# Performance Analysis of Error-Control B-spline Gaussian Collocation Software for PDEs ☆

Jack Pew[1], Zhi Li[2], Connor Tannahill[1], Paul Muir[3], Graeme Fairweather[4]

**Abstract**

B-spline Gaussian collocation software has been widely used in the numerical solution of boundary value ordinary differential equations (BVODEs) and partial differential equations (PDEs) in one space dimension (1D) for many years. The software package, BACOL, developed over a decade ago, was one of the first 1D PDE packages to provide both temporal and spatial error control. A new package, BACOLI, improves upon the efficiency of BACOL through the use of new types of spatial error estimation and control. The complexity of the interactions among the component numerical algorithms used by these packages implies that extensive testing and analysis of the test results is an essential factor in their development. In this paper, we investigate the performance of the BACOL and BACOLI packages with respect to several important machine independent algorithmic measures and examine the effectiveness of the new error estimation and error control strategies. We also investigate the influence of the choice of the degree of the B-splines on the efficiency and reliability of the solvers. These results will provide new insights into how to improve BACOLI, lead to improvements in the Gaussian collocation BVODE solvers, COLSYS and COLNEW, and guide the further development of B-spline Gaussian collocation software with error control for 2D PDEs.

**Subject Classification:** 65L10, 65M20, 65M70
**Keywords:** Partial differential equations, collocation, B-Splines, error control, efficiency, reliability.

☆This work was supported by the Mathematics of Information Technology and Complex Systems Network, the Natural Sciences and Engineering Research Council of Canada and Saint Mary's University.

[1]Saint Mary's University, Halifax, NS, Canada, B3H 3C3
[2]Michigan State University, East Lansing, MI 48824, USA
[3]*Corresponding Author:* Mathematics and Computing Science, Saint Mary's University, Halifax, NS, Canada, B3H 3C3, *E-mail Address:* muir@smu.ca
[4]Mathematical Reviews, American Mathematical Society, Ann Arbor, MI 48103, USA

## 1. Introduction

For several decades, software packages implementing B-spline Gaussian collocation algorithms, often called orthogonal spline collocation (OSC) algorithms in the literature, have been widely used in the numerical solution of systems of boundary value ordinary differential equations (BVODEs) and 1D partial differential equations (PDEs). In these packages, the approximate solution is represented as a linear combination of B-spline basis functions [5] of a user-chosen degree, with unknown coefficients which are determined by requiring the approximate solution to satisfy, in addition to the boundary conditions, the differential equations at certain points (images of Gauss points) within each subinterval of a mesh that partitions the spatial domain.

An essential component of a high quality numerical software package is a framework that provides an error-controlled computation of the approximate solution. A package that implements adaptive error control returns an approximate numerical solution for which an associated error estimate satisfies a user-prescribed tolerance. This type of computation has two important advantages:

(i) the user can have reasonable confidence that the numerical solution has an error that is consistent with the requested tolerance, and

(ii) the cost of the computation will be consistent with the requested accuracy. Error-control software typically adapts the computation so that the amount of work performed is appropriate for the desired accuracy, using, for example, a spatial mesh with an appropriate number and distribution of points.

In this paper, we consider B-spline Gaussian collocation software packages for solving coupled systems of PDEs of the form

$$\mathbf{u}_t(x,t) = \mathbf{f}\left(x,t,\mathbf{u}(x,t),\mathbf{u}_x(x,t),\mathbf{u}_{xx}(x,t)\right), \qquad x \in I, \quad t \geq t_0, \tag{1}$$

with separated boundary conditions

$$\mathbf{b}_L\left(t,\mathbf{u}(a,t),\mathbf{u}_x(a,t)\right) = \mathbf{0}, \qquad \mathbf{b}_R\left(t,\mathbf{u}(b,t),\mathbf{u}_x(b,t)\right) = \mathbf{0}, \quad t \geq t_0, \tag{2}$$

and initial conditions

$$\mathbf{u}(x,t_0) = \mathbf{u}_0(x), \qquad x \in I, \tag{3}$$

where $I$ denotes the interval $(a,b)$.

In order to describe the spatial discretization of this problem using B-spline Gaussian collocation, we introduce the following notation. Let $\pi_h = \{x_i\}_{i=0}^{NINT}$ denote a partition of the interval $\overline{I} \equiv [a,b]$ with

$$a = x_0 < x_1 < \ldots < x_{NINT-1} < x_{NINT} = b,$$

and let

$$I_i = [x_{i-1}, x_i], \quad h_i = x_i - x_{i-1}, \quad i = 1, \ldots, NINT.$$

We define the finite dimensional space $\mathcal{M}_p$ by

$$\mathcal{M}_p = \{v_h \in C^1(I) : v_h|_{I_j} \in \mathcal{P}_p(I_j), \;\; j = 1, \ldots, NINT\},$$

where $\mathcal{P}_p(I_j)$ denotes the set of all polynomials of degree $\leq p$ on $I_j$. Note that

$$\dim \mathcal{M}_p = NINT(p-1) + 2 \equiv NC_p.$$

As the name of the method suggests, it is customary to choose as a basis for $\mathcal{M}_p$ the space of B-splines, $\{B_{p,i}(x)\}_{i=1}^{NC_p}$ of degree $p \geq 3$.

Let $\{\rho_j\}_{j=1}^{p-1}$ denote the nodes of the $(p-1)$-point Gauss-Legendre quadrature rule on the interval $[0, 1]$. Let

$$\mathcal{G} = \{\xi_r\}_{r=1}^{NINT(p-1)}$$

be the set of Gauss points in $I$, where

$$\xi_\ell = x_{i-1} + h_i \rho_k, \quad \ell = (i-1)(p-1) + j, \quad j = 1, 2, ...p-1, \quad i = 1, ..., NINT.$$

These are the collocation points.

With this notation, we express the collocation solution, $\mathbf{U}(x, t)$, in the form,

$$\mathbf{U}(x, t) = \sum_{i=1}^{NC_p} \mathbf{y}_{p,i}(t) B_{p,i}(x), \tag{4}$$

where $\mathbf{y}_{p,i}(t)$ is an unknown time dependent (vector) coefficient of $B_{p,i}(x)$. The coefficients, $\mathbf{y}_{p,i}(t)$, are obtained by requiring that $\mathbf{U}(x, t)$ satisfy the PDE at the $p-1$ Gauss points within each spatial mesh subinterval, $I_j, \; i = 1, \ldots, NINT$, and that $\mathbf{U}(x, t)$ satisfy the boundary conditions. The former set of conditions, known as the collocation conditions, give equations of the form,

$$\mathbf{U}_t(\xi_l, t) = \mathbf{f}\left(\xi_l, t, \mathbf{U}(\xi_l, t), \mathbf{U}_x(\xi_l, t), \mathbf{U}_{xx}(\xi_l, t)\right), \tag{5}$$

for $l = 2, \ldots, NC_p - 1$. The latter pair of conditions lead to equations having the form,

$$\mathbf{b}_L\left(t, \mathbf{U}(a, t), \mathbf{U}_x(a, t)\right) = \mathbf{0}, \qquad \mathbf{b}_R\left(t, \mathbf{U}(b, t), \mathbf{U}_x(b, t)\right) = \mathbf{0}. \tag{6}$$

The ODEs (5), $l = 2, \ldots, NC_p - 1$, together with the equations (6), represent a system of index-1 Differential-Algebraic Equations (DAEs).

For systems of BVODEs, the COLSYS package [3] was the first software package based on B-spline Gaussian collocation which incorporated a mechanism for error control. This package has been used extensively for many decades and has been implemented in a number of numerical software libraries, for example, [14], and in several problem-solving environments such as Scilab (bvode)

[21] and Python (scikits.bvp1lg) [23]. (In some cases, the implementation is based on COLNEW [4], a modified version of COLSYS, in which the primary difference is the replacement of the B-spline basis with a monomial basis, the Gaussian collocation and error estimation and control algorithms remaining largely unchanged).

For 1D parabolic PDEs of the form (1)–(2), the B-spline Gaussian collocation packages PDECOL [15] and EPDCOL [12] were the earliest packages of this family to provide any form of error control. However they provide control only of the temporal error and not the spatial error. These packages require the user to provide the time derivatives of the boundary equations. These ODEs, together with the ODEs (5), $l = 2, \ldots, NC_p - 1$, are solved by an ODE solver which provides the temporal error control. PDECOL employs a banded solver to treat the linear systems arising from the B-spline collocation spatial discretization. However these systems have an almost block diagonal (ABD) structure - see, for example, [7]. EPDCOL, a modification of PDECOL that replaces the banded solver with the ABD linear system solver, COLROW [6], in order to avoid fill-in and the associated computational and memory overheads, was shown in [12] to be about twice as fast as PDECOL.

Subsequently, the packages, BACOL [25, 27] and BACOLR [24] were developed. These also implement B-spline Gaussian collocation for the spatial discretization but provide both temporal and spatial error control. The temporal error-controlled solution of the DAE system, (5), $l = 2, \ldots, NC_p - 1$, (6), to obtain the time-dependent B-spline coefficients, is performed in BACOL using the DASSL [4] package, which is based on Backward Differentiation Formulas (BDFs), and in BACOLR using the RADAU5 package [9], which is based on a fifth order implicit Runge-Kutta method of Radau IIA type. In a comparison with several other packages for 1D parabolic PDEs, for example, EPDCOL, MOVCOL [10], and HPNEW [16], BACOL was shown in [26] to provide superior performance, especially for problems with solutions exhibiting sharp moving layers and for sharp tolerances. In [24], numerical comparisons of BACOL and BACOLR show that the two codes perform similarly on several standard test problems and that BACOLR has much superior performance on problems for which the stability of the higher order BDFs is an issue. The stability regions of the higher order BDFs do not include the imaginary axis and thus problems that lead to DAEs having Jacobians with eigenvalues near the imaginary axis, for example, Schrödinger type problems, cannot be treated using the higher order BDFs. The paper [24] shows that BACOL fails on problems of this type unless DASSL is restricted to using only lower order BDFs, in which case the efficiency of the computation is substantially degraded.

BACOL obtains its spatial error estimate by computing a second approximate solution (based on B-splines of degree $p+1$), essentially doubling the cost of the computation. The recently released package, BACOLI [18] improves on the efficiency of the spatial error estimation scheme employed by BACOL by introducing two new interpolation-based schemes, each of which is coupled with a different spatial error control mode. Since the interpolant can be used to obtain the spatial error estimate, the second approximate solution is not computed. In

[18], BACOLI is shown to be approximately twice as efficient as BACOL. We discuss the BACOL and BACOLI packages in more detail in the next section.

We are currently developing a new version of BACOLR, called BACOLRI, that employs the new spatial error estimation and control schemes that have been implemented in BACOLI. Since we will compare BACOLRI with BACOLR and BACOLI in a subsequent paper, we do not consider BACOLR within this paper.

A common feature of the COLSYS and COLNEW packages and the 1D PDE packages mentioned earlier is that each implements a *family* of Gaussian collocation methods over a range of values of $p$. For example, for BACOLI, the allowable range of $p$ values is 4 to 11. While the user is required to select a value for $p$, the packages provide little or no guidance regarding how it should be chosen, and, to our knowledge, there has been little investigation of the impact of $p$ on the overall code performance, particularly for the PDE case. *A primary goal of this paper is therefore to investigate the role that the degree $p$ of the B-spline basis plays in the efficiency and reliability of the computation.*

The development of B-spline Gaussian collocation software for 2D parabolic PDEs in a rectangular spatial domain partitioned by a rectangular grid has been considered in [13]. Current work on this project is focused on the development of efficient spatial error estimation techniques involving the extension of techniques we have developed for the 1D case, and the development of spatial mesh adaptation (for spatial error control) based on ideas from moving mesh methods - see, for example, [11]. This work depends heavily on the approaches employed in the 1D case. *Another primary goal of this paper, therefore, is an investigation of the performance of 1D PDE solvers in order to apply the results in the further development of B-spline Gaussian collocation software for the 2D case.*

Performance analysis of scientific software for differential equations has a long history. See, for example, [22] and references therein. While efficient, stable and robust numerical methods are the foundation upon which all high quality numerical software packages are developed, the sophistication of such packages and the complexity of the interaction among the component algorithms of the packages implies that extensive numerical testing together with subsequent analysis of the results of the numerical experiments is an essential factor in the development of such software. For example, the software package BACOLI, considered in this paper, includes implementations of a number of complex numerical algorithms such as adaptive error-controlled time-stepping and method order selection for the DAEs and spatial mesh refinement for the discretization of the spatial domain of the PDEs. In addition, the code also has available two spatial error control schemes and a family of collocation methods. *The ways in which these algorithms interact to affect the efficiency and reliability of the software must be assessed through detailed numerical experimentation and performance analysis.*

This paper is organized as follows. In Section 2, we provide an overview of several features of BACOL and BACOLI. This is followed by a brief description of the spatial error estimation scheme employed by BACOL and the two

interpolation-based spatial error estimation/error control schemes implemented in BACOLI. Section 3 provides selected results from a set of numerical experiments reported in [20] in which the performance of the BACOL and BACOLI packages is investigated. These packages are compared with respect to several important machine independent measures that provide insight into the performance of the algorithms implemented in these packages. In addition, results are presented that compare the work required vs. accuracy achieved by the packages. Section 3 also provides results which examine the role that the degree of the B-spline basis plays in both the efficiency and reliability of the solvers. We conclude in Section 4 with a discussion of the results and present suggestions for future research.

## 2. The software packages BACOL and BACOLI

As has already been mentioned, BACOL solves the DAEs arising from the collocation equations (5) and boundary conditions (6) using the DAE solver DASSL, which provides a temporal error-controlled computation of the B-spline coefficients. After each accepted time step taken by DASSL, BACOL computes an estimate of the spatial error in the collocation approximation and checks if it satisfies the user-specified tolerance. If the tolerance is not satisfied, the numerical solution in question is rejected, and a spatial remeshing, based on the principle of equidistributing the spatial error estimate, is performed. Both the location and the number of spatial mesh points may be changed during the remeshing in order to adapt to the magnitude (with respect to the user-specified tolerance) and distribution of the spatial error estimate over the spatial domain. See [27] for further details. Based on the new spatial mesh, the computation of the solution on the current time step is repeated using DASSL in what is referred to as a "warm start" mode, meaning that the current time step and method (that is, a BDF of a given order) are used. If, after several failed attempts to obtain a spatial mesh such that the corresponding numerical solution has a spatial error estimate that satisfies the desired tolerance, BACOL restarts the time step using DASSL in "cold start" mode meaning that it restarts with a very small time step and with the BDF of order one. Cold starts are computationally expensive since it can take a substantial number of time steps and method order increases before DASSL is able to return to the larger step size and the higher order method that it was using before the cold start was enforced.

As mentioned earlier, the BACOL spatial error estimate is determined by computing a second approximate solution, $\bar{\mathbf{U}}(x,t)$, on the same spatial mesh and at the same time $t$, but based on B-splines of degree $p+1$. A scaled difference of $\mathbf{U}(x,t)$ and $\bar{\mathbf{U}}(x,t)$ is then computed to provide a spatial error estimate for $\mathbf{U}(x,t)$. It has the form,

$$\sqrt{\int_a^b \left( \frac{U_s(x,t) - \bar{U}_s(x,t)}{ATOL_s + RTOL_s |U_s(x,t)|} \right)^2 dx}, \quad s = 1, \ldots, NPDE,$$

where $U_s(x,t)$ is the $s$th component of $\mathbf{U}(x,t)$, $\bar{U}_s(x,t)$ is the $s$th component of $\bar{\mathbf{U}}(x,t)$, $ATOL_s$ and $RTOL_s$ are the user-provided absolute and relative tolerances for the $s$th component of the spatial error estimate, and $NPDE$ is the number of PDEs.

The computation of $\bar{\mathbf{U}}(x,t)$ *essentially doubles the overall cost.* This inefficiency is addressed in the BACOLI package which replaces the expensive computation of $\bar{\mathbf{U}}(x,t)$ with a more efficient approach involving the computation of an interpolant based only on $\mathbf{U}(x,t)$. Two types of piecewise polynomial interpolants have been developed. One, the *SuperConvergent Interpolant (SCI)* [1], is based on interpolating values of $\mathbf{U}(x,t)$ at the mesh points and at certain other points (within and immediately adjacent to each subinterval) where it is superconvergent in space; that is, where the rate of convergence of the spatial error at these points is at least one order higher than at an arbitrary point in the spatial domain. A sufficient number of such points is chosen so that the interpolation error is dominated by the error of the superconvergent values and thus the resultant interpolant is of one order of spatial accuracy higher than $\mathbf{U}(x,t)$. See [1] for further details. A scaled difference of the SCI and $\mathbf{U}(x,t)$,

$$\sqrt{\int_a^b \left( \frac{U_s(x,t) - \hat{U}_s(x,t)}{ATOL_s + RTOL_s|U_s(x,t)|} \right)^2 dx}, \quad s = 1, \ldots, NPDE,$$

where $\hat{U}_s(x,t)$ is the $s$th component of the SCI, is then computed to provide a spatial error estimate for $\mathbf{U}(x,t)$.

The second type of interpolant, the *Lower Order Interpolant (LOI)* [2], is based on interpolating values of $\mathbf{U}(x,t)$ at a set of points on each spatial subinterval such that the interpolant has an interpolation error that is asymptotically equivalent to the error in a collocation solution of one order lower. In this case, the number of interpolation points is chosen so that the interpolation error of the LOI dominates the error associated with the $\mathbf{U}(x,t)$ values. See [2] for further details. Then a scaled difference of the LOI and $\mathbf{U}(x,t)$,

$$\sqrt{\int_a^b \left( \frac{U_s(x,t) - \tilde{U}_s(x,t)}{ATOL_s + RTOL_s|U_s(x,t)|} \right)^2 dx}, \quad s = 1, \ldots, NPDE,$$

where $\tilde{U}_s(x,t)$ is the $s$th component of the LOI, provides a spatial error estimate for a collocation solution that is of one lower order than $\mathbf{U}(x,t)$. This type of error control, known as *Local Extrapolation (LE) error control*, is similar to what is done in the context of Runge-Kutta formula pairs for the numerical solution of initial value ODEs [8]. BACOLI offers an option for the use of either the SCI or the LOI scheme, coupled with its associated error control mode.

In BACOL, $\mathbf{U}(x,t)$ is the primary solution returned to the user and $\bar{\mathbf{U}}(x,t)$ is computed only to obtain a spatial error estimate for $\mathbf{U}(x,t)$. Since the spatial error estimate is for $\mathbf{U}(x,t)$, we refer to this as *Standard (ST) Error Control*, and we refer to BACOL when running in this error control mode as **BAC/ST**. Since

the SCI scheme also computes a spatial error estimate for $\mathbf{U}(x,t)$, BACOLI, when using the SCI option, is also using ST error control, and we refer to BACOLI when running in this error control mode as **SCI/ST**. With a simple modification, it would be possible to have BACOL return $\bar{\mathbf{U}}(x,t)$. However the error control would continue be based on the error estimate for $\mathbf{U}(x,t)$ and would thus be an example of LE error control. We refer to BACOL, when running in this error control mode, as **BAC/LE**. Since BACOLI, when using the LOI option, also returns $\mathbf{U}(x,t)$ with the error control based on a spatial error estimate for a collocation solution of one lower order, this is also LE error control. We refer to BACOLI when running in this mode as **LOI/LE**. See [18] for further details.

## 3. Numerical Results

In this section, we present a subset of the numerical results from [20] which investigate the performance of the four codes, BAC/ST, BAC/LE, SCI/ST and LOI/LE. We consider *machine independent* measures of efficiency, error vs. execution time comparisons, and a reliability measure comparing error achieved vs. tolerance requested. We also examine the effects on performance of the error estimation schemes, the error control modes, and the choice of the degree of the B-spline basis.

The computations were performed on a system with two Intel(R) Xeon(R) CPU E5420 @ 2.50GHz processors. The operating system was Ubuntu 12.04.5 LTS, and the Fortran compiler was GNU Fortran (Ubuntu/Linaro 4.6.3-lubuntu5) 4.6.3.

### *3.1. Test Problems*

In [20] nine test problems are considered. Here we consider the following three of these which are of particular interest because they have solutions that feature moving spatial layer regions. This places demands on the spatial error estimation and spatial mesh adaptivity algorithms to adequately detect and track the moving layers as the solution evolves in time. See, for example, [27], page 251, Figure 2, which shows this layer-tracking capability.

- **OLBE:** The One Layer Burgers Equation:

$$u_t = \epsilon u_{xx} - u u_x, \quad x \in (0,1), \quad t \in (0,1], \tag{7}$$

with the initial condition and boundary conditions chosen so that the exact solution is

$$u(x,t) = \frac{1}{2} - \frac{1}{2}\tanh\left(\frac{x - \frac{t}{2} - \frac{1}{4}}{4\epsilon}\right), \tag{8}$$

where $\epsilon$ is a problem-dependent parameter. We set $\epsilon = 10^{-4}$. The solution has a sharp layer region at $x \approx 0.25$ when $t = 0$. As $t$ increases from 0 to 1, the layer moves to the right and is located at $x \approx 0.75$ when $t = 1$.

- **TLBE:** The Two Layer Burgers Equation:

  We again consider the PDE (7) but with the initial condition and boundary conditions chosen so that the exact solution is

  $$u(x,t) = \frac{0.1e^{-A} + 0.5e^{-B} + e^{-C}}{e^{-A} + e^{-B} + e^{-C}},$$

  where

  $$A = \frac{0.05}{\epsilon}(x - 0.5 + 4.95t), \quad B = \frac{0.25}{\epsilon}(x - 0.5 + 0.75t), \quad C = \frac{0.5}{\epsilon}(x - 0.375).$$

  When $t = 0$, the solution has two sharp layers at $x \approx 0.25$ and $x \approx 0.5$. As $t$ increases, these layers move to the right and merge, forming a single layer at $x \approx 0.7$ when $t \approx 0.5$. As time increases further, the single layer continues to move to the right, and is located at $x \approx 0.9$ when $t = 1$. We again choose $\epsilon = 10^{-4}$.

- **TLBEx6:** This test problem is a system of PDEs consisting of six copies of **TLBE**, with $\epsilon = 10^{-4}$.

*3.2. Machine Independent Efficiency Measures*

In this subsection, we compare BAC/ST, BAC/LE, SCI/ST, and LOI/LE with respect to several *machine independent* measures of the algorithms employed in the codes that can contribute significantly to their overall performance. These machine independent measures provide an important complement to standard machine dependent timing results. Many additional insights regarding code performance can be obtained by considering such performance measures.

The machine independent measures that we consider are:

- the number of subintervals in the spatial mesh at the final time (**Final NINT**),
- the total number of accepted time steps (**Accepted Time Steps**),
- the total number of spatial remeshings (**Remeshings**), and
- the total number of cold starts (**Cold Starts**).

As mentioned earlier, BACOL and BACOLI employ COLROW for the efficient solution of the ABD linear systems that arise during the computations. Within COLROW, the CRDCMP routine performs factorizations of the ABD coefficient matrices, while the CRSLVE routine solves the factored ABD systems. Thus we also consider:

- the total number of ABD matrix factorizations (**Calls to CRDCMP**),
- the total number of solves of ABD linear systems (**Calls to CRSLVE**).

Here we present machine independent results from [20] for **TLBEx6** with $\epsilon = 10^{-4}$. We choose $p = 5, 7, 9$, and tolerances, $tol = 10^{-4}, 10^{-6}, 10^{-8}$. We set $ATOL_s = RTOL_s = tol, s = 1, \ldots, NPDE$. *For all tests, we choose p so that we are comparing computations that return collocation solutions based on polynomials of the same degree on each subinterval.* Table 1 gives machine independent measures for the four codes. In Table 2, we provide machine dependent timings that include the cases considered in Table 1.

| *tol* | $10^{-4}$ | $10^{-6}$ | $10^{-8}$ |
|---|---|---|---|
| *p* | 5 | | |
| BACOL/ST | 14, 9899, 521<br>(7)[2654, 33308] | 17, 23551, 976<br>(5)[7014, 76450] | 28, 46825, 1064<br>(3)[9346, 131442] |
| BACOL/LE | 14, 9346, 613<br>(12)[2896, 32612] | 19, 21873, 943<br>(1)[4308, 67264] | 44, 47092, 943<br>(0)[6616, 128772] |
| BACOLI/ST | 15, 9816, 523<br>(3)[1501, 17062] | 18, 37008, 756<br>(2)[3675, 37541] | 35, 49545, 882<br>(0)[7473, 73273] |
| BACOLI/LE | 15, 8988, 765<br>(1)[1759, 16817] | 23, 20060, 961<br>(1)[2178, 30941] | 46, 45188, 1067<br>(0)[2963, 63104] |
| *p* | 7 | | |
| BACOL/ST | 15, 10841, 399<br>(6)[2422, 34782] | 15, 25576, 754<br>(3)[8314, 81886] | 16, 78797, 1130<br>(2)[18004, 154172] |
| BACOL/LE | 15, 10042, 476<br>(3)[2534, 33486] | 15, 24653, 899<br>(1)[7870, 80468] | 23, 95628, 1099<br>(8)[13204, 139426] |
| BACOLI/ST | 14, 10749, 417<br>(0)[1266, 17696] | 16, 26350, 717<br>(1)[4580, 42141] | 19, 50825, 906<br>(0)[8429, 77783] |
| BACOLI/LE | 15, 10569, 540<br>(6)[1457, 17900] | 15, 24456, 942<br>(1)[4032, 40437] | 22, 48071, 1070<br>(7)[5425, 69758] |
| *p* | 9 | | |
| BACOL/ST | 15, 11863, 325<br>(0)[2466, 36356] | 14, 28769, 638<br>(3)[9344, 88620] | 15, 54217, 950<br>(3)[18700, 156698] |
| BACOL/LE | 14, 12065, 360<br>(6)[2766, 37744] | 15, 26579, 658<br>(0)[8234, 83140] | 15, 54894, 1115<br>(0)[19788, 164190] |
| BACOLI/ST | 15, 11626, 352<br>(0)[1431, 18607] | 15, 28684, 600<br>(0)[4604, 44084] | 14, 54695, 938<br>(1)[9308, 82389] |
| BACOLI/LE | 15, 10968, 425<br>(0)[1400, 17791] | 14, 34408, 765<br>(9)[4714, 44146] | 15, 61190, 1235<br>(6)[10315, 88796] |

Table 1: *Machine independent results for* **TLBEx6** *with* $\epsilon = 10^{-4}$, $p = 5, 7, 9$, *and* $tol = 10^{-4}, 10^{-6}, 10^{-8}$. *Each table entry gives* **Final NINT, Accepted Time Steps**, *and Number of* **Remeshings** *in row 1, and* **(Cold Starts) [Calls to CRDCMP, Calls to CRSLVE]** *in row 2.*

From Table 1, we see that, while the performances of BAC/ST, BAC/LE, SCI/ST, and LOI/LE of course exhibit some relative variations over the $p$ and *tol* values considered, several general observations can be made:

- **Final NINT:** For smaller $p$ values, the final NINT value increases as *tol*

| $tol = 10^{-4}/p =$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|
| BACOL/ST | 4.16 | 5.40 | 7.38 | 9.96 | 12.74 | 15.68 | 21.24 | 26.38 |
| BACOL/LE | 3.25 | 4.16 | 5.40 | 7.38 | 9.96 | 12.74 | 15.68 | 21.24 |
| BACOLI/ST | 2.18 | 2.59 | 3.16 | 4.16 | 5.73 | 7.02 | 8.89 | 10.46 |
| BACOLI/LE | 2.46 | 2.78 | 3.50 | 4.39 | 5.68 | 6.97 | 10.68 | 11.20 |
| $tol = 10^{-6}/p =$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| BACOL/ST | 13.14 | 16.34 | 20.99 | 28.09 | 35.89 | 47.81 | 59.47 | 76.61 |
| BACOL/LE | 13.28 | 13.14 | 16.34 | 20.99 | 28.09 | 35.89 | 47.81 | 59.47 |
| BACOLI/ST | 7.69 | 8.67 | 10.21 | 13.03 | 16.89 | 20.60 | 26.68 | 31.38 |
| BACOLI/LE | 8.36 | 8.03 | 9.89 | 12.36 | 16.25 | 20.80 | 25.01 | 29.22 |
| $tol = 10^{-8}/p =$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| BACOL/ST | 58.20 | 52.24 | 56.82 | 73.24 | 90.09 | 106.72 | 133.77 | 155.88 |
| BACOL/LE | 91.12 | 58.20 | 52.24 | 56.82 | 73.24 | 90.09 | 106.72 | 133.77 |
| BACOLI/ST | 29.63 | 29.36 | 29.38 | 34.72 | 40.63 | 47.09 | 57.77 | 64.13 |
| BACOLI/LE | 71.78 | 32.75 | 30.66 | 32.13 | 42.40 | 50.76 | 56.98 | 67.54 |
| $tol = 10^{-10}/p =$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| BACOL/ST | 231.94 | 124.86 | 134.63 | 130.26 | 150.47 | 176.64 | 202.04 | 213.03 |
| BACOL/LE | 338.08 | 231.94 | 124.86 | 134.63 | 130.26 | 150.47 | 176.64 | 202.04 |
| BACOLI/ST | 146.50 | 87.59 | 78.55 | 73.45 | 79.87 | 84.08 | 99.86 | 111.83 |
| BACOLI/LE | 356.08 | 125.11 | 82.80 | 79.74 | 80.18 | 97.44 | 108.21 | 128.52 |

Table 2: *Machine dependent timings (in seconds),* **TLBEx6** *with* $\epsilon = 10^{-4}$*,* $p = 4, \ldots, 11$*,* $tol = 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}$.

becomes sharper. In contrast, for larger $p$ values, the final NINT value is independent of *tol*. For the coarsest *tol* value, the final NINT value is independent of $p$. However, for sharper *tol* values, the final NINT value is larger for the smallest $p$ value, and decreases as $p$ increases. Combining these points, we see that the final NINT value is largest when $p$ is smallest and *tol* is sharpest. Otherwise, the final NINT values employed by the codes are quite similar.

- **Accepted Time Steps:** The number of accepted time steps increases as *tol* becomes sharper, as expected. The number of accepted time steps used by the four codes is generally about the same and is largely independent of $p$ (although the codes do use slightly more time steps when $p$ is larger.)

- **Remeshings:** Compared to the total number of accepted time steps, there are relatively few remeshings. The number of remeshings is larger when *tol* is sharper. For a given *tol*, the number of remeshings decreases as $p$ increases. This effect is more significant when *tol* is coarse. For given $p$ and *tol* values, the number of remeshings performed by the four codes is generally about the same.

- **Cold Starts:** The number of cold starts is quite insignificant compared

to the total number of accepted time steps, for all codes and $p$, *tol* combinations.

- **Calls to CRDCMP/Calls to CRSLVE:** For small $p$ values, the total number of calls to CRDCMP performed by BAC/ST or BAC/LE is roughly the same as the number of calls made by SCI/ST or LOI/LE. For larger $p$ values, the total number of calls to CRDCMP is roughly twice that of SCI/ST or LOI/LE. The number of calls to CRSLVE performed by BAC/ST or BAC/LE is roughly twice that of SCI/ST or LOI/LE, for all $p$ values. Also, the number of calls to CRSLVE is typically about ten times the number of calls to CRDCMP. These observations provide insight for the timing results in Table 2 that show that, generally, SCI/ST and LOI/LE are about twice as fast as BAC/ST or BAC/LE. This strong correlation between the number of CRDCMP and CRSLVE calls and the overall execution time is expected. See Figure 75 of [17] which shows that, for BAC/ST and the **TLBEx12** test problem, a PDE system consisting of 12 copies of the **TLBE**, with $\epsilon = 10^{-4}$, the execution times for the CRDCMP and CRSLVE routines represent about two thirds of the overall execution time of BAC/ST. For a given code, the number of CRDCMP and CRSLVE calls increases substantially for sharper *tol* values, as expected. The type of error control that is employed (ST or LE) does not significantly impact the number of CRDCMP and CRSLVE calls.

These observations are also supported by the tables of machine independent results for the other test problems presented in [20]. Figures that give comparison plots of some of these machine independent efficiency measures are also provided in [20]. These figures provide further clarification of the relationships among the measures that are discussed in the observations presented in this subsection.

*3.3. Error vs. Execution Time across Codes*

In [20], error vs. execution time results are presented for the test problems **OLBE** and **TLBE** with $\epsilon = 10^{-3}$ and $10^{-4}$, over the allowable range of $p$ values for each code. The tests were conducted for a range of 81 *tol* values from $10^{-2}$ to $10^{-10}$. The *tol* values were uniformly distributed - on a log scale - over $10^{-2}$ to $10^{-10}$.

We consider a representative case from [20], namely, **TLBE** with $\epsilon = 10^{-4}$ and $p = 6$. In Figure 1, a plot of error vs. execution time for each code is given. The plots also show lines fitted to the data for each code to help clarify comparisons among the codes. We see from Figure 1 that over the entire range of errors, BAC/ST and BAC/LE have comparable execution times, SCI/ST and LOI/LE have comparable execution times, and SCI/ST and LOI/LE are consistently less expensive than BAC/ST and BAC/LE.

The comparisons among these codes can be seen more clearly if we plot the execution times of BAC/LE, SCI/ST, and LOI/LE *relative* to that of BAC/ST. These plots were developed as follows. We describe this process for the BAC/LE data.

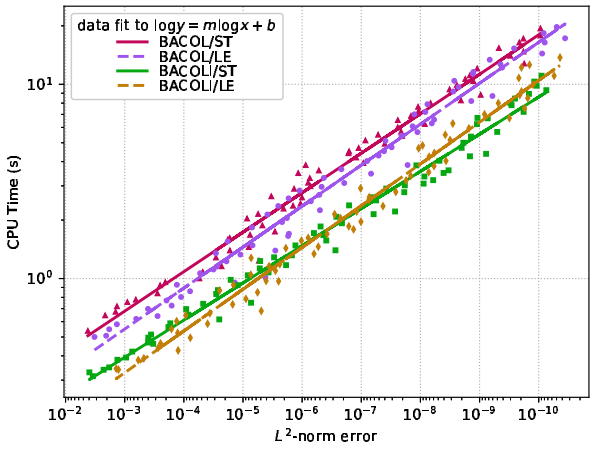


Figure 1: Error vs. execution time for BAC/ST, BAC/LE, SCI/ST, LOI/LE, for **TLBE** with $\epsilon = 10^{-4}$ and $p = 6$.

- We first perform a linear fit to the log of the error vs. log of time data associated with BAC/ST in order to obtain a continuous representation of the baseline BAC/ST data.
- Then, for each (error,time) ordered pair from the BAC/LE data set, we use this linear fit to the BAC/ST data to obtain a corresponding time estimate for BAC/ST; that is, an estimate of how much time BAC/ST would take to compute a solution with the same error as BAC/LE.
- We then calculate the ratio of the actual BAC/LE time to this estimated BAC/ST time. This yields a set of ordered pairs of the form (error, time ratio) that we can associate with BAC/LE.
- Finally, we fit a line to the (log of error, time ratio) ordered pairs and plot this line on a semi-log scale.

This process is repeated for the SCI/ST and the LOI/LE data. A plot is given for the **TLBE** with $\epsilon = 10^{-4}$, $p = 6$ case in Figure 2. We can now see more clearly that SCI/ST and LOI/LE are consistently less expensive than BAC/ST and BAC/LE over the entire error range. The average costs for SCI/ST and LOI/LE are about 50%-60% of the costs for BAC/ST and BAC/LE. For coarse tolerances, the cost for BAC/LE is about 80% of that of BAC/ST. Also, we see that for coarse tolerances, LOI/LE is less expensive than SCI/ST but for sharp

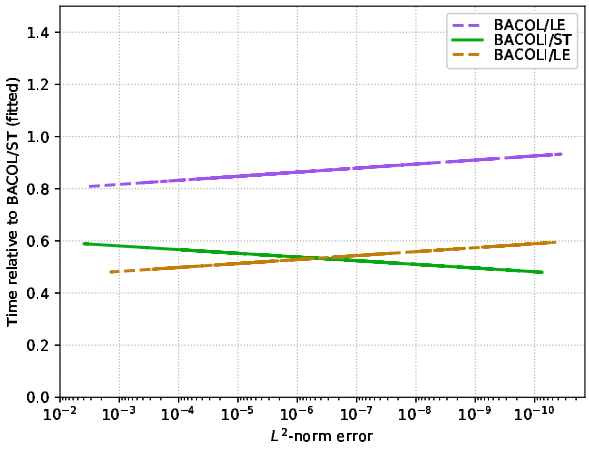


Figure 2: Error vs. execution time for BAC/LE, SCI/ST, and LOI/LE, relative to BAC/ST execution time, for **TLBE** with $\epsilon = 10^{-4}$ and $p = 6$.

tolerances, LOI/LE error is more expensive than SCI/ST. Similar results were obtained in [20] for the **TLBE** with $\epsilon = 10^{-3}$ and for the **OLBE** with $\epsilon = 10^{-3}$ and $10^{-4}$. The only exception is for LOI/LE with $p = 4$. For this case, LOI/LE is much less expensive than any of the other codes when the tolerances are coarse but much more expensive than any of the other codes for sharp tolerances.

### *3.4. Error vs. Execution Time across p Values*

In [20], error vs. execution time results for each code over a range of $p$ values, for the **OLBE** and the **TLBE**, with $\epsilon = 10^{-3}$ and $10^{-4}$, are provided. In Figure 3, we present representative results involving the **OLBE** with $\epsilon = 10^{-4}$ for LOI/LE. Similar results were obtained by the other codes for this problem and by all the codes for the **OLBE** with $\epsilon = 10^{-3}$ and the **TLBE**. *Since these graphs give error vs. execution time results over a range of p values, we can examine the impact that the choice of p has on performance.*

From Figure 3, a general observation is that, for coarse tolerances, LOI/LE is more efficient when $p$ is small, while for sharp tolerances, a larger $p$ value leads to a more efficient computation. For sharp tolerances, small $p$ values lead to substantially higher costs than do larger $p$ values.

### *3.5. Reliability: Tolerance vs. Error*

In [19], an analysis of requested tolerance vs. error achieved performance for the four codes considered in this paper is presented. In Figure 4, we provide a

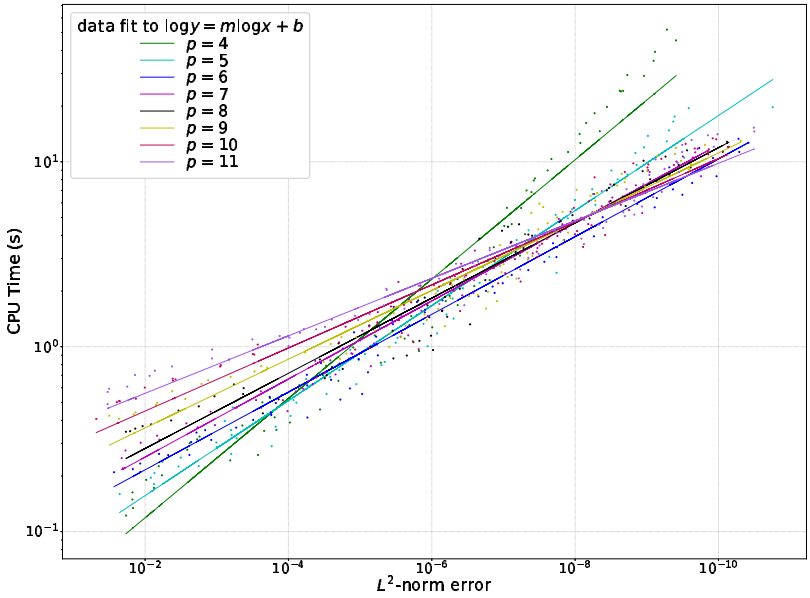


Figure 3: Error vs. Execution time for LOI/LE for the **OLBE** with $\epsilon = 10^{-4}$ and $p = 4, \ldots, 11$.

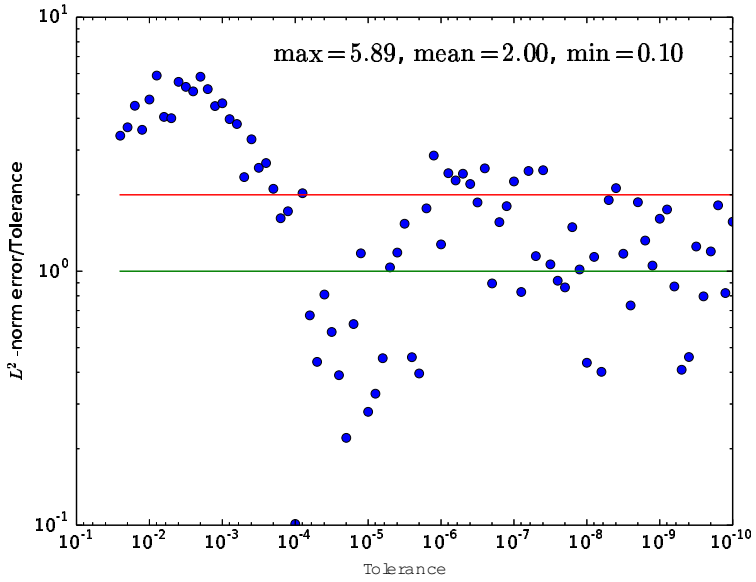


Figure 4: Tolerance vs. Error/Tolerance, BAC/ST, TLBE with $\epsilon = 10^{-4}$, $p = 6$. The bottom horizontal line at $10^0 = 1$ represents the case where the error equals the tolerance. The top horizontal line is the mean error over all tolerances.

typical result showing the performance of BAC/ST with $p = 6$ on the **TLBE** with $\epsilon = 10^{-4}$ for a range of *tol* values from $10^{-1}$ to $10^{-10}$. We plot tolerance vs. the corresponding error/tolerance ratio. From this figure, we see that the mean error is about twice the tolerance and that the maximum factor by which the error exceeds the tolerance is slightly less than 6.

The results reported in [19] show that, over all tests and codes, it is generally the case that, on average, the error is typically only a small multiple of the tolerance, and that, with respect to error estimation scheme and error control mode, the codes have similar reliability. *However, the results presented in [19] also show that the reliability of the codes, with respect to error vs. tolerance, is better for larger p values.* In particular, for larger $p$ values, the error is, on average, typically equal to or slightly less than the tolerance, rather than being somewhat larger than the tolerance, as is the case for the smaller $p$ values. In addition, the maximum amount by which the error exceeds the tolerance is smaller for larger $p$ values.

## 4. Summary, Conclusions and Future Work

This paper presents a detailed examination of the performance of BACOL and BACOLI in which several important performance measures are considered. The results show that:

- the new error estimation schemes/error control modes generally lead to performance measures for BACOLI that are comparable to those of BACOL with the exception of those associated with the factorization and solution of the ABD linear systems. A primary observation is that BACOLI (employing either error estimation scheme/error control mode) uses approximately half as many ABD matrix factorizations and ABD linear system solves as BACOL which leads to substantial savings in execution time,

- for small $p$ and sharp $tol$, LE error control mode leads to the use of more spatial subintervals. This leads to greater execution time costs compared to ST error control mode. Conversely, for coarse tolerances, LE error control mode leads to a more efficient computation than does ST error control mode,

- compared to the total number of accepted steps taken by the codes, the total number of remeshings and cold starts is small. This is important because the computational cost associated with a remeshing or a cold start is significant compared to the cost of a single time step,

- for coarse tolerances, all codes generally have smaller execution times when $p$ is small. However, for sharper tolerances, larger $p$ values lead to better efficiency,

- the correlation between the error achieved and the tolerance requested improves for larger $p$ values.

There are several directions for future work. The results of this paper suggest modifying BACOLI in the following ways:

- have it automatically choose $p$ based on the tolerance requested, and choose the $p$ values over only an intermediate range of values since these give the best combination of efficiency and reliability,

- have it automatically choose the error control mode, since it appears that LE error control is better for coarse tolerances while ST error control is better for sharp tolerances.

As mentioned earlier, another direction for future work involves the development and analysis of BACOLRI, the modification of BACOLR that implements the interpolation-based spatial error estimation schemes introduced in BACOLI. The development of BACOLRI will make extensive use of the results presented in this paper.

The results of this paper will also be used to inform the ongoing development of B-spline collocation software for 2D PDEs [13]. Moreover, we plan to investigate the error estimation schemes employed in the BVODE solvers COLSYS/COLNEW to explore the possibility of modifying these packages to employ the interpolation based error estimations schemes used in BACOLI.

[1] T. Arsenault, T. Smith, and P.H. Muir. Superconvergent interpolants for efficient spatial error estimation in 1D PDE collocation solvers. *Can. Appl. Math. Q.*, 17:409–431, 2009.

[2] T. Arsenault, T. Smith, P.H. Muir, and J. Pew. Asymptotically correct interpolation-based spatial error estimation for 1D PDE solvers. *Can. Appl. Math. Q.*, 20:307–328, 2012.

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

[4] K.E. Brenan, S.L. Campbell, and L.R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, volume 14 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1996.

[5] C. de Boor. *A Practical Guide to Splines*, volume 27 of *Applied Mathematical Sciences*. Springer-Verlag, New York, revised edition, 2001.

[6] J.C. Díaz, G. Fairweather, and P. Keast. Algorithm 603. COLROW and ARCECO: FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination. *ACM Trans. Math. Software*, 9(3):376–380, 1983.

[7] G. Fairweather and I. Gladwell. Algorithms for almost block diagonal linear systems. *SIAM Rev.*, 46(1):49–58, 2004.

[8] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations. I*, volume 8 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 1993.

[9] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations. II*, volume 14 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 1996.

[10] W. Huang and R.D. Russell. A moving collocation method for solving time dependent partial differential equations. *Appl. Numer. Math.*, 20(1-2):101–116, 1996.

[11] W. Huang and R.D. Russell. *Adaptive Moving Mesh Methods*, volume 174 of *Applied Mathematical Sciences*. Springer, New York, 2011.

[12] P. Keast and P.H. Muir. Algorithm 688: EPDCOL: a more efficient PDECOL code. *ACM Trans. Math. Softw.*, 17(2):153–166, 1991.

[13] Z. Li and P.H. Muir. B-spline Gaussian collocation software for two-dimensional parabolic PDEs. *Adv. Appl. Math. Mech.*, 5:528–547, 2013.

[14] NAG Numerical Algorithms Group Fortran library. *d02tlc*. The Numerical Algorithms Group, Ltd., Wilkinson House, Oxford, UK.

[15] N.K. Madsen and R.F. Sincovec. Algorithm 540: PDECOL, general collocation software for partial differential equations. *ACM Trans. Math. Softw.*, 5(3):326–351, 1979.

[16] P.K. Moore. Interpolation error-based a posteriori error estimation for two-point boundary value problems and parabolic equations in one space dimension. *Numer. Math.*, 90(1):149–177, 2001.

[17] J. Pew, Z. Li, and P.H. Muir. A computational study of the efficiency of collocation software for 1D parabolic PDEs with interpolation-based spatial error estimation. *Saint Mary's University, Dept. of Mathematics and Computing Science Report Series, Technical Report 2013_001, http://cs.smu.ca/tech_reports*, 2013.

[18] J. Pew, Z. Li, and P.H. Muir. Algorithm 962: BACOLI: B-spline adaptive collocation software for PDEs with interpolation-based spatial error control. *ACM Trans. Math. Softw.*, 42(3):25:1–25:17, 2016.

[19] J. Pew and P.H. Muir. Tolerance vs. error results for a class of error control B-spline Gaussian collocation PDE solvers. *Saint Mary's University, Dept. of Mathematics and Computing Science Technical Report Series, Technical Report 2015_001, http://cs.smu.ca/tech_reports*, 2015.

[20] J. Pew, C. Tannahill, and P.H. Muir. Performance analysis results for error control B-spline Gaussian collocation PDE solvers. *Saint Mary's University, Dept. of Mathematics and Computing Science Technical Report Series, Technical Report 2018_001, http://cs.smu.ca/tech_reports*, 2018.

[21] Scilab. *bvode, bvodeS.* Scilab Enterprises, 143 bis rue Yves Le Coz, 78000 Versailles, France.

[22] G. Söderlind and L. Wang. Evaluating numerical ODE/DAE methods, algorithms and software. *J. Comput. Appl. Math.*, 185(2):244–260, 2006.

[23] P. Virtanen. *scikits.bvp1lg 0.2.8., https://pv.github.io/scikits.bvp1lg/.*

[24] R. Wang, P. Keast, and P. H. Muir. Algorithm 874: BACOLR: Spatial and temporal error control software for PDEs based on high-order adaptive collocation. *ACM Trans. Math. Softw.*, 34(3):15:1–15:28, 2008.

[25] R. Wang, P. Keast, and P.H. Muir. BACOL: B-spline Adaptive COLlocation software for 1D parabolic PDEs. *ACM Trans. Math. Software*, 30(4):454–470, 2004.

[26] R. Wang, P. Keast, and P.H. Muir. A comparison of adaptive software for 1D parabolic PDEs. *J. Comput. Appl. Math.*, 169(1):127–150, 2004.

[27] R. Wang, P. Keast, and P.H. Muir. A high-order global spatially adaptive collocation method for 1-D parabolic PDEs. *Appl. Numer. Math.*, 50(2):239–260, 2004.