

BACOL: B-Spline Adaptive COLlocation Software for 1-D Parabolic PDEs

R. WANG and P. KEAST

Dalhousie University

and

P. MUIR

Saint Mary's University

BACOL is a new, high quality, robust software package in Fortran 77 for solving one-dimensional parabolic PDEs, which has been shown to be significantly more efficient than any other widely available software package of the same class (to our knowledge), especially for problems with solutions exhibiting rapid spatial variation. A novel feature of this package is that it employs high order, adaptive methods in both time and space, controlling and balancing both spatial and temporal error estimates. The software implements a spline collocation method at Gaussian points, with a B-spline basis, for the spatial discretization. The time integration is performed using a modification of the popular DAE solver, DASSL. Based on the computation of a second, higher order, global solution, a high quality *a posteriori* spatial error estimate is obtained after each successful time step. The spatial error is controlled by a sophisticated new mesh selection algorithm based on an equidistribution principle. In this article we describe the overall structure of the BACOL package, and in particular the modifications to the DASSL package that improve its performance within BACOL. An example is provided in the online Appendix to illustrate the use of the package.

Categories and Subject Descriptors: G.1.0 [Numerical Analysis]: General—*Numerical algorithms*; G.1.8 [Numerical Analysis]: Partial Differential Equations—*Method of lines, parabolic equations*; G.4 [Mathematics Software]—*Certification and testing, efficiency*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: 1-D Parabolic PDEs, B-splines, collocation, high-order, equidistribution principle, mesh selection

1. INTRODUCTION

In recent years there has been considerable interest in developing method of lines (MOL) packages for the numerical solution of systems of nonlinear

The work of these authors was partially supported by the Natural Sciences and Engineering Research Council of Canada.

Authors' addresses: R. Wang and P. Keast, Department of Mathematics and Statistics, Dalhousie University, Halifax, Nova Scotia B3H 3J5, Canada; email: {wang,keast@mathstat.dal.ca}; P. Muir, Department of Mathematics and Computing Science, Saint Mary's University, Halifax, Nova Scotia B3H 3C3, Canada; email: muir@saintmarys.ca.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
2004 ACM 0098-3500/04/1200-0454 \$5.00

parabolic partial differential equations (PDEs) in one space dimension (e.g. PDECOL [Madsen and Sincovec 1979], EPDCOL [Keast and Muir 1991], PDECHEB [Berzins and Dew 1991], TOMS731 [Blom and Zegeling 1994], MOVCOL [Huang and Russell 1996], and HPNEW [Moore 2001]). Some of these packages implement the standard MOL approach, which employs finite difference, finite element, or collocation methods for the spatial discretization, based on a fixed spatial mesh. The resultant system of ordinary differential equations (ODEs) or differential-algebraic equations (DAEs) is then solved using a standard ODE/DAE solver, for example, DASSL [Petzold 1982]. The success of those codes largely relies on the availability of high-quality ODE and DAE solvers. Modern ODE solvers employ adaptive time integration: variable time step sizes and possibly variation of the order of the integration formula, and attempt to control the temporal error. However, there are very few robust general-purpose MOL packages that use spatially adaptive techniques. Even fewer control both the temporal and spatial errors (some codes apply spatial adaptation, but without spatial error control, e.g., MOVCOL). To our knowledge, the only packages previously reported in the literature that are able to control the error in both space and time using high order discretization schemes, are the codes HPSIRK and HPDASSL [Moore 1995], and HPNEW, a modification of HPDASSL.

In this article, we describe the BACOL (*B*-spline Adaptive *COL*location) software package, a new software package for one-dimensional parabolic PDEs, which uses high order schemes in both time and space, and which provides both spatial and temporal adaptivity and error control. This code employs collocation with a B-spline basis for the spatial discretization. A modification of the widely used initial value DAE solver, DASSL, is used for the time integration of the DAE system arising from the spatial discretization and boundary conditions. A new mesh selection strategy is employed for controlling an estimate of the spatial error, which is balanced with the temporal error. Numerical results based on extensive testing demonstrate that BACOL is generally significantly more efficient than similar packages, especially for problems having solutions with rapid spatial variation. In Wang et al. [2004a] we discussed the algorithms for the spatial discretization and mesh adaptation, while in Wang et al. [2004b] we compared BACOL with several other MOL codes for 1-D parabolic PDEs. In this article we will concentrate on describing some of the software development aspects of the BACOL project. This will include a detailed description of several modifications to DASSL, an overview of the structure of the BACOL package, and a demonstration of its use. The software is based on solid, well established algorithms that have already appeared in the literature. The BACOL package and two sample driving programs can be found at <http://www.mathstat.dal.ca/~keast/research/bacol.html>.

In Section 2 we will briefly review the numerical algorithms employed for the spatial discretization and the remeshing strategy. In Section 3 we consider the implementation of the time integration within DASSL; this will include consideration of the addition of a specialized linear system solver, COLROW, for the efficient treatment of the Newton systems, the introduction of a scaling technique that improves the conditioning of the Newton iteration matrices arising in the treatment of nonlinear systems within DASSL, the modification

of the DASSL subroutine, DANRM, to improve its efficiency, and some other minor modifications to DASSL. This section will also describe how, in order to improve efficiency after a remeshing, we implement a warm start with DASSL and how we obtain consistent initial conditions for DASSL. In Section 4, we will provide an overview of the structure of BACOL and a brief description of the subroutines that make up the package. This section will also discuss the output provided by the package, as well as an overview of its performance. In Section 5, we will give conclusions and suggestions for future work. Finally an example is provided in the online Appendix to illustrate the use of the package.

In this article we will discuss systems of one-dimensional (1-D) parabolic PDEs of the form

$$u_t(x, t) = f(t, x, u(x, t), u_x(x, t), u_{xx}(x, t)), \quad x_a \leq x \leq x_b, \quad t \geq t_0, \quad (1)$$

with the initial conditions

$$u(x, t_0) = u_0(x), \quad x_a \leq x \leq x_b, \quad (2)$$

and separated boundary conditions (BCs)

$$b_L(t, u(x_a, t), u_x(x_a, t)) = 0, \quad b_R(t, u(x_b, t), u_x(x_b, t)) = 0, \quad t \geq t_0, \quad (3)$$

where $u(x, t)$ and $f(t, x, u(x, t), u_x(x, t), u_{xx}(x, t))$ are vectors of dimension $NPDE$, where $NPDE$ 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. We will assume $x_a = 0$, $x_b = 1$, and $t_0 = 0$ to simplify the presentation although BACOL is capable of handling the general domain $[x_a, x_b] \times [t_0, \infty)$.

2. THE SPATIAL DISCRETIZATION AND MESH ADAPTATION TECHNIQUES

2.1 Spatial Discretization

Let the spatial mesh points be an increasing sequence as follows,

$$0 = x_0 < x_1 < \dots < x_N = 1. \quad (4)$$

The approximate solution is computed in a piecewise polynomial subspace of degree p (where p satisfies $3 \leq p \leq 11$ and is a user-supplied parameter in the BACOL package): the solution is approximated using a polynomial of degree p for each subinterval, $[x_{i-1}, x_i]$, $i = 1, \dots, N$. C^1 -continuity at internal mesh points is imposed. Thus the dimension of this piecewise polynomial subspace is given by $NC = N(p - 1) + 2$.

The spatial discretization of the BACOL package is similar to that of PDECOL [Madsen and Sincovec 1979]/EPDCOL [Keast and Muir 1991]. A B-spline basis is employed for the piecewise polynomial subspace using the de Boor B-spline package [de Boor 1977]. Let $\{B_i(x)\}_{i=1}^{NC}$ be the B-spline basis associated with the mesh (4) and having C^1 -continuity (see de Boor [1978] for details). The s -th component of the solution, $u_s(x, t)$, is then approximated by a piecewise polynomial, $U_s(x, t)$ of the form

$$U_s(x, t) = \sum_{m=1}^{NC} B_m(x) y_{m,s}(t), \quad (5)$$

where $y_{m,s}(t)$ represents the (unknown) coefficient of the m -th B-spline basis function for the s -th component of the solution. We choose $p - 1$ Gaussian points in each subinterval as the collocation points; thus the approximate solution (5) is required to satisfy the PDE system (1) at these points. Using the fact that there are at most $p + 1$ of the basis functions with non-zero values in each subinterval [de Boor 1978], substitution of (5) into (1) leads to

$$\sum_{m=(i-1)(p-1)+1}^{i(p-1)+2} B_m(\xi_l) y'_{m,s}(t) = f_s(t, \xi_l, U(\xi_l, t), U_x(\xi_l, t), U_{xx}(\xi_l, t)), \quad (6)$$

where $s = 1, \dots, NPDE$, $l = (i - 1)(p - 1) + j$, $i = 1, \dots, N$, $j = 1, \dots, p - 1$, and ξ_l is the j -th collocation point in the i -th subinterval. We now have an ODE system in which the unknowns, $y_{m,s}(t)$, are the B-spline coefficients for the approximate piecewise polynomial solution.

Unlike the way in which the boundary conditions are handled in PDECOL/EPDCOL, BACOL treats the BCs in their original form. (PDECOL/EPDCOL differentiates the BCs with respect to time in order to obtain a set of ODEs.) A set of algebraic equations is obtained from the BCs in BACOL:

$$b_L(t, U(0, t), U_x(0, t)) = 0, \quad b_R(t, U(1, t), U_x(1, t)) = 0. \quad (7)$$

The ODE system, (6), coupled with the algebraic constraints, (7), gives a system of index-1 DAEs provided that at least one of $\partial b_L/\partial U$ or $\partial b_L/\partial U_x$ and at least one of $\partial b_R/\partial U$ or $\partial b_R/\partial U_x$ are not zero. This requirement ensures that differentiating the algebraic constraints (which come from the boundary conditions) once will lead to ODEs as in Keast and Muir [1991]. We solve this DAE system using a modified version of the well-known DAE software package, DASSL. The implementation of the time integration using DASSL and the modifications to DASSL will be discussed in detail in the next section.

2.2 Adaptive Mesh Refinement

In our spatial mesh refinement strategy, a second piecewise polynomial solution approximation of degree $p + 1$ is simultaneously computed. After each time step, an *a posteriori* spatial error estimate is obtained by comparing these two approximate solutions. We compute a normalized spatial error estimate, E_s , associated with the s -th PDE component over $[0, 1]$, of the form

$$E_s = \sqrt{\int_0^1 \left(\frac{U_s(x, t) - \bar{U}_s(x, t)}{ATOL_s + RTOL_s |U_s(x, t)|} \right)^2 dx}, \quad s = 1, \dots, NPDE, \quad (8)$$

where t is the current time, $ATOL_s$ and $RTOL_s$ denote the absolute and relative tolerances for the s -th 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 s -th PDE component.

A remeshing will be carried out when $E \equiv \max_{s=1}^{NPDE} E_s \geq 1$ or the mesh is not well-distributed—when the normalized error estimates associated with each subinterval are not roughly the same size. \hat{E}_i , the normalized error estimate

for the i -th subinterval over all $NPDE$ components, has the form

$$\hat{E}_i = \sqrt{\sum_{s=1}^{NPDE} \int_{x_{i-1}}^{x_i} \left(\frac{U_s(x, t) - \bar{U}_s(x, t)}{ATOL_s + RTOL_s |U_s(x, t)|} \right)^2 dx}, \quad i = 1, \dots, N. \quad (9)$$

Prior to remeshing, the number of subintervals, N^* , needed for the new mesh, is predicted from

$$N^* = N \left(\frac{E}{var} \right)^{1/(p+1)}, \quad (10)$$

where N is the number of subintervals in the old mesh, and the constant var is chosen empirically to be 0.2 in BACOL. Theoretical justification of (10) can be found in Wang et al. [2004b]. In contrast to the local refinement as employed, for example, by Adjrid et al. [1992], BACOL employs a global mesh refinement strategy (similar to the scheme in COLSYS [Ascher et al. 1981]) based on an equidistribution principle. The new mesh points, $\{x_i^*\}_{i=1}^{N^*}$, are chosen to satisfy

$$\sqrt{\sum_{s=1}^{NPDE} \int_{x_{i-1}^*}^{x_i^*} \left(\frac{U_s(x, t) - \bar{U}_s(x, t)}{ATOL_s + RTOL_s |U_s(x, t)|} \right)^2 dx} = \left(\frac{\sum_{i=1}^N \hat{E}_i^{1/(p+1)}}{N^*} \right)^{p+1},$$

where the right hand side is a constant. That is, the normalized error estimate for each new subinterval is the same. After the new mesh is computed, BACOL will repeat the current time step using a *warm start*, as suggested in Berzins et al. [1998], which will be discussed in more detail in Section 3.3.

BACOL is designed to control both the spatial and temporal errors. The most efficient approach is to integrate the DAE system with a given temporal tolerance such that the temporal error is roughly the same size as the spatial error. On the other hand, a tighter tolerance must be employed on the temporal error to ensure that the temporal error does not corrupt the spatial error estimate. However, as discussed in Brenan et al. [1996], DASSL controls the temporal error by requiring the temporal error estimate to be less than 1/3 of the temporal tolerance given to the code. Thus we set the absolute and relative tolerances for the spatial error control to be equal to those for the temporal error control. This ensures that the temporal error will be slightly less than the spatial error.

As mentioned earlier, there are two sets of DAEs resulting from spatial discretizations based on piecewise polynomials of degree p and $p + 1$ respectively. It would be natural to treat these two DAE systems separately. However, this would require running two instances 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 (8)), interpolation of one of the solutions would be unavoidable. This would considerably increase the complexity of the algorithm and also contribute to the overall error. A more serious difficulty is that this would make it harder 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, again significantly contributing to the overall 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. Because the DAEs associated with the two spatial discretizations are decoupled, it is however possible, with a careful treatment of the linear algebra computation, to achieve a very efficient implementation of this “double” DAE system; we discuss this further in Section 3.1.

3. IMPLEMENTATION OF THE TIME INTEGRATION

The most attractive feature of the standard MOL approach is that one can use a high quality and high order time integrator for the solution of the DAE system generated from the spatial discretization. The backward differentiation formulas (BDF) [Brenan et al. 1996] are the most popular linear implicit multi-step methods for DAEs. DASSL, uses a fixed leading coefficient implementation [Brenan et al. 1996] of the BDF methods. In BACOL, the time integration is performed using a modified version of DASSL. We now describe the modifications.

3.1 Addition of the COLROW Linear Algebra Package

Since a major computational effort is associated with the linear systems that arise within the Newton iterations performed by DASSL, it is important to employ a linear system solver that takes advantage of any special structure possessed by the Newton matrices. We note that there is a close relationship between the structure of the Jacobian matrices in BACOL and those in EPDCOL. A modification to DASSL, similar to that made to PDECOL in order to obtain EPDCOL, provides an efficient way of solving this type of linear system.

DASSL solves DAE systems of the form

$$F(t, y, y') = 0. \quad (11)$$

using a k -step BDF method, which leads to the approximation of (11) by

$$F\left(t_{n+1}, y_{n+1}, \frac{\alpha}{h_{n+1}} y_{n+1} + \beta\right) = 0,$$

where α and β are known. The unknown solution value, y_{n+1} , is obtained by using a Newton iteration of the form

$$G\left(y_{n+1}^{(m+1)} - y_{n+1}^{(m)}\right) = -F\left(t_{n+1}, y_{n+1}^{(m)}, \frac{\alpha}{h_{n+1}} y_{n+1}^{(m)} + \beta\right),$$

where $h_{n+1} = t_{n+1} - t_n$, $y_{n+1}^{(m)}$ is the m -th Newton approximation to y_{n+1} and G is an iteration matrix of the form,

$$G = \frac{\alpha}{h_{n+1}} \frac{\partial F}{\partial y'} + \frac{\partial F}{\partial y}. \quad (12)$$

Both $\frac{\partial F}{\partial y'}$ and $\frac{\partial F}{\partial y}$ have a matrix structure that is called almost block diagonal (ABD) [Diaz et al. 1983]. The matrix G in (12) is therefore also of ABD form.

DASSL offers two options for the solution of linear systems, a full matrix solver and a banded matrix solver. However, the linear system solver, COLROW [Diaz et al. 1983], is significantly more efficient in treating the ABD systems than a band solver and also does not introduce fill-in of the matrix structure,

which implies that no extra storage is required. We have therefore introduced a new linear system solver option within DASSL to allow it to employ COLROW for the efficient treatment of the Newton systems that arise in our context.

As mentioned previously, BACOL uses two different spatial discretizations, one based on piecewise polynomials of degree p , and the other based on degree $p + 1$, in order to obtain an error estimate. As explained in Section 2.2, the two sets of DAEs are combined into one large set of DAEs. The Jacobian associated with these equations consists of two *decoupled* ABD matrices. Consequently we can treat the linear systems by solving the two ABD linear systems independently, calling COLROW twice for each full Newton iteration.

3.2 Improvement of the Conditioning of the Jacobian matrices

We know from Brenan et al. [1996] that the condition number of the iteration matrix for a DAE system with index γ is $O((h_{time})^{-\gamma})$, where h_{time} is the current stepsize in the time integration, and therefore, when the time stepsize is small, the Jacobian matrix will have a large condition number. In DASSL, the initial stepsize is chosen to be $O(TOL)$, where TOL is the user-defined tolerance. We therefore see that a poorly conditioned Jacobian will arise at the first step of the time integration if the user supplies a sharp tolerance. This will sometimes lead to failure of the associated Newton iteration. After such an unsuccessful step, DASSL will respond by reducing the stepsize, which leads to a Jacobian matrix with an even larger condition number.

In Petzold and Lötstedt [1986] a scaling technique to overcome this difficulty is suggested. The same approach is used in SPRINT [Berzins et al. 1989]. Employing this technique within BACOL involves scaling the Jacobian subblocks and right hand side elements associated with the boundary conditions by $1/h_{time}$. We note that, as a result, each of the decoupled ABD matrices becomes the same as the Jacobian matrix in EPDCOL [Keast and Muir 1991]. We know the Jacobian matrix in EPDCOL has a better condition number because EPDCOL generates a pure ODE system an index-0 DAE system, after the spatial discretization. (Recall that the boundary conditions are differentiated in EPDCOL.) We have verified through numerical experiments that this scaling technique significantly reduces the condition number of the Jacobian matrix. A typical result is as follows. We considered the sample problem given in Section 4.5 of this article, with $\epsilon = 10^{-2}$, absolute and relative tolerances of 10^{-5} , $p = 3$, and an initial uniform mesh of 10 subintervals. In this case, we found that at $t = 0$ the condition number of the Jacobian matrix without scaling is approximately 7×10^5 , while the scaled Jacobian matrix has a condition number of about 5.

It is possible for DASSL to use the same Jacobian matrix (already decomposed in LU form) even when the stepsize has changed slightly. Since we save the stepsize used in the scaling of the Jacobian, we can use it to perform scaling over the corresponding components of the right-hand side in cases where the current time stepsize differs from that used in the scaling of the Jacobian. Three subroutines, DAINI, DASTP and DASLV, are modified to implement this scaling process.

3.3 Efficient Restarting of DASSL after a Remeshing: A Warm Start

After a remeshing, a new DAE system is obtained and it is then necessary to continue the time integration starting from this point in time. The simplest approach is called a *cold start*. In this approach, one interpolates the solution from the old mesh to the new mesh at the current step, and then restarts the integration as though solving a new problem—with a low order method and a small time step. This is obviously very inefficient. The preferred approach is called a *warm start*. In this approach, the same stepsize and order as for the last time step are then tried again. As suggested in Berzins et al. [1998], one should interpolate solution information from the old mesh to the new mesh, for as many previous steps as necessary. This means interpolating all the history vectors that are required by DASSL. Computational results in Berzins et al. [1998] support the conclusion that the use of a warm start can significantly improve efficiency. In order to perform a warm start, we need to employ interpolation at the current step and at most the last 5 time steps (because DASSL implements BDF methods up to a maximum order of 5).

For the current spatial mesh, let $U(x, t)$ and $\bar{U}(x, t)$ denote the two approximate solutions computed by BACOL, using polynomials of degree p and $p + 1$ respectively. Let $U^*(x, t)$ and $\bar{U}^*(x, t)$ be the corresponding approximate solutions on the new mesh. Since $\bar{U}(x, t)$ generally gives a more accurate approximation, we obtain approximate values for $U^*(x, t)$ and $\bar{U}^*(x, t)$ on the new mesh by evaluating $\bar{U}(x, t)$. That is, we obtain approximate values for $U^*(x, t)$ and $\bar{U}^*(x, t)$ on the new mesh by evaluating $\bar{U}(x, t)$, the higher order solution approximation associated with the current mesh:

$$U^*(\xi_l^*, t_{n-i}) = \bar{U}(\xi_l^*, t_{n-i}), \quad l = 1, \dots, NC^*, \quad (13)$$

$$\bar{U}^*(\bar{\xi}_l^*, t_{n-i}) = \bar{U}(\bar{\xi}_l^*, t_{n-i}), \quad l = 1, \dots, \bar{NC}^*, \quad (14)$$

where $i = 0, \dots, k$, k is the order of the BDF method, NC^* is the number of new collocation points, and ξ_l^* , $l = 1, \dots, NC^*$, are the new collocation points. Note that, from (5), the left hand sides of (13) and (14) involve the coefficients of the B-spline basis functions over the new mesh, $\{y_m^*\}_{m=1}^{NC^*}$ and $\{\bar{y}_m^*\}_{m=1}^{\bar{NC}^*}$, and (13) and (14) therefore represent linear systems that are solved to provide values for these coefficients.

3.4 Consistent Initial Conditions for DASSL

As discussed in Leimkuhler et al. [1991], small inconsistencies in the initial values may cause DASSL to fail on the first step or to become very inefficient: a very small stepsize is used for the first few steps after several failed attempted steps. In order to be consistent, the initial conditions of an index-1 DAE, must satisfy not only the algebraic constraints, but also the differentiated algebraic constraints [Leimkuhler et al. 1991]. Therefore, we require that the B-spline coefficients $y_{m,s}(0)$, $m = 1, \dots, NC$, $s = 1, \dots, NPDE$, satisfy the BCs and, at the internal collocation points ξ_l , $l = 2, \dots, NC - 1$, we require that the

piecewise polynomial agrees with the initial conditions of the PDE system:

$$b_L(0, U(0, 0), U_x(0, 0)) = 0, \quad (15)$$

$$U(\xi_l, 0) = u_0(\xi_l), \quad l = 2, \dots, NC - 1, \quad (16)$$

$$b_R(0, U(1, 0), U_x(1, 0)) = 0. \quad (17)$$

We note that Newton iterations are needed in order to solve the system of algebraic equations unless Dirichlet BCs are applied on both boundaries. The initial guess for the Newton iteration is obtained by solving the linear system, $U(\xi_l, 0) = u_0(\xi_l)$, $l = 1, \dots, NC$, that is, we require that the piecewise polynomial agrees with the initial conditions of the PDEs on both boundaries.

Once we obtain $y(0)$, the value $y'(0)$ is obtained by solving

$$\frac{d}{dt}b_L(0, U(0, 0), U_x(0, 0)) = 0, \quad (18)$$

$$U_t(\xi_l, 0) = f(0, \xi_l, U(\xi_l, 0), U_x(\xi_l, 0), U_{xx}(\xi_l, 0)), \quad (19)$$

$$l = 2, \dots, NC - 1,$$

$$\frac{d}{dt}b_R(0, U(1, 0), U_x(1, 0)) = 0. \quad (20)$$

We see that the differentiated BCs are included in the above equations, which ensures consistent initial conditions for DASSL.

Note that this approach is different from the one in PDECOL/EPDCOL. In those codes, the initial conditions for the ODE system (we recall that PDECOL/EPDCOL differentiates the BCs) must be satisfied not only at the internal collocation points, but also at boundary points. Therefore, if any of the BCs are not Dirichlet, the initial conditions for the piecewise polynomial approximation may not satisfy the BCs exactly.

As mentioned in Madsen and Sincovec [1979], one of the weaknesses of PDECOL/EPDCOL is its inability to solve problems whose BCs are inconsistent with the initial conditions. However, an example in Wang [2002] is provided to demonstrate that, by using (15)–(17) to obtain the initial conditions for the DAEs, BACOL is able to deal with problems whose BCs are inconsistent with the initial conditions, while EPDCOL will lead to incorrect solutions.

3.5 Other Modifications to DASSL

In addition to the modifications mentioned above, we have made some further modifications to DASSL. First, during our numerical experiments, we found that DANRM, a subroutine in DASSL, which computes the norm of a vector, represented about 20% of the total running time. Within this routine, the quantity $V(I)/WT(I)$ is computed several times, for each loop iteration, where $I = 1, \dots, NEQ$, the number of DAEs. If we employ a vector of length NEQ to serve as storage for the $V(I)/WT(I)$ values, we do not need to calculate these values more than once. We therefore modified DANRM to use a part of *RPAR*, which is a parameter array input to DANRM, to save the $V(I)/WT(I)$ values. Our numerical experiments showed that this saves about 25% in DANRM and about 5% in the total running time.

Second, within DASSL there is an option for specifying an absolute tolerance, *ATOL*, and a relative tolerance, *RTOL* as vectors, in which case each of them must have the dimension $NC * NPDE$ —the number of DAEs. This flexibility is unnecessary for a MOL code, which requires the same tolerance on all equations corresponding to the same PDE component. Therefore, we have changed the dimensions of *ATOL* and *RTOL* to be *NPDE* in the case where they are to be vectors. This modification is limited to the routines, DASSL and DAWTS.

Third, if we decide to perform a warm start after a remeshing, we first discard the current step and go back to the last accepted step. We do not allow DASSL to increase the stepsize for the next attempt. (This is similar to the standard strategy in the ODE case after a failed time step; see, for example, Brenan et al. [1996].) If the predicted stepsize is greater than the previous one, we set it equal to the previous stepsize. The philosophy is that we want to be conservative in the first step after a remeshing. Based on the same reasoning, the order of the BDF method that is employed in the first step after a remeshing is not allowed to be greater than the order used in the previous successful step.

DASSL was not originally developed specifically for MOL codes. Thus it is possible that some parts of DASSL may not be optimal if the number of ODEs or DAEs is large, as is the case when the number of subintervals is large. This is an open question, which deserves attention in the future.

4. DESCRIPTION OF BACOL

In this section we first describe the structure of the BACOL software. Then we will give a brief description of the subroutines included in BACOL. This will be followed by a description of the subroutines that must be provided by the user in order to provide the problem description. Finally we discuss the output from BACOL and an overview of the performance of the package. A sample driver program, as well as the description of the parameter list of two subroutines, BACOL and VALUES, is presented in the online Appendix.

4.1 Structure of BACOL

The way in which all the subroutines in the BACOL software work together is illustrated in Figure 1. Here we let MAIN denote the user's driver program; it is inside the rectangle constructed with dashed lines. Any subroutine inside a single rectangle constructed with solid lines represents software developed by others. Any subroutine inside double rectangles constructed with solid lines represents new software developed as part of the BACOL package. The notation, $A \rightarrow B$, means subroutine *A* calls subroutine *B*.

4.2 A Description of the Subroutines in BACOL

We will now describe all the subroutines in BACOL. As indicated earlier, a number of the important computational tasks performed in BACOL are implemented using previously existing packages. Therefore we will first introduce the subroutines that are new. The previously existing packages employed in BACOL will be considered afterwards.

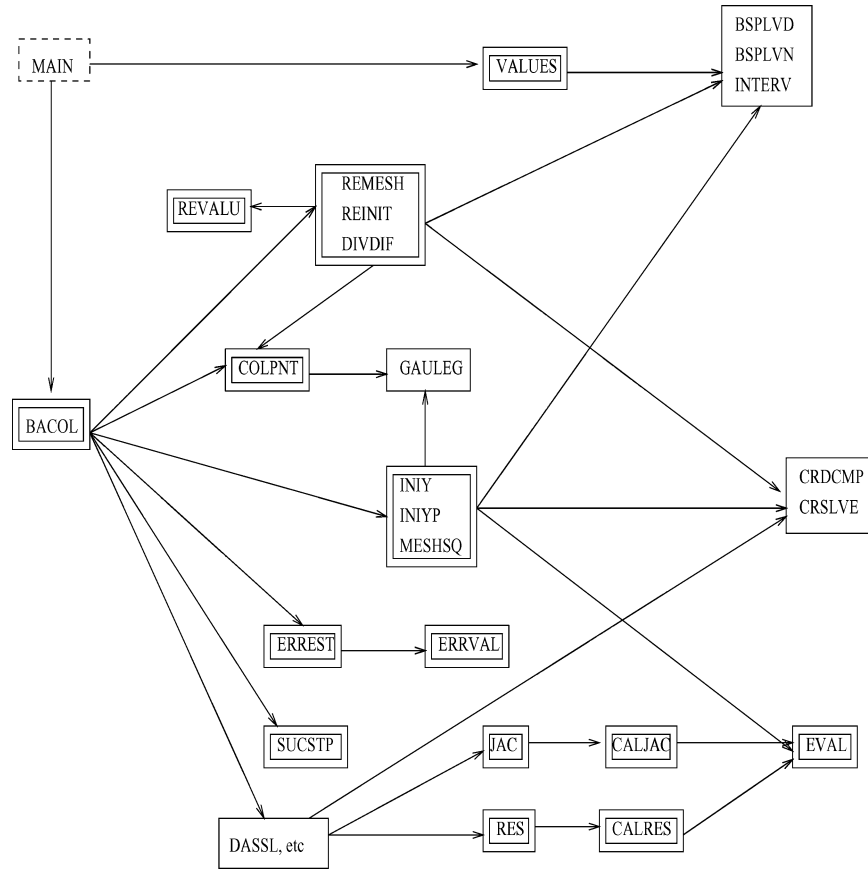


Fig. 1. Structure of the BACOL software.

The newly developed subroutines are as follows:

BACOL This routine is the driver for the entire package. It first checks the user's input for legality and performs initialization tasks such as setting parameters and allocating and defining the locations and lengths of the storage array required by the package. It also calls COLPNT, INIY, INIYP, and MESHSQ to complete the initialization. After that, BACOL makes repeated calls to the core time integrator DASSL in order to take time steps. After each step, the spatial error estimate is computed by calling ERREST. If the spatial estimate is satisfactory then this step will be accepted and SUCSTP will be called to update the current information in case there is a remeshing in the future (see below). On the other hand, if the spatial error estimate is unsatisfactory, this step will be rejected and BACOL will then call REMESH, REINIT, and DIVDIF to carry out a remeshing. After a remeshing, BACOL will call DASSL again, beginning at the last successful step. This process is repeated until a time step is accepted or an error condition occurs, that is, BACOL is unable to continue the computation. When the output time T_{out} is reached, BACOL returns the B-spline coefficients, which are used by the

- VALUES** routine to calculate the values of $U(x, T_{out})$ for any x , $x_a \leq x \leq x_b$.
- COLPNT** This subroutine calculates the Gaussian points for the collocation point sequence.
- ERREST** This subroutine computes the normalized spatial error estimate. It also decides whether a remeshing is necessary or not. It calls **ERRVAL**.
- ERRVAL** This subroutine computes approximate solution values at the Gaussian points when a Gaussian quadrature rule is applied to obtain the values of the integral in the formulas for the spatial error estimates.
- EVAL** The purpose of this routine is to evaluate the piecewise polynomial functions and their first and second derivatives at a given collocation point. The difference between **VALUES** and **EVAL** is that **VALUES** is able to compute the piecewise polynomial solution at *any* point or points and for *any* derivatives that are less than or equal to p . Furthermore, **VALUES** can also estimate these values at previous time steps while **EVAL** can only calculate the values for the current step. However, **EVAL** is able to perform this task more efficiently than **VALUES**.
- INITY, INIYP, MESHSQ** The main purposes of these subroutines are to determine consistent initial conditions for **DASSL**, to prepare for the calculation of the Newton iteration matrix, and to prepare for the calculation of a future spatial error estimate.
- JAC, CALJAC** These two subroutines compute the Jacobian matrices. We recall that two ABD matrices are combined in a decoupled form to represent the Jacobian matrix. The **JAC** routine calls **CALJAC** twice, once for each of the ABD matrices.
- REMESH, REINIT, DIVDIF** These subroutines implement the remeshing process, a very important component of the **BACOL** package. **REMESH** generates a new mesh by equidistributing the error measure. **REINIT** prepares for a warm start (or cold start, if necessary) after a remeshing. **DIVDIF** generates the divided differences that are required by **DASSL**.
- RES, CALRES** These two subroutines compute the residual of a Newton system as required by **DASSL**. For the same reason as in the **JAC** routine, **RES** calls **CALRES** twice, and each time **CALRES** computes the residual corresponding to one of the ABD matrices.
- REVALU** During the remeshing process, **REVALU** is called by **REINIT** to obtain the values at the new collocation points at the current step and, if necessary, at previous steps. This subroutine calls the B-spline subroutines **BSPLVD** and **INTERV**.
- SUCSTP** After each accepted time step, this subroutine stores the information that is necessary if a remeshing is required at a later step.
- VALUES** This is the subroutine that the user must call in order to obtain the values of the approximate solution and derivative values at T_{out} and at any given spatial points.

We now briefly describe the subroutines that are included in **BACOL** but were developed by others.

BSPLVD, BSPLVN, INTERV These subroutines, part of the B-spline package developed by de Boor [1977, 1978], are used to compute the values of the B-spline basis functions and their derivatives at any desired points.

CRDCMP, CRSLVE These subroutines comprise the COLROW package written by Diaz et al. [1983]. CRDCMP is used to factor the ABD matrix and CRSLVE is used to solve the corresponding linear system.

DASSL The DASSL package, developed by Petzold [1982], is the core time integrator for the BACOL package. As discussed in Section 3, we have made some modifications to it.

GAULEG This subroutine was developed by Keast and used in MSCPDE [Nokonechny et al. 1996]. It calculates the points and weights for a Gauss-Legendre or Gauss-Lobatto quadrature rule over the interval $[-1, 1]$ or $[0, 1]$, by calculating the eigenvalues and eigenvectors of tri-diagonal matrices. It is called by COLPNT and MESHQS. GAULEG uses the three EISPACK routines IMQTL1 and IMQTL2 to compute the eigenvalues and eigenvector and PYTHAG to compute the Euclidean norm.

4.3 User Supplied Subroutines

The user is required to provide seven subroutines, which define the form of the PDE problem. For given input values of x and t and corresponding input values of $U(x, t)$, $U_x(x, t)$, and $U_{xx}(x, t)$, the subroutines do the following tasks:

BNDXA, BNDXB These subroutines compute the boundary conditions that are needed in the calculation of the Newton residuals; i.e., $b_L(t, U(x_a, t), U_x(x_a, t))$ at the left boundary point x_a , and $b_R(t, U(x_b, t), U_x(x_b, t))$ at the right boundary point x_b .

DIFBXA, DIFBXB These subroutines are used to define the differentiated boundary conditions. DIFBXA computes $\partial b_L/\partial U$ and $\partial b_L/\partial U_x$, which are necessary for computing the analytic Jacobian matrix, and $\partial b_L/\partial t$, which is necessary for calculating consistent initial conditions. DIFBXB computes $\partial b_R/\partial U$, $\partial b_R/\partial U_x$, and $\partial b_R/\partial t$, the corresponding values associated with the right boundary.

DERIVF This subroutine is used to provide the information to form the analytic Jacobian matrix for the DAE system. (BACOL currently requires that this analytic Jacobian information be provided by the user.) It computes $\partial f/\partial U$, $\partial f/\partial U_x$, and $\partial f/\partial U_{xx}$.

F This subroutine computes the values, $f_s(t, x, U(x, t), U_x(x, t), U_{xx}(x, t))$, $s = 1, \dots, NPDE$, representing the right-hand side of (1). It is needed in the computation of the Newton residuals, and for setting up the collocation equations.

UINIT This subroutine computes the initial values $U(x, t_0)$ for any given x .

We refer the reader to the online Appendix of this article where a sample problem is treated using BACOL and for which all the above user supplied routines are given.

4.4 Output from BACOL

The number and locations of the mesh points used at the requested output time T_{out} , as well as the corresponding B-spline coefficients are provided on a successful return from BACOL. At the requested output time, the user can obtain the values of the approximate solution and derivative values at any points in the spatial domain by calling the subroutine VALUES. However, in some cases, an advanced user may want to obtain more information during the computation. If the user wishes, BACOL can also return after a certain number of accepted time steps. This is a good way to proceed if the user wants to monitor the behavior of the solution over a range of time. If the user only wants the solution at T_{out} , she or he should set $MFLAG(4) = 0$; otherwise, she or he should set $MFLAG(4) = 1$ and assign a positive integer for $IPAR(8)$ to define the number of time steps before BACOL returns. For example, setting $MFLAG(4) = 1$ and $IPAR(8) = 1$ will allow the user to observe the behavior of the solution after each accepted step.

An error message is given when there is an unsuccessful return from BACOL. There are several possible errors. First, an error message can be issued if the input parameters are incorrect, for example the initial mesh is not given in an increasing order. Second, DASSL can issue an error message if the time integration fails before reaching the desired output time. Third, BACOL will give an error message if there is a difficulty in the remeshing process, for example BACOL will return unsuccessfully if it has attempted 20 remeshings at the same time and fails to pass the spatial error test. If such an error message is given, the user is recommended to run BACOL with a finer initial mesh or with a different tolerance.

4.5 Numerical Comparisons

Extensive numerical comparisons with several existing software packages were reported in Wang et al. [2004a]. In this section we give results using the following problem.

$$\begin{aligned} u_t &= -uu_x + \epsilon u_{xx}, & 0 < x < 1, \quad t > 0, \\ u(x, 0) &= 0.5 - 0.5 \tanh\left(\frac{1}{4\epsilon}(x - 0.25)\right), & 0 \leq x \leq 1, \\ u(0, t) &= 0.5 - 0.5 \tanh\left(\frac{1}{4\epsilon}(-0.5t - 0.25)\right), & t \geq 0, \\ u(1, t) &= 0.5 - 0.5 \tanh\left(\frac{1}{4\epsilon}(0.75 - 0.5t)\right), & t \geq 0, \end{aligned}$$

where $\epsilon = 10^{-3}$. The exact solution is

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

It is plotted in Figure 2 for $\epsilon = 10^{-4}$, $0 \leq t \leq 1$, $0 \leq x \leq 1$.

The exact solution of this problem has a sharp wavefront with thickness ϵ , which moves from right to left as t increases. The other software packages used

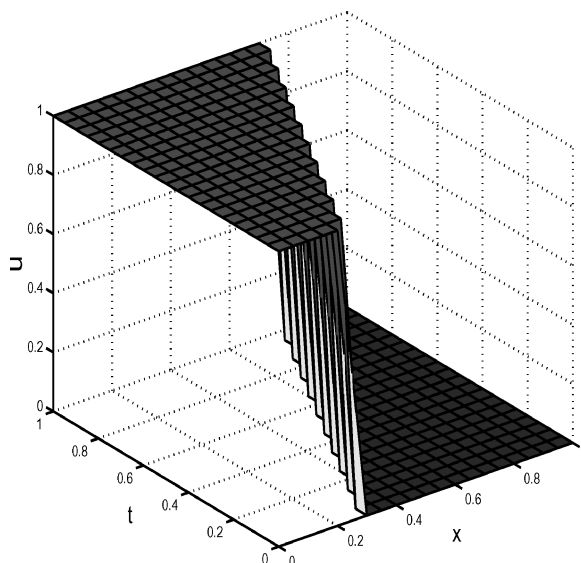


Fig. 2. $u(x, t)$ for $\epsilon = 10^{-4}$.

(described in Wang et al. [2004a]) were: EPDCOL [Madsen and Sincovec 1979], D03PPF from the NAG library, TOMS731 [Blom and Zegeling 1994], MOVCOL [Huang and Russell 1996], and HPNEW [Moore 2001], all of which are based on the method of lines. The basis of comparison was the time required to achieve a particular L^2 -norm error. The first 4 packages do not have the facility to add new points in the way that HPNEW and BACOL do. For these packages, therefore, we experimented with the value of N until we found a value for which the requested error was given. Then the CPU time was measured, using that value of N only. In some cases these four routines were not able, in any reasonable time, to get a sufficiently accurate solution. In Figure 3 we plot the L^2 -error at $t = 1$ for the six codes running on the above problem with $\epsilon = 10^{-4}$.

The results are typical of those given in Wang et al. [2004a] for several other problems. Where the plot for a particular code ends, this corresponds to the point after which the execution time for that code became too large. As the graph shows, only EPDCOL, with degree $p = 6$, and BACOL, with degree 3 and 6, can produce errors smaller than about 10^{-7} in a reasonable time. For coarse tolerances, up to 10^{-3} or so, TOMS731 and HPNEW are more efficient, but both take much more time than BACOL for smaller tolerances.

In general, BACOL works more efficiently than the other codes for tolerances below 10^{-3} , particularly on harder problems with, for example, sharp peaks or wave fronts. The reader is referred to Wang et al. [2004a] for more extensive comparisons.

5. CONCLUSIONS AND FUTURE WORK

BACOL is a software package designed to solve second order parabolic partial differential equations in one spatial dimension. It employs the adaptive

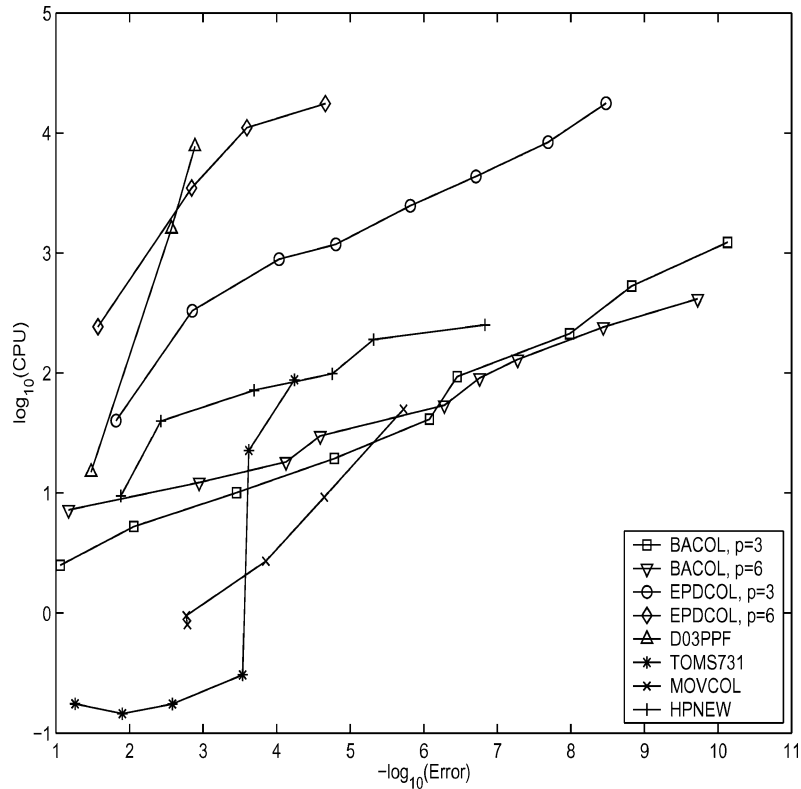


Fig. 3. The CPU time and the L^2 -norm error with $\epsilon = 10^{-4}$.

method of lines approach, and provides both spatial and temporal error control and adaptivity. Numerical results for a wide range of test problems show that among MOL-based packages for 1-D parabolic problems, BACOL is the most efficient, especially when high accuracy is required and the problem is difficult. In this article, we have provided a detailed description of the BACOL package, including an overview of the structure of the package. We have given a detailed description of the modifications to the well-known DAE solver, DASSL, that were required in order to allow it to perform efficiently within the context of a method of lines algorithm. An example illustrating the use of BACOL is provided in the online Appendix.

There are several possibilities for future work. One project would be to develop an algorithm to vary the degrees of the piecewise polynomials employed in the spatial discretization. As shown in Wang et al. [2004b], it is generally more efficient to apply a higher order discretization when a higher accuracy is required. A second research direction is based on the observation that BACOL calculates a second, global approximate solution in order to obtain an a posteriori error estimate. This takes about half of the total computation effort. It would be preferable for an error estimate (obtained, for example, through a local computation) to require only a small fraction of the solution cost. A third research direction is to extend our approach to parabolic equations coupled with elliptic

or algebraic equations. A fourth research direction is to use a Runge-Kutta integrator instead of DASSL. It is then not necessary to employ a warm start, which will save significant computation time during the remeshing process.

The Appendix is available in the ACM Digital Library.

REFERENCES

- ADJERID, S., FLAHERTY, J. E., MOORE, P. K., AND WANG, Y. 1992. High-order adaptive methods for parabolic systems. *Physica D* 60, 94–111.
- ASCHER, U., CHRISTIANSEN, J., AND RUSSELL, R. D. 1981. Collocation software for boundary value ODEs. *ACM Trans. Math. Softw.* 7, 209–222.
- BERZINS, M., CAPON, P. J., AND JIMACK, P. K. 1998. On spatial adaptivity and interpolation when using the method of lines. *Appl. Num. Math.* 26, 117–133.
- BERZINS, M. AND DEW, P. M. 1991. Algorithm 690: Chebyshev polynomial software for elliptic-parabolic systems of PDEs. *ACM Trans. Math. Softw.* 17, 178–206.
- BERZINS, M., DEW, P. M., AND FURZELAND, R. M. 1989. Developing software for time-dependent problems using the method of lines and differential-algebraic integrators. *Appl. Num. Math.* 5, 375–397.
- BLOM, J. AND ZEGELING, P. 1994. Algorithm 731: A moving-grid interface for systems of one-dimensional time-dependent partial differential equations. *ACM Trans. Math. Softw.* 20, 194–214.
- BRENAN, K. E., CAMPBELL, S. L., AND PETZOLD, L. R. 1996. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. SIAM, Philadelphia.
- DE BOOR, C. 1977. Package for calculating with B-Splines. *SIAM J. Numer. Anal.* 14, 441–472.
- DE BOOR, C. 1978. *A Practical Guide to Splines*. Springer-Verlag, New York.
- DIAZ, J. C., FAIRWEATHER, G., AND KEAST, P. 1983. FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination. *ACM Trans. Math. Softw.* 9, 358–375.
- HUANG, W. AND RUSSELL, R. D. 1996. A moving collocation method for solving time dependent partial differential equations. *Appl. Numer. Math.* 20, 101–116.
- KEAST, P. AND MUIR, P. H. 1991. Algorithm 688. EPDCOL: A more efficient PDECOL code. *ACM Trans. Math. Softw.* 17, 153–166.
- LEIMKUEHLER, B., PETZOLD, L., AND GEAR, C. W. 1991. Approximation methods for the consistent initialization of differential-algebraic equations. *SIAM J. Numer. Anal.* 28, 205–226.
- MADSEN, N. K. AND SINCOVEC, R. F. 1979. Algorithm 540. PDECOL, general collocation software for partial differential equations. *ACM Trans. Math. Softw.* 5, 326–351.
- MOORE, P. K. 1995. Comparison of adaptive methods for one dimensional parabolic systems. *Appl. Numer. Math.* 16, 471–488.
- MOORE, P. K. 2001. Interpolation error-based a posteriori error estimation for two-point boundary value problems and parabolic equations in one space dimension. *Numer. Math.* 90, 149–177.
- NOKONECHNY, T. B., KEAST, P., AND MUIR, P. H. 1996. A method of lines package, based on monomial spline collocation, for systems of one dimensional parabolic differential equations. In *Numerical Analysis (A.R. Mitchell 75th birthday volume)*. World Scientific Publishing, Singapore, 207–224.
- PETZOLD, L. R. 1986. A description of DASSL: A differential/algebraic system solver. Tech. rep., Sandia Labs, Livermore, CA.
- PETZOLD, L. R. AND LÖTSTEDT, P. 1986. Numerical solution of nonlinear differential equations with algebraic constraints II: Practical implications. *SIAM J. Sci. Stat. Comput.* 7, 720–733.
- WANG, R. 2002. High order adaptive collocation software for 1-D parabolic PDEs. Ph.D. thesis, Dalhousie University, http://www.mathstat.dal.ca/~keast/research/grad_super.html.
- WANG, R., KEAST, P., AND MUIR, P. 2004a. A comparison of adaptive software for 1-D parabolic PDEs. *J. Comput. Appl. Math.* 169, 127–150.
- WANG, R., KEAST, P., AND MUIR, P. 2004b. A high-order global spatially adaptive collocation method for 1-D parabolic PDEs. *Appl. Numer. Math.* 50, 239–260.

Received September 2003; revised March 2004, July 2004; accepted July 2004

ACM Transactions on Mathematical Software, Vol. 30, No. 4, December 2004.