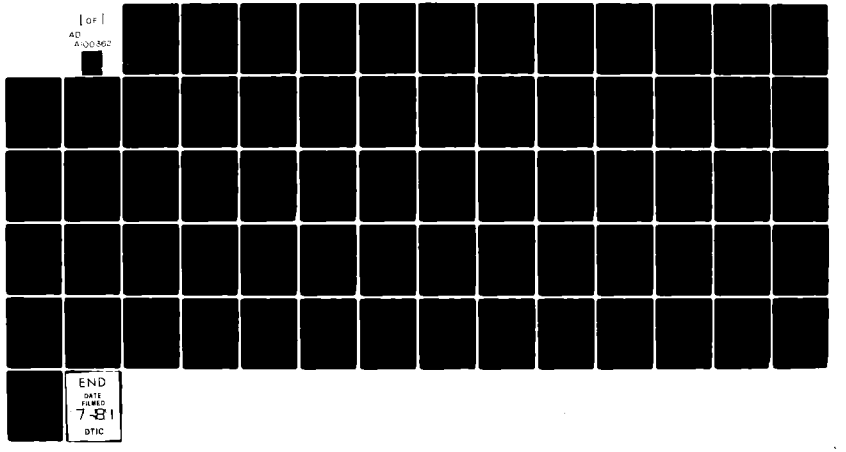
AD-A100 362 COLORADO UNIV AT BOULDER DEPT OF ASTRO-GEOPHYSICS F/G 8/8
NUMERICAL TECHNIQUES FOR OCEAN FORECASTING.(U)
SEP 80 E GRAHAM N00014-78-C-0706
UNCLASSIFIED NL



END
DATE FILMED
7-81
DTIC

AD A100362

UNIVERSITY OF COLORADO, BOULDER
Department of Astro-Geophysics

LEVEL II

FINAL REPORT

to the

Naval Ocean Research and Development Activity
NSTL Station, MS 39529

and to the

Office of Naval Research

for work carried out under

NUMERICAL TECHNIQUES FOR CLAN FORECASTING

CONTRACT N00014-78-C-0706

to the University of Colorado at Boulder

Contract Interval: 1 July 1978 to 30 September 1980

Principal Investigator: John E. Hart
Department of Astro-Geophysics
University of Colorado
Boulder, CO 80309
303/492-8568

Report prepared by Eric Graham
in a form requested by the Scientific Officer, Steve Piacsek, NCRDA

DTIC FILE COPY

DTIC
ELECTE
JUN 1 8 1981
C

Report is unclassified and available for unlimited circulation.

1 INTRODUCTION

The general equations that describe the flow of the ocean are well known, but for many cases of interest, analytical solutions are not possible because of irregular boundaries or significant non-linearities in the equations. It is often possible to obtain useful approximate solutions using numerical techniques. However, for reasonably detailed numerical models, very large amounts of computer time are required, and it becomes imperative to seek out the most efficient numerical algorithms. In the case where forecasts are being made on a real time basis, the use of non-optimum schemes may result in unacceptable delays.

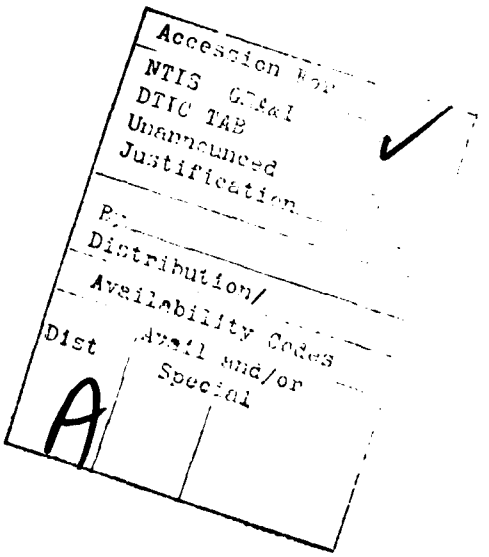
This report describes a class of techniques for the efficient treatment of bodies of water with irregular boundaries. Realistic models of the ocean state must include adequate treatment of the boundaries, the irregular bottom topography, the shape of the coastline and the complicated geometry of groups of islands.

Traditional numerical methods using a rectangular finite difference grid are suitable for exploratory surveys of idealised problems where the aim is to see why a particular system responds as it does, and what effects variations in the parameters of the problem and the forcing terms have on the outcome. In order to use these models for complicated geometries, it is necessary to use a very fine grid, and a

corresponding increase in computer time, or suffer from reduced accuracy.

The methods described here employ an irregular triangular grid, instead of a regular rectangular grid, so it is possible to fit empirical boundaries with high precision without using prohibitively fine grids throughout the domain of the solution. The penalty for this is a slight increase in the complexity of the algorithm.

Triangular grids have been extensively used for elastic and plastic flow problems with the finite element method (see for example, Strang and Fix, 1973). These grids have also been used in Lagrangian formulations of the equations of hydrodynamics by Crowley (1971), Boris et al (1975) and Fritts (1976), but the Lagrangian method is most suitable for flows where the total deformation of the fluid is small. Eulerian calculations have been performed by Sadourny et al (1968), Williamson (1968) and Thacker (1977).



2 THE CODE

This report describes both a methodology for constructing and using irregular triangular grids for solving ocean flow problems, and a set of computer programsfor implementing, testing and evaluating the methods. The complete set of subroutines is given later as an appendix. Certain details of the basic algorithm are best understood by reference to the code itself. The code has been designed to be highly modular, so that some effort must be expended in describing the interface to each module.

To maximise the flexibility of the code, it has been written in a superset of FORTRAN which includes macro expansions. By the use of the file inclusion macro, $INCLUDE, common block maintenance is greatly facilitated, while the bulk of the source code is reduced. Many compilers support some form of file inclusion. The other use of macros in the code is to support parameters. Parameters are used for values which must take the form of constants, for example to dimension an array. The values may need to be changed to produce different versions of the code, for example to be able to change the number of grid points or to handle different hardware configurations. The easiest way to be able to employ macros is to use a simple pre-processor, which produces as output a standard FORTRAN program, which any compiler can then accept. Alternatively, a text editor may be used to perform the text

substitutions implied by the macros.

In the work that is described here, a simple macro preprocessor based on that of Kernighan and Plauger (1976) was used.

The following predefined macros are assumed:

$MACRO(NAME,VALUE)	Define a new macro.
$INCLUDE(filename)	Include a file in the source code at this point.
$DELTOK	Delete the next input token, normally a new line character.

The remaining macros are defined in the code and have the following uses:

SMAX	Size of array S.
IVMAX	Size of array IV.
ICMAX	Size of array IC.
SERAS	Size of array ERAS.

MAPMAXH	Horizontal page size in character widths.
MAPMAXV	Vertical page size in character heights.
OUTLUN	Logical unit number for the output device.
CHARACTER	Type for character datum.

3 THE GRID

The computational grid consists of an irregular triangular tesselation which is tailored to approximately correspond to the shape of the ocean basin and any islands that are to be modelled. In order to be able to conveniently handle such a grid on a computer, we must formulate a data structure that represents the connectivity and the geometry of the grid. First let us introduce two definitions.

A vertex is defined as the point of intersection of a number of grid lines.

A cell is a triangular region bounded by three grid lines.

Next we must refer specifically to the code (given as an appendix) in order to see how the data structure is organised. Cell connectivity information is held in the array IC and vertex connectivity information is held in the array IV.

An integer expression I is said to point to a cell if IC(I) is the first word describing the cell, similarly an expression J is said to point to a vertex if IV(J) is the first word describing the vertex.

In addition, a number of words of real storage are associated with each cell and vertex for storing physical quantities. These are held in the

array S.

The first two words of storage for vertices are assumed to be the coordinates of the vertex. Where a vector field is to be represented, adjacent words in S are used to hold the components of the vector.

As an example, consider the shallow water equations. The physical storage layout associated with vertices is given below:

Word	
0,1	Coordinates of the vertex.
2-8	Weights for lumping.
9-15	Transport weights in the x-direction.
16-22	Transport weights in the y-direction.
23	Depth.
24,25	Momentum.
26-28	Momentum flux.
29-31	Lumped depth and momentum field A.
32-34	Lumped depth and momentum field B.
35-37	Lumped depth and momentum field C.
38,39	Force.
40,41	Coriolis term.

The meanings of these fields are described elsewhere.

The layout of connectivity information for a cell is as given below:

IC(I)	Pointer to the S array.
IC(I+1)	Pointer to the 1st vertex that defines the cell.
IC(I+2)	Pointer to the 2nd vertex that defines the cell.
IC(I+3)	Pointer to the 3rd vertex that defines the cell.

The layout of connectivity information for a vertex is given below:

IV(I)	Pointer to the S array.
IV(I+1)	Number of adjacent vertices, N.
IV(I+2)	Pointer to the 1st adjacent vertex.
IV(I+3)	Pointer to the 2nd adjacent vertex.
IV(I+4)	Pointer to the 3rd adjacent vertex.
IV(I+5)	Pointer to the 4th adjacent vertex.
IV(I+6)	Pointer to the 5th adjacent vertex.
IV(I+7)	Pointer to the 6th adjacent vertex, or link to next boundary vertex.

The grid is assumed to have the same topology as a tessalation of equilateral triangles. Arbitrary boundaries may be imposed subject to the constraint that interior boundaries (islands) do not have acute angles. Since the grid is deformed before use, this does not impose any restriction on the actual shapes of islands that may be handled, however the grid would be more than usually deformed in the neighborhood

of such acute features.

At this point it is appropriate to examine portions of the code in detail, so we will examine the subroutines that play an important part in constructing the grid data structure.

HEXTOP is a subroutine that constructs a grid with the same topology as a hexagonal region divided into triangular cells. It is called by:

```
CALL HEXTOP(N)
```

Each side of the hexagon is divided into N segments.

By inspection, we may see that the grid has 6 N triangular cells, 3(3N+1)N edges and 3(N+1)N+1 vertices.

HEXTOP generates the pointers to vertices that are stored in IC, by using the subroutines GUC and GDC which generate 'upward' and 'downward' pointing triangular cells. HEXTOP counts the total number of cells allocated in NUMC, and the number of vertices in NUMV.

Next, HEXTOP generates the links in each vertex data structure, by examining each cell and making sure that all vertices surrounding a cell are joined together by using the subroutine VLINK.

Finally, HEXTOP determines which vertices lie on the boundary of the region by counting the number of adjacent vertices. The boundary points are contained in a linked list data structure.

HEXTOP is designed as a specific tool for generating a class of topologies that are useful for investigating the properties of advection schemes and in simulations of any unbroken region that has approximately circular boundaries. In addition HEXTOP illustrates the methods that may be used to generate even more complicated topologies, such as grids with imbedded holes. From the human engineering standpoint, an interactive computer system with a display screen and light pen would be more convenient for generating grids from real ocean maps.

GDC and GUC are cell generating subroutines for downward and upward pointing cells in a triangular tesselation. A downward pointing cell is generated by the call:

```
CALL GDC(IP)
```

where IP points to the upper left hand vertex. It is assumed that the following vertex, IP+IVSIZ, is the upper right hand vertex and that the most recently defined vertex is the lower vertex. After generating the cell-vertex links for a new cell, GDC returns after incrementing IP to the next vertex.

GUC works in a complementary fashion.

A call

```
CALL GUC(IP)
```

assumes that IP points to the upper verte of a cell. The most recent vertex is assumed to be the lower left vertex, so GUC generates a new vertex for the lower right hand vertex and constructs a new set of cell-vertex pointers. IP is left unchanged.

By alternating calls to GUC and GDC it is simple to construct an arbitrary tesselation of triangular cells, with all the required pointers to define the topology of the grid.

VLINK is a subroutine that ensures that two vertices are linked together in the order given. It is called as follows:

```
CALL VLINK(IVA,IVB)
```

Vertex IVA is examined to see if it is already marked as having IVB as a neighbor, if so VLINK exits without doing anything extra. Otherwise the vertex count for IVA is incremented and a link to IVB is placed in the next available vertex link position. To correctly link two vertices, two calls to VLINK are required:

```
CALL VLINK(IVA,IVB)
CALL VLINK(IVB,IVC)
```

The subroutine VSTO assigns storage space to the vertices from the array of storage S. It is called as follows:

```
CALL VSTO(IORIG,NWDS)
```

The first word allocated is S(IORIG) and a total of NWDS words are assigned for each vertex. NWDS should be equal to the number of physical variable fields required in the solutions of the equations of hydrodynamics.

CIRBND is an example of a procedure that fixes the co-ordinates of the boundary vertices, in this case by distributing them uniformly in a circle. It may be called without any arguments.

CIRBND is used with HEXTOP and SUGRID to define a circular ocean basin for testing purposes.

An assumption is made in CIRBND that all boundary vertices lie on a single exterior boundary, and that the links between boundary vertices follow serially around the boundary. This is true for HEXTOP, but need not be so for other topology generation subroutines. A more general

boundary fixing routine would be needed for multiple boundaries, and shouldprobably be embedded in the framework of an interactive computer system, as discussed elsewhere.

VCDUMP is a subroutine for printing out the links associated with each vertex and cell. It is called without any arguments. This routine has proved useful as a diagnostic during the debugging phase of new topology generation subroutines. It is also invoked when code detects an inconsistency in the links, caused by a logic error in the topology routine, or more usually by storage corruption caused by array bound overflow or subroutine argument inconsistencies, which traditionally are not detected by FORTRAN compilers.

The last subroutine in the grid building suite is SUGRID, which calculates the co-ordinates of all the interior vertices. It is called as follows:

```
CALL SUGRID(NITER)
```

Interior vertices are moved so that the co-ordinates of each vertex is equal to the average of those of all its neighbors. This involves solving a system of 2N linear equations, where N is the number of interior vertices.

The equations are amenable to a relaxation process of the most

straightforward kind. A relaxation parameter RELPA is used to improve the convergence of the process.

Empirical tests with simple grids of various sizes have shown that RELPA = 1.388 seems to be close to the optimum value.

A total of NITER iterations is performed, and for each iteration, the maximum displacement of a vertex is printed, in order to illustrate the convergence of the iterative solution.

On the basis of experience with other relaxation methods applied to elliptical systems of partial differential equations, it is believed that SUGRID is unconditionally stable for a finite range of RELPA.

The suite of subroutines that has been described above has been specifically written to construct the data structure for a circular ocean basin, but simple modification to CIRBND, for example would permit irregular quasi-circular basins to be treated. If the basin was grossly dissimilar to a circle, or had a different topology, for example, containing islands, then the routine HEXTOP would have to be replaced. The following algorithm is proposed for the generation of grids for real-world oceanographic simulations:

First assemble a number of equilateral triangles to approximately represent the region of interest. For example, HEXTOP uses six

triangles in the form of a hexagon to represent a circle.

The next step is to subdivide the triangles into a tesselation of smaller equilateral triangles until it is estimated that there are sufficient cells to resolve the solution structure that is desired.

Triangular cells are next removed from the interior to represent islands and from the exterior to better represent the ocean shore. The vertices on the exterior and interior boundaries are then assigned the actual coordinates of associated points on the actual coastline.

The final step is to relax the interior points to obtain a grid with a smooth transition of cell size. The whole procedure would best be done on a computer system under interactive command. A video graphics display and a lightpen would be most appropriate for adjusting the coastline points in order make the grid as uniform as possible.

4 THE ALGORITHM FOR HYDRODYNAMICS

In most fluid dynamics simulations the most troublesome terms in the equations are the non-linear ones. The advection of a scalar field may be used illustrate the algorithm that is used for more complicated cases.

The governing equation for a scalar field is:

$$\frac{\partial \phi}{\partial t} + \nabla . F = 0$$

Alternatively, by Stokes' theorem, we can say that for any region 1, bounded by a curve S, we have

$$\frac{\partial \bar{\phi}_1}{\partial t} + \int_S \underline{F} . \underline{ds} = 0$$

where $\underline{ds}$ is normal to the curve S, and $\bar{\phi}_1$ is the average value of ϕ in the region 1.

In a finite representation of ϕ using a grid of points on a triangular mesh, it is convenient to store the value of ϕ at the vertices of the grid, and to consider a region surrounding each vertex which we will

call a flux cell.

A flux cell is an irregular and sometimes non-convex polygon that contains only one vertex. Each side of the polygon starts at the centroid of a cell adjacent to the contained vertex and finishes on the mid-point of a side of the cell that passes through the contained vertex. It is clear that no part of the region defined by the grid lies outside of all flux cells.

A flux cell surrounding a vertex with N neighboring vertices is a polygon with 2N sides.

Averages of a variable within a flux cell are termed 'lumped' values and are associated with the contained vertex. If the values of the variable are to be represented by its values at the vertices only, as is usual for any finite difference scheme, then we may uniquely approximate the variable at any point by linear interpolation between the vertices of the cell containing the point. With this representation of the variable, we may exactly calculate lumped values by integrating over the flux cell. Similarly, the line integral above, may be exactly evaluated around the flux cell.

Our advection equation is conservative in the lumped values of the scalar variable. Fluxes are derived by first evaluating the unlumped values at vertices, evaluating the non-linear flux terms at each vertex

and then performing linear interpolation.

Thus, we have a simple, non-ambiguous algorithm for discretising the continuous equations of hydrodynamics.

The line and surface integrals of interpolated fields may be exactly represented by a weighted sum of vertex values in the vicinity of the region of integration. The weights do not change with time, so the complicated integrals may be simply evaluated with a minimum of computational effort.

The subroutine CVLW calculates the weights A_i, B_j required to calculate the vertex lumped values $\bar{F}_i$

$$\bar{F}_i = A_i F_i + \sum_j F_j B_j$$

where j is a vertex which neighbors vertex i.

Consider a vertex i and a cell k. The field F is defined at i and the other two vertices p and q such that the cell k is defined by the triangle ipq. If we approximate F within k by linear interpolation, we may integrate F within the quadrilateral $ip'gq'$ where q' is the midpoint of ip and p' is the midpoint of ip, and g is the centroid of ipq.

This integral is

$$\phi_k = \Delta_k [\ 11/54\ F_i + 7/108\ F_p + 7/108\ F_q]$$

where Δ_k is the area of cell k.

The vertex lumped value of F at i is the sum of ϕ_k over all quadrilaterals as above which have i as a vertex. The weights A and B_j may thus be seen to be

$$A = 11/54 \sum_k \Delta_k$$

$$B_j = 7/108 \sum_k \Delta_{jk}$$

where $\Delta_{jk} = \Delta_k$ if j is a vertex of k, else $\Delta_{jk} = 0$.

CVLW is invoked as follows:

```
CALL CVLW(IPOS)
```

IPOS is a pointer to a set of 7 variable locations which are to contain the values of A and B_j.

The subroutine CIW computes the weights A and B_j to calculate the line integral I of a vector flux F around a flux cell surrounding a vertex i,

where

$$T = \underline{F}_i.\underline{A} + \sum_j \underline{F}_j.\underline{B}_j$$

and j is a vertex neighboring i.

$\underline{A}$ and $\underline{B}_j$ are vectors associated with vertex i and are stored with their x-components at IPOS and IPOS+J respectively. The y-components are stored 7 locations after the corresponding x-component.

The flux cell around a boundary vertex is truncated by the actual boundary, so that part of the line integral must be evaluated along sections of the boundary, however, straightforward use of linear interpolation suffices to calculate $\underline{A}$ and $\underline{B}_j$. The details of the algorithm need not be described here, since the code itself serves to define the method.

The subroutine is called as shown:

```
CALL CTW(IPOS)
```

where, as before, IPOS is a pointer to the region of storage that is to contain the weights. In this case 14 words are required to store the vector weights.

The correctness of the algorithm may be tested by the subroutine TSTCTW, which evaluates the line integrals for a flux of (1,0) and (0,1) for the boundaries of the flux cells containing each vertex. Within the limits of rounding errors, the two integrals are zero.

TSTCTW is called as shown:

```
CALL TSTCTW(IPOS)
```

where IPOS is the pointer to the weights generated by CTW.

LUMP is the subroutine that calculates the integral of a physical variable field over the flux cell that surrounds each vertex. The subroutine LUMP is invoked as follows:

```
CALL LUMP(IPOS,IFROM,ITO)
```

IPOS is a pointer to the set of weights that have previously been calculated by CVLW, and IFROM and ITO are pointers to the source and destination fields respectively.

The lumped variable is stored at location ITO after calculating

$$\bar{F}_i = A.F_i + \sum_j F_j.B_j$$

where A and B_j are the weights and F_j is the value of the field at vertex j. j is a neighbor vertex of vertex i.

UNLUMP is a subroutine which is the formal inverse of LUMP. It will recover the original values of a field that has previously been smoothed by LUMP. It is invoked as shown:

```
CALL  UNLUMP(IPOS,IFROM,ITO,ERR)
```

IPOS, IFROM and ITO have the same meaning as in the above description of the subroutine LUMP.

The system of linear equations for F_i, given $\bar{F}_i$ is solved by relaxation methods, with a final accuracy estimated to be on the order of ERR if possible, otherwise an error message is printed.

The subroutine ADVECT is called as follows:

```
CALL ADVECT(IPCN,IPCL,IPC,IPU,IPW,DT)
```

This subroutine performs the basic advection of a lumped scalar field at location IPCL using the unlumped velocity whose x- and y-components are located at IPU and IPU+1. Transport weights are located at IPW. The time interval for integration is DT and the new scalar field is returned

to the location IFCN.

The subroutine ADVECT modifies a lumped momentum field to ensure that the corresponding momentum values on the grid boundaries are zero. This is equivalent to imposing a rigid boundary condition at the edge of the computational domain.

The subroutine STEP is called as follows:

```
CALL STEP(IPLN,IPULN,IPVLN,IPLO,IPULO,IPVLO,IPL,
IPUL,IPVL,IP,IPU,IPV,IPUU,IPUV,IPVV,
IPFX,IPFY,IPFXL,IPFYL,IPW,DT,F,CF,
WSX,WSY)
```

The STEP subroutine advances the fields of the physical variables one time step for the shallow water equations. The values of the dependent variables, height and horizontal momentum components, at the start of the step are denoted with a suffix 'O' for 'Old' while the values corresponding to the end of the step have a suffix 'N' for 'New'. STEP first unlumps the fields in order to calculate pressure and Reynolds stress terms, then uses the transport algorithm and finally imposes the boundary conditions.

The subroutine TRNSPT, which is called as shown:

CALL TRNSPT(INEW,IOLD,IFX,IFY,IPW,DT)

evaluates the changes to a lumped field for a time step DT. The original lumped field is located at IOLD. The field after modification for the effects of transportation is returned to location INEW. The transportation is effected by a flux field whose x- and y-components are located at IFX and IFY respectively.

The shallow water equations contain several terms which are not of an advective nature. These terms are generally easier to treat in a numerical scheme than the non-linear advective terms. We will refer to the non-advective terms as forcing terms. The subroutine FRF includes the effects of the following forcing terms:

1) Frictional term with a coefficient of friction Cf and a linear dependency with the fluid speed.

2) Coriolis term with a parameter F.

3) An externally imposed stress (WSX,WSY) which may be regarded as representing the effects of wind on the fluid.

The subroutine is called with the arguments

```
FRF(IPL,IPUL,IPVL,IP,IPU,IPV,IPFX,IPFY,IPFXL,
IPFYL,DT,F,CF,WSX,WSY)
```

IP, IPU and IPV are pointers to unlumped depth and momentum components. IPL, IPUL and IPVL are the locations of the corresponding lumped fields. IPFX and IPFY are fields used to store the combined frictional and external stresses. IPFXL and IPFYL are used to locate the lumped values of the previous two fields. Upon exit from FRF the momentum fields are modified to reflect the time integrated effects of the forcing terms over a time interval DT.

Two main program are included in the appendix. The first is used in simple test solutions for the color equation, in order to examine the stability and accuracy of the basic advection algorithm.

The second subroutine is a driver for the shallow water equations and includes calls to the graphics routines which are described in the next section. The subprogram makes two calls to STEP, since the basic time differencing scheme that is employed is the robust pseudo-backward Euler method.

5 OUTPUT ROUTINES

In debugging the code and evaluating the performance of the numerical scheme that it embodies, graphical output is an invaluable tool. Simple listings of field values give little indication of the actual behavior of the model being simulated, especially when an irregular grid is used. A number of routines have been produced, which serve in themselves as valuable aids, while at the same time forming a set of primitives for more elaborate displays.

The QPRCON subroutine is a simple display procedure for producing contour plots of a field on a hardware device such as a lineprinter. It is called as follows:

```
CALL QPRCON(IFLD,XMIN,XMAX)
```

IFLD is a pointer to the field that is to be displayed. A single character, either a digit or a plus or minus sign is placed on the page at a position corresponding to a vertex location. The character represents the value of the specified field, with 0 corresponding to XMIN and 9 corresponding to XMAX. Intermediate values are represented by an appropriate decimal value using a linear transfer function. Values less that XMIN are shown by a minus sign while values larger that XMAX are shown with a plus sign. If the grid is coarse, the display will be sparse, but QPRCON has the advantage of representing both the grid structure and the field, so that judgements may be made as to the

reality of features with scales near that of the basic grid.

PRCON is a subroutine that is functionally similar to QPRCON, having the same argument list, except that the field values for space between vertices are also displayed. Each character space on the output page is mapped into the grid, and if the point lies within the boundary of the grid, the value of the input field, IFLD, is evaluated by linear interpolation. The running time for PRCON may be much larger than that for QPRCON, especially when the the number of vertices is less that the number of resolvable positions on the output page.

The routine CONPLT generates a contour line plot of the specified field. It is called by means of:

```
CALL CONPLT(IFLD,CONVAL,N)
```

Contour values are printed for values CONVAL(I), I=1,N. CONPLT is suitable for all types of contour plotting devices such as pen plotters and electrostatic printer-plotters, since it makes use of only one system dependent subroutine, LINE, for drawing a straight line segment.

The DRAWV subroutine is called as follows:

```
DRAWV(IPL,IPUL,IPVL,IP,IPU,IPV,ASCALE,IPW)
```

The DRAWV subroutine generates a display of the velocity field implied by the lumped depth and momentum fields located at IPL, IPUL and IPVL. The fields IP, IPU and IPV are used as workspace for calculating unlumped values of the physical variables.

The COMMON variable SCALE is used to transform from grid coordinates to plotter coordinates. After unlumping the fields, the velocity components are extracted from the momentum fields. At each vertex of the grid, an arrow is drawn with a length proportional (using the scale factor ASCALE) to the velocity at the specified point. No attempt is made to draw arrows that are so small as to be unresolvable.

DRGRID is a subroutine that has no arguments but that may be used for drawing the triangular grid on a scale similar to the other plotting displays, so that it may be used to overlay contour or vector field representations.

The subroutine DRCELL is similar to DRGRID, also having no arguments, except that it draws the outline of the flux cells that are used for the line integrals in Stokes' theorem.

EWPLOT is a subroutine with the following calling sequence:

```
CALL EWPLOT(IP,ZMAX,ZMIN)
```

The EWPLOT subroutine is used to generate a plot of the field located at IP, along the line Y = 0. The actual plot is produced by printing characters, in a way suitable for use with a lineprinter. The maximum and minimum values that may be displayed on the page are given by ZMAX and ZMIN respectively.

INTPOL is the basic interpolation subroutine used by the other graphics subroutines for extracting field values from arbitrary positions on the grid by means of linear interpolation. It is called as follows:

```
CALL INTPOL(IFLD,X,Y,Z,FAIL)
```

The postion specified is given by the coordinates (X,Y) and the value of the field indicated by IFL is returned in the variable Z. If the point lies outside the boundary of the grid, the logical variable FAIL is set to betrue. INTPOL works by examining each cell in turn until one is found that contains the required point, so it is relatively time consuming, especially when it is called many times, as in the subroutine PRCON.

The routine could be increased in speed by introducing a relatively coarse rectangular grid and associating a linked list of triangular cell numbers with each rectangular cell. The search could then be performed on a much shorter list, the particular list being determined by direct

computation from X and Y. This would be analogous to hash coded searches in standard table look up problems, with the added advantage that each list would be approximately the same size, so the look up time would be both short and predictable.

INTRI is function that return a value of LOGICAL type. It may be referenced by:

```
INTRI(X,Y,ICELL)
```

The returned value is only true if the point (X,Y) lies inside, or on the boundary of the triangular cell number ICELL.

The subroutine MAXMIN, which is called by:

```
CALL MAXMIN(IFLD)
```

prints out the maximum and minimum values of the field specified by IFLD.

The function NEIGH returns a LOGICAL value if the two vertices specified by IP and IQ are neighbors when it is called by the sequence:

```
NEIGH(IP,IQ)
```

The subroutine PRFLDS, called by

```
CALL PRFLDS(I1,I2)
```

prints out the values of all the fields in the range from I1 to I2 in tabular form for each vertex. If needed the table is split into several parts so as not to exceed the width of the lineprinter page.

The VCDUMP subroutine displays, on the printer, all the links associated with the cell and vertex data structures. VCDUMP does not take any arguments.

CONSRV is a subroutine called as follows:

```
CALL CONSRV(IFROM)
```

This diagnostic routine evaluates and prints the sum of the values of a variable at each vertex of the grid. If the variable is a lumped field, the sum is the spatial integral of the corresponding unlumped field. The routine has proved itself useful in code testing by evaluating the integral of fields that should be formally conserved in time by the system of equations.

The COPY utility, which is called by:

```
CALL COPY(ITO,IFROM)
```

is used in many places throughout the code to transfer a physical variable field from one location IFROM to another location ITO.

The subroutine FLDAV is a utility procedure which is called by:

```
CALL FLDAV(IPOSA,IPOSB,IPOSC)
```

It calculates the average of two fields indicated by IPOSB and IPOSC and returns the result to the location IPOSA.

INITF is a subroutine used to set up initial fields of unit depth and zero momentum. It is called by:

```
CALL INITF(IP,IPU,IPV)
```

The subroutine SBROT imposes a velocity field appropriate for a solid body rotation with an angular velocity of OMEGA. The resulting components are stored at locations IPOSU and IPOSU+1 when it is called by:

```
CALL SBROT(OMEGA,IPOSU)
```

The TFIELD subroutine sets up a scalar test field which is a compact,

continuous and differentiable piecewise bivariate cubic, in the shape of an off-center 'hump'. This field has proved useful for studying the advection of a scalar field under solid body rotation. Although this problem is analytically trivial since the field merely rotates with the flow without changing shape, it is quite non-trivial computationally and provides a powerful test for comparing the efficacy of different numerical advection schemes. It may be invoked by the call:

```
CALL TFIELD(IPOS)
```

6 RESULTS

A sequence of numerical experiments were performed with the code during its development in order to provide answers to the following questions:

1) Can a simple algorithm be developed which is capable of solving the equations of hydrodynamics with the physical variables approximated by values on an irregular grid?

2) What are the limits of the stability of the numerical scheme?

3) What is the accuracy of the scheme, especially in the treatment of the dominant non-linear advective terms?

4) What computing resources are required to use the algorithm?

5) How does the algorithm compare to orthodox finite difference schemes?

The answer to question 1) is clearly yes. The scheme that has been described is certainly simple and elegant. Some slight complications are inherent in using an irregular grid, but all the geometrical and

topological factors are resolved once at the start of a simulation and are incorporated in the weight terms and the cell and vertex links. During execution the scheme is fully explicit, except for the simple relaxation scheme that is used for 'unlumping' the physical variable fields.

One especially satisfactory feature of the algorithm is that, for the treatment of spatial derivatives at least, there are no arbitrary decisions to be made in setting up the difference equations. Most finite difference schemes are overdetermined, in the sense that several different representations have similar spatial order of accuracy, with often no obvious means of resolution. Also the present algorithm may be readily generalised to more complicated situations.

The answer to question 2) is not capable of analytical solution, so we must perform experiments and make cautious inferences. Most finite difference schemes that are explicit in time differencing must have some constraint on the maximum time step in order to satisfy stability criteria. If these stability criteria are not satisfied then a 'computational mode' solution may be obtained. These frequently exhibit high spatial frequency oscillations and bear no resemblance to the solutions of the underlying differential equations.

Some numerical schemes, such as the Dufort-Frankel scheme, while being explicit in time, have no formal constraint on the time step. However

they achieve stability at the expense of introducing increasing amounts of artificial diffusivity as the time step is made larger. In addition, even schemes that make use of implicit time differencing, such as ADI methods are still in practice limited by the need to maintain adequate accuracy in the time integration.

A number of numerical tests have been performed both for the scalar advection code and the shallow water equations. For semi-regular grids the maximum time step for stability was observed to be inversely proportional to the mean inter-vertex spacing, while for more irregular grids the maximum stable time step was found to be more closely related to the inverse of the minimum inter-vertex spacing. The latter result is hardly suprising but it does mean that when constructing an irregular grid to follow a coastline of great complexity, it pays not to use gratuitously small cells. However if the boundary points are reasonably distributed along the coast, and the grid topology is appropriate to the computational domain the use of SUGRID or its generalisation will ensure that no excessively small inter-vertex spacings are generated.

Overall, it has been found that the stability behaviour of the triangular grid scheme is qualitatively and quantitatively similar to more conventional rectangular grid finite difference schemes.

Some answers to question 3) have been obtained by solving the color equation for a solid body velocity field. Such an equation has a

trivial analytical solution so we may investigate the accuracy of the scheme in its treatment of the all important advection term. For a circular basin of unit radius, a solid body rotation law and an initial field determined by the subroutine IFIELD, good results were obtained even with a very coarse grid. For example with a typical inter-vertex spacing of .125, the bell shaped test field only suffered about 10% error after a complete circuit of the grid. Since positivity is not ensured by the scheme, some small negative and positive ripples were observed, especially in the wake of the "hump", but these always remained small in magnitude. The algorithm has second order spatial accuracy so in the limit of fine grids, error estimates may be easily calculated.

Question 4 may be answered both on theoretical estimates and on the results of the numerical experiments that have been performed. If we take as a measure of the grid complexity some number N, for example NUMDIV as used in the code, or the reciprocal of the inter-vertex spacing. Then the number of grid points for which calculations must be performed is N**2. The solution of the system of linear equations for the unlumping operation requires on the order of N**3 operations, since it is iterative, and information must be propagated over the whole grid in order to effect a solution. For each time step, the computational effort has two factors, one proportional to the square of N and the other proportional to the cube of N. For moderate values of N up to about 20, the first term is dominant. For fine scale simulations,

however, the effort of the UNLUMP operation will dominate the solution. The maximum permitted time step varies inversely with N, so a complete simulation will take a time that is proportional to the cube of N for coarse grids and the fourth power for sufficiently fine grids.

There is no clear cut answer to the final question. Some cartesian grid methods maintain N**3 timing, and so for very fine grids, will be more efficient than the irregular triangular grid method. On the other hand, irregular boundaries may require excessively fine grids for the cartesian grid methods, so their theoretical edge may not be maintained in practice. It has been demonstrated that irregular triangular grid methods can be applied to the equations of ocean flow, without undue penalties in computer resources, or complexity of the final code.

7 REFERENCES

Boris, J.P., Hain, K. L. and Fritts, M.J., 1975: "Free surface hydrodynamics using a Lagrangian triangular mesh". Proc. First Intern. Conf. Numerical Ship Hydrodynamics, 20-23 October, David W. Taylor Naval Ship Research and Development Center.

Crowley, W.P., 1971: "FLAG: A free-Lagrange method for numerically simulating hydrodynamic flows in two dimensions". Proc. Second Inter. Conf. Numerical Methods in Fluid Mechanics. Springer Verlag.

Kernighan, B. W. and Plauger, P. J., 1976: "Software tools". Adison Wesley, 338pp.

Strang, G. and Fix, F. J., 1973: "An analysis of the finite element method". Prentice-Hall, 306pp.

Sadourny, R., Arakawa, A. and Mintz, Y., 1968: "Integration of the non-divergent barotropic vorticity equation with an icosahedral hexagonal grid for the sphere". Numerical Simulations of Weather and Climate Technical Report number 2, U.C.L.A.

Thacker, W.C., 1977: "Irregular grid finite difference techniques: Simulation of oscillations in shallow circular basins". J. Phys. Oceanography 7, 284.

Appendix

Typical set of macro definitions

```
$MACRO(^,[$DELTOK])^
$MACRO(SMAX,6510)^
$MACRO(IVMAX,1736)^
$MACRO(ICMAX,1668)^
$MACRO(MAPMAXH,72)^
$MACRO(MAPMAXV,22)^
$MACRO(SERAS,1)^
$MACRO(OUTLUN,5)^
$MACRO(INLUN,5)^
$MACRO(CHARACTER,LOGICAL *1)^
```

Main program for driving the scalar advection code

```
$INCLUDE(CMAC.MAC)$DELTOK
      TYPE 1
    1 FORMAT(' N,DT')
      ACCEPT *,N,DT
      WRITE(OUTLUN,2) N,DT
    2 FORMAT(' N,DT',I4,F12.5)
      CALL CTEST(N,DT)
      STOP
      END
      SUBROUTINE CTEST(NUMDIV,DT)
      WRITE(1,96) NUMDIV,DT
   96 FORMAT(' N,DT',I6,F14.6)
      OMEGA=6.283185
      NSTEPS=1./DT
      CALL HEXTOP(NUMDIV)
      CALL VSTO(1,30)
      CALL CIRBND
      CALL SUGRID(20)
      IPLW=2
      CALL CVLW(IPLW)
      IPW=9
      CALL CTW(IPW)
C     CALL TSTCTW(IPW)
      CALL GRDLEN
      IP=23
      IU=24
      IV=25
      IPL=26
      IPN=27
      IUL=28
      IVL=29
      CALL TFIELD(IP)
      CALL SBROT(OMEGA,IU)
      CALL LUMP(IPLW,IP,IPL)
C     Remove the Cs from the next three lines to impose rigid b.c.
C     CALL LUMP(IPLW,IU,IUL)
C     CALL LUMP(IPLW,IV,IVL)
C     CALL RIGBND(IUL,IVL,IU,IV)
      DO 10 I=1,NSTEPS
      CALL ADVECT(IPN,IPL,IP,IU,IPW,DT)
      CALL COPY(IPL,IPN)
      CALL UNLUMP(IPLW,IPL,IP,1.E-5)
      IF (MOD(I,10) .EQ. 0) CALL MAXMIN(IP)
   10 CONTINUE
      CALL MAXMIN(IP)
      RETURN
      END
```

```
      SUBROUTINE GRDLEN
      $INCLUDE(TRG.DCL)'
      J=1
      NSEG=0
      XMIN=1.E10
      XMAX=0.
      XMEAN=0.
      DO 1 I=1,NUMV
      JP=J+1
      NV=IV(JP)
      JA=IV(J)
      X=S(JA)
      Y=S(JA+1)
      DO 2 K=1,NV
      JK=JP+K
      JC=IV(JK)
      JB=IV(JC)
      XX=S(JB)
      YY=S(JB+1)
      D=SQRT((X-XX)**2+(Y-YY)**2)
      NSEG=NSEG+1
      XMEAN=XMEAN+D
      IF (D .GT. XMAX) XMAX=D
      IF (D .LT. XMIN) XMIN=D
    2 CONTINUE
    1 J=J+IVSIZ
      XMEAN=XMEAN/NSEG
    3 FORMAT(' MIN,MEAN,MAX',3F12.5)
      WRITE(1,3) XMIN,XMEAN,XMAX
      RETURN
      END
      SUBROUTINE TFIELD(IPOS)
      $INCLUDE(TRG.DCL)^
      YC=0.
      XC=1.
      RC=.9
      I=1
      DO 1 II=1,NUMV
      K=IV(I)
      X=S(K)
      Y=S(K+1)
      R=SQRT((X-XC)**2+(Y-YC)**2)/RC
      C=0.
      IF (R .LT. 1.) C=(2.*R-3.)*R*R+1.
      S(K+IPOS)=C
    1 I=I+IVSIZ
      RETURN
      END
```

Main program for the shallow water equations code

```
$INCLUDE(TMAC.MAC)$DELTOK
      DIMENSION ERAS(SERAS)
      DIMENSION CONVAL(10)
      LOGICAL PLOT
      PLOT=.FALSE.
      ASCALE=1.
      DT=.04
      CALL INPUT('DT....',DT)
      X=2
      CALL INPUT('NUMDIV',X)
      NUMDIV=X
      X=101
      CALL INPUT('NSTEPS',X)
      NSTEPS=X
      X=10
      CALL INPUT('NCONT ',X)
      NCONT=X
      DO 2 I=1,10
    2 CONVAL(I)=I*.2-.1
      CALL HEXTOP(NUMDIV)
      CALL VSTO(1,42)
      CALL CIRBND
      CALL SUGRID(20)
      IPLW=2
      CALL CVLW(IPLW)
      IPW=9
      CALL CTW(IPW)
      CALL TSTCTW(IPW)
      IP=23
      IPU=24
      IPV=25
      IPUU=26
      IPUV=27
      IPVV=28
      IPLA=29
      IPULA=30
      IPVLA=31
      IPLB=32
      IPULB=33
      IPVLB=34
      IPLC=35
      IPULC=36
      IPVLC=37
      IPFX=38
      IPFY=39
      IPFXL=40
      IPFYL=41
      CALL INITF(IP,IPU,IPV)
      CALL LUMP(2,IP,IPLA)
```

```
      CALL LUMP(2,IPU,IPULA)
      CALL LUMP(2,IPV,IPVLA)
      CF=0.
      CF=1.
      CALL INPUT('CF....',CF)
      F=0.
      CALL INPUT('F.....',F)
      WSX=.3
      WSY=0.
      CALL INPUT('WSX...',WSX)
      CALL INPUT('WSY... ',WSY)
      IF (PLOT) CALL PLOTS(ERAS,900,5.5)
      DO 10 I=1,NSTEPS
      IF (MOD(I-1,NCONT) .NE. 0) GO TO 9
      CALL UNLUMP(2,IPLA,IP,1.E-5)
      CALL QPRCON(IP,0.,2.)
      CALL EWPLOT(IP,2.,0.)
      IF (PLOT) CALL ORIGIN(5.5,0.)
      IF (PLOT) CALL DRAWV(IPLA,IPULA,IPVLA,IP,IPU,IPV,ASCALE,IPLW)
    9 CONTINUE
      CALL STEP(IPLB,IPULB,IPVLB,
     1 IPLA,IPULA,IPVLA,
     2 IPLA,IPULA,IPVLA,
     3 IP,IPU,IPV,IPUU,IPUV,IPVV,
     4 IPFX,IPFY,IPFXL,IPFYL,IPW,DT,F,CF,WSX,WSY)
      CALL MAXMIN(IP)
      CALL MAXMIN(IPU)
      CALL MAXMIN(IPV)
      CALL STEP(IPLC,IPULC,IPVLC,
     1 IPLA,IPULA,IPVLA,
     2 IPLB,IPULB,IPVLB,
     3 IP,IPU,IPV,IPUU,IPUV,IPVV,
     4 IPFX,IPFY,IPFXL,IPFYL,IPW,DT,F,CF,WSX,WSY)
      CALL COPY(IPLA,IPLC)
      CALL COPY(IPULA,IPULC)
      CALL COPY(IPVLA,IPVLC)
   10 CONTINUE
      IF (PLOT) CALL ORIGIN(5.5,0.)
      IF (PLOT) CALL ENDPLT
      STOP
      END
      SUBROUTINE INPUT(STRING,X)
      CHARACTER STRING(6)
      WRITE(OUTLUN,1) STRING,X
    1 FORMAT(' Enter value for ',6A1,'   Perhaps',1F12.3)
      ACCEPT *,X
      RETURN
      END
```

Common block TRG.DCL

```
      COMMON /TRG/ S(SMAX),IV(IVMAX),IC(ICMAX),
     . SCALE,
     . NEWV,NEWC,IVSIZ,ICSIZ,NUMC,NUMV,NBV,IBVF
```

Library of subroutines for triangular grid codes

```
$INCLUDE(TMAC.MAC)$DELTOK
      SUBROUTINE ADVECT(IPCN,IPCL,IPC,IPU,IPW,DT)
$INCLUDE(TRG.DCL)^
      J=1
      DO 1 I=1,NUMV
      K=IV(J)
      NV=IV(J+1)
      KIPC=K+IPC
      KIPU=K+IPU
      KIPW=K+IPW
      KIPCN=K+IPCN
      S(KIPCN)=S(KIPC)*(S(KIPU)*S(KIPW)+S(KIPU+1)*S(KIPW+7))
      DO 2 IN=1,NV
      JIN=J+IN
      JP=IV(JIN+1)
      KP=IV(JP)
      KIPCN=K+IPCN
      KPIPC=KP+IPC
      KPIPU=KP+IPU
      KIPWIN=K+IPW+IN
    2 S(KIPCN)=S(KIPCN)+S(KPIPC)*(S(KPIPU)*S(KIPWIN)+
     1 S(KPIPU+1)*S(KIPWIN+7))
      KIPCN=K+IPCN
      KIPCL=K+IPCL
      S(KIPCN)=S(KIPCL)-DT*S(KIPCN)
    1 J=J+IVSIZ
      RETURN
      END


      SUBROUTINE CIRBND
$INCLUDE(TRG.DCL)^
      DTH=6.283184/NBV
      J=IBVF
      TH=0.
      DO 1 I=1,NBV
      IF (J .EQ. 0) GO TO 9
      IF (IV(J+1) .EQ. 6) GO TO 9
      K=IV(J)
      S(K)=COS(TH)
      S(K+1)=SIN(TH)
      TH=TH+DTH
    1 J=IV(J+7)
      IF (J .EQ. 0) RETURN
    8 FORMAT(' CIRBND ERROR',2I10)
    9 WRITE(OUTLUN,8) I,J
      STOP
      END
```

```
      SUBROUTINE CONFLT(IFLD,CONVAL,N)
$INCLUDE(TRG.DCL)^
      DIMENSION CONVAL(1),F(3),KV(3),XV(3),YV(3),XL(2),YL(2)
      J=1
      DO 1 I=1,NUMC
      DO 2 JJ=1,3
      JJJ=J+JJ
      JV=IC(JJJ)
      KA=IV(JV)
      KV(JJ)=KA
      KAIFLD=KA+IFLD
      F(JJ)=S(KAIFLD)
      XV(JJ)=S(KA)*SCALE
    2 YV(JJ)=S(KA+1)*SCALE
      DO 4 NC=1,N
      NI=0
      FC=CONVAL(NC)
      DO 3 JA=1,3
      JB=JA+1
      IF (JB .EQ. 4) JB=1
      IF ((F(JA)-FC)*(FC-F(JB)) .LE. 0.) GO TO 3
      NI=NI+1
      ALPHA=(FC-F(JA))/(F(JB)-F(JA))
      XL(NI)=(1.-ALPHA)*XV(JA)+ALPHA*XV(JB)
      YL(NI)=(1.-ALPHA)*YV(JA)+ALPHA*YV(JB)
    3 CONTINUE
      IF (NI .EQ. 0) GO TO 4
      CALL LINE(XL,YL,2,1,0,0)
    4 CONTINUE
    1 J=J+ICSIZ
      RETURN
      END



      SUBROUTINE CONSRV(IFROM)
$INCLUDE(TRG.DCL)^
      SUM=0.
      J=1
      DO 1 I=1,NUMV
      K=IV(J)
      KIFROM=K+IFROM
      SUM=SUM+S(KIFROM)
    1 J=J+IVSIZ
    3 FORMAT(' FIELD',I5,' INTEGRAL',1F15.9)
      WRITE(OUTLUN,3) IFROM,SUM
      RETURN
      END
```

```
      SUBROUTINE COPY(ITO,IFROM)
$INCLUDE(TRG.DCL)^
      J=1
      DO 1 I=1,NUMV
      K=IV(J)
      KITO=K+ITO
      KIFROM=K+IFROM
      S(KITO)=S(KIFROM)
    1 J=J+IVSIZ
      RETURN
      END



      SUBROUTINE CTW(IPOS)
      LOGICAL NEIGH
$INCLUDE(TRG.DCL)^
      I=1
      DO 1 II=1,NUMV
      NV=IV(I+1)
      K=IV(I)
      X=S(K)
      Y=S(K+1)
      KIPOS=K+IPOS
      S(KIPOS)=0.
      DO 2 J=1,NV
      KIPOSJ=KIPOS+J
    2 S(KIPOSJ)=0.
      NVM1=NV-1
      DO 3 JP=1,NVM1
      IJP1=I+JP+1
      NP=IV(IJP1)
      KP=IV(NP)
      PX=S(KP)
      PY=S(KP+1)
      JPP1=JP+1
      DO 3 JQ=JPP1,NV
      IJQ1=I+JQ+1
      NQ=IV(IJQ1)
      IF (.NOT. NEIGH(NP,NQ)) GO TO 3
      KQ=IV(NQ)
      QX=S(KQ)
      QY=S(KQ+1)
      PQ=(PX-X)*(QY-Y)-(PY-Y)*(QX-X)
      SIG=1.
      IF (PQ .LT. 0.) SIG=-1.
      IF (IV(JP+1) .EQ. 6) GO TO 4
      DX=.5*SIG*(PY-Y)
      DY=.5*SIG*(PX-X)
      KJP=KIPOS+JP
```

```
      S(KIPOS)=S(KIPOS)+.75*DX
      S(KIPOS+7)=S(KIPOS+7)+.75*DY
      S(KJP)=S(KJP)+.25*DX
      S(KJP+7)=S(KJP+7)+.25*DY
    4 IF (IV(JQ+1) .EQ. 6) GO TO 5
      DX=-.5*SIG*(QY-Y)
      DY=.5*SIG*(QX-X)
      S(KIPOS)=S(KIPOS)+.75*DX
      S(KIPOS+7)=S(KIPOS+7)+.75*DY
      KJP=KIPOS+JQ
      S(KJP)=S(KJP)+.25*DX
      S(KJP+7)=S(KJP+7)+.25*DY
    5 CONTINUE
      DX=-SIG*(PY-.5*(QY+Y))/3.
      DY=SIG*(PX-.5*(QX+X))/3.
      EX=SIG*(QY-.5*(PY+Y))/3.
      EY=-SIG*(QX-.5*(PX+X))/3.
      S(KIPOS)=S(KIPOS)+5.*(DX+EX)/12.
      S(KIPOS+7)=S(KIPOS+7)+5.*(DY+EY)/12.
      KJP=KIPOS+JP
      S(KJP)=S(KJP)+DX/6.+5.*EX/12.
      S(KJP+7)=S(KJP+7)+DY/6.+5.*EY/12.
      KQ=KIPOS+JQ
      S(KQ)=S(KQ)+5.*DX/12.+EX/6.
      S(KQ+7)=S(KQ+7)+5.*DY/12.+EY/6.
    3 CONTINUE
      I=I+IVSIZ
    1 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE CVLW(IPOS)
      LOGICAL NEIGH
$INCLUDE(TRG.DCL)^
      A1=11./54.
      A2=7./108.
      I=1
      DO 1 II=1,NUMV
      NV=IV(I+1)
      K=IV(I)
      X=S(K)
      Y=S(K+1)
      KIPOS=K+IPOS
      S(KIPOS)=0.
      DO 2 J=1,NV
      KIPOSJ=K+IPOS+J
    2 S(KIPOSJ)=0.
      NVM1=NV-1
      DO 3 JP=1,NVM1
      IJP=I+JP
      NP=IV(IJP+1)
      KP=IV(NP)
      XP=S(KP)
      YP=S(KP+1)
      JPP1=JP+1
      DO 3 JQ=JPP1,NV
      IJQ=I+JQ
      NQ=IV(IJQ+1)
      IF (.NOT. NEIGH(NP,NQ)) GO TO 3
      KQ=IV(NQ)
      XQ=S(KQ)
      YQ=S(KQ+1)
      DELTA=.5*ABS((XP-X)*(YQ-Y)-(YP-Y)*(XQ-X))
      KIPOS=K+IPOS
      S(KIPOS)=S(KIPOS)+A1*DELTA
      KIPOSJP=KIPOS+JP
      S(KIPOSJP)=S(KIPOSJP)+A2*DELTA
      KIPOSJQ=KIPOS+JQ
      S(KIPOSJQ)=S(KIPOSJQ)+A2*DELTA
    3 CONTINUE
      I=I+IVSIZ
    1 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE DRAWV(IPL,IPUL,IPVL,IP,IPU,IPV,ASCALE,IPW)
$INCLUDE(TRG.DCL)
      DIMENSION X(5),Y(5)
      ERR=1.E-4
      CALL UNLUMP(IPW,IPL,IP,ERR)
      CALL UNLUMP(IPW,IPUL,IPU,ERR)
      CALL UNLUMP(IPW,IPVL,IPV,ERR)
      J=1
      DO 10 I=1,NUMV
      K=IV(J)
      XX=S(K)*SCALE
      YY=S(K+1)*SCALE
      KIP=K+IP
      P=S(KIP)
      U=S(KIPU)/P*SCALE
      KIPV=K+IPV
      V=S(KIPV)/P*SCALE
      UV=SQRT(U*U+V*V)
      IF (UV .LT. .01*ASCALE) GO TO 10
      DX=ASCALE*U*.5
      DY=ASCALE*V*.5
      X(1)=XX+DX
      X(2)=XX-DX
      Y(1)=YY+DY
      Y(2)=YY-DY
      DX=.4*DX
      DY=.4*DY
      X(3)=X(1)-DX-DY
      Y(3)=Y(1)-DY+DX
      X(5)=X(1)-DX+DY
      Y(5)=Y(1)-DY-DX
      Y(4)=Y(1)
      X(4)=X(1)
      CALL LINE(X,Y,2,1,0,0)
      CALL LINE(X(3),Y(3),3,1,0,0)
   10 J=J+IVSIZ
      RETURN
      END
```

```
      SUBROUTINE DRCELL
$INCLUDE(TRG.DCL)^
      DIMENSION X(2),Y(2)
      J=1
      DO 1 I=1,NUMV
      K=IV(J)
      X(1)=S(K)*SCALE
      Y(1)=S(K+1)*SCALE
      CALL SYMBOL(X,Y,.05,2,0.,1)
    1 J=J+IVSIZ
      J=1
      DO 2 I=1,NUMC
      JA=IC(J+1)
      JB=IC(J+3)
      JC=IC(J+3)
      KA=IV(JA)
      KB=IV(JB)
      KC=IV(JC)
      XA=S(KA)*SCALE
      YA=S(KA+1)*SCALE
      XB=S(KB)*SCALE
      YB=S(KB+1)*SCALE
      XC=S(KC)*SCALE
      YC=S(KC+1)*SCALE
      XG=(XA+XB+XC)/3.
      YG=(YA+YB+YC)/3.
      X(1)=XG
      Y(1)=YG
      X(2)=.5*(XA+XB)
      Y(2)=.5*(YA+YB)
      CALL LINE(X,Y,2,1,0,0)
      X(2)=.5*(XB+XC)
      Y(2)=.5*(YB+YC)
      CALL LINE(X,Y,2,1,0,0)
      X(2)=.5*(XC+XA)
      Y(2)=.5*(YX+YA)
      CALL LINE(X,Y,2,1,0,0)
    2 J=J+ICSIZ
      RETURN
      END
```

```
      SUBROUTINE DRGRID
$INCLUDE(TRG.DCL)'
      DIMENSION X(2),Y(2)
      J=1
      DO 1 I=1,NUMV
      K=IV(J)
      NV=IV(J+1)
      X(1)=S(K)*SCALE
      Y(1)=S(C+1)*SCALE
      DO 2 IP=1,NV
      JIP=J+IP
      JP=IV(JIP+1)
      IF (JP .GT. J) GO TO 2
      KP=IV(JP)
      X(2)=S(KP)*SCALE
      Y(2)=S(KP+1)*SCALE
      CALL LINE(X,Y,2,1,0,0)
    2 CONTINUE
    1 J=J+IVSIZ
      RETURN
      END



      SUBROUTINE EWPLOT(IP,ZMAX,ZMIN)
$INCLUDE(PAGE.DCL)'
      LOGICAL FAIL
      CHARACTER ISP,ISTAR,IVERT,IHOR
      DATA ISP/' '/,ISTAR/'*'/,IVERT/'!'/,IHOR/'_'/
      DO 1 J=1,MAPMAXV
      DO 2 I=1,MAPMAXH
      IMAP(I,J)=ISP
      IF (I .EQ. 1 .OR. I .EQ. MAPMAXH) IMAP(I,J)=IVERT
      IF (J .EQ. 1 .OR. J .EQ. MAPMAXV) IMAP(I,J)=IHOR
    2 CONTINUE
    1 CONTINUE
      A=2./(MAPMAXH-1.)
      B=-1.-A
      C=(MAPMAXV-1.)/(ZMIN-ZMAX)
      D=1.-C*ZMAX
      DO 3 I=1,MAPMAXH
      X=A*I+B
      CALL INTPOL(IP,X,0.,Z,FAIL)
      IF (FAIL) GO TO 3
      J=C*Z+D
      IF (J .LT. 1 .OR. J .GT. MAPMAXV) GO TO 3
      IMAP(I,J)=ISTAR
    3 CONTINUE
    4 FORMAT('1',/,(5X,MAPMAXH A1))
      WRITE(OUTLUN,4) IMAP
      RETURN
      END
```

```
      SUBROUTINE FLDAV(IPOSA,IPOSB,IPOSC)
$INCLUDE(TRG.DCL)
      J=1
      DO 1 I=1,NUMV
      K=IV(J)
      KIPOSA=K+IPOSA
      KIPOSB=K+IPOSB
      KIPOSC=K+IPOSC
      S(KIPOSA)=.5*(S(KIPOSB)+S(KIPOSC))
    1 J=J+IVSIZ
      RETURN
      END



      SUBROUTINE FRF(IPL,IPUL,IPVL,IP,IPU,IPV,IPFX,IPFY,IPFXL,IPFYL,
     . DT,F,CF,WSX,WSY)
$INCLUDE(TRG.DCL)
      CALL UNLUMP(2,IPL,IP,1.E-2)
      CALL UNLUMP(2,IPUL,IPU,1.E-2)
      CALL UNLUMP(2,IPVL,IPV,1.E-2)
      J=1
      DO 1 I=1,NUMV
      IF (IV(J+1) .NE. 6) GO TO 1
      K=IV(J)
      KIP=K+IP
      P=S(KIP)
      KIPU=K+IPU
      U=S(KIPU)/P
      KIPV=K+IPV
      V=S(KIPV)/P
      UV=SQRT(U*U+V*V)
      KIPFX=K+IPFX
      S(KIPFX)=(WSX-CF*U*UV)*P
      KIPFY=K+IPFY
      S(KIPFY)=(WSY-CF*V*UV)*P
    1 J=J+IVSIZ
      CALL LUMP(2,IPFX,IPFXL)
      CALL LUMP(2,IPFY,IPFYL)
      J=1
      DO 2 I=1,NUMV
      K=IV(J)
      A11=1.
      A12=-DT*F
      A21=DT*F
      A22=1.
      KIPFXL=K+IPFXL
      KIPUL=K+IPUL
      B1=DT*S(KIPFXL)+S(KIPUL)
      KIPFYL=K+IPFYL
      KIPVL=K+IPVL
```

```
      B2=DT*S(KIPFYL)+S(KIPVL)
      D=A11*A22-A11*A21
      S(KIPUL)=(B1*A22-A12*B2)/D
      S(KIPVL)=(A11*B2-B1*A21)/D
    2 J=J+IVSIZ
      RETURN
      END



      SUBROUTINE GDC(IP)
$INCLUDE(TRG.DCL)'
      IC(NEWC+1)=IP
      IC(NEWC+2)=IP+IVSIZ
      IC(NEWC+3)=NEWV-IVSIZ
      IP=IP+IVSIZ
      NEWC=NEWC+ICSIZ
      RETURN
      END



      SUBROUTINE GUC(IP)
$INCLUDE(TRG.DCL)'
      IC(NEWC+1)=IP
      IC(NEWC+2)=NEWV
      IC(NEWC+3)=NEWV-IVSIZ
      NEWV=NEWV+IVSIZ
      NEWC=NEWC+ICSIZ
      RETURN
      END
```

```
      SUBROUTINE HEXTOP(N)
$INCLUDE(TRG.DCL)
      SCALE=4.
      DO 100 I=1,ICMAX
  100 IC(I)=0
      DO 101 I=1,IVMAX
  101 IV(I)=0
      NEWC=1
      IVSIZ=8
      ICSIZ=4
      IP=1
      NEWV=1+(N+2)*IVSIZ
      IW=N
      DO 1 I=1,N
      DO 2 J=1,IW
      CALL GUC(IP)
    2 CALL GDC(IP)
      CALL GUC(IP)
      IP=IP+IVSIZ
      NEWV=NEWV+IVSIZ
    1 IW=IW+1
      IW=IW-1
      DO 3 I=1,N
      DO 4 J=1,IW
      CALL GDC(IP)
    4 CALL GUC(IP)
      CALL GDC(IP)
      IP=IP+IVSIZ
      NEWV=NEWV+IVSIZ
    3 IW=IW-1
      NEWV=NEWV-IVSIZ
      NUMC=(NEWC-1)/ICSIZ
      NUMV=(NEWV-1)/IVSIZ
      WRITE(OUTLUN,5) NUMC,NUMV
    5 FORMAT(' Allocated cells and vertices',2I10)
      JC=1
      DO 10 I=1,NUMC
      IVA=IC(JC+1)
      IVB=IC(JC+2)
      IVC=IC(JC+3)
      CALL VLINK(IVA,IVB)
      CALL VLINK(IVB,IVC)
      CALL VLINK(IVC,IVA)
      CALL VLINK(IVB,IVA)
      CALL VLINK(IVC,IVB)
      CALL VLINK(IVA,IVC)
   10 JC=JC+ICSIZ
      IP=1
      DO 20 I=1,NUMV
      IF (IV(IP+1) .LT. 6) GO TO 21
   20 IP=IP+IVSIZ
```

```
   22 FORMAT(' NO BOUNDARY POINTS')
      WRITE(OUTLUN,22)
      STOP
   21 IB=IP
      IBVF=IB
      IP=0
      NBV=1
   27 IV(IB+7)=IP
      NV=IV(IB+1)
      DO 23 I=1,NV
      IBI1=IB+I+1
      J=IV(IBI1)
      IF (J .EQ. IP) GO TO 23
      IF (J .EQ. IBVF) GO TO 25
      IF (IV(J+1) .LT. 6) GO TO 24
   23 CONTINUE
   26 FORMAT(' BOUNDARY ERROR')
      WRITE(OUTLUN,26)
      CALL VCDUMP
      STOP
   24 NBV=NBV+1
      IP=IB
      IB=J
      GO TO 27
   30 FORMAT(' NUMBER OF BOUNDARY VERTICES',I10)
   25 WRITE(OUTLUN,30) NBV
      IBVF=IB
      RETURN
      END



      SUBROUTINE INITF(IP,IPU,IPV)
$INCLUDE(TRG.DCL)'
      J=1
      DO 1 I=1,NUMV
      K=IV(J)
      KIP=K+IP
      S(KIP)=1.
      KIPU=K+IPU
      S(KIPU)=0.
      KIPV=K+IPV
      S(KIPV)=0.
    1 J=J+IVSIZ
      RETURN
      END
```

```
      SUBROUTINE INTPOL(IFLD,X,Y,Z,FAIL)
$INCLUDE(TRG.DCL)^
      LOGICAL FAIL
      FAIL=.TRUE.
      ICELL=1
      DO 10 I=1,NUMC
      IP=IC(ICELL+1)
      IQ=IC(ICELL+2)
      IR=IC(ICELL+3)
      JP=IV(IP)
      JQ=IV(IQ)
      JR=IV(IR)
      XP=S(JP)-X
      YP=S(JP+1)-Y
      XQ=S(JQ)-X
      YQ=S(JQ+1)-Y
      XR=S(JR)-X
      YR=S(JR+1)-Y
      PQ=XP*YQ-YP*XQ
      QR=XQ*YR-YQ*XR
      RP=XR*YP-YR*XP
      IF (PQ*QR .LT. 0.) GO TO 1
      IF (QR*RP .LT. 0.) GO TO 1
      IF (PQ*RP .LT. 0.) GO TO 1
      JPIFLD=JP+IFLD
      ZP=S(JPIFLD)
      JQIFLD=JQ+IFLD
      ZQ=S(JQIFLD)
      JRIFLD=JR+IFLD
      ZR=S(JRIFLD)
      D=XP*(YQ-YR)-YP*(XQ-XR)+XQ*YR-YQ*XR
      Z=(XP*(YQ*ZR-ZQ*YR)-YP*(XQ*ZR-ZQ*XR)+ZP*(XQ*YR-YQ*XR))/D
      FAIL=.FALSE.
      RETURN
    1 ICELL=ICELL+ICSIZ
   10 CONTINUE
      RETURN
      END
```

```
      FUNCTION INTRI(X,Y,ICELL)
$INCLUDE(TRG.DCL)'
      LOGICAL INTRI
      IP=IC(ICELL+1)
      IQ=IC(ICELL+2)
      IR=IC(ICELL+3)
      JP=IV(IP)
      JQ=IV(IQ)
      JR=IV(JR)
      XP=S(JP)-X
      YP=S(JP+1)-Y
      XQ=S(JQ)-X
      YR=S(JQ+1)-Y
      XR=S(JR)-X
      YR=S(JR+1)-Y
      PQ=XP*YQ-YP*XQ
      QR=XQ*YR-YQ*XR
      RP=XR*YP-YR*XP
      IF (PQ*QR .LT. 0.) GO TO 1
      IF (QR*RP .LT. 0.) GO TO 1
      IF (PQ*RP .LT. 0.) GO TO 1
      INTRI=.TRUE.
      RETURN
    1 INTRI=.FALSE.
      RETURN
      END
```

```
      SUBROUTINE LUMP(IPOS,IFROM,ITO)
$INCLUDE(TRG.DCL)^
      J=1
      DO 1 I=1,NUMV
      K=IV(J)
      NV=IV(J+1)
      KITO=K+ITO
      KIPOS=K+IPOS
      KIFROM=K+IFROM
      S(KITO)=S(KIPOS)*S(KIFROM)
      DO 2 IN=1,NV
      JIN1=J+IN+1
      JP=IV(JIN1)
      KP=IV(JP)
      KIPIN=KIPOS+IN
      KPIFR=KP+IFROM
    2 S(KITO)=S(KITO)+S(KIPIN)*S(KPIFR)
    1 J=J+IVSIZ
      RETURN
      END



      SUBROUTINE MAXMIN(IFLD)
$INCLUDE(TRG.DCL)^
      XMAX=S(IFLD+1)
      XMIN=XMAX
      J=1
      DO 1 I=1,NUMV
      K=IV(J)
      KIFLD=K+IFLD
      X=S(KIFLD)
      IF (X .GT. XMAX) XMAX=X
      IF (X .LT. XMIN) XMIN=X
    1 J=J+IVSIZ
    2 FORMAT(' MAX,MIN',2F10.6)
      WRITE(OUTLUN,2)XMAX,XMIN
      RETURN
      END
```

```
      FUNCTION NEIGH(IP,IQ)
$INCLUDE(TRG.DCL)^
      LOGICAL NEIGH
      NV=IV(IP+1)
      DO 1 I=1,NV
      IPI1=IP+I+1
      IF (IV(IPI1) .EQ. IQ) GO TO 2
    1 CONTINUE
      NEIGH=.FALSE.
      RETURN
    2 NEIGH=.TRUE.
      RETURN
      END



      SUBROUTINE PRCON(IFLD,XMIN,XMAX)
      LOGICAL FAIL
      DIMENSION ICON(13),LIN(MAPMAXH)
      DATA ICON/'-','0','1','2','3','4','5','6','7','8','9','+',' '/
      BX=-(MAPMAXH+1.)/MAPMAXH
      BY=(MAPMAXV+1.)/MAPMAXV
      WRITE(OUTLUN,5)
      DO 1 J=1,MAPMAXV
      Y=-J/(MAPMAXV/2.)+BY
      DO 2 I=1,MAPMAXH
      X=I/(MAPMAXH/2.)+BX
      R=X*X+Y*Y
      K=13
      IF (R .GT. 1.) GO TO 3
      CALL INTPOL(IFLD,X,Y,Z,FAIL)
      IF (FAIL) GO TO 3
      K=10.*(Z-XMIN)/(XMAX-XMIN)
      IF (K .LT. 0) K=-1
      IF (K .GT. 10) K=10
      K=K+2
    3 LIN(I)=ICON(K)
    2 CONTINUE
    4 FORMAT(5X,MAPMAXH A1)
      WRITE(OUTLUN,4) LIN
    1 CONTINUE
    5 FORMAT('1')
      RETURN
      END
```

```
      SUBROUTINE PRFLDS(I1,I2)
$INCLUDE(TRG.DCL)^
      DIMENSION V(10)
      WRITE(OUTLUN,4) I1,I2
    4 FORMAT(' FIELDS',I4,' THROUGH',I4)
      WRITE(OUTLUN,4) I1,I2
      J=1
      DO 1 I=1,NUMV
      K=IV(J)
      NN=1
      DO 2 N=I1,I2
      KN=K+N
      V(NN)=S(KN)
      NN=NN+1
      IF (NN .LE. 10) GO TO 2
      WRITE(OUTLUN,3) I,V
    3 FORMAT(' ',I8,10F12.6)
      NN=1
    2 CONTINUE
      NN=NN-1
      IF (NN .GT. 0) WRITE(OUTLUN,3) I,(V(N),N=1,NN)
    1 J=J+IVSIZ
      RETURN
      END



      SUBROUTINE QPRCON(IFLD,XMIN,XMAX)
      CHARACTER ICON(13)
$INCLUDE(TRG.DCL)^
$INCLUDE(PAGE.DCL)^
      DATA ICON/'-','0','1','2','3','4','5','6','7','8','9','+',' '/
      DO 2 J=1,MAPMAXV
      DO 2 I=1,MAPMAXH
    2 IMAP(I,J)=ICON(13)
      IVERT=1
      DO 1 II=1,NUMV
      KK=IV(IVERT)
      X=S(KK)
      Y=S(KK+1)
      KKIFLD=KK+IFLD
      Z=S(KKIFLD)
      I= (MAPMAXH/2.)*X+MAPMAXH/2.+1.
      IF (I .LT. 1) I=1
      IF (I .GT. MAPMAXH) I=MAPMAXH
      J=-(MAPMAXV/2.)*Y+MAPMAXV/2.+1.
      IF (J .LT. 1) J=1
      IF (J .GT. MAPMAXV) J=MAPMAXV
      K=10.*(Z-XMIN)/(XMAX-XMIN)
      IF (K .LT. 0) K=-1
      IF (K .GT. 10) K=10
```

```
      K=K+2
      IMAP(I,J)=ICON(K)
    1 IVERT=IVERT+IVSIZ
    3 FORMAT('1',/,(5X,MAPMAXH A1))
      WRITE(OUTLUN,3) IMAP
      RETURN
      END


      SUBROUTINE RIGBND(IPUL,IPVL,IPU,IPV)
      $INCLUDE(TRG.DCL)^
      CALL UNLUMP(2,IPUL,IPU,1.E-5)
      CALL UNLUMP(2,IPVL,IPV,1.E-5)
      J=IBVF
      DO 1 I=1,NBV
      K=IV(J)
      KIPU=K+IPU
      S(KIPU)=0.
      KIPV=K+IPV
      S(KIPV)=0.
    1 J=IV(J+7)
      CALL LUMP(2,IPU,IPUL)
      CALL LUMP(2,IPV,IPVL)
      RETURN
      END


      SUBROUTINE SBROT(OMEGA,IPOSU)
      $INCLUDE(TRG.DCL)^
      I=1
      DO 1 II=1,NUMV
      K=IV(I)
      X=S(K)
      Y=S(K+1)
      I=I+IVSIZ
      KIPOSU=K+IPOSU
      S(KIPOSU)=-Y*OMEGA
    1 S(KIPOSU+1)=X*OMEGA
      RETURN
      END
```

```
      SUBROUTINE STEP(IPLN,IPULN,IPVLN,IPLO,IPULO,IPVLO,IPL,IPUL,IPVL,
     . IP,IPU,IPV,IPUU,IPUV,IPVV,IPFX,IPFY,IPFXL,IPFYL,IPW,DT,
     . F,CF,WSX,WSY)
      $INCLUDE(TRG.DCL)
      CALL UNLUMP(2,IPL,IP,1.E-2)
      CALL UNLUMP(2,IPUL,IPU,1.E-2)
      CALL UNLUMP(2,IPVL,IPV,1.E-2)
      J=1
      DO 1 I=1,NUMV
      K=IV(J)
      KIP=K+IP
      KIPU=K+IPU
      KIPV=K+IPV
      KIPUU=K+IPUU
      KIPUV=K+IPUV
      KIPVV=K+IPVV
      P=S(KIP)
      U=S(KIPU)/P
      V=S(KIPV)/P
      S(KIPUU)=P*(U*U+.5*P)
      S(KIPUV)=P*U*V
      S(KIPVV)=P*(V*V+.5*P)
    1 J=J+IVSIZ
      CALL TRNSPT(IPLN,IPLO,IPU,IPV,IPW,DT)
      CALL TRNSPT(IPULN,IPULO,IPUU,IPUV,IPW,DT)
      CALL TRNSPT(IPVLN,IPVLO,IPUV,IPVV,IPW,DT)
      CALL FRF(IPLN,IPULN,IPVLN,IP,IPU,IPV,IPFX,IPFY,IPFXL,IPFYL,DT,
     . F,CF,WSX,WSY)
      CALL RIGBND(IPULN,IPVLN,IPU,IPV)
      RETURN
      END
```

```
      SUBROUTINE SUGRID(NITER)
      $INCLUDE(TRG.DCL)^
      RELPA=1.388
      DO 9 MI=1,NITER
      J=1
      CHMAX=0.
      DO 1 I=1,NUMV
      NV=IV(J+1)
      IF (NV .LT. 6) GO TO 1
      X=0.
      Y=0.
      DO 2 K=1,6
      JK1=J+1+K
      JN=IV(JK1)
      JS=IV(JN)
      X=X+S(JS)
    2 Y=Y+S(JS+1)
      JS=IV(J)
      XO=S(JS)
      YO=S(JS+1)
      X=X/6.
      Y=Y/6.
      CH=SQRT((X-XO)**2+(Y-YO)**2)
      IF (CH .GT. CHMAX) CHMAX=CH
      S(JS)=RELPA*X+(1.-RELPA)*S(JS)
      S(JS+1)=RELPA*Y+(1-RELPA)*S(JS+1)
    1 J=J+IVSIZ
    8 FORMAT(' MAXIMUM DISPLACEMENT',1F12.8)
      WRITE(OUTLUN,8) CHMAX
    9 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE TRNSF1(INEW,IOLD,IFX,IFY,IPW,DT)
      $INCLUDE(TRG.DCL)'
      J=1
      DO 1 I=1,NUMV
      K=IV(J)
      NV=IV(J+1)
      KINEW=K+INEW
      KIFX=K+IFX
      KIPW=K+IPW
      KIFY=K+IFY
      S(KINEW)=S(KIFX)*S(KIPW)+S(KIFY)*S(KIPW+7)
      DO 2 IN=1,NV
      JIN=J+IN
      JP=IV(JIN+1)
      KP=IV(JP)
      KPIFX=KP+IFX
      KIPWIN=KIPW+IN
      KPIFY=KP+IFY
    2 S(KINEW)=S(KINEW)+S(KPIFX)*S(KIPWIN)+S(KPIFY)*S(KIPWIN+7)
      KIOLD=K+IOLD
      S(KINEW)=S(KIOLD)-DT*S(KINEW)
    1 J=J+IVSIZ
      RETURN
      END



      SUBROUTINE TSTCTW(IPOS)
      $INCLUDE(TRG.DCL)'
      WRITE(OUTLUN,4)
    4 FORMAT(' TEST T-WEIGHTS'/8X,'I',8X,'NV',18X,'TEST X',14X,'TEST Y')
      I=1
      DO 1 II=1,NUMV
      NV=IV(I+1)
      K=IV(I)
      KIPOS=K+IPOS
      X=S(KIPOS)
      Y=S(KIPOS+7)
      DO 2 J=1,NV
      KIPOSJ=KIPOS+J
      X=X+S(KIPOSJ)
      Y=Y+S(KIPOSJ+7)
    2 CONTINUE
    3 FORMAT(1X,2I10,2F20.10)
      WRITE(OUTLUN,3) I,NV,X,Y
    1 I=I+IVSIZ
      RETURN
      END
```

```
      SUBROUTINE UNLUMP(IPOS,IFROM,ITO,ERR)
      $INCLUDE(TRG.DCL)^
      RELPA=1.3
      NITER=25
      DO 11 NIT=1,NITER
      CHMAX=0.
      J=1
      DO 10 I=1,NUMV
      K=IV(J)
      KITO=K+ITO
      SOLD=S(KITO)
      KIFROM=K+IFROM
      S(KITO)=S(KIFROM)
      NV=IV(J+1)
      DO 12 IN=1,NV
      JIN=J+IN
      JP=IV(JIN+1)
      KP=IV(JP)
      KIPSIN=K+IPOS+IN
      KPITO=KP+ITO
   12 S(KITO)=S(KITO)-S(KIPSIN)*S(KPITO)
      KIPOS=K+IPOS
      S(KITO)=S(KITO)/S(KIPOS)
      CH=ABS(S(KITO)-SOLD)
      S(KITO)=RELPA*S(KITO)+(1.-RELPA)*SOLD
      IF (CHMAX .LT. CH) CHMAX=CH
   10 J=J+IVSIZ
      IF (CHMAX .LT. ERR) GO TO 14
   11 CONTINUE
   13 FORMAT(' MAXIMUM CHANGE IN UNLUMP ',1F12.8)
      WRITE(OUTLUN,13) CHMAX
   14 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE VCDUMP
$INCLUDE(TRG.DCL)
      WRITE(OUTLUN,1)
    1 FORMAT(' CELL LIST')
      J=1
      DO 2 I=1,NUMC
      JL=J+ICSIZ-1
      WRITE(OUTLUN,3) I,J,(IC(K),K=J,JL)
    2 J=J+ICSIZ
    3 FORMAT(1X,6I10)
      J=1
    4 FORMAT('VERTEX LIST')
      WRITE(OUTLUN,4)
      DO 5 I=1,NUMV
      JL=J+IVSIZ
      WRITE(OUTLUN,6) I,J,(IV(K),K=J,JL)
    6 FORMAT((1X,10I10))
    5 J=J+IVSIZ
      RETURN
      END



      SUBROUTINE VLINK(IVA,IVB)
$INCLUDE(TRG.DCL)^
      NV=IV(IVA+1)
      IF (NV .EQ. 0) GO TO 1
      DO 2 I=1,NV
      IVAI1=IVA+I+1
      IF (IV(IVAI1) .EQ. IVB) RETURN
    2 CONTINUE
    1 IV(IVA+1)=NV+1
      IVANV=IVA+NV
      IV(IVANV+2)=IVB
      RETURN
      END



      SUBROUTINE VSTO(IORIG,NWDS)
$INCLUDE(TRG.DCL)^
      K=IORIG
      J=1
      DO 1 I=1,NUMV
      IV(J)=K
      K=K+NWDS
    1 J=J+IVSIZ
      RETURN
      END
```

END

DATE
FILMED

7-81

DTIC