Error Control B-spline Gaussian Collocation PDE Software with Time-Space Event Detection *

Jack Pew†, Connor Tannahill‡, Paul Muir§

Abstract

This report introduces BACOLIKR, a new *event detection* software package for the *error controlled* numerical solution of systems of one-dimensional time-dependent partial differential equations (PDEs). BACOLIKR employs B-spline Gaussian collocation for the spatial discretization in a spatial error control framework. A novel feature of this package is that it allows the user to specify solution dependent conditions that are used to determine a point in time when the specified time and space dependent event occurs. The event detection capability in BACOLIKR is based on its use of a modified version of the time integrator, DASKR, which implements time-dependent event detection as well as providing temporal error control.

BACOLIKR was developed through major modifications of the error control PDE solver, BACOLI, and the DASKR package. This report first provides an overview of the BACOLI and DASKR packages and then describes the software modifications required in order to develop BACOLIKR. The rest of the report investigates the application of BACOLIKR to solve a number of PDE-based event detection problems including solution layer-boundary intersection detection and solution layer merge detection in fluid mechanics models, critical tumor mass detection in a brain tumor model, steady state detection in the Cahn-Allen equation and the Gierer-Meinhardt model, and boundary event detection for a heat flow model.

Subject Classification: 65L06, 65L10, 65L80, 65M20, 65M70
Keywords: B-Splines, collocation, interpolation, error estimation, error control, event detection, partial differential equations.

1 Introduction

This report introduces a new software package, called BACOLIKR, for the error controlled numerical solution of time-dependent partial differential equations

*This work was supported by the Natural Sciences and Engineering Research Council of Canada and Saint Mary's University.

†Saint Mary's University, Halifax, NS, Canada, B3H 3C3
‡Saint Mary's University, Halifax, NS, Canada, B3H 3C3
§Saint Mary's University, Halifax, NS, Canada, B3H 3C3, muir@smu.ca

(PDEs) in one space dimension; this package is novel in that it features a *time and space dependent event detection capability*. This capability allows the user to specify a solution dependent condition that can be used to determine the time at which the specified time and space dependent event occurs. To our knowledge, BACOLIKR, is the only available error control PDE solver that offers time and space dependent event detection.

A simple example of an event detection question would be to ask when the solution to a PDE, at a given point in the spatial domain, takes on a specified value. However, as we will see in this report, more complex events that depend on, for example, the temporal or spatial derivative of the solution, or on the integral of the solution over the spatial domain, can be treated. Examples discussed in this report include solution layer-boundary intersection detection and solution layer merge detection in fluid mechanics models, critical tumor mass detection in a brain tumor model, steady state detection in the Cahn-Allen equation and the Gierer-Meinhardt model, and boundary event detection for a heat flow model.

The advantage of software with an event detection capability is that the software itself, to within the accuracy with which the numerical solution is computed, can determine when a specified condition arises. This is in contrast to a standard PDE solver where the user must specify explicitly when the solver should finish, and in this case there is no straightforward way for the user to determine the point in time when a solution dependent event of interest occurs. As well, when one uses software with built-in event detection, once an event has been detected, it is possible to modify the problem and then continue the computation from the point of the event, if so desired. We demonstrate how this can be done using BACOLIKR later in this report.

Accurate event detection requires that the numerical solution of the PDE be computed using error control. Since the event itself will depend on the value of the solution and/or its derivatives and/or its spatial integral, if the numerical solution is not computed accurately, then it will be impossible to determine the event time accurately. Error control means that for every time step taken by the solver, high quality estimates of the temporal and the spatial errors are computed, and the numerical solution associated with the time step is not accepted until these error estimates satisfy the user tolerance. Advantages of computing an error controlled numerical solution include the facts that the user can have reasonable confidence that the numerical solution has an error that is within the requested tolerance, that the cost of the computation will be proportional to the requested tolerance, and that, when the solver has an event detection capability, the point in time when the event happens will also be determined to within the requested tolerance.

The capability for time-space event detection depends heavily on the fact that the approximate solution computed by BACOLIKR is represented as a continuous function of time and space. We will discuss these features of the approximate solution later in this report.

The problem class we consider in this report is a PDE system of size *NPDE*

of the form,

$$\underline{u}_t(x,t) = \underline{f}\left(t, x, \underline{u}(x,t), \underline{u}_x(x,t), \underline{u}_{xx}(x,t)\right), \qquad a \le x \le b, \quad t \ge t_0, \tag{1}$$

with separated boundary conditions,

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

and initial conditions,

$$\underline{u}(x,t_0) = \underline{u}_0(x), \qquad a \le x \le b. \tag{3}$$

In addition, there is a vector function of size *NRT*, *called the gstop function*, which has the form

$$\underline{g}\left(t, x, \underline{u}(x,t), \underline{u}_x(x,t), \underline{u}_{xx}(x,t), \underline{u}_t(x,t)\right). \tag{4}$$

Each component of the gstop function defines an event and must be set to an expression whose value changes sign at the point in time where the event occurs. That is, *each event is defined to occur at the root of one of the components of the gstop function, and the task of locating the time at which an event occurs is implemented by finding the roots of the components of the gstop function.* For example, for an event where we want to determine the point in time where the solution, evaluated at $x = 0.5$, is equal to 1, then we have $NRT = 1$ and the single component of (4) should be set equal to,

$$U(0.5,t) - 1,$$

where $U(x,t)$ is the approximate solution.

The BACOLIKR package has been developed through a major modification of the BACOLI package [20], which employs B-spline Gaussian collocation for the spatial discretization of the PDEs; this discretization process yields a system of time-dependent ordinary differential equations (ODEs). These ODEs, together with the boundary conditions, represent a system of differential-algebraic equations (DAEs) that are solved in BACOLIKR using a major modification of the DAE solver, DASKR [5, 6], which is based on a family of Backward Differentiation Formulas (BDFs). For each time step, DASKR computes an estimate of the temporal error and then uses both adaptive time stepping and BDF order selection to control this error estimate so that it is less than the user tolerance. For each time step accepted by DASKR, BACOLIKR then computes an estimate of the spatial error of the numerical solution. If this spatial error estimate satisfies the user tolerance, then the solution computed on the current time step is accepted. Otherwise, BACOLIKR will adapt the spatial mesh and then restart the computation at the beginning of the current time step.

The source code for BACOLIKR and the examples considered in this report are available at

`http://cs.smu.ca/~muir/BACOLI-3_Webpage.htm`.

This report is organized as follows. In Section 2 we provide an overview of BACOLI and DASKR. In Section 3, we describe the software development process involved in developing BACOLIKR. As mentioned above, this will include describing the modifications that were made to the BACOLI and DASKR packages, in order to develop BACOLIKR. Section 4 investigates the application of BACOLIKR to a collection of problems that require event detection that depends on the solution (or its derivatives or spatial integral) to a PDE or a PDE system. We close in Section 5 with our summary, conclusions, and future work.

2 Overview of BACOLI and DASKR

2.1 BACOLI

For the problem, (1), (2), (3), as mentioned earlier, BACOLI employs B-spline Gaussian collocation for the spatial discretization. The numerical solution, $\underline{U}(x,t)$, is represented in terms of a B-spline basis of C^1-continuous piecewise polynomials in x, of degree p on each subinterval of a spatial mesh, $\{x_i\}_{i=0}^{NINT}$, which partitions $[a,b]$. $\underline{U}(x,t)$ has the form,

$$\underline{U}(x,t) = \sum_{i=1}^{NC_p} \underline{y}_{p,i}(t) B_{p,i}(x), \tag{5}$$

where $\underline{y}_{p,i}(t)$ is the (unknown) time dependent (vector) coefficient of the i-th B-spline basis function, $B_{p,i}(x)$, and $NC_p = NINT(p-1)+2$. The B-spline basis is implemented in BACOLI using the de Boor B-spline package [8]. The use of Gaussian collocation for the spatial discretization of the PDE means that $\underline{U}(x,t)$ is required to satisfy (1) at $p-1$ *collocation* points on each spatial mesh subinterval, where the collocation points are the images of the Gauss points, $\{\rho_j\}_{j=1}^{p-1}$, mapped on to each subinterval. Letting η_j be the jth collocation point, the corresponding collocation condition has the form,

$$\underline{U}_t(\eta_j,t) - \underline{f}\left(t,\eta_j,\underline{U}(\eta_j,t),\underline{U}_x(\eta_j,t),\underline{U}_{xx}(\eta_j,t)\right) = \underline{0}, \tag{6}$$

$j = 2,\ldots,NC_p-1$. $\underline{U}(x,t)$ is also required to satisfy the boundary conditions at $\eta_1 = a$ and $\eta_{NC_p} = b$; these have the form,

$$\underline{b}_L\left(t,\underline{U}(a,t),\underline{U}_x(a,t)\right) = \underline{0}, \qquad \underline{b}_R\left(t,\underline{U}(b,t),\underline{U}_x(b,t)\right) = \underline{0}. \tag{7}$$

The equations, (6) and (7), represent an index-1 system of differential algebraic equations (DAEs) and the solution of this DAE system gives the B-spline coefficients, $\underline{y}_{p,i}(t)$.

This collocation discretization yields a numerical solution that, for an arbitrary point in the spatial domain, has a spatial error that is $O(h^{p+1})$, where $h = \max_{i=1}^{NINT} h_i$ [7, 10]. (The numerical solution is said to be of order $p+1$.)

In BACOLI, this DAE system is solved using a modified version of the DASSL package [19], which computes *error controlled* approximations to the B-spline coefficients using a variable time step/variable order algorithm based on a family of BDFs of orders 1 to 5. One of the most significant modifications made to DASSL was the introduction of a new option for the treatment of the type of linear systems that arise during the computation of the B-spline coefficients. Due to the use of a B-spline basis, these linear systems have what is known as an almost block diagonal (ABD) structure [9], and therefore the modified version of DASSL employed in BACOLI makes use of the COLROW package [9], which is designed to efficiently treat such systems. The tolerance employed in the modified version of DASSL is slightly sharper than the user tolerance which is employed in the spatial error control algorithm implemented in BACOLI. This means that, generally, the time error associated with the computation of the B-spline coefficients will be slightly smaller than the spatial error of the approximate solution.

The spatial error control algorithm implemented in BACOLI requires that a high quality estimate of the spatial error of the numerical solution be computed. BACOLI has two options for computing a spatial error estimate for $\underline{U}(x,t)$.

The first option, described in [2], is based on the observation that, at certain points within the spatial domain, the spatial accuracy of $\underline{U}(x,t)$ is at least one order higher than it is at an arbitrary point in the spatial domain; these solution values are said to be superconvergent. The points at which $\underline{U}(x,t)$ is superconvergent include the mesh points as well as certain other points (see [2]) internal to each subinterval. It is also the case that the $\underline{U}_x(x,t)$ values at the mesh points are superconvergent. For each spatial mesh subinterval, a Hermite-Birkhoff polynomial interpolant is constructed that interpolates the superconvergent $\underline{U}(x,t)$ and $\underline{U}_x(x,t)$ mesh point values at each end of the subinterval as well as all the superconvergent $\underline{U}(x,t)$ values within the subinterval. Furthermore, in order to obtain an interpolant *whose interpolation error is dominated by the error of the interpolated* $\underline{U}(x,t)$ *values internal to the subinterval*, this Hermite-Birkhoff interpolant also interpolates the two closest superconvergent $\underline{U}(x,t)$ values internal to the left and right adjacent subintervals.

This Hermite-Birkhoff interpolant, known as the SuperConvergent Interpolant (SCI), has the following form on the ith subinterval, $[x_i, x_{i+1}]$. Let $s_1 = x_i$ and $s_2 = x_{i+1}$ and let $w_j, j = 1, \ldots, k$, be the non-mesh points where the superconvergent solution values are to be interpolated. (Here $k = p - 3$.) Then, at time t, the Hermite-Birkhoff interpolant has the form [12]

$$\sum_{j=1}^{2} H_j(x)\underline{U}(s_j,t) + h\sum_{j=1}^{2} \overline{H}_j(x)\underline{U}_x(s_j,t) + \sum_{j=1}^{k} G_j(x)\underline{U}(w_j,t), \tag{8}$$

where $x \in [x_i, x_{i+1}]$, $h = x_{i+1} - x_i$,

$$H_j(x) = (1 - (x - s_j)\gamma_j)\frac{\xi_j^2(x)\phi(x)}{\xi_j^2(s_j)\phi(s_j)}, \quad \overline{H}_j(x) = (x - s_j)\frac{\xi_j^2(x)\phi(x)}{\xi_j^2(s_j)\phi(s_j)},$$

$$G_j(x) = \frac{\phi_j(x)\xi^2(x)}{\phi_j(w_j)\xi^2(w_j)},$$

where

$$\phi(x) = \prod_{r=1}^{k}(x - w_r), \quad \phi_j(x) = \prod_{\substack{r=1 \\ r\neq j}}^{k}(x - w_r),$$

$$\xi(x) = \prod_{r=1}^{2}(x - s_r), \quad \xi_j(x) = \prod_{\substack{r=1 \\ r\neq j}}^{2}(x - s_r),$$

and

$$\gamma_j = \sum_{i=1}^{k}\frac{1}{s_j - w_i} + 2\sum_{\substack{i=1 \\ i\neq j}}^{2}\frac{1}{s_j - s_i}.$$

Since the interpolation error is dominated by the spatial error of the interpolated values, the spatial error of the Hermite-Birkhoff interpolant is the same as the error of the interpolated values, i.e., $O(h^{p+2})$. Over $[a, b]$, these Hermite-Birkhoff interpolants, taken together, represent a C^1-continuous piecewise polynomial solution approximation, which we call $\widetilde{\underline{U}}(x, t)$. A scaled difference (see below) of $\underline{U}(x, t)$ and $\widetilde{\underline{U}}(x, t)$ is computed in order to obtain a spatial error estimate for $\underline{U}(x, t)$. This error estimate is employed in BACOLI to provide what is known as *standard (ST) spatial error control*.

The second option available in BACOLI for computing a spatial error estimate for $\underline{U}(x, t)$ costs slightly less to implement than the SCI, does not make use of approximate solution values from outside a given subinterval when constructing the polynomial interpolant for that subinterval, and provides a conservative (overestimate) of the spatial error. On the ith subinterval, this interpolant, which we will refer to as $\overline{\underline{U}}(x, t)$, is also a Hermite-Birkhoff interpolant of the form (8), but in this case the w_j values are chosen so that *the interpolation error of this Hermite-Birkhoff interpolant is asymptotically equivalent to the spatial error for a collocation solution of one order lower than* $\underline{U}(x, t)$. (In this case, $k = p - 4$.) We therefore refer to $\overline{\underline{U}}(x, t)$ as the Lower Order Interpolant (LOI); the number of interpolation points is chosen so that the interpolation error dominates the error of the values being interpolated, and thus the LOI has a spatial error that is $O(h^p)$. See [3] for further details.

A scaled difference (see below) of $\overline{\underline{U}}(x, t)$ and $\underline{U}(x, t)$ can then provide a conservative estimate of the spatial error for $\underline{U}(x, t)$. Since $\underline{U}(x, t)$ is returned to the user but the spatial error control is based on a spatial error estimate that is for an approximation to the solution that is of one lower spatial order than $\underline{U}(x, t)$, we have an example of what is known as Local Extrapolation (LE) spatial error control - see, e.g., [15].

When BACOLI is called with a given input value for p, it computes and returns a numerical solution based on B-splines of degree p. If the ST spatial error control mode is chosen, then the code constructs the SCI to generate a spatial error estimate which is then used as the basis for ST spatial error control.

If the LE spatial error control mode is chosen, then BACOLI constructs the LOI and uses it to generate a spatial error estimate which is then used to provide LE spatial error control. Thus the availability of the two types of interpolants corresponds to providing options for two modes of spatial error control, ST mode or LE mode, similar to what is available when a Runge-Kutta formula pair is used to provide error control for an initial value ODE - see, e.g., [15]. See [21] for a detailed performance analysis of BACOLI using these two error control modes.

For either error control mode, two types of spatial error estimates for $\underline{U}(x,t)$ are computed by BACOLI. The first is the set of error estimates, $E_j(t), j = 1,\ldots,NPDE$; each of these values represents a scaled spatial error estimate, over the entire spatial domain, for one component of the solution. These have the form,

$$E_j(t) = \sqrt{\int_a^b \left(\frac{U_j(x,t) - \widehat{U}_j(x,t)}{ATOL_j + RTOL_j|U_j(x,t)|} \right)^2 dx}, \tag{9}$$

where t is the current time, $ATOL_j$ and $RTOL_j$ are the absolute and relative tolerances for the j-th approximate solution component, $U_j(x,t)$, and $\underline{\widehat{U}}_j(x,t)$ is the jth component of either $\underline{\widetilde{U}}(x,t)$ or $\underline{\overline{U}}(x,t)$. The second set of spatial error estimates are, $\widehat{E}_i(t), i = 1,\ldots,NINT$; each of these values provides a scaled spatial error estimate over all components of $\underline{U}(x,t)$ for the ith subinterval. These are of the form,

$$\widehat{E}_i(t) = \sqrt{\sum_{j=1}^{NPDE} \int_{x_{i-1}}^{x_i} \left(\frac{U_j(x,t) - \widehat{U}_j(x,t)}{ATOL_j + RTOL_j|U_j(x,t)|} \right)^2 dx}. \tag{10}$$

Again, $\underline{\widehat{U}}_j(x,t)$ is the jth component of either $\underline{\widetilde{U}}(x,t)$ or $\underline{\overline{U}}(x,t)$.

These spatial error estimates are computed after each accepted time step taken by DASKR. A step is accepted when $E_j(t) < 1$ for $j = 1,\ldots,NPDE$. Otherwise the step is rejected and the $\widehat{E}_i(t)$ values are used as the basis for a mesh refinement algorithm that attempts to construct a new mesh such that (i) the numerical solution computed on that mesh will have a spatial error estimate that satisfies the tolerance and (ii) the spatial error estimates over the subintervals of that mesh will be approximately equidistributed. Both the location and number of mesh points can be changed during a remeshing in order to adapt to the size (with respect to the user tolerance) and distribution of the spatial error estimates over the spatial domain. See [24] for further details.

2.2 DASKR

The DASKR solver was obtained through an extension of the solver DASPK [5], which itself was developed from the original member of this software family, DASSL. As mentioned earlier in this report, DASKR is based on a family of BDFs and uses both adaptive time stepping and BDF order selection to control

an estimate of the temporal error. DASKR represents the approximate solution it computes in terms of a continuous piecewise polynomial interpolant. The temporal error control is applied only to the solution approximation at the end of each time step; the order of the interpolant is chosen to be consistent with the order of the BDF used to obtain the solution approximation at the end of the step. In addition to a choice of direct methods (dense or banded) for the treatment of the linear systems that arise during the computation of a numerical solution, DASKR also provides the user with the option of using a Krylov method, the Generalized Minimum Residual (GMRES) method, in either complete or incomplete form, with scaling and preconditioning [5], for use when the linear systems are large. As well, DASKR improves upon DASSL by providing an option for the calculation of consistent initial conditions for the DAE system to be solved, when the user is not able to provide these.

Furthermore, as mentioned earlier, DASKR has a *time-dependent event detection capability* that can be employed while the solver is computing a numerical solution to a DAE system. At the end of each accepted time step, DASKR calls a routine which evaluates the user's gstop function in order to monitor for sign changes in any of the components of the gstop function. When a sign change is detected, DASKR uses the interpolant to the solution together with a search algorithm [16] to locate the root of the corresponding component of the gstop function, and then returns to the calling program.

(While, as mentioned above, DASKR has the capability for treating large DAE systems using the preconditioned GMRES algorithm, we have not yet incorporated this feature of DASKR into the BACOLIKR package; this would require a major modification to BACOLIKR and we therefore identify this as a potential project for future work in the final section of this report. Also, the algorithm provided within DASKR for computing consistent initial conditions is not employed because BACOLIKR computes its own consistent initial conditions for the DAE system before calling DASKR.)

3 Development of BACOLIKR

As mentioned earlier, event detection is implemented through a user defined gstop function, each component of which is used to characterize an event; this is done by writing each component of the gstop function so that it has a root at the point in time where the event occurs. It is therefore common for event detection software to be described in terms of a root finding capability where the goal of determining the time at which an event occurs is described in terms of finding roots of the gstop function. Thus, in this section, we make reference to root finding rather than event detection when describing the software modifications.

3.1 Major modifications

This subsection describes the major modifications that were made to BACOLI and DASKR in order to develop BACOLIKR. The overall structure and user

interface for BACOLIKR is similar to that of BACOLI and we therefore refer the reader to [20] for additional details.

- As mentioned earlier in this report, BACOLI makes use of a modified version of DASSL. See Section 3 of [22] for a detailed description of the changes that were made to DASSL.

  In order to use DASKR within BACOLIKR it was therefore necessary to make similar changes to DASKR. One major change involved modifying DASKR to provide an option for it to use the ABD linear system solver, LAMPAK [17], to solve the ABD linear systems that arise. (BACOLIKR uses LAMPAK, rather than the COLROW package used by BACOLI, in order to remove any proprietary dependencies. The use of LAMPAK rather than COLROW does not cause any significant impact in performance.)

- A subroutine called BACRT was added. This subroutine calls the user's gstop subroutine, which we will refer to as RT; for a given time, t, and current solution approximation, $\underline{U}(x,t)$ (or $\underline{U}_t(x,t)$, $\underline{U}_x(x,t)$, or $\underline{U}_{xx}(x,t)$ approximation if necessary), RT evaluates the components of the user's gstop function (4).

  The BACRT routine provides an interface between the rest of BACOLIKR and the user's RT routine in order to simplify the argument list for the RT routine and hide a number of implementation details.

- A number of small changes were made in order to manage the root finding capability. The major changes of this type implemented communication between DASKR, BACOLIKR, and the BACRT routines.

- A capability was added to allow the user to force BACOLIKR to restart DASKR for the next time step using a cold start. (BACOLIKR itself has the capability to force a cold start but for some event detection problems it is important that the user be able to do this as well. A typical example is when the PDE or boundary conditions must be changed after an event has been detected. In such cases, it is more efficient to perform a cold start (in which case DASKR begins with a low order BDF and a small step size) than it is to have DASKR attempt to continue with the current order BDF and a large step size.)

3.2 User interface modifications

This subsection describes changes to the user interface, as well as the motivations for each change. The BACOLIKR package is based on a Fortran95 wrapper that wraps a Fortran 77 solver within which the primary computations take place.

3.2.1 Fortran 95

- Added to the BACOLIKR initialization routine, BACOLI95_INIT, the optional integer argument, NRT, which specifies the number of roots of the

gstop function that will be searched for over the course of the computation. This is equal to the number of components of the gstop function. If NRT is greater than 0 then the main solver subroutine, BACOLI95, expects to be passed the name for the user's root finding subroutine; see below. (As mentioned above, this routine is known internally as RT, but the actual name can of course be different.)

- Added to the main solver routine, BACOLI95, an optional argument that allows the user to specify the actual name of the root finding subroutine. It has the signature:

  ```
  subroutine RT(T, X, NINT, UB, UTB, NEQ, RVAL, NRT),
  ```

  where T, the current point in time, X, the current spatial mesh, NINT, the current number of mesh subintervals, UB, the array of B-spline coefficients at the current time, and UTB, the array of time derivatives of the B-spline coefficients at the current time, are input arguments, and RVAL, the array for which RVAL(i), i = 1, ..., NRT, gives the value of the ith component of the gstop function, is the lone output argument.

- Added a new field to the structured solution type, BACOLI95_SOL; the new field is an integer array, JROOT, where the value of JROOT(i), i = 1, . . . , NRT, can indicate when there has been a change in sign in the ith component of the gstop function. On any return:

  - BACOLI95_SOL%JROOT(i) = 0 indicates that the ith component of the gstop function has not changed sign during the current time step,
  - BACOLI95_SOL%JROOT(i) = 1 indicates that the ith gstop root has been found and the sign of the ith component of the gstop function during the current time step has gone from negative to positive,
  - BACOLI95_SOL%JROOT(i) = -1 indicates that the ith gstop root has been found and the sign of the ith component of the gstop function during the current time step has gone from positive to negative.

  When a root is found, BACOLIKR sets BACOLI95_SOL%IDID = 5. Then the BACOLI95_SOL%JROOT array can be examined to detect which root have been found by checking for a non-zero component.

- When the user sets BACOLI95_SOL%MFLAG(1) = 2, BACOLIKR is forced to call DASKR using a cold start for the next time step. This is useful, as mentioned earlier, for problems where the discovery of a root may require a change in the problem. Attempting to restart, in such a case, with a warm start could lead to a failure by DASKR, or alternatively, multiple failed time steps as DASKR reduces the order of the BDF method and the size of the time step, in order to step past the discontinuity in an error controlled manner, leading to a very inefficient computation. The

cold start forces it to begin with the first order BDF and a small time step, making the computation more likely to succeed despite the change to the problem.

3.2.2 Fortran 77

The interface to the Fortran 77 solver that is contained within BACOLIKR differs from the interface to the Fortran 77 solver that is contained within BACOLI in two major ways:

- In BACOLIKR, the integer work array IPAR and the floating point work array RPAR have different sizes than they do in BACOLI; the change in size depends on the value of NRT.
- In BACOLIKR, the arguments, NRT, JROOT and RT must always be passed to the Fortran 77 solver since optional arguments are not available in Fortran 77. In the case where no root finding is to be done, NRT is set to 0 and dummy arguments for JROOT and RT are given.

4 Application of BACOLIKR to event detection problems

In this section, we demonstrate the capabilities of the BACOLIKR package by showing how to apply it to a number of event detection problems that are based on the solution of a PDE or a system of PDEs. Each subsection will consider one problem and will demonstrate how to call BACOLIKR and write an appropriate root finding routine and corresponding main program in order to implement the specific type of event detection required for each problem.

In order to apply BACOLIKR to an event detection problem, the user must customize a module called ROOTFINDING. The primary component of this module is the subroutine (mentioned earlier) called RT (internally), within which the details associated with the characterization of a single event or multiple events must be given. As mentioned earlier, the specification of a event may require, within RT, an auxiliary computation involving, for example, the numerical solution, its derivatives in time or space, or its integral in space. The evaluation of the numerical solution and/or its time or space derivative is performed through a call to the VALUES routine. Within the RT routine, assignments are made to the gstop vector, RVAL, in order to define the conditions that characterize each event as a root of one component of the gstop function. Within each subsection associated with an event detection problem, we will explain how the RT routine can be written to define the specific event or events.

In addition to the RT routine, the ROOTFINDING module includes the SETSOL routine. The latter dynamically allocates a work array that is used within the RT routine, and, for those applications in which a spatial integral

must be computed, it also computes the Gauss points that will be used to obtain a numerical approximation to the integral.

The other primary software component associated with the use of BACOLIKR for event detection is the main program that will make calls to the solver routine, BACOLI95, as appropriate for whatever event or events are to be detected. In the case of multiple events, BACOLI95 must be called multiple times. For each problem we consider, we will describe in general terms how the main program is organized.

The other software components of BACOLIKR that are used in the main programs are the BACOLI95_INIT routine, which initializes the computation, the BACOLI95_VALS routine, which is used to evaluate the numerical solution and its first and second spatial derivatives, the BACOLI95_SOL data structure which contains a number of fields where information associated with the approximate solution is stored, and the BACOLI95_SOL_TEARDOWN routine which must be called at the end of the computation to release the dynamic memory that is allocated during the computation.

In addition to customizing the ROOTFINDING module and the main program, the user must also provide problem definition routines that define the PDE(s), the boundary conditions, and the initial condition(s). We refer the reader to

`http://cs.smu.ca/~muir/BACOLI-3_Webpage.htm.`

to see complete source code for the main program, the ROOTFINDING module, and the problem definition routines for each problem we consider.

4.1 Solution value detection for the One Layer Burgers' Equation

Burgers' equation is a standard model in fluid mechanics. Here we consider an instance of this problem which we call the One Layer Burgers' Equation (**OLBE**); it has the form,

$$u_t = \epsilon u_{xx} - u u_x, \tag{11}$$

with boundary conditions at $x = 0$ and $x = 1$ ($t > 0$) and an initial condition at $t_0 = 0$ ($0 \le x \le 1$) taken from the exact solution,

$$u(x,t) = \frac{1}{2} - \frac{1}{2}\tanh\left(\frac{x - \frac{t}{2} - \frac{1}{4}}{4\epsilon}\right), \tag{12}$$

where ϵ is a problem-dependent parameter. We will choose $\epsilon = 10^{-3}$ for this example. We solve this problem from $t_0 = 0$ to $t_{out} = 1$ and choose a tolerance of 10^{-6}. For $t_0 = 0$, the solution has a sharp layer located at $x \approx 0.25$. As t goes from 0 to 1, the layer moves to the right and is located at $x \approx 0.75$ for $t_{out} = 1$. See Figure 4 of [23] for a plot of the solution to this problem.

In order to demonstrate the event detection capability of BACOLIKR in a simple form, we will define a very basic event for this problem. The task will

be to determine the time at which the approximate solution, $U(x,t)$, satisfies the condition that $U(0.4,t) = 0.5$. Thus there will be one root and the gstop function will be $[U(0.4,t) - 0.5]$.

Within the RT routine, we make one call to the VALUES routine which computes the value of the approximate solution at the current time, for a given choice of x. A key point here is that we pass the vector of B-spline coefficients into the VALUES routine so that the solution itself is computed. (There is the option of passing to the VALUES routine the vector of *time derivatives* of the B-spline coefficients in order to evaluate the time derivative of the numerical solution; this will be considered in a later example. When the vector of time derivatives of the B-spline coefficients is passed to the VALUES routine, it is also possible to obtain, at desired locations within the spatial domain, approximate values for $u_{tx}(x,t)$ and $u_{txx}(x,t)$, should these be needed to characterize a given event.)

We call VALUES with $x = 0.4$. The difference between the returned solution approximation and the target value, 0.5, is then assigned to the RVAL output argument of RT, in order to define the event for this example.

The corresponding main program first initializes the computation with a call to the BACOLI95_INIT routine. This is followed by a call to the SETSOL routine and then a call to the BACOLI95 solver routine. The BACOLI95 routine monitors the output from the RT routine looking for a time step on which there is a change in the sign of the gstop function. Once this happens and the specific point in time where the root occurs has been found, BACOLI95 returns to the main program with an indication that the root has been found. For this example it turns out that the root of the gstop function, $[U(0.4,t) - 0.5]$, occurs when $t \approx 0.3$. The main program writes out the solution at the time of the event and then calls the solver again in order to complete the computation through to $t_{out} = 1$. Since the problem has not changed, a warm start can be employed. The solver returns with an indication that t_{out} has been reached, and the solution at t_{out} is printed out. See Figure 1 for a plot of the solution at the time of the event.

4.2 Solution value detection involving multiple events for the Catalytic Surface Reaction Model

The Catalytic Surface Reaction Model (**CSRM**) [25] has a PDE system of the form,

$$\begin{aligned}
(u_1)_t &= -(u_1)_x + n(D_1 u_3 - A_1 u_1 \gamma) + (u_1)_{xx}/Pe_1, \\
(u_2)_t &= -(u_2)_x + n(D_2 u_4 - A_2 u_2 \gamma) + (u_2)_{xx}/Pe_1, \\
(u_3)_t &= A_1 u_1 \gamma - D_1 u_3 - R u_3 u_4 \gamma^2 + (u_3)_{xx}/Pe_2, \\
(u_4)_t &= A_2 u_2 \gamma - D_2 u_4 - R u_3 u_4 \gamma^2 + (u_4)_{xx}/Pe_2,
\end{aligned} \tag{13}$$

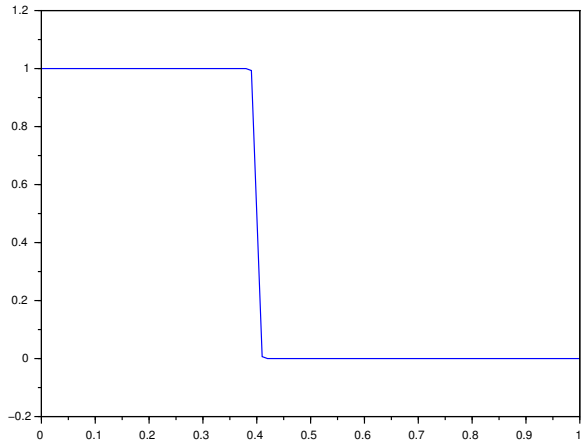

Figure 1: Numerical solution to **OLBE** when $U(0.4, t) \approx 0.5$; this occurs when $t \approx 0.3$.

where $\gamma = 1 - u_3 - u_4$, and $n, r, Pe_1, Pe_2, D_1, D_2, R, A_1$, and A_2 are problem dependent parameters. The initial conditions at $t = 0$ $(0 \leq x \leq 1)$ are,

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

and the boundary conditions at $x = 0$ and $x = 1$ $(t > 0)$ are,

$$(u_1)_x(0, t) = -Pe_1(2 - r - u_1(0, t)), \quad (u_2)_x(0, t) = -Pe_1(r - u_2(0, t)),$$

$$(u_3)_x(0, t) = (u_4)_x(0, t) = 0,$$

$$(u_1)_x(1, t) = (u_2)_x(1, t) = (u_3)_x(1, t) = (u_4)_x(1, t) = 0.$$

See Figures 12-15 of [23] for plots of the solution components for $Pe_1 = Pe_2 = 100, D_1 = 1.5, D_2 = 1.2, R = 1000, r = 0.96, n = 1$, and $A_1 = A_2 = 30$. Here we choose the problem dependent parameters as above except, in order to make the problem more challenging, we choose $Pe_1 = Pe_2 = 10000$.

We choose a tolerance of 10^{-5}. This example is included to show how to treat a problem with multiple events and multiple solution components.

We define two simple events. We wish to find the time at which $U_3(0.2, t) = 0.24$ and the time at which $U_4(0.2, t) = 0.24$. Thus there are two roots and the gstop vector function is $[U_3(0.2, t) = 0.24, U_4(0.2, t) = 0.24]^T$.

Inside RT, we make one call to the VALUES with $x = 0.2$. The VALUES routine returns a vector of the four solution component values for this value of x and at the current time. The difference between the third component of the returned solution approximation and the target value, 0.24, is then assigned to

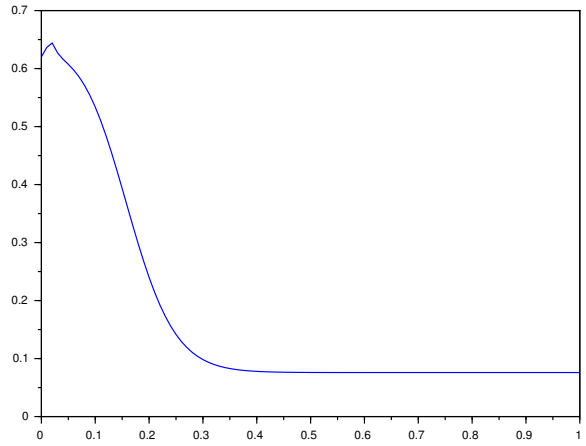

Figure 2: Third component of the numerical solution to **CSRM** when $U_3(0.2, t) \approx 0.24$; this occurs when $t \approx 1.2008$.

the first component of RVAL while the difference between the fourth component of the returned solution approximation and the target value, 0.24, is assigned to the second component of RVAL. DASKR checks, on every step, to see if either component of RVAL changes sign, thereby monitoring both of the events over the duration of the computation.

After initializing the computation with calls to the BACOLI95_INIT and SETSOL, the main program calls BACOLI95. The solver returns with an indication that one of the roots has been found. For this example it turns out that the root of the gstop function component, $U_3(0.2, t) - 0.24$, is found for $t \approx 1.2008$. The main program writes out the solution and then calls BACOLI95 again in order to continue the computation. See Figure 2 for a plot of the solution at the time of this first event. Since the problem has not changed a warm start can be employed. BACOLI95 returns a second time with an indication that a root has been found. In this case the root of the gstop function component, $U_4(0.2, t) - 0.24$, is found for $t \approx 1.8816$. The solution at this time is then printed out and the computation is terminated. See Figure 3 for a plot of the solution at the time of the second event.

4.3 Layer merge detection for the Two Layer Burgers' Equation

The Two Layer Burgers' Equation (**TLBE**) is based on the PDE, (11), but with boundary conditions at $x = 0$ and $x = 1$ ($t > 0$) and an initial condition

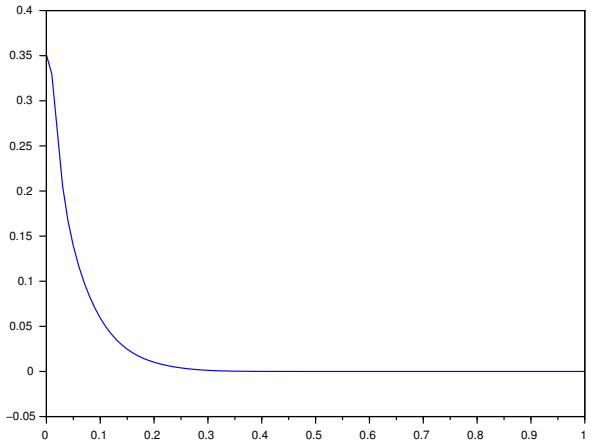

Figure 3: Fourth component of the numerical solution to **CSRM** when $U_4(0.2, t) \approx 0.24$; this occurs when $t \approx 1.8816$.

at $t_0 = 0$ $(0 \leq x \leq 1)$ taken from the exact solution,

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

where,

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

where ϵ is a problem-dependent parameter. For this example, we choose $\epsilon = 10^{-3}$. When $t_0 = 0$, the solution has two sharp layers, one at $x = 0.25$ and one at $x = 0.5$. As t increases, these layers move to the right and eventually merge, forming a single layer. See Figure 1 of [23] for a plot of the solution to this problem.

Our goal in this example is to determine the time at which the two layers merge. The left layer corresponds to a sharp transition in the solution value from 1 to 0.5. We will therefore define the location of this layer to be at the point, x_L, where the solution has the value 0.75, half way through the transition in the solution values to the left and right of the layer. Similarly, the right layer corresponds to a transition in the solution value from 0.5 to 0.1. We therefore define the location of this layer to be at the point, x_R, where the solution has the value 0.3, halfway through the transition in those solution values to the left and right of that layer. We will define the layers to have merged when x_L and x_R are within a distance of 5ϵ. (See below for further discussion on this point.) Thus there will be one root and the gstop function will be $[|x_L - x_R| = 5\epsilon]$.

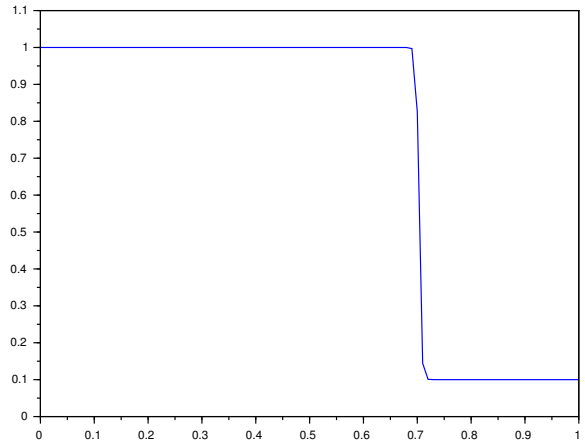

Figure 4: Numerical solution, $U(x,t)$, to **TLBE** when the two layers merge; this occurs when $t \approx 0.622056$.

We will solve this problem using a tolerance of 10^{-6}.

Within the RT routine, we begin by evaluating the approximate solution at the endpoints of the spatial domain. These values are then used as input to a bisection algorithm that is used to determine the point, x_L, in the spatial domain where $U(x_L,t) \approx 0.75$. Similarly, we use a second bisection algorithm to determine the point, x_R, in the spatial domain where $U(x_R,t) \approx 0.3$. When the left layer and right layer have merged, the points x_L and x_R will be on the same single layer that remains, with x_L always slightly to the left of x_R. As mentioned above, for the choice of ϵ we have made, when the layers merge, x_L and x_R will be within a distance of about 5ϵ. After this time they are located within the same layer and the distance between the two points stays on the order of ϵ. We have found that this distance is approximately 5ϵ. (We have determined experimentally that the two points, x_L and x_R, remain a distance of about 0.0049 after the layers merge.) Thus the difference between x_L and x_R and 5ϵ is then assigned to the RVAL output argument of RT, in order to define the event for this example.

The corresponding main program first initializes the computation with a call to the BACOLI95_INIT routine and then makes a call to the SETSOL routine. This is followed by a call to the BACOLI95 solver. The solver then returns with an indication that the root has been found. For this example it turns out that the two layers are determined to have merged for $t \approx 0.622056$. The main program writes out the solution at the time of the event and then ends. See Figure 4 for a plot of the solution at the time of the event.

4.4 Critical tumor mass detection for a Brain Tumor Model

This problem (**BTM**) models the growth of a brain tumor within a region of the brain that includes three consecutive regions involving grey-white-grey matter. We consider a modification of the model discussed in [4] in which the discontinuous diffusion coefficient that arises due to the different brain matter regions (grey or white) is replaced with a continuous diffusion coefficient that has sharp layer regions corresponding to the transitions between the brain matter regions. (The reason we do this is that the general form for the approximate solution employed in BACOLIKR assumes C^1-continuity in x. However, the discontinuous diffusion coefficient in the original form of **BTM** forces the first spatial derivative of the exact solution to be discontinuous [4].)

The PDE for this problem is

$$u_t(x,t) = (D(x)u_x(x,t))_x = D_x(x)u_x(x,t) + D(x)u_{xx}(x,t), \tag{14}$$

where,

$$D(x) = \left(\left(\frac{1}{e^{-l(x-w_1)}+1}\right) + \left(\frac{1}{e^{l(x-w_2)}+1}\right) - 1\right)(1-\gamma) + \gamma, \tag{15}$$

approximates a step function whose value to the left and right of the region $[w_1, w_2]$ (a subregion of the spatial domain, $[a, b]$) is γ, and whose value within the region $[w_1, w_2]$ is 1. The parameter l controls the sharpness of the transition layers between $[w_1, w_2]$ and the rest of the spatial domain. We choose $w_1 = -0.5$, $w_2 = 0.5$, $\gamma = 0.2$, and $l = 30$. In Figure 5, we give a plot of $D(x)$ for the above choice of parameters on $[-5, 5]$. The boundary conditions are

$$u_x(a,t) = 0, \quad u_x(b,t) = 0,$$

where $a = -5$, $b = 5$, and the initial condition is

$$u(x,0) = \frac{e^{\frac{-(x-\xi)^2}{\eta^2}}}{\eta\sqrt{\pi}},$$

where $\xi = -2$ and $\eta = 0.2$. This gives an initial solution that has a spike of height approximately 2.8 centered at $x = -2$. We choose a tolerance of 10^{-6}.

The tumor concentration, $c(x,t)$, is obtained from the solution, $u(x,t)$, of the above PDE with the indicated boundary and initial conditions through the equation,

$$c(x,t) = e^t u(x,t).$$

The tumor grows both in size and width across the spatial domain as time progresses, with different growth rates in the grey and white matter regions, as determined by the different values of the diffusion coefficient across the spatial domain.

For this problem we wish to find the time at which the mass of the tumor reaches the critical value of 10. The mass of the tumor at a given point in

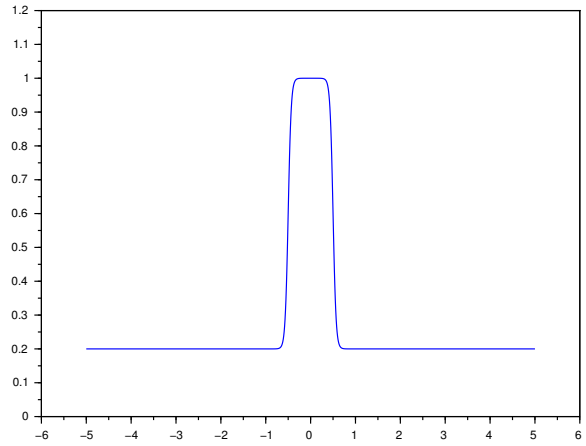

Figure 5: Diffusion coefficient, $D(x)$ (15), with $w_1 = -0.5$, $w_2 = 0.5$, $\gamma = 0.2$, and $l = 30$.

time is obtained by integrating the tumor concentration, $c(x,t)$, over the spatial domain $[a,b]$. Thus, each time the RT routine is called, we need to compute an approximation to the integral of the approximate solution, $U(x,t)$, over $[a,b]$. To do this, we will use a Gaussian quadrature rule on each subinterval of sufficiently high degree that the integral of $U(x,t)$ over the subinterval will be computed to high accuracy. (It may be possible to obtain a more efficient computation by using an adaptive quadrature routine to approximate the integral of $U(x,t)$ over $[a,b]$ to an accuracy that is consistent with the tolerance employed in the computation of $U(x,t)$ but we do not consider this here since the specific way in which the integral is computed is not central to the current discussion.)

The canonical Gauss points and weights are pre-computed within the SETSOL routine which is called once by the main program, as mentioned earlier.

Inside the RT subroutine, we first compute the specific Gauss points and weights for use on each subinterval of the current spatial mesh. The calculations access the canonical Gauss points and weights computed by the SETSOL routine. We then call the VALUES routine with the Gauss points as input to (simultaneously) evaluate the approximate solution at all Gauss points on all subintervals. We then multiply these values by the appropriate Gauss weights and sum over all subintervals to get an approximation to the integral of $U(x,t)$, over $[a,b]$. We then multiply this integral approximation by e^t to obtain the mass of the tumor over $[a,b]$ at the current time t. The gstop function in this case is the difference between the mass of the tumor at the current time and the target value of 2.

The corresponding main program, after calling BACOLI95_INIT and SET-

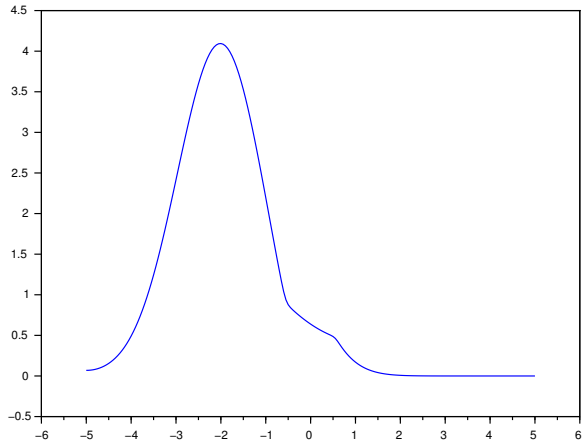

Figure 6: Approximate concentration, $C(x,t)$, for **BTM** when the mass of the tumor reaches the critical value of 10; this occurs when $t \approx 2.30258$.

SOL, calls BACOLI95 to determine the point in time where the tumor mass, as approximated above, reaches the critical value. We find that this happens when $t \approx 2.30258$. The main program prints out the solution and then terminates. See Figure 6 for a plot of the approximate concentration, $C(x,t) = e^t U(x,t)$, where $U(x,t)$ is the approximate solution to (14), with the boundary and initial conditions indicated above, at the time of the event.

4.5 Steady state detection via layer-boundary tracking for the One Layer Burgers Equation

This problem again considers the One Layer Burgers Equation, and for this example, the event detection task is to discover when the solution reaches steady state.

For a typical problem, the detection of steady state requires a more elaborate computation (see the next two examples) but for the One Layer Burgers Equation, the behavior of the solution makes obtaining the answer to this question straightforward.

Recall that the One Layer Burgers Equation has a solution that, initially, has a sharp layer region at $x \approx 0.25$, and, as t goes from 0 to 1, the layer moves to the right and is located at $x \approx 0.75$ for $t = 1$. At any point in time, to the left of the layer, the solution value is 1 while to the right of the layer, the solution is value is 0.

At some point in time after $t = 1$, the layer moves past the right boundary, and from that point in time onward, the solution is equal to 1 across the entire

spatial domain $[0, 1]$, i.e., the solution has reached a steady state. Thus the determination of the time at which the solution reaches steady state is, for this problem, equal to the time at which the moving layer passes beyond the right boundary. This will be the time at which the solution at the right boundary equals 1, since prior to this time, the solution at the right boundary will always be less than 1.

Thus, for this problem, the determination of the time when the solution reaches steady state reduces to determining when the approximate solution, $U(x, t)$, satisfies the condition $U(1, t) = 1$.

This means that this problem reduces to a minor variation of the first example we have considered. Instead of searching for the time when $U(x, t)$ satisfies the condition that $U(0.4, t) = 0.5$, we search for the time when $U(1, t) = 1$. We choose a tolerance of 10^{-6}.

The RT subroutine and the main program are quite similar to those for the first example.

For this example, BACOLIKR determines that steady state has been reached, i.e., $U(1, t) = 1$, for $t \approx 1.69363$. (We do not provide a figure showing the solution at this point in time since it is identically equal to 1 across [0,1].)

4.6 Steady state detection for the Cahn-Allen Equation

The Cahn-Allen Equation (**CaAl**) [1] models phase separation in multi-component alloy systems. It has the form

$$u_t(x, t) = \epsilon u_{xx}(x, t) - u(x, t)^3 + u(x, t),$$

where ϵ is a problem dependent parameter which we choose to be 10^{-6} for this investigation. The boundary conditions are,

$$u_x(0, t) = 0, \quad u_x(1, t) = 0,$$

and initial solution is,

$$u(x, 0)) = 0.01 \cos(10\pi x).$$

This initial solution is a low amplitude oscillating function with a period of 0.2. As time proceeds, this initial solution grows in amplitude and develops (at steady state) into a step function that has a series of regions where the solution value is constant, alternating in value between 1 and -1, with sharp transition layers from one region to the next. (See [23], Figure 7, to see the evolution of the solution to this problem over the time period, $t \in [0, 8]$.)

Our goal is to determine the time at which the numerical solution to the Cahn-Allen equation reaches steady state (to within the tolerance with which the numerical solution is computed). We choose a tolerance of 10^{-8}.

We will define steady state to have been reached when the absolute value of the time derivative of the solution over the spatial domain has (effectively) reached a value of 0. This will be when the time derivative of the solution is as small as the tolerance with which the numerical solution has been computed.

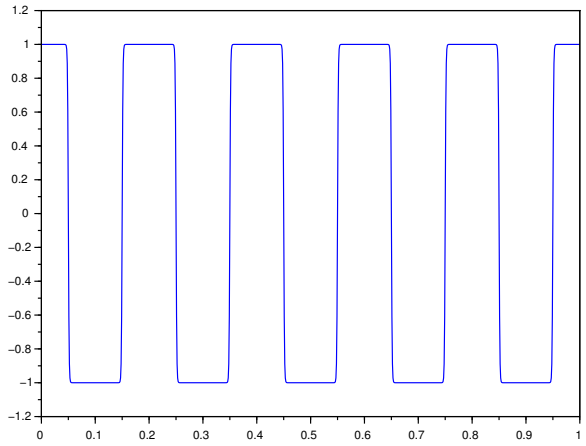

Figure 7: Numerical solution, $U(x,t)$, to **CaAl** when the solution reaches steady state; this occurs when $t \approx 17.684440$.

In order to assess the size of the time derivative of the solution, at a given point in time, over the entire spatial domain, we will compute an approximation to the integral of $|U_t(x,t)|$ over $[0,1]$. The gstop function will therefore be $[\int_0^1 |U_t(x,t)|dx - tol]$, where *tol* is the user tolerance.

Within the RT routine, we first compute the Gauss points and weights on each subinterval of the spatial mesh (based on the canonical Gauss points and weights computed during the call to SETSOL by the main program). We then call the VALUES routine with the Gauss points as input to evaluate the time derivative of the approximate solution at the Gauss points on all subintervals. *In order to obtain values of the time derivative of the approximate solution, we pass the vector of time derivatives of the B-spline coefficients into the VALUES routine.* We then multiply the absolute values of these time derivatives by the appropriate Gauss weights and sum over all subintervals to get an approximation to the integral of $|U_t(x,t)|$ over $[0,1]$, for the current time. The gstop function in this case is the difference between this integral approximation and the tolerance with which the numerical solution is computed.

The corresponding main program, after calling BACOLI95_INIT and SETSOL, calls BACOLI95 to determine the time at which the approximate solution reaches steady state (as defined above). We find that the solution reaches steady state when $t \approx 17.684440$. The main program prints out the solution and then terminates. The solution at this time is shown in Figure 7.

4.7 Steady state detection for the Gierer-Meinhardt Model

The Gierer-Meinhardt Model (**GMM**) [14] is an activator-inhibitor system associated with the modeling pattern formation in biological systems. A key phenomenon in such systems is that for certain parameter choices it is possible to observe spontaneous pattern formation from an initially (almost) homogeneous initial state. The form of the Gierer-Meinhardt Model we will consider has a PDE system of the form,

$$a_t(x,t) = \epsilon^2 a_{xx}(x,t) - a(x,t) + a(x,t)^2/h(x,t),$$

$$\tau h_t(x,t) = D h_{xx}(x,t) - \mu h(x,t) + a(x,t)^2,$$

where ϵ, τ, D, and μ are problem dependent parameters. For this investigation, we choose $\epsilon = 0.1$, $\tau = 1$, $D = 0.1$, and $\mu = 1$. The boundary conditions are

$$a_x(a,t) = h_x(a,t) = 0, \quad a_x(b,t) = h_x(b,t) = 0,$$

where the spatial domain is $[a,b]$. For this example, we choose $a = -2, b = 2$. The initial solutions are,

$$a(x,0) = h(x,0) = 0.1564 + 0.01\sin(10\pi x), \quad h(x,0) = 0.1564 + 0.01\sin(10\pi x),$$

which corresponds to setting each solution component to a constant plus a small amount of high frequency noise.

The two solution components start out as almost constant functions but, over time, the first component develops stable spikes, reaching a maximum of approximately 1.8, with troughs in between of magnitude approximately 0.1. The second solution component develops a smoother solution profile, oscillating between values of approximately 1.4 and approximately 0.7 over the spatial domain.

Out goal in this example is to determine the time at which the spike solution, i.e., the first solution component of this model, reaches a steady state. That is, we will look for a time at which the time derivative of the first solution component is as small as the tolerance with which the numerical solution has been computed. We choose a tolerance of 10^{-6}.

As was the case for the previous example, we will define the time derivative of the first solution component to be sufficiently small when the integral of the absolute value of the time derivative of the first solution component over $[a,b]$ is less than the user tolerance. Thus the gstop function will be $[\int_a^b |a_t(x,t)|dx - tol]$.

The organization of the RT routine in this case is similar to that of the previous example. We compute the Gauss points and weights on each subinterval of the spatial mesh and then call the VALUES routine to obtain values of the time derivative of the approximate solution at the Gauss points. We then compute the weighted sum of the absolute values of the time derivatives to obtain an approximation to the integral of $|a_t(x,t)|$ over $[a,b]$. The gstop function is the difference between this integral approximation and the tolerance.

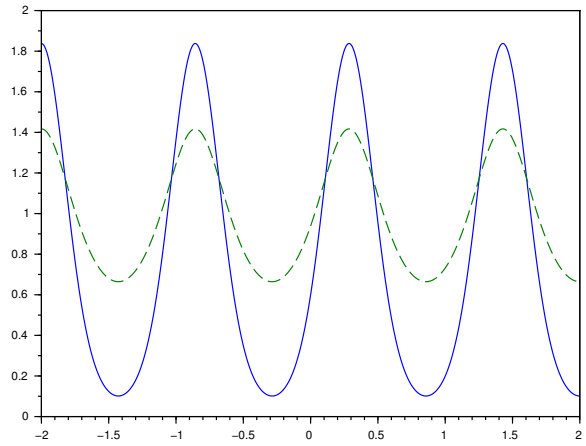

Figure 8: Numerical solution components, $a(x,t)$ (solid curve) and $h(t)$ (dashed curve), to **GMM** when the solution reaches steady state; this occurs when $t \approx 746.846$.

The main program, after calls BACOLI95_INIT and SETSOL, then calls BACOLI95 to determine the point in time where the first component of the approximate solution reaches steady state. We find that the solution reaches steady state when $t \approx 746.846$. The main program then prints out the solution and terminates. See Figure 8 for a plot of the two solution components at steady state.

4.8 Event detection in the Heat Equation with changes in the boundary conditions at unknown times

(This problem was communicated (private communication) to us from Sandeep Chatterjee while he was a student at the University of Toronto, Department of Computer and Electrical Engineering.)

For this problem, the PDE is the simple heat equation,

$$u_t(x,t) = \kappa u_{xx}(x,t),$$

where κ is a problem dependent parameter. The left boundary condition is initially $u_x(a,t) = \alpha$ and stays in this form until some unknown point in time at which $u(a,t)$ reaches a critical value, u_{crit}. At that point in time, the left boundary condition suddenly changes to $u_x(a,t) = -\gamma u(a,t)$. Similarly, the right boundary condition is initially $u_x(b,t) = \alpha$ until some unknown point in time (generally different from the time of the event at the left boundary) at which $u(b,t) = u_{crit}$, and then the right boundary condition suddenly changes

to $u_x(b,t) = -\gamma u(b,t)$. Depending on the values of the problem dependent parameters, α, γ, and u_{crit}, there can be sudden, potentially discontinuous, change in each of the boundary conditions.

The change in each boundary condition will be discontinuous unless the boundary continuity condition,

$$\alpha = -\gamma u_{crit},$$

is satisfied. (To see this, note that at the point in time when $u(a,t) = u_{crit}$, we switch from the original left boundary condition, $u_x(a,t) = \alpha$, to the new one, $u_x(a,t) = -\gamma u(a,t) = -\gamma u_{crit}$. Therefore, unless $\alpha = -\gamma u_{crit}$, there is a discontinuous change in the left boundary condition. A similar argument holds for the right boundary condition.)

We must also choose the initial solution so that it is consistent with the boundary conditions. That is, there cannot be a jump discontinuity in the solution for $t = 0$ and for x equal to either a or b. We will therefore choose an initial solution that (essentially) satisfies this condition. We will also choose the initial solution to be asymmetric so that the events at the left and right boundaries arise at different times.

For this example, we choose the initial solution to be

$$u(x,0) = 10e^{-100(x-0.75)^2} + \alpha x.$$

This initial solution has a peak equal to $10 + \alpha \times 0.75$ at $x = 0.75$ and then it rapidly tails off to αx to the left and right of $x = 0.75$. This makes the spatial derivative of the initial solution (essentially) consistent with the boundary conditions at t immediately after $t = 0$. (At $t = 0$, the first spatial derivative of the solution at the boundaries is (essentially) α; at t immediately after $t = 0$, the boundary conditions require that the first spatial derivative of the solution must equal α.)

We choose the tolerance to be 10^{-6}.

4.8.1 Continuous boundary conditions

We will refer to this instance of this problem as **HBC**.

In this subsection, we consider a version of this problem in which the parameters, α, γ, and u_{crit} are chosen to satisfy the above boundary continuity condition. This forces the first spatial derivative of the exact solution at each boundary to be continuous in time.

We will choose u_{crit} to have the value of 1.1. This means that the boundary continuity condition will force $\alpha = -\gamma \times 1.1$. One trivial choice for α and γ that satisfies this condition is $\alpha = \gamma = 0$ but this implies that the boundary conditions are constant ($u_x(a,t) = u_x(b,t) = 0$) throughout the computation. A more interesting choice is to require that α and γ are both non-zero and choose these parameters such that $\alpha = -\gamma \times 1.1$. Then, the left boundary condition transitions continuously from $u_x(a,t) = \alpha$ to $u_x(a,t) = -\gamma u(a,t)$ and the right boundary condition transitions continuously from $u_x(b,t) = \alpha$ to

$u_x(b,t) = -\gamma u(b,t)$. For this example, we choose $\alpha = 1$ and then γ becomes $1/1.1$.

Since the boundary conditions correspond to the algebraic conditions in the DAE system that is solved by DASKR, the above choice of parameters also means that the algebraic equations change in a continuous way at the time of each event, which in turn means that the B-spline coefficients that appear in the algebraic equations change in a continuous way. *However, an examination of the time derivatives of the algebraic equations shows that they do not change continuously at the time of an event. (To see this, note that the time derivative of the algebraic constraint corresponding to the left boundary condition prior to the event is simply* $u_{xt}(a,t)$ *but after the event, it is* $u_{xt}(a,t) + \gamma u_t(a,t)$*. A similar situation holds for the right boundary condition.) Consequently, the time derivative of at least one of B-spline coefficients that appears in each algebraic constraint must change discontinuously at the time of the event, in order to satisfy the new boundary condition that is imposed immediately after the event.* Due to this discontinuity in the time derivative of the B-spline coefficient(s), a cold start should be performed after each event. See below for further discussion on this point.

The organization of the RT routine in this case is straightforward. We call the VALUES routine to get solution values at the endpoints of the spatial domain and then the gstop vector function is $[U(0,t) - u_{crit}, U(1,t) - u_{crit}]^T$.

In the main program, we first need to call BACOLI95_INIT with NRT = 2 since there are two events to be tracked. We also call SETSOL as usual. Then BACOLI95 is called at $t = 0$ with the boundary routines BNDXA1 and BNDXB1, corresponding to the $u_x(a,t) = \alpha$ and $u_x(b,t) = \alpha$ conditions. The main program is setup to handle either boundary event happening first. At the end of this first return from BACOLI95, we access the JROOT array to determine which of the events has been detected. Based on an inspection of JROOT, the main program writes out a message indicating which of the two boundaries satisfied the event condition. The time, solution, and first spatial derivative values across the spatial domain are written out and then the computation proceeds. BACOLI95 is called again, with a cold start. The next call to BACOLI95 uses the BNDXA2 routine instead of the BNDXA1 routine if the left boundary event is detected, or the BNDXB2 routine instead of the BNDXB1 routine if the right boundary event is detected. The BNDXA2 routine imposes the $u_x(a,t) = -\gamma u(a,t)$ boundary condition while the BNDXB2 routine imposes the $u_x(b,t) = -\gamma u(b,t)$ boundary condition. Next BACOLI95 returns when the second event is found. Again, JROOT is examined to determine which event has been found and the time, solution, and first spatial derivative values across the spatial domain are written out. BACOLI95 is called again, with a cold start, and with inputs BNDXA2 and BNDXB2 so that both of the new boundary conditions, $u_x(a,t) = -\gamma u(a,t)$ and $u_x(b,t) = -\gamma u(b,t)$, are imposed. BACOLI95 returns at t_{out} and outputs the final solution and first spatial derivative values.

For this example, with the parameter values chosen as indicated above, we find that the first event occurs at the right boundary when $t \approx 5.45821 \times 10^{-3}$. The solution at this point in time is shown in Figure 9. We find that the second

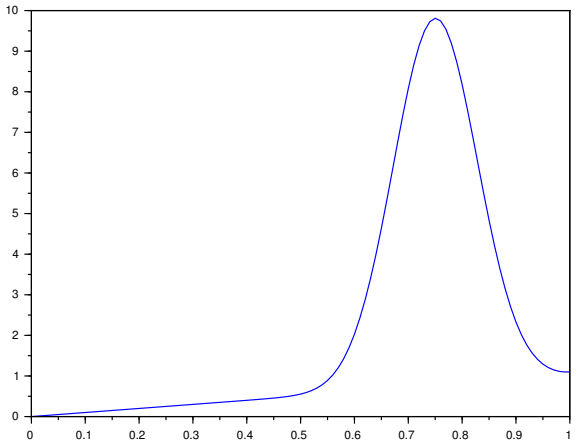

Figure 9: Numerical solution, $U(x,t)$, to **HBC**, when the solution at the right boundary reaches the critical value, $u_{crit} = 1.1$; this occurs when $t \approx 5.45821 \times 10^{-3}$.

event occurs at the left boundary when $t \approx 1.30837$. The solution at this point is time is shown in Figure 10. The integration terminates at $t_{out} = 5$. The solution at this point is time is shown in Figure 11.

For the computation described above, where we first determine the location of the event and then employ a cold start to restart DASKR after each event, we find that the time integration requires a total 316 accepted time steps. When we repeat the above computation, except that we perform a warm start after each event, we find that the computation requires DASKR to take 498 accepted time steps. As expected, due to the discontinuity in the time derivative of at least one B-spline coefficient immediately after each event, DASKR has substantial difficulty in stepping past the discontinuity. It is well-known that the presence of discontinuities can lead to substantial difficulties for the time integration. See, e.g., [13], [11], where the multi-step and Runge-Kutta methods, respectively, are studied for ODEs where discontinuities arise. See, e.g., [18] and references within, for work on the determination and handling of discontinuities for the DAE case.

4.8.2 Discontinuous boundary conditions

In this subsection we choose the parameters so that there are discontinuous changes in the boundary conditions after each event. We will refer to this instance of the problem as **HBCD**. We choose $\alpha = 0$, $u_{crit} = 1$, and $\gamma = 0.5$. This means that there will be jump discontinuities, of magnitude 0.5, imposed

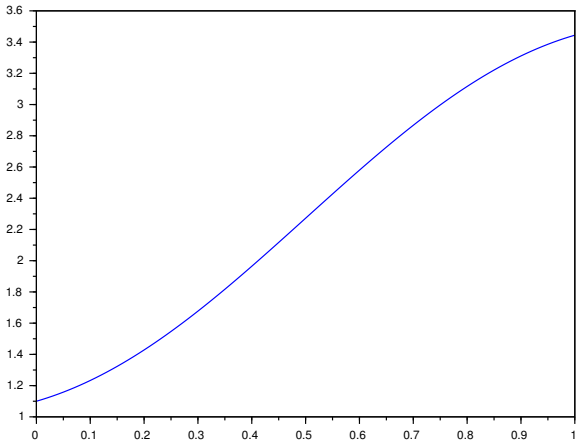

Figure 10: Numerical solution, $U(x,t)$, to **HBC**, when the solution at the left boundary reaches the critical value, $u_{crit} = 1.1$; this occurs when $t \approx 1.30837$.

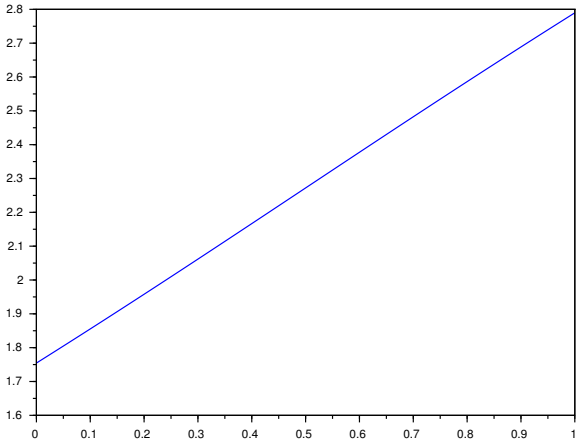

Figure 11: Numerical solution, $U(x,t)$, to **HBC**, when $t = 5$.

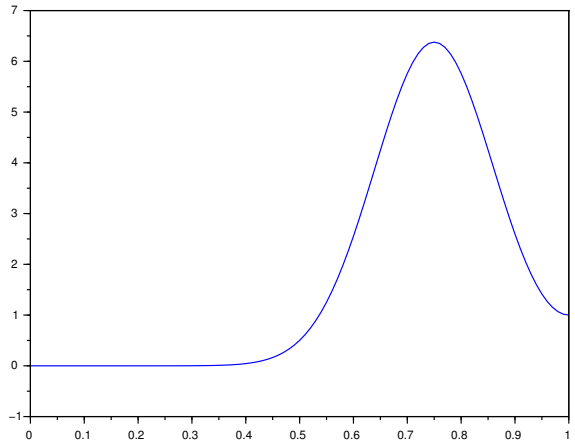

Figure 12: Numerical solution, $U(x,t)$, to **HBCD**, when the solution at the right boundary reaches the critical value, $u_{crit} = 1$; this occurs when $t \approx 3.64470 \times 10^{-2}$.

on the first spatial derivative of the solution at the boundaries.

As discussed earlier, the boundary conditions correspond to the algebraic equations in the DAE system that is given to DASKR. In this instance of the problem, since there is a jump discontinuity in one of the boundary conditions after each event, there is a jump discontinuity in the corresponding algebraic equation, which in turn forces a jump discontinuity in at least one of the B-spline coefficients at the time of each event. Since the discontinuous change occurs in an algebraic equation, it is handled by the Newton iteration inside the DAE solver, i.e., the B-spline coefficient can be changed to satisfy the post-event algebraic equation, but the difficulty arises with the time integration of the ODEs that appear in the DAE system. Some of these ODEs depend on the discontinuous B-spline coefficient(s), and these discontinuous ODEs will lead to difficulties for the time integration. We therefore expect that the presence of a discontinuity in at least one of the B-spline coefficients will lead to substantial inefficiencies in the time integration as DASKR attempts to step past the discontinuity, unless a cold start is performed after each event.

For this case we find that the first event occurs at the right boundary for $t \approx 3.64470 \times 10^{-2}$. The solution at this point in time is shown in Figure 12. BACOLI95 is then restarted with a cold start and the discontinuous right boundary condition is imposed. We find that the second event occurs at the left boundary for $t \approx 1.16926$. The solution at this point in time is shown in Figure 13. We then restart BACOLI95 again with a cold start and impose the discontinuous left boundary condition. The code then integrates to $t_{out} = 5$.

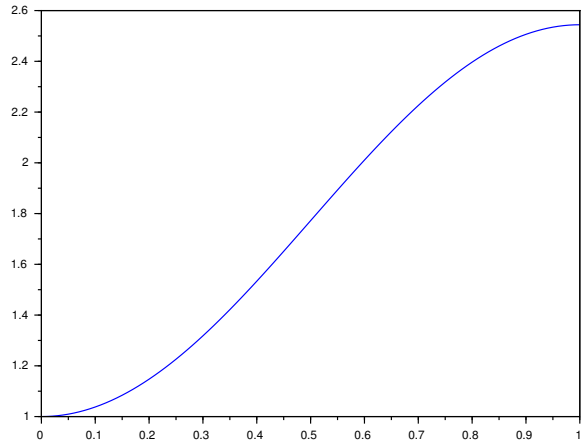

Figure 13: Numerical solution, $U(x,t)$, to **HBCD**, when the solution at the left boundary reaches the critical value, $u_{crit} = 1$; this occurs when $t \approx 1.16926$.

The solution at this point is time is shown in Figure 14.

For the computation described above, DASKR requires 291 accepted time steps to complete the integration. When we repeat the above computation, except that we perform a warm start after each event, we find that the computation requires DASKR to take 568 accepted time steps. As expected, due to discontinuity after each event, DASKR has substantial difficulty in stepping past the two event times unless cold starts are imposed.

Comparing the number of time steps taken by DASKR for the cases of continuous and discontinuous boundary conditions, we note that the performance of DASKR is comparable in both cases. This is due to the fact that the time integrator must deal with discontinuities immediately after each event in either case. In particular, even for the case where the boundary conditions change continuously at the time of the event, the time derivatives of the boundary conditions do not and this leads to discontinuities in the time derivatives of the B-spline coefficients at the time of the event.

5 Summary, Conclusions, and Future Work

This report introduces, BACOLIKR, a new error control PDE solver that features time and space dependent event detection. To our knowledge, this is the only error control PDE solver with this capability. The report describes the substantial modifications to the earlier error control PDE solver, BACOLI and the time-integrator DASKR, that were required in order to obtain BACOLIKR.

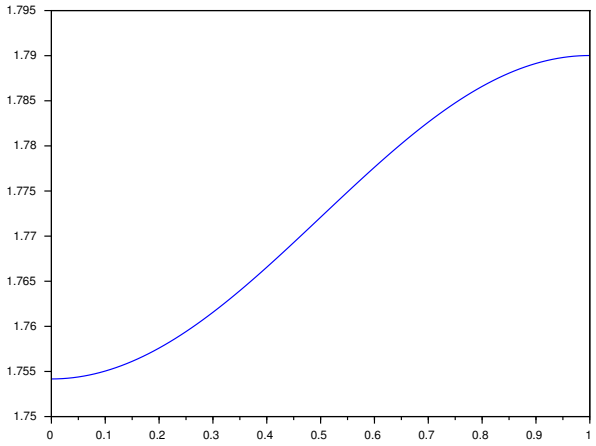

Figure 14: Numerical solution, $U(x,t)$, to **HBCD**, when $t = 5$.

The report provides a number of examples to demonstrate how a variety of event detection problems can be handled by the new solver.

We find that, with a relatively straightforward customization of the rootfinding routine that characterizes a given event or events, it is possible to use BACOLIKR to fairly easily solve a variety of PDE based event detection problems.

We observe that when the problem is altered after an event and there is an expectation that the computation should continue past the event, it is important that BACOLIKR be restarted with a cold start due to the well-known difficulties that arise for error control time integration algorithms in the presence of discontinuities in either the solution or the derivative of the solution to the DAE system arising from the discretization of the PDE(s) and boundary conditions.

Regarding future work, as mentioned earlier, since DASKR also has a feature that allows it to efficiently treat large DAE systems using Krylov methods, it would be worthwhile to modify BACOLIKR to take advantage of this capability in order to improve the efficiency of the solver for problems in which the number of PDEs together with the number of subintervals in the spatial mesh lead to large DAE systems.

References

[1] S.M. Allen and J.W. Cahn. Ground state structures in ordered binary alloys with second neightbor interations. *Acta Metall.*, 20:423–433, 1972.

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

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

[4] I.E. Athanasakis, M.G. Papadomanolaki, E.P. Papadopoulou, and Y.G. Saridakis. Discontinuous Hermite collocation and diagonally implicit RK3 for a brain tumour invasion model. *Proceedings of the World Congress on Engineering*, I, 2013.

[5] P.N. Brown, A.C. Hindmarsh, and L. R. Petzold. Using Krylov methods in the solution of large-scale differential-algebraic systems. *SIAM J. Sci. Comp.*, 15:1467–1488, 1994.

[6] P.N. Brown, A.C. Hindmarsh, and L. R. Petzold. Consistent initial condition calculation for differential-algebraic systems. *SIAM J. Sci. Comp.*, 19:1495–1512, 1998.

[7] J.H. Cerutti and S.V. Parter. Collocation methods for parabolic partial differential equations in one space dimension. *Numer. Math.*, 26(3):227–254, 1976.

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

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

[10] J. Douglas, Jr. and T. Dupont. *Collocation Methods for Parabolic Equations in a Single Space Variable*. Lecture Notes in Mathematics, Vol. 385. Springer-Verlag, Berlin, 1974.

[11] W.H. Enright, K.R. Jackson, S.P. Nørsett, and P.G. Thomsen. Effective solution of discontinuous IVPs using a Runge-Kutta formula pair with interpolants. *Appl. Math. Comp.*, 27:313–355, 1988.

[12] W.F. Finden. An error term and uniqueness for Hermite-Birkhoff interpolation involving only function values and/or first derivative values. *J. Comput. Appl. Math.*, 212(1):1–15, 2008.

[13] C.W. Gear and O. Osterby. Solving ordinary differential equations with discontinuities. *ACM Trans. Math. Softw.*, 10:23–44, 1984.

[14] A. Gierer and H. Meinhardt. A theory of biological pattern formation. *Kybernetik*, 12:30–39, 1972.

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

[16] K.L. Hiebert and L.F. Shampine. Implicitly defined output points for solutions of ODEs. *Sandia Report SAND80-0180*, 1980.

[17] P. Keast. LAMPAK: a Fortran package for solving certain almost block diagonal matrices. *Unpublished software*, 1982.

[18] G. Mao and L.R. Petzold. Efficient integration over discontinuities for differential-algebraic systems. *Comput. Math. Applic.*, 43:65–79, 2002.

[19] L.R. Petzold. *A Description of DASSL: A Differential/Algebraic System Solver*. Sandia Labs, Livermore, CA, 1982.

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

[21] J. Pew, Z. Li, C. Tannahill, P.H. Muir, and G. Fairweather. Performance analysis of error-control B-spline Gaussian collocation software for PDEs. *Comput. Math. Appl.*, 77(7):1888–1901, 2019.

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

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

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

[25] W. Zhang. Diffusive effects on a catalytic surface reaction: an initial boundary value problem in reaction-diffusion-convection equations. *J. Bifur. Chaos*, 3:79–95, 1993.