Grid Adaptation and Multidimensional Upwinding [1]

M.E.Hubbard

Numerical Analysis Report 8/94

Department of Mathematics
P.O.Box 220
University of Reading
Whiteknights
Reading
RG6 2AX
United Kingdom

[1]The work reported here forms part of the research programme of the Oxford/Reading Institute for Computational Fluid Dynamics and was supported by DRA Farnborough.

Abstract

The quality of the solution to systems of differential equations can be improved significantly by adapting the computational grid on which they are calculated so that nodes are concentrated in significant regions of the solution.

In this report, two very simple schemes are described for moving the grid to improve the solution without altering the number of nodes or the connectivity of the grid. These are used on unstructured triangular meshes to find accurate steady state solutions to the linear advection equation and the Euler equations for inviscid fluid flow in two dimensions using multidimensional upwinding.

Contents

1 Introduction

It is widely known that the quality of the solution of a scalar differential equation or a system of differential equations can be improved by means of grid adaptation techniques. The philosophy behind these methods is that the error will be reduced if the computational mesh on which the calculation is being carried out can be changed to better represent the solution.

On triangular grids this can be done in two distinct ways; refinement and movement. Edge swapping is another technique which can be used, since many schemes for solving differential equations give better results when edges of the mesh are aligned with particular flow features. A detailed discussion of grid adaptation techniques can be found in [9, 10].

The simplest way to refine the grid is to do so throughout the flow domain, but this is unnecessarily expensive, as it introduces extra nodes in regions where their effect on the solution quality is negligible, *e.g.* if the solution is constant over the domain then increasing the number of nodes will have no effect on it. Instead, some form of selective refinement is required, dependent on the solution itself.

However, both selective refinement and edge alignment can, to a large extent, be achieved by the third option, grid movement, which has the added advantage of involving no change in the number of nodes or the connectivity of the grid, which can become expensive and messy. This report has been written to illustrate the use of a couple of very simple and cheap grid movement strategies used in conjunction with recently developed multidimensional upwinding techniques [2, 4, 5, 6, 8] for solving test problems involving the linear advection equation and the Euler equations for inviscid fluid flow. It is known [3] that shocks are captured across 2 or 3 cells when these methods are used, so it is obvious that an appropriate repositioning of the nodes can significantly improve the definition of such flow features. The task is greatly simplified by considering only steady state solutions to these equations. This allows a number of difficulties to be ignored which would otherwise have to be dealt with if the solutions were time dependent.

The following section describes the grid movement strategies which have been used. First, the concept of the monitor surface is defined and, in particular, its rôle in driving the adaptation. If the surface is chosen so as to accurately represent the character of the solution, then if the grid can be moved to improve the resolution of the monitor surface it can be expected to also decrease the error in the solution. Therefore, the adaptation algorithms are designed to move nodes towards regions of the solution with a particular character. For the purposes of this report these will be taken to be regions of high gradient. This means that for the Euler equations, for example, nodes will tend to move towards features such as shocks. Future work will also take into account those areas where the surface curvature is high, to further improve results. Two very simple algorithms are described for moving nodes in one dimension and these are generalised for use in two dimensions. The overall solution strategy is then described for producing converged, steady state solutions on adapted grids using multidimensional upwinding, and a wide variety of results is presented to show the features of each of the two-dimensional methods.

2 Grid Movement

The intention of this work is to investigate and develop a simple grid movement algorithm which can be used in conjunction with multidimensional upwinding techniques to improve the accuracy of the solution of systems of hyperbolic conservation laws without significantly increasing the work needed to achieve them. This task is simplified greatly by considering only steady state solutions to these equations. None of the methods described here could be used in their current form to solve time dependent problems since they are unable to track rapidly changing flow features and are non-conservative.

The adaptation schemes used here could hardly be simpler since they consist only of a single local node movement step. The number of nodes and the connectivity of the grid remains unchanged throughout the solution procedure, thus avoiding a significant contribution to the expense of the algorithm. Also, the displacement of a node depends only on information stored at adjacent nodes (or in adjacent cells) which ensures that the extra cost of moving the grid is negligible. The main extra expense incurred is due to the occurrence of very small cells, reducing the allowable time-step (for stability) which is proportional to the cell size. This is a problem with any grid adaptation scheme used in conjunction with an explicit solver, though, and can be partially alleviated by the use of local time-stepping, giving a remarkably cheap and effective algorithm. It is also necessary to ensure that the procedure does not cause the grid to become highly distorted, since this may hinder convergence to the steady state solution.

2.1 Monitor Surfaces

It is instructive here to introduce the idea of a monitor surface [9], or monitor curve in one dimension. The behaviour of the solution of a system of differential equations is usually well monitored by a quantity or combination of quantities, so it can be expected that by accurately resolving these quantities on the computational mesh the error in the solution to the differential equation can be kept low.

Taking the Euler equations as an example, the quantity may be chosen to be the density whose values at the nodes of the mesh define a function which, when raised above the plane of the grid, creates the monitor surface such as that shown in Figure 2.1. The monitor curve is defined in a similar way in one dimension.

To gain a more accurate approximation to the solution as it develops in time, the grid should be adapted to improve the resolution of the monitor surface. The adaptation is carried out so as to cluster nodes in appropriate regions of the domain. Initially these regions can be considered to be where the gradient of the

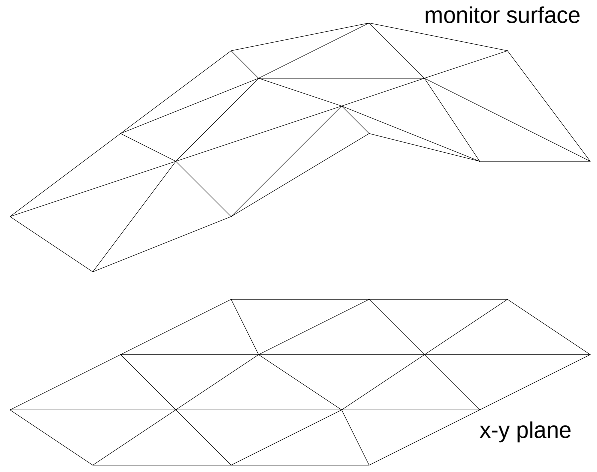


Figure 2.1: An example of a monitor surface.

monitor surface is high, and this report is concerned with methods of this form. However, for even greater accuracy it may be necessary to consider adapting to regions of high surface curvature, so that large variations in the gradient of the surface can be resolved as well as steep slopes. All the algorithms presented here can be altered quite simply to account for this but such improvements are not considered here.

2.2 One Dimension

Two distinct one-dimensional algorithms have been considered, although both actually reduce to very similar methods and are essentially iterative schemes which attempt to satisfy some form of mesh equidistribution at convergence.

2.2.1 Baines' Algorithm

The first algorithm was developed by Baines [7] with the intention of producing best L_2 fits to continuous functions by piecewise polynomials with variable nodes. Two distinct algorithms were devised, one which attempts to find best piecewise linear discontinuous approximations and the other to find the best piecewise constant approximation. The nodal movement is designed to minimise the l_2 error at the discontinuities. Since the underlying continuous function used in multidimensional upwinding methods is already piecewise linear, finding the best piecewise linear approximation to this function would not achieve any node movement. Therefore, it is only the piecewise constant algorithm which is of any use here. If the monitor function is represented by the piecewise linear continuous function u then the scheme in one dimension can be written in the algorithmic form

a) For each node n, calculate an average value of u,

$$\overline{u}_n = \frac{u_{n-1} + 2u_n + u_{n+1}}{4} = \frac{1}{2}\left(\frac{u_{n-1} + u_n}{2} + \frac{u_n + u_{n+1}}{2}\right). \tag{2.1}$$

b) Extend the linear approximations in the adjacent cells until they intersect the horizontal line $u(x) = \overline{u}_n$.

c) Choose the intersection point closest to x_n and move node n there.

If the intersection points in c) are both the same distance from the node then take an average of these two positions and move the node there. This is shown graphically in Figure 2.2. The intersections of the horizontal line, $u(x) = \overline{u}_n$, and the extended solution approximations in the neighbouring cells give the two possibilities for the the new position of node n (shown by the filled circles) and the closer of these to the original position is taken. The new nodal position is shown by a circle.

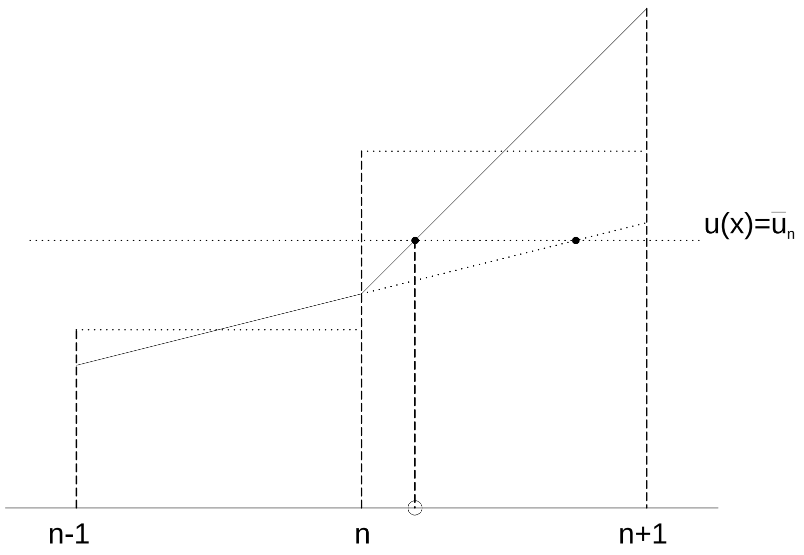


Figure 2.2: Baines' algorithm in one dimension.

It is easy to see from this construction that, as long as the monitor curve is monotonic within the two cells, this algorithm will always move nodes towards the cell which has the larger value of $|\Delta u|$ and never move them beyond the centroid of either cell. This latter property is important because it ensures that mesh tangling cannot occur. The former indicates that at convergence (when $\Delta x = 0$) the grid equidistributes $|\Delta u|$. This should cluster nodes in regions where the solution gradient is high, as desired.

Unfortunately, none of this remains true when the function has a turning point in the interior of the domain. As Figure 2.3 demonstrates, when a minimum (or a maximum) occurs, particular on highly irregular grids, not only is it possible for a node to move away from the larger value of $|\Delta u|$, it can also move beyond the neighbouring node, causing tangling. Worse still, nodes can be moved large distances when $|\Delta u|$ is almost the same on either side of the node and the convergence of the method is affected.

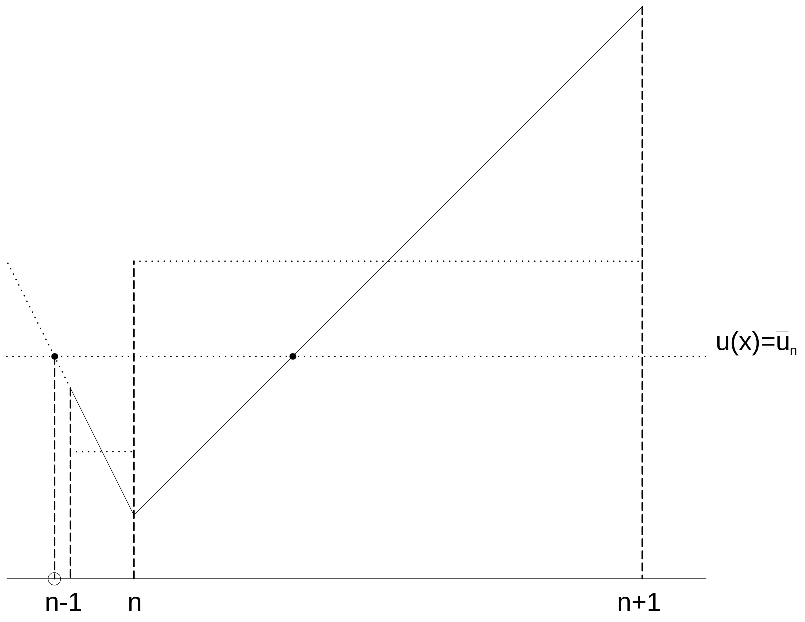


Figure 2.3: The failure of Baines' algorithm in one dimension at a minimum.

This problem can be rectified easily by not allowing nodes at turning points to move. The function is now split into monotonic portions and the converged grid will equidistribute $|\Delta u|$ within each of these regions separately. The same procedure must also be carried out in regions where the gradient is zero, otherwise the algorithm does not produce a unique displacement for the nodes.

An interesting side issue here is the existence and uniqueness of grids which equidistribute $|\Delta u|$ when the monitor curve is not monotonic. To illustrate the problem, consider the simple case of a function with a single maximum in the domain and a grid of N nodes. If $u_0 = u_N$ and N is even, then no equidistributed grid exists. If N is odd, though, there are infinitely many. If $u_0 \neq u_N$ then there is always at least one equidistributed grid, while sometimes, very rarely, there is a second. The inclusion of more turning points further complicates the problem. This gives another reason for dealing separately with nodes at turning points, although it suggests that a more sophisticated approach is needed than that used above.

Returning to the algorithm, simple algebraic manipulation leads to an expression for the nodal displacement in terms of Δu and Δx,

$$\Delta x_n = \frac{\Delta u_{n+\frac{1}{2}} - \Delta u_{n-\frac{1}{2}}}{4\left(\frac{\Delta u}{\Delta x}\right)_{n+\frac{\sigma}{2}}} \tag{2.2}$$

as long as the derivative is not zero here and where

$$\sigma = sign\left(\left|\frac{\Delta u}{\Delta x}\right|_{n+\frac{1}{2}} - \left|\frac{\Delta u}{\Delta x}\right|_{n-\frac{1}{2}}\right). \tag{2.3}$$

2.2.2 Weighted Averaging

The explicit expression (2.2) for the nodal displacement given by Baines' algorithm leads neatly to the consideration of a second kind of method which is, if anything, even simpler. It consists of moving each node to a weighted average of the positions of the centroids of the adjacent cells, so the new position of node n is given by

$$x_n^{new} = \frac{w_{n-\frac{1}{2}}\overline{x}_{n-\frac{1}{2}} + w_{n+\frac{1}{2}}\overline{x}_{n+\frac{1}{2}}}{w_{n-\frac{1}{2}} + w_{n+\frac{1}{2}}} \tag{2.4}$$

where the weights $w_{n\pm\frac{1}{2}}$ can be chosen appropriately and $\overline{x}_{n\pm\frac{1}{2}}$ are the coordinates of the midpoints of the cells. If the weights are of the same sign throughout the grid then the node cannot move beyond the midpoints of its neighbouring cells and so tangling is automatically avoided.

If the weights are chosen to be $w = \frac{\Delta u}{\Delta x}$ then the nodal displacement becomes

$$\Delta x_n = \frac{\Delta u_{n+\frac{1}{2}} - \Delta u_{n-\frac{1}{2}}}{2\left(\frac{\Delta u_{n+\frac{1}{2}}}{\Delta x_{n+\frac{1}{2}}} + \frac{\Delta u_{n-\frac{1}{2}}}{\Delta x_{n-\frac{1}{2}}}\right)} \tag{2.5}$$

which differs from that in equation (2.2) only in the denominator. For comparison with Baines' algorithm, Figure 2.4 shows the construction of the node movement for a weighted average of the positions of the adjacent *nodes*. It is clear, in this case, that for monotonic functions this choice of weight will move nodes in the same direction as Baines' algorithm but, in general, further - possibly too far - since the displacement depends on the average of the two gradients rather than the one with higher magnitude. However, carrying out a weighted average of the nodal positions only ensures that node n doesn't overtake either of its neighbours provided that the updates are carried out in a Gauss-Seidel manner (each node is moved before the displacement of the next node is calculated). If a Jacobi update is used instead (all the displacements are found before any of the nodes are moved) then tangling can occur. This is rectified by halving the displacements or, equivalently, using the centroids of the adjacent cells in the averaging as in (2.4). Unless otherwise stated, Jacobi updating is used in this report because it is independent of the order in which the nodes are moved although, in practice, little difference has been found between the two updating procedures.

Generally, the choice of $w = \frac{\Delta f}{\Delta x}$ leads to grids which equidistribute $|\Delta f|$ at convergence, where f is usually a function of the solution data and should be monotonic to assure convergence. The above choice of weight should therefore equidistribute $|\Delta u|$. Figure 2.4 also shows that these weights can be considered as $w = \tan\theta$ where θ is the angle between the normal to the monitor surface and the vertical. This is useful when considering generalisations to two dimensions.

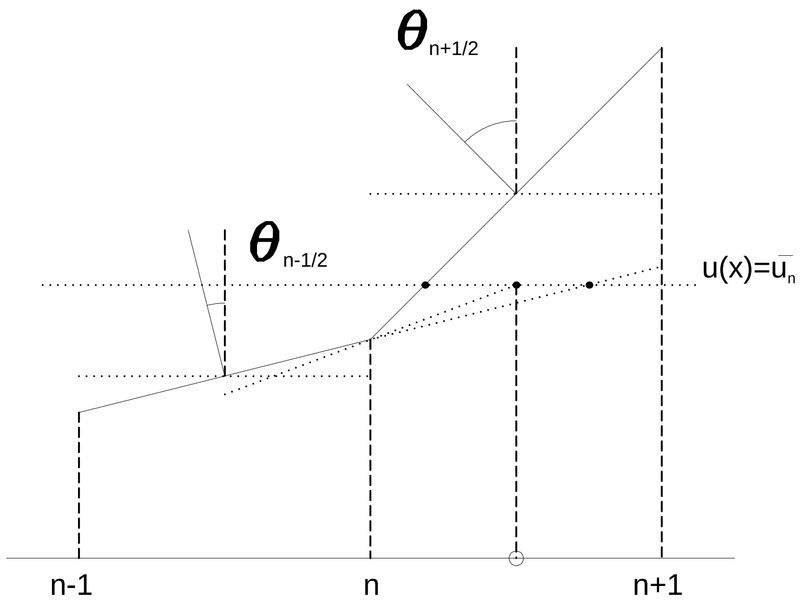


Figure 2.4: Weighted averaging in one dimension.

This similarity between the two methods also means that they have the same problems when the monitor curve is not monotonic. At turning points the weights now have different signs, so that tangling can occur. Moreover, a unique equidistributed grid can no longer be guaranteed and convergence suffers as a result. This can be partially rectified by modifying the weights to give $w = |\tan\theta|$ which removes the possibility of tangling, by ensuring that the weights are always positive, and significantly improves the behaviour at turning points. However, as before, nodes at maxima and minima must be fixed to improve convergence and gain ‘piecewise equidistribution’.

An alternative, and more usual, choice for the weights is $w = \frac{\Delta s}{\Delta x}$, where s is the cumulative arc length of the monitor curve [1]. It can be seen from Figure 2.4 that these weights can also be written as $w = \sec\theta$ which are always positive, so the scheme is tangle free. Nodes always move into the adjacent cell with the higher arc length and it is this quantity that is equidistributed on the converged grid. Unfortunately, since the arc length is not calculated exactly by integrating along the curve but is approximated by the chord length between the nodes, the equidistribution is not necessarily unique for non-monotonic functions. However, the likelihood of this difficulty causing problems, particularly when the algorithm is used in conjunction with a time-stepping scheme, is negligible, so nodes do not need to be fixed at turning points. As with the previous choice of weight, nodes will be attracted towards regions where the gradient of the monitor curve is high, but the tangling and equidistribution problems have largely been eradicated and there is no longer a singularity when the derivative of the monitor curve is zero.

Clearly other quantities could be equidistributed by an appropriate choice of weight, and this is one way by which the method can be extended to include curvature monitoring on the surface although this is not considered here.

2.3 Two Dimensions

Each of the two-dimensional node movement algorithms used in this report is, in some sense, a generalisation of one of the schemes presented in the previous section and, as before, they tend to attract nodes towards regions on the monitor surface where the gradient is high. All the schemes can also be extended further into three dimensions, but that is not dealt with in this report.

2.3.1 Baines' Algorithm

The first method tried was an extension, by Baines [7] of his own one-dimensional algorithm. As before, it is only the piecewise constant algorithm which is of any use since the underlying solution is already represented by a continuous piecewise linear function.

The algorithm in two dimensions is

a) For each node n, calculate an average value of u,

$$\overline{u}_n = \frac{1}{N} \sum_{i=1}^{N} \frac{u_i + u_n}{2} \tag{2.6}$$

where i is one of the N grid edges emanating from node n (spokes) and u_i is the value of u at the end of the spoke i.

b) Extend each of the spokes until it intersects the horizontal plane $u(x, y) = \overline{u}_n$.

c) Choose the intersection closest to (x_n, y_n) and move node n there.

As in one dimension, if two or more intersection points in c) are the same distance from the node then an average of the displacements is taken. The scheme is shown graphically in Figure 2.5. The dotted lines represent the cells raised on to the plane $u(x, y) = \overline{u}_n$ and the larger dots mark the intersections of the extended spokes with this plane. (There are only 5 because one spoke is parallel to the plane.) The nearest of these to the original nodal position is taken and the node is moved to the position marked by the circle.

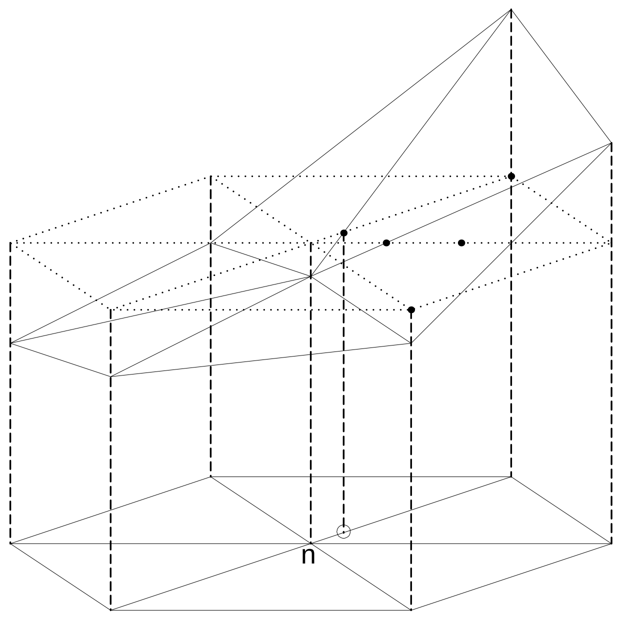


Figure 2.5: Baines' algorithm in two dimensions.

On closer examination of this scheme, it soon becomes apparent how few of the nice one-dimensional properties remain satisfied in higher dimensions. It is no longer true that the node always moves towards the neighbour which gives the highest value of $|\Delta u|$, although it will move along the extended spoke with the gradient of highest magnitude. Also, the grid no longer satisfies an equidistribution property at convergence, even when the underlying function has no turning points. Convergence itself is also affected since mesh tangling is no longer avoided automatically, even when there are no turning points, and must be imposed artificially.

The link with weighted averaging has become more tenuous too, since in Baines' algorithm the choice of the direction of the nodal displacement can be easily separated from the calculation of how far it should move in that direction. This gives an expression for the displacement of the form

$$\Delta \vec{x}_n = \left(\frac{\overline{u}_n - u_n}{u_j - u_n} \right) (\vec{x}_j - \vec{x}_n) \tag{2.7}$$

where j is the index of the node at the far end of the chosen spoke.

2.3.2 Weighted Averaging

The two-dimensional form of the weighted averaging algorithm is a very straightforward generalisation, whereby each node is moved to a weighted average of the positions of the centroids of the surrounding triangles, giving

$$\vec{x}_n^{new} = \frac{\sum_{i=1}^{N} w_i \vec{x}_i}{\sum_{i=1}^{N} w_i} \tag{2.8}$$

where the $\vec{x}_i$ are the positions of the centroids of the neighbouring triangles and N is the number of cells adjacent to node n. It would be just as simple to take an average of the positions of the neighbouring nodes, but the weights used here are essentially cell-based quantities so centroids are more appropriate.

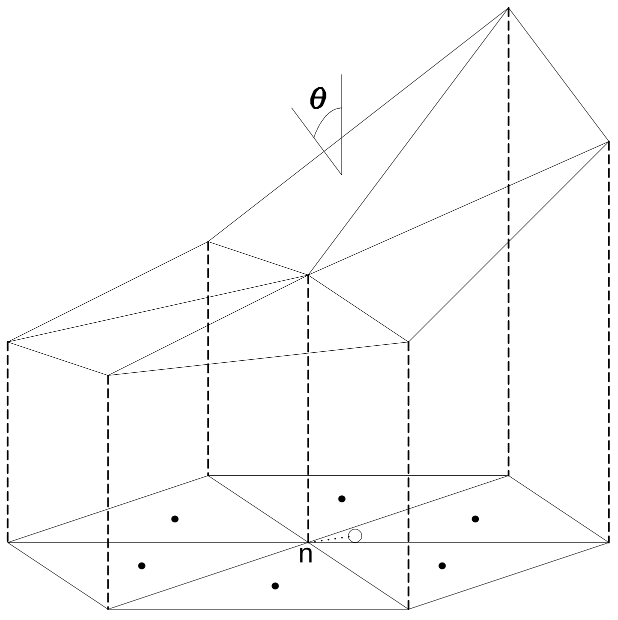


Figure 2.6: The two-dimensional weighted averaging algorithm.

However, because of the lack of a neat equidistribution property, the weighting function has to be chosen on a somewhat *ad hoc* basis. An obvious extension is to use precisely the same form of weight as in one dimension, so the choice leading to the equidistribution of $|\Delta u|$ would give $w_i = |\tan\theta_i|$ whereas the arc length equidistribution gives $w_i = \sec\theta_i$. θ_i is simply the angle between the normal to the triangle i in the monitor surface and the normal to the grid plane, as shown in Figure 2.6. Alternatively, Eiseman and Erlebacher [1] suggest moving each node to the 'centre of mass' of the surrounding triangles on the monitor surface. The weights then become $w_i = A_i \sec\theta_i$ where A_i is the corresponding cell area. These weights are, in fact, simply the areas of the triangles on the monitor surface. Both choices tend to move nodes towards regions where the solution gradient is high and can be modified to account for curvature modelling but, as in one dimension, the best method is not yet apparent.

From a practical point of view, the loss of most of the properties satisfied by the schemes in one dimension is not significant because the methods retain the more important qualitative features. This is not true of the non-tangling property, however. In two or more dimensions, particularly on the highly distorted grids which become common once the mesh is allowed to move, tangling occurs quite readily. The restriction to positive weights merely stops nodes from moving beyond the convex hull created by the centroids of the surrounding cells (a form of dual cell) so the algorithm is only guaranteed to be tangle free if every patch of cells associated with a node is convex. Figure 2.7 shows how tangling can occur. The dotted lines indicate the dual cells which bound the node movement and it is easy to see that, even if node 1 remains fixed, nodes 2 and 3 can move far enough upwards to cause triangle 123 to ‘flip’.

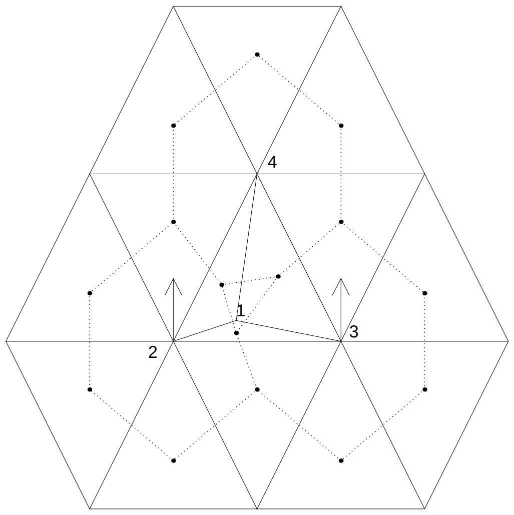


Figure 2.7: Tangling in two dimensions.

This can be avoided by artificially limiting the distance which a node can move. A simple but rather restrictive limit is

$$(\Delta x)_{max} = \min_i \left(\frac{A_i}{\max_{j=1,3} l_{ij}} \right) \tag{2.9}$$

where i indexes the cells surrounding the node, A_i is the area of cell i and l_{ij} is the length of the edge j of cell i. This expression is equivalent to half the smallest height of the surrounding triangles. This can be increased by a factor of two if Gauss-Seidel updating is used. A more sophisticated limit would require knowledge of the direction and magnitude of the displacement of the adjacent nodes and its calculation would cause unnecessary expense.

The final step in each of these algorithms is the interpolation of the solution on to the new grid. This is very simple since the underlying solution has been considered throughout to be piecewise linear continuous. However, the new approximation is then assumed to be piecewise linear continuous on the new grid, so information and accuracy is lost with each iteration. This results in a loss of conservation when it is combined with a time-stepping algorithm, which must be rectified by considering Langrangian type node movement algorithms [10] if time dependent problems are to be considered.

3 Solution Strategy

The method by which steady state solutions to all of the equations used here are found can be expressed very simply:

a) Run the time-stepping algorithm on an initial, fixed grid until the solution appears steady (but long before convergence is achieved).

b) Run the time-stepping interspersed with the grid movement until the mesh has adapted to the steady solution.

c) Run the time-stepping algorithm to convergence using the new grid and the solution from step b) as initial conditions.

Multidimensional upwind schemes are used here for the time-stepping [4, 5].

The decision to start moving the grid in step b) can be taken when the RMS of the residual over the grid drops below a certain level (typically a drop of 2 or 3 orders of magnitude from the initial residual), so that the significant flow features, such as shocks, have effectively stopped moving, but the choice of when to stop the grid movement is currently made on a much more *ad hoc* basis. In this work the number of grid iterations is prespecified but in the future some form of grid convergence monitor might be used.

It is not possible to simply run the time-stepping and grid movement to convergence together for a number of reasons. The convergence of the two-dimensional grid movement schemes on fixed functions is open to question, but when the function is changing with each iteration, as it is here, there is no reason to believe that the overall strategy will converge. Also, this stage of the method is not, as it stands, conservative due to the interpolation step of the grid movement. This can be rectified by moving the nodes in a modified Lagrangian manner [10] and including node velocities in the time-stepping scheme, but for steady state calculations this is not necessary. Either the grid converges (unlikely) or it is fixed after a certain number of iterations, after which the solution strategy returns solely to the conservative time-stepping scheme.

In one dimension it is necessary to fix nodes at turning points of the solution in order to gain converged grids for fixed functions. This also seems desirable in two dimensions but, with the introduction of a solution iteration between each grid iteration, convergence is lost anyway so it may be unnecessary. With the insistence on positive weights, it is assured that nodes at turning points (maxima, minima, saddles, etc.) behave in a correct qualitative manner and this, together with the solution iterations and the limit on the number of grid iterations, ensures that these features aren't rapidly eroded by interpolation and node movement. In fact, the fixing of these nodes increases the significance of node locking. This is a phenomenon where nodes which would, under normal circumstances, be moved by the adaptation scheme are fixed by other restrictions, such as imposing a limit on the cell size or forcing boundary nodes to remain on the boundary. To avoid this, no special treatment is given to nodes at turning points.

For the purposes of this report, stage b) of the strategy has always used alternating grid and solution iterations. It is possible, and may be even desirable to carry out more time-steps between each grid iteration, but that option is not considered here. It could though be too expensive to increase the number of grid iterations since the nodes could no longer be guaranteed to remain within its patch of adjacent cells.

One remaining problem in two dimensions is the treatment of boundary nodes. All interior grid points can use the full two-dimensional strategy but the position of the boundary nodes can be updated in one of two ways. One method is to compute a new position using the corresponding one-dimensional adaptation scheme along the boundary. Alternatively, a displacement can be found by the two-dimensional strategy and the node projected back on to the boundary. 'Corner' nodes remain fixed. In fact the first option also involves a projection step if the boundary is curved, a step which, if it is not done carefully, can lead to tangling when the curvature of the boundary is high. This means that the one-dimensional treatment is, in fact, no simpler or cheaper than using a modified two-dimensional update, so the second method, which is more consistent with the movement in the interior should be used.

4 Results

A large number of results are presented in this section, which are used to highlight the specific features of the different schemes described in the previous sections and to show the importance of certain parameters which need to be specified for the algorithms.

Two-dimensional steady state solutions of both the linear advection equation,

$$u_t + \vec{a} \cdot \vec{\nabla} u = 0, \tag{4.1}$$

and the Euler equations,

$$\underline{\mathbf{u}}_t + \vec{\nabla} \underline{\mathbf{f}}(\underline{\mathbf{u}}) = \underline{\mathbf{0}}, \tag{4.2}$$

are shown for a variety of test cases. A detailed understanding of these equations is not necessary - equation (4.1) represents the translation of an initial solution profile without changing its shape and with velocity $\vec{a}$, while (4.2) is the standard system of equations for compressible inviscid fluid flow.

One final adjustment to the solution algorithm is necessary from a practical point of view. This is the imposition of a lower limit on the cell size in the adapted grids. Here a limit of 10^{-3} is placed on the radius of the inscribed circle of each triangle. This not only stops cell sizes from reaching machine precision, but also ensures that the limit on the time-step - which is proportional to the cell size - does not become prohibitively small, hindering convergence. Surprisingly, if this limit is reduced, the results can become worse, especially when the solution contains a curved discontinuity. In such a case, cells can become too thin to adequately capture the discontinuity, so it extends into larger cells and becomes more diffuse.

No one-dimensional results are shown here since this work focuses on the interaction between these grid adaptation algorithms and multidimensional upwinding techniques.

In two dimensions, the linear advection equation is considered first, since it provides very simple steady state solutions on which to test the various grid movement algorithms. The test case used has a nonuniform velocity field $\vec{a} = (y, -x)$ over a rectangular domain, $[-1, 1] \times [0, 1]$, with the boundary conditions, $u = 1$ when $y = 0$ and $-0.65 < x < -0.35$ and $u = 0$ on all other inflow boundaries [4]. This represents the clockwise semicircular advection of a square profile. These were also used, with $u = 0$ everywhere else, as the initial conditions for all the calculations, including the adaptive ones, *i.e.* no solution iterations were carried out before starting to move the mesh.

In all cases the mesh used for calculating the fixed grid solutions, and as the initial grid for all the adaptive calculations, is very simple and regular, formed by dividing the domain up evenly into quadrilaterals and then inserting diagonals which alternate in direction between cells. Figure 4.1 shows the grid used in this case (2145 nodes, 4096 cells) and the steady state solution produced on it using Roe's Level scheme [4]. This was chosen from the many positive, linearity preserving schemes available simply because it produced the least diffusive results on the adapted meshes for linear advection. Here, as for all the linear advection results, contours are plotted between $u = 0$ and $u = 1$ at intervals of $\Delta u = 0.05$.

Figure 4.2 shows the grid resulting from the use of the basic algorithm of Baines and the converged solution obtained on that grid. There are two things which are immediately obvious from this, that the solution looks very good and the grid looks very bad. The actual quality of the grid is probably better represented by the accuracy of the solution on it but a grid such as this still introduces a sense of unease. The result was attained from 1000 grid iterations (the number used throughout this section) and already the mesh has become wildly distorted since the method attempts to move all of the nodes into the discontinuity, thus depleting the rest of the domain of nodes and producing huge triangles. Also, the grid is still moving, and if more iterations are used all the interior nodes move inexorably towards the discontinuities. This has little effect on the results here because the depleted region is flat, but it is both unnecessary and may become significant when the solution is more sensitive.

This can be improved, but only by introducing a tolerance to indicate where the solution is nearly constant. In these regions, a smoothing iteration can be carried out, moving each node to the mean position of its neighbours. This gives the results shown in Figure 4.3. The solution is slightly better than the previous one, particularly near the outflow boundary, and the grid is significantly smoother, although large triangles are still being produced in the region between the discontinuities.

The weighted averaging produces altogether more satisfactory results. The initial choice of $w = |\tan\theta|$, which was intended to emulate Baines' algorithm, gives a similar solution quality, Figure 4.4, although there is still a worrying tendency for all the nodes to drift towards the discontinuities. This seems to be because both methods are based on one-dimensional schemes intended to equidistribute $|\Delta u|$, so the final grid would not be expected to have any nodes in regions where the solution gradient is zero. Figures 4.5 and 4.6 show the results which are obtained using the weights $w = \sec\theta$ and $w = A\sec\theta$ respectively. Again the solution has been greatly improved from that achieved on the fixed grid, particularly in the first case, but now the grids exhibit all the desired (aesthetic) qualities, *i.e.* they are no longer highly distorted away from the discontinuities. Figure 4.6 is slightly misleading because it can be improved significantly by changing the scale of the grid. With these last two choices of weight, the size of the cells relative to the change in the solution across a cell becomes significant. For example, if the grid is scaled so that the cell sizes are very large compared to the solution values then $\sec\theta \approx 1$ over the whole domain and the grid is hardly moved at all. Figure 4.7 shows the quality of result which can be achieved if a grid scaling parameter is introduced. In this case the grid is reduced in size by a factor of ten, giving extremely sharp discontinuities without unnecessary distortion of the mesh, and the solution at the outflow face is an almost square profile of height 0.994. If either the weight is chosen to be $w = |\tan\theta|$ or Baines' algorithm is used, then scaling the grid has no effect.

The results for this calculation bear comparison with those on a fixed fine grid (8385 nodes, 16192 cells), Figure 4.8, giving much sharper discontinuities and using less than half the cpu time. It should be noted though that the grid must be fine enough to begin with. Figure 4.10 shows the result of attempting to adapt a grid, using $w = A\sec\theta$, with too few nodes to adequately cover the region. Although it improves on the results obtained on the fixed mesh, Figure 4.9, there are not enough nodes to achieve the quality of the solution on the fixed medium grid, even when grid scaling is used.

A second linear advection test case can be mentioned briefly here in order to exhibit the occurrence of node locking. This has a uniform velocity field of $\vec{a} = (\cos\pi/8, \sin\pi/8)$ over the domain $[0,1]\times[0,1]$, and boundary conditions $u = 0$ when $y = 0$, $u = 1$ when $x = 0$ [4]. These are used as the initial conditions for the calculations with $u = 0$ everywhere else. The unadapted solution on a 1089 node, 2048 cell grid is shown in Figure 4.11. The result shown in Figure 4.12 was created using $w = A\sec\theta$ and the grid is scaled down by a factor of ten. At first glance the result appears extremely good, but on closer examination of the bottom left hand corner, node locking becomes apparent. Ideally, the first node along the lower boundary would move nearer to the discontinuity to further improve the solution. In fact, the movement algorithm attempts to move the node upwards but it is immediately projected back on to the boundary so it effectively remains fixed. There seems to be no simple solution to this problem, but it only becomes significant in regions where surrounding nodes are severely restricted in their movement, such as at corners, and so provides a strong argument against the fixing of nodes at turning points in the solution.

Although the effect on the steady state solution is small here, it does suggest that some form of selective grid refinement is necessary to produce solutions of best possible accuracy. In fact this could be used in most cases simply to introduce nodes in regions where the movement algorithm requires them to obtain high accuracy. For example, in both of the above test cases, solutions of similar quality could be obtained with far fewer nodes in the regions away from the discontinuities.

The results from these simple test problems suggest that the best strategies for moving the nodes are those which involve weighted averaging and are generalisations of the one-dimensional scheme which equidistributed arc length. To give the methods a more stringent test, more complicated flows are needed, and these can be provided by the Euler equations. The first test case shown here is air flow (left to right) with freestream Mach number $M_\infty = 1.4$ through a channel of length 3 and unit height with a circular arc bump of length 1 and height 0.04 in the middle of the lower surface [12]. The solution obtained using multidimensional upwinding (the PSI advection scheme with the Mach angle wave model [4, 5]) on a fixed, regular 2145 node, 4096 cell grid is shown in Figure 4.13. The steady state solution exhibits a distinctive shock pattern which one would expect to be mirrored by the grid after the adaptation has taken place.

The adapted grids - all obtained by running the time-stepping to create a near steady state solution and then carrying out 1000 grid/solution iterations - and the converged solutions produced on them are shown in Figures 4.14-4.19. The solution contours shown are of local Mach number, plotted at intervals of 0.05 with one contour at the freestream Mach number of 1.4, and density has been used to create the monitor surface. In some cases, particularly when the adapted grid is highly distorted, converged solutions were not achieved, but the RMS of the residual was still reduced to below 10^{-12}.

It is immediately apparent that Baines' algorithm and the weighted averaging with $w = |\tan\theta|$ provide the nodes with the greatest incentive to move towards the high gradients. As expected, they are both tending to drag all of the nodes out of the constant flow region in front of the bump, and would do so if the grid were allowed to continue to move. With this more complicated flow structure, the node depletion away from the shocks is more significant because the solution here is not always constant. Immediately behind the shock reflection off the upper boundary, the Mach number reaches a minimum which isn't captured nearly so well on the adapted meshes because the nodes have moved away. (The maximum and minimum values of the local Mach number are given in the captions to the figures.) The maximum value is achieved at the upper end of the outflow boundary and suffers in a similar way. The resolution of any flow feature in the regions where the cells have increased in size is bound to suffer and this, together with the high distortion of the mesh, means that these solutions are of limited accuracy.

Of the other two choices of weight, $w = \sec\theta$ gives the greater bias towards high gradients, as can be seen in Figures 4.16 and 4.18, but the two solutions shouldn't be compared until the grid scaling has been taken into account. Figures 4.17 and 4.19 show the results obtained by reducing the grid size by a factor of ten.

It is now apparent that, $w = A\sec\theta$ is probably the better choice since it achieves a similar sharpness of the shock with less node depletion in the nearby regions of the grid. When compared with the result from the same test case on a finer grid (which took nearly 8 times as long to produce), the shocks are less diffuse, and the only region where the solution suffers is on either side of the shocks where nodes are drawn from. As a result the ridges created on the solution surface, particularly at their the maxima and minima, are not so well captured - another case for selective grid refinement.

It is also a case for the inclusion of curvature modelling in the adaptive scheme. This would be used to bias the grid movement towards regions of high solution curvature as well as high gradient and would be expected to stop nodes from leaving these turning points of the solution. However, no results using either selective refinement or curvature modelling are presented here.

The last of these weighted averaging schemes, where $w = A \sec \theta$, has also been used with other test cases, the most interesting of which is the flow, with $M_\infty = 3.0$, in a channel of height 1 and length 3 over a forward facing step of height 0.2 at a distance of 0.6 into the channel [11]. The results from this can be seen in Figures 4.21 and 4.22, where the scaling has been chosen to give the best results from the adaptation. The adapted grid nicely mirrors the shock structure of the solution and the shock capturing has correspondingly been improved. This time neither of the solutions shown have converged, but this is due to the multidimensional upwinding being unable to cope with the large subsonic region in front of the step (and the stagnation point).

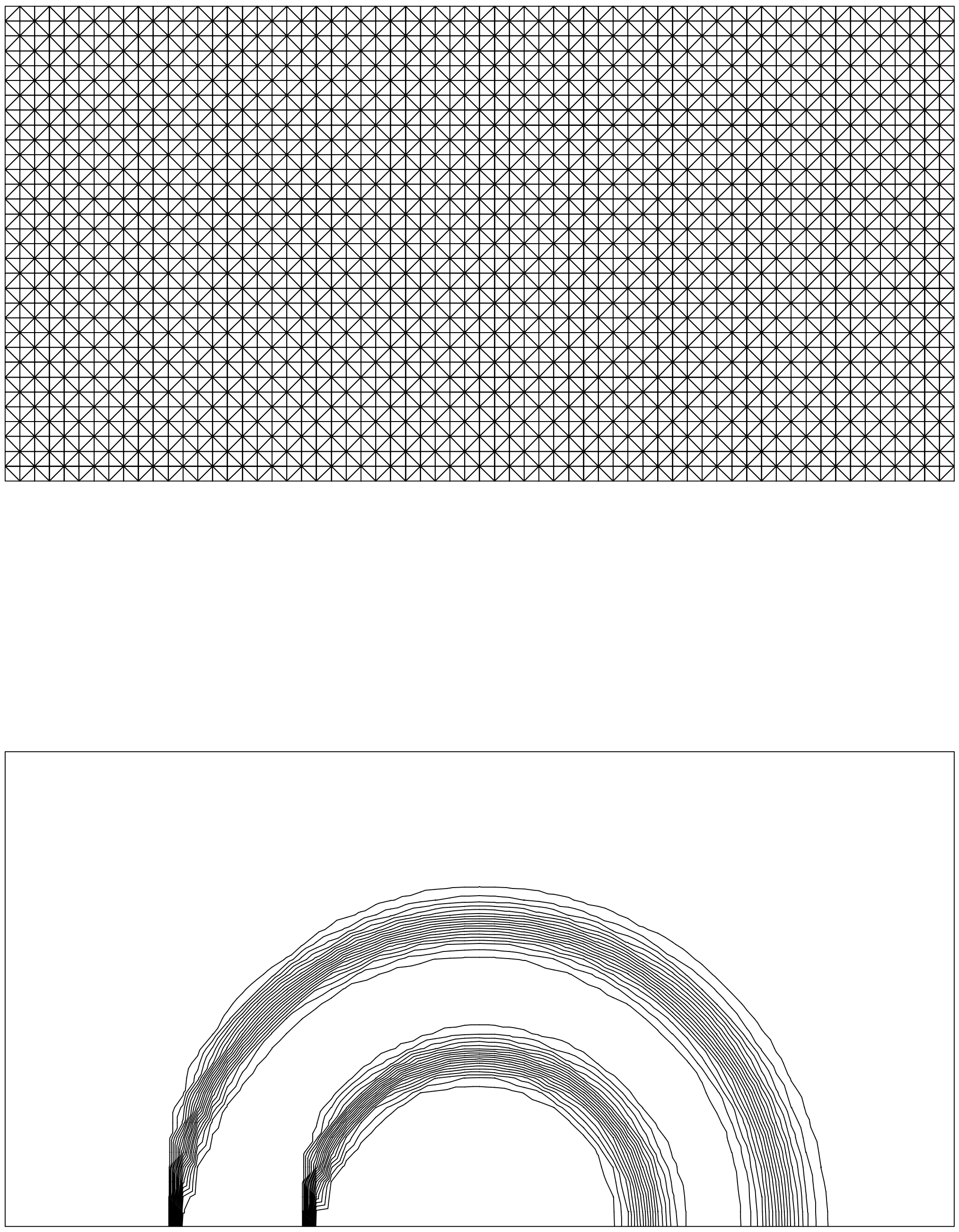
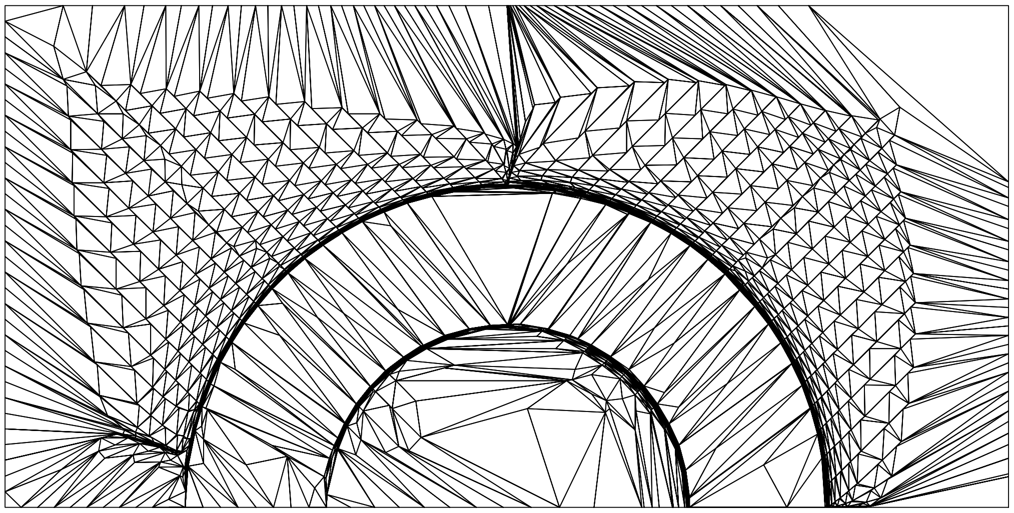


Figure 4.1: The unadapted grid (top) and the converged solution contours on this grid (bottom) for the circular shear test case.



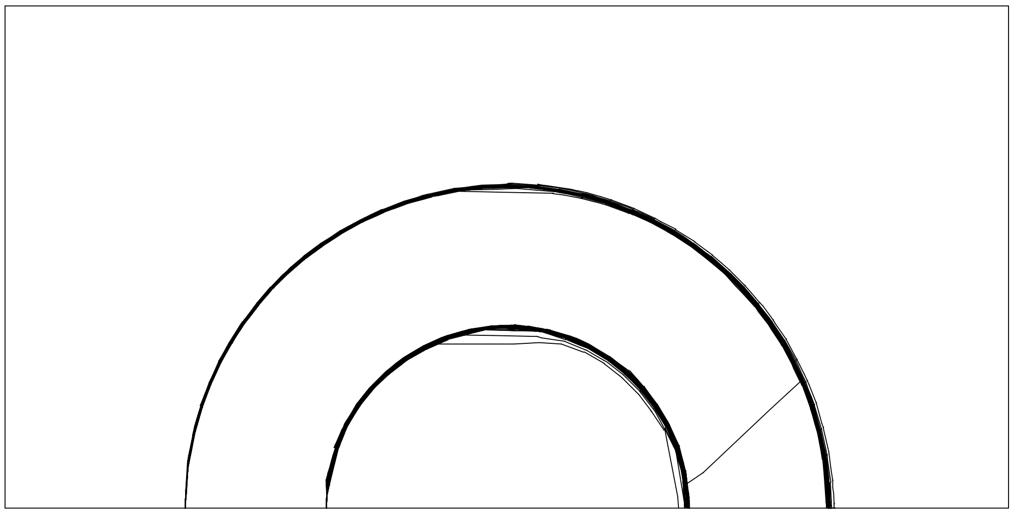


Figure 4.2: The adapted grid using Baines' algorithm (top) and the converged solution contours on this grid (bottom) for the circular shear test case.

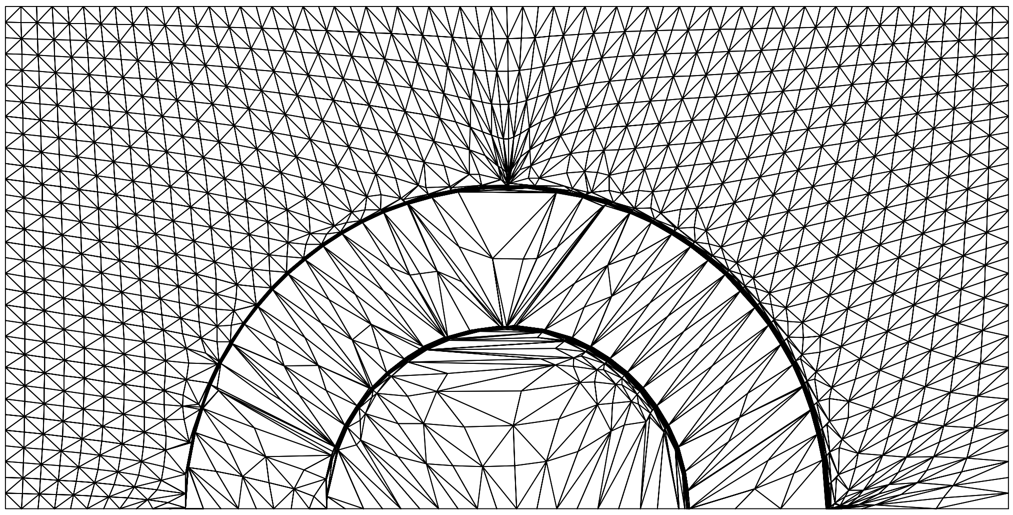
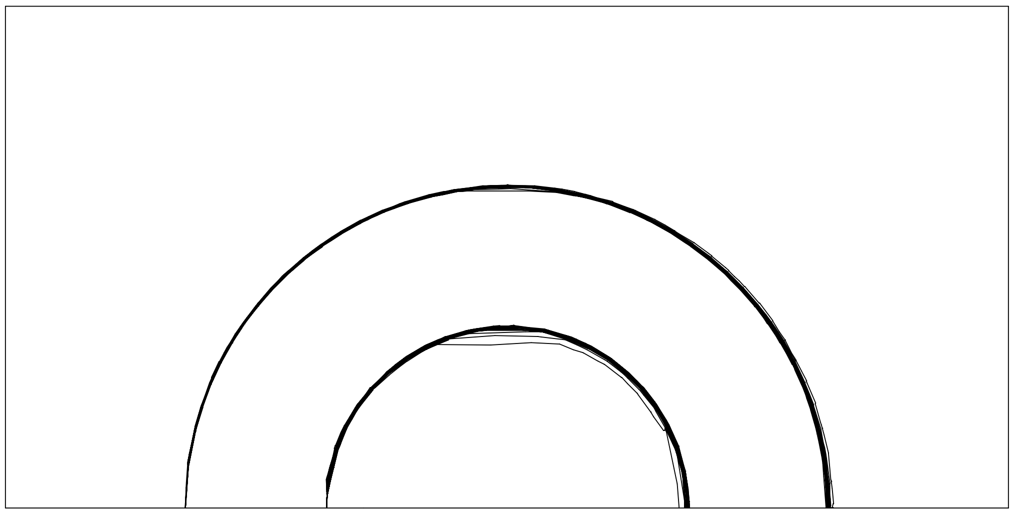



Figure 4.3: The adapted grid using Baines' algorithm with smoothing (top) and the converged solution contours on this grid (bottom) for the circular shear test case.

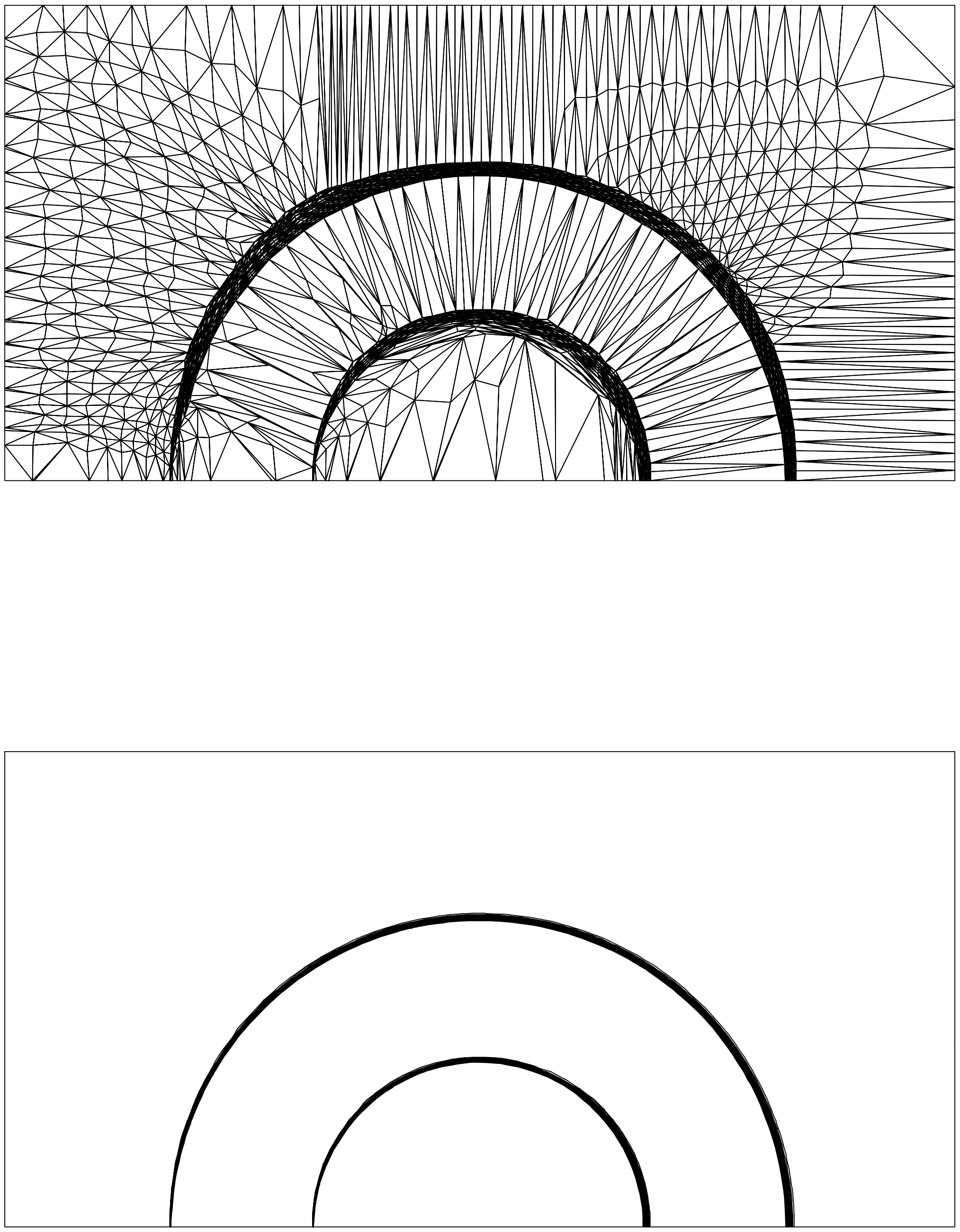


Figure 4.4: The adapted grid using $w = |\tan\theta|$ (top) and the converged solution contours on this grid (bottom) for the circular shear test case.

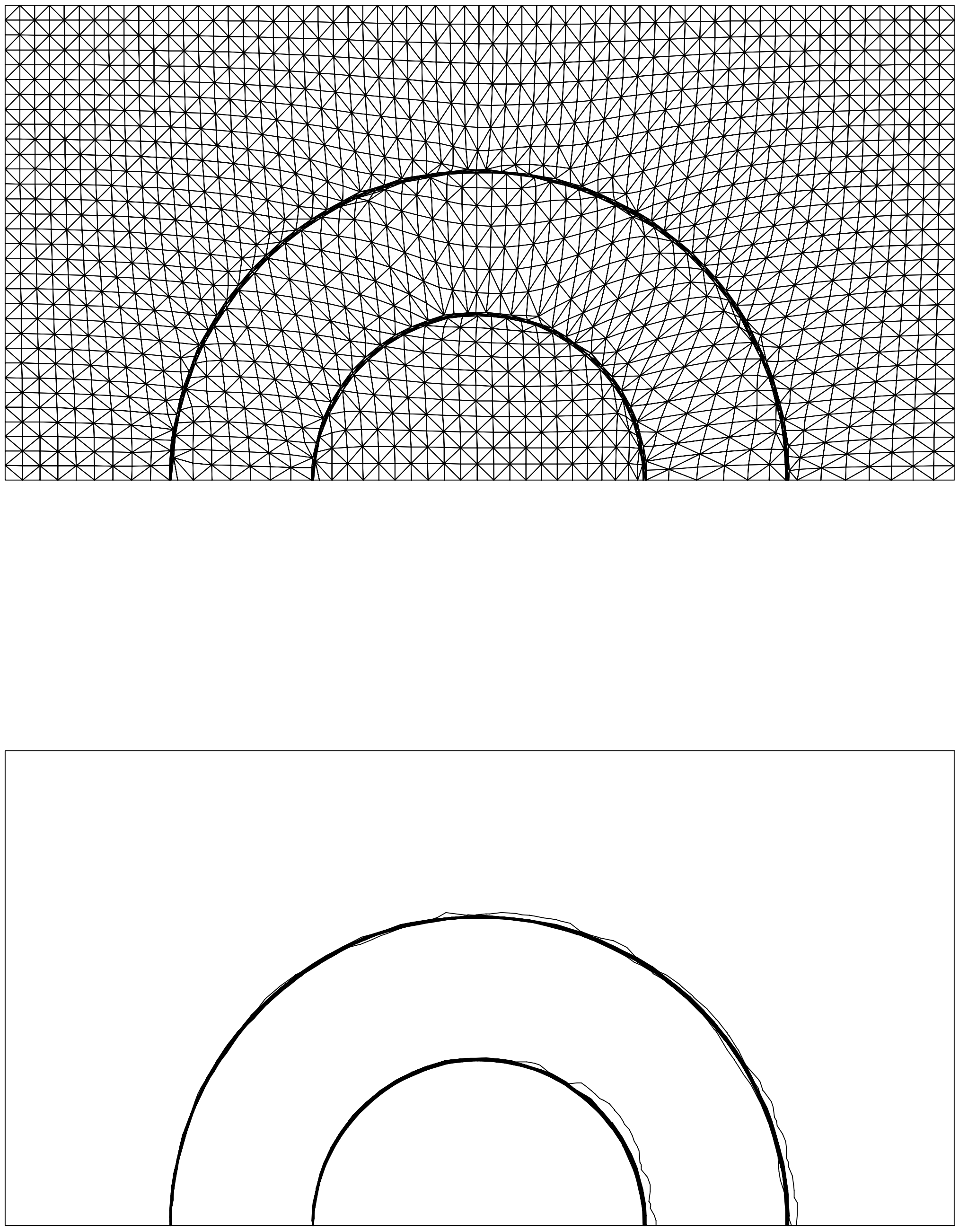


Figure 4.5: The adapted grid using $w = \sec\theta$ (top) and the converged solution contours on this grid (bottom) for the circular shear test case.

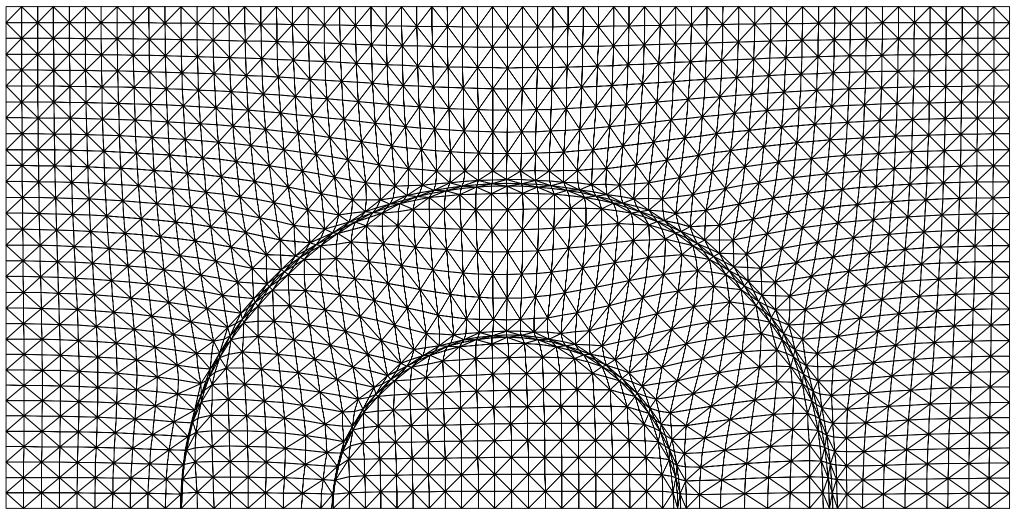
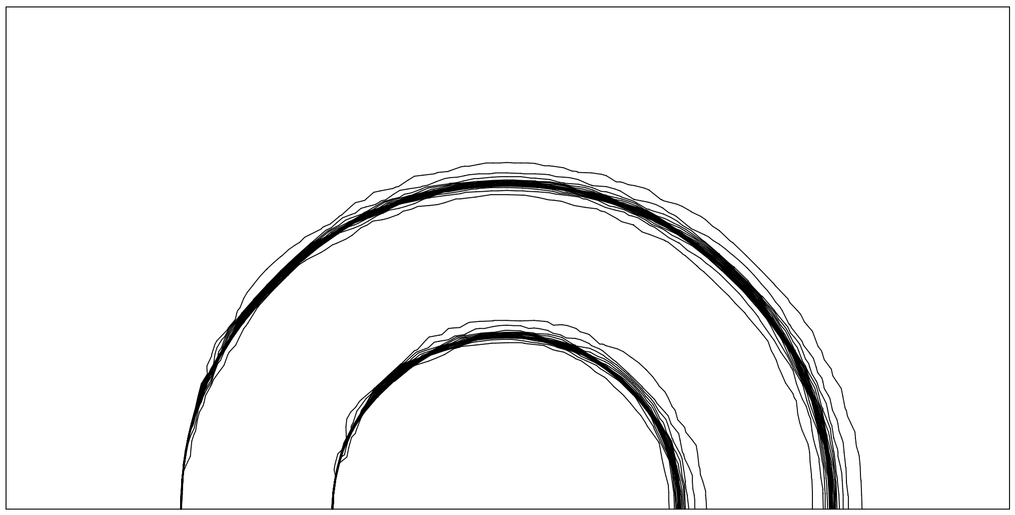
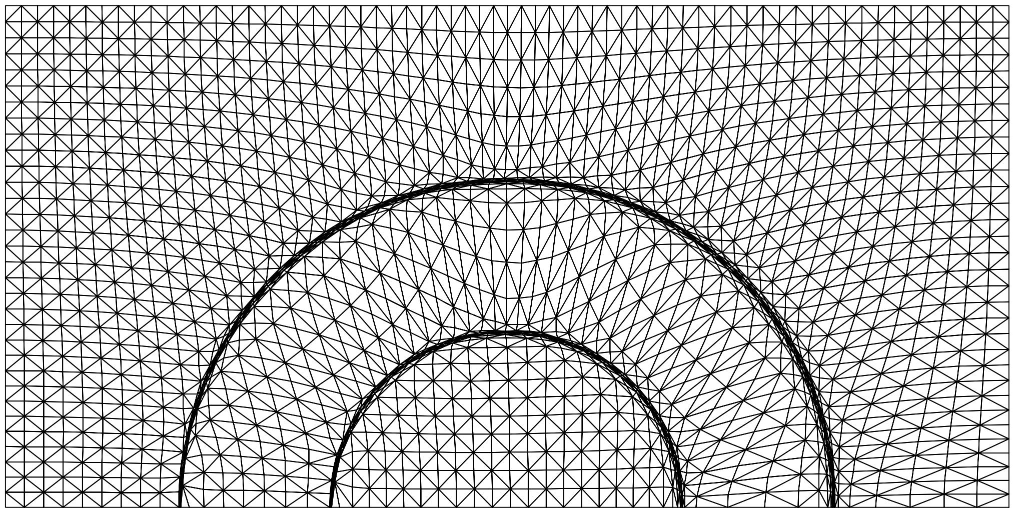



Figure 4.6: The adapted grid using $w = A\sec\theta$ (top) and the converged solution contours on this grid (bottom) for the circular shear test case.



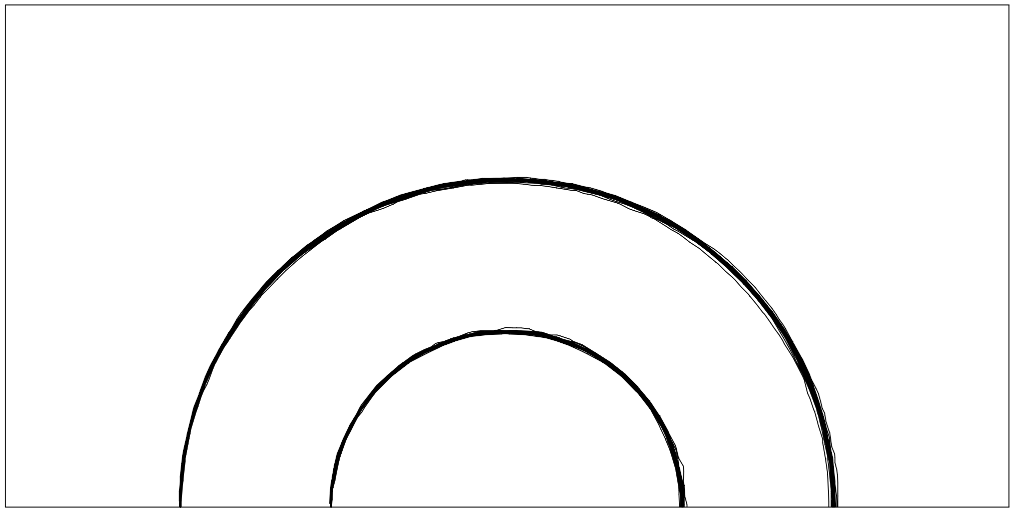


Figure 4.7: The adapted grid using grid scaling and $w = A \sec \theta$ (top) and the converged solution contours on this grid (bottom) for the circular shear test case.

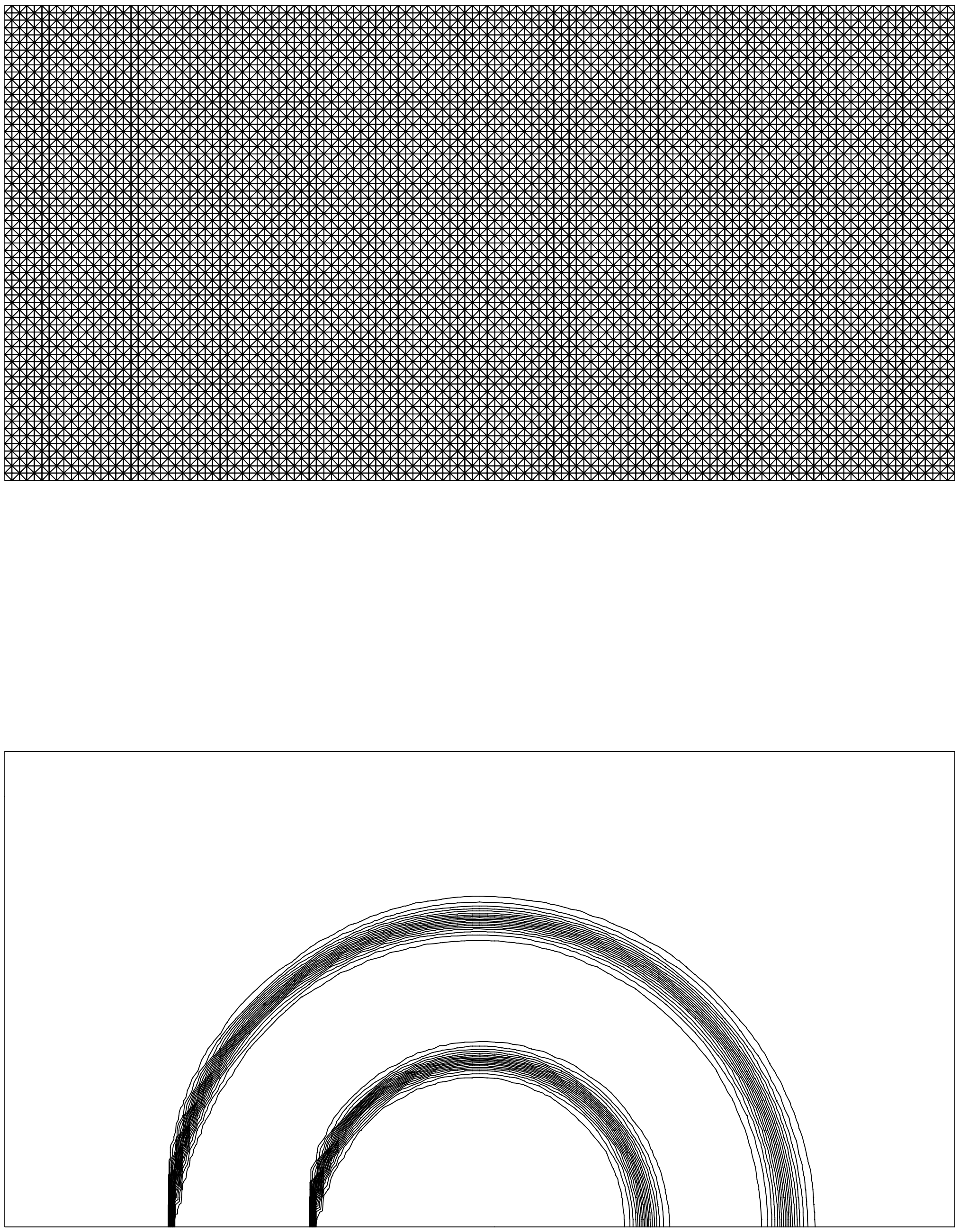
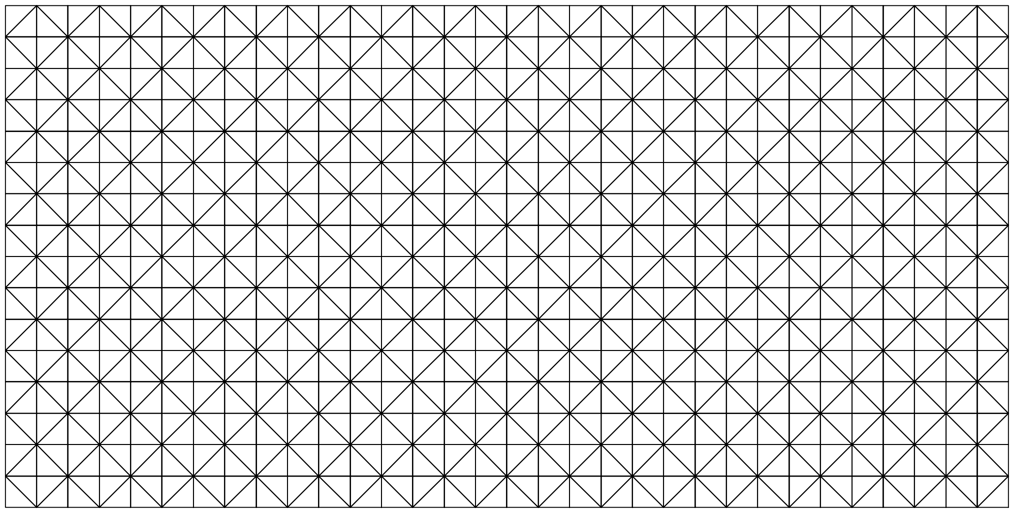


Figure 4.8: The unadapted fine grid (top) and the converged solution contours on this grid (bottom) for the circular shear test case.



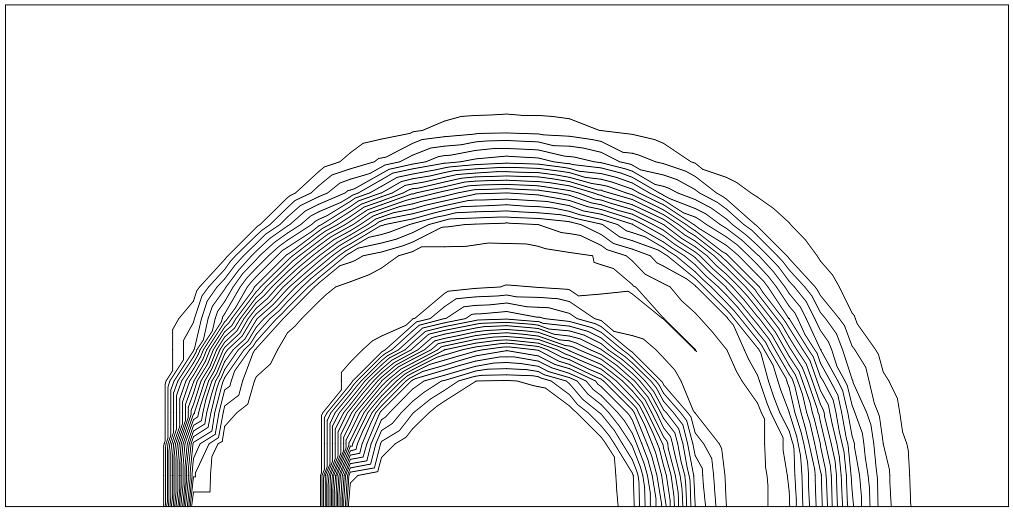
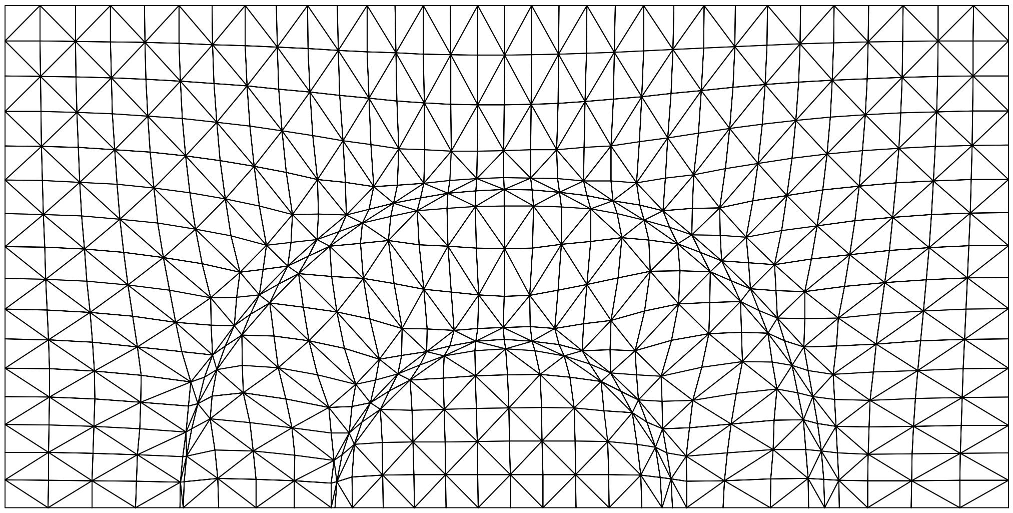


Figure 4.9: The unadapted coarse grid (top) and the converged solution contours on this grid (bottom) for the circular shear test case.



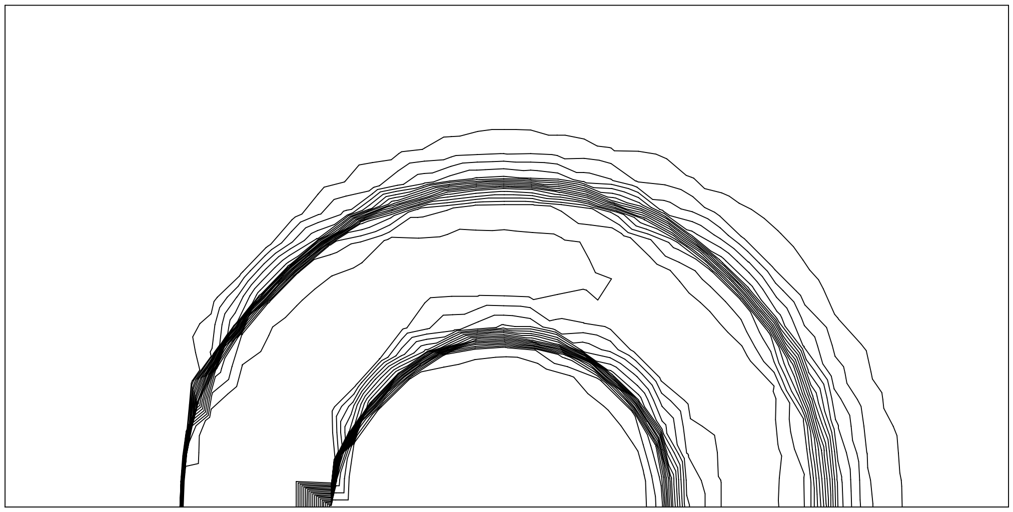
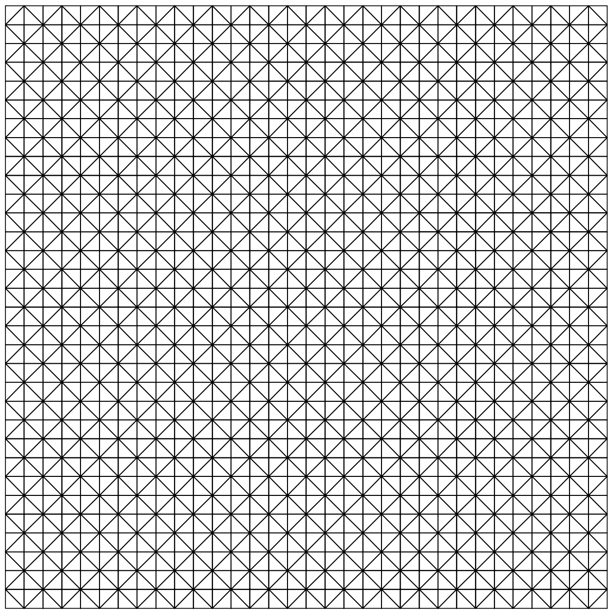


Figure 4.10: The adapted coarse grid using grid scaling and $w = A \sec\theta$ (top) and the converged solution contours on this grid (bottom) for the circular shear test case.



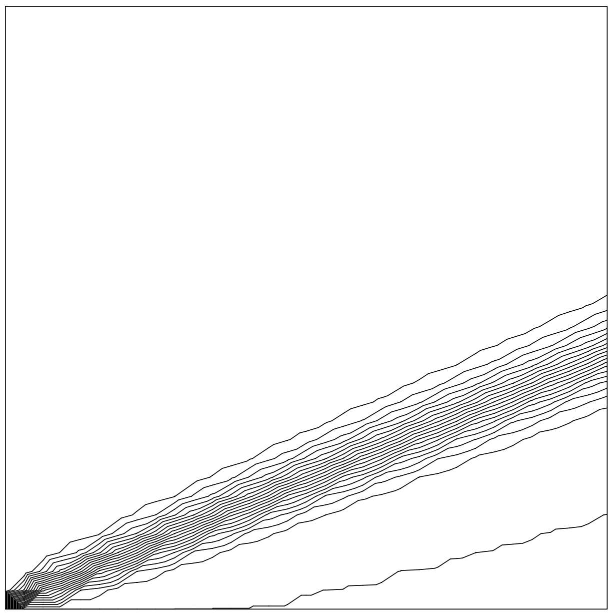
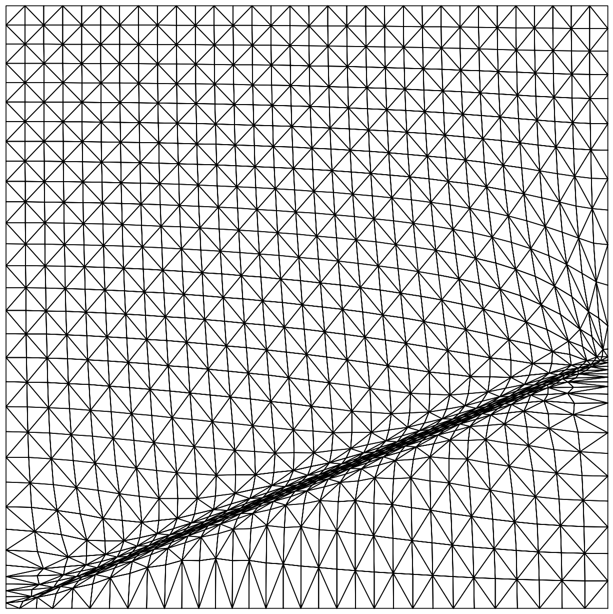


Figure 4.11: The unadapted grid (top) and the converged solution contours on this grid (bottom) for the linear shear test case.



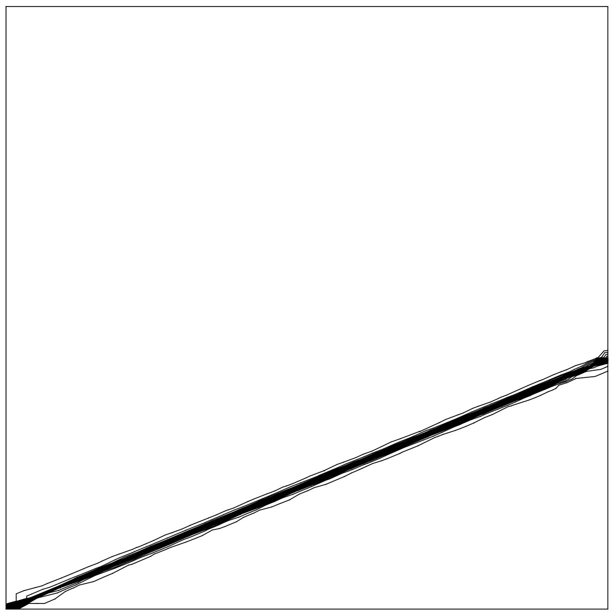
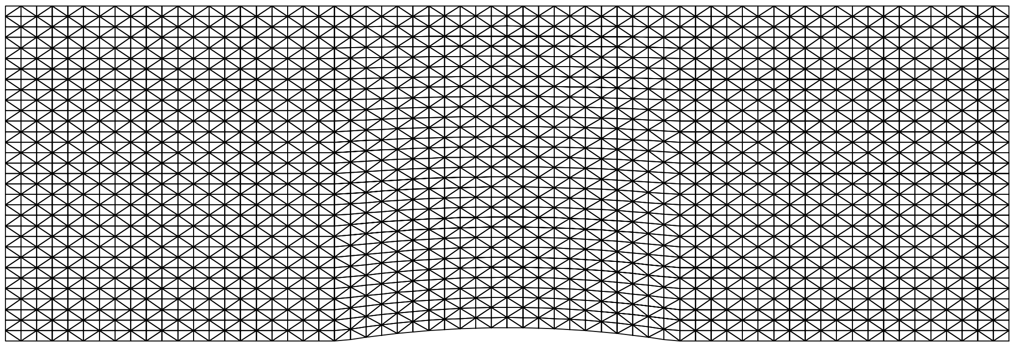


Figure 4.12: The adapted grid using $w = A \sec \theta$ (top) and the converged solution contours on this grid (bottom) for the linear shear test case.



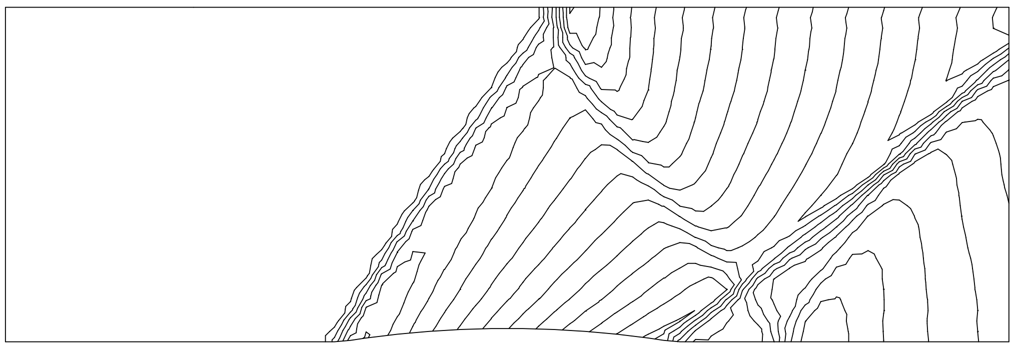
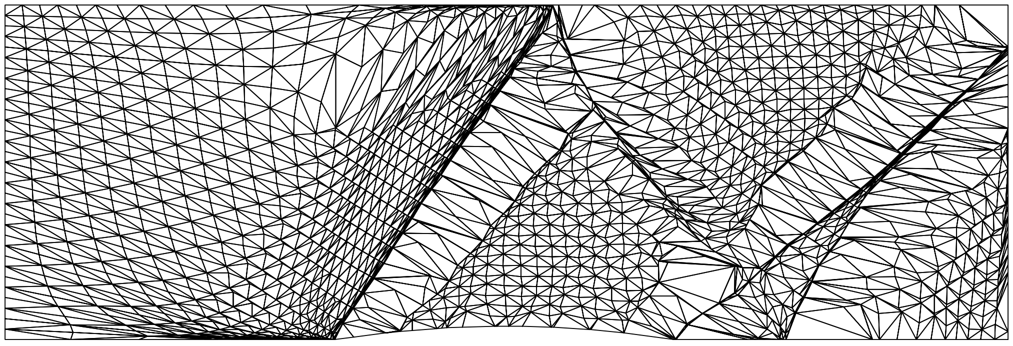


Figure 4.13: The unadapted grid (top) and the local Mach number contours of the converged solution on this grid (bottom) for a 4% circular arc bump in a channel - $M_\infty = 1.4$, $M_{min} = 0.946696$, $M_{max} = 1.67190$.



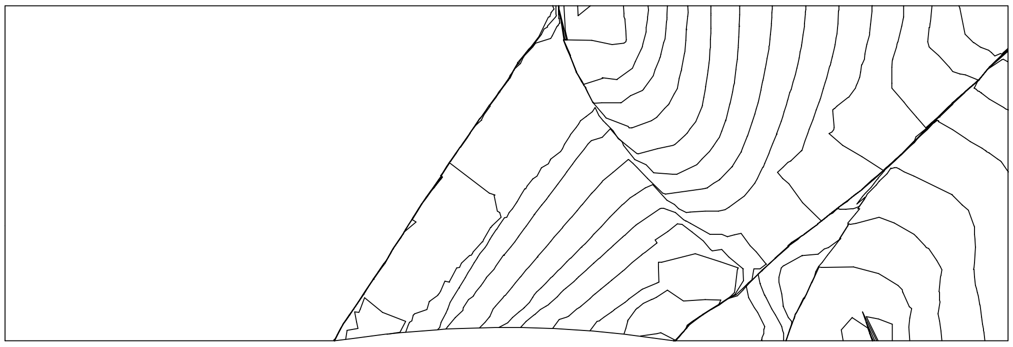
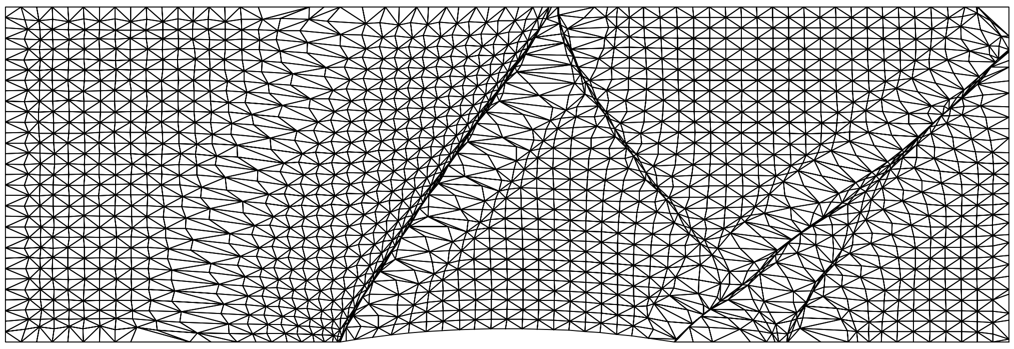


Figure 4.14: The adapted grid using Baines' algorithm with smoothing (top) and the local Mach number contours of the converged solution on this grid (bottom) for a 4% circular arc bump in a channel - $M_\infty = 1.4$, $M_{min} = 0.982458$, $M_{max} = 1.67612$.



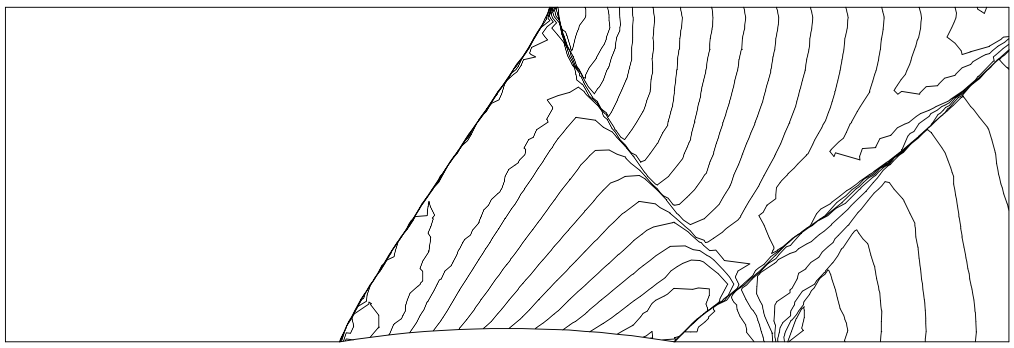
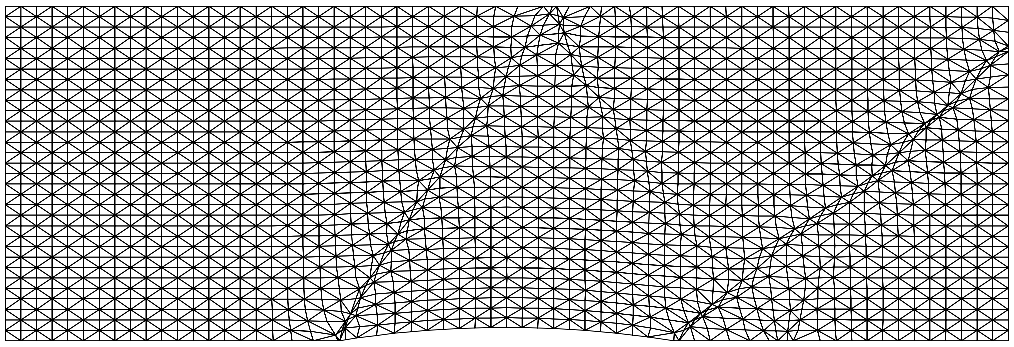


Figure 4.15: The adapted grid using $w = |\tan\theta|$ (top) and the local Mach number contours of the converged solution on this grid (bottom) for a 4% circular arc bump in a channel - $M_\infty = 1.4$, $M_{min} = 0.927394$, $M_{max} = 1.65941$.



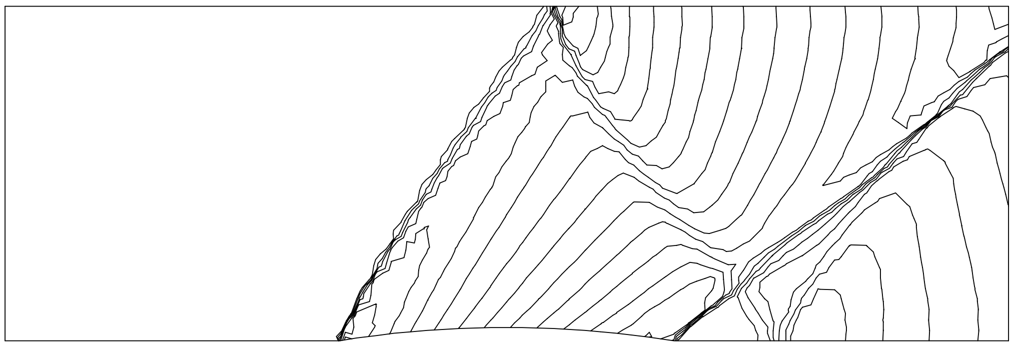
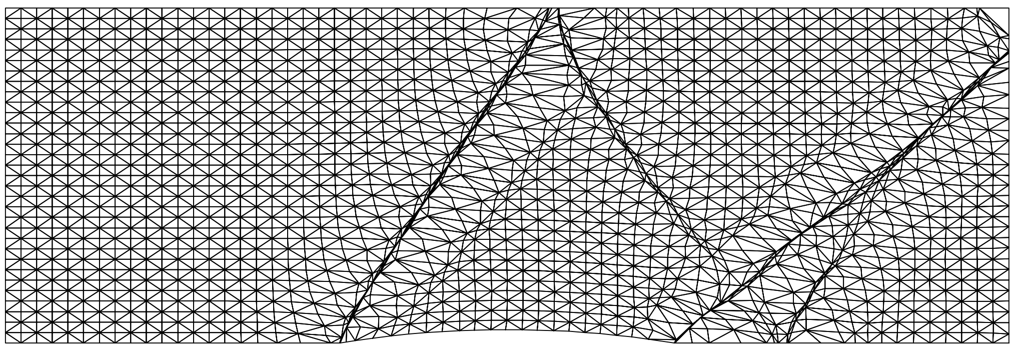


Figure 4.16: The adapted grid using $w = \sec\theta$ (top) and the local Mach number contours of the converged solution on this grid (bottom) for a 4% circular arc bump in a channel - $M_\infty = 1.4$, $M_{min} = 0.934373$, $M_{max} = 1.67046$.



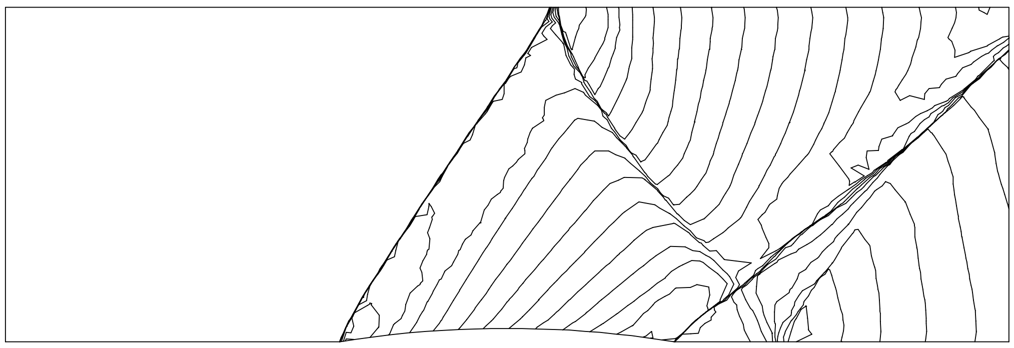
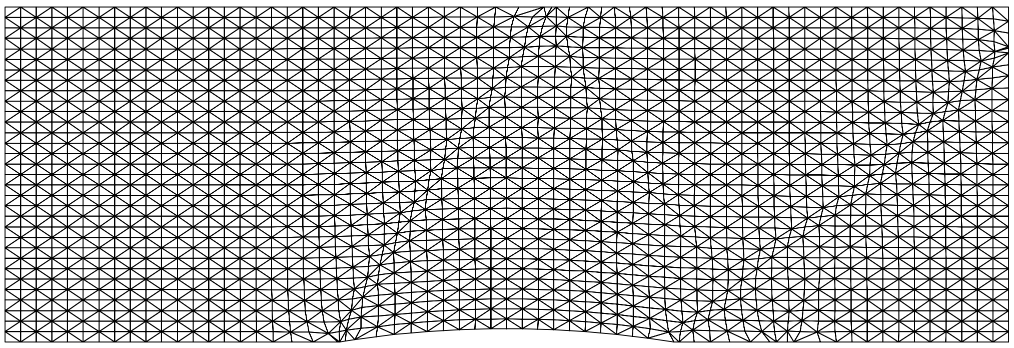


Figure 4.17: The adapted grid using $w = \sec\theta$ and grid scaling (top) and the local Mach number contours of the converged solution on this grid (bottom) for a 4% circular arc bump in a channel - $M_\infty = 1.4$, $M_{min} = 0.925287$, $M_{max} = 1.65952$.



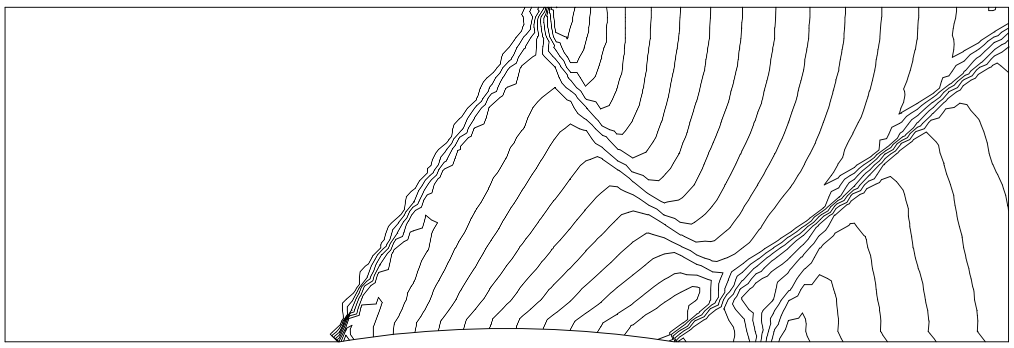
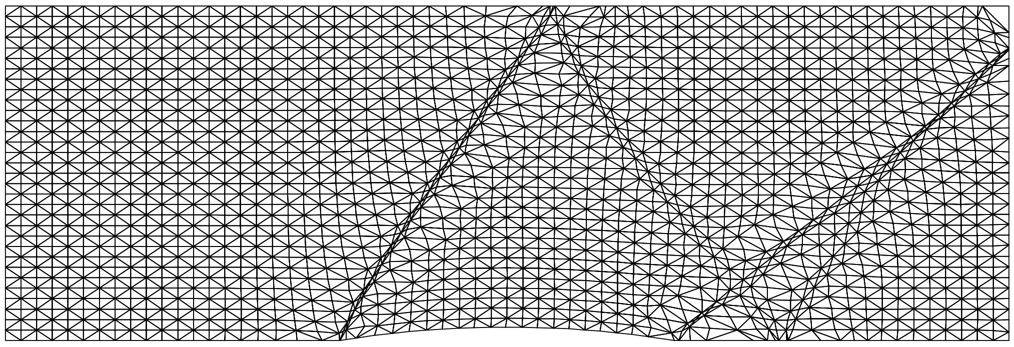


Figure 4.18: The adapted grid using $w = A \sec\theta$ (top) and the local Mach number contours of the converged solution on this grid (bottom) for a 4% circular arc bump in a channel - $M_\infty = 1.4$, $M_{min} = 0.899419$, $M_{max} = 1.65642$.



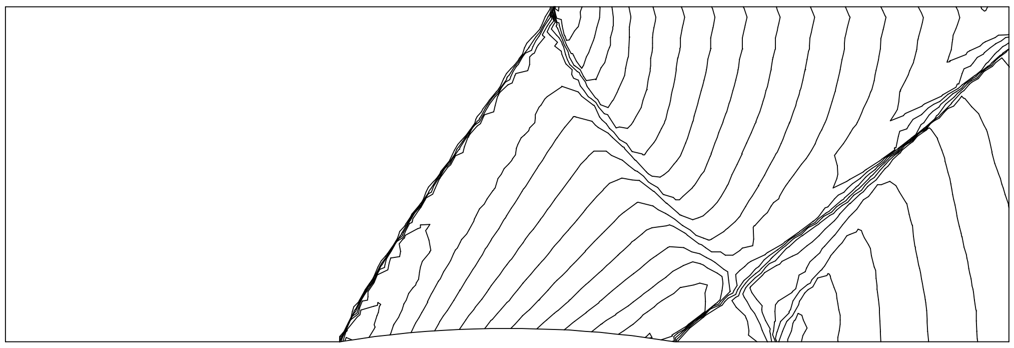


Figure 4.19: The adapted grid using $w = A \sec\theta$ and grid scaling (top) and the local Mach number contours of the converged solution on this grid (bottom) for a 4% circular arc bump in a channel - $M_\infty = 1.4$, $M_{min} = 0.894084$, $M_{max} = 1.65589$.

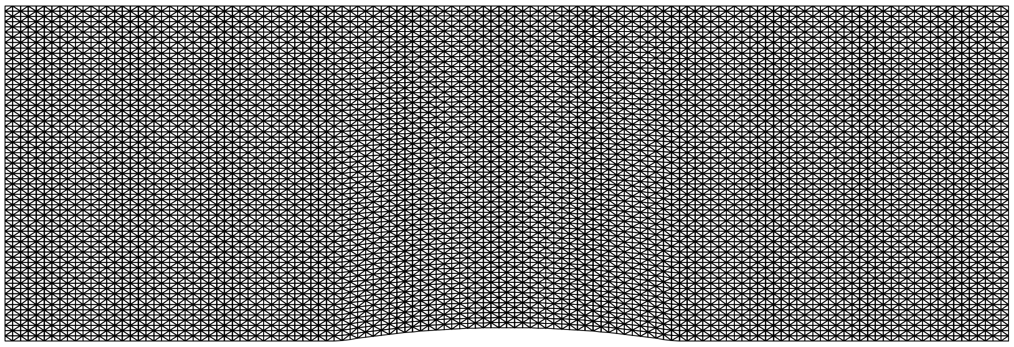
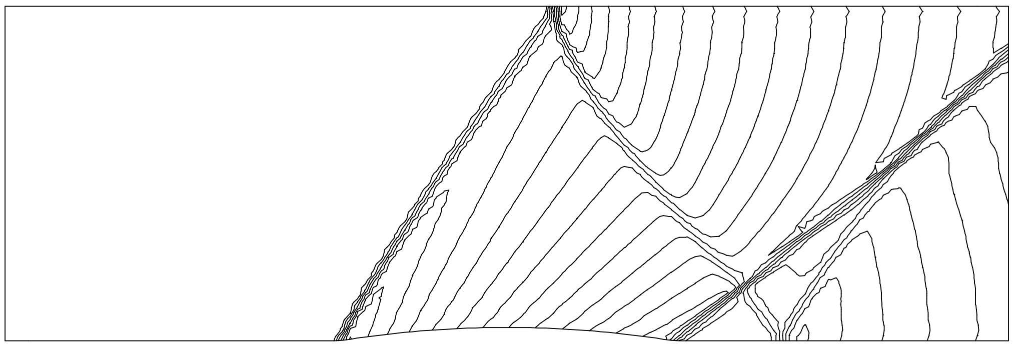
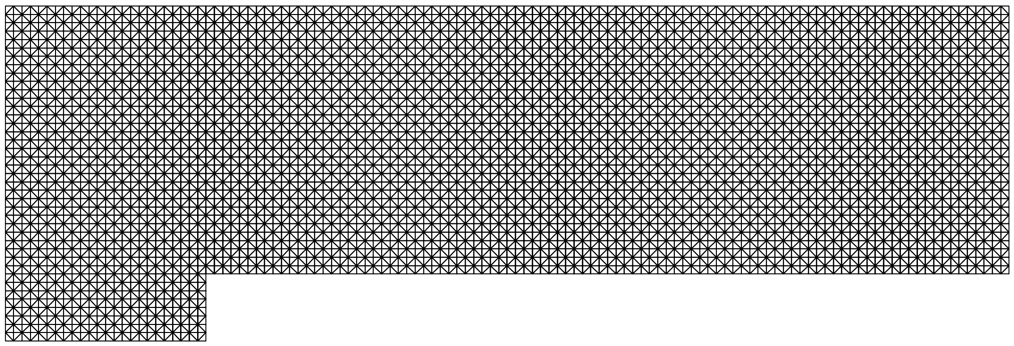



Figure 4.20: The unadapted fine grid (top) and the local Mach number contours of the converged solution on this grid (bottom) for a 4% circular arc bump in a channel - $M_\infty = 1.4$, $M_{min} = 0.886538$, $M_{max} = 1.68859$.



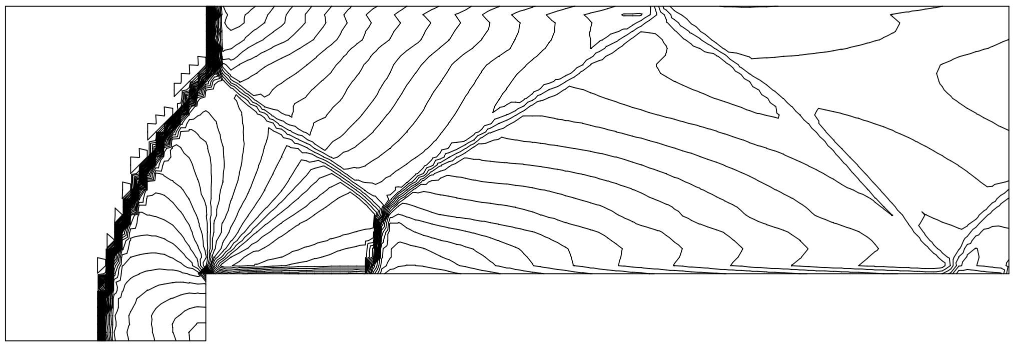
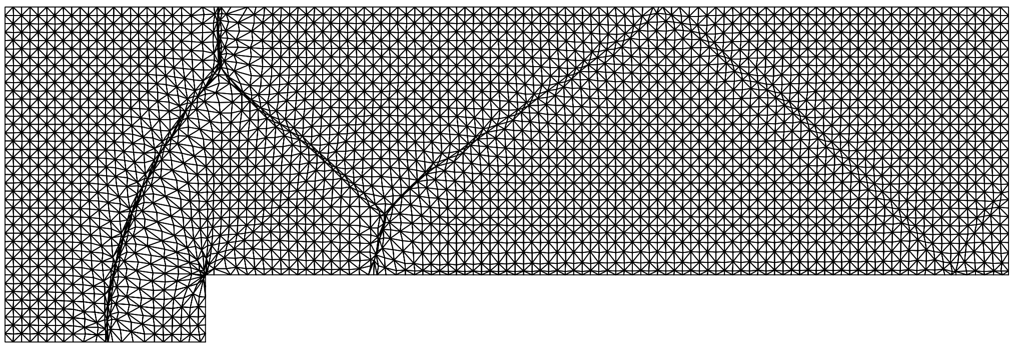


Figure 4.21: The unadapted grid (top) and the local Mach number contours of the solution on this grid (bottom) for a 20% step in a channel - $M_\infty = 3.0$, $M_{min} = 0.0$, $M_{max} = 3.00140$.



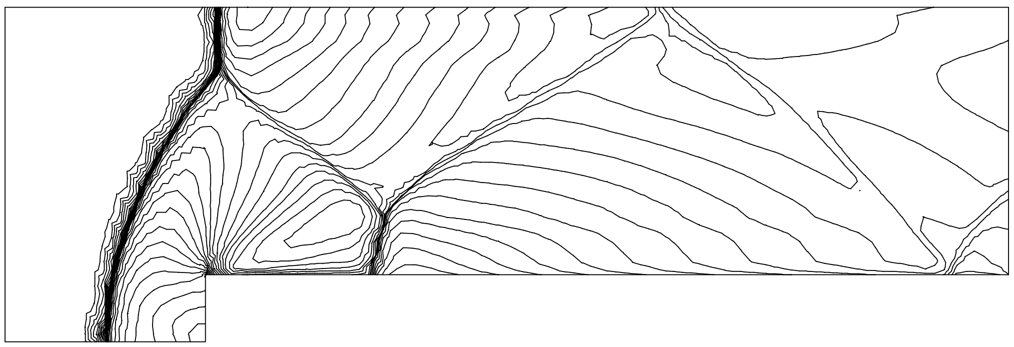


Figure 4.22: The adapted grid using $w = A \sec\theta$ (top) and the local Mach number contours of the solution on this grid (bottom) for a 20% step in a channel - $M_\infty = 3.0$, $M_{min} = 0.0$, $M_{max} = 3.0$.

5 Conclusions

The simple and very cheap algorithms described in this report can be used to significantly improve the quality of the solution by not only clustering nodes in regions where the solution gradient is high, but also aligning the cell edges with features such as discontinuities and with negligible increase in effort. All of this has been achieved without any change in the connectivity of the grid or increasing the number of cells, although it is almost certain that the optimal grid adaptation strategy will include selective refinement and edge swapping as well as movement. Even so, the current method can already be used to gain great improvements to steady state solutions of the linear advection and the Euler equations, without much increase in the expense.

Further improvement in the grid movement methods should be made possible by increasing the sophistication of these schemes, either by choosing the direction of the displacement in a more intelligent manner or the inclusion of solution curvature in the monitoring procedure, although it is not yet clear how best to do this. It would also be preferable to automate the switching on and off of the movement in the overall solution strategy and to find a less *ad hoc* method of deciding the mesh scaling.

Acknowledgements

M.E. Hubbard is supported financially by a contract funded by DRA Farnborough. He wishes to thank Prof. M.J. Baines of the Mathematics Department, University of Reading his advice and assistance during this work.

References

[1] G.Erlebacher and P.R.Eiseman. Adaptive triangular mesh generation. *AIAA Journal*, 25:1356–1364, 1987.

[2] H.Deconinck. Analysis of wave propagation properties for the Euler equations in two space dimensions. In *Computational Fluid Dynamics*, number 1994-05 in VKI Lecture Series, 1994.

[3] H.Deconinck, R.Struijs, G.Bourgois, H.Paillere, and P.L.Roe. Multidimensional upwind methods for unstructured grids. In *Unstructured Grid Methods for Advection Dominated Flows*, May 1992. AGARD Report 787.

[4] H.Deconinck, R.Struijs, G.Bourgois, and P.L.Roe. High resolution shock capturing cell vertex advection schemes for unstructured grids. In *Computational Fluid Dynamics*, number 1994-05 in VKI Lecture Series, 1994.

[5] H.Paillere, J-C.Carette, and H.Deconinck. Multidimensional upwind and SUPG methods for the solution of the compressible flow equations on unstructured grids. In *Computational Fluid Dynamics*, number 1994-05 in VKI Lecture Series, 1994.

[6] M.A.Rudgyard. Multidimensional wave decompositions for the Euler equations. In *Computational Fluid Dynamics*, number 1993-04 in VKI Lecture Series, 1993.

[7] M.J.Baines. On algorithms for best l_2 fits to continuous functions with variable nodes. Report 1/93, Department of Mathematics, University of Reading, 1993.

[8] P.L.Roe. Multidimensional upwinding: Motivation and concepts. In *Computational Fluid Dynamics*, number 1994-05 in VKI Lecture Series, 1994.

[9] P.R.Eiseman. Grid generation for fluid mechanics computations. *Ann. Rev. Fluid Mech.*, 17:487–522, 1985.

[10] P.R.Eiseman and G.Erlebacher. Grid generation for the solution of partial differential equations. Report 87-57, ICASE, 1987.

[11] P.Woodward and P.Colella. The numerical simulation of two-dimensional fluid flow with strong shocks. *J. Comp. Phys.*, 54(1):115–173, April 1984.

[12] R.H.Ni. A multiple grid scheme for solving the Euler equations. *AIAA Journal*, 20(11):1565–1571, 1982.