THE UNIVERSITY OF READING

# Numerical Experiments on the Solution of a Scalar Advection Equation by Least Squares on Optimal Grids

by

S.J. Leary

Numerical Analysis Report 9/97

DEPARTMENT OF MATHEMATICS

# Numerical Experiments on the Solution of a Scalar Advection Equation by Least Squares on Optimal Grids.

S. J. Leary

Numerical Analysis Report 9/97

# 1 Introduction

A moving grid method is described in [1] which attempts to solve advection on optimal grids by minimising a least squares functional using a steepest descent procedure.

Using scalar advection in 2-D, we define a fluctuation and then introduce a functional that we wish to minimise which depends on this fluctuation. Differentiation of this functional with respect to the solution values $u$ and the gridpoints $\mathbf{x}$ gives rise to a steepest descent update which we use to find new values of $u$ and $\mathbf{x}$.

This method is slow to converge and we consider ways to speed up the rate of convergence.

We note that the variational equation for the solution $u$ can be written in the form of a matrix equation and when solved also gives updates which appear to be better than those obtained from steepest descent. This is easy to implement in the case of solution updates, but appears more difficult when we consider the grid.

We can also take an upwinding approach to the method described in [1] where the updates for upwind nodes in each triangle are set to zero and this also improves the rate of convergence.

We also attempt to optimise the relaxation factor in the steepest descent method in order to make this method as efficient as possible. Other ideas for improving the rate of convergence are briefly discussed.

In section 2 we introduce some notation, then in sections 3 and 4 we consider the cases of constant advection and circular advection respectively, together with ideas to speed up the rate of convergence. Some numerical results are given in section 5.

## 2 Notation

Before looking at the problem, we introduce the following notation.

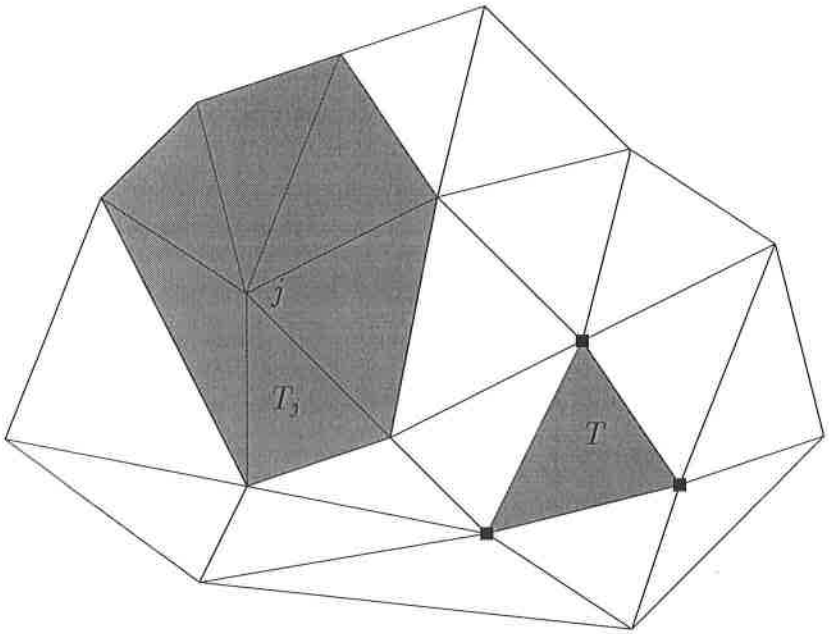


Figure 1: An arbitrary triangulation.

The triangulation in figure 1 shows part of an unstructured grid. A typical node is denoted by $j$ and $\{T_j\}$ is the set of triangles (shaded) surrounding that node. A typical triangle is denoted by $T$ and $\{j_T\}$ is the set of points (marked with squares) forming its vertices.

## 3 Scalar Advection

Approximate solutions of the two-dimensional equation

$$\mathbf{a}.\nabla u = 0 \tag{1}$$

on an unstructured grid have been considered in [1]. The function $u$ is approximated by a piecewise linear function. The fluctuation is defined by

$$\begin{aligned} \phi_T &= -\int\int_T \mathbf{a}.\nabla u dx dy \\ &= -\left(\int\int_T \mathbf{a} dx dy\right).\nabla u \\ &= -(\bar{\mathbf{a}}.\nabla u)\, S_T \end{aligned} \tag{2}$$

where the advection speed $\bar{\mathbf{a}}$ is a constant averaged value in each cell. Then we may write [1]

$$\phi_T = -\sum_{j \in j_T} k_j u_j. \tag{3}$$

Here $u_j$ is the value of $u$ at vertex $j$ and $k_{j_T}$ is given by

$$k_{j_T} = \frac{1}{2}\bar{\mathbf{a}}.\mathbf{n}_{j_T} \tag{4}$$

where $\mathbf{n}_{j_T}$ is the scaled (by length of edge) inward normal to the side $T$ opposite vertex $j$.

We minimise the least squares functional

$$F = \frac{1}{2}\int\int_{\Omega}(\bar{\mathbf{a}}.\nabla u)^2 dxdy. \tag{5}$$

From (2)

$$\bar{\mathbf{a}}.\nabla u = -\frac{\phi_T}{S_T}, \tag{6}$$

where $S_T$ is the area of triangle $T$ given by

$$S_T = \frac{1}{2}(x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)) \tag{7}$$

so that (5) becomes

$$F = \frac{1}{2}\sum_{T \in \{T\}} \frac{{\phi_T}^2}{S_T} \tag{8}$$

as in [1].

Allowing the minimisation to be carried out over both the solution $u$ and the grid coordinates $\mathbf{x} = (x, y)$, using the steepest descent method gives an update at each iteration of the form

$$\delta \mathbf{u_j} = -\frac{\partial F}{\partial \mathbf{u_j}}\delta\tau \tag{9}$$

at the point $j$, where $\mathbf{u_j} = (x_j, y_j, u_j)$ and $\delta\tau$ is some relaxation factor. The derivative in (9) is

$$\frac{\partial F}{\partial \mathbf{u_j}} = \frac{1}{2}\sum_{T \in \{T_j\}}\left(\frac{\phi_T}{S_T}\frac{\partial \phi_T}{\partial \mathbf{u_j}} - \frac{{\phi_T}^2}{2{S_T}^2}\frac{\partial S_T}{\partial \mathbf{u_j}}\right). \tag{10}$$

From (9) the updates to node $j$ are given by

$$\left.\begin{array}{rcl} {u_j}^{new} & = & {u_j}^{old} + \tau_u \delta u_j \\ {\mathbf{x_j}}^{new} & = & {\mathbf{x_j}}^{old} + \tau_{\mathbf{x}} \delta \mathbf{x} \end{array}\right\} \tag{11}$$

where $\tau_u, \tau_{\mathbf{x}}$ are relaxation factors chosen small enough to ensure reduction in $F$, and which are to be specified in the examples. We refer to (11) as the standard method.

## 3.1 Matrix Representation for $u$

Instead of solving for $u$ by steepest descent, we can construct the variational equation and solve it directly. That is, solve $\frac{\partial F}{\partial u_j} = 0$. From (10) this leads to

$$\frac{\phi_T}{S_T}\frac{\partial \phi_T}{\partial u_j} - \frac{{\phi_T}^2}{2{S_T}^2}\frac{\partial S_T}{\partial u_j} = b \tag{12}$$

where $b$ comes from the boundary conditions and is zero at all points except boundary points. If $\bar{\mathbf{a}} = (\bar{a}, \bar{b})$ is constant in each triangle this equation is locally of the form

$$-\frac{1}{S_T}\begin{bmatrix} \alpha^2 & \alpha\beta & \alpha\gamma \\ \beta\alpha & \beta^2 & \beta\gamma \\ \gamma\alpha & \gamma\beta & \gamma^2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}, \tag{13}$$

where

$$\begin{aligned} \alpha &= \bar{a}(y_2 - y_3) - \bar{b}(x_2 - x_3) \\ \beta &= \bar{a}(y_3 - y_1) - \bar{b}(x_3 - x_1) \\ \gamma &= \bar{a}(y_1 - y_2) - \bar{b}(x_1 - x_2). \end{aligned}$$

We overwrite nodal values which correspond to boundary nodes. The system is already linear in $u_i$ but nonlinear in $x_i$ and $y_i$ $(i = 1, 2, 3)$. We obtain a linear equation to solve for $u$ if we freeze the $x_i, y_i$, that is, treat them as constants and solve the system for the unknowns $u_i$. The local matrix on the left hand side of equation (13) is labelled $A_T$.

We can construct the global system

$$\mathbf{Au} = \mathbf{b} \tag{14}$$

by assembling a global matrix from the local matrices $A_T$ (cf. assembly of matrices in the FEM). We can also build up a local system from the cells sharing node $j$. Here we form a local matrix by assembling the contributions of element matrices from the patch $T_j$ surrounding this node. Since we are only interested in updating $u_j$, this is the only unknown, so we end up with only a single equation to solve.

The second of these methods is used in the numerical experiments in section 5 as it is computationally more efficient. Global solutions gave essentially the same results.

When we solve for $u$ using either the global approach with a matrix solver, or by solving the local system, the method converges at approximately twice the rate of the standard method (see section 5.)

### 3.2 Upwinding Approach for $u$

Another way in which we can try to speed up the rate of convergence is to only update the solution $u$ for downwind nodes.

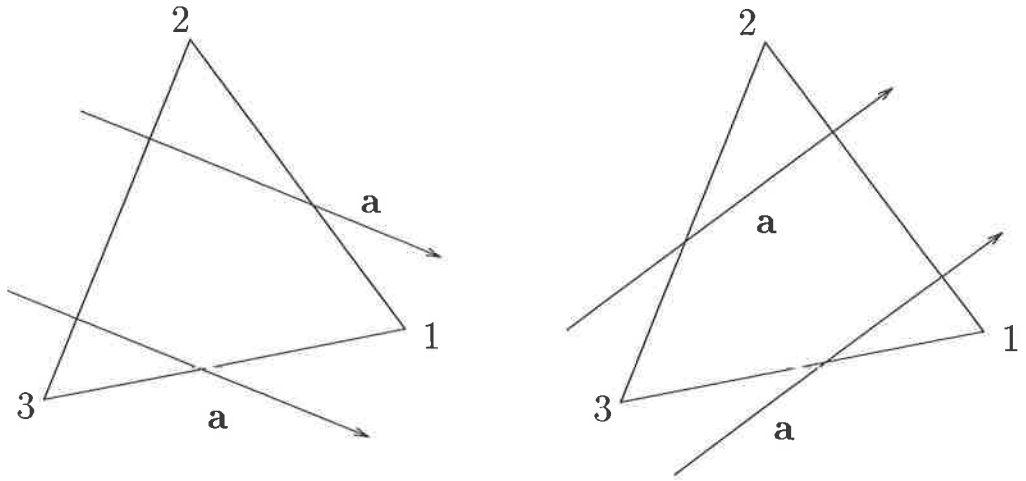


Figure 2: A triangle with one inflow side, $k_1 > 0$, $k_2$, $k_3 < 0$ (left), and one with two inflow sides, $k_1$, $k_2 > 0$, $k_3 < 0$ (right).

Here we consider the 'direction of the wind' $\mathbf{a}$ and $k_i = \frac{1}{2}\mathbf{a}.\mathbf{n}_i$ where $\mathbf{n}_i$ is the unit inward normal to the side opposite vertex $i$. In the examples in figure 2, the downwind node for the left hand example is node 1, in the example on the right hand side, nodes 1 and 2 are downwind nodes. If $k_1 > 0$, $k_2$ , $k_3 < 0$ then we update node 1 only, and if $k_1$, $k_2 > 0$, $k_3 < 0$ then we update nodes 1 and 2 only. Hence in this method we work out the update at each node of a triangle, then overwrite the updates to zero for upwind nodes. Then we assemble the update to each node $j$ from the contributions of each triangle in $T_j$. This again results in an increased rate of convergence . This method does not correspond to the original Least Squares minimisation, however.

# 4 Circular Advection Problem

A slightly more complicated example is the circular advection equation

$$yu_x - xu_y = 0, \tag{15}$$

that is equation (1) where $\mathbf{a} = (y, -x)$. We define the fluctuation as we did before (3) but this time the vector $\mathbf{a}$ in (4) is a suitably averaged value of $(y, -x)$ taken over the sides of the relevant triangle. The suitably averaged values are taken to be averages over the edges of each triangle, that is, if

we are considering the side joining edges 1 and 2, then the averaged value of $(y, -x)$ is $(\frac{1}{2}(y_1+y_2), -\frac{1}{2}(x_1+x_2))$. If we do this we obtain for the fluctuation

$$\phi_T = \frac{1}{4} \sum_{j \in \{j_T\}} u_j \Delta_{j_T}(x^2 + y^2) \tag{16}$$

where $\Delta_{j_T}$ is a difference taken counterclockwise along the edge of triangle $T$ that is opposite vertex $j$.

The steepest descent updates (distribution formulae) to vertex $j$ from triangle $T$ are then

$$\delta u_j = \frac{1}{4} \left( \frac{\phi_T}{S_T} \right) \Delta_{j_T}(x^2 + y^2), \tag{17}$$

$$\delta \mathbf{x}_j = -\frac{1}{2} \left( \frac{\phi_T}{S_T} \right) \mathbf{x}_j \Delta_{j_T} u + \frac{1}{4} \left( \frac{\phi_T}{S_T} \right)^2 \mathbf{n}_{j_T}, \tag{18}$$

where the nodes are again updated as in (11) using a relaxation factor small enough to ensure that $F$ decreases. Once again the drawback with this method is the number of iterations it takes to converge. Some more ways in which we could try to speed it up are described in 4.1 - 4.6.

## 4.1 Accelerated Steepest Descent

The steepest descent procedure can be made second order as follows:
A sequence of points $\mathbf{x_1}, \mathbf{x_2}, \ldots$ is generated as follows:

- Point $\mathbf{x_1}$ is specified as the starting point.
- Then $\mathbf{x_2}$ is obtained from one step of steepest descent from $\mathbf{x_1}$.
- In the general step, if $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}$ have been obtained, we find a point $\mathbf{z}$ by steepest descent from $\mathbf{x_m}$.
- Then $\mathbf{x_{m+1}}$ is taken as the minimum point on the line $\mathbf{x_{m-1}} + t(\mathbf{z} + \mathbf{x_{m-1}})$.

When this method was applied to the above problem, the effect was to speed up the rate of convergence, but not by much. The convergence remained very slow.

## 4.2 Shanks' Method

In a paper by Shanks [3] it is shown that methods which have exponential convergence can be speeded up by the following procedure.

Given approximate solutions $a_{n-1}, a_n, a_{n+1}$ from three consecutive iterations of our procedure, if these three approximations are converging exponentially then from these we can obtain an improved approximation

$$a = \frac{a_{n-1}a_{n+1} - a_n^2}{a_{n-1} + a_{n+1} - 2a_n}. \tag{19}$$

The problem here is that although the cost functional is exponentially decreasing in general, the nodal updates aren't. To apply Shanks' method to our scheme to speed up convergence we would need $\delta u$ and $\delta \mathbf{x}$ to be exponentially decreasing. In general they aren't.

## 4.3 Optimising the relaxation factor

The grid convergence seems to be slow partly due to the relaxation factor. This can be seen by changing the parameter and seeing what effect it has on the speed of convergence. This beckons the question, can we find an optimal relaxation factor within each patch? (We must use a local approach for this to ensure mesh tangling does not occur.) One thing tried was to identify the steepest descent direction and to look at how far the node could move without tangling $(\Delta x)_{\max}$ (see (20)) in the steepest descent direction. Consider the line of length $\frac{1}{2}(\Delta x)_{\max}$ in the steepest descent direction from the node we wish to move. Split this into 9 equal intervals and evaluate the cost functional in the patch at the 10 points. Then move the node to whichever of these points gives the minimum value of $F$. The relaxation factor for $u$ is then chosen which minimises the cost functional on 10 equally spaced points in $[0, 0.5]$. This has the effect of finding an optimal relaxation factor and making the cost functional smaller in each patch. It was hoped that this would speed up convergence of the overall problem.

## 4.4 Matrix equations for $u$ and $\mathbf{x}$

Convergence can be speeded up as before by solving for the solution $u$ using the variational equation that arises from (12). This involves the local assembly of element matrices of the same form as (13) . However, this time

$$\begin{aligned} \alpha &= (x_2^2 + y_2^2 - x_3^2 - y_3^2) \\ \beta &= (x_3^2 + y_3^2 - x_1^2 - y_1^2) \\ \gamma &= (x_1^2 + y_1^2 - x_2^2 - y_2^2). \end{aligned}$$

See section 3.1 for further details on the way we formulate and solve the matrix equation arising from (12).

Again this method converges at approximately twice the rate of the standard method. One question is can we do the same thing to $\mathbf{x}$? If we

try to obtain a global matrix using the same ideas as before, the diagonal entries are always of the form $f(x_i, y_i)(u_1 - u_2)$, where $f$ is some function, for $i = 1, 2, 3$ and where $u_1$ and $u_2$ are replaced by their cyclic counterparts. Since $u_1 = u_2$ almost everywhere initially the matrix is highly singular and the system cannot be solved.

Rather than look at the global matrix equation, we could look at all the equations locally again. Where the solution is constant anywhere in a patch the local matrix is singular and we have problems, but these nodes are the nodes we don't want to update anyway. So we could just ignore these nodes and locally update the ones where the local matrix is non-singular.

## 4.5 Order in which the nodes are updated

Another thing we can try to do to see the effect on the speed of convergence is instead of running through the nodes in a 'natural' ordering, order the nodes so that the nodes with largest residuals are updated first. That is, order the residuals largest through to smallest, then update the nodes in this order and see what effect this has on the overall speed of convergence.

In the following table, the first column contains the number of iterations, the second column contains the residual when we update all the nodes at the same time (Jacobi type iteration), the third column when we update the nodes one by one (Gauss-Seidel type iteration) in the natural ordering and in the fourth column when we order the nodes as described above and solve in the Gauss-Seidel manner. The results were obtained using the relaxation factors $\tau_u = 0.5$ and $\tau_{\mathbf{x}} = 0.01$. As can be seen from the table,

| no. its | Jacobi | G-S | G-S with ordering |
|---|---|---|---|
| 100 | 0.25780557953231 | 0.25269975407991 | 0.25105049499222 |
| 200 | 0.22015283653773 | 0.21342898356537 | 0.21416536607059 |
| 300 | 0.19370864310136 | 0.18721340527752 | 0.18739333210444 |
| 400 | 0.17536142945494 | 0.16862435271070 | 0.16849578707215 |
| 500 | 0.16053601824937 | 0.15346463739737 | 0.15354462765720 |
| 600 | 0.14778521173567 | 0.13979527492773 | 0.14009945109866 |
| 700 | 0.13576640329657 | 0.12729403953335 | 0.12770102927813 |
| 800 | 0.12497032439340 | 0.11642666027934 | 0.11671492458661 |
| 900 | 0.11534463671945 | 0.10688381860330 | 0.10696599915345 |
| 1000 | 0.10671577744691 | 9.8549423430357D-02 | 9.8409987322490D-02 |

Table 1: Comparison of J,GS and GS with ordering

the GS type and ordered GS type iterations converge faster than the Jacobi

type iteration, but the increase in the speed of convergence is only very slight.

## 4.6 Upwinding Approach on $u$ and $\mathbf{x}$

We can also do the same thing we did with the linear case in section 3.2 and update $u$ only for downwind nodes. This has the effect of increasing the rate of convergence (see figure 8(b)). Upwinding improves things even more if we also update $\mathbf{x}$ for downwind nodes. Using this approach with the relaxation factors $\tau_u = 0.5$ and $\tau_{\mathbf{x}} = 0.01$ (the same as before) improves the convergence rate by approximately 4 times. See figure 12(c).

## 4.7 Local approach to avoid tangling

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. Figure 3 shows how tangling can occur. The dotted lines indicate the convex hulls which bound the node movement and it can be seen that, even if node 1 remains fixed, nodes 2 and 3 can easily move far enough upwards to cause edge 23 to overtake node 1 and triangle 123 to 'flip', giving it a negative area.

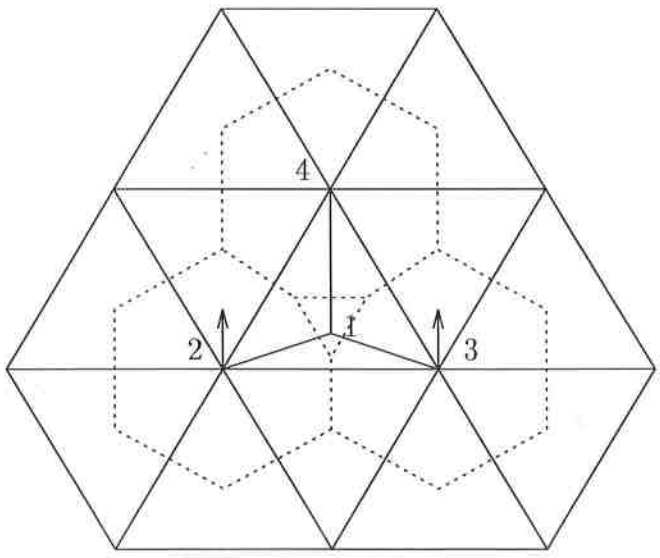


Figure 3: Mesh tangling in two dimensions.

The problem can be avoided by artificially limiting the distance which a

node can move. A simple but rather restrictive limit for a node $i$ is

$$(\Delta x)_{\max} = \min_{j \in \cup \Delta_i} \left( \frac{S_{\Delta_j}}{\max_{l=1,3} L_{jl}} \right) \tag{20}$$

where $j$ indexes the cells surrounding the node, $S_{\Delta_j}$ is the area of cell $j$ and $L_{jl}$ is the length of edge $l$ of cell $j$. The right hand side of (20) is equivalent to half the smallest height of the surrounding triangles. A more sophisticated limit would require prior knowledge of the direction and magnitude of the displacement of the adjacent nodes and its calculation would cause unnecessary expense. It should be noted that restricting the movement of nodes in this manner increases the probability of node locking. This is a phenomenon where nodes which would, under normal circumstances, be moved by the adaptation scheme are prevented from doing so by outside influences, such as any artificial bound placed on the nodes or by requiring boundary nodes to remain on the boundary.

# 5 Results

### Constant Advection Problem

The standard method was used for equation (1) on the domain $\Omega = [0,1] \times [0,1]$, with $\mathbf{a}$ defined to be the constant vector $(0,1)$. The initial solution was defined to be $u = 1$ at $(0.5, 0)$ and $u = 0$ everywhere else and the initial grid was taken as in figure 4. Unless stated otherwise the nodes are swept through in a *natural ordering*, that is we start in the bottom left corner, sweep right through the nodes, then move on to the leftmost node on the next level and repeat.

There are two ways in which we can perform the updates:

- Jacobi type Iteration. This is where the updates for ALL the nodes are calculated before the nodes are updated.
- Gauss-Seidel type Iteration. Here the update for one node is calculated and then that node is updated before the next node is considered.

The results are given in figures 4 - 8. The convergence history shown in figure 7(a) is obtained using the Jacobi update and shows that this method is slow to converge. Similar results are obtained if we use the Gauss-Seidel update. A comparison between the Jacobi and Gauss-Seidel updates was given in section 4 when we looked at the circular advection equation.

These results show that if we solve for $u$ from the matrix equation as described in section 3.1 then the rate of convergence is approximately that of the standard method.

Convergence was also speeded up using the upwind method as described in section 3.2, although here the relaxation factors had to be reduced to ensure that mesh tangling did not occur.

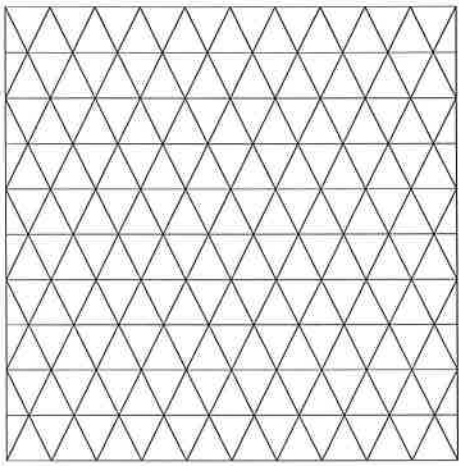

Figure 4: Initial grid for the constant advection equation.

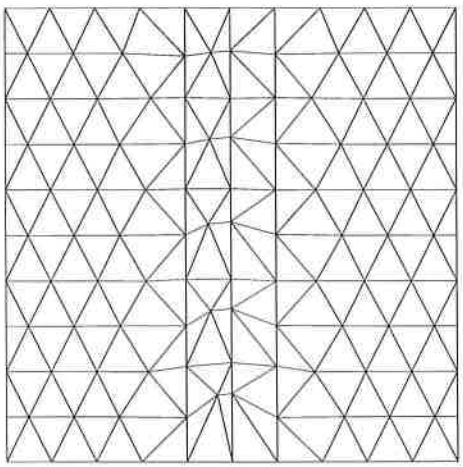

Figure 5: Final grid for the constant advection equation using the standard method with $\tau_u = 0.05$ and $\tau_{\mathbf{x}} = 0.001$.

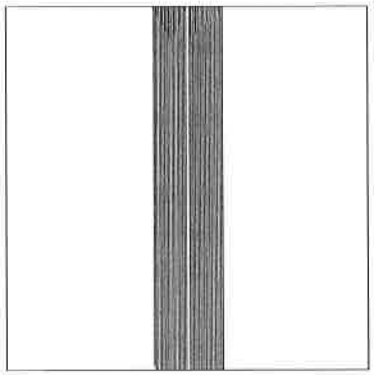

Figure 6: Final solution for the constant advection equation by the standard method with $\tau_u = 0.05$ and $\tau_{\mathbf{x}} = 0.001$.

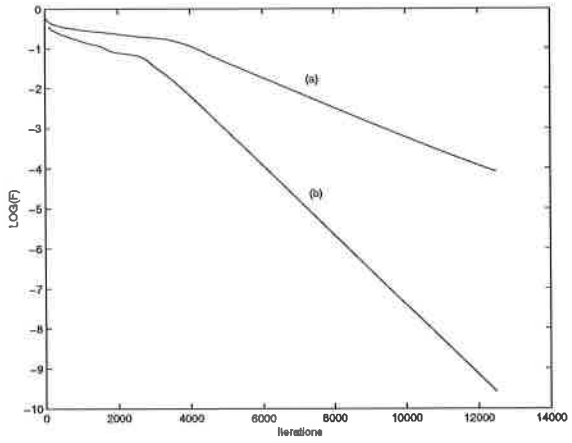


Figure 7: Convergence history for the constant advection equation with (a) the standard method with $\tau_u = 0.05$ and $\tau_{\mathbf{x}} = 0.001$ and (b) local GS type solver for $u$ and $\tau_{\mathbf{x}} = 0.001$.

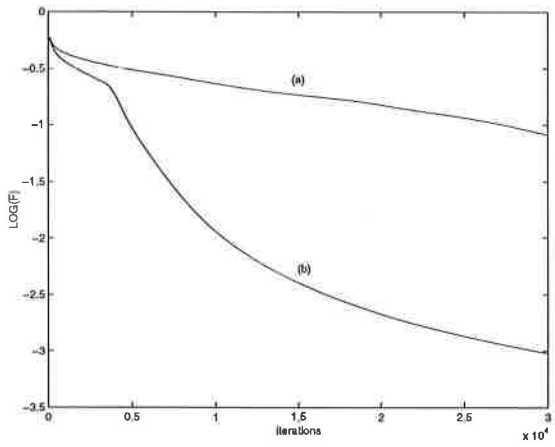


Figure 8: Convergence history for the constant advection equation with (a) the standard steepest descent method and (b) the upwind method on $u$ only, with $\tau_u = 0.005$ and $\tau_{\mathbf{x}} = 0.001$.

### Circular Advection Problem

The initial grid is given in figure 9 and the solution is defined to be zero everywhere except for the two points $(-0.6, 0)$ and $(-0.5, 0)$ where it is defined to be 1. The solution is poor when grid adaptation is suppressed [1].

When the standard method is used (11) with the constant relaxation factors $\tau_u = 0.5$ and $\tau_{\mathbf{x}} = 0.01$ the much improved results in figures 10, 11 and 12(a) were obtained.

The methods described in 4.1 and 4.2 had little effect on the rate of convergence. As can be seen from figure 13, $\delta u$ does not decrease exponentially to start off with but appears to after a while (around 5000) iterations. The same thing can be said of $\delta \mathbf{x}$. Although this is only for one node, other nodes exhibit similar behaviour. It should then be possible to switch on the Shanks update after say 5000 iterations to see if this speeds things up. Again convergence remained very slow.

Using the method decribed in section 4.3 it can be seen from the results (see figures 14, 15), the overall cost functional does initially go down much faster but then slows down and eventually gets stuck in a local minimum. The nodes where the solution is changing most rapidly have moved large distances and become out of place; they seem to continue moving in this direction and the grid never gets to its final solution (see for example figure 14). As can be seen from the solution data, although the cost functional is reasonably small after 200 iterations (much smaller than the original method) the solution is hopeless and it is unlikely that machine accuracy in the cost

functional would ever be achieved. Also working out the cost functional locally at every iteration is computationally expensive and any benefit this method may have had in terms of number of iterations needed to converge is outweighted by the length of time it takes the program to run. Diagonal swapping may improve things here. If we use the local solver for $u$ and the standard method for $\mathbf{x}$ we obtain similar results.

When $u$ is solved using a matrix equation as described in section 4.4, convergence history in figure 12(b) shows that the solution and grid converge at approximately twice the rate at which they do using the standard method.

If we update $u$ and $\mathbf{x}$ using the upwind approach described in 4.6, this has a more dramatic effect. The solution and grid converge at approximately 4 times the rate of the standard method (see figure 12(c).)

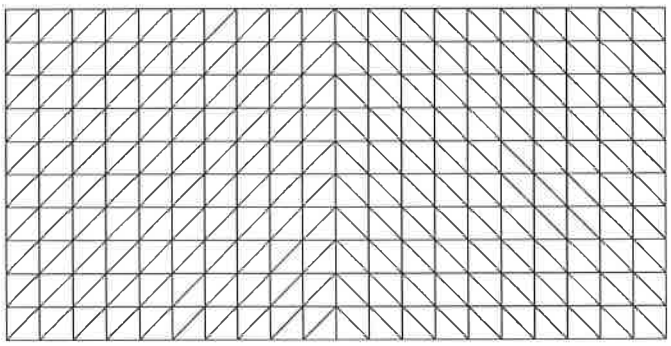

Figure 9: Initial grid for circular advection problem.

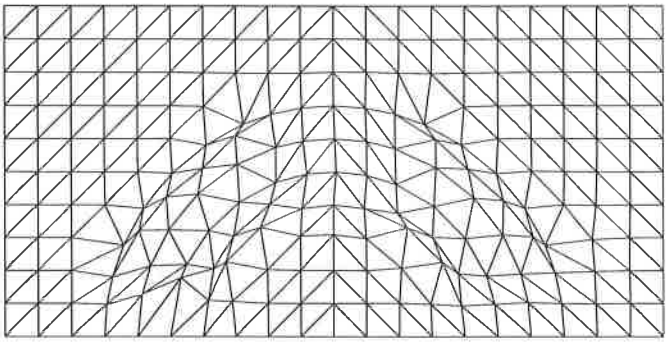

Figure 10: Final grid for the circular advection problem using the standard method with $\tau_u = 0.5$ and $\tau_{\mathbf{x}} = 0.01$.

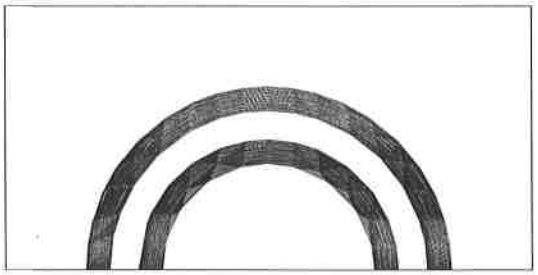

Figure 11: Final solution for the circular advection problem using the standard method with $\tau_u = 0.5$ and $\tau_{\mathbf{x}} = 0.01$.

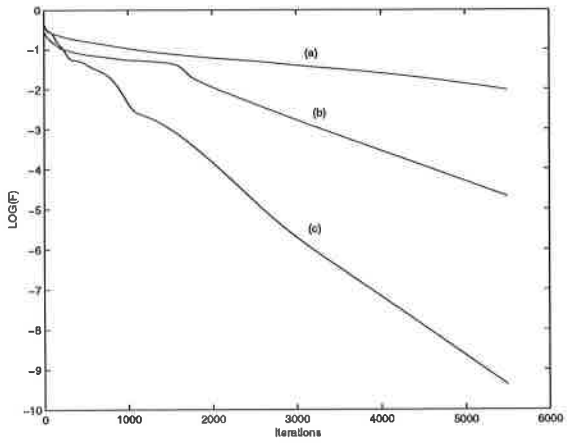


Figure 12: Convergence history for the circular advection problem with (a) the standard method with $\tau_u = 0.5$ and $\tau_{\mathbf{x}} = 0.01$, (b) local GS type solver for $u$ and steepest descent for $\mathbf{x}$ with $\tau_{\mathbf{x}} = 0.01$, (c) upwind descent approach for $u$ and $\mathbf{x}$ with $\tau_u = 0.5$ and $\tau_{\mathbf{x}} = 0.01$ .

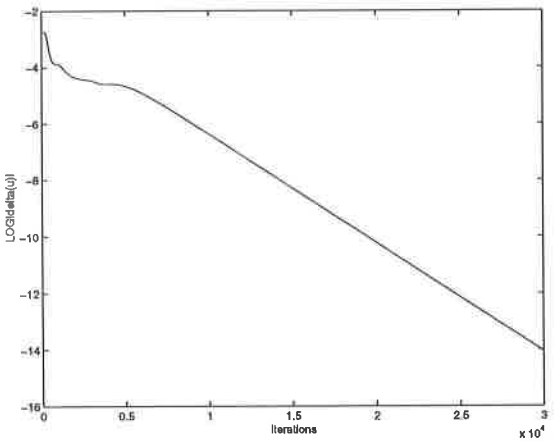


Figure 13: Log of $|\delta u|$ for node 26, showing that for a general node $\delta u$ does not decrease exponentially, although after a while it settles down to exponential decay.

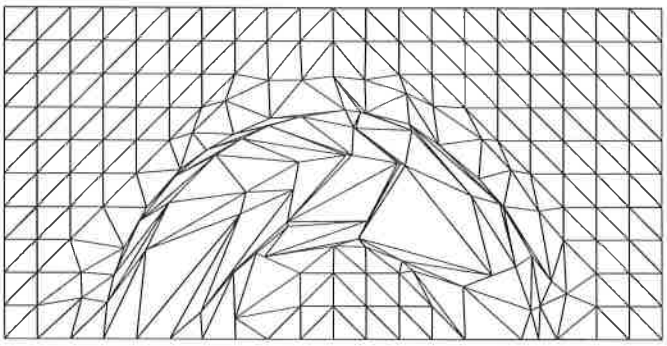

Figure 14: Grid after 200 iterations for the method which tries to optimise the relaxation factors for $u$ and $\mathbf{x}$ in the manner of section 4.3.

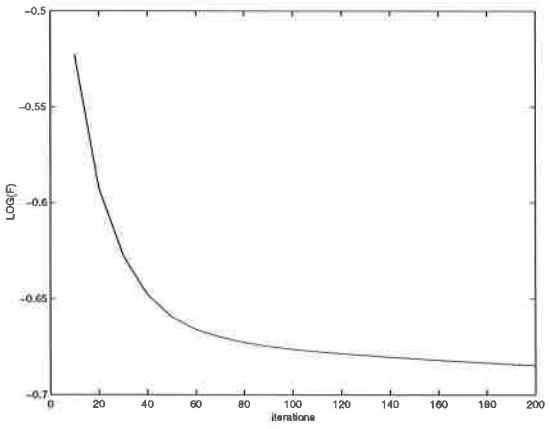


Figure 15: Convergence History for the method which tries to optimise the relaxation factors for $u$ and $\mathbf{x}$.

# 6 Conclusions

The method described in [1] for the solution of advection problems is slow to converge in the cases considered, and I have attempted to speed up the rate of convergence. It still remains unclear what choice of relaxation factor in the steepest descent procedure optimises the rate of convergence. The rate of convergence of the method described in section 4.3 was the best to begin with, but was spoiled by nodes moving with the solution front. The idea

described in [2] where grid points move perpendicular to characteristics may help here and diagonal swapping may also be beneficial. Updating $u$ from a matrix equation which arises from solving the variational equation directly improves convergence, and further work should include the use of this idea locally for solving the $\mathbf{x}$ equation on patches which have non-zero residual. So far the best convergence results have come from an upwind approach on both the solution and the grid using Steepest Descent. Here the relaxation factors are chosen only to achieve a reduction in $F$, and faster convergence would be achieved if these relaxation factors were optimised.

**Acknowledgements**

I would like to thank Professor M.J.Baines and Dr.M.E.Hubbard of the Department of Mathematics, University of Reading for their supervision and advice during this work.

# References

[1] P.L.Roe, Compounded of Many Simples, in Proceedings of on Barriers and Challenges is CFD, ICASE, NASA Langley, August 1996, Kluwer (to appear).

[2] M.J.Baines, A Note on Duality in a Scalar Hyperbolic Equation, Numerical Analysis Report 8/97, University of Reading, 1997.

[3] D.Shanks, Non-Linear Transformations of Divergent and Slowly Convergent Sequences, 1954.

[4] M.E.Hubbard, Ph.D. Thesis, University Of Reading, 1996

[5] W.Cheney and D.Kincaid, Numerical Mathematics and Computing, Brooks/Cole Publishing Company.