Efficient implicit algorithm for the equations of 2-D viscous compressible flow: application to shock–boundary layer interaction

W. N. Dawes*

A fully implicit algorithm has been developed to time integrate the equations of 2-D compressible viscous flow. The algorithm was constructed so as to optimize computational efficiency. The time-consuming block matrix inversions usually associated with implicit algorithms have been reduced to the trivial non-iterative inversion of four sets of scalar bidiagonal matrices. Thus, the algorithm requires virtually no more computer storage than an explicit algorithm. The efficient structure of the implicit algorithm is reflected in comparative timings which show that it requires only a factor of two more computer time per point per time step than a typical explicit algorithm. Therefore, the algorithm allows more economical solution of given flows than existing explicit methods and also allows more difficult problems to be attempted using available computer resources. Application of the algorithm to the problem of shock–boundary layer interaction produces results consistent with both experimental measurements and other calculations.

Key words: *fluid dynamics, Navier-Stokes equations, compressible flow*

Many problems of current interest in both turbomachinery and elsewhere cannot be approached using conventional inviscid flow analyses or even inviscid–viscous interactive methods but require solution of the full equations of compressible viscous flow, the Navier–Stokes equations. Numerical integration of the Navier–Stokes equations using explicit algorithms suffers from severe time-step restrictions. By contrast, many implicit algorithms allow larger time steps but are inefficiently structured so that their potential advantages are not realized. In particular, the major portion of the work in an implicit finite difference algorithm is contained in solving a set of simultaneous equations. When an implicit algorithm is applied to a system of partial differential equations, such as the Navier–Stokes equations, a coupled set of block matrix–vector equations are obtained that are complicated and time consuming to solve.

This paper shows how an efficient implicit algorithm may be constructed by restructuring the coupled set of implicit equations into a set of scaler equations which require only the trivial inversion of four sets of scalar bidiagonal matrices to advance the solution process.

To illustrate the capabilities of the new algorithm it is applied to the problem of the interaction of a shock with a laminar boundary layer.

* Central Electricity Generating Board, Marchwood Engineering Laboratories, Marchwood, Southampton, Hants, UK, SO4 4ZB
Received 5 August 1982 and accepted for publication on 18 November 1982

Governing equations

The Navier–Stokes equations

The Navier–Stokes equations for the 2-D compressible viscous flow of a perfect gas are written in conservation law form as:

$$\frac{\partial \bar{U}}{\partial t}+\frac{\partial \bar{F}}{\partial X}+\frac{\partial \bar{G}}{\partial Y}=\frac{1}{Re}\left[\frac{\partial \bar{V}}{\partial X}+\frac{\partial \bar{W}}{\partial Y}\right] \tag{1}$$

where:

$$\bar{U}=\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho E\end{bmatrix},\ \bar{F}=\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ (\rho E+p)u\end{bmatrix},\ \bar{G}=\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ (\rho E+p)v\end{bmatrix}$$

$$\bar{V}=\begin{bmatrix}0\\ \tau_{xx}\\ \tau_{xy}\\ (\gamma/Pr)k(\rho E)_x+u\tau_{xx}+v\tau_{xy}\end{bmatrix},$$

$$\bar{W}=\begin{bmatrix}0\\ \tau_{xy}\\ \tau_{yy}\\ (\gamma/Pr)k(\rho E_y+u\tau_{xy}+v\tau_{yy}\end{bmatrix}$$

and:

$$\tau_{xx}=\tfrac{4}{3}\mu u_x-\tfrac{2}{3}\mu v_y$$

$$\tau_{xy}=\mu(u_y+v_x)$$

$$\tau_{yy}=\tfrac{4}{3}\mu v_y-\tfrac{2}{3}\mu u_x$$

0142–727X/83/010017–10$03.00

All the variables have been suitably non-dimensionalized with Re, the Reynolds number, and Pr, the Prandtl number. The Reynolds number and Prandtl number are formed using suitable reference quantities (for example free stream conditions) so dimensionless coefficients of viscosity and thermal conductivity, μ and k, have been included to allow μ and k to vary spatially (with local temperature, for example, or by the inclusion of an eddy viscosity turbulence model).

The equation of state is:

$$p = \rho RT \tag{2}$$

and the pressure is related to the other variables by:

$$p = (\gamma - 1)(\rho E - \tfrac{1}{2}\rho(u^2 + v^2)) \tag{3}$$

Co-ordinate transformation

For convenience and computational efficiency, the physical $X - Y$ plane is mapped into a rectangular computational plane, $\xi - \eta$, as shown in Fig 1. The following independent variable transformation is used:

$$\begin{aligned} \xi &= \xi(X, Y) \\ \eta &= \eta(X, Y) \end{aligned} \tag{4}$$

with the associated area Jacobian:

$$J^{-1} = X_\xi Y_\eta - X_\eta Y_\xi \tag{5}$$

The geometrical factors (metrics) of the transformation are related by:

$$\begin{aligned} \xi_x &= JY_\eta \qquad \eta_x = -JY_\xi \\ \xi_y &= -JX_\eta \qquad \eta_y = JX_\xi \end{aligned} \tag{6}$$

Applying these transformations to the governing Eq (1) and manipulating them so as to retain their conservation form in the $\xi - \eta$ plane gives (see Refs 1–3):

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial \xi} + \frac{\partial G}{\partial \eta} = \frac{1}{Re}\left[\frac{\partial V}{\partial \xi} + \frac{\partial W}{\partial \eta}\right] \tag{7}$$

where

$$\begin{aligned} U &= J^{-1}\bar{U} \\ F &= J^{-1}(\xi_x \bar{F} + \xi_y \bar{G}) \\ G &= J^{-1}(\eta_x \bar{F} + \eta_y \bar{G}) \\ V &= J^{-1}(\xi_x \bar{V} + \xi_y \bar{W}) \\ W &= J^{-1}(\eta_x \bar{V} + \eta_y \bar{W}) \end{aligned} \tag{8}$$

Introducing the so-called contravarient velocities:

$$\begin{aligned} \hat{u} &= \xi_x u + \xi_y v \\ \hat{v} &= \eta_x u + \eta_y v \end{aligned} \tag{9}$$

Notation

A	The Jacobean $\partial F/\partial U$
A^+, A^-	See Eq (24)
B	The Jacobean $\partial G/\partial U$
B^+, B^-	See Eq (24)
c	Sonic speed $(\gamma p/\rho)^{1/2}$
C	The Jacobean $\partial V/\partial U$
D	The Jacobean $\partial W/\partial U$
F	ξ-wise flux vector
G	η-wise flux vector
i	ξ-wise index
j	η-wise index
J	Area Jacobean of axis transformation
k	Thermal conductivity
n	Time-wise index
$\hat{N}$	See Eq (30)
p	Static pressure
Pr	Prandtl number
R	Gas constant
Re	Reynolds number
t	Time
T	Static temperature
T_ξ, T_ξ^{-1} T_η, T_η^{-1}	Diagonalizing matrices in similarity transformation
u	X-wise velocity
$\hat{u}$	Contravariant velocity: $\xi_x u + \xi_y v$
U	State vector $J^{-1}[\rho, \rho u, \rho v, \rho E]$
v	Y-wise velocity
$\hat{v}$	Contravariant velocity, $\eta_x u + \eta_y v$
V	ξ-wise viscous flux vector
W	η-wise viscous flux vector
X Y	Co-ordinates in the physical plane
γ	Ratio of specific heats
δ^+	Backward difference operator, $\delta^+\phi = \phi_i - \phi_{i-1}$
δ^-	Forward difference operator, $\delta^-\phi = \phi_{i+1} - \phi_i$
δ^0	Centred difference operator, $\delta^0\phi = (\phi_{i+1} - \phi_{i-1})/2$
ΔU	Change in U due to local effects
δU	Change in U corrected for global effects
Δt	Time step
ξ η	Co-ordinates in the transformed plane
μ	Coefficient of viscosity
Λ_ξ Λ_η	Diagonal eigenvalue matrices
$\Lambda_\xi^+, \Lambda_\xi^-$ $\Lambda_\eta^+, \Lambda_\eta^-$	See Eq (23)
ρ	Density
ρE	Total internal energy $\rho(c_v T + \frac{1}{2}(u^2 + v^2))$
τ_{xx} τ_{xy} τ_{yy}	Viscous stresses, see Eq (1)
BC	Boundary condition
CFL	Courant Freidrichs Lewy
LHS	Left-hand side
RHS	Right-hand side
$\bar{}$	Vector in physical X-Y plane
$*$	Intermediate value

allows the expressions for F and G to be written in the compact form:

$$F = J^{-1}\begin{bmatrix} \rho u \\ \rho u\hat{u} + \xi_x p \\ \rho v\hat{u} + \xi_y p \\ (\rho E + p)\hat{u} \end{bmatrix}$$

and

$$G = J^{-1}\begin{bmatrix} \rho \hat{v} \\ \rho u\hat{v} + \eta_x p \\ \rho v\hat{v} + \eta_y p \\ (\rho E + p)\hat{v} \end{bmatrix} \qquad (10)$$

which are little more complex than the original $\bar{F}$ and $\bar{G}$.

The viscous stress terms in V and W are evaluated by straight-forward application of the chain rule, for example;

$$\tau_{xx} = \tfrac{4}{3}\mu(\xi_x u_\xi + \eta_x u_\eta) - \tfrac{2}{3}\mu(\xi_y v_\xi + \eta_y v_\eta) \qquad (11)$$

It is noteworthy that the transformation of the governing equations is analogous to recasting the equations in general finite volume form (see, for example, Ref 4) with the 'metrics' representing the various areas of the finite volume faces. The transformation allows grid points to be aligned exactly with the boundaries to the flow (for example blade surfaces) and also allows gradients of flow properties local to the boundary to be accurately represented. In addition, the generality of the transformation allows suitable treatment of difficult regions in the flow like leading and trailing edges (as indicated in Fig 1). Grid control is important in a viscous flow algorithm because, in physical terms, vorticity is generated at solid flow boundaries (ie the blade surface boundary layer) and then diffuses and convects into the main flow; it is necessary for this vorticity generation (represented by the near-wall vorticity gradients) to be modelled accurately. A significant additional advantage of using such a transformed system of equations is that rapidly refining grids can be used without introducing any numerical viscous effects to mask the physical effects[5].

The Variable ΔU

To integrate the equations of motion forward in time an appropriate finite-difference algorithm must be constructed. For this purpose it is convenient to introduce a new variable, ΔU:

$$\Delta U^n = -\Delta t\left\{\frac{\partial F}{\partial \xi} + \frac{\partial G}{\partial \eta} - \frac{1}{Re}\left[\frac{\partial V}{\partial \xi} + \frac{\partial W}{\partial \eta}\right]\right\}^n \qquad (12)$$

where Δt is the time step used for the integration, and n is the temporal index $t = n\,\Delta t$.
The variable ΔU^n represents the temporal variation of the flow variables in response to the spatial fluxes of these flow variables. If the integration converges to a steady state then ΔU^∞ becomes zero.

Explicit time-marching algorithms

It is relatively straightforward to construct explicit algorithms to integrate the flow equations. It has been shown[6] that the Brailovskaya method is one of the more suitable. This may be expressed in terms of ΔU as:

$$\left.\begin{aligned} U^*_{ij} &= U^n_{ij} + \Delta U^n_{ij} \\ U^{n+1}_{ij} &= U^n_{ij} + \Delta U^*_{ij} \end{aligned}\right\} \qquad (13)$$

where i and j are the spacial indices

$$X = i\,\Delta X \quad \text{and} \quad Y = j\,\Delta Y$$

* denotes the intermediate values in this two step algorithm, and ΔU is evaluated using conventional second-order accurate centred differences.

The algorithm is conditionally stable (as far as convection is concerned) with:

$$\Delta t \leq \underset{ij}{\text{MIN}}\left\{\frac{1}{|\hat{u}| + c(\xi_x^2 + \xi_y^2)^{1/2}}, \frac{1}{|\hat{v}| + c(n_x^2 + \eta_y^2)^{1/2}}\right\} \qquad (14)$$

where c is the sonic velocity, $(\gamma p/\rho)^{1/2}$.
This restriction is simply the familiar CFL condition cast in terms of the contravarient velocities (and with $\Delta\xi = \Delta\eta = 1$). The essential disadvantage of using an explicit algorithm like this is that these allowable time steps are very small where the grid has been highly refined to resolve, for example, steep velocity gradients near blade surfaces.

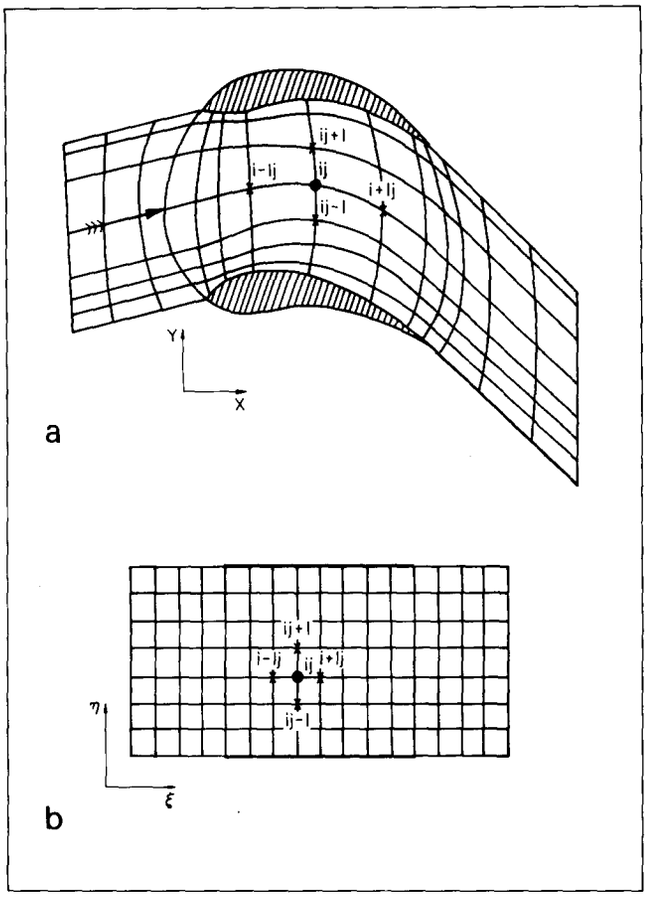

Fig 1 Co-ordinate mapping of physical to computational planes

Physical basis for an implicit algorithm

Implicit algorithms can be constructed to integrate the flow equations allowing the use of much longer time steps and faster convergence to a steady stage (if one exists). Here, a physical basis for an implicit algorithm is developed.

The variable, ΔU (Eq (12)) represents the temporal change of the flow properties at a point due to local spatial fluxes; it is this 'localness' which gives rise to the CFL restriction on time step. Implicit algorithms are characterized by their 'global' nature, ie each point in the flow influencing every other point every time step. So it is natural to look for some global transport equation for ΔU. Following MacCormack[7] the equations of motion (Eq (7)) are differentiated once with respect to time, giving:

$$\frac{\partial}{\partial t}\left(\frac{\partial U}{\partial t}\right)+\frac{\partial}{\partial \xi}\left(A\frac{\partial U}{\partial t}\right)+\frac{\partial}{\partial \eta}\left(B\frac{\partial U}{\partial t}\right) = \frac{1}{Re}\left[\frac{\partial}{\partial \xi}\left(C\frac{\partial U}{\partial t}\right)+\frac{\partial}{\partial \eta}\left(D\frac{\partial U}{\partial t}\right)\right] \tag{15}$$

where A, B, C and D are the Jacobean matrices:

$$A=\partial F/\partial U, \quad B=\partial G/\partial U, \quad C=\partial V/\partial U, \quad D=\partial W/\partial U \tag{16}$$

A, B, C and D are given, for example, in Refs (3) and (8); in two dimensions these matrices are (4×4). Eq (15) is clearly a convection–diffusion equation for the variable '$(\partial U/\partial t)$'. An *im*plicit equation for integrating Eq (15) may be written as:

$$\left[I+\Delta t\left\{\frac{\partial}{\partial \xi}(A.+\frac{\partial}{\partial \eta}(B.-\frac{1}{Re}\left[\frac{\partial}{\partial \xi}(C.+\frac{\partial}{\partial \eta}(D.\right]\right\}\right] \times\left(\frac{\partial U}{\partial t}\right)^{n+1}=\left(\frac{\partial U}{\partial t}\right)^{n} \tag{17}$$

where: $(A.$ etc. indicates that the matrix–vector multiply remains within the spatial differential, and A, B, C and D are evaluated at the known time level, n. This equation has first-order temporal accuracy.

Defining a new variable, δU^{n+1}, as:

$$\delta U^{n+1}=\left(\frac{\partial U}{\partial t}\right)^{n+1}\Delta t$$

and identifying

$$\Delta U^{n}=\left(\frac{\partial U}{\partial t}\right)^{n}\Delta t$$

allows Eq (17) to be recast in the desired form;

$$\left[I+\Delta t\left\{\frac{\partial}{\partial \xi}(A.+\frac{\partial}{\partial \eta}(B. -\frac{1}{Re}\left[\frac{\partial}{\partial \xi}(C.+\frac{\partial}{\partial \eta}(D.\right]\right\}\right]\delta U^{n+1}=\Delta U^{n} \tag{18}$$

where:

$$\Delta U^{n}=-\Delta t\left\{\frac{\partial F}{\partial \xi}+\frac{\partial G}{\partial \eta}-\frac{1}{Re}\left[\frac{\partial V}{\partial \xi}+\frac{\partial W}{\partial \eta}\right]\right\}^{n}$$

and:

$$U^{n+1}=U^{n}+\delta U^{n+1}$$

The *ex*plicit RHS of this equation represents the temporal change of U due to *local* fluxes; the *im*plicit LHS represents the *global* convection and diffusion of these temporal changes resulting in a modified local change, δU. The explicit RHS is nonlinear; the implicit LHS is, however, linear and its 'inversion' may be attacked using the many available matrix handling methods. Eq (18) represents a sound physical basis for an implicit algorithm and is expressed in a form suitable for efficient numerical integration. The equation is *fully* implicit in that the components of δU^{n+1} (ie $\delta\rho^{n+1}$, $\delta\rho u^{n+1}$, $\delta\rho v^{n+1}$ and $\delta\rho E^{n+1}$) are coupled implicitly together at the new time level via the Jacobeans A, B, C and D and are thus simultaneously updated.

Eq (18) has been derived before by Beam and Warming[9] but not by the physically relevant route suggested by MacCormack and used here.

Construction of an efficient implicit algorithm

Implicit algorithms allow larger time steps than explicit methods and hence faster convergence to a steady state (if one exists). However, implicit algorithms inevitably require more computer processing time per point per time step so it is essential for the algorithms to be efficiently constructed so as not to waste their potential advantages. A suitable basis for an implicit algorithm has been derived, Eq (18), and it is possible to construct a very efficient algorithm for time-marching the equation; in fact the algorithm is so efficient it requires only twice the computer time per point per time step used by the explicit Brailovskaya algorithm (see previously).

The implicit time-marching of Eq (18) is obtained by inverting at each time step the matrix represented by the linear LHS, ie

$$\delta U^{n+1}=\left[I+\Delta t\left\{\frac{\partial}{\partial \xi}(A.\ldots.\right]^{-1}\Delta U^{n} \tag{19}$$

The efficiency of the algorithm is largely determined by the efficiency of this inversion. If only a steady-state solution is required then the inversion need not be carried out exactly (provided the time-marching remains stable) because in the steady state $\Delta U = \delta U \equiv 0$, by definition. Thus an acceptable approximation to the inversion may be devised so as to optimize the computational efficiency and rate of convergence. Accordingly, an efficient implicit algorithm is constructed as follows.

Modified form of the implicit operator

Firstly, the viscous terms are dropped from the LHS of Eq (18), giving:

$$\left[I+\Delta t\left\{\frac{\partial}{\partial \xi}(A.+\frac{\partial}{\partial \eta}(B.\right\}\right]\delta U^{n+1}=\Delta U^{n} \tag{20}$$

with ΔU defined in Eq (18). This halves the computational work required to assemble the LHS. The viscous terms are, of course, retained in the formation of ΔU so that the steady state, $\Delta U = 0$, is correct.

To compensate for the absence of the viscous terms on the implicit LHS of Eq (20) the differential operators are replaced by simple one-sided or 'up-winded' difference operators. This introduces significant dissipation which not only balances the viscous terms retained on the RHS of Eq (20) but also helps control any violent transients. The use of one-sided difference operators is limited by stability constraints to regions of the computational domain where the eigenvalues of the Jacobeans A and B are of the same sign[9]; ie the difference operators must be strictly up-winded in accordance with the physical flow of information through the computational domain. In general, these eigenvalues are of mixed sign and so the use of one-sided differences requires special treatment. As A and B are the Jacobeans of a hyperbolic system of equations, similarity transformations occur which diagonalize A and B:

$$\left.\begin{aligned} &\Lambda_\xi = T_\xi^{-1} A T_\xi \\ &\text{and} \\ &\Lambda_\eta = T_\eta^{-1} B T_\eta \end{aligned}\right\} \tag{21}$$

Here, Λ_ξ and Λ_η are diagonal matrices containing the eigenvalues of A and B, ie

$$\Lambda_\xi = \begin{bmatrix} \hat{u} & 0 & 0 & 0 \\ 0 & \hat{u} & 0 & 0 \\ 0 & 0 & \hat{u}+c(\xi_x^2+\xi_y^2)^{1/2} & 0 \\ 0 & 0 & 0 & \hat{u}-c(\xi_x^2+\xi_y^2)^{1/2} \end{bmatrix}$$

$$\Lambda_\eta = \begin{bmatrix} \hat{v} & 0 & 0 & 0 \\ 0 & \hat{v} & 0 & 0 \\ 0 & 0 & \hat{v}+c(\eta_x^2+\eta_y^2)^{1/2} & 0 \\ 0 & 0 & 0 & \hat{v}-c(\eta_x^2+\eta_y^2)^{1/2} \end{bmatrix} \tag{22}$$

The values of T_ξ, T_ξ^{-1}, T_η and T_η^{-1} are given in Ref 10 and are reproduced for reference in the Appendix. It has been shown[9] that by extracting the positive and negative eigenvalues:

$$\left.\begin{aligned} \Lambda^+ &= \tfrac{1}{2}(\Lambda + |\Lambda|) \\ \Lambda^- &= \tfrac{1}{2}(\Lambda - |\Lambda|) \end{aligned}\right\} \tag{23}$$

A and B may be split into two parts, one associated with positive eigenvalues and one with negative:

$$\left.\begin{aligned} &A = A^+ + A^- \\ &B = B^+ + B^- \\ &\text{where:} \\ &A^+ = T_\xi \Lambda_\xi^+ T_\xi^{-1} \\ &A^- = T_\xi \Lambda_\xi^- T_\xi^{-1} \\ &\text{and:} \\ &B^+ = T_\eta \Lambda_\eta^+ T_\eta^{-1} \\ &B^- = T_\eta \Lambda_\eta^- T_\eta^{-1} \end{aligned}\right\} \tag{24}$$

Appropriate one-sided difference operators, for example:

$$\left.\begin{aligned} \delta_\xi^+ \phi &= \phi_i - \phi_{i-1} \\ \delta_\xi^- \phi &= \phi_{i+1} - \phi_i \end{aligned}\right\} \tag{25}$$

may be applied to the split forms of A and B allowing Eq (20) to be rewritten as;

$$[I + \Delta t\{\delta_\xi^+(A^+ \cdot) + \delta_\xi^-(A^- \cdot) + \delta_\eta^+(B^+ \cdot) + \partial_\eta^-(B^- \cdot)\}] \times \delta U^{n+1} = \Delta U^n \tag{26}$$

Approximate factorization of the implicit operator

The matrix operator on the LHS of Eq (26) is block -5-diagonal and its inversion could be attacked iteratively using Stone's strongly implicit algorithm. However, it was decided that greater computational efficiency would be achieved by approximately factoring the LHS into more readily invertible matrices (see, for example, Refs 3, 8, 9). The obvious factorization is a lower and an upper block -3-diagonal matrix, both of which are readily and non-iteratively invertible, ie

$$[I + \Delta t\{\partial_\xi^+(A^+ \cdot) + \partial_\eta^+(B^+ \cdot)\}] \times [I + \Delta t\{\partial_\xi^-(A^- \cdot) + \partial_\eta^-(B^- \cdot)\}\, \delta U^{n+1} = \Delta U^n \tag{27}$$

Nevertheless, it was decided that the most efficient factorization would be into four one-dimensional operators, ie

$$[I + \Delta t \partial_\xi^+(A^+ \cdot)][I + \Delta t \partial_\xi^-(A^- \cdot)] \times [I + \Delta t \partial_\eta^+(B^+ \cdot)][I + \Delta t \partial_\eta^-(B^- \cdot)]\, \delta U^{n+1} = \Delta U^n \tag{28}$$

This is clearly an approximation, but does not affect the accuracy of the converged steady state, $\Delta U = 0$. This algorithm is efficient and non-iterative as each of the four matrix operators is block bidiagonal and so can be inverted simply by sweeping through the computational domain in the appropriate directions. Nevertheless, these sweeps still require the inversion of the (4×4) matrices contained in the block structure. These (4×4) matrices are, in general, full and so some form of Gaussian elimination must be used to invert them. Therefore, it is possible that the algorithm could be made even more efficient.

Diagonalization of the implicit operator

Ref 10 shows how the similarity transformations, Eq (21), may be used to 'diagonalize' a set of hyperbolic equations. A similar approach is adopted here. Substituting for A^+ etc. from Eq (24) into Eq (28), with suitable substitutions for the unit matrices, gives:

$$\begin{aligned} &[(T_\xi T_\xi^{-1}) + \Delta t\, \partial_\xi^+(T_\xi \Lambda_\xi^+ T_\xi^{-1} \cdot)] \\ &\times [(T_\xi T_\xi^{-1}) + \Delta t\, \partial_\xi^-(T_\xi \Lambda_\xi^- T_\xi^{-1} \cdot)] \\ &\times [(T_\eta T_\eta^{-1}) + \Delta t\, \partial_\eta^+(T_\eta \Lambda_\eta^+ T_\eta^{-1} \cdot)] \\ &\times [(T_\eta T_\eta^{-1}) + \Delta t\, \partial_\eta^-(T_\eta \Lambda_\eta^- T_\eta^{-1} \cdot)]\, \delta U^{n+1} = \Delta U^n \end{aligned} \tag{29}$$

Making an approximation allows this to be rewritten as the desired implicit algorithm:

$$T_\xi [1 + \Delta t\, \partial_\xi^+(\Lambda_\xi^+ \cdot)][1 + \Delta t\, \partial_\xi^-(\Lambda_\xi^- \cdot)] \hat{N} [1 + \Delta t\, \partial_\eta^+(\Lambda_\eta^+ \cdot)] \times [1 + \Delta t\, \partial_\eta^-(\Lambda_\eta^- \cdot)] T_\eta^{-1}\, \delta U^{n+1} = \Delta U^n \tag{30}$$

where:

$$\hat{N} = T_\xi^{-1} T_\eta$$

$$\Delta U = -\Delta t \left\{ \partial_\xi^0 F + \partial_\eta^0 G - \frac{1}{Re} [\partial_\xi^0 V + \partial_\eta^0 W] \right\}$$

$$U^{n+1} = U^n + \delta U^{n+1}$$

and

δ^0 represents conventional second-order accurate difference operators

The four 1-D matrix operators are still block-diagonal in form but now, because Λ_ξ and Λ_η are *diagonal* matrices (see Eq 22) each of the 1-D operators can be dealt with as four trivially inverted *scalar* bidiagonal operators. This gives the algorithm great efficiency and in addition means that virtually no extra computer storage is required.

A simple von Neumann stability analysis indicates that this algorithm is unconditionally stable. In practice the time step may be limited by non-linear effects and by particular forms of boundary condition.

Solution Process for the Implicit Algorithm

Eq (30) is integrated forward in time using the following straight-forward stages,

1 ΔU^n is evaluated at each interior node of the computational domain using conventional second-order central difference operators (as indicated in Eq (30)). The converged steady state, $\Delta U = 0$, is therefore resolved to second-order spatial accuracy.

2 A matrix–vector multiply at each interior mode:

$$D_1 = (T_\xi^{-1})^n \Delta U^n \tag{31}$$

3 Four trivial inversions of the first set of scalar bidiagonal operators:

$$D_2 = [1 + \Delta t\, \partial_\xi^+ (\Lambda_\xi^+ .)]^{-1} D_1 \tag{32a}$$

This is carried out by four simple sweeps through the computational domain in the direction of increasing 'i':

$$(D_2)_i + \Delta t (\Lambda_\xi^+ D_2)_i = (D_1)_i + \Delta t (\Lambda_\xi^+ D_2)_{i-1} \tag{32b}$$

4 Four, trivial inversions of the second set of scalar bi-diagonal operators:

$$D_3 = [1 + \Delta t\, \partial_\xi^- (\Lambda_\xi^- .)]^{-1} D_2 \tag{33a}$$

This is carried out by four simple sweeps in the direction of decreasing 'i':

$$(D_3)_i - \Delta t (\Lambda_\xi^- D_3)_i = (D_2)_i - \Delta t (\Lambda_\xi^- D_3)_{i+1} \tag{33b}$$

5 A matrix–vector multiply at each interior node:

$$D_4 = (\hat{N}^{-1})^n D_3 \tag{34}$$

6 Two more sets of four trivial inversions of the third and fourth scalar bi-diagonal operators,

$$D_5 = [1 + \Delta t \partial_\eta^+ (\Lambda_\eta^+ .)]^{-1} D_4 \tag{35}$$

$$D_6 = [1 + \Delta t\, \partial_\eta^- (\Lambda_\eta^- .)]^{-1} D_5 \tag{36}$$

the first being carried out by four simple sweeps in the direction of increasing 'j' and the second with four sweeps for decreasing 'j'.

7 A final matrix–vector multiply at each interior node:

$$\delta U^{n+1} = (T_\eta)^n D_6 \tag{37}$$

and

8 Up-dating the state vector at each interior node:

$$U^{n+1} = U^n + \delta U^{n+1} \tag{38}$$

Boundary conditions

As well as applying appropriate boundary conditions to U at each successive time-level, suitable boundary conditions must also be devised for δU for use with the four sets of matrix inversions, Eqs (32), (33), (35) and (36). The most straight-forward boundary condition for δU is simply to set $\delta U_{BC} = 0$; this allows the algorithm great generality of application and is correct in the converged steady state. However, the setting of $\delta U_{BC} = 0$ means effectively that the boundary conditions are applied in an explicit manner and this may reduce the maximum stable time step. Nevertheless, time steps larger than the CFL limit for explicit algorithms can still be achieved with correspondingly more rapid convergence to a steady state.

Computer processing time

As stated before, the careful structuring of the implicit algorithm (Eq (30)) gives it great computational efficiency. Table 1 compares the computation time in s per node point per time step for the implicit algorithm with that of the explicit Brailovskaya algorithm (see earlier). The implicit algorithm, despite being fully implicit (ie each member of the state vector U being implicitly coupled together and simultaneously updated) requires only a factor of two more computer time per point per time step than the explicit method, but the implicit algorithm has more favourable stability properties.

Interpretation of the implicit algorithm as an iterative replacement scheme

The derivation of the implicit algorithm, Eq 30 is mathematical and it is useful to interpret the algorithm as a type of iterative replacement scheme (IRS). The aim of an IRS (see, for example, Refs 4, 13 and 14) is to use a simplified and less accurate discretization of the equations of motion during the time-marching itself and to add correction terms

Table 1

Method	Computer time/point/time step (IBM 370-168 OPT = 2)	Relative cost
Current implicit algorithm	6×10^{-4} seconds	2
Brailovskaya explicit algorithm	3×10^{-4} seconds	1

iteratively as the time-marching proceeds so that the converged steady solution has the desired accuracy. For ease of presentation, consider the one-dimensional inviscid equation of motion:

$$\frac{\partial U}{\partial t}+\frac{\partial F}{\partial \xi}=0$$

An implicit algorithm for time-marching this can be written as:

$$\frac{\partial U}{\partial t}+\delta_\xi^u F^{n+1}=\{\delta_\xi^u F^n-\delta_\xi^0 F^n\}$$

where δ_ξ^u is an appropriate one-sided first order accurate difference operator, δ_ξ^0 is the conventional second order accurate difference operator and $\{\ldots\}$ represents the explicit (ie known time-level) correction term.

In a converged steady solution, $\partial U/\partial t=0$ and $F^{n+1}=F^n$ so the desired result, $\delta_\xi^0 F=0$ is attained.

Clearly, as F is a non-linear function of U it must be linearized;

$$F^{n+1}=F^n+A^n\,\delta U^{n+1}$$

where

$$A=\partial F/\partial U$$

and

$$\delta U^{n+1}=U^{n+1}-U^n$$

So the implicit algorithm can be re-written as:

$$\frac{\delta U^{n+1}}{\Delta t}+\delta_\xi^u(F^n+A^n\,\delta U^{n+1})=\{\delta_\xi^u F^n-\delta_\xi^0 F^n\}$$

ie

$$[I+\Delta t\,\delta_\xi^u(A^n_\cdot)]\,\delta U^{n+1}=\Delta U^n\equiv-\Delta t\,\delta_\xi^0 F^n$$

Replacing $\delta_\xi^u(A^n_\cdot$ by eigenvalue-split operators, as in Eqs (23)–(25), results in the 1-D version of Eq (26); diagonalizing results in the 1-D version of Eq (30).

Application to the problem of shock-boundary layer interaction

To test the implicit algorithm it was applied to the problem of the interaction of a shock-wave with a laminar boundary layer. The flowfield is shown in Fig 2. An externally generated shock is incident on a laminar boundary layer developing on a flat plate. If the shock is strong enough the boundary layer will separate and, if the shock is not too strong, re-attach

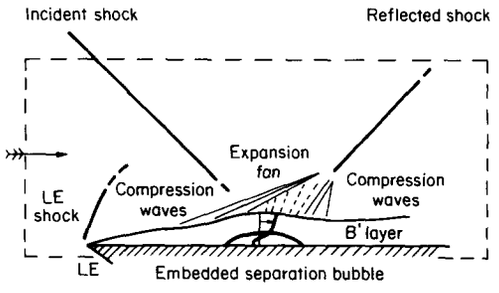

Fig 2 Sketch of shock–boundary layer interaction

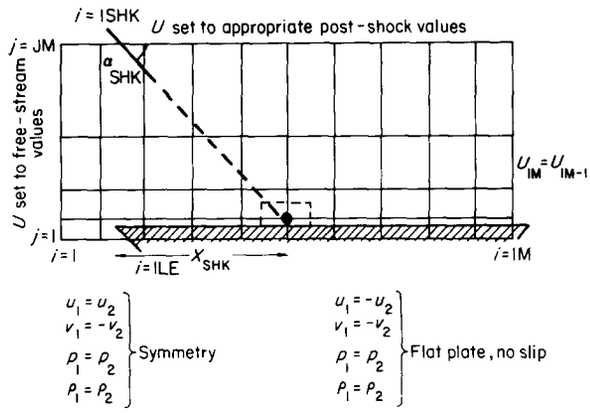

Fig 3 Computational mesh and boundary conditions

forming an embedded separation bubble. This flowfield is a good test for the calculation method.

The computation mesh and boundary conditions are shown schematically in Fig 3. The lower boundary is part plane of symmetry and part flat-plate and is aligned with cell faces rather than nodes so that boundary conditions are applied to *fluxes* of U rather than to U itself. Accordingly, the plane of symmetry boundary conditions were:

$$\left.\begin{aligned}u_{i1}&=u_{i2}\\v_{i1}&=-v_{i2}\\p_{i1}&=p_{i2}\\\rho_{i1}&=\rho_{i2}\end{aligned}\right\}\ 1<i<\mathrm{ILE}\qquad(39)$$

and the flat-plate boundary conditions were:

$$\left.\begin{aligned}u_{i1}&=-u_{i2}\\v_{i1}&=-v_{i2}\\p_{i1}&=p_{i2}\\\rho_{i1}&=\rho_{i2}\end{aligned}\right\}\ \mathrm{ILE}\leq i<\mathrm{IM}\qquad(40)$$

Thus, for example, the flat-plate boundary conditions simulate:

$$\left.\begin{aligned}u_{\mathrm{PLATE}}&=0\\v_{\mathrm{PLATE}}&=0\\\left(\frac{\partial p}{\partial Y}\right)_{\mathrm{PLATE}}&=0\\\left(\frac{\partial T}{\partial Y}\right)_{\mathrm{PLATE}}&=0\end{aligned}\right\}\qquad(41)$$

which are appropriate conditions for a no-slip adiabatic wall in high Reynolds number flow. At the inflow boundary U_{1j} is maintained fixed at the appropriate free-stream values. Along the outer flow boundary, $U_{i\mathrm{JM}}$ is held constant at either free-stream values (for $1\leq i<\mathrm{ISHK}$) or the appropriate post-shock values (for $\mathrm{ISHK}<i<\mathrm{IM}$). Finally, at the downstream outflow boundary zero order extrapolation ($U_{\mathrm{IM}j}=U_{\mathrm{IM}-1j}$) was used as the exit flow is either parabolic (near the plate) or hyperbolic (away from the plate) in the stream-wise direction. In addition δU_{BC} was set equal to zero along each of the four boundaries, as discussed previously.

The initial conditions consisted of uniform flow in the interior of the computational domain with the flow variables at the top mesh boundary set to either free stream values, or the appropriate post-shock values thereby generating a shock directed towards the plate.

The mesh consisted of 32×45 points equally spaced in the X-wise direction with $\Delta X/X_{SHK} = 0.0625$ and exponentially stretched away from the plate in the Y-wise direction with $\Delta Y_{MIN}/X_{SHK} = 0.000625$ near the plate and $\Delta Y_{MAX}/X_{SHK} = 0.0399$ near the outer flow boundary.

Two cases were computed, Case (A) without separation and Case (B) with an embedded separation bubble. The relevant parameters are given in Table 2. The incident shock in Case (B) is twice as strong as in Case (A). It was found necessary to filter the shortest wavelength components of the solution using a fourth-order dissipation term[9]. In both cases the time-step used was five times the maximum allowed for an explicit algorithm and the computations converged within 1000 time steps. Similar computations described in Ref 7 using an explicit algorithm required over 5600 time steps to achieve an acceptable level of convergence. There is evidence that if appropriate implicit boundary conditions were formulated for δU_{BC} (rather than setting $\delta U_{BC} = 0$) then larger time steps would be possible with a perhaps substantial reduction in number of time steps (and thus in computational time) to achieve convergence. Work is currently in progress to include implicit boundary conditions.

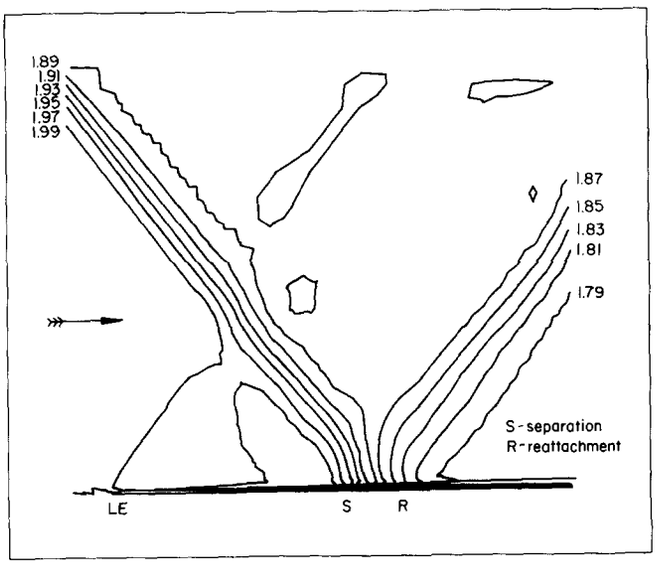

Fig 4 Case (B), Mach number contours: interval 0.02 for the shocks and 0.2 for the boundary layer (Note: vertical scale = 2 × horizontal scale).

Table 2

	M_∞	$Re_{X_{SHK}}$	α_{SHK}	Incident shock strength, $p_{s/p}$
Case (A)	2.0	0.284×10^6	31.35°	1.0963
Case (B)	2.0	0.296×10^6	32.59°	1.1868

Fig 4 shows Mach number contours computed for the case with separation, Case (B). A contour interval of 0.02 was used away from the plate to pick out the shock system and 0.2 near the plate to pick out the boundary layer. The incident and reflected shock are clearly visible, albeit smeared. No special provision is made for shocks in the algorithm; they are captured automatically as the computation proceeds. Also present is the weak leading edge shock. The growth of the boundary layer is apparent and the separation bubble formed by the shock impingement can be seen (although with some difficulty due to the reduced scale of this Figure).

The variations of static pressure and skin friction along the surface of the plate as calculated by the present method are shown in Fig 5 for Case (A) (without separation) and in Fig 6 for Case (B) (with the embedded separation bubble). Shown for comparison are experimental measurements made by Hakkinen[11] and computations made by MacCormack and Baldwin[12]. The experimental skin friction probes were unable to measure skin friction in the separated region other than to show that it was zero or negative. The small discrepancies between the predictions of the present method and the measurements in the region of the separation bubble are attributed to too coarse a grid being used in the X-wise direction reducing the streamwise resolution. A certain amount of discrepancy between the calculated and measured results must be expected near the leading edge of the plate because the boundary layer is, of course, very thin there and difficult to resolve on an economical grid. In general, however, the predictions of the present method are entirely consistent with experimental and other numerical results.

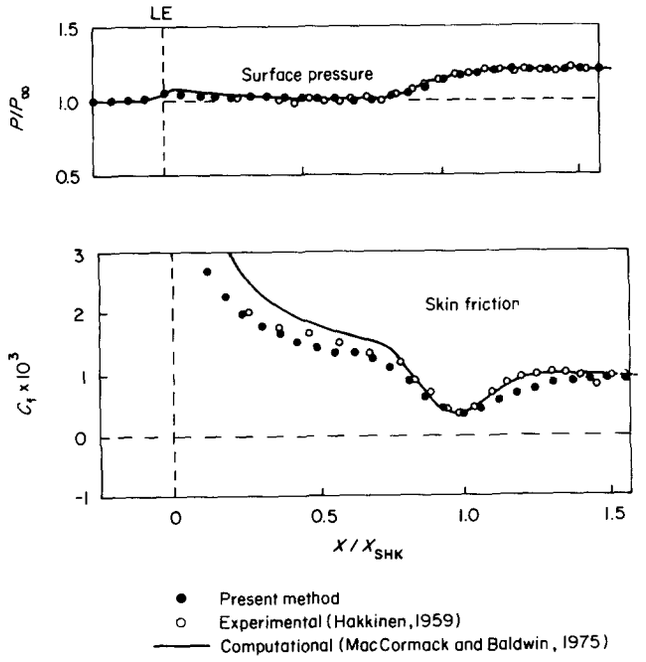

Fig 5 Case (A), surface pressure and skin friction. Mach number = 2.0 $Re_{x_{SHK}} = 0.284 \times 10^6$

It is appropriate here to make some remarks about this test case. Firstly, simple boundary layer theory is unable to cope with embedded separation bubbles because the classical boundary layer equations are parabolic and become ill-posed beyond a separation point. The Navier–Stokes equations, however, retain the full elliptic influence of the pressure field and thus 'capture' separations without numerical difficulty. Secondly, a shock impinging on a boundary layer loses its Rankine–Hugoniot form and this can no longer be dealt with using classical shock-expansion theory. Solving the full Navier–Stokes equations in conservation law form allows such shocks to be 'captured' automatically, with no prior knowledge of the flowfield.

Clearly the test-case presented here is relatively straightforward, particularly in terms of geometrical complexity. However, the derivation and coding of the method remains valid for quite general two-dimensional geometries, provided the physical plane is topologically rectangular (as indicated in Fig 1). Currently, the method presented in this paper, and derivatives of it, are being applied to transonic two-dimensional blade-to-blade flows in turbomachinery.

Conclusions

(1) A fully implicit algorithm has been derived to time integrate the equations of 2-D compressible viscous flow.

(2) The algorithm has been very efficiently constructed and requires virtually no extra computer storage and only a factor of two more computer time per point per time step than similar explicit algorithms, but the implicit algorithm has more favourable stability properties.

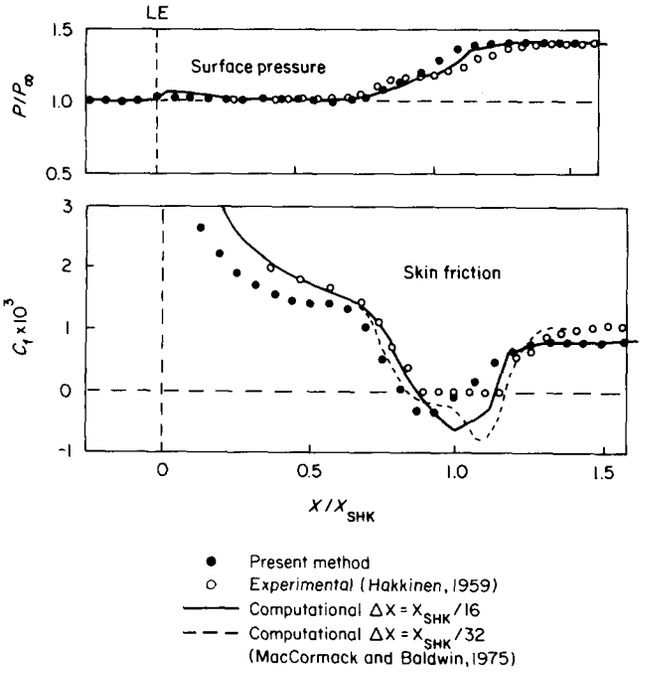

Fig 6 Case (B), surface pressure and skin friction. Mach number = 2.0 $\mathrm{Re}_{x_{SHK}} = 0.296 \times 10^6$

(3) Application of the algorithm to the problem of a shock–laminar boundary layer interaction produces results consistent with both experimental measurements and other calculations.

Acknowledgements

This work was carried out at Marchwood Engineering Laboratories and the paper is published by permission of the Central Electricity Generating Board.

References

1. **Lapidus A.** A Detached Shock Calculation by Second Order Finite Differences. *J. Comput., Phys., 1968,* **2**, *154–177*
2. **Viviand H.** Formes Conservatives de equations de la dynamique des gaz. *La Recherche Aerospatiale, 1974,* **1**, *65–68*
3. **Steger J. L.** Implicit Finite Difference Simulation of Flow about Arbitrary Geometries with Application to Air Foils. *AIAA Paper, 1979, 77–665*
4. **Denton J. D.** Transonic Flows in Axial Turbomachinery: Extension of the Finite Area Time-marching Method to 3D. *VKI-LS-84, 1975*
5. **Roache P. J.** Computational Fluid Dynamics, *Hermosa Pub, Albuquerque, 1972*
6. **Dawes W. N.** *Internal CEGB Memorandum, 1980*
7. **MacCormack R. W.** A Numerical Method for Solving the Equations of Compressible Viscous Flow., *AIAA Paper, 1981, 81–0110*
8. **Kutler P.** Supersonic Flow over Ablated Nose Tips using an Unsteady Numerical Procedure. *AIAA Paper, 1978, 78–213*
9. **Beam R. M. and Warming R. F.** On the Construction and Application of Implicit Factored Schemes for Conservation Laws. *SIAM-AMS Procs, 1978,* **11**, *85–129*
10. **Chausee D. W. and Pulliam T. H.** Two Dimensional Inlet Simulation using a Diagonal Implicit Algorithm. *AIAA J. 1981, 19, 153–159*
11. **Hakkinen R J.** *et al.* The Interaction of an Oblique Shock Wave with a Laminar Boundary Layer, *Memo 2–18–59W, NASA, 1959*
12. **MacCormack R. W. and Baldwin B. G.** A Numerical Method for Solving the Navier–Stokes Equations with Application to Shock-Boundary Layer Interactions. *AIAA Paper, 1975, 75–1*
13. **Jacobs, D. A. H.** Iterative Replacement Schemes for Complicated Banded Difference Equations. *CEGB Report RD/L/N169/75*
14. **Rubin, S. G. and Khosla, P. K.** Navier–Stokes Calculations with a Coupled Strongly Implicit Method—I. *Computers and Fluids, 1981,* **9**, *163–180*

Appendix

Diagonalizing similarity transformations of the gas dynamics matrices

The similarity transformations which diagonalize the Jacobians A and B are (Ref (10)):

$$A = T_\xi \Lambda_\xi T_\xi^{-1}, \qquad B = T_\eta \Lambda_\eta T_\eta^{-1}$$

where:

$$\Lambda_\xi = D[\hat{u}, \hat{u}, \hat{u}+c(\xi_x^2+\xi_y^2)^{1/2}, \hat{u}-c(\xi_x^2+\xi_y^2)^{1/2}]$$

$$= \begin{bmatrix} \hat{u} & 0 & 0 & 0 \\ 0 & \hat{u} & 0 & 0 \\ 0 & 0 & \hat{u}+c(\xi_x^2+\xi_y^2)^{1/2} & 0 \\ 0 & 0 & 0 & \hat{u}-c(\xi_x^2+\xi_y^2)^{1/2} \end{bmatrix}$$

$$\Lambda_\xi = D[\hat{v}, \hat{v}, \hat{v}+c(\eta_x^2+\eta_y^2)^{1/2}, \hat{v}-c(\eta_x^2+\eta_y^2)^{1/2}]$$

$$T_k = \begin{bmatrix} 1 & 0 & \alpha & \alpha \\ u & \tilde{k}_y\rho & \alpha(u+\tilde{k}_x c) & \alpha(u-\tilde{k}_x c) \\ v & -\tilde{k}_x\rho & \alpha(v+\tilde{k}_y c) & \alpha(v-\tilde{k}_y c) \\ \frac{\Phi^2}{\gamma-1} & \rho(\tilde{k}_y u-\tilde{k}_x v) & \alpha\left(\frac{\Phi^2+c^2}{\gamma-1}+c\tilde{\theta}\right) & \alpha\left(\frac{\Phi^2+c^2}{\gamma-1}-c\theta\right) \end{bmatrix}$$

$$T_k^{-1} = \begin{bmatrix} (1-\Phi^2/c^2) & (\gamma-1)u/c^2 & (\gamma-1)v/c^2 & -(\gamma-1)/c^2 \\ -(\tilde{k}_y u-\tilde{k}_x v)/\rho & \tilde{k}_y/\rho & -\tilde{k}_x/\rho & 0 \\ \beta(\Phi^2-C\theta^2) & \beta(\tilde{k}_x c-(\gamma-1)u) & \beta(\tilde{k}_y c-(\gamma-1)v) & \beta(\gamma-1) \\ \beta(\Phi^2+C\tilde{\theta}) & -\beta(\tilde{k}_x c+(\gamma-1)u) & -\beta(\tilde{k}_y c+(\gamma-1)v) & \beta(\gamma-1) \end{bmatrix}$$

with:

$$c = (\gamma p/\rho)^{1/2}$$

$$\hat{u} = \xi_x u + \xi_y v \qquad \hat{v} = \eta_x u + \eta_y v$$

$$\alpha = \rho/(c2^{1/2}) \qquad \beta = 1/(\rho c 2^{1/2})$$

$$\tilde{\theta} = \tilde{k}_x u + \tilde{k}_y v$$

$$\Phi^2 = \tfrac{1}{2}(\gamma-1)(u^2+v^2)$$

and:

$$\tilde{k}_x = k_x/(k_x^2+k_y^2)^{1/2}, \quad \tilde{k}_y = k_y/(k_x^2+k_y^2)^{1/2}$$

Relations exist between T_ξ and T_η of the form:

$$\hat{N} = T_\xi^{-1} T_\eta, \qquad \hat{N}^{-1} = T_\eta^{-1} T_\xi$$

where:

$$\hat{N}^j = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & m_1 & -j\mu m_2 & j\mu m_2 \\ 0 & j\mu m_2 & \mu^2(1+m_1) & \mu^2(1-m_1) \\ 0 & -j\mu m_2 & \mu^2(1-m_1) & \mu^2(1+m_1) \end{bmatrix}$$

with:

$$m_1 = (\tilde{\xi}_x\tilde{\eta}_x + \tilde{\xi}_y\tilde{\eta}_y), \; m_2 = \tilde{\xi}_x\tilde{\eta}_y - \tilde{\xi}_y\tilde{\eta}_x$$

$$\mu = 1/2^{1/2}$$

$j=1$ for $\hat{N}$ and $j=-1$ for $\hat{N}^{-1}$.