On spatial adaptivity and interpolation when using the method of lines

Martin Berzins [a] Philip J. Capon [b] Peter K. Jimack [a]

[a] *School of Computer Studies, University of Leeds, Leeds, LS2 9JT, UK*

[b] *Tessella Support Services plc, Abingdon, UK*

The solution of time-dependent partial differential equations with discrete time static remeshing is considered within a method of lines framework. Numerical examples in one and two space dimensions are used to show that spatial interpolation error may have an important impact on the efficiency of integration. Analysis of a simple problem and of the time integration method is used to confirm the experimental results and a computational test for monitoring the impact of this error is derived and tested.

Key words: Adaptive remeshing, method of lines, interpolation errors.

1 Introduction

A common approach to finding numerical solutions of time-dependent partial differential equations (PDEs) is to use the method of lines (MOL). This technique involves reducing an initial boundary value problem (IBVP) to a system of ordinary differential equations (ODEs) in time through the use of a discretization in space. This system of ODEs, which takes the form of an initial value problem (IVP), can then be solved using standard software. The spatial discretization may take many forms (e.g. finite difference, finite element or finite volume) but the resulting ODE system can be solved using standard initial value problem software, which may use a variable time-step/variable order approach with time local error control.

Although it is possible to use a fixed spatial mesh in such an approach it has long been recognized, [10,13], that adapting the spatial mesh offers important advantages as regards the efficiency and accuracy of the solution process, particularly for problems with moving or highly localized features. There are in general two approaches used to modify a spatial mesh. One approach is to continuously move the grid in time, [15,19,20], often by solving an extra set of differential equations to determine the mesh point positions.

The alternative approach, often termed static rezoning (e.g. [13]) or discrete remeshing, is to change the mesh only at discrete times, interpolate the solution from the old mesh to the new mesh, and then restart the time integration. This approach has long been used for time dependent PDEs with one of the earliest methods being that of Bank and Dupont [2]. The approach adopted by Bank and Dupont in 1976 in their MDTRI code was to use a triangular mesh and data structures similar to those later used in the well-known PLTMG steady code [3]. The static rezoning approach was also used in a number of petroleum engineering codes (see [9] and the references therein) and for more general problems: [5,13] and the papers in [1], for example.

There are two ways to implement static rezoning. The simplest approach is to interpolate the solution from the old mesh to the new mesh and restart the time integrator as though solving a new problem. This may be inefficient however as the computational overhead associated with starting up an ODE solver is considerable. For an IBVP in which spatial remeshing will occur quite frequently these continual restarts of the ODE solver can have an extremely detrimental effect on the overall efficiency of the computation. Hence an alternative is to interpolate more than just the latest estimate of the solution from the old spatial mesh to the new one, as in [4] for example. Typically this means interpolating all of the history vectors that are used by the ODE solver, as well as carrying over the latest time-step size and order. Computational results have shown (see section 3) that this approach can reduce the overhead of the discrete remeshing approach significantly and, should it fail, it is always possible to restart by only interpolating the solution, [4].

This static rezoning approach is a common form of spatial adaptivity and general purpose NAG library software is available which implements this using finite difference semi-discretizations, implicit time integrators , and cubic splines to perform the interpolation: see [4]. In this particular case the new mesh is calculated by using an equidistribution principle (as outlined in [4]). Other related techniques include the use of finite element semi-discretizations with error estimates and spatial adaptivity via local mesh refinement (see [14,18] for example).

The motivation for this paper has come from the use of static rezoning in connection with compressible Navier Stokes calculations [8] in which the choice of interpolant appeared to have important consequences for the accuracy and efficiency of the integration. This paper will demonstrate that this is indeed the case and propose a means of identifying when this occurs for a specific class of time integration methods. A description of the method of lines discrete remeshing approach and the time integration algorithm will be given in section 2. Two computational examples will be used in section 3 to illustrate how the performance of the time integrator may be degraded. The second of these examples is analyzed in section 4 and the analysis extended to consider a more

general effect of the residual error introduced through discrete remeshing. As a result of this analysis a means of monitoring the discrete remeshing error is suggested. The performance of this monitor is illustrated in section 5.

2 Time integration with discrete remeshing

Consider a single time-dependent PDE of the form

$$\frac{\partial u}{\partial t} \;=\; L(u(x,t),x,t)\,, \tag{1}$$

together with appropriate boundary and initial conditions. Suppose that this PDE is discretized in space on a mesh $\underline{X}_N$ of N spatial points leading to the system of N ordinary differential equations:

$$\underline{F}_N\;(t,\;\underline{Y}_N(t),\;\underline{Y}'_N(t)\;)\;=\;\underline{0}\,,\quad \underline{Y}_N(0)\;=\;\underline{C}$$

where $\underline{Y}_N(t)$ is the exact solution of this ODE system and its ith component corresponds to the solution of the discretized PDE at the ith spatial mesh point. The initial condition is obtained by assigning the initial value at the ith mesh point to the ith ODE component, or by projecting the initial PDE solution onto a finite element space.

In the remainder of the paper it will be assumed that for the finite element or finite difference semi-discretizations used the ODE system has the form:

$$\underline{F}_N(t,\;\underline{Y}_N(t),\;\underline{Y}'_N(t)\;)\;\equiv\;A_N\;\underline{Y}'_N(t)\;-\;\underline{G}_N(\;\underline{Y}_N(t),\;t)\;=\;\underline{0},\tag{2}$$

where the matrix A_N is strictly positive definite in the finite element case or the identity matrix in the finite difference case.

2.1 Time integration

In order to integrate the ODE system (2) use will be made of general initial value ODE integrators, such as those in [4]. For the sake of clarity here however the time integration method discussed in this paper will be the first/second-order variable-step, variable-order Backward Differentiation code as outlined in [6] (which will henceforth be referred to as BDF2). The use of such an adaptive time-stepping code produces numerical approximations, $\underline{y}_N(t_j)$ and

$\underline{y}'_N(t_j)$, to the true ODE solution and its derivative, $\underline{Y}_N(t_j)$ and $\underline{Y}'_N(t_j)$ respectively, at a set of discrete times t_j. At these discrete times the numerical solution satisfies the perturbed equation

$$\underline{F}_N(t_j,\ \underline{y}_N(t_j),\ \underline{y}'_N(t_j)\ )\ =\ \underline{R}_N(t_j)\ , \tag{3}$$

where $\underline{R}_N(t)$ is the numerical residual due, for example, to not exactly solving the nonlinear equations at each time-step. The BDF2 formula will now be used to explain how the numerical solution is calculated as a necessary precursor to the analysis of the discrete remeshing case. It is then straightforward to extend this analysis to multistep formulae of other orders and other implementations. The BDF2 method applied to the ODE given by equation (2) is given by Skeel [21] as:

$$A_N\ \left(\frac{3\underline{y}_N(t_j) - \underline{z}_j}{2k_j}\right)\ =\ \underline{G}_N(\underline{y}_N(t_j),\ t_j)\ ,$$

where

$$\underline{z}_j = (3 + r_j^2)\underline{y}_N(t_{j-1}) - r_j^2\ \underline{y}_N(t_{j-2}) + (1 - r_j)\ k_j\ \underline{y}'_N(t_{j-1})\ ,$$

$r_j = k_j/k_{j-1}$, k_j is the size of the j^{th} time-step and $t_j = \sum_{i=1}^{j} k_i$.

The Nordsieck implementation of this formula makes use of a stored history array of values for each component, consisting of

$$\left[\underline{y}_N(t_j)\ ,\ k_j\underline{y}'_N(t_j)\ ,\frac{k_j^2}{2}\underline{y}''_N(t_j)\right]\ , \tag{4}$$

and the predictor-corrector approach outlined below to compute the new solution and its time derivatives at the next time-step (see [24]).

The predicted values $\underline{y}^P_N(t_j)$, $\underline{y}'^P_N(t_j)$ and $\underline{y}''^P_N(t_j)$ are defined by:

$$\underline{y}^P_N(t_j) = \underline{y}_N(t_{j-1}) + k_j\underline{y}'_N(t_{j-1}) + \frac{1}{2}\ k_j^2\underline{y}''_N(t_{j-1})\ ,$$

$$\underline{y}'^P_N(t_j) = \underline{y}'_N(t_{j-1}) + k_j\underline{y}''_N(t_{j-1})$$

and

$$\underline{y}''^P_N(t_j) = \underline{y}''_N(t_{j-1})\ .$$

An update vector, $\underline{\alpha}$, is then defined by solving the system of nonlinear equations:

$$A_N \ (\underline{y}'^P_N(t_j) + \underline{\alpha}) = \underline{G}_N(\underline{y}^P_N(t_j) + \frac{2k_j}{3}\underline{\alpha}, t_j) \ .$$

The predicted values are then corrected according to:

$$\underline{y}_N(t_j) = \underline{y}^P_N(t_j) + \ \frac{2k_j}{3}\underline{\alpha} \ , \tag{5}$$

$$\underline{y}'_N(t_j) = \underline{y}'^P_N(t_j) + \underline{\alpha} \tag{6}$$

and

$$\underline{y}''_N(t_j) = \underline{y}''^P_N(t_j) + \frac{2}{3k_j}\underline{\alpha} \ . \tag{7}$$

The next stage is to ensure that the estimated local error over each time-step is less than some user-defined tolerance. The standard local error control requirement is that

$$c\|\underline{\alpha}\|_W \leq 1 \ , \tag{8}$$

where the subscript W refers to a weighted norm whose weights reflect user-supplied tolerance values and c is a method-dependent constant.

2.2 Restarting the ODE solver after remeshing

The discrete remeshing procedures used in [4,13,14], along with many other approaches, all consist of three main components. The first component is an algorithm for constructing a new mesh of M points $\underline{X}_M$ from the existing mesh of N points $\underline{X}_N$ and solution components. Typically the new mesh is defined by trying to minimize error criteria or certain space derivatives in the solution. The second component is an interpolation operator which defines the solution on the new mesh $\underline{y}_M(t_j)$ in terms of the solution on the old mesh $\underline{y}_N(t_j)$:

$$\underline{y}^\chi_M(t_j) \ = \ \underline{I}^\chi_{M,N}(\underline{y}_N(t_j)) \tag{9}$$

where χ refers to the type of spatial interpolant: $\chi = L$ being a linear interpolant and $\chi = C$ being a cubic spline interpolant for example. The third component in the remeshing algorithm is a means of restarting the time integrator.

2.2.1 *Full restarts*

One approach is to interpolate only the solution from the old mesh to the new mesh, to recalculate the time derivatives and to restart the time integrator with a first order method. This method is termed a full restart.

2.2.2 *Flying restarts*

The second approach (used by [4] and in the associated NAG library routines) is to interpolate the whole of the history array used by the integrator (e.g. (4)) and to attempt to continue integration as though the mesh had not been changed. In this case, as well as (9),

$$\underline{y}'^{\chi}_M(t_j) = \underline{I}^{\chi}_{M,N}(\underline{y}'_N(t_j)) \,, \tag{10}$$

$$\underline{y}''^{\chi}_M(t_j) = \underline{I}^{\chi}_{M,N}(\underline{y}''_N(t_j)) \tag{11}$$

and an attempt is made to continue integration with the same step size and order as would have been used had remeshing not taken place. This "flying restart" approach tries to ensure that the step size is not needlessly affected by the spatial remeshing. Should the time integrator meet difficulties it is then possible to revert to the first approach (i.e. a full restart). In the case of PDEs in one space dimension this approach appears to work well when cubic spline spatial interpolation is used with second-order finite difference methods [4,22]. An important side-effect of performing a flying restart is that although the solution components and time derivatives are interpolated onto the new mesh, the effect of this on the residual of the ODE system is more complicated and so in general the current residual is not interpolated onto the new mesh:

$$\underline{R}^{\chi}_M(t_j) \neq \underline{I}^{\chi}_{M,N}(\underline{R}_N(t_j)) \,.$$

Here $\underline{R}_N(t_j)$ is given by (3) and

$$\underline{R}^{\chi}_M(t_j) = \underline{F}_M(t_j,\, \underline{y}^{\chi}_M(t_j),\, \underline{y}'^{\chi}_M(t_j)\,) \,, \tag{12}$$

where $\underline{F}_M(\cdot,\cdot,\cdot)$ comes from the spatial discretization of (1) on $\underline{X}_M$ (as opposed to $\underline{X}_N$) and $\underline{y}^{\chi}_M(t_j)$ and $\underline{y}'^{\chi}_M(t_j)$ are given by (9) and (10) respectively. This is explained fully in section 4.

3 Computational examples

The main motivation for this work comes from the need to use discrete time remeshing when solving time-dependent problems in two and three space dimensions on triangular and tetrahedral meshes. The following example illustrates this type of calculation and points to a potential problem area.

3.1 A 2-d Navier-Stokes example

In [7] the non-dimensionalized compressible Navier-Stokes equations are presented in terms of the primitive variables of density, velocity and temperature. In [8] a method for the solution of these equations is presented based upon the Galerkin least-squares approach of Hughes *et al* ([12,23]), along with the use of adaptive h-refinement in space (this is a particular example of discrete remeshing, described in more detail in [14,18] for example). Unlike in [12] however the approach of [8] is to only take finite element discretizations of the spatial operators, thus yielding a nonlinear set of ODEs in time, which may be solved using the BDF2 method for example.

Typical results in [8], such as those obtained for the unsteady flow around a NACA0012 aerofoil at $Re = 5000$ with a free stream Mach number, M_∞, of 0.55 and an angle of attack of 8.34°, compare favourably with those reported elsewhere (see [25] for example). It may be observed however that when piecewise linear interpolation is used to transfer solution data from one grid to another after spatial adaptivity has occurred, the size of the time-step is frequently reduced significantly after each discrete remesh, even when a flying restart is attempted. This is illustrated in Figure 1 which shows the value of the time variable, t, in the Navier-Stokes equations against the number of time-steps taken during a typical run. These reductions in the time-step after mesh refinement has occurred clearly cause a substantial computational overhead, leading to a significant loss of efficiency in the overall algorithm.

3.2 A simple test problem

Due to the complexity of the above example we now consider a much simpler test problem, which will then be analyzed in order to try to isolate the cause of the reductions in the time-step after discrete remeshing has occurred. Consider the one-dimensional reaction-diffusion equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + f(u) \qquad \text{where } u(x,t) : (0,10) \times (0,1.5) \longrightarrow \Re \qquad (13)$$

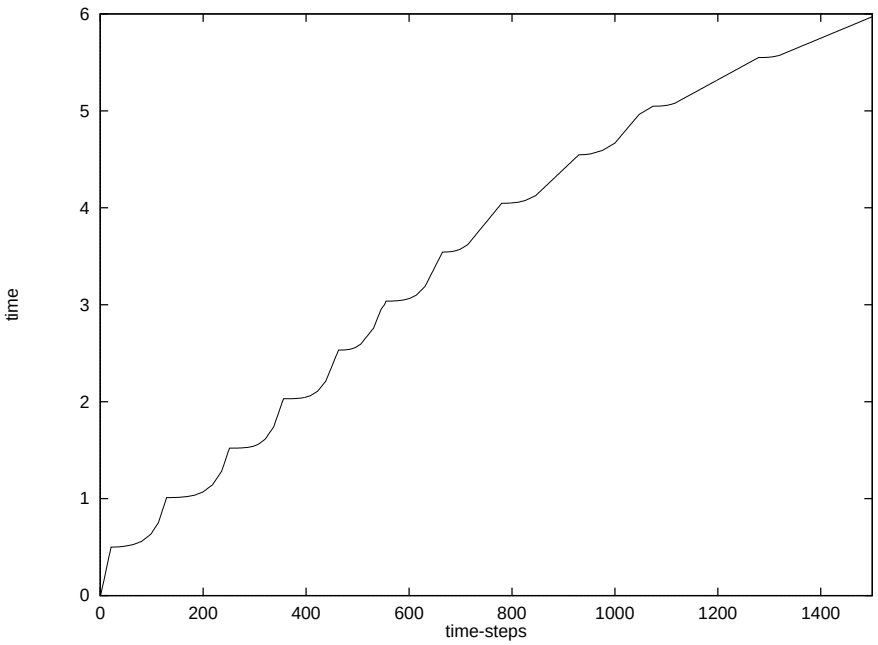

Fig. 1. Plot of time against time-steps taken for the unsteady Navier-Stokes problem.

and $f(u) = u^2(1-u)$, with initial and Dirichlet boundary conditions consistent with the analytic solution

$$u(x,t) = \frac{1}{1 + e^{p(x-pt)}} , \qquad (\text{where } p = \tfrac{1}{\sqrt{2}}).$$

The standard piecewise-linear Galerkin finite element semi-discretization in space with $N+1$ elements (as described in [16] for example) leads to an ODE system of the form of equation (2):

$$A_N \, \underline{Y}'_N(t) \; - K_N \, \underline{Y}_N(t) \; - \; \underline{f}_N(\, \underline{Y}_N(t), \; t) \; = \; 0, \qquad (14)$$

where A_N and K_N are the Galerkin mass and stiffness matrices respectively and the vector $\underline{f}_N$ incorporates both the reaction terms in the PDE and its time-dependent Dirichlet boundary conditions. The vector $\underline{Y}_N(t)$ contains the solution values at each of the interior knot points at the time t and, for the conventional ordering of these unknowns, the matrices A_N and K_N are tridiagonal.

In our computational experiments the local spatial error indicator used is the 1-norm of the gradient of the solution on each element. Although this is crude, it is sufficient for illustrative purposes. The form of static rezoning that is used is again simple h-refinement where, in one space dimension, each element of the mesh may be bisected at the end of each time-step if the local spatial error indicator exceeds some fixed tolerance. When this occurs a new degree of freedom is introduced to the ODE system (2) for each element that

is bisected (this being the solution value at its midpoint).

Figure 2 shows plots of the value of the time variable in (13) against the number of time-steps taken (including those taken but rejected) by the ODE solver (BDF2) for three different computations, each with the same spatial and temporal error tolerances. In the first calculation a full restart of the ODE solver is performed when the mesh is refined, with the initial data taken from the piecewise linear interpolant of the final value on the old mesh. In the second case linear interpolation is used in a flying restart for the new ODE system. Finally, a third run is considered in which cubic spline interpolation is used before attempting a flying restart.

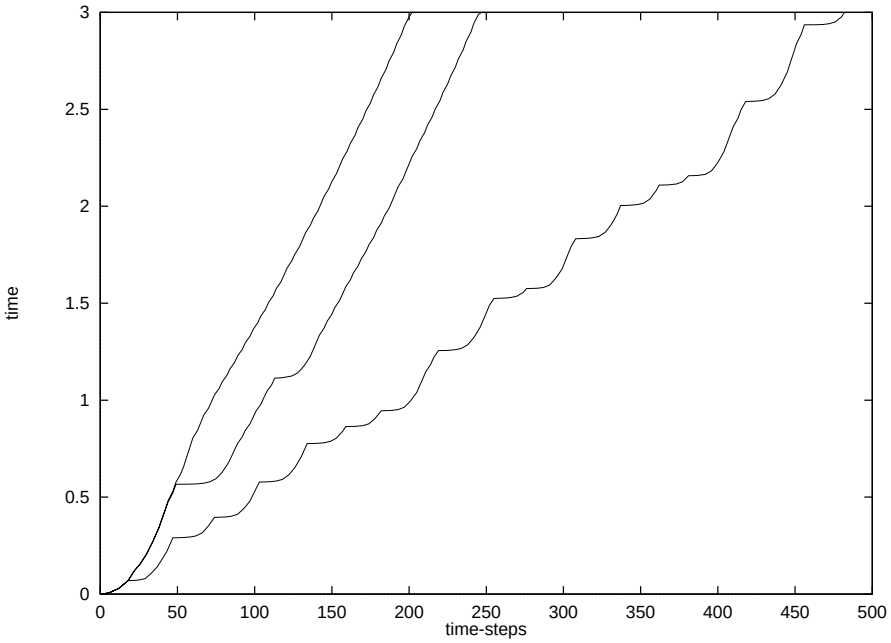

Fig. 2. Plot of time against time-steps taken for the reaction-diffusion problem using (from left to right): (i) flying restarts after cubic spline interpolation, (ii) flying restarts after linear interpolation, (iii) full restarts.

Inspection of Figure 2 shows that there is a significant overhead associated with performing a complete restart of the ODE solver each time the spatial mesh is refined. Whenever the time-step size begins to grow, the spatial remeshing causes it to be reduced as a new ODE system is tackled. In contrast, the flying restart allows the ODE solver to carry on integrating with the same order and step size as before, even though we are now solving a different semi-discrete system. It is notable however that when linear interpolation is used, there are still occasions where the step size (and order) are reduced by the ODE solver (in order to satisfy its local error tests). For this example, this does not occur when cubic spline interpolation is used and thus only using linear interpolation to transfer between meshes is, in some sense, not always sufficiently accurate. This observation matches that made for the

Navier-Stokes problem in subsection 3.1 where the same deficiencies appear to occur when piecewise linear interpolation is used.

4 An analysis of discrete remeshing

In this section the simpler of the two examples above is used to show that the choice of interpolation method may make a significant difference to the residual of the ODE system after interpolation onto the new mesh. An analysis of the more general problem (2) is then used to show how this residual affects the performance of the time integrator and to suggest a means of monitoring these effects.

4.1 The ODE residual after interpolation

This subsection demonstrates that for problem (13) (and any other 1-d problem that leads to a semi-discrete system of the form (14)), the residual which results from the application of cubic spline interpolation after refinement is superior to that which results from the application of linear interpolation, at least in the asymptotic limit as $h \to 0$.

For algebraic simplicity we consider the specific case of a uniform mesh, containing $N+1$ elements of size h, being uniformly refined at time t_j into $2N+2$ elements of size $\frac{h}{2}$. From (2), (3) and (14), we see that the residual of the ODE system before refinement may be written as

$$\underline{R}_N(t_j) = A_N \; \underline{y}'_N(t_j) - K_N \; \underline{y}_N(t_j) - \underline{f}_N(\underline{y}_N(t_j), t_j) \; , \tag{15}$$

where $\underline{y}_N(t_j) \in \Re^N$. Using the notation that $\underline{X}_N = (x_1, ..., x_N)^T$, $\underline{y}_N(t_j) = \underline{y} = (y_1, ..., y_N)^T$ and $\underline{y}'_N(t_j) = \underline{y}' = (y'_1, ..., y'_N)^T$, the ith component of this residual at time t_j is

$$R_{N,i} = \frac{h}{6}\left[y'_{i+1} + 4y'_i + y'_{i-1}\right] - \frac{1}{h}\left[y_{i+1} - 2y_i + y_{i-1}\right] - \int_{x_{i-1}}^{x_{i+1}} f(\sum_{j=0}^{N+1} y_j \alpha_j^N)\alpha_i^N \, dx \; , \tag{16}$$

where y_0 and y_{N+1} are the Dirichlet conditions at the end points x_0 and x_{N+1} respectively, and each of the $\alpha_i^N(x)$ are the usual linear finite element basis functions on $\underline{X}_N$.

At this point it is notationally convenient to introduce two continuous functions of x on (x_0, x_{N+1}): $y(x)$ is a four times differentiable function which takes the values $\underline{y}$ at $\underline{X}_N$, and $y'(x)$ is a twice differentiable function which takes the values $\underline{y}'$ at $\underline{X}_N$. It then follows that

$$R_{N,i} = hy'(x_i) - hy_{xx}(x_i) - hf(y_i) + O(h^3) \ , \tag{17}$$

where the integral in (16) has been approximated using the trapezium rule on each subinterval of (x_0, x_{N+1}). This residual is $O(h)$ and so if the mesh size is halved the residual should similarly decrease in size. We now define the linear interpolation operator $\underline{I}^L_{M,N} : \Re^N \to \Re^M$ and the cubic spline interpolation operator $\underline{I}^C_{M,N} : \Re^N \to \Re^M$, where $M = 2N + 1$, by

$$\underline{I}^L_{M,N}(\underline{y}_N) = \begin{bmatrix} \frac{1}{2}(y_0 + y_1) \\ y_1 \\ \vdots \\ y_i \\ \frac{1}{2}(y_i + y_{i+1}) \\ y_{i+1} \\ \vdots \\ y_N \\ \frac{1}{2}(y_N + y_{N+1}) \end{bmatrix} \ ,$$

$$\underline{I}^C_{M,N}(\underline{y}_N) = \begin{bmatrix} \frac{1}{2}(y_0 + y_1) + \frac{h}{8}(m_0 - m_1) \\ y_1 \\ \vdots \\ y_i \\ \frac{1}{2}(y_i + y_{i+1}) + \frac{h}{8}(m_i - m_{i+1}) \\ y_{i+1} \\ \vdots \\ y_N \\ \frac{1}{2}(y_N + y_{N+1}) + \frac{h}{8}(m_N - m_{N+1}) \end{bmatrix} \ ,$$

where $\underline{y}^T_N = (y_1, ..., y_N)$, y_0 and y_{N+1} are known boundary values, and the terms $m_0, ..., m_{N+1}$, which are the nodal first space derivative values of the

spline, are obtained from the solution of the tridiagonal system

$$\begin{bmatrix} 2 & 1 & 0 & \dots & 0 & 0 & 0 \\ \frac{1}{2} & 2 & \frac{1}{2} & \dots & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 2 & \dots & 0 & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & \dots & \frac{1}{2} & 2 & \frac{1}{2} \\ 0 & 0 & 0 & \dots & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ \vdots \\ m_N \\ m_{N+1} \end{bmatrix} = \begin{bmatrix} \frac{3}{h}(y_1 - y_0) \\ \frac{3}{2h}(y_2 - y_0) \\ \frac{3}{2h}(y_3 - y_1) \\ \vdots \\ \frac{3}{2h}(y_{N+1} - y_{N-1}) \\ \frac{3}{h}(y_{N+1} - y_N) \end{bmatrix}.$$

(See [17], for example, for details of how this system is obtained for a natural spline.) The different vectors that are obtained from using these interpolants are denoted as in equations (9) and (10), where, again, $M = 2N + 1$. Using this notation, the residual after refinement and linear interpolation have taken place may be defined by:

$$\underline{R}_M^L(t_j) \;=\; A_M \; \underline{y}_M'^L(t_j) - K_M \; \underline{y}_M^L(t_j) - \underline{f}_M(\underline{y}_M^L(t_j), t_j) \tag{18}$$

where the components of this vector are given by

$$\begin{aligned} R_{M,2i}^L = & \frac{h}{12}\left[\frac{1}{2}y'_{i+1} + 5y'_i + \frac{1}{2}y'_{i-1}\right] - \frac{1}{h}\left[y_{i+1} - 2y_i + y_{i-1}\right] - \\ & \int_{x_i - \frac{h}{2}}^{x_i + \frac{h}{2}} f(\sum_{j=0}^{M+1} y_j^M \alpha_j^M)\alpha_{2i}^M \, dx \\ = & \frac{h}{2}y'(x_i) - h y_{xx}(x_i) - \frac{h}{2}f(y_i) + O(h^3) \end{aligned} \tag{19}$$

and

$$\begin{aligned} R_{M,2i+1}^L = & \frac{h}{12}\left[3y'_{i+1} + 3y'_i\right] - \int_{x_i}^{x_{i+1}} f(\sum_{j=0}^{M+1} y_j^M \alpha_j^M)\alpha_{2i+1}^M \, dx \\ = & \frac{h}{2}\left[\frac{y'(x_{i+1}) + y'(x_i)}{2}\right] - \frac{h}{2}f(\frac{y_i + y_{i+1}}{2}) + O(h^3) \\ = & \frac{h}{2}\left[\frac{y'(x_i) + y'(x_{i+1})}{2}\right] - \frac{h}{2}\left[\frac{f(y_i) + f(y_{i+1})}{2}\right] + O(h^3)\,. \end{aligned} \tag{20}$$

Note that the diffusive terms, $K_M \underline{y}_M^L(t_j)$, are, incorrectly, unchanged in size at the common points of the old and new mesh and are, also incorrectly, absent

from the new mesh points. In the other case, where cubic spline interpolation is used, then

$$\underline{R}^C_M(t_j) \;=\; A_M\ \underline{y}'^C_M(t_j) - K_M\ \underline{y}^C_M(t_j) - \underline{f}_M(\underline{y}^C_M(t_j), t_j) \tag{21}$$

where the components of this vector are

$$R^C_{M,2i} = R^L_{M,2i} + \frac{h^2}{96}\left[m'_{i-1} - m'_{i+1}\right] - \frac{1}{4}\left[m_{i-1} - m_{i+1}\right] \tag{22}$$

and

$$R^C_{M,2i+1} = R^L_{M,2i+1} + \frac{h^2}{24}\left[m'_i - m'_{i+1}\right] + \frac{1}{2}\left[m_i - m_{i+1}\right]\ , \tag{23}$$

and where the m'_i values are the nodal first space derivatives of the cubic spline which interpolates the mesh point values $\underline{y}'_N(t_j)$.

The following lemma demonstrates that the size of $\underline{R}^C_M(t_j)$ is generally much closer to the size of the residual before refinement took place, $\underline{R}_N(t_j)$, than that of $\underline{R}^L_M(t_j)$, at least in the limit as $h \to 0$.

Lemma 1 *In the limit as* $h \to 0$*:*

$$\underline{R}^C_M(t_j) = \frac{1}{2}\underline{I}^L_{M,N}(\underline{R}_N(t_j)) + O(h^3) \tag{24}$$

and

$$\underline{R}^L_M(t_j) = \frac{1}{2}\underline{I}^L_{M,N}(\underline{R}_N(t_j)) + \frac{h}{2}\underline{y}_{xx} + O(h^3)\ , \tag{25}$$

where the components of the vector $\underline{y}_{xx}$ *are defined by*

$$y_{xx,2i} = -\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

and

$$y_{xx,2i+1} = \frac{y_{i+2} - y_{i+1} - y_i + y_{i-1}}{2h^2}$$

PROOF. Note that the cubic spline interpolant of $\underline{y}$, $S_y(x)$ say, is fourth

order accurate, therefore at each newly created node, $x_i + \frac{h}{2}$,

$$S_y(x_i + \frac{h}{2}) = y(x_i + \frac{h}{2}) + O(h^4)$$

and

$$\frac{d^2}{dx^2} S_y(x_i + \frac{h}{2}) = y_{xx}(x_i + \frac{h}{2}) + O(h^2) \; .$$

However,

$$\frac{d^2}{dx^2} S_y(x_i + \frac{h}{2}) = \frac{-1}{h}(m_i - m_{i+1}) \; ,$$

therefore

$$m_i - m_{i+1} = -h y_{xx}(x_i + \frac{h}{2}) + O(h^3) \; . \tag{26}$$

Hence,

$$\begin{aligned} m_{i-1} - m_{i+1} &= (m_{i-1} - m_i) \; + \; (m_i - m_{i+1}) \\ &= -h y_{xx}(x_i - \frac{h}{2}) \; - \; h y_{xx}(x_i + \frac{h}{2}) + O(h^3) \\ &= -2h y_{xx}(x_i) + O(h^3) \; . \end{aligned} \tag{27}$$

Applying linear interpolation to the original residual at time t_j gives

$$\left[I^L_{M,N}(\underline{R}_N(t_j)) \right]_{2i} = R_{N,i} \; . \tag{28}$$

For notational convenience denote $R_{N,i+1/2}$ by

$$R_{N,i+1/2} = \left[I^L_{M,N}(\underline{R}_N(t_j)) \right]_{2i+1}$$

so that, using equation (17),

$$\begin{aligned} R_{N,i+1/2} = h \left[\frac{y'(x_i) + y'(x_{i+1})}{2} \right] - h \left[\frac{y_{xx}(x_i) + y_{xx}(x_{i+1})}{2} \right] - \\ h \left[\frac{f(y_i) + f(y_{i+1})}{2} \right] + O(h^3) \; . \end{aligned} \tag{29}$$

From equations (17), (19), (20), (28) and (29) it now follows that

$$R^L_{M,2i} = \frac{1}{2}R_{N,i} - \frac{h}{2}y_{xx}(x_i) + O(h^3)$$

and

$$\begin{aligned} R^L_{M,2i+1} &= \frac{1}{2}R_{N,i+1/2} + \frac{h}{4}\left[y_{xx}(x_i) - y_{xx}(x_{i+1})\right] + O(h^3) \\ &= \frac{1}{2}R_{N,i+1/2} + \frac{h}{2}y_{xx}(x_i + \frac{h}{2}) + O(h^3)\ , \end{aligned}$$

which are equivalent to (25).

Finally, in the case where cubic spline interpolation is used, substituting the above two expressions, along with (26) and (27), into (22) and (23) gives

$$R^C_{M,2i} = \frac{1}{2}R_{N,i} - \frac{h}{2}y_{xx}(x_i) + \frac{h}{2}y_{xx}(x_i) + O(h^3)$$

and

$$R^C_{M,2i+1} = \frac{1}{2}R_{N,i+1/2} + \frac{h}{2}y_{xx}(x_i + \frac{h}{2}) - \frac{h}{2}y_{xx}(x_i + \frac{h}{2}) + O(h^3)\ .$$

These are equivalent to equation (24) and so the proof is complete.

This lemma shows that if the original residual $\underline{R}_N(t_j)$ is small then it remains so after refinement and cubic spline interpolation. It also demonstrates that this is not necessarily true when linear interpolation onto the new mesh is used since there is an additional term of size $O(h)$ which occurs when a linear interpolant is differentiated twice to get zero. In subsection 4.3 below a numerical example is presented which illustrates this result very clearly. It is also clear from the proof of the lemma that this result carries across to a simple central difference discretization in which the matrices A_N and A_M are replaced by the identity matrix scaled by h: this is also demonstrated in subsection 4.3. In [8] C^1 Hermite cubic interpolation is also considered, where it is demonstrated that this has similar properties to the cubic spline interpolation considered here.

4.2.1 Second order backward differentiation formula

Suppose that a remesh has occurred at time t_j and consider an attempt to take the next step to t_{j+1} with the BDF2 method of section 2. The predicted values are given by

$$\underline{y}_M^P(t_{j+1}) = \underline{I}_{M,N}^{\chi}(\underline{y}_N(t_j)) + k_{j+1}\underline{I}_{M,N}^{\chi}(\underline{y}'_N(t_j)) + \frac{1}{2}\, k_{j+1}^2\, \underline{I}_{M,N}^{\chi}(\underline{y}''_N(t_j)) \quad (30)$$

and

$$\underline{y}_M'^P(t_{j+1}) = \underline{I}_{M,N}^{\chi}(\underline{y}'_N(t_j)) + k_{j+1}\, \underline{I}_{M,N}^{\chi}(\underline{y}''_N(t_j)) \quad (31)$$

where the superscript χ is again used to indicate either linear (L) or cubic (C) spline interpolation operators. An update vector, $\underline{\alpha}$, is then defined by solving the system of nonlinear equations:

$$A_M\ (\underline{y}_M'^P(t_{j+1}) + \underline{\alpha}) = \underline{G}_M(\underline{y}_M^P(t_{j+1}) + \frac{2k_{j+1}}{3}\underline{\alpha}, t_{j+1})\ . \quad (32)$$

The important observation is that the predicted values indirectly reflect the residual after interpolation and thus perturbations in these predicted values will potentially alter $\underline{\alpha}$. In order to quantify this effect equation (32) is first rewritten using (30) and (31), along with expressions (9), (10) and (11). This is then expanded about $(\underline{y}_M^{\chi}(t_j), t_j)$ to get

$$\begin{aligned} A_M(\underline{y}'_M(t_j) + k\underline{y}''_M(t_j) + \underline{\alpha}) & \\ &= \underline{G}_M(\underline{y}_M(t_j) + k\underline{y}'_M(t_j) + \frac{k^2}{2}\underline{y}''_M(t_j) + \frac{2k}{3}\underline{\alpha}, t_{j+1}) \\ &= \underline{G}_M + k\left[\frac{\partial \underline{G}_M}{\partial t} + \frac{\partial \underline{G}_M}{\partial \underline{y}}(\underline{y}'_M(t_j) + \frac{2}{3}\underline{\alpha})\right] + O(k^2)\ , \end{aligned}$$

where, for notational convenience, we have replaced k_{j+1} by k and dropped the superscript χ from the $\underline{y}_M$, $\underline{y}'_M$ and $\underline{y}''_M$ terms. In addition, each of the terms $\underline{G}_M$, $\frac{\partial \underline{G}_M}{\partial t}$ and $\frac{\partial \underline{G}_M}{\partial \underline{y}}$ are to be evaluated at $(\underline{y}_M(t_j), t_j)$. Hence, using the definition of $\underline{R}_M^{\chi}(t_j)$, (2) and (12), we obtain

$$\left[A_M - \frac{2}{3}k\frac{\partial \underline{G}_M}{\partial \underline{y}}\right]\underline{\alpha} =$$

$$-\underline{R}^{\chi}_M(t_j) + k\left[-A_M\ \underline{y}''_M(t_j) + \frac{\partial \underline{G}_M}{\partial t} + \frac{\partial \underline{G}_M}{\partial \underline{y}}\underline{y}'_M(t_j)\right] + O(k^2)\ . \tag{33}$$

This last expression implies that there is a contribution to $\underline{\alpha}$ from the residual $\underline{R}^{\chi}_M(t_j)$ that is independent of the time-step k. When no remeshing is done this residual is kept small through requiring that the nonlinear equations are solved with sufficient accuracy, however it may be observed from the above lemma that a poor choice of interpolant may lead to a significant increase in the size of this residual after mesh refinement. This contribution to $\underline{\alpha}$ will perturb the local error estimate and the solution time derivative by an amount $\underline{\alpha}^*$, where

$$\underline{\alpha}^* = -\left[A_M - \frac{2}{3}k\frac{\partial \underline{G}_M}{\partial \underline{y}}\right]^{-1} \underline{R}^{\chi}_M(t_j)\ ,$$

and will perturb the solution by an amount $\frac{2k}{3}\underline{\alpha}^*$. In particular if $c\|\underline{\alpha}^*\|_W \geq c\|\underline{\alpha}\|_W$ (where c is the method-dependent constant in equation (8)) then the perturbation to the local error test is greater than the local error itself. This suggests that the value of the ratio

$$\alpha_R = \|\underline{\alpha}^*\|_W / \|\underline{\alpha}\|_W \tag{34}$$

will be a useful indicator as to whether or not the local error estimate is likely to be contaminated by the interpolation error after mesh refinement has taken place. It is fortunate that the size of this ratio may be calculated quite easily since the decomposition of the Jacobian matrix, $[A_M - \frac{2}{3}k\frac{\partial \underline{G}_M}{\partial \underline{y}}]$, is stored by the BDF code. It is therefore possible to monitor the size of this ratio in a number of numerical examples to verify that it does indeed vary significantly from one interpolation method to another, and to observe the effect that this has on the performance of the ODE code.

4.2.2 Other time integration schemes

In 4.2.1 equation (33) shows that the correction, $\underline{\alpha}$, to the solution calculated by solving the nonlinear equations (32) is dominated by the residual $\underline{R}^{\chi}_M(t_j)$ as $k \to 0$. In fact, it is actually shown that, to leading order, $\underline{\alpha}$ is this residual scaled by the inverse Jacobian matrix $\left[A_M - \frac{2}{3}k\frac{\partial \underline{G}_M}{\partial \underline{y}}\right]^{-1}$. Similar results may be obtained for other implicit time integration formulae such as fully implicit Runge-Kutta formulae (see [11], for example, for an introduction to these).

It is not just implicit methods that are affected by poor interpolation between meshes however. For example, explicit Runge-Kutta formulae can also be adversely affected by the discontinuity in the ODE system that is introduced

when remeshing occurs. To demonstrate this briefly, we consider the simplest explicit Runge-Kutta method of all, the Euler scheme, and assume that the matrix A_N in equation (2) is the identity (this simplifies the algebra which follows and can easily be achieved by incorporating A_N^{-1} into the definition of $\underline{G}_N$).

Without any remeshing at time t_j the Euler formula is

$$\begin{aligned} \underline{K}_N &= \underline{G}_N(\underline{y}_N(t_j), t_j) \\ \underline{y}_N(t_{j+1}) &= \underline{y}_N(t_j) + k_{j+1}\underline{K}_N \ . \end{aligned}$$

(Note that the vector $\underline{K}_N$ is not related to the stiffness matrix K_N, used in subsection 4.1.) Alternatively, if remeshing were to occur at time t_j, the next time-step would be

$$\begin{aligned} \underline{K}_M &= \underline{G}_M(\underline{y}_M^\chi(t_j), t_j) \\ \underline{y}_M(t_{j+1}) &= \underline{y}_M^\chi(t_j) + k_{j+1}\underline{K}_M \ . \end{aligned}$$

In order to contrast these two formulae we may compare the respective update vectors $\underline{K}_N$ and $\underline{K}_M$. This is achieved by interpolating $\underline{K}_N$ onto the new mesh:

$$\begin{aligned} \underline{K}_M - \underline{I}_{M,N}^\chi(\underline{K}_N) &= \underline{G}_M(\underline{y}_M^\chi(t_j), t_j) - \underline{I}_{M,N}^\chi(\underline{G}_N(\underline{y}_N(t_j), t_j)) \\ &= \underline{G}_M(\underline{y}_M^\chi(t_j), t_j) - \underline{I}_{M,N}^\chi(\underline{y}'_N(t_j)) \\ &= \underline{G}_M(\underline{y}_M^\chi(t_j), t_j) - \underline{y}'^\chi_M(t_j) \\ &= -\underline{R}_M^\chi(t_j) \ . \end{aligned}$$

As with the BDF2 formula we again see that the difference between the updates with and without remeshing is dominated by the residual after interpolation, $\underline{R}_M^\chi(t_j)$. Unlike the implicit case however the residual is not smoothed by the inverse Jacobian matrix. Similar results may be obtained for other, more complex, Runge-Kutta schemes. This demonstrates that the need to use appropriate spatial interpolation schemes arises almost regardless of the type of time integration formula used.

4.3 Numerical experiments

This subsection illustrates the results in 4.1 and 4.2 above by way of some simple numerical examples. In 4.3.1 the results of lemma 4.1 are verified by using an adaptive finite element solver on equation (13) of subsection 3.2. Then, in 4.3.2, it is demonstrated that the cubic spline interpolant is indeed

also superior when using a different spatial discretization to that explicitly analyzed above. Again equation (13) is used, as well as an additional convection-diffusion example.

4.3.1 A finite element example

In this example we again use the adaptive finite element solver described in subsection 3.2 to solve equation (13). Unlike in 3.2 however we now start with a uniform mesh (of 128 elements in this case) and refine this globally at a fixed time, t=0.2 (see subsection 4.1). The consequences of doing this are summarized in Table 1 below. In the first column of this table we see the time-step size, k_j, immediately before refinement and also the values of each of the terms in a single component of the residual, $\underline{R}_N(t_j)$ (see (15)), again immediately before refinement. In the next two columns are the terms in the corresponding components of the new residuals immediately after refinement with linear and cubic spline interpolation (see (18) and (21)). Also given are the values of k_{j+1}, the next time-steps successfully used by the ODE solver when flying restarts are attempted.

Table 1
The effects of the linear (L) and cubic (C) spline interpolants after uniform mesh refinement.

Before refinement		After refinement		
			χ = L	χ = C
N	127	M	255	255
k_j	3.68E-2	k_{j+1}	8.85E-4	2.84E-2
$[A_N \underline{y}'_N(t_j)]_{64}$	9.80E-4	$[A_M \underline{y}'^{\chi}_M(t_j)]_{128}$	4.90E-4	4.90E-4
$[K_N \underline{y}_N(t_j)]_{64}$	1.08E-3	$[K_M \underline{y}^{\chi}_M(t_j)]_{128}$	1.08E-3	5.39E-4
$[\underline{f}_N(\underline{y}_N, t_j)]_{64}$	-9.94E-5	$[\underline{f}_M(\underline{y}^{\chi}_M, t_j)]_{128}$	-4.97E-5	-4.97E-5
$[R_N(t_j)]_{64}$	-2.10E-13	$[R^{\chi}_M(t_j)]_{128}$	-5.40E-4	-4.06E-8

It is immediately clear from inspection of this table that the numerical computations are consistent with the results of lemma 4.1. The residual terms are correctly halved when cubic interpolation is used whereas with linear interpolation the incorrect diffusive term pollutes the residual. As predicted by the analysis of the BDF2 time integrator in subsection 4.2, this then leads to a significant reduction in time-step from k_j = 3.6791E-2 to k_{j+1} = 8.8546E-4. It should also be noted, although it is not shown in Table 1, that there is a reduction in order, from 2 to 1, when linear interpolation is used after refinement at $t = 0.2$. Neither this order reduction nor this significant drop in the time-step occur when cubic spline interpolation is used before a flying restart.

Two final points to note about this example are that even though the attempt at a flying restart after linear interpolation appears to have failed in this case, the new time-step (k_{j+1} = 8.8546E-4) is still no worse than that obtained when using a full restart on this problem. Also, it should be emphasized that there is a dependence upon the mesh size h and the time-step k in the results of subsections 4.1 and 4.2. This means that when, for example, the above numerical experiment is repeated on a much coarser spatial mesh ($N = 16$ say), even cubic spline interpolation fails to allow a successful flying restart. Conversely, when an extremely fine spatial mesh is used (e.g. $N = 1024$) the flying restart with linear interpolation is significantly more effective than using a full restart.

4.3.2 An alternative spatial discretization

We complete this section with some further numerical examples to illustrate the practical advantages of cubic spline over linear interpolation. This is done by monitoring the behaviour of the ratio α_R (defined by (34) above) for two test problems using the semi-discretization of Skeel and Berzins [22] and the remeshing method of [4]. The first test problem is again defined by equation (13) but this time with a spatial interval of $(0, 120)$. The second test problem is a linear convection-diffusion equation defined by

$$\frac{\partial u}{\partial t} = 0.1\frac{\partial^2 u}{\partial x^2} - \frac{\partial u}{\partial x} \qquad \text{where } u(x,t) : (0,1) \times (0,1) \longrightarrow \Re,$$

with initial and Dirichlet boundary conditions consistent with the analytic solution

$$u(x,t) = \frac{e^{50(x-t)^2/s}}{\sqrt{s}} \qquad (\text{where } s = 1 + 20t).$$

Both problems were solved with meshes of between 41 and 81 points with remeshing on the basis of equidistribution of the second space derivative, as in [4]. Remeshing occurred every two time-steps. The time error test tolerances were 0.1E-4 (both absolute and relative). The integration statistics and the average value of α_R over the number of remeshes are given in Table 2 below. This table shows that when linear interpolation is used there is a substantial increase in the overall computing cost, a decrease in the final accuracy and that the average value of the ratio α_R is greater than 1.0 (unlike the cubic case in which it is less than 1.0). The value No. $\alpha_R > 1$ shows how many of the values of α_R were greater than 1.0 after remeshing. Hence for problems 1 and 2 with linear interpolation this indicator exceeds 1.0 for 84 and 73 percent of the time respectively after remeshing. This rarely happens with cubic interpolation. Hence the α_R indicator appears to provide useful information about potential

difficulties due to discrete time remeshing, and once again these difficulties appear to arise more often when linear rather than cubic spline interpolation is used between meshes.

Table 2
Integration statistics for the two 1-d test problems.

	Problem 1		Problem 2	
	linear	cubic	linear	cubic
Steps Taken	106	71	195	147
No. of remesh	51	33	97	72
Jacob. evals	56	38	100	76
Func. calls	484	317	847	631
Aver. α_R	1.08	0.7	1.04	0.84
No. $\alpha_R > 1$	43	0	71	5
Final Error .	2.4E-3	3.6E-4	2.2E-4	8.2E-5

5 Conclusions

This paper has demonstrated that the use of a poor interpolant in discrete time remeshing is a potential source of inaccuracy and inefficiency. This has been shown on a practical Navier-Stokes example and on a simple test problem. Analyses of this test problem and of a more general case in the limits as the mesh size and time-step size tend to zero have highlighted the source of the difficulty, and also suggest a way of monitoring it. The effectiveness of this monitor has been confirmed for typical choices of mesh and time-step size by further experimental work. Nevertheless, further analysis would be required in order to guarantee an optimal choice for the interpolant needed for a particular problem with given values of h and k. In conclusion therefore, it is clear that significant care must be taken in choosing a suitable interpolant when using discrete remeshing in one, two or three space dimensions.

Acknowledgement

PJC would like to thank EPSRC and British Aerospace for funding in the form of a CASE studentship.

References

[1] I. Babuska, J. Chandra and J. E. Flaherty, Adaptive computational methods for PDEs, *Proceedings of Workshop at University of Maryland* (SIAM Publications, Philadelphia, 1983).

[2] R. E. Bank, *Private Communication* (1997).

[3] R. E. Bank, PLTMG: a software package for solving elliptic partial differential equations, Users Guide 7.0, *Frontiers in Applied Mathematics 15* (SIAM Publications, Philadelphia, 1994).

[4] M. Berzins, P. M. Dew and R. M. Furzeland, Developing software for time-dependent problems using the method of lines and differential-algebraic integrators, *Appl. Num. Math.* **5** (1989) 373–397.

[5] M. Berzins and R. M. Furzeland, A user's manual for SPRINT - a versatile software package for solving systems of algebraic ordinary and partial differential equations. Part 2: partial differential equations, *Report TNER.86.050* (Shell Research Ltd, Thornton Research Centre, Chester, UK, 1986).

[6] M. Berzins, A C^1 interpolant for codes based on backward differentiation formulae, *Appl. Num. Math.* **2** (1986) 109–118.

[7] M. O. Bristeau, R. Glowinski, L. Dutto, J. Periaux and G. Roge, Compressible viscous flow calculations using compatible finite element approximations, *Int. J. Num. Meth. Fluids* **11** (1990) 719–749.

[8] P. J. Capon, Adaptive stable finite element methods for the compressible Navier-Stokes equations, *Ph.D. Thesis* (University of Leeds, 1995).

[9] R. E. Ewing, Adaptive mesh refinements in large scale fluid-flow simulation, in: I. Babuska, O. C. Zienkiewicz, J. Gago and E. R. de A. Oliviera, eds., *Accuracy Estimates and Adaptive Refinements in Finite Element Computations* (John Wiley and Sons, 1986) Chapter 16.

[10] J. E. Flaherty, P. J. Paslow, M. S. Shephard and J. D. Vasilakis, *Adaptive methods for partial differential equations* (SIAM Publications, Philadelphia, 1989).

[11] E. Hairer and G. Wanner, *Solving ordinary differential equations II: stiff and differential-algebraic problems* (Springer Series in Computational Mathematics 14, Springer-Verlag, 1991).

[12] G. Hauke and T. J. R. Hughes, A unified approach to compressible and incompressible flows, *Comp. Meth. in Appl. Mech. and Eng.* **113** (1994) 389–395.

[13] J. M. Hyman and M. J. Naughton, Static rezone methods for tensor product grids, *Lectures in Applied Mathematics* **22** (American Mathematical Society, 1985) 321–343.

[14] P. K. Jimack, A new approach to finite element error control for time-dependent problems, in: M. J. Baines and K. W. Morton, eds., *Numerical Methods for Fluid Dynamics 4* (Oxford University Press, 1993) 567–573.

[15] P. K. Jimack and A. J. Wathen, Temporal derivatives in the finite element method on continuously deforming grids, *SIAM J. Numer. Anal.* **28** (1991) 990–1003.

[16] C. Johnson, *Numerical solution of partial differential equations by the finite element method* (Cambridge University Press, 1987).

[17] P. Lancaster and K. Šalkauskas, *Curve and surface fitting* (Academic Press, 1986).

[18] R. Lohner, An adaptive finite element scheme for transient problems in CFD, *Comp. Meth. in Appl. Mech. and Eng.* **50** (1987) 323–338.

[19] B. J. Lucier, A moving mesh numerical method for hyperbolic conservation laws, *Math. Comp.* **46** (1981) 59–69.

[20] K. Miller and R. Miller, Moving finite elements: part I, *SIAM J. Numer. Anal.* **18** (1981) 1019–1032.

[21] R. D. Skeel, Construction of variable stepsize multistep formulas, *Math. Comp.* **v47, 176** (1986) 503–510.

[22] R. D. Skeel and M. Berzins, A method for the spatial discretization of parabolic equations in one space variable, *SIAM J. Sci. Statist. Comput.* **11** (1990) 1–32.

[23] F. Shakib, T. J. R. Hughes, and Z. Johan, A new finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier-Stokes equations, *Comp. Meth. in Appl. Mech. and Eng.* **89** (1991) 141–219.

[24] L. F. Shampine, *Numerical solution of ordinary differential equations* (Chapman and Hall, 1994).

[25] A. Soulaimani and M. Fortin, Finite element solution of compressible viscous flows using conservative variables, *Comp. Meth. in Appl. Mech. and Eng.* **118** (1994) 319–350.