2.74-4	4.0	4.39-2	3.0	3.71-5	3.6	9.80-1	2.3
256	1.79-5	3.9	5.97-3	2.9	2.88-6	3.7	1.82-1	2.4
CSC	w_{c7}				w_{c8}			
	$\lvert\epsilon^3(s_i)\rvert$		$\lvert\epsilon^{3'}(s_i)\rvert$		$\lvert\epsilon^3(s_i)\rvert$		$\lvert\epsilon^{3'}(s_i)\rvert$	
32	1.68-2		1.16+1		8.37-3		5.27+1	
64	8.34-4	4.3	6.01-1	4.3	5.64-4	3.9	4.51-0	3.6
128	6.05-5	3.8	2.07-2	4.9	4.00-5	3.8	3.27-1	3.8
256	3.92-6	3.9	1.15-3	3.9	2.67-6	3.9	2.60-2	3.7

Table 2: Observed errors and respective orders of convergence corresponding to Problem 2 with $\eta = 10,000$, solved by QSC and mapping functions w_{q5} and w_{q6}, and by CSC and mapping functions w_{c7} and w_{c8}.

w_{c8}, w_{c9} and w_{c2} required 19, 19, 6 and 6 iterations, respectively, to reach the tolerance.

We now present numerical results from (the mapping-free) CSC integrated with the adaptive procedure **AdaptSolve1** of Section 3 and from HPCC as implemented in COLSYS on Problems 1, 2 and 3, as well as two singular perturbation problems taken from [3]. Results on more problems are found in [11]. Problems 4 and 5 have a sharp interior layer when ε is close to zero. In the experiments, we set $\varepsilon = 10^{-4}$.

PROBLEM 4 $\varepsilon u'' + 2xu' = g \quad \text{in} \quad (-1,1), \quad u(-1) = -1, \;\; u(1) = 1.$

The exact solution to this problem is $u(x) = \text{erf}(x/\sqrt{\varepsilon})$.

PROBLEM 5 $\varepsilon u'' + xu' = g \quad \text{in} \quad (-1,1), \quad u(-1) = -2, \;\; u(1) = 0.$

The exact solution to this problem is $u(x) = \cos(\pi x) + \text{erf}(x/\sqrt{2\varepsilon})/\text{erf}(1/\sqrt{2\varepsilon})$.

For both CSC and HPCC (COLSYS), the tolerance is first set to $TOL = 10^{-6}$, the grading function is ξ_{c8}, and the initial grid has 32 uniform subintervals. Table 4 shows the CSC and HPCC sequences of grid sizes the adaptive algorithms generated. When the grid size remains the same, the method attempts to improve error equidistribution. Note that the error estimator in COLSYS is applied only when the grid is doubled, while in CSC, it is also applied when a new redistribution takes place after doubling.

For all problems and methods the tolerance is successfully reached, and the actual error obtained by CSC and HPCC are about compatible. Taking into account that HPCC uses two Gaussian points per subinterval, on Problems 3 and 4 CSC requires the same number of collocation points as HPCC, and gives about the same error as HPCC; on Problems 2 and 5 CSC requires less collocation points than HPCC, for about the same error; and on Problem 1 CSC requires more collocation points. Overall, CSC is a competitive method with respect to both the number of collocation points and the reliability of the error estimator.

N	error	order	error	order	error	order	error	order
QSC	w_{q5}		w_{q6}		w_{q7}		w_{q2}	
32	2.43-3		7.14-4		1.19-3		1.27-1	
64	2.84-4	3.1	4.82-5	3.9	1.12-4	3.4	5.78-3	4.5
128	1.77-5	4.0	3.03-6	4.0	7.09-6	4.0	4.63-4	3.6
256	1.12-6	4.0	1.91-7	4.0	4.43-7	4.0	3.08-5	3.9
CSC	w_{c7}		w_{c8}		w_{c9}		w_{c2}	
32	1.79-3		1.12-3		5.23-4		2.23-1	
64	6.37-5	4.8	2.87-5	5.1	4.09-5	3.7	7.49-3	4.9
128	3.12-6	4.4	1.28-6	4.5	2.21-6	4.2	4.63-4	4.0
256	1.75-7	4.2	7.06-8	4.2	1.24-7	4.2	2.80-5	4.0

Table 3: Observed grid point errors and respective orders of convergence corresponding to Problem 3 with $\mu = .5$ and $\nu = 100$, solved by QSC and mapping functions w_{q5}, w_{q6}, w_{q7} and w_{q2}, and by CSC and mapping functions w_{c7}, w_{c8}, w_{c9} and w_{c2}.

In Figure 1, we plot the exact solution and the non-uniform grids generated by CSC and HPCC for Problems 2, 4 and 5. The non-uniform grid points (y-coordinates) are plotted versus the respective uniform grid points (x-coordinates). This type of plot, gives a visualization of the mappings w generated by each of the two methods. (We emphasize that the computation of the mappings is not required by CSC and **AdaptSolve1**, but we only compute them in order to visualize them by the plot.) In regions where the graph of w is flat, the non-uniform grid points are very dense (i.e. the uniform points span a large region, and the non-uniform points a small one), while when w rises sharply, the non-uniform grid points are very sparse. We note that in some regions of the domains the mappings of the two methods may differ in a visible way, but in the layer regions, the differences are invisible. In the same figure, we show the location of selected CSC and HPCC grid points. Because the number of points is large, we select to show one grid point for every two of them. It is worth pointing out that, when the solution to some problem exhibits certain symmetry (e.g. Problems 2, 4), we expect the mapping (and the distribution of points) to reflect that symmetry to some extent. It turns out that COLSYS deviates visibly from the symmetric mapping in Problem 2 and even more visibly in Problem 4. We believe that this may be attributed partly to the too few redistributions that COLSYS applies.

In Table 5, we present brief results on Problems 1 to 5 for more values of the tolerance TOL. The final grid sizes N to reach the estimated errors and the respective actual errors for CSC and HPCC are shown. For CSC, there is one case where the tolerance is missed by a little (Problem 3, tolerance 10^{-8}), and for HPCC, there are two such cases (Problem 4, tolerances 10^{-5} and 10^{-7}). There are also few discrepancies in both methods, in that the grid obtained for a tougher tolerance may be coarser than for a less tough tolerance. However, both methods perform reasonably well. There are about five cases where CSC uses fewer collocation points than HPCC and about four cases where HPCC uses fewer collocation points.

In Table 6, we present results from QSC integrated with algorithms **AdaptSolve1** and **AdaptSolve2**. We emphasize that both algorithms, when implemented with QSC, require the construction of a mapping function for each new mesh, including the last (largest) one. The difference between the two algorithms is

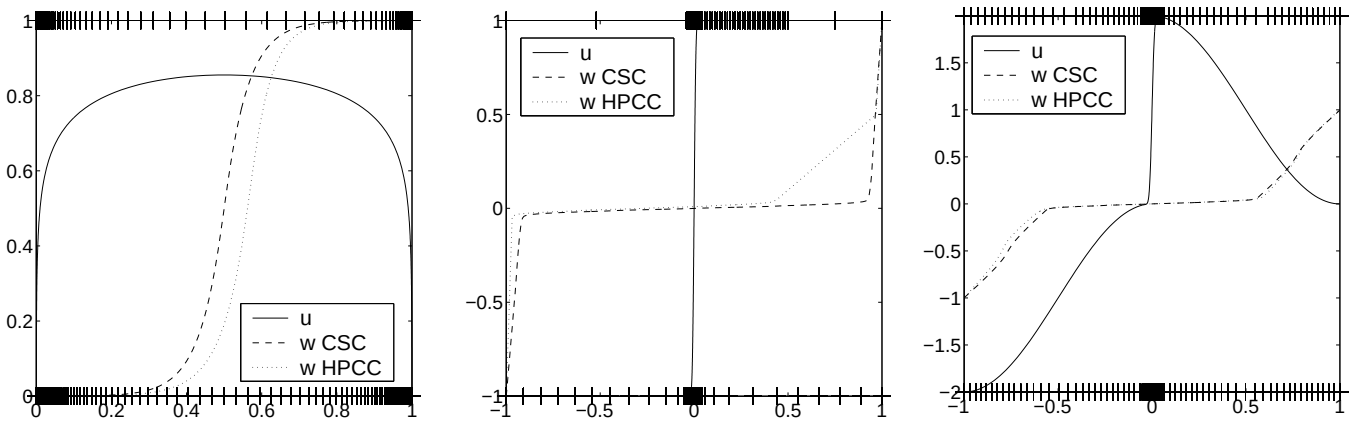

Figure 1: Exact solutions u and non-uniform grids by CSC and HPCC (COLSYS) for Problems 2, 4 and 5 (from left to right). The location of every second CSC and HPCC grid point is shown by | along the bottom and top axes, respectively.

that, in **AdaptSolve1** the mapping is used to find only the location of the collocation points ("midpoints"), while in **AdaptSolve2** the mapping is used to find the location of the collocation points, as well as the location of the grid points and midpoints of the double size mesh when doubling. From the experiments, it seems that **AdaptSolve2** for QSC is slightly preferable to **AdaptSolve1**, in that it requires a smaller grid size to reach the tolerance, for some of the cases considered. (A similar comparison of **AdaptSolve1** and **AdaptSolve2** for CSC did not reveal any significant differences.) Both algorithms miss the tolerance by a little bit in few cases, but both algorithms perform satisfactorily. Comparing QSC and CSC, CSC requires a smaller grid size than QSC for several cases (especially for tough tolerances), while there are few cases among those considered where the opposite happens.

In Table 7, we present results from QSC and CSC integrated with algorithm **AdaptSolve3**. This algorithm turns out to be very effective for the cases considered. In several cases, both QSC and CSC with **AdaptSolve3** require smaller grid sizes than the respective methods with **AdaptSolve1** or **AdaptSolve2**. We emphasize that **AdaptSolve3** generates a mapping at a relatively small, but problem and tolerance dependent grid size $\hat{N}$. This extra computation is not substantial compared to the total, therefore **AdaptSolve3** is a competitive alternative to **AdaptSolve1** or **AdaptSolve2**. However, we note that **AdaptSolve3** relies heavily on extrapolation, which, in turn, relies on whether the computed mapping function properly captured the behaviour of the solution. Clearly, there is no theory that guarantees that this is always the case, but, similarly, no theory guarantees that **AdaptSolve1** always computes a solution within the given tolerance. In general, all methods considered performed satisfactorily for the cases considered.

CSC - **AdaptSolve1**									
Probl	N and est. error								act. error
1	32	32	32	64	36	72	36	72	
				2.8-3		1.0-2		6.8-3	
	36	72	144	72	144	144	288		
		2.7-2	3.4-2		1.6-5	1.6-5	8.5-7		2.8-7
2	32	32	32	64	62	124	109	218	
				2.4-2		1.3+1		4.1+0	
	109	218	109	218	218	436			
		3.7-3		1.1-5	1.0-5	6.9-7			2.3-7
3	32	32	32	64	128	128	256		
				7.1-5	4.2-6	5.8-6	2.8-7		6.1-7
4	32	64	128	256	128	256	128	256	
		6.2-2	3.8-2	7.3-3		1.1-2		5.1-3	
	512	256	512						
	3.9-4		1.8-7						5.6-8
5	32	64	128	128	64	128	128	64	
		5.4-2	2.0-2	1.9-2		2.5-2	2.2-2		
	128	256	128	256	128	256			
	3.0-2	1.4-2		8.7-4		2.7-7			5.5-7

HPCC-COLSYS									
Probl	N and est. error								act. error
1	32	32	32	64	51	102	51	102	
				9.5-4		2.0-6		1.5-7	1.8-7
2	32	32	32	64	64	64	128	256	
				2.3-3			5.8-6	5.0-7	1.1-7
3	32	32	32	64	128				
				8.1-6	6.1-7				5.3-7
4	32	64	64	64	128	64	128		
		8.6-2			1.2-5		2.5-6		
	64	128	256						
		1.4-5	1.8-7						1.3-7
5	32	32	32	64	62	124	248		
				8.0-4		3.7-6	2.9-7		2.6-7

Table 4: Sequences of grid sizes N, estimated errors, and actual errors for CSC and HPCC (COLSYS) with a tolerance of 10^{-6} on the indicated problems. The estimated error is given in the form $x.y \pm z$ below the respective grid size. For some problems, the sequence of grid sizes continues over in a second line.

	TOL	1.0-4	1.0-5	1.0-6	1.0-7	1.0-8
CSC - **AdaptSolve1**						
Probl	N	128	256	288	512	1024
1	act.err.	7.3-6	4.6-7	2.8-7	8.1-8	5.1-9
	est.err.	2.2-5	1.4-6	8.5-7	8.1-8	5.1-9
Probl	N	128	256	436	864	2048
2	act.err.	8.6-5	5.4-6	2.3-7	4.1-8	6.6-10
	est.err.	6.9-5	5.4-6	6.9-7	4.1-8	1.4-9
Probl	N	64	128	256	512	512
3	act.err.	3.5-5	2.9-6	6.1-7	1.5-8	1.2-8
	est.err.	6.7-5	3.2-6	2.8-7	1.3-8	1.0-8
Probl	N	256	512	512	394	1024
4	act.err.	1.2-6	9.4-8	5.6-8	2.9-8	3.0-10
	est.err.	7.1-7	3.9-8	1.8-7	4.1-8	3.3-10
Probl	N	146	256	256	500	1024
5	act.err.	2.4-6	3.4-7	5.5-7	2.1-8	1.2-9
	est.err.	1.1-4	6.8-7	2.7-7	1.3-8	8.7-10
HPCC-COLSYS						
Probl	N	64	116	102	176	256
1	act.err.	1.8-6	1.9-7	1.8-7	1.9-8	4.1-9
	est.err.	9.2-6	1.2-6	1.5-7	1.7-8	3.8-9
Probl	N	108	100	256	512	564
2	act.err.	1.3-5	5.3-6	1.1-7	7.2-9	4.2-9
	est.err.	2.7-5	6.6-6	5.0-7	3.5-8	6.0-9
Probl	N	36	64	128	256	512
3	act.err.	5.9-5	2.4-6	5.3-7	2.7-8	2.1-9
	est.err.	5.6-5	3.9-6	6.1-7	4.1-8	2.7-9
Probl	N	118	116	256	348	544
4	act.err.	2.4-7	1.3-5	1.3-7	2.9-7	2.7-9
	est.err.	1.1-6	3.8-6	1.8-7	7.8-8	1.4-8
Probl	N	64	140	248	512	1024
5	act.err.	2.6-5	1.9-6	2.6-7	1.3-8	8.1-10
	est.err.	4.4-5	2.1-6	2.9-7	2.5-8	1.5-9

Table 5: Grid sizes N, estimated errors, and actual errors for CSC and HPCC of COLSYS, with the indicated tolerances on the indicated problems.

	TOL	1.0-04	1.0-05	1.0-06	1.0-07	1.0-08
QSC - **AdaptSolve1**						
Probl	N	128	256	512	1024	1664
1	act.err.	7.1-5	2.4-6	1.4-7	9.5-9	1.7-9
	est.err.	8.3-5	4.3-6	2.8-7	2.1-8	3.9-9
Probl	N	128	256	512	1536	2976
2	act.err.	1.6-4	1.1-5	8.2-7	1.8-8	3.9-9
	est.err.	6.8-5	8.4-6	8.0-7	2.6-8	4.4-9
Probl	N	128	512	1024	2048	8192
3	act.err.	1.0-5	1.1-6	2.0-7	6.2-8	4.3-9
	est.err.	1.2-5	1.4-6	2.3-7	6.7-8	4.7-9
Probl	N	144	256	1024	1024	2048
4	act.err.	8.1-6	3.9-7	1.2-7	5.1-8	8.5-9
	est.err.	1.2-5	9.6-6	1.3-7	9.5-8	8.7-9
Probl	N	128	432	1184	4096	8192
5	act.err.	1.7-5	7.2-6	2.1-7	2.0-8	2.5-9
	est.err.	5.9-5	8.8-6	2.2-7	2.2-8	2.6-9
QSC - **AdaptSolve2**						
Probl	N	128	256	512	1024	1664
1	act.err.	6.6-5	2.0-6	1.2-7	7.7-9	1.0-9
	est.err.	3.9-5	2.0-6	1.3-7	9.3-9	1.4-9
Probl	N	128	256	512	1536	1488
2	act.err.	8.4-5	5.8-6	4.4-7	7.2-9	8.8-9
	est.err.	1.0-4	4.3-6	4.0-7	7.2-9	8.7-9
Probl	N	64	128	1024	2048	4096
3	act.err.	4.4-5	2.7-6	1.2-7	1.2-7	1.1-8
	est.err.	5.0-5	4.9-6	9.7-8	7.5-8	7.8-9
Probl	N	144	256	1024	1024	2048
4	act.err.	5.1-6	2.3-6	1.3-8	3.0-8	4.3-9
	est.err.	2.1-5	9.8-6	2.5-8	2.2-8	3.0-9
Probl	N	128	216	592	4096	8192
5	act.err.	1.2-5	7.5-6	9.4-7	1.5-8	1.1-8
	est.err.	2.8-5	6.9-6	7.6-7	1.1-8	6.6-9

Table 6: Grid sizes N, estimated errors, and actual errors for QSC with algorithms **AdaptSolve1** and **AdaptSolve2**, with the indicated tolerances on the indicated problems.

	TOL	1.0-4	1.0-5	1.0-6	1.0-7	1.0-8
QSC - **AdaptSolve3**						
Probl	$\hat{N}, N$	32, 81	64, 128	112, 224	200, 400	128, 820
1	act.err.	4.52-5	7.17-6	7.83-7	7.78-8	9.72-9
	est.err.	5.36-5	7.22-6	8.39-7	8.86-8	5.19-9
Probl	$\hat{N}, N$	32, 115	46, 220	104, 480	128, 910	156,2080
2	act.err.	6.46-5	1.25-6	4.56-7	6.59-8	7.36-9
	est.err.	4.83-5	4.97-6	4.63-7	4.18-8	3.94-9
Probl	$\hat{N}, N$	32, 64	40, 261	64, 518	128, 807	128,1434
3	act.err.	6.79-5	8.25-6	3.74-7	1.60-7	6.92-9
	est.err.	9.10-5	1.63-6	2.32-7	2.66-8	2.81-9
Probl	$\hat{N}, N$	58, 116	46, 114	70, 241	70, 428	70, 761
4	act.err.	2.28-6	9.79-7	6.18-7	7.12-8	1.00-8
	est.err.	5.48-6	1.45-5	3.38-7	3.58-8	3.60-9
Probl	$\hat{N}, N$	90, 180	146, 292	134, 805	128,1200	128,2134
5	act.err.	2.24-5	5.32-6	3.26-7	1.11-7	1.30-8
	est.err.	3.61-5	9.66-6	1.86-7	2.56-8	2.53-9
CSC - **AdaptSolve3**						
Probl	$\hat{N}, N$	32, 106	64, 185	64, 329	64, 584	64, 1039
1	act.err.	4.37-5	4.71-6	4.70-7	4.73-8	4.72-9
	est.err.	4.44-5	4.73-6	4.72-7	4.76-8	4.75-9
Probl	$\hat{N}, N$	64, 128	64, 264	64, 469	78, 834	96, 1481
2	act.err.	8.59-5	4.73-6	4.74-7	4.75-8	4.71-9
	est.err.	8.69-5	4.75-6	4.77-7	4.77-8	4.80-9
Probl	$\hat{N}, N$	32, 64	64, 128	64, 196	64, 347	64, 618
3	act.err.	5.42-5	3.10-6	5.71-7	5.86-8	5.84-9
	est.err.	4.36-5	2.60-6	5.63-7	5.71-8	5.68-9
Probl	$\hat{N}, N$	132, 264	132, 264	132, 264	214, 428	214, 515
4	act.err.	2.21-7	2.21-7	2.21-7	9.26-9	4.61-9
	est.err.	3.68-7	3.68-7	3.68-7	1.01-8	4.51-9
Probl	$\hat{N}, N$	128, 256	118, 236	124, 248	124, 386	146, 696
5	act.err.	2.18-7	2.68-7	2.38-7	4.07-8	3.61-9
	est.err.	4.91-7	2.98-7	2.81-7	4.01-8	3.60-9

Table 7: Grid sizes $\hat{N}$ (for which the mapping is computed) and N (for which the final solution is computed), estimated errors, and actual errors on N for QSC and CSC with algorithm **AdaptSolve3**, with the indicated tolerances on the indicated problems.

5 Outlook and Future Work

In this paper, we focused the discussion to the one-dimensional BVP but some of the ideas and methods considered are useful for higher order and multi-dimensional BVPs. In [11], adaptive spline collocation methods for two-dimensional BVPs are presented using two approaches, one that uses skipped rectangular grids (grids that refine the discretization in a tree-like hierarchical manner), and another one that uses moving mesh partial differential equations as in [10].

A discussion on comparing the efficiency of CSC and HPCC is a bit premature at this point, since efficiency issues become more critical when solving multi-dimensional problems, than when solving one-dimensional ones. However, we think it is worth making a few points. In most cases, HPCC of COLSYS requires fewer grid points than CSC to reach a certain error, but it has to collocate on $2N$ points, which results in a double size linear system to solve. Thus, it turns out that in most cases, HPCC requires more collocation points to reach the required tolerance. Moreover, the two-step CSC matrix is tridiagonal, while the HPCC matrix has 4 non-zero entries per row. Note that the two-step CSC matrix is solved twice, and if that is done by Gauss elimination, the LU factorization is performed once only. The one-step CSC matrix is pentadiagonal and again half-size as the HPCC matrix.

If we consider two-dimensional problems, the HPCC matrix is four times as large as the two-step CSC matrix, it has about twice the bandwidth of two-step CSC, and 16 nonzero entries per row, while two-step CSC has 9. The one-step CSC matrix has about the same bandwidth as HPCC, but is one quarter size. Moreover, the HPCC matrix requires pivoting, while the CSC matrix (as well as the QSC matrix) in most cases do not. Taking into account these facts, we expect that the asymptotic rate of computation time versus error of CSC will be better than that of HPCC.

Comparing QSC and CSC, the main advantage of CSC is that it requires only the (non-uniform) grid points, while QSC requires both the grid points and the midpoints. Thus CSC does not require the computation of a mapping function. This allows the efficient application of the adaptive technique **AdaptSolve1** of Section 3 which applies redistributions of the points at the large grids. However, the adaptive technique **AdaptSolve3**, which applies redistributions of the points at relatively small (or medium) grid sizes, seems to be a competitive alternative with both CSC and QSC. Moreover, as mentioned in [7], QSC may be advantageous for certain problems, since it does not apply the differential operator on the boundary (or any other grid) points, thus avoiding potential singularities.

Acknowledgements

The authors wish to thank the referees for the thorough reading of the paper and their helpful suggestions. This research was supported by NSERC (National Science and Engineering Research Council of Canada) and OGS (Ontario Graduate Scholarship).

References

[1] Jr. A. B. White. On selection of equidistributing meshes for two-point boundary-value problems. *SIAM J. Numer. Anal.*, 16(3):472–502, 1979.

[2] U. Ascher, J. Christiansen, and R. D. Russell. A collocation solver for mixed order systems of boundary value problems. *Math. Comp.*, 33(146):659–679, 1979.

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

[4] G. Bader and U. Ascher. A new basis implementation for a mixed order boundary value ODE solver. *SIAM J. Sci. Stat. Comp.*, 8(4):483–500, 1987.

[5] G. F. Carey and H. T. Dinh. Grading functions and mesh redistribution. *SIAM J. Numer. Anal.*, 22(5):1028–1040, 1985.

[6] M. A. Celia and W. G. Gray. *Numerical Methods for Differential Equations*. Prentice Hall, 1992.

[7] C. C. Christara and K. S. Ng. Optimal quadratic and cubic spline collocation on non-uniform partitions. To appear in *Computing*, 2005.

[8] C. de Boor. Good approximation by splines with variable knots II. *Lecture notes in Mathematics*, 363:12–20, 1973. Conference on Numerical Solution of Differential Equations.

[9] F. N. Fritsch and R. E. Carlson. Monotone piecewise cubic interpolation. *SIAM J. Numer. Anal.*, 17(2):238–246, 1980.

[10] W. Huang and R. D. Russell. Moving mesh strategy based on a gradient flow equation for two-dimensional problems. *SIAM J. Sci. Comput.*, 20:998–1015, 1999.

[11] K. S. Ng. *Spline Collocation on Adaptive Grids and Non-Rectangular Regions*. PhD thesis, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, 2005. http://www.cs.toronto.edu/pub/reports/na/ccc/ngkit-05-phd.ps.gz.