Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Computational and Applied Mathematics 169 (2004) 127–150

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

www.elsevier.com/locate/cam

A comparison of adaptive software for 1D parabolic PDEs☆

Rong Wang[a,*], Patrick Keast[a], Paul Muir[b]

[a] *Department of Mathematics and Statistics, Dalhousie University, Chase Building, Halifax, Nova Scotia, Canada B3H 3J5*

[b] *Department of Mathematics and Computing Science, Saint Mary's University, Nova Scotia, Canada B3H 3C3*

Received 5 June 2003; received in revised form 29 November 2003

Abstract

In this paper we describe BACOL, a high-order, spatially and temporally adaptive software package for solving systems of one-dimensional parabolic partial differential equations and then compare it with several related software packages. BACOL employs collocation at Gaussian points with a *B*-spline basis for the spatial discretization. A modification of DASSL is used for the time integration of the resulting differential-algebraic equations. An equidistribution principle is implemented for the spatial mesh adaptation based on a high-quality a posteriori error estimate, and the spatial error tolerance is coupled with the temporal error tolerance to provide a balanced spatial–temporal error control. We compare BACOL with a related software package, EPDCOL, which uses a fixed-spatial-mesh approach, with several other packages which provide spatial and temporal adaptivity, namely D03PPF, TOMS731, MOVCOL, and with one package, HPNEW, which provides spatial and temporal error control. Numerical results demonstrate that BACOL is robust and that it is generally significantly more efficient than existing solvers for problems having solutions with rapid spatial variation.
© 2003 Elsevier B.V. All rights reserved.

MSC: 65M20; 65M50; 65M70

Keywords: 1D Parabolic PDEs; *B*-splines; Collocation; Equidistribution principle; High-order; Software

1. Introduction

In the conventional method of lines (MOL) approach, one solves a system of parabolic partial differential equations (PDEs) by reducing the PDE system to a system of ordinary differential equations (ODEs) or differential algebraic equations (DAEs). Temporal error control is obtained through the adaptive time stepsize selection (and possibly order selection) algorithm employed by the ODE/DAE

☆ Partially supported by the Natural Sciences and Engineering Research Council of Canada.
* Corresponding author. Tel.: +1-19024310718; fax: +1-19024945130.
E-mail addresses: wang@mathstat.dal.ca (R. Wang), keast@mathstat.dal.ca (P. Keast), muir@stmarys.ca (P. Muir).

0377-0427/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2003.12.016

software. In the past 20 years, adaptive methods for the discretization of the spatial domain of a system of PDEs have drawn considerable attention. The use of adaptive mesh refinement techniques provides a far more efficient way to treat problems with solutions exhibiting rapid spatial variation.

Three basic spatial mesh adaptation strategies are mesh refinement or coarsening (h-refinement) [2], moving mesh (r-refinement) [20], and order variation in the spatial discretization method (p-refinement) [9,10]. Combining two or three different strategies, we can generate more sophisticated techniques, e.g., hp-refinement [2,3]. (In case of time-dependent PDEs (e.g., parabolic or hyperbolic PDEs), p-refinement, to our knowledge, is never applied alone as this would imply a fixed spatial mesh.) Although the adaptive method of lines (AMOL) for solving parabolic PDE systems has been studied for some time, there are relatively few robust, general purpose software packages using these techniques for the solution of systems of one-dimensional (1D) parabolic PDEs. Even fewer of these packages employ *high-order* spatial and temporal discretization schemes. To our knowledge, only the HPSIRK [25], HPDASSL [25] and HPNEW [26] packages provide spatial and temporal error control (and they are experimental codes). Furthermore, little comparison has been made between these software packages (e.g., [25]).

In Section 2, we briefly describe the new software we have developed, the *B*-spline Adaptive *COL*location (BACOL) package, for 1D parabolic PDE systems. A major feature of BACOL is that it uses high-order, adaptive approximations to control the error in both space and time. An approximate solution is computed in a degree p piecewise polynomial subspace represented by a *B*-spline basis [17]. The spatial discretization is performed using collocation at Gaussian points. A modification of DASSL [27], a well-known DAE solver which uses backward differentiation formulas (BDF), is employed for solving the resultant DAE system (providing temporal adaptivity and temporal error control). A high-quality spatial error estimate is obtained using a second approximate solution computed in a degree $p+1$ piecewise polynomial subspace. A remeshing strategy is developed based on an error equidistribution principle; this algorithm can vary the number of subintervals as well as the location of the mesh points, in order to efficiently control the spatial error.

The main goal of this paper is to present a comparison of BACOL with several other related software packages which we will describe in Section 3. This will include most of the currently existing spatially and temporally adaptive software packages for 1D parabolic PDE systems, including D03PPF from the NAG library, TOMS731 [12], MOVCOL [20] and HPNEW [26]. We will also include a comparison with the collocation solver, PDECOL [23]/EPDCOL [21], even though it does not have a spatial mesh adaptation capability, because it is closely related to BACOL in terms of its spatial discretization algorithm. In Section 4, we present a number of numerical experiments, in which we compare BACOL with above solvers on a challenging set of test problems. Section 5 provides our conclusions.

The numerical results presented in this paper clearly show that BACOL is robust and generally considerably more efficient than any other available software package for 1D parabolic PDEs for difficult problems having solutions with rapid spatial variation, especially when a highly accurate approximate solution is required.

1.1. Problem class

We will consider systems of 1D time-dependent parabolic PDEs of the form

$$u_t(x,t) = f(t,x,u(x,t),u_x(x,t),u_{xx}(x,t)), \qquad x_a \leqslant x \leqslant x_b, \quad t \geqslant t_0, \tag{1}$$

where initial conditions are given by $u(x,t_0)=u_0(x)$, $x_a \leqslant x \leqslant x_b$, and separated boundary conditions (BCs) are given by

$$b_L(t,u(x_a,t),u_x(x_a,t))=0, \quad t \geqslant t_0, \qquad b_R(t,u(x_b,t),u_x(x_b,t))=0, \quad t \geqslant t_0, \tag{2}$$

where $u(x,t)$ is a vector of dimension NPDE, and NPDE is the number of PDEs. It is assumed that $f(t,x,u(x,t),u_x(x,t),u_{xx}(x,t))$ is such that the system is parabolic.

The spatial domain for our problem class is $[x_a,x_b]$, and the time domain is (t_0,∞). However, without loss of generality (see [29] for explanation), in order to simplify the discussion in this paper, we assume $x_a=0$, $x_b=1$, and $t_0=0$, although we do wish to emphasize that BACOL is able to handle the general domain $[x_a,x_b]\times(t_0,\infty)$.

2. Description of BACOL software package

2.1. Spatial discretization

To solve (1)–(2) numerically, we consider a mesh consisting of an increasing sequence of $N+1$ points $(N>1)$ in $[0,1]$ such that

$$0=x_0<x_1<\cdots<x_N=1. \tag{3}$$

We associate with the mesh piecewise polynomials of degree p, i.e., we introduce a polynomial of degree p for each subinterval, $[x_{i-1},x_i]$, $i=1,\ldots,N$; C^1-continuity at the internal mesh points is imposed. Thus, the dimension of this piecewise polynomial subspace is $\mathrm{NC}=N(p-1)+2$.

The spatial discretization is closely related to that of the PDECOL [23]/EPDCOL [21] packages. We employ B-splines of degree p, where p $(3\leqslant p\leqslant 11)$ is specified by the user, implemented via the De Boor B-spline package [16]. Let $\{B_i(x)\}_{i=1}^{\mathrm{NC}}$ be the B-spline basis associated with mesh (3) and having C^1-continuity (see [17] for details). The sth component of the solution, $u_s(x,t)$, of (1)–(2) is then approximated by a piecewise polynomial, $U_s(x,t)$, of degree p in x, which is of the form

$$U_s(x,t)=\sum_{i=1}^{\mathrm{NC}} B_i(x)y_{i,s}(t), \tag{4}$$

where $y_{i,s}(t)$ represents the (unknown) time-dependent coefficient of the ith B-spline basis function for the sth component of the solution. We then collocate at $p-1$ Gaussian points in each subinterval; that is, we require the piecewise polynomial (4) to satisfy the PDE at these points. We thus have (see [29] for more details), for $i=1,\ldots,N$ and $j=1,\ldots,p-1$,

$$\sum_{m=(i-1)(p-1)+1}^{i(p-1)+2} B_m(\xi_l)y'_{m,s}(t)=f_s(t,\xi_l,U(\xi_l,t),U_x(\xi_l,t),U_{xx}(\xi_l,t)), \tag{5}$$

where $s=1,\ldots,\mathrm{NPDE}$, $l=(i-1)(p-1)+j$, ξ_l is the jth collocation point in the ith subinterval, $U(x,t)=[U_1(x,t),U_2(x,t),\ldots,U_{\mathrm{NPDE}}(x,t)]$, and we have used the fact that, at each internal collocation point, at most $p+1$ of the basis functions are nonzero. Thus, we obtain a large ODE system in which the unknowns are the B-spline coefficients for the approximate collocation solution, (4).

A significant difference between the spatial discretization scheme of PDECOL and EPDCOL and that of BACOL is that BACOL treats the boundary conditions (BCs) directly, while PDECOL/ EPDCOL treats the BCs in a differentiated form. That is, BACOL treats the BCs in their original form; i.e.,

$$b_{\mathrm{L}}(t, U(0,t), U_x(0,t)) = 0, \quad b_{\mathrm{R}}(t, U(1,t), U_x(1,t)) = 0. \tag{6}$$

The spatial discretization gives the ODE system, (5), which is coupled with the algebraic constraints, (6), to give a system of DAEs which we solve using the DAE software package, DASSL, which we have modified to improve its efficiency in our context. (We will refer the interested reader to [28] for further details.)

2.2. Adaptive mesh refinement

In our spatial mesh refinement strategy, an a posteriori spatial error estimate is obtained by computing a second piecewise polynomial solution approximation having a higher degree $(p+1)$ than that used for the representation of the solution. After each time step, a normalized spatial error estimate, E_s, associated with the sth PDE component over $[0,1]$, is computed and has the form

$$E_s = \sqrt{\int_0^1 \left(\frac{U_s(x,t) - \bar{U}_s(x,t)}{\mathrm{ATOL}_s + \mathrm{RTOL}_s |U_s(x,t)|} \right)^2 \mathrm{d}x}, \quad s = 1, \ldots, \mathrm{NPDE}, \tag{7}$$

where t is the current time, ATOL_s and RTOL_s denote the absolute and relative tolerances for the sth component of the PDE system, respectively, $U_s(x,t)$ is the degree p solution approximation, and $\bar{U}_s(x,t)$ is the degree $p+1$ solution approximation for the sth PDE component.

A remeshing is performed when the error estimate is larger than the tolerance, i.e., $\max_{1 \leqslant s \leqslant \mathrm{NPDE}} E_s \geqslant 1$, or when the mesh is not well distributed, i.e., when the error estimates associated with each subinterval are not roughly the same size. In order to assess the error distribution over the mesh subintervals, we compute two parameters

$$r_1 = \max_{1 \leqslant i \leqslant N} (\hat{E}_i)^{1/(p+1)} \quad \text{and} \quad r_2 = \frac{\sum_{i=1}^{N} (\hat{E}_i)^{1/(p+1)}}{N},$$

where $\hat{E}_i$, the normalized error estimate for the ith subinterval over all NPDE components, is

$$\hat{E}_i = \sqrt{\sum_{s=1}^{\mathrm{NPDE}} \int_{x_{i-1}}^{x_i} \left(\frac{U_s(x,t) - \bar{U}_s(x,t)}{\mathrm{ATOL}_s + \mathrm{RTOL}_s |U_s(x,t)|} \right)^2 \mathrm{d}x}, \quad i = 1, \ldots, N. \tag{8}$$

We note that r_1 represents a measure of the maximum error estimate on each subinterval while r_2 represents the corresponding average. We consider the mesh to be not well distributed if $r_1/r_2 > 2$. Our mesh refinement process is similar to the one in COLSYS [4, p. 370]. As in [4], we employ the $(p+1)$th root in the definition of r_1 and r_2 and choose 2 as the threshold for r_1/r_2.

Priori to the remeshing, BACOL also estimates the number of subintervals needed for the new mesh, N^*. (See [29] for details.) Then instead of using a local refinement as considered, for example, by Adjerid et al. [2], BACOL employs a global mesh refinement strategy based on an equidistribution principle. That is, the new mesh points, $\{x_i^*\}_{i=1}^{N^*}$, are chosen so that

$$\sqrt{\sum_{s=1}^{\mathrm{NPDE}} \int_{x_{i-1}^*}^{x_i^*} \left(\frac{U_s(x,t)-\bar{U}_s(x,t)}{\mathrm{ATOL}_s+\mathrm{RTOL}_s|U_s(x,t)|}\right)^2 \mathrm{d}x} = \text{constant}.$$

Further detail on the spatial error estimation technique and the adaptive mesh refinement algorithm employed in BACOL is available in [29].

2.3. Time integration

A modification of DASSL, a well-known DAE solver based on BDF methods, is employed for the time integration in BACOL. Owing to the type of spatial discretization and the properties of B-splines, the DAE system (5), (6), which is solved by DASSL, leads to a Newton iteration matrix having an almost block diagonal (ABD) form [18]. The linear system solver COLROW [18], takes full advantage of this special structure, and thus is more efficient in treating the ABD systems than the banded solver which is available in DASSL. We therefore introduced a new linear system solver option within DASSL to allow it to employ COLROW for the efficient treatment of the Newton systems.

After each spatial remeshing, BACOL will repeat the current time step using a *warm start* as suggested in [5]. That is, BACOL interpolates all the history vectors, which are required by the DAE solver, from the old mesh to the new mesh, and the same stepsize and order as in the last time step are then tried for the next step. The use of a warm start can significantly improve efficiency [5].

The interpolation must be done carefully. Let $U(x,t)$ denote the approximate solution over the old mesh, and $U^*(x,t)$ denote the approximate solution over the new mesh. We determine $U^*(x,t)$ by requiring it to interpolate $U(x,t)$ at the new collocation points over the new mesh.

As mentioned earlier, we employ a degree p piecewise polynomial for the spatial discretization, which yields the solution approximation, $U(x,t)$, and in order to obtain the spatial error estimate we also do a second computation with a degree $p+1$ piecewise polynomial, which yields a second, higher-order, solution approximation $\bar{U}(x,t)$. Thus, we need interpolants for both approximations over the new mesh, $U^*(x,t)$ and $\bar{U}^*(x,t)$. In order to be sure that error associated with the degree p solution does not affect the spatial error estimate, both interpolations are performed by evaluating $\bar{U}(x,t)$, the $p+1$ solution approximation associated with the old mesh. That is,

$$U^*(\xi_l^*, t_{n-i}) = \bar{U}(\xi_l^*, t_{n-i}), \quad l = 1,\ldots,\mathrm{NC}^*,$$

$$\bar{U}^*(\bar{\xi}_l^*, t_{n-i}) = \bar{U}(\bar{\xi}_l^*, t_{n-i}), \quad l = 1,\ldots,\mathrm{NC}^*,$$

where $i=0,\ldots,k$, k is the order of the BDF method, NC^* is the number of new collocation points, and ξ_l^*, $l=1,\ldots,\mathrm{NC}^*$, are the collocation points over the new mesh. See [29] for further details.

It would be natural to treat the two DAE systems (for the computation of $U(x,t)$ and $\bar{U}(x,t)$) separately. However, this would require running two copies of DASSL, and it would mean that different time steps would likely be employed for the computation of the two solutions. Thus, in order to obtain evaluations of the two solutions at the same time (in order to compute error estimates (7), (8)), interpolation of one of the solutions would be unavoidable. This would considerably increase the complexity of the algorithm and also contribute to the error. A more serious difficulty is that this would make it difficult to perform a warm start after a remeshing, because the values of the two solutions at the previous steps would have to be interpolated from different sources significantly contributing to the error. For these reasons, we have chosen to treat the systems arising from the degree p and $p+1$ spatial discretizations as a single DAE system. This results in a combined Newton iteration matrix consisting of two *decoupled* ABD linear systems. We can solve the decoupled systems separately and COLROW is therefore called twice for each full Newton iteration.

Since BACOL controls both the spatial and temporal error, the most efficient approach is to solve the DAE system with an accuracy requirement, i.e., a time tolerance, which is roughly the same size as the spatial tolerance. On the other hand, it is usual to employ a somewhat tighter tolerance on the temporal error in order to ensure that the spatial error estimate is not contaminated by the temporal error. As discussed in [13], DASSL controls the temporal error by requiring the temporal error estimate to be less than $\frac{1}{3}$ of the temporal tolerance submitted to the code. Therefore, in BACOL we set the absolute and relative tolerance for the spatial error control to be equal to the temporal tolerance given to DASSL. This ensures that the temporal error will be slightly less than the spatial error. We refer the interested reader to [28] for further details.

3. A brief survey of related software

We will now give brief descriptions of the other codes we use in our comparisons. These codes are written in Fortran 77 except for HPNEW which is written in Fortran 90:

- EPDCOL by Keast and Muir [21] is a modification of PDECOL [22,23]. A collocation method is applied using *B*-splines as the piecewise polynomial basis. The collocation points in each subinterval are chosen to be Gaussian points. The boundary conditions are differentiated and coupled with the ODE system from the spatial discretization. GEARIB [19] is employed to solve the resulting ODE system, on which only a relative tolerance can be imposed. EPDCOL uses a fixed but possibly nonuniform user-provided mesh and thus does not provide spatial error control.
- D03PPF, which is based on SPRINT [7], is in the NAG library [15]. It is able to solve a system of 1D PDEs coupled with ordinary differential equations. The spatial discretization is based on second-order finite difference methods. The boundary conditions are treated in their original form. The resulting DAE system is integrated using a BDF method [13] or a Theta method [8] with the associated nonlinear systems solved by an algorithm which switches between Newton's method and functional iteration. This software package employs an h-refinement approach using a fixed number of mesh points, i.e., it is only able to redistribute the given number of mesh points, and therefore no spatial error control is available. The user needs to supply a subroutine MONITF which is used as a monitor function by the code in order to choose a mesh which equally distributes the integral of the monitor function over the spatial domain. The user also needs to specify how often the code will check a remeshing criterion to decide whether a remeshing is necessary; e.g., after a fixed number of time steps, or after a specified fixed time interval.
- TOMS731 is an r-refinement package written by Blom and Zegeling [12]. A finite-element method of second-order accuracy is used for the spatial discretization. It treats the boundary conditions directly. The time integration is performed using DASSL. The user is required to provide a monitor

function which will be used by the code to control mesh movement. As with all r-refinement codes, the number of mesh points is fixed and spatial error control is therefore not available.

- MOVCOL is an experimental code developed by Huang and Russell [20]. It employs the r-refinement approach. A cubic Hermite collocation approach is used for spatial discretization of the PDE, and a standard second-order central finite difference discretization is used for the mesh equations. The boundary conditions are treated directly. DASSL is employed to solve the resulting DAE system. No spatial error control is available.
- HPNEW [26], also an experimental code, is a modification of HPDASSL [25] developed by Moore. It applies hp-refinement using a local refinement strategy. The boundary conditions are treated in their original form. A remeshing is performed after every 5 time steps taken by DASSL. The inclusion of an h-refinement capability allows this code to control the spatial error. In [25] HPSIRK is generally shown to be less efficient compared with HPDASSL. Thus, in this paper we will only consider HPNEW since it is a modification of HPDASSL.

All of the packages provide temporal adaptivity and error control through the time-stepping algorithm tolerance associated with the ODE or DAE integrator. The code, EPDCOL, employs only a fixed spatial mesh. The other codes, except for HPNEW, are able to redistribute or move a given fixed number of spatial mesh points to adapt to the solution behavior but cannot control spatial error. Only the code HPNEW is able to both redistribute and refine the spatial mesh, giving it the capability to control a spatial error estimate.

4. Numerical comparisons

BACOL has been extensively tested; some results are reported in [29]. In this section, we will provide comparisons of BACOL with the other packages.

A major decision when one is assessing software against others is what performance criteria to use. There are many important criteria, such as accuracy, robustness, storage requirements, flexibility, efficiency, ease of use, etc. Some of these design criteria may conflict with each other. For instance, providing users with many options may affect the ease of use although it may improve the flexibility; requiring users to provide a subroutine for calculating an analytical Jacobian matrix, which is a common source of programming errors especially when the system of PDEs is large and complex, improves the efficiency but also affects the ease of use. Thus, it is clear that the task of comparison of different codes is nontrivial, and a comparison is obviously biased by the class of test problems and comparison criteria. However, this is not to say that attempting a comparison is impossible. As well, one can sometimes specify the types of problems for which a given code is more suitable. Often a primary concern is to obtain an approximate solution to a desired accuracy as quickly as possible. Therefore, *efficiency* will be the major criterion in our code comparison. For each test problem, we will measure how much time each code requires to achieve a certain accuracy.

In this section, we will consider five different time-dependent 1D parabolic PDE problems, one of which is a system. All the numerical experiments are done on an SUN SPARC station, with a CPU clock rate of 480.0 MHz and a main memory clock rate of 96.0 MHz. SUNWspro/bin/f 77 is the FORTRAN compiler for all the software packages except HPNEW, for which SUNWspro/bin/f90 is used. Compilation is done using the -O switch, which provides a basic level of optimization.

The notation used and the statistics collected include:

N	the number of subintervals
p	the degree of the piecewise polynomial, $3 \leqslant p \leqslant 11$, for the primary solution approximation
T_{out}	the time at which the numerical solution is requested
TOL	the user supplied tolerance (we set ATOL=RTOL=TOL, where ATOL is the scalar absolute tolerance and RTOL is the scalar relative tolerance)
CPU	the execution time in seconds
Error	the L^2-norm of the absolute error (see (9) below), measured at time $t = T_{\text{out}}$.

At the output time, T_{out}, we calculate the L^2-norm error; i.e., the difference between the approximate solution and the exact solution:

$$\|U_s(x,t) - u_s(x,t)\|_2 = \sqrt{\int_0^1 (U_s(x,t) - u_s(x,t))^2 \, \mathrm{d}x}, \tag{9}$$

where $s = 1, \dots, \text{NPDE}$ and $u_s(x,t)$ represents the exact solution for the sth component of the system of PDEs, and $U_s(x,t)$ represents the approximate solution for the sth component of the system of PDEs. BACOL controls a blended absolute and relative spatial error estimate, (7), instead of the pure absolute error. A different normalized spatial error estimate is used by HPNEW, which controls the H^1-norm error. (Recall that the other codes do not control the spatial error and thus do not compute spatial error estimates.) However, we note that the values of the solution components in all of our experiments are O(1), and the (absolute) L^2-norm error is therefore reasonable for our comparisons. In our testing, for each code and each problem, we ran the code over a range of tolerances, and plotted CPU time versus actual error, in the L^2-norm.

From the descriptions in Section 3, we see that, apart from BACOL, HPNEW is the only other package in this comparison which is able to add or remove mesh points during the computation and thus attempt to compute a solution to a given spatial error tolerance. Therefore, to obtain "almost optimal" results using EPDCOL, D03PPF, TOMS731, and MOVCOL, for a given TOL, we assume the initial mesh to be uniform and find a suitable value for N, for a given code, as follows. For a given tolerance, we gradually increase N and compute solutions and L^2-norm errors until the latter is close to 2 or 3 times the tolerance (in which case the packages generally work most efficiently), or until the L^2-norm of the error appears to no longer be dependent on N (which implies that the time error is dominating). We then consider this value of N to be "almost optimal" for the given TOL and code. *It should be noted that this process provides a substantial advantage to the codes which are not able to estimate and control the spatial error as there is certainly a nontrivial amount of effort expanded by HPNEW and BACOL to perform these tasks.*

We now describe the details for the initial settings and user supplied parameters for all the software packages employed in our numerical experiments:

- BACOL requires the degree of the piecewise polynomial, p, to satisfy $3 \leqslant p \leqslant 11$, and we use BACOL with $p = 3$ and 6 in our numerical experiments. The initial mesh is chosen to be a uniform mesh with $N = 10$.
- EPDCOL requires the user to supply the initial stepsize for the ODE solver, which is chosen to be 10^{-9} in our experiments.

- D03PPF has an option to employ a BDF method or a Theta method in the time integration. Since all the other software packages use BDF methods, we will choose the BDF option for D03PPF. We also choose the banded linear system solver option which is more suitable than using a dense linear system solver for systems arising from the method of lines. In our experiments, the monitor function for D03PPF is based on the second derivative (curvature), $|u(x,t)_{xx}|$, for a single equation, and on $1+(1/\text{NPDE})\sum_{s=1}^{\text{NPDE}}|u_s(x,t)_{xx}|$, for a system of NPDE equations. We also ask D03PPF to check the remeshing criterion after every three time steps.
- For TOMS731, the monitor function we choose is given by the formula

$$\sqrt{0.01+\frac{1}{\text{NPDE}}\sum_{s=1}^{\text{NPDE}}(u_s(x,t)_x)^2},$$

  which is recommended by the authors. To our knowledge, there is no subroutine provided by TOMS731 which allows one to compute solution values for nonmesh points. Therefore, after TOMS731 reaches T_{out}, our driver program will call a subroutine from the NAG library, D03PZF, which uses a cubic interpolant to generate solution values. (Since the TOMS731 spatial discretization is second-order, a cubic interpolant is of more than sufficient accuracy.)
- HPNEW has an option to allow the user to specify an initial mesh, or employ a default initial mesh, which is a uniform mesh with $N=250$. We use HPNEW with the default initial mesh. We also use the default initial order, $p=3$, provided by HPNEW.

The following problems have been used by many authors for the evaluation of software for the numerical solution of 1D parabolic PDE systems. Some problems include one or more parameters which can be varied to adjust the difficulty of the problem. We now present our test problems and computational results. We plot the exact L^2-norm error, (9), at T_{out} against the CPU time for all the codes. There are some problems for which the exact solutions are unknown. We will solve those problems with a very sharp tolerance ($\text{TOL}=10^{-13}$) with BACOL to obtain an sufficiently accurate approximation which serves as the "exact" solution. During our numerical experiments, some codes are unable to compute an approximate solution in a reasonable amount of time for a desired high accuracy. As well, some of the codes stop with an error message before they reach T_{out} for some of the sharper tolerance values. We thus stop reporting the numerical results for those codes if either case happens. Additional details will be provided for the latter case within the comments for each experiment.

4.1. Problem 1

This is Burgers' equation

$$u_t=-uu_x+\varepsilon u_{xx}, \qquad 0<x<1, \quad t>0 \tag{10}$$

with initial condition

$$u(x,0)=\frac{0.1e^{-A_0}+0.5e^{-B_0}+e^{-C_0}}{e^{-A_0}+e^{-B_0}+e^{-C_0}}, \qquad 0\leqslant x\leqslant 1$$

and boundary conditions

$$u(0,t) = \frac{0.1e^{-A_{\rm L}} + 0.5e^{-B_{\rm L}} + e^{-C_{\rm L}}}{e^{-A_{\rm L}} + e^{-B_{\rm L}} + e^{-C_{\rm L}}}, \qquad t \geqslant 0,$$

$$u(1,t) = \frac{0.1e^{-A_{\rm R}} + 0.5e^{-B_{\rm R}} + e^{-C_{\rm R}}}{e^{-A_{\rm R}} + e^{-B_{\rm R}} + e^{-C_{\rm R}}}, \qquad t \geqslant 0,$$

where

$$A_0 = \frac{0.05}{\varepsilon}(x-0.5), \quad B_0 = \frac{0.25}{\varepsilon}(x-0.5), \quad C_0 = \frac{0.5}{\varepsilon}(x-0.375),$$

$$A_{\rm L} = \frac{0.05}{\varepsilon}(-0.5+4.95t), \quad B_{\rm L} = \frac{0.25}{\varepsilon}(-0.5+0.75t), \quad C_{\rm L} = \frac{0.5}{\varepsilon}(-0.375),$$

$$A_{\rm R} = \frac{0.05}{\varepsilon}(0.5+4.95t), \quad B_{\rm R} = \frac{0.25}{\varepsilon}(0.5+0.75t), \quad C_{\rm R} = \frac{0.5}{\varepsilon}(0.625),$$

where, for the viscosity parameter, ε, we will consider two values, $\varepsilon = 10^{-3}$ and $\varepsilon = 10^{-4}$.

This problem is taken from [1,11], and has the exact solution

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

where

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

(In our numerical experiments, when we compute the error in the numerical solution by comparing it with the exact solution, we scale the denominator and the numerator of the exact solution to avoid overflow. That is, if $D = \min(A,B,C)$, then we will multiply both the denominator and the numerator by e^D and the exact solution becomes

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

To avoid underflow, we check $D-A$, $D-B$, and $D-C$. If any of them is less than -35, we will let the corresponding exponential function be zero (since we use double precision).

The exact solution is plotted in Fig. 1 for $\varepsilon = 10^{-4}$, $0 \leqslant t \leqslant 1$, $0 \leqslant x \leqslant 1$.

The solution begins with two wavefronts. They move from left to right and merge to form one wavefront. As mentioned in [24], the thickness of the wavefronts is $O(\varepsilon)$.

Figs. 2 and 3 show the performance of all the codes for Problem 1 with $\varepsilon = 10^{-3}$ and 10^{-4}, respectively. The L^2-norm error, (9), at $T_{\rm out} = 1$, is plotted against the CPU time, both in logarithmic scales.

We make the following observations:

- For both test problems (with $\varepsilon = 10^{-3}$ or 10^{-4}), BACOL is comparable in performance to most of the other codes at coarse tolerances, and it is significantly more efficient when higher accuracy solutions are required. Furthermore, for $\varepsilon = 10^{-4}$, for accuracy requirements sharper than 10^{-7}, it

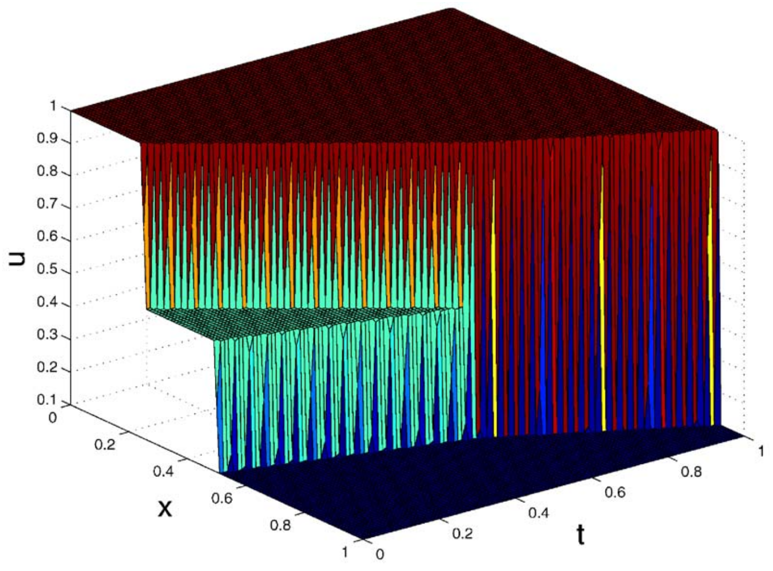

Fig. 1. $u(x,t)$ for Problem 1, for $\varepsilon = 10^{-4}$.

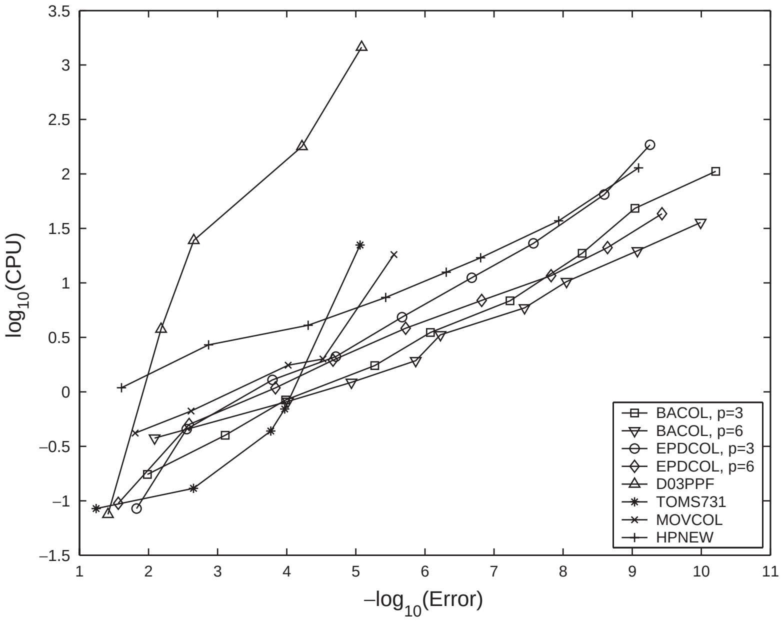

Fig. 2. The CPU time and the L^2-norm error for Problem 1 with $\varepsilon = 10^{-3}$.

is the only package that is able to compute a solution in a reasonable amount of time, other than EPDCOL (which has been provided with the advantage of having a predetermined mesh).

- For the test with $\varepsilon = 10^{-3}$, HPNEW is less efficient than any of the other codes for coarse tolerances. The code has a tendency to use many more mesh points than necessary. For example,

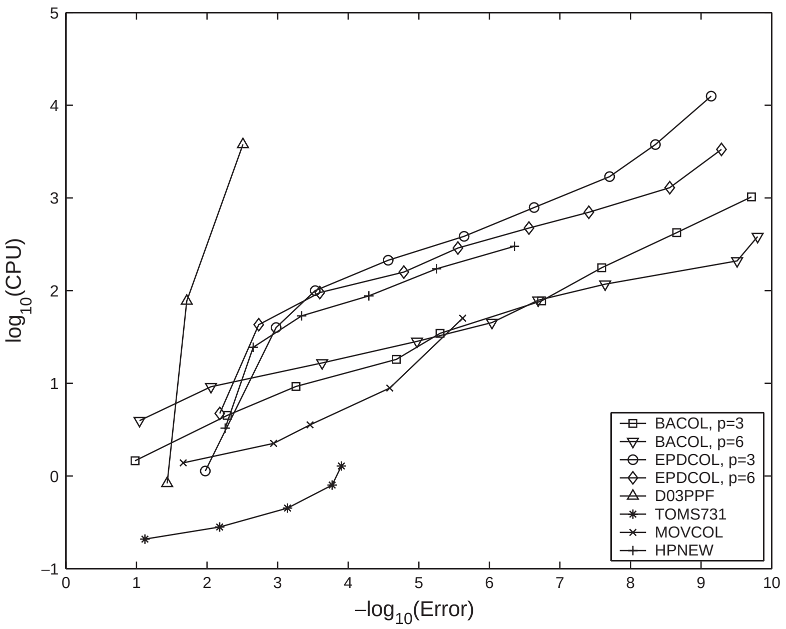

Fig. 3. The CPU time and the L^2-norm error for Problem 1 with $\varepsilon = 10^{-4}$.

with $\varepsilon = 10^{-3}$ and $\mathrm{TOL} = 10^{-2}$, HPNEW uses $N = 250$, compared with $N = 13$ for BACOL with $p = 3$. We use HPNEW with a default initial mesh of 250 elements. It is likely that HPNEW could perform better with a coarser initial mesh. However HPNEW has the ability to coarsen the mesh as well as refine it. For the $\varepsilon = 10^{-4}$ case, for coarse tolerances HPNEW is relatively more efficient but it becomes less efficient for smaller tolerances. When we request an approximate solution with an error smaller than 10^{-7}, HPNEW outputs an error message and terminates.

- When the problem becomes more difficult (i.e., as ε changes from 10^{-3} to 10^{-4}), the fixed-mesh package, EPDCOL, is relatively less efficient compared to most of the other packages. However, it is able to compute approximate solutions with high accuracy in a reasonable amount of time.

Results for D03PPF, TOMS731 and MOVCOL are provided for errors only as small as about 10^{-6}, because these codes could not compute a solution in a reasonable amount of time, for smaller error requests:

- D03PPF is the least efficient software package for a given accuracy, and in fact cannot, in a reasonable amount of time, provide as much accuracy as the other packages.
- When the requested accuracy is relatively modest, TOMS731 is the most efficient package (with the assumption that an optimal number of mesh points is found a priori). However, it is unable to provide an approximate solution with an error smaller than 10^{-4} or 10^{-5}. Furthermore, for some TOL values greater than 10^{-4}, we found some N values for which TOMS731 fails with an error message.

- MOVCOL performs well for coarse or moderate tolerances. But when we require an error smaller than 10^{-6}, MOVCOL becomes less efficient.

4.2. Problem 2

Our second problem is also Burgers' equation, but with a different set of initial and boundary conditions leading to a relatively more difficult solution in terms of its spatial behavior. The PDE is again (10) with initial condition

$$u(x,0) = 0.5 - 0.5\tanh\left(\frac{1}{4\varepsilon}(x-0.25)\right), \quad 0 \leqslant x \leqslant 1$$

and boundary conditions

$$u(0,t) = 0.5 - 0.5\tanh\left(\frac{1}{4\varepsilon}(-0.5t-0.25)\right), \quad t \geqslant 0,$$

$$u(1,t) = 0.5 - 0.5\tanh\left(\frac{1}{4\varepsilon}(0.75-0.5t)\right), \quad t \geqslant 0.$$

The exact solution is

$$u(x,t) = 0.5 - 0.5\tanh\left(\frac{1}{4\varepsilon}(x-0.5t-0.25)\right).$$

It is plotted in Fig. 4 for $\varepsilon = 10^{-4}$, $0 \leqslant t \leqslant 1$, $0 \leqslant x \leqslant 1$.

We note that this solution has a wavefront whose thickness is O(ε), moving from left to right with a constant speed.

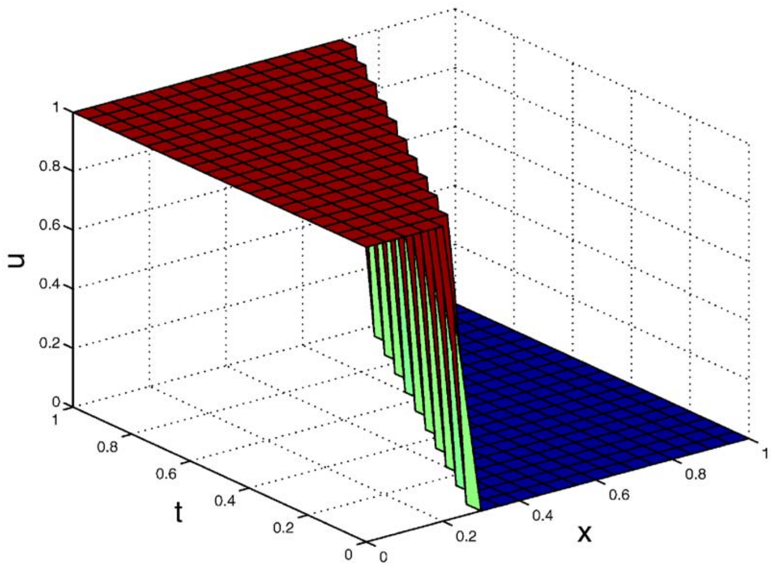

Fig. 4. $u(x,t)$ for Problem 2, for $\varepsilon = 10^{-4}$.

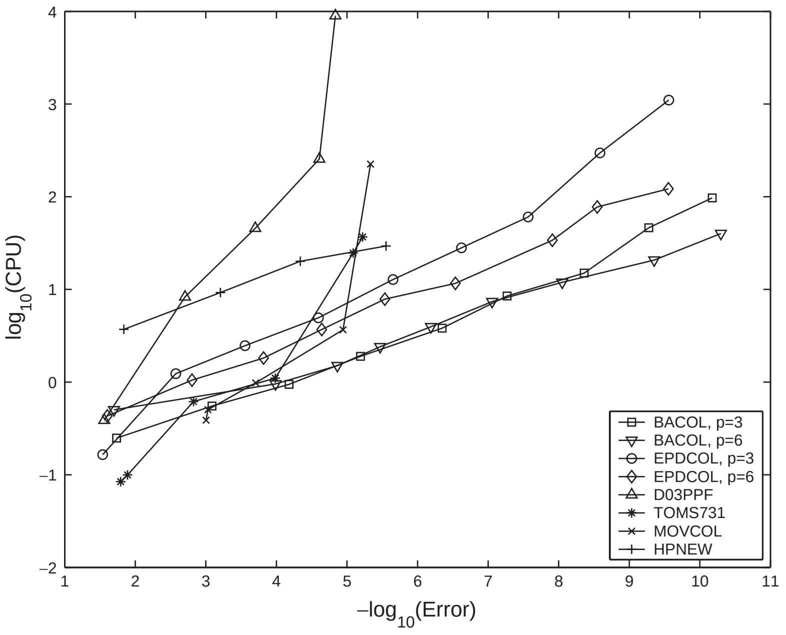

Fig. 5. The CPU time and the L^2-norm error for Problem 2 with $\varepsilon = 10^{-3}$.

Figs. 5 and 6 show the performance of all the codes for Problem 2 with $\varepsilon = 10^{-3}$ and $\varepsilon = 10^{-4}$, respectively. The L^2-norm error is computed at $T_{\text{out}} = 1$.

- From Figs. 5 and 6, we see that, except for EPDCOL, BACOL is the only code that is able to obtain, in a reasonable amount of time, approximate solutions with an error that is less than about 10^{-6}. Even though EPDCOL is given the significant advantage of a preselected mesh, BACOL is still more efficient than EPDCOL, particularly in the $\varepsilon = 10^{-4}$ case.
- HPNEW is relatively less efficient than several of the other codes for coarse tolerances and is unable to produce a solution with an error smaller than about 10^{-6} or 10^{-7}. For error requests smaller than this, the code fails with an internal error message. BACOL is more efficient than HPNEW for this problem, even at coarse tolerances.
- EPDCOL is the only code, except BACOL, that is able to produce solutions with errors smaller than about 10^{-6} or 10^{-7}. It is less efficient than BACOL over all tolerances.
- We see that D03PPF, TOMS731, and MOVCOL are only able to compute solutions in a reasonable amount of time when low accuracy is sufficient.

We can use Figs. 2, 3, 5, and 6 to make an additional comment about the relationship between efficiency and the order of the collocation method. Generally, when the solution of the PDE is smooth and one is using a fixed mesh, it is more efficient to use a high-order discretization for the spatial domain, (see numerical results in [6,22]) when a sharp tolerance is requested. For coarser tolerances, in the fixed mesh context, lower-order methods can be more efficient. In Figs. 2, 3, and 5, EPDCOL with $p = 3$ is more efficient than EPDCOL with $p = 6$. However, in Fig. 6, where the

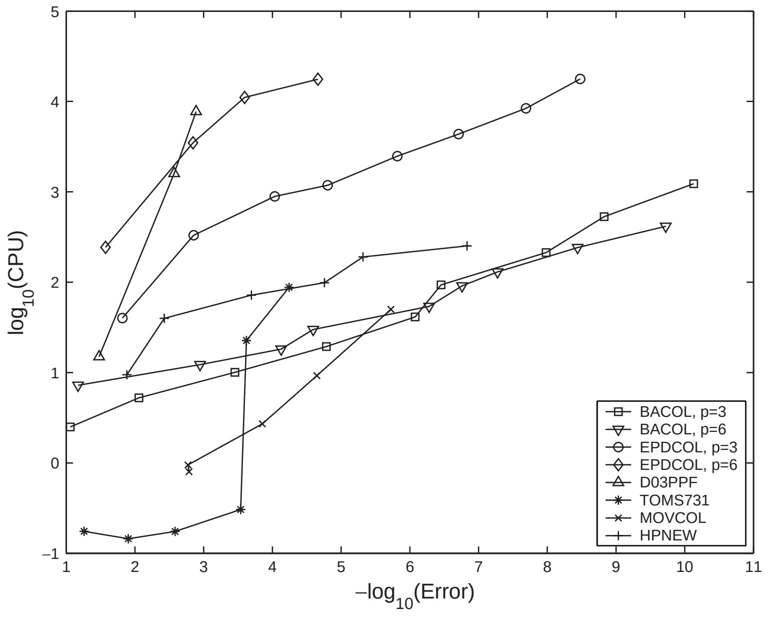

Fig. 6. The CPU time and the L^2-norm error for Problem 2 with $\varepsilon = 10^{-4}$.

corresponding solution is less smooth, we see that the high-order discretization ($p = 6$) for EPDCOL is substantially less efficient than the low-order discretization, for all tolerances. We further note that for BACOL, where adaptive spatial meshes are employed, the performances of the code with $p = 3$ and 6 are comparable, with the high-order method being slightly more efficient for sharper tolerances.

4.3. Problem 3

The third problem is the Cahn–Allen equation [25], which has the form

$$u_t = \varepsilon u_{xx} - u^3 + u, \qquad 0 < x < 1, \quad t > 0 \tag{11}$$

with initial condition

$$u(x,0) = 0.01\cos(10\pi x), \qquad 0 \leqslant x \leqslant 1$$

and boundary conditions

$$u_x(0,t) = 0, \quad u_x(1,t) = 0, \qquad t \geqslant 0.$$

We include this problem because it allows us to investigate a PDE with Neumann boundary conditions.

No exact solution is available; a high-precision numerical solution obtained using BACOL is shown in Fig. 7. The solution is plotted for $0 \leqslant x \leqslant 1$ and $0 \leqslant t \leqslant 8$.

There are two phases of solution behavior. During the first phase, a transition phase, the solution begins relatively flat and close to zero and then quickly develops sharp interfaces, leading to

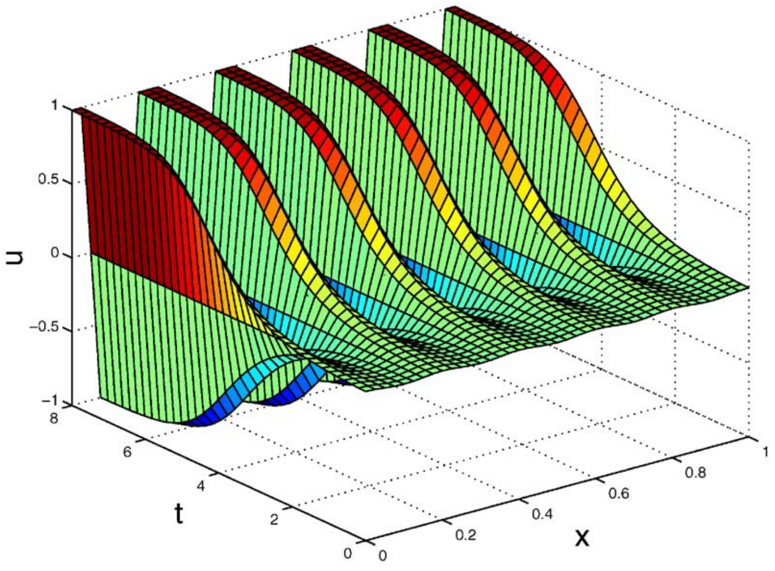

Fig. 7. $u(x,t)$ for Problem 3 with $\varepsilon = 10^{-6}$.

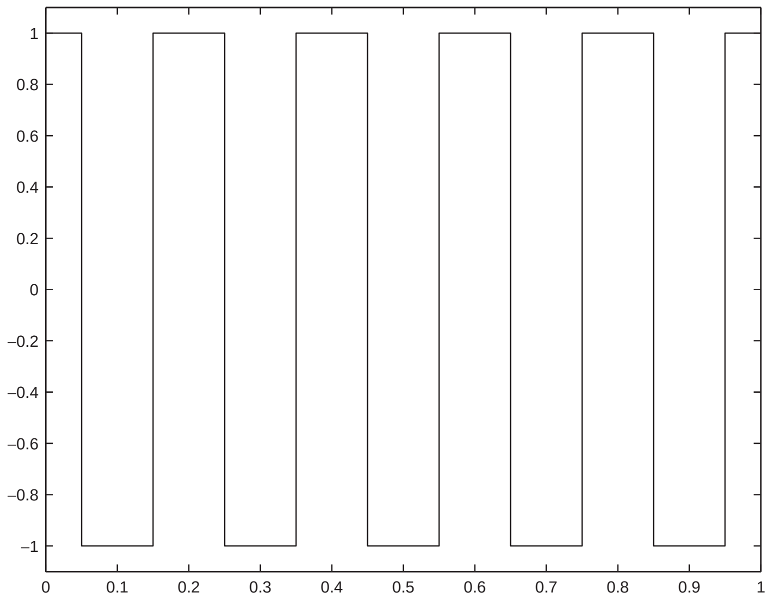

Fig. 8. $u(x,t)$ at $t = 8$.

step function behavior. During the second phase, the solution exhibits this consistent step function behavior. See Figs. 7 and 8.

Fig. 9 shows the performance of all the codes for Problem 3 with $\varepsilon = 10^{-6}$. The L^2-norm error is computed at $T_{\text{out}} = 36$, which is the T_{out} value used in [25].

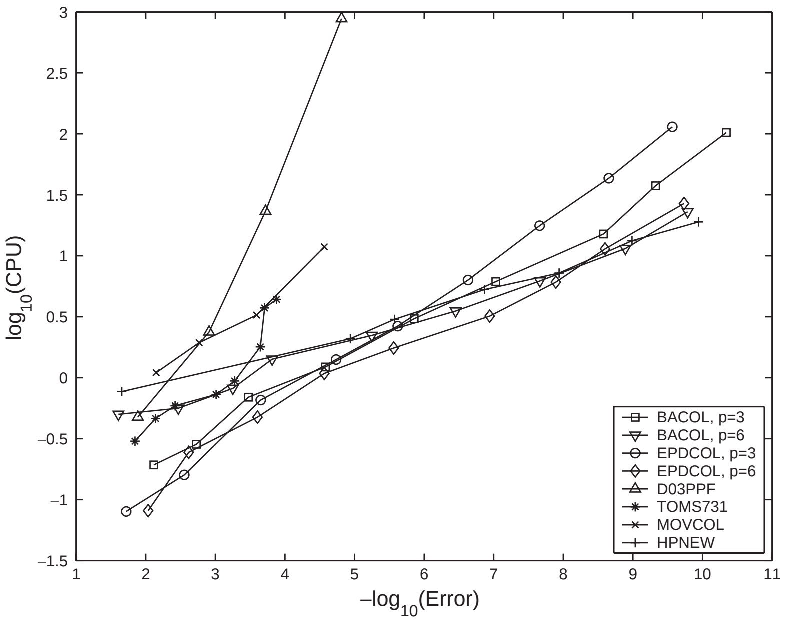

Fig. 9. The CPU time and the L^2-norm error for Problem 3.

We note that EPDCOL, BACOL, and HPNEW work very well for this problem. When the tolerance is sharp, HPNEW is comparable in efficiency to BACOL and EPDCOL although it needs relatively more CPU time when a coarser tolerance is requested. We also see that MOVCOL, D03PPF, and TOMS731 are generally less efficient compared with HPNEW, EPDCOL, and BACOL.

4.4. Problem 4

The fourth problem is taken from [14]. We include this problem because the PDE contains an exponential nonlinear coefficient for the term u_{xx}; as well, we will see that some of the codes fail on this problem. It has the form

$$u_t = (e^{au} u_x)_x, \qquad 0 < x < 1, \quad t > 0 \tag{12}$$

with initial condition

$$u(x,0) = bx, \qquad 0 \leqslant x \leqslant 1$$

and boundary conditions

$$u(0,t) = 0, \quad u(1,t) = b, \qquad t \geqslant 0.$$

We consider the case when $a = 5$ and $b = 2$. The steady-state solution is given (in [14]) by

$$u(x, t \to \infty) = \frac{1}{a} \log[1 + (e^{ab} - 1)x].$$

The solution begins with a straight line, and the steady-state solution is obtained in a short period of time. The exact solution is not available; however, a high-precision numerical solution obtained

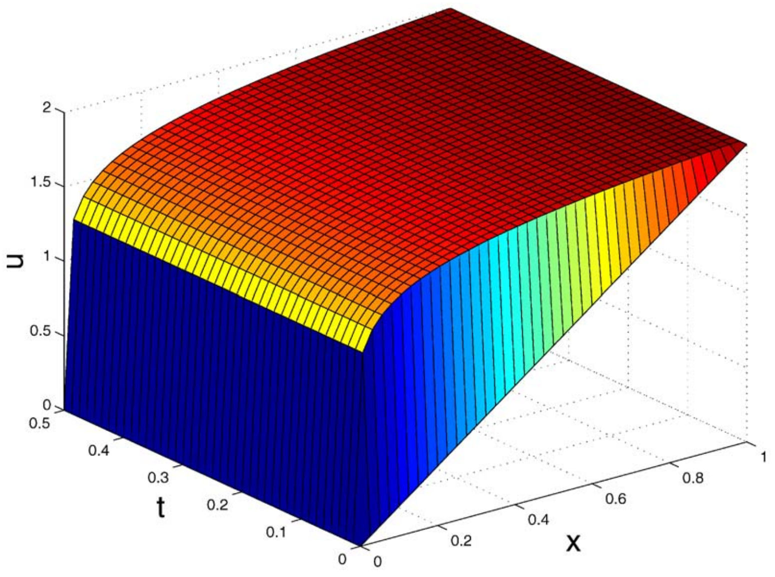

Fig. 10. $u(x,t)$ for Problem 4.

using BACOL is shown in Fig. 10. The solution is plotted for $0 \leqslant x \leqslant 1$ and $0 \leqslant t \leqslant 0.5$. This is a relatively simple problem in terms of its spatial difficulty.

In our experiments, we found that, for all tolerances, when we attempted to solve this problem using EPDCOL, the Newton iteration repeatedly failed to converge before it reached the output time, $T_{\text{out}} = 0.5$. When we attempted to solve this problem using HPNEW for any tolerance, the code gave an error message indicating that, after a remeshing, two adjacent mesh points of the new mesh were so close as to appear coincident. Therefore, in Fig. 11 we do not include results for EPDCOL and HPNEW. The L^2-norm error is computed at $T_{\text{out}} = 0.5$.

We note that MOVCOL is able to compute an approximate solution of the L^2-norm error within 10^{-8} in a reasonable CPU time, which it is unable to do for the other problems. Once again TOMS731 is the most efficient when a coarse tolerance is used, and D03PPF is the least efficient of all the codes. As before, BACOL exhibits comparable performance for coarse tolerances and is the most efficient when a small tolerance is requested.

4.5. Problem 5

The fifth problem is also taken from [25], where the author considers the reaction–diffusion–convection system for modeling a catalytic surface reaction. We include this problem because it allows us to investigate code performance on a system of PDEs exhibiting challenging solution behavior. It has the form

$$(u_1)_t = -(u_1)_x + n(D_1 u_3 - A_1 u_1(1 - u_3 - u_4)) + \frac{1}{Pe_1}(u_1)_{xx},$$

$$(u_2)_t = -(u_2)_x + n(D_2 u_4 - A_2 u_2(1 - u_3 - u_4)) + \frac{1}{Pe_1}(u_2)_{xx},$$

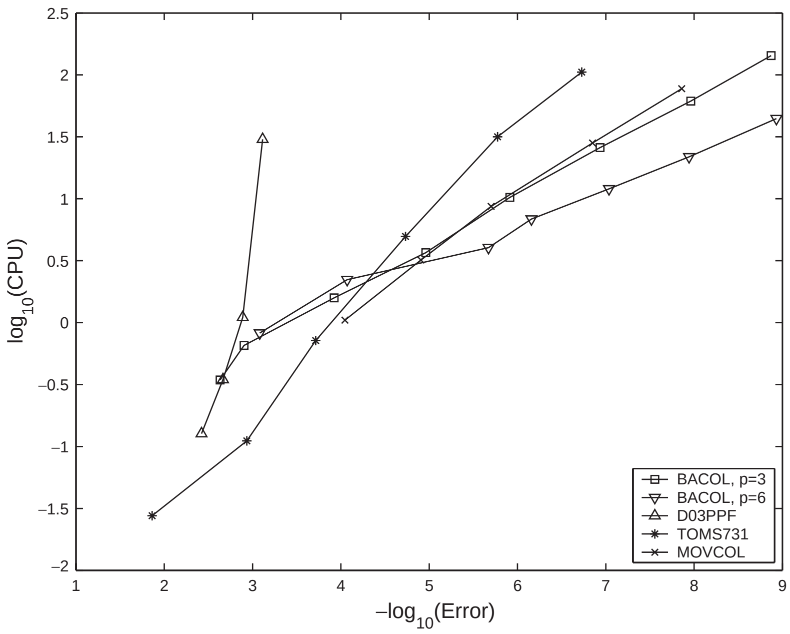

Fig. 11. The CPU time and the L^2-norm error for Problem 4.

$$(u_3)_t = A_1 u_1(1 - u_3 - u_4) - D_1 u_3 - R u_3 u_4 (1 - u_3 - u_4)^2 + \frac{1}{Pe_2}(u_3)_{xx},$$
$$(u_4)_t = A_2 u_2(1 - u_3 - u_4) - D_2 u_4 - R u_3 u_4 (1 - u_3 - u_4)^2 + \frac{1}{Pe_2}(u_4)_{xx}, \qquad (13)$$

where $0 < x < 1$ and $t > 0$, with initial conditions

$$u_1(x,0) = 2 - r, \quad u_2(x,0) = r, \quad u_3(x,0) = u_4(x,0) = 0, \qquad 0 < x < 1$$

and boundary conditions

$$\frac{1}{Pe_1}(u_1)_x(0,t) = -(2 - r - u_1), \quad \frac{1}{Pe_1}(u_2)_x(0,t) = -(r - u_2), \qquad t > 0,$$
$$(u_3)_x(0,t) = (u_4)_x(0,t) = 0, \qquad t > 0,$$
$$(u_1)_x(1,t) = (u_2)_x(1,t) = (u_3)_x(1,t) = (u_4)_x(1,t) = 0, \qquad t > 0,$$

where $u_1(x,t)$ and $u_2(x,t)$ are nondimensionalized concentrations, $u_3(x,t)$ and $u_4(x,t)$ are coverage of adsorbed reactants on the catalytic wall, Pe_1 and Pe_2 are Peclet numbers, and D_1, D_2, R, A_1 and A_2 are Damkohler numbers. This problem includes diffusion, reaction and convection. We choose $A_1 = A_2 = 30$, $D_1 = 1.5$, $D_2 = 1.2$, $R = 1000$, $r = 0.96$, $n = 1$ and $Pe_1 = Pe_2 = 100$.

The problem does not have an exact solution; high-precision numerical approximations for the solution components are shown in Figs. 12–15, obtained using BACOL. The solution is plotted for $0 \leqslant x \leqslant 1$ and $0 \leqslant t \leqslant 18$.

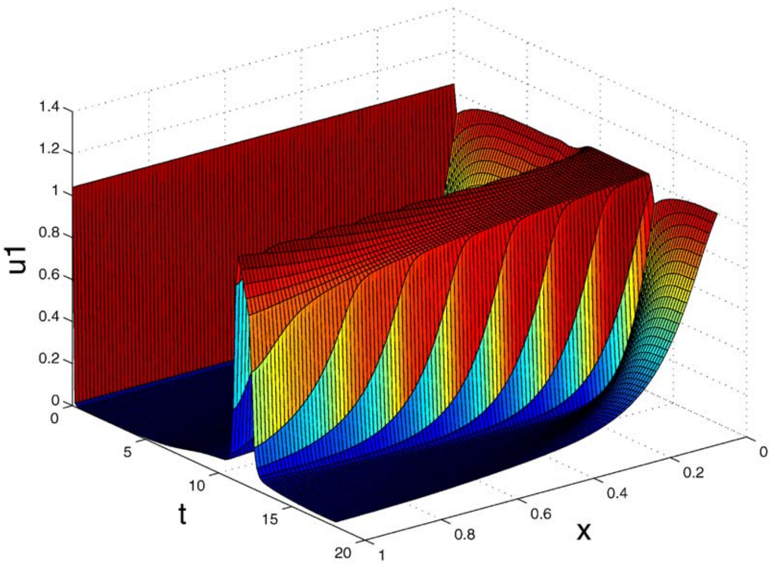

Fig. 12. $u_1(x,t)$ for Problem 5.

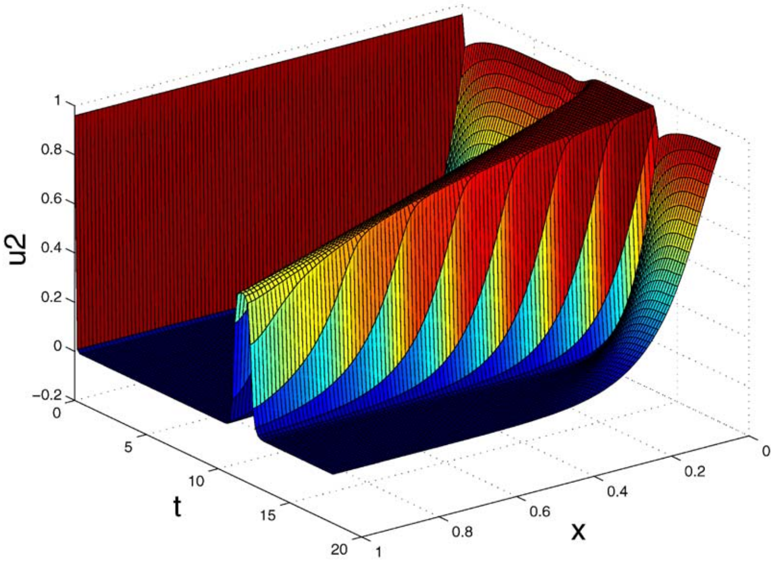

Fig. 13. $u_2(x,t)$ for Problem 5.

We note that some of the boundary conditions for this problem are mixed, i.e., they involve conditions on both solution and first derivative components. Fig. 16 shows the performance of all the codes for this problem. The L^2-norm error is computed at $T_{\text{out}}=18$. We first calculate the L^2-norm

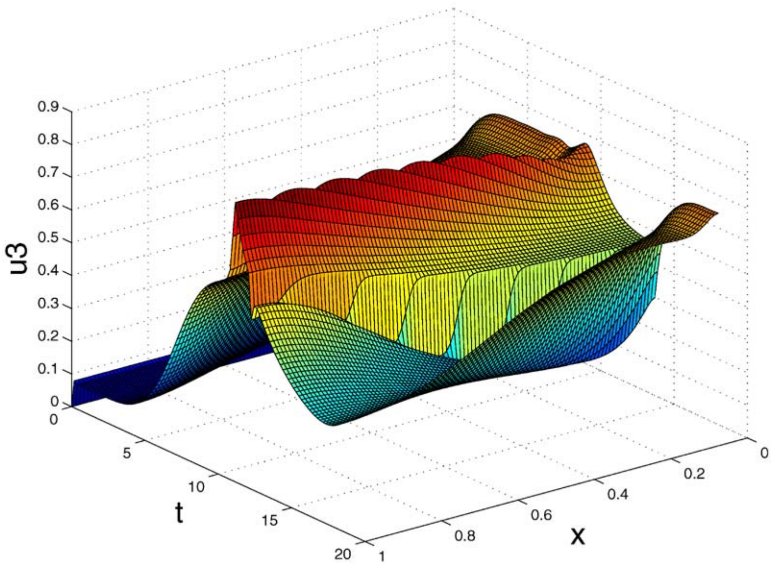

Fig. 14. $u_3(x,t)$ for Problem 5.

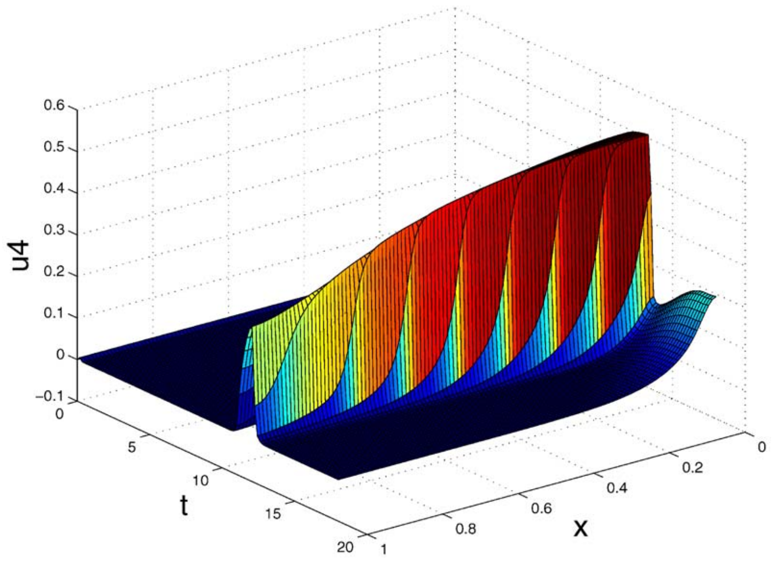

Fig. 15. $u_4(x,t)$ for Problem 5.

error for each PDE component, and then the maximum L^2-norm error over all PDE components is plotted in Fig. 16.

This problem has a smooth solution for most of the temporal domain. However, there is a wave-front moving from left to right between $t = 9$ and 10. This solution behavior can be efficiently

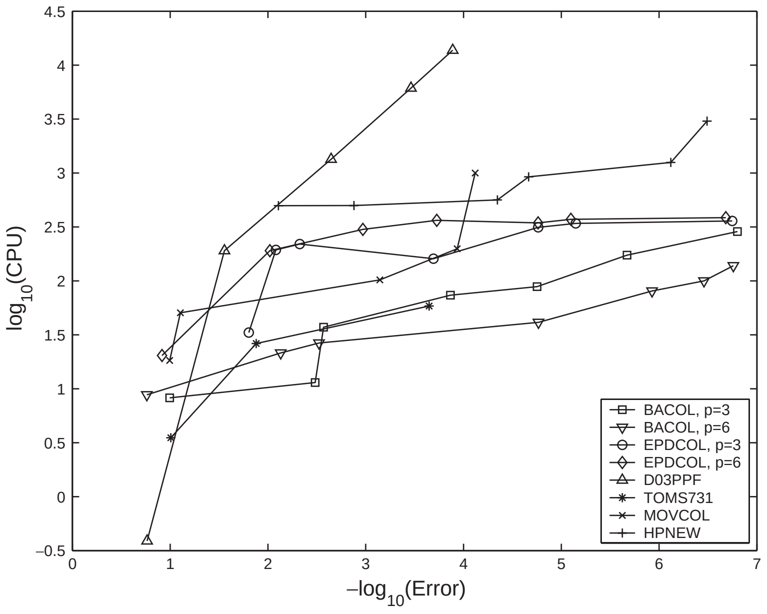

Fig. 16. The CPU time and the maximum L^2-norm error for Problem 5.

treated by BACOL and HPNEW since these codes have the ability to adjust the number of mesh points while the other codes have to use a fixed number of mesh points throughout the computation. Fig. 16 shows that BACOL is again the most efficient, and is comparable with TOMS731 even when a small tolerance is requested. Furthermore only BACOL, EPDCOL, and HPNEW are able to compute a solution for sharper tolerances. D03PPF, TOMS731, and MOVCOL are only able to compute solutions with an error greater than about 10^{-4}. HPNEW performs about as well as the other codes for coarse tolerances, and is less efficient than EPDCOL or BACOL for sharper tolerances. EPDCOL exhibits performance that is better than HPNEW but worse than BACOL. For coarse tolerances TOMS731 has very good performance, while that of MOVCOL is comparable to that of the other codes. D03PPF has the best efficiency for very low accuracy but quickly becomes less efficient as the accuracy request is sharpened.

5. Conclusions

In this paper, BACOL, a spatially and temporally adaptive MOL software package for solving 1D parabolic PDEs is described and compared with a number of other MOL software packages on several test problems. While software for the treatment of one-dimensional, time-dependent PDEs using a method-of-lines approach has been available, in some form, for approximately 30 years, this software was only able to control the temporal error (via the error control of the initial value ODE or DAE solver). Thus, no control of the spatial error was available. To our knowledge, the only software packages which can adapt to the spatial behavior of the solution *and* efficiently control the spatial error in a way that approximately balances it with the temporal error are HPNEW and

BACOL. (The moving mesh codes, such as MOVCOL, can for a given number of mesh points, adapt the location of these points based on solution behavior but they cannot adjust the number of points to control a spatial error estimate.)

The significance of this paper is that it provides a comparison of the only two packages, HPNEW and BACOL, that can provide both spatial and temporal error control, as well as comparing these packages with most of the other currently available packages for the numerical treatment of 1D parabolic PDEs.

The results presented in Section 4 illustrate that when high accuracy is required, BACOL is clearly the most efficient among all the packages. For a coarse tolerance, BACOL is still one of the fastest codes for most test problems. We note that except for BACOL and HPNEW, none of the other codes has the ability to change the number of mesh points and thus control the spatial error. Therefore, in general, for all the codes except BACOL and HPNEW, it would be difficult to achieve the optimal performances for these codes shown in Section 4. Comparing BACOL and HPNEW, BACOL is clearly more efficient than HPNEW for almost all test problems and tolerance ranges.

The amount of storage used by a code is, in some cases, of some concern. Comparing BACOL with EPDCOL, the storage for BACOL is roughly twice that of EPDCOL if the same number of subintervals and the same degree of polynomial is employed. However, in order to achieve the same accuracy for problems having solutions with rapid variation, EPDCOL would need many more subintervals. That is, in order to obtain the same accuracy, EPDCOL actually needs more storage than BACOL. This is also true for other codes which cannot adapt the number of subintervals to the computation. We have done some preliminary analysis of storage using the codes considered in this paper. The storage usage for HPNEW appears to be higher (more than 10 times higher) than that of other codes. We are not able to explain this behavior. Overall, if the same accuracy is required, BACOL tends to use the least amount of storage.

References

[1] S. Adjerid, M. Aiffa, J.E. Flaherty, High-order finite element methods for singularly-perturbed elliptic and parabolic problems, SIAM J. Appl. Math. 55 (1995) 520–543.

[2] S. Adjerid, J.E. Flaherty, P.K. Moore, Y. Wang, High-order adaptive methods for parabolic systems, Physica D 60 (1992) 94–111.

[3] S. Adjerid, J.E. Flaherty, Y. Wang, A posteriori error estimation with finite element methods of lines for one-dimensional parabolic systems, Numer. Math. 65 (1993) 1–21.

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

[5] M. Berzins, P.J. Capon, P.K. Jimack, On spatial adaptivity and interpolation when using the method of lines, Appl. Numer. Math. 26 (1998) 117–133.

[6] M. Berzins, P.M. Dew, Algorithm 690: Chebyshev polynomial software for elliptic-parabolic systems of PDEs, ACM Trans. Math. Software 17 (1991) 178–206.

[7] M. Berzins, P.M. Dew, R.M. Furzeland, Developing software for time-dependent problems using the method of lines and differential-algebraic integrators, Appl. Numer. Math. 5 (1989) 375–397.

[8] M. Berzins, R.M. Furzeland, An adaptive Theta method for the solution of stiff and non-stiff differential equations, Appl. Numer. Math. 9 (1992) 1–19.

[9] M. Bieterman, I. Babuška, The finite element method for parabolic equations. I. A posteriori error estimation, Numer. Math. 40 (1982) 339–371.

[10] M. Bieterman, I. Babuška, The finite element method for parabolic equations. II. A posteriori error estimation and adaptive approach, Numer. Math. 40 (1982) 373–406.

[11] J.G. Blom, J.M. Sanz-Serna, J.G. Verwer, On simple moving grid methods for one-dimensional evolutionary partial differential equations, J. Comput. Phys. 74 (1988) 191–213.
[12] J.G. Blom, P.A. Zegeling, Algorithm 731: a moving-grid interface for systems of one-dimensional time-dependent partial differential equations, ACM Trans. Math. Software 20 (1994) 194–214.
[13] K.E. Brenan, S.L. Campbell, L.R. Petzold, Numerical Solution of Initial-value Problems in Differential-Algebraic Equations, SIAM, Philadephia, 1996.
[14] J. Carroll, A composite integration scheme for the numerical solution of systems of parabolic PDEs in one space dimension, J. Comput. Appl. Math. 46 (1993) 327–343.
[15] D03PPF, NAG Fortran library, Mark 16A. The Numerical Algorithms Group, Ltd. Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, UK.
[16] C. De Boor, Package for calculating with B-splines, SIAM J. Numer. Anal. 14 (1977) 441–472.
[17] C. De Boor, A Practical Guide to Splines, Springer, New York, 1978.
[18] J.C. Diaz, G. Fairweather, P. Keast, FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination, ACM Trans. Math. Software 9 (1983) 358–375.
[19] A.C. Hindmarsh, Preliminary documentation of GEARIB: solution of implicit systems of ordinary differential equations with banded Jacobian, Technical Report, UCID-30130, Lawrence Livermore National Laboratory, 1976.
[20] W. Huang, R.D. Russell, A moving collocation method for solving time dependent partial differential equations, Appl. Numer. Math. 20 (1996) 101–116.
[21] P. Keast, P.H. Muir, Algorithm 688. EPDCOL: a more efficient PDECOL code, ACM Trans. Math. Software 17 (1991) 153–166.
[22] N.K. Madsen, R.F. Sincovec, General software for partial differential equations, in: L. Lapidus, W.E. Schiesser (Eds.), Numerical Methods for Differential Systems, Academic Press, New York, 1976, pp. 229–242.
[23] N.K. Madsen, R.F. Sincovec, Algorithm 540. PDECOL, general collocation software for partial differential equations, ACM Trans. Math. Software 5 (1979) 326–351.
[24] K. Miller, R.N. Miller, Moving finite elements. I, SIAM J. Numer. Anal. 18 (1981) 1019–1032.
[25] P.K. Moore, Comparison of adaptive methods for one dimensional parabolic systems, Appl. Numer. Math. 16 (1995) 471–488.
[26] 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 (2001) 149–177.
[27] L.R. Petzold, A Description of DASSL: A Differential/Algebraic System Solver, Sandia Labs, Livermore, CA, 1982.
[28] R. Wang, High order adaptive collocation software for 1-D parabolic PDEs, Ph.D. Thesis, Dalhousie University, 2002, http://www.mathstat.dal.ca/∼keast/research/grad_super.html.
[29] R. Wang, P. Keast, P.H. Muir, A high-order global spatially adaptive collocation method for 1-D parabolic PDEs, Appl. Numer. Math. (2003), http://www.mathstat.dal.ca/∼keast/research/wang_keast_muir_ANM.pdf.