Advances in Computer Methods for Partial Differential Equations VII
R. Vichnevetsky, D. Knight and G. Richter (Editors)
©1992 IMACS. All rights reserved.

SPATIAL AND TEMPORAL ERROR CONTROL IN THE ADAPTIVE SOLUTION OF SYSTEMS OF CONSERVATION LAWS.

M. Berzins , J. Lawson and J.Ware.
School of Computer Studies,
The University of Leeds, Leeds LS2 9JT.

1. Introduction.

The area of fluid mechanics has long been recognised as one for the application of automated analysis capabilities. The advent of reliable and robust mesh generators, physically realistic spatial discretization methods, sophisticated time integration software, and error balancing techniques for time-dependent p.d.e. problems has made it possible to write reliable adaptive finite element programs for time-dependent fluid flow calculations. The programs are intended to be *reliable* in that they make use of spatial and temporal error estimates to meet automatically the users accuracy requirements. This paper follows the framework laid down by Berzins et. al.[2]. in describing the basic components of a prototype automated solver for the solution of the time-dependent p.d.e.s. In such software the accuracy of the numerical solution is influenced by the spatial discretisation method used, the spatial mesh and the method of time integration.

The spatial discretisation method and positioning of the spatial mesh points should ensure that the spatial error is controlled to meet the users requirements. Two key parts of the solver are the spatial mesh generator and the adaptivity software used to refine and coarsen the mesh as part of the procedure to control the spatial discretization error. The spatial discretization procedure used on the mesh should provide numerical solutions that are free of spurious oscillations. Such discretizations are considered by very many authors e.g. by Koren [7] for the steady Euler and Navier-Stokes equations using quadrilateral meshes. The general approach of Koren is extended by Ware and Berzins [11] to the use of unstructured triangular meshes in the spatial semi-discretization of the p.d.e.

This method of lines approach results in a system of time dependent o.d.e.s which can be solved by using o.d.e. software. preferably with sufficient accuracy so that the spatial error is not degraded while maintaining efficiency. This is achieved by using the error balancing approach of Berzins [4] in conjunction with the time integrator of Berzins and Furzeland [3]. This error balancing approach makes use of local estimates of the space and temporal errors to calculate the time accuracy tolerance in such a way that the spatial discretization and time integration errors are of the same order of magnitude, but so that the spatial discretization error dominates the time integration error.

2. Spatial Discretization for Compressible Navier-Stokes Equations.

The solution strategy for the Navier-Stokes equations is to split the equations into their convective and diffusive parts. This enables upwind discretization methods developed for the Euler equations to be used for the convective part of the system and the centered discretization methods to be used for the diffusive part. The class of p.d.e.s to be considered is written in cartesian co-ordinates as

$$\frac{\partial q}{\partial t} = \frac{\partial}{\partial x}(f_1(q)+f_2(q)) + \frac{\partial}{\partial y}(g_1(q)+g_2(q)) \;,\; t \;\varepsilon\; (0,t_e] \;,\; (x\,,y) \;\varepsilon\; \Omega \qquad (2.1)$$

with appropriate boundary and initial conditions. For the Navier Stokes equations the solution vector has the form $q(x,y,t) = [e\,,\rho\,,\rho u\,,\rho v]^T$. Here, ρ is the fluid density; u,v are the cartesian components of the velocity vector, e is the internal energy. The pressure p is evaluated according to an equation of state. The fluxes f_1 and g_1 represent the convective fluxes while f_2 and g_2 are the diffusive fluxes. For simplicity only one p.d.e. of the form of equation (2.1) will be considered in the rest of this paper.

The first step in the discretization process is to triangulate the region Ω using a mesh generator, see Section 4 below. With a finite volume approach equations (2.1) are integrated over a triangular element i (with vertices A,B and C) and the divergence theorem is applied and a one point quadrature rule applied along the edges of the triangle to obtain

$$Area_{ABC}\,\frac{\partial q_{ABC}}{\partial t} = -\,[\,(f_1(q_{AB}) + f_2(q_{AB}))\,\Delta y_{AB} - (g_1(q_{AB}) + g_2(q_{AB}))\,\Delta x_{AB} +$$

$$(f_1(q_{BC}) + f_2(q_{BC}))\,\Delta y_{BC} - (g_1(q_{BC}) + g_2(q_{BC}))\,\Delta x_{BC} +$$

$$(f_1(q_{CA}) + f_2(q_{CA}))\,\Delta y_{CA} - (g_1(q_{CA}) + g_2(q_{CA}))\,\Delta x_{CA}\,] \qquad (2.2)$$

where q_{AB} is the solution value midway along the edge AB , Δx_{AB} is the change in the x co-ordinate in going from A to B , q_{ABC} is the solution value associated with the centroid of the triangle ABC and the other values in the equation are similarly defined. As the solution values are only piecewise constant inside each triangle the evaluation of the convective fluxes midway along the edge involves the approximate solution of three one-dimensional Riemann problems in the direction of the normals to the edges of the triangle. This is done by using the scheme of Engquist and Osher as described by Koren [7].

The piecewise constant finite volume schemes using Engquist and Osher's flux evaluations are only first-order accurate. The extension to a second order scheme on an unstructured triangular mesh is given by Ware and Berzins [11]. This spatial discretization scheme results in a system of differential equations, each of which is of the form of equation (2.2) . This system of equations can be written as the i.v.p.

$$\dot{\underline{Q}} = \underline{F}_N\ (t\,,\ \underline{Q}(t))\,, \qquad (2.3)$$

where the N dimensional vector, $\underline{Q}(t)$, is defined by

$$\underline{Q}(t) = \left[Q(x_1\,,y_1\,,t)\ ,\ Q(x_2\,,y_2\,,t)\,,\ \cdots\ ,Q(x_N\,,y_N\,,t)\right]^T\,,$$

where $(\,x_i\,,y_i\,)$ is the centroid of the ith triangle, $Q(x_i\,,y_i\,,t)$ is a numerical approximation to $q(x_i\,,y_i\,,t)$

3. Time Integration.

In practice the system of equations (2.3) is integrated in time to compute the approximation, $\underline{V}(t)$, to the true solution, $q(t)$, of the p.d.e. The global error in the numerical solution can be expressed as the sum of the spatial discretization error, $\underline{es}(t) = q(t) - \underline{Q}(t)$, and the global time error, $\underline{ge}(t) = \underline{Q}(t) - \underline{V}(t)$. That is,

$$\underline{E}(t) = \underline{q}(t) - \underline{V}(t) = (q(t) - \underline{Q}(t)) + (\underline{Q}(t) - \underline{V}(t)) \qquad (3.1)$$
$$= \underline{es}(t) + \underline{ge}(t).$$

The new adaptive Theta algorithm, [3], used here for time integration can handle both stiff (diffusion or source term dominated) and non-stiff (convection dominated) problems by automatically selecting the best value of theta and the most efficient iteration method (Newton or functional iteration) with respect to step size and cost per step. The criteria for selecting theta and for switching are established by optimising the permissible step size. For non-stiff o.d.e.s the method uses functional iteration, [3]. The method assumes the approximations $\underline{V}(t_n)$ to $\underline{U}(t(t_n)$, and $\underline{V}(t_n)$ to $\underline{U}_n)$, then the numerical solution at t_{n+1} , where $t_{n+1} = t_n + k$ and k is the time step size, as denoted by $\underline{V}(t_{n+1})$ is defined by

$$\underline{V}(t_{n+1}) = \underline{V}(t_n) + (1 - \theta)k\,\dot{\underline{V}}(t_n) + \theta\,k\,\underline{F}_N(t_{n+1}\,,\underline{V}(t_{n+1}))\ ,\ 0 < \theta < 1\ . \qquad (3.2)$$

The new integrator is part of the SPRINT software package, an important feature of which is the capability to handle p.d.e. space remeshing schemes. After each time step taken by the integrator a routine, generic name MONITR, is called which is designed for tasks such as o.d.e. global error estimation and remeshing at discrete times.

In most time dependent fluid-flow p.d.e. codes either a CFL stability control is employed or a standard o.d.e. solver is used which employs local time error per step, (LEPS), control with respect to a user supplied accuracy tolerance, TOL, i.e.

$$||\underline{le}_{n+1}(t_{n+1}\ ,\mathrm{TOL})|| \le \mathrm{TOL}. \qquad (3.3)$$

or the local time error per unit step (LEPUS),

$$||\underline{le}_{n+1}(t_{n+1}\ ,\mathrm{TOL})|| < k\ \mathrm{TOL}. \qquad (3.4)$$

When controlling the LEPS it is difficult to establish a relationship between the accuracy tolerance, TOL, and the global time error and to relate TOL to the spatial error. On the other hand, if the LEPUS is controlled then it can be shown, see [8], that the time global error is proportional to the tolerance that is

$$\underline{ge}(t) = \underline{v}(t)\,\mathrm{TOL} + o(\mathrm{TOL}), \tag{3.5}$$

where $\underline{v}(t)$ is independent of TOL and $\underline{v}(t)$ and $\underline{v}'(t)$ are bounded on $[0, t_e]$. Although LEPUS control is generally thought to be inefficient for standard o.d.e.s, there is a fundamentally different situation in the time integration of p.d.e.s in that the time error control strategy must take account of the spatial discretization error already present.

4. Balancing the Space and Time Errors.

In order that the solution is computed efficiently, the time integration error should not dominate the error due to the spatial discretization of the p.d.e. but nor should the o.d.e.s be integrated with a much higher degree of accuracy than that already attained in space. In practice, the spatial discretization error must dominate the temporal error so that the estimate of the spatial discretization error is unpolluted by temporal error. Lawson and Berzins [8] have developed one such strategy which controls the local time error to be a fraction of the growth in the spatial discretization errors over the interval $[t_n, t_{n+1}]$, that is,

$$|| \underline{le}_{n+1}(t_{n+1}, \mathrm{TOL}) || < \varepsilon \, || \underline{es}(t_{n+1}) - \underline{es}(t_n) || , \tag{4.1}$$

and have shown that, for a suitable value of ε, this yields a time integration error which is dominated by the spatial discretization error. That this is a form of LEPUS control is seen by expanding using Taylor's series about t_{n+1} and thus extracting a factor of the stepsize k.

The Lawson and Berzins approach is designed for parabolic problems and is not appropriate for convection dominated p.d.e.s, see [4]. For this reason the error control strategy used here measures the growth in the spatial error over the timestep in a similar way to that used by Moore and Flaherty, [10], but modified for convection-dominated p.d.e.s. Define the local in time spatial error, $\hat{\underline{es}}(t)$, as the spatial error at time t given the assumption that $\underline{es}(t_n) = 0$. A local in time error balancing approach is then given by

$$|| \underline{le}_{n+1}(t_{n+1}, \mathrm{TOL}) || < \varepsilon \, k_{n+1} \, || \hat{\underline{es}}(t_{n+1}) || . \tag{4.2}$$

equations, see Berzins [4]. The error control strategy used here differs from that of [10] in that no nodal superconvergence results are assumed, the computed solution is treated as the high-order solution and a low-order solution is estimated in a computationally inexpensive way. The difference between the two solutions will only be an estimate of the error in the low-order solution and so local extrapolation in space is effectively being used when the high-order solution is used to move forward in time. The low order solution is obtained by using only the piecewise constant finite volume scheme thus defining a modified o.d.e. system which will be denoted by

$$\dot{\underline{v}}_{n+1}(t) = \hat{\underline{F}}_N(t, \underline{v}_{n+1}(t)) \quad , \quad \underline{v}_{n+1}(t_n) = \underline{V}(t_n) \quad , \quad \dot{\underline{v}}_{n+1}(t_n) = \dot{\underline{V}}(t_n). \tag{4.3}$$

The local in time space error is then given by

$$\hat{\underline{es}}(t_{n+1}) = \underline{V}(t_{n+1}) - \underline{v}_{n+1}(t_{n+1}) . \tag{4.4}$$

The local in time space error is computed by applying the θ method of Section 3 to equation (4.3) to get

$$\underline{v}_{n+1}(t) = \underline{V}(t_n) + \theta\, k\, \hat{\underline{F}}_N(t, \underline{v}_{n+1}(t_{n+1})) + (1-\theta)\, k\, \dot{\underline{V}}(t_n).$$

Using this equation with equations (3.2) and (4.4) gives

$$\hat{\underline{es}}(t_{n+1}) = \theta k \left[\underline{F}_N(t_{n+1}, \underline{V}(t_{n+1})) - \hat{\underline{F}}_N(t_{n+1}, \underline{v}_{n+1}(t_{n+1})) \right] \tag{4.5}$$

where $k = t_{n+1} - t_n$. From this equation the LEPUS tolerance used in the code on the step to t_{n+1} is then given by

$$TOL = \varepsilon\, \theta\, || \underline{F}_N(t_{n+1}, \underline{V}(t_{n+1})) - \hat{\underline{F}}_N(t_{n+1}, \underline{v}_{n+1}(t_{n+1})) || \tag{4.6}$$

5. Mesh Generation.

In two space dimensions it is important to be able to solve problems defined on irregular regions. This requires the use of a mesh generator for unstructured triangular spatial meshes. At the same time it is necessary to ensure that the spatial mesh is fine or coarse enough so that the solution satisfies the users' accuracy and efficiency requirements. This requires the use of adaptive mesh techniques.

The mesh generator used was written at Shell Research and is based on the ideas of George et al. [6]. An equilateral triangle which totally surrounds the whole of the user's geometry is constructed. Each of the user supplied vertices is added to the mesh, one at a time. Typically, there will be two cases - the new vertex lies within an existing triangle, in which case edges from the new vertex to each of the triangle's vertices are drawn. On the other hand, the vertex may lie on or near the common edge of two triangles. In this case, edges from the new vertex to each of the triangles' vertices are drawn and so the two triangles are replaced by four new ones.

At the end of this process, a triangulation may not respect the user's geometry and so is modified using the algorithm of [6] so that the boundary edges are present in the mesh. At this point, any triangles lying outside the geometry (that is, connected to the vertices of the large equilateral triangle) are discarded.

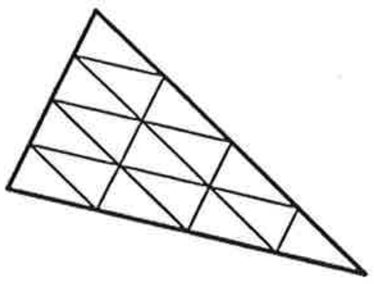
The triangulation may still contain very long thin triangles. These are removed by diagonal flipping in which the diagonal of the quadrilateral formed by two adjacent triangles may be swapped with the other diagonal. This flipping will only occur if it leads to the creation of two better shaped triangles. At the end of the flipping process there may still be some long thin triangles, in which case, vertices may be added at strategic points in the mesh. The meshes produced by this mesh generator are generally coarse, unless the user has provided extra vertices or triangles, and so must be selectively refined to be used with p.d.e. solvers.

6. Adaptivity and Iterated Function Systems.

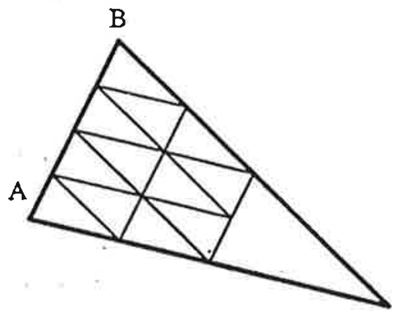
The aim of the adaptivity routines is to change the mesh during the computation by using smaller elements where the solution changes rapidly and leaving larger elements where the solution is smooth. The potential of adaptive methods is demonstrated by Lohner [9] who shows that there is a factor of between 4 and 11 difference between the number of points required in a uniform mesh compared to the number required in the final mesh obtained through adaptivity.

The adaptive algorithm used here is based on h−refinement in which a new mesh is created by subdividing or deleting elements in the existing mesh. The meshes are usually relatively fine and are quasi-uniform. When solving time dependent problems, it is important to be able to coarsen the mesh as well as refine it, particularly when moving features in the solution are being tracked. The approach used here is a combination of the the regular subdivision approach of Lohner [9] and Bank [1] by using the iterated function systems approach of Bova and Carey [5].

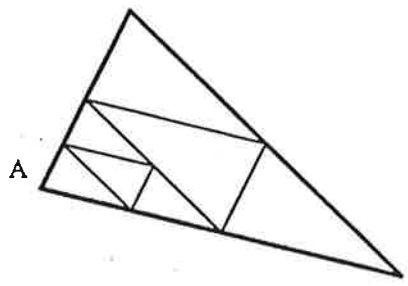
Iterated Function Systems (IFS), Bova and Carey [5], are mappings that can be applied recursively to any triangle to generate points in a conceptually elegant fashion. These points may be used to refine an existing mesh. Depending on the mapping used, points can be generated to lie over the whole of a triangle, towards an edge or towards a vertex. The following diagrams show triangles that have been refined using a TRIANGLE, EDGE and VERTEX attractor, respectively. The attractors specify the mapping to be applied, for example, an EDGE attractor indicates that a triangle is to be refined towards an edge.



Refined using TRIANGLE attractor.
Refinement level 2



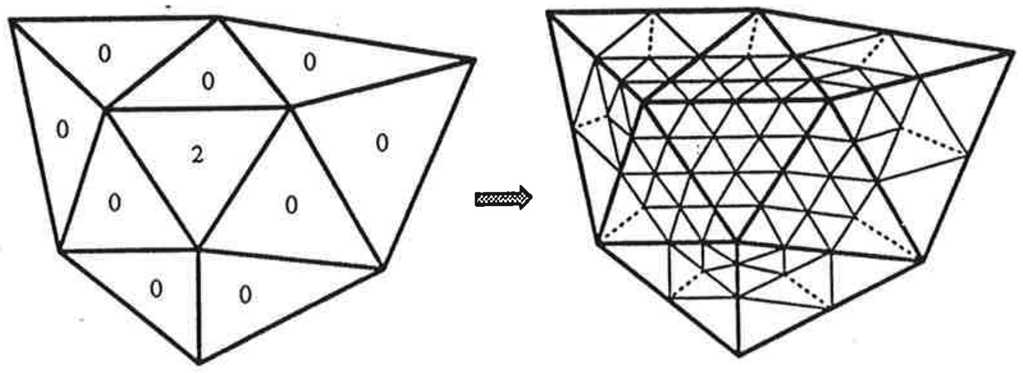
Refined using EDGE attractor towards AB.
Refinement level 2



Refined using VERTEX attractor towards A.
Refinement level 2

Graded meshes can be obtained in the following way. Consider a triangle to be refined with a TRIANGLE attractor. Then all its neighbouring triangles which share an edge with it, are refined towards that shared edge. Similarly, all triangles with a vertex in common with the original triangle are refined towards that vertex. Finally

Bank's green rule is applied to ensure the mesh is conforming, [1]. This is illustrated in the diagram below in which a level 2 refinement is applied to the central triangle



where dashed lines represent the bisecting edges of green triangles. The concept of IFS attractors thus provides an elegant way to implement multiple levels of regular refinement.

The input to the refinement routines consists of a list of triangles to be refined using a TRIANGLE attractor and a list of triangles to be deleted. These lists are generated according to an adaptive strategy based on computed error estimates for each triangle. De-refinement is a reversal of the refinement process, that is, the four children created through regular subdivision can be deleted, leaving the parent. Only one level of de-refinement is allowed at any one remeshing time, in addition, all four children must be marked for deletion. De-refinement will not be allowed if a triangle in the initial mesh, produced by the mesh generator, is specified. The refinement/de-refinement strategy is described below. These rules are applied to each triangle marked for deletion. Appropriate triangles are then deleted. For each triangle marked for refinement with the TRIANGLE attractor adjacent triangles are marked for refinement with EDGE or VERTEX attractor and same refinement level. The mesh is refined by using the attractors. A conforming triangulation is obtained by applying Bank's green rule where necessary. If the triangles in the initial mesh are of good quality then most of the triangles in subsequent meshes (created as a result of regular subdivision) will also be of good quality. The exceptions will be those triangles created as a result of application of the green rule, when triangles are bisected. These triangles are required to complete the triangulation, but will be removed before any further modification of the mesh takes place. This prevents deterioration in the quality of the triangles.

The refinement technique used here allows both uniform refinement of the mesh and development of graded meshes. This technique complements the mesh generator which aims to produce coarse meshes in which the triangles are as close to equilateral as the user requests.

7. Numerical Experiments.

A number of experiments with both one and two space-dimensional problems using the new error control strategy are described by Berzins [4] The experiment described illustrates the approach for a two dimensional Burgers equation problem and provides a comparison were made with the standard local error control strategy. The test problems is:

$$\frac{\partial u}{\partial t} + v(x,t)\frac{\partial u}{\partial x} + v(y,t)\frac{\partial u}{\partial y} - \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0 \ , \ \nu = 0.0001 \ ,$$

$$\text{where} \quad (x\,,y,t) \ \varepsilon \ (0,1)\times(0,1)\times(0\,,1]\,.$$

The analytic solution is given by $u(x,y,t) = v(x,t)\,v(y,t)$ where $v(x,t)$ is defined by

$$v(x\,,t) = \frac{0.1A + 0.5B + C}{A + B + C}$$

where $A = e^{(-0.05(x - 0.5 + 4.95t)/\nu)}$, $B = e^{-0.25(x - 0.5 + 0.75t)/\nu)}$ and $C = e^{(-0.5(x - 0.375)/\nu)}$.

The problem was solved using fixed evenly spaced space meshes of 9x9 , 27x27 and 81x81 points. The meshes were constructed so that the mesh points were the centres of square cells and so that the meshes were nested. The Theta method described in Section 3 was used with the functional iteration option switched on . Two different error control strategies were used within the time integration routines :-

(i) The LEPUS strategy given by equations (4.2) and (4.5) with $\varepsilon = 0.5$.

(ii) The standard absolute LEPS strategy given by equation (3.3).

Table 1 shows the results obtained with these two strategies with the best accuracy tolerance in terms of efficiency, as well as with smaller and larger accuracy tolerances. The *CPU time* quoted, however, does not reflect the experimentation needed to actually find the best tolerance. From these results it can be seen that by using smaller tolerances the same accuracy is achieved but at a greater cost, while using larger tolerances may increase the efficiency, but larger global errors are obtained, proving that the spatial error is no longer dominating. *NPTS* is the number of points used in the spatial mesh. *TOL* is the LEPS tolerance used in the o.d.e. integration routine except that New 0.5 refers to the new strategy, with $\varepsilon = 0.5$. *MAX L1 ERR* is the maximum grid error found at the specified output times. *CPU* is the amount of CPU time used, measured in seconds. *NSTEPS* is the number of time steps used in the integration of the o.d.e.'s.

Table 1 Results for Burgers Equation.

NPTS	TOL	Max. L1 err. at Time				NSTEPS	NFCN	CPU
		T=0.11	T= 0.44	T= 0.77	T= 1.00			
9x9	0.5D-2	0.27D-1	0.15D-1	0.30D-1	0.49D-1	38	91	0.95
	0.1D-2	0.25D-1	0.14D-1	0.34D-1	0.52D-1	61	160	1.60
	0.5D-3	0.26D-1	0.15D-1	0.32D-1	0.52D-1	88	234	2.36
	New 0.5	0.25D-1	0.14D-1	0.31D-1	0.51D-1	157	433	4.36
27x27	0.5D-2	0.11D-1	0.95D-2	0.22D-1	0.29D-1	55	126	10.2
	0.3D-2	0.11D-1	0.92D-2	0.22D-1	0.26D-1	71	183	14.0
	0.5D-3	0.10D-1	0.87D-2	0.22D-1	0.27D-1	144	407	30.6
	New 0.5	0.10D-1	0.87D-2	0.22D-1	0.27D-1	298	797	62.0
81x81	0.1D-3	0.37D-2	0.82D-2	0.49D-2	0.70D-2	493	1297	1060
	0.5D-4	0.37D-2	0.82D-2	0.49D-2	0.70D-2	645	1703	1460
	0.2D-4	0.36D-2	0.81D-2	0.48D-2	0.69D-2	788	2022	1790
	New 0.5	0.36D-2	0.81D-2	0.47D-2	0.68D-2	659	1717	1350

8. Towards an Automatic Algorithm.

The prototype automatic algorithm consists of the error control strategies described in the previous Sections. The strategies for deciding when to remesh are essentially those of Lawson and Berzins [8]. The input required from the user consists only of the problem specification, an initial spatial mesh from the mesh generator. and an error tolerance for the spatial discretization error, *EPS*. At each time step the estimate of $|| \underline{\hat{es}}(t) ||$ is calculated, and if it is greater than some fraction (say between 0.25 and 0.9) depending on the estimator of *EPS*, then a new mesh is constructed that ensures that the subsequent error is less than *EPSDN* where *EPSDN* is a fraction of EPS. The underlying assumption in this adaptive process is that the introduction of extra mesh points will cause the error to decrease. Should this not be the case it will be necessary to backtrack to an earlier time at which the solution and error estimates have been saved.

Once a new mesh has been found, the computed solution and the time history array used by the time integrator are interpolated, using a conservative interpolation scheme of Ramshaw, see [2], onto this mesh and the time integration is restarted. A " flying restart " , which uses the same stepsize and order used immediately before remeshing, is performed. This is often faster than performing a full restart. Once the time integration has been restarted, the time integration proceeds until the next point where remeshing is required is reached or the end of the computation, whichever is soonest.