VP

TOWARDS AN AUTOMATED FINITE ELEMENT SOLVER FOR TIME-DEPENDENT FLUID-FLOW PROBLEMS.

M. Berzins [*], P.L. Baehmann [**], J.E.Flaherty [**] and J. Lawson [*]

[*] *School of Computer Studies, University of Leeds, Leeds LS6 2JT, U.K.*

[**] *Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY 12180, U.S.A.*

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 [1], physically realistic spatial discretization methods [8], sophisticated time integration software [2], and error balancing techniques for time-dependent p.d.e. problems [9] 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. One example of such a program for p.d.e.s in one space variable is that of Lawson and Berzins [10] which is based on the adaptive mesh method of Bieterman and Babuska [5]. This paper describes the basic components of a prototype automated solver for the solution of the time-dependent compressible Navier Stokes equations in two space variables.

One key part of the solver is the spatial mesh generator. This needs to be able to cope with complicated geometries and to be able to locally refine and coarsen the spatial mesh as part of a procedure to control the spatial discretization error. Such a mesh generator is the Finite Quadtree mesh generator developed by Baehmann et al. [1],[2],[3].

The spatial discretization procedure used on the mesh should provide numerical solutions that are free of spurious oscillations. Such discretizations are considered by many authors e.g. by Ludwig et al.[11] for the Euler equations and by Koren [8] for the steady Navier-Stokes equations using quadrilateral meshes. The general approach of Koren is extended by considering the time-dependent case and by making 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.

Although the mesh generator provides spatial error control facilities it is still necessary to integrate in time with sufficient accuracy so that the spatial error is not degraded while maintaining efficiency. This is achieved by using a modified form of the error balancing approach of Lawson, Berzins and Dew [9] in conjunction with the time integrators of the SPRINT software [4]. Key components of this error balancing approach are the estimate of the space truncation error (which is obtained by using h-extrapolation) and the space and time error estimates used in error control.

THE MATHEMATICS OF FINITE ELEMENTS AND APPLICATIONS VII

ISBN 0-12-747257-6

The novel feature of the algorithm is that the accuracy tolerance used in the integration of the o.d.e.s is calculated automatically. It is calculated 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.

The paper is structured in the following way. Section 2 describes the problem class, spatial discretization method and time integration software. Section 3 indicates how the global error may be decomposed and estimated. This allows the error control strategy for the time integration to be summarised. Section 4 contains a summary of how the mesh generator works and how it is used to control the spatial error, while Section 5 explains how this algorithm is modified to work with the error balancing approach in a prototype fully automatic mathematical software package for the numerical solution of time dependent p.d.e.s.

2. SPATIAL AND TEMPORAL DISCRETIZATION FOR COMPRESSIBLE NAVIER-STOKES EQUATIONS.

The solution strategy for the Navier-Stokes equations is to follow Koren [8] by splitting 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

$$\frac{\partial q}{\partial t} = \frac{\partial}{\partial x}\left(f_1(q) + f_2(q)\right) + \frac{\partial}{\partial y}\left(g_1(q) + g_2(q)\right) \ , t \in (0,t_e] \ , \ (x,y) \in \Omega \quad (2.1)$$

with appropriate boundary and initial conditions. 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 [8].

The first step in the discretization process is to triangulate the region Ω using the quadtree 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 (see Chakravarthy and Osher [6]) to obtain

$$\begin{aligned} Area_{ABC}\frac{\partial q_{ABC}}{\partial t} = - [& \left(f_1(q_{AB}) + f_2(q_{AB})\right)\Delta x_{AB} - \left(g_1(q_{AB}) + g_2(q_{AB})\right)\Delta y_{AB} + \\ & \left(f_1(q_{BC}) + f_2(q_{BC})\right)\Delta x_{BC} - \left(g_1(q_{BC}) + g_2(q_{BC})\right)\Delta y_{BC} + \\ & \left(f_1(q_{CA}) + f_2(q_{CA})\right)\Delta x_{CA} - \left(g_1(q_{CA}) + g_2(q_{CA})\right)\Delta y_{CA}] \quad (2.2) \end{aligned}$$

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 upwind scheme of Engquist and Osher as described by Koren [8]. This appears to be an efficient, accurate, and robust means of solving compressible flow problems (e.g. [11]) on finite quadtree meshes.

The piecewise constant finite volume schemes using Engquist and Osher's flux evaluations are only first-order accurate. In addition it is difficult to estimate the derivatives present

in the diffusive fluxes. Consider the edge AB. A piecewise linear interpolant can be built up on say the side of the edge interior to the triangle ABC by using solution values from the triangle ABC and its neighbours on the sides BC and CA. This can be extended to a bilinear interpolant by including the centroid value from the other triangle having AB as an edge. Limited combinations of these interpolants can be used to produce more accurate estimates of solutions on the edge AB. Applying the divergence theorem and using both the interpolants on either side of the edge AB also allows derivative values to be estimated on that edge in a similar way to that of Koren [8].

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.

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

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 $q(x_i \ , y_i \ , t)$ and $A_N(t)$ is an $N \times N$ matrix which may or may not be the identity matrix depending on the discretization method. 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, $\underline{es}(t) = \underline{q}(t) - \underline{Q}(t)$, and the global time error, $\underline{ge}(t) = \underline{Q}(t) - \underline{V}(t)$. That is,

$$\begin{aligned} \underline{E}(t) = \underline{q}(t) - \underline{V}(t) &= (\underline{q}(t) - \underline{Q}(t)) + (\underline{Q}(t) - \underline{V}(t)) \\ &= \underline{es}(t) + \underline{ge}(t). \end{aligned} \tag{2.4}$$

2.1. SOFTWARE FOR TIME INTEGRATION.

The SPRINT package (Software for PRoblems IN Time) of Berzins, Dew and Furzeland [2] is a general-purpose computer program for the numerical solution of mathematical models that involve mixed systems of time-dependent algebraic, ordinary and partial differential equations (o.d.e.s and p.d.e.s). Shell Research Limited and the School of Computer Stu- es at Leeds University collaborated to write SPRINT and so provide a flexible and open-ended software tool to enable a user to solve a wide range of problems within a single framework . The software package consists of a set of well-defined and independent modules that are controlled by a supervisory routine. The internal structure of the package allows the individual modules to be easily replaced and in this way the user has access to different combinations of modules from the three main component areas in the package, - the time integration method, the spatial discretization method and the linear algebra routines. The modules incorporate recent developments in numerical analysis and software such as o.d.e. integrators for differential-algebraic equations (d.a.e.s) and for handling discontinuities, type-insensitive codes for o.d.e.s where the degree of stiffness varies, and adaptive space remeshing methods for p.d.e.s. The core of the software package is a versatile set of differential-algebraic implicit integrators with the flexibility to deal with stiff or non-stiff d.a.e.s coupled with algebraic equations and full/banded/sparse Jacobian matrices computed analytically or numerically when Newton's method is used to solve the non-linear equations. In the case of two space dimensional problems the systems of equations are large. In the case of non-stiff o.d.e.s functional iteration is used while in the stiff case iterative methods, such the reduced storage SPRINT module written by Seward [13], are used to reduce the computational cost.

The greatest flexibility is obtained when the user writes a program which calls the SPRINT driving routine directly. This program consists of initialisations of the parameters to be passed into the SPRINT driving routine, calls to the linear algebra module setup routine and the d.a.e. integrator setup routine followed by a call to the SPRINT driving routine to per-

form the integration. This call specifies the names of the time integration module, LU decomposition and back-substitution routines, RESID (problem definition) routine and the name of the MONITR routine (which is called at the end of every timestep).

The user-supplied RESID routine defines the system of differential-algebraic equations to be solved. In the case of p.d.e. problems this means that it is this routine that performs the semi-discretization of the p.d.e. The integrator supplies approximate vectors for the solution and its time derivative, $\underline{V}(t)$ and $\dot{\underline{V}}(t)$. The main purpose of the RESID routine is to compute the residual vector $\underline{R}$ which is obtained by substituting the vectors $\underline{V}(t)$ and $\dot{\underline{V}}(t)$ into the d.a.e system that is being solved. I.e. for equation (2.3),

$$\underline{R} = A_N\dot{\underline{V}}(t) - \underline{F}_N\ (t\ ,\ \underline{V}(t))\ , \tag{2.5}$$

The integration may be interrupted by the user from RESID to force the integrator to either stop the integration, reduce the time-step to avoid a physically impossible solution value or to terminate the current step and enter the MONITR routine.

An important feature of SPRINT is the capability to handle p.d.e. space remeshing schemes. After each step taken by the SPRINT integrator a routine, generic name MONITR, is called which allows the user to perform intermediate output or calculations (e.g. the integration may be restarted , the step-size changed or restricted to satisfy a CFL condition). The unique feature of the MONITR routine is that it has the power to access the whole of the non-linear equations solver in SPRINT. The MONITR routine was designed for tasks such as o.d.e. global error estimation and remeshing at discrete times.

2.2. TIME ERROR CONTROL IN CODES FOR SOLVING O.D.E.S.

Most codes, such as SPRINT, for solving time dependent o.d.e.s control either the local time error per step, (LEPS), with respect to a user supplied accuracy tolerance, TOL, or the local time error per unit step (LEPUS), $\dfrac{\underline{le}_{n+1}(t_{n+1}\ ,\ \mathrm{TOL})}{k_n}$. When controlling the LEPS it is difficult to establish a relationship between the accuracy tolerance, TOL, and the global time error. On the other hand, if the LEPUS is controlled then it can be shown, Stetter [14], that the time global error is proportional to the tolerance that is

$$\underline{ge}(t) = \underline{v}(t)\,\mathrm{TOL} + o(\mathrm{TOL}), \tag{2.6}$$

where $v(t)$ 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. In particular it is not generally efficient to use a fixed value of TOL.

3. 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. Although these errors should be balanced, in practice the spatial discretization error must actually dominate so that the estimate of the spatial discretization error and the spatial remeshing process remain unpolluted by temporal error. One way to balance the error in this way is to make use of the equation for the evolution of the spatial error.

$$A_N(t)\dot{\underline{es}} = \underline{F}_N(t\ ,\ \underline{Q}(t) + \underline{es}(t)) - \underline{F}_N(t,\ \underline{Q}(t)) + \underline{TE}(t,\underline{q}(t)) \tag{3.1}$$

where the vector of spatial truncation errors as denoted by $\underline{TE}(t,\underline{q})$, is defined by

$$\underline{TE}(t,\underline{q}) = A_N(t)\dot{\underline{q}} - \underline{F}_N(t\ ,\ \underline{q}(t)) \quad \text{and} \quad \underline{es}(0) = 0\ . \tag{3.2}$$

The estimate of the spatial truncation error used is calculated by using h-extrapolation. The 'coarse' mesh Δ^c is that created by the Finite Quadtree mesh generator described below while the actual mesh Δ used to compute the numerical solution to the p.d.e. is created by uniformly subdividing each coarse element into four. Let $q^c(t)$ be the restriction of the p.d.e. solution $q(x,y,t)$ to the new mesh Δ^c . The vector of spatial truncation errors , $\underline{TE}^c(t\ ,\ q^c(t))$, on the coarse mesh Δ^c is defined in the same way as the truncation error on the fine mesh (equation (3.2)) by

$$\underline{TE}^c(t\ ,\ \underline{q}^c(t)) \ = \ A_M(t)\ \dot{\underline{q}}^c(t) \ - \ \underline{F}_M(t\ ,\ \underline{q}^c(t)) \tag{3.3}$$

and the components of the coarse and fine truncation errors at the centroid of the ith coarse triangle and the 4i th fine triangle are approximately related by

$$[\underline{TE}^c(t\ ,\ \underline{q}^c(t))]_i \approx 2^p\ [\underline{TE}(t,\underline{q}(t))]_{4i} \tag{3.4}$$

where p is the order of the space truncation error. The spatial truncation error can then be estimated by defining the M dimensional vector $\underline{V}^c(t)$ as

$$[\ \underline{V}^c(t)\ ]_i \ = \ [\ \underline{V}(t)\ ]_{4i}\ \ i = 1, \ldots, M$$

(and $\dot{\underline{V}}^c(t)$ is similarly defined using $\dot{\underline{V}}(t)$) . Lawson et al. [9] show that providing the space error dominates the time error then the truncation error on the coarse mesh can be estimated by

$$\underline{TE}^c(t\ ,\ \underline{q}^c(t)) \approx \frac{2^p}{2^p - 1}\ [\ A_M(t)(\dot{\underline{V}}^c) \ - \underline{F}_M(t\ ,\ \underline{V}^c(t))\ ] \tag{3.5}$$

The spatial truncation error in the solution on the mesh δ may then be estimated by using equation (3.4) and by using linear interpolation to estimate the truncation error at those centroids of the fine mesh not present in the coarse mesh.

Equation (3.1) shows how the spatial accuracy varies with time and can be used to define an error control strategy in which the accuracy tolerance is related to and varied with the spatial discretization error. Lawson, Berzins and Dew [9] 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,

$$||\underline{le}_{n+1}(t_{n+1}\ ,\ \mathrm{TOL})|| \ < \ \varepsilon\ ||\underline{es}(t_{n+1}) - \underline{es}(t_n)||. \tag{3.6}$$

It can be shown, [9], 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 can be applying the mean value theorem to the right side of the equation.

$$||\underline{le}_{n+1}(t_{n+1}\ ,\ \mathrm{TOL})|| \ < \ \varepsilon\ k_{n+1}\ ||\dot{\underline{es}}(t^*)||\ \text{ for some } t^*\ \varepsilon\ [t_n\ ,\ t_{n+1}]. \tag{3.7}$$

where $k_{n+1} \ = \ t_{n+1} - t_n$.

Other approaches in a similar spirit are discussed by Lawson et al. [9]. In practice we need to integrate equation (3.1) for the spatial error at the same time as the main equation is implemented. In the case when an implicit method is used for the main integration computationally simple methods can be devised, [9]. In the case of explicit methods low order Runge-Kutta methods can be applied to equation (3.1). In both cases this subsidiary integration is performed by the MONITR routine in a modified LEPUS version of SPRINT.

It is worth noting that the approach defined by equation (3.6) also has similarities with existing local refinement methods such as those described by Flaherty et al. and Berger, see [7]. These methods balance the local time error against a local estimate of the space error. Define $\hat{\underline{es}}(t)$ as the local solution of equation (3.1) given the assumption that $\underline{es}(t_n) = 0$ - in other words as the local in time space error. A local in time error balancing approach is then given by

$$||\underline{le}_{n+1}(t_{n+1}, \text{TOL})|| < \varepsilon \; k_{n+1} \; ||\underline{\hat{es}}(t_{n+1})||. \tag{3.8}$$

It is by no means clear as to which of the approaches (3.6) or (3.8) will prove the most robust and reliable in the long term. It is however clear that whichever is used it is desirable to have an estimate of the spatial error that reflects how this error changes globally in time.

4. FINITE QUADTREE MESH GENERATION AND ADAPTION.

The Finite Quadtree meshing procedures consist of two main steps [1]. In the first step, a quadtree is utilized to discretize the model and to keep track of the discrete information The quadtree is a collection of hierarchically structured cells that are subdivided to the required sizes and are tied together through the use of a tree data structure. Each cell contains discrete topological information about the portion of the model where the geometry spatially overlaps the position of the cell. The model intersections with the cells, and the cell corners and portions of the cell edges that are within the model become finite element nodes and finite element edges in the mesh. In the second step, additional edges and nodes are added to the tree and mesh databases as the terminal cells of the quadtree are broken down and grouped into finite elements.

The Finite Quadtree mesh generator contains two coupled databases, that of the tree and the mesh [2]. The meshing procedures access the model information through geometric communication operators, allowing the use of other modelers, and no duplication of data. The tree database contains the root, the continuation quadrants, and the terminals quadrants of the tree. All quadrants point to their parents, a continuation quadrant points to its four subquadrants, and a terminal quadrant points to the edges and the nodes within it. The mesh database contains the finite elements, the finite element edges, and the finite element nodes. A finite element points to its finite element edges, the quadrant from which it was created, and the model face it is in. A finite element edge points to its finite element nodes, the finite elements and terminal quadrants on either side of it, and the model edge if appropriate. The finite element nodes point to its finite element edges, and its parametric value along a model edge when appropriate. The finite element edges and nodes are the same entities that are accessed by the terminal quadrants. The abundancy of information in the tree and mesh databases are important in adaptive mesh updating.

A very simple approach has been taken to link the analysis program with that of the mesh generator. Functions were written for each analysis mesh database entity. For example the function $edges(i,j)$ returns the pointer information $i = 1,2$ for mesh points at the ends of the j th edge or the element pointer information to the elements on either side of the j th edge $i = 3,4$. When called, the functions act as retrieval operators and access the corresponding information in the Finite Quadtree mesh database. Using this approach, the existing mesh database in the analysis program did not have to change.

In this paper, the local remeshing capabilities [2] of the mesh generator are being used to guide the mesh updating. The input to the local remeshing procedure consists of a list of elements and the level of refinement or unrefinement being requested. The quadtree in the area of the elements requesting new sizes is changed, creating new quadrants, and therefore new element sizes. The local mesh updating procedures consist of the following steps. First, the quadrants to be changed are obtained from the element-to-quadrant pointers in the finite element mesh database. Next the old information that is associated with the quadrants to be changed is deleted. The only information that is saved is the starting and ending parameter values of the discrete boundary edges, along with the corresponding model edge pointers. This information is examined in a later step to determine which portions of the boundary are to be rediscretized. With the discrete edge and node information already removed in the quadrants to be refined, it is simply a matter of subdividing empty quadrants to the levels requested by the adaptive error estimator. The reverse holds true when unrefinement is requested. For unrefinement, the information is deleted in all siblings of a parent quadrant, the empty terminal quadrants are deleted, and the parent becomes the new terminal quadrant. The

unrefinement process can continue upward to the requested level in the quadtree, creating larger terminal quadrants, until a quadrant is encountered that has requested refinement. Unrefinement has to stop at this point, since refinement takes precedence over unrefinement. After the tree has been changed to the new levels, a transition zone is created around the locally changed quadrants. The mesh information is deleted in these quadrants since a new mesh will have to be generated at these locations that will hook up the old unchanged mesh with the new locally updated mesh. The portions of the model edges that need to be locally rediscretized are now reintersected with the new locally changed tree. The last step of the local remeshing algorithm is the local generation of the finite elements within the locally updated quadrants and the quadrants in the transition zone [2].

5. TOWARDS AN AUTOMATIC ALGORITHM.

The automatic algorithm basically 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 [10]. One fundamental difference from the one-dimensional case is that of specifying how many levels of quadtree should be refined or coarsened. In most cases only one level of refinement or coarsening is requested.

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 $\mathcal{E}(t)$ of $||\underline{es}(t)||$ is calculated, and if

$$\mathcal{E}(t) > 0.95 \cdot EPS$$

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 [12] 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, but there is an increased risk of convergence failures. A full restart will be performed automatically by SPRINT [4] in the event of repeated convergence failures. Since the accuracy tolerance for the time integration over the next time step depends partially on the error incurred prior to spatial remeshing, this tolerance must be modified according to the expected reduction in the spatial discretization error. 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.

6. CONCLUSIONS.

In this paper a blueprint has been produced for the construction of a prototype solver for time dependent fluid flow applications. The results obtained from preliminary experiments on time-dependent convection-dominated problems indicate that the algorithm is a promising start to developing codes which automatically control the error in the computed solution.

ACKNOWLEDGEMENTS.

M. Berzins gratefully acknowledges the financial support and warm hospitality of the Rensselaer Design Research Center while on leave from Leeds University.

J. Lawson acknowledges the support of Shell Research Limited.

J.E. Flaherty acknowledges the support of the U.S. Airforce Office of Scientific Research under grant No. AFOSR-90-0194.

REFERENCES.

1. Baehmann, P. L., Wittchen, S. L., Shephard, M. S., Grice, K. R., Yerry, M. A., Robust, Geometrically Based, Automatic Two-Dimensional Mesh Generation, *IJNME*, 24, 1043-1078, 1987.
2. Baehmann P.L. Shephard M.S. Adaptive Multiple-Level h-Refinement in Automated Finite Element Analysis, *Engineering with Computers*, 5, 235-247, 1989.
3. P.L. Baehmann *Automated Finite Element Modelling and Simulation*, Ph.D. Thesis, May 1989, Rensselaer Design Research Center Report TR 89014, Rensselaer Polytechnic Institute, Troy, New York, 12180.
4. Berzins, M. Dew, P.M. and Furzeland, R.M. Developing P.D.E. Software Using the Method of Lines and Differential Algebraic Integrators, *Appl. Num. Maths.* 5, 1989.
5. Bieterman, M. and Babuška, I. An Adaptive Method of Lines with Error Control for Parabolic Equations of the Reaction-Diffusion Type, *J. of Comp. Phys.*, 63, 33-66.
6. Chakravarthy, S.R. and Osher, S. Computing with High Resolution Upwind Schemes for Hyperbolic Equations. *Lectures in Applied Mathematics Vol 22* 1985, American Mathematical Society.
7. Flaherty J.E., Paslow P.M., Shephard M.S. and Vasilakis J.D., *Adaptive Methods for P.D.E.s* S.I.A.M. Philadelphia 1989.
8. Koren B. , *Multigrid and Defect Correction for the Steady Navier Stokes Equations - Applications to Aerodynamics*, Ph.D. Thesis, Centrum Voor Wiskunde en Informatica, Amsterdam, 1989.
9. Lawson, J. Berzins, M. and Dew, P.M. Balancing Space and Time Errors for Parabolic Equations. *SIAM J. Sci. Stat. Comput.*, to appear 1990.
10. Lawson J.L. and Berzins M., Towards an Automatic Algorithm for the Numerical Solution of Parabolic P.D.E.s using the Method of Lines. Paper presented as a high-lighted talk at the 1989 ODE Conference, London, (to appear in proceedings).
11. Ludwig, R.A., Flaherty, J.E., Guerinoni, F., Baehmann, P.L. and Shephard, M.S., Adaptive Solutions of the Euler Equations Using Finite Quadtree and Octree Grids, *Computers and Structures* 30, No.1/2, pp. 327-336, 1988.
12. Ramshaw, J.D., Conservative Reasoning Algorithms for Generalized Two-Dimensional Meshes, *J. Comput. Phys.* 59, pp. 193-199, 1985.
13. Seward , W.L. Solving Large ODE Systems Using a Reduced System Iterative Matrix Solver *Research Report Cs-89-38* Department of Computer Science, University of Waterloo, Ontario, Canada.
14. Stetter, H.J. *Considerations Concerning a Theory for O.D.E. Solvers, Numerical Treatment of Differential Equations*, ed. by R. Bulirsch, R.D. Grigorieff and J. Schroder, Lecture Notes in Mathematics 631, Springer Verlag, New York 188-200.