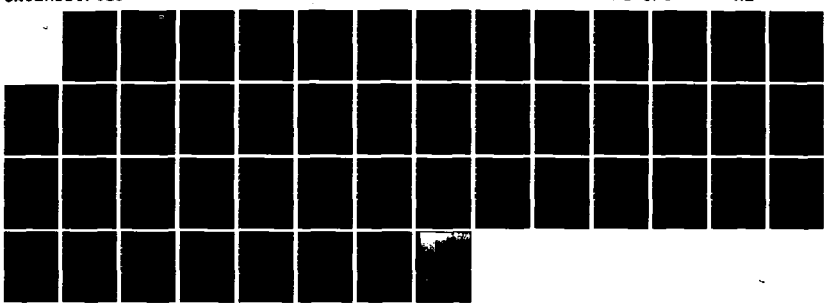
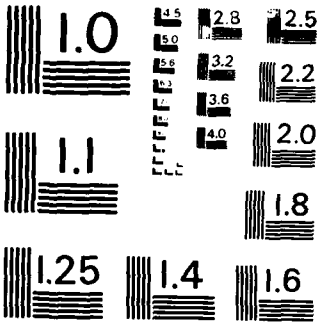
AD-A141 947 A PROCEDURE FOR CALCULATING GROUNDWATER FLOW LINES(U) 1/1
COLD REGIONS RESEARCH AND ENGINEERING LAB HANOVER NH
C J DALY APR 84 CRREL-SR-84-9

UNCLASSIFIED F/G 8/8 NL





Special Report 84-9

April 1984

US Army Corps
of Engineers
Cold Regions Research &
Engineering Laboratory

A procedure for calculating groundwater flow lines

Charles J. Daly

Prepared for
OFFICE OF THE CHIEF OF ENGINEERS
Approved for public release; distribution unlimited

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: Special Report 84-9	2. GOVT ACCESSION NO.: AD-A141947
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: A PROCEDURE FOR CALCULATING GROUNDWATER FLOW LINES	5. TYPE OF REPORT & PERIOD COVERED
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Charles J. Daly	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: U.S. Army Cold Regions Research and Engineering Laboratory, Hanover, NH 03755	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: DA Project 4A161102AT24, Task A, Work Unit 006
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of the Chief of Engineers, Washington, DC 20314	12. REPORT DATE: April 1984
	13. NUMBER OF PAGES: 48
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)*: Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Contaminant transport
Flow lines
Groundwater
Numerical methods

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A methodology for the calculation of flow lines in steady or unsteady two-dimensional velocity fields is described. Although the principal application is intended to be determining fluid particle trajectories in groundwater flow, components of the methodology are relevant to more general problems of fluid flow. Two alternative numerical procedures form the core of the methodology. Each employs the method of characteristics to solve for the advection of fluid particles. The first uses an efficient, fourth-order Runge-Kutta, predictor-corrector algorithm based upon a constant time step. The second uses a fifth-

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

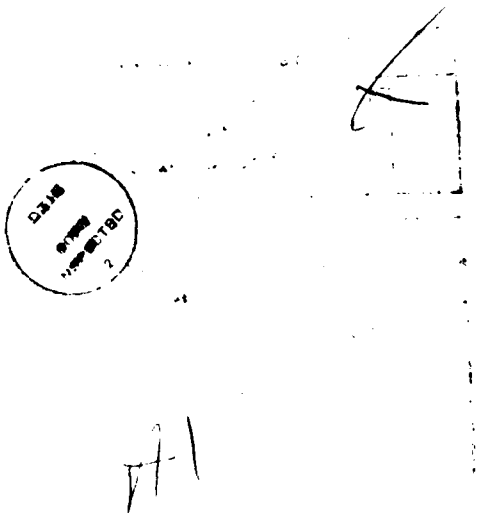
20. Abstract (cont'd).

order Runge-Kutta algorithm incorporating an embedded fourth-order result. This latter alternative includes automatic time-step modification and guarantees a prescribed level of accuracy. Several utility routines are provided in support of the method of characteristics. There is a two-dimensional spline calculation procedure for the analytic description of flow-field parameters, steady potential, and potential gradient. Spline interpolation subroutines enable the user to incorporate function and gradient evaluations directly into computer program code. There is also a routine for calculating average linear velocity for those flow situations where Darcy's law is appropriate. A plotting routine features an option for producing flow-line map overlays to a user-supplied scale.

PREFACE

This report was written by Dr. Charles J. Daly, Hydrologist, Earth Sciences Branch, Research Division, U.S. Army Cold Regions Research and Engineering Laboratory, Hanover, New Hampshire. Funding for the final phase of this investigation was provided through DA Project 4A161102AT24, Task A, Work Unit 006, Chemistry of Freezing and Frozen Soils.

The author wishes to thank Jeff Doyle for performing some of the computer programming involved in the spline generating routines. The author also wishes to thank Steven Daly and Timothy Pangburn of CRREL for technically reviewing the manuscript of this report.



CONTENTS

	Page
Abstract	i
Preface	iii
Overview of modeling	2
Spline generating package	2
Description	2
Step 1	8
Step 2	8
Step 3	9
Step 4	9
Documentation	10
Average linear velocity routine	12
Description	12
Documentation	13
Flow-line determination routines	13
Description	13
Method 1	13
Method 2	15
Documentation	17
Flow-line plotting	18
Description	18
Documentation	18
Application	18
Summary	22
Literature cited	22
Appendix A: Program listing	23

ILLUSTRATIONS

Figure	
1. Schematic of modeling procedure for two-dimensional steady groundwater flow	3
2. Results of model application shown in Figure 1	4
3. Spline function domain	5
4. An arbitrary subrectangle of the domain D	6
5. Base map for demonstration site	19
6. Water-table contours in ft above MSL	20
7. Stream line map	21
8. Composite flow net map	22

TABLES

Table	
1. Function of XYSC and FSCALE	12

A PROCEDURE FOR CALCULATING GROUNDWATER FLOW LINES

by

Charles J. Daly

Any continuous mathematical function of spatial coordinates $\vec{x}$ and time t can constitute a field. Fields may be scalar, vector, or tensor, e.g. hydraulic head $h(\vec{x},t)$, velocity $\vec{v}(\vec{x},t)$, and stress $\tau(\vec{x},t)$ respectively. The scalar field is the most elementary type. Components of vector and tensor fields are themselves scalar fields.

The principles of classical physics generally suffice to describe the motion of a group of individual rigid objects. For example, the velocity of the center of mass of the ith object in a group can be specified as a function of time:

$$\vec{v}_i(t) = (v_{x_i}(t), v_{y_i}(t), v_{z_i}(t)). \tag{1}$$

In considering fluid flow, however, the same principles of classical physics can not be applied to the uncountable number of molecules contained in even a small volume of fluid. Instead, the field approach is employed, and the velocity of fluid particles is specified by the velocity vector field:

$$\vec{v}(\vec{x},t) = (v_x(\vec{x},t), v_y(\vec{x},t), v_z(\vec{x},t)), \tag{2}$$

that is, a fluid particle located at point $\vec{x}_o$ at time t_o has velocity $\vec{v}(\vec{x}_o, t_o)$.

The significance of the velocity vector field can be appreciated from two perspectives. The first, called the Eulerian viewpoint, focuses attention on specific spatial locations $\vec{x}_p$ and observes the velocity of particles moving past those locations. The velocity field is revealed by the set of observations:

$$\vec{v}(\vec{x}_p, t) \qquad p = 1,2,3,... \tag{3}$$

The second perspective, called the Lagrangian viewpoint, focuses on a number of specific particles moving with the flow. Over time each particle follows a trajectory, or flow line, defined by the locus of points:

$$\vec{x}_k(t) = (x_k(t), y_k(t), z_k(t)) \qquad k = 1,2,3,... \tag{4}$$

Monitoring the velocity of each moving particle would give the set of data again revealing the velocity field:

$$\vec{v}\ (\vec{x}_k(t), t) \qquad k = 1,2,3,... \tag{5}$$

The intent of this report is to describe a methodology for determining the flow lines of particles carried along as part of a moving fluid. It is natural then to adopt the Lagrangian viewpoint of the velocity field.

OVERVIEW OF MODELING

Figure 1 is a schematic of the flow-line calculation procedure as applied to the case of steady ground-water flow. The system is assumed to be governed by Darcy's law and the mass conservation principle. The main components are:

1) the spline generator and function subroutines,
2) the average linear velocity calculation subroutine,
3) the flow-line determination routines, and
4) the flow-line plotting routine.

Figure 2 illustrates the results of applying the procedure in Figure 1. The composite map, in this case a flow net, could be used to predict the trajectory of ground-water contamination and the propagation rate of a contamination front. (In practice one must also consider the importance of sorption and mechanical dispersion on the transport (Daly 1983).)

In the sections that follow, each component in the complete methodology of Figure 1 is described, documented, and then demonstrated.

SPLINE GENERATING PACKAGE

Description

Consider the two-dimensional domain, shown in Figure 3, defined by $x_1 \leq x \leq x_N$ and $y_1 \leq y \leq y_M$. Let D denote the domain and ∂D its boundary. Values of a function $f(x,y)$, whose analytic form is unknown, are given at a

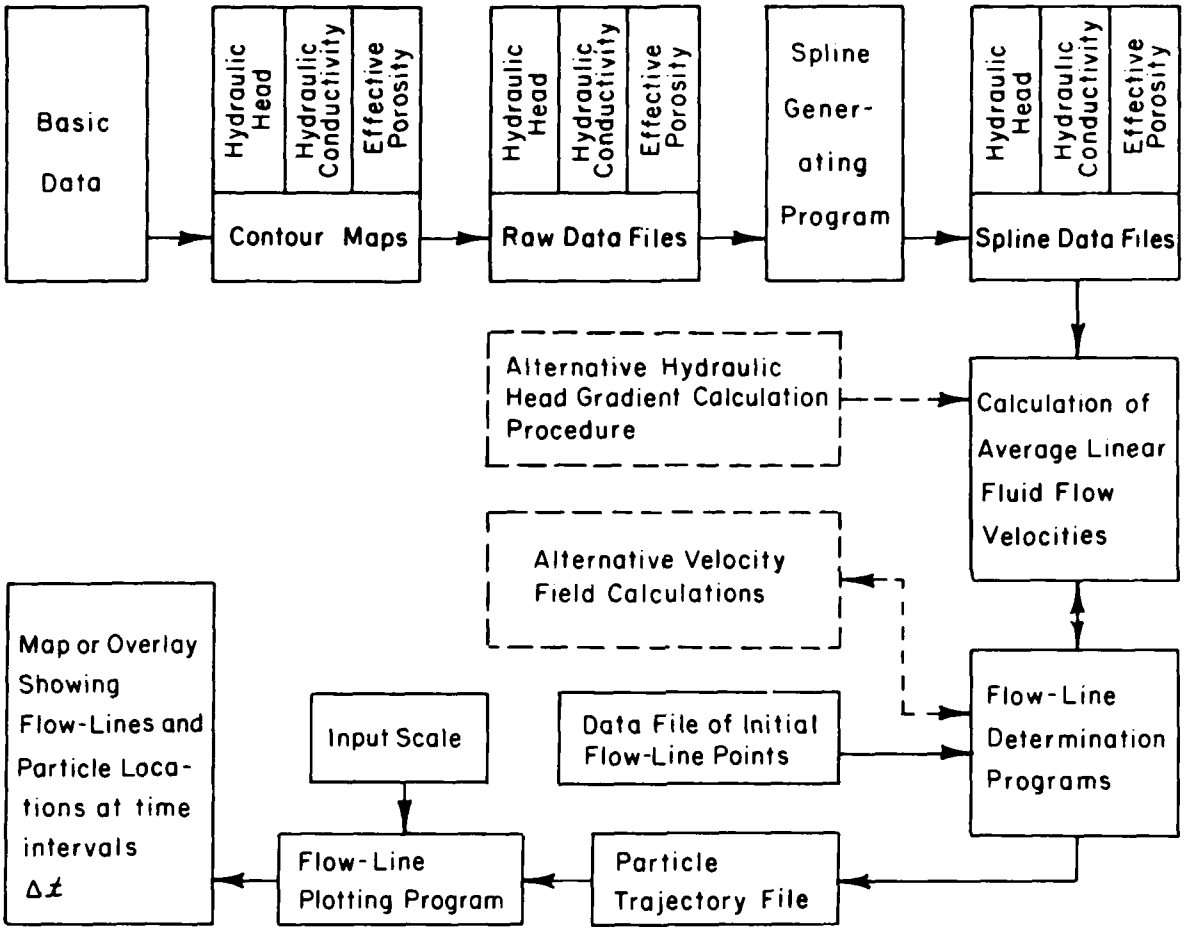


Figure 1. Schematic of modeling procedure for two-dimensional steady groundwater flow.

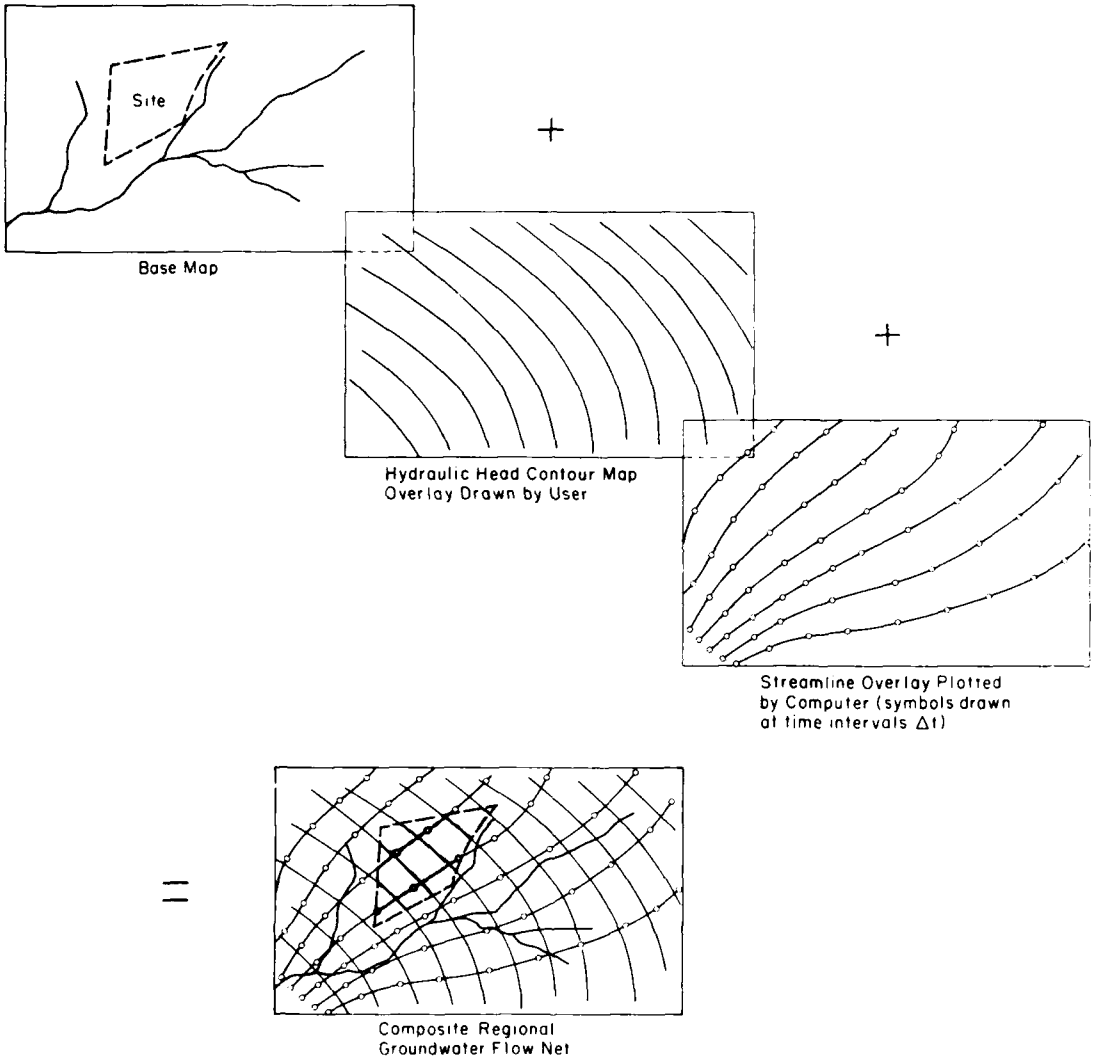


Figure 2. Results of model application shown in Figure 1.

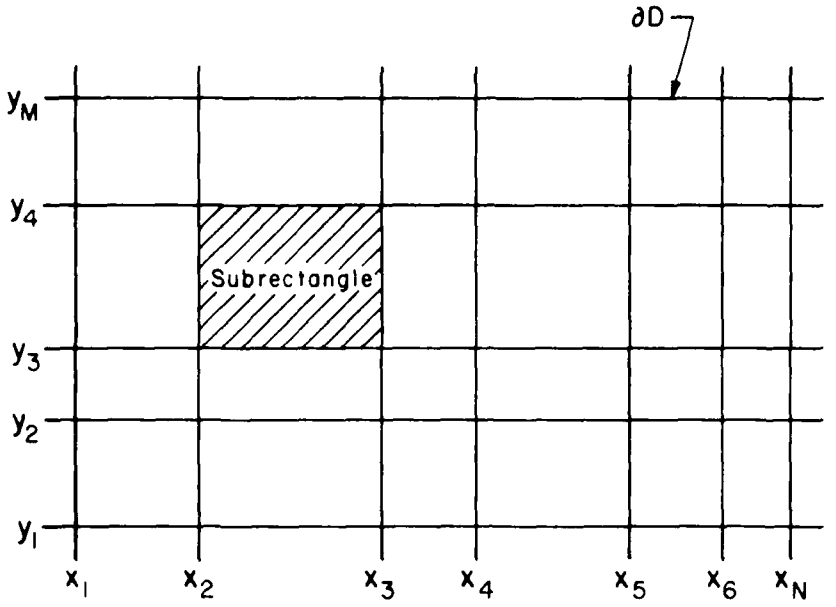


Figure 3. Spline function domain.

number of points in D and on ∂D. These points are arranged so that one point is located at the intersection of every vertical (x = constant) and every horizontal line (y = constant). Each point arranged in this fashion is called a knot. (Regular placement of the knots, as in Figure 3, is not essential to the derivation of a spline interpolate. This assumption does lead, however, to the relatively simple calculation procedure described later.)

Let the approximating expression for the unknown function f(x,y) be designated by S(x,y). The particular form of S(x,y) depends on the set of functions from which it is to be constructed. A variety of so-called basis functions are available for this purpose. In cases where the smoothness of f(x,y) can be estimated, some justification can be made for the selection of particular basis functions. If f(x,y) is believed to vary smoothly, it would appear to be more advantageous to construct S(x,y) from functions exhibiting this property (Rivlin 1969, Schultz 1973).

A spline is a composite function. For domain D of Figure 3, a bicubic spline interpolate S(x,y) will be composed of (N-1) × (M-1) bicubic polynomials. (The polynomials are bicubic since they contain powers of both x and y up to order three.) Each individual polynomial is valid only in and on the boundary of a particular subrectangle of D. Thus there is a one-to-one correspondence between the polynomials and the subrectangles.

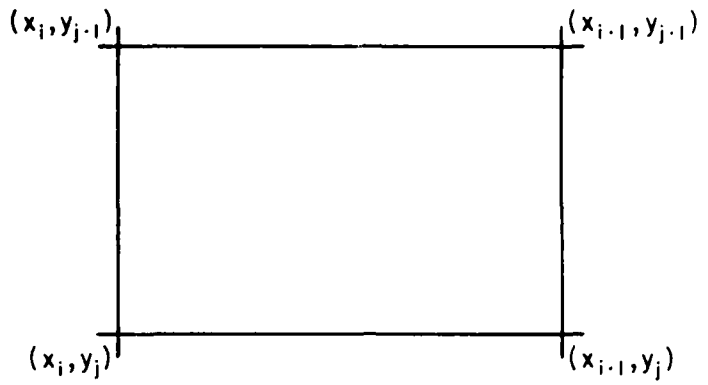


Figure 4. An arbitrary subrectangle of the domain D

Each of the constituent bicubic polynomials has the same s: _ :al formula. This formula can be illustrated by writing the equation of the polynomial associated with the isolated subrectangle of Figure 4. In total, (N-1) × (M-1) of these equations must be written to define the entire spline. For the subrectangle of Figure 4, the polynomial is:

$$
\begin{aligned}
s(x,y) = {} & f(x_i,y_j)C_i(x)C_j(y) + f(x_i,y_{j+1})C_i(x)C_{j+1}(y) \\
& + f(x_{i+1},y_j)C_{i+1}(x)C_j(y) + f(x_{i+1},y_{j+1})C_{i+1}(x)C_{j+1}(y) \\
& + \frac{\partial f}{\partial x}(x_i,y_j)D_i(x)C_j(y) + \frac{\partial f}{\partial x}(x_i,y_{j+1})D_i(x)C_{j+1}(y) \\
& + \frac{\partial f}{\partial x}(x_{i+1},y_j)D_{i+1}(x)C_j(y) + \frac{\partial f}{\partial x}(x_{i+1},y_{j+1})D_{i+1}(x)C_{j+1}(y) \\
& + \frac{\partial f}{\partial y}(x_i,y_j)C_i(x)D_j(y) + \frac{\partial f}{\partial y}(x_i,y_{j+1})C_i(x)D_{j+1}(y) \\
& + \frac{\partial f}{\partial y}(x_{i+1},y_j)C_{i+1}(x)D_j(y) + \frac{\partial f}{\partial y}(x_{i+1},y_{j+1})C_{i+1}(x)D_{j+1}(y) \\
& + \frac{\partial^2 f}{\partial x \partial y}(x_i,y_j)D_i(x)D_j(y) + \frac{\partial^2 f}{\partial x \partial y}(x_i,y_{j+1})D_i(x)D_{j+1}(y) \\
& + \frac{\partial^2 f}{\partial x \partial y}(x_{i+1},y_j)D_{i+1}(x)D_j(y) + \frac{\partial^2 f}{\partial x \partial y}(x_{i+1},y_{j+1})D_{i+1}(x)D_{j+1}(y). \qquad (6)
\end{aligned}
$$

The interpolating functions C_i, C_{i+1}, D_i, and D_{i+1} are defined by the following generalized formulas:

$$
C_i(\vartheta) = \frac{(\vartheta-\vartheta_{i+1})^2}{(\Delta\theta_i)^2} + 2\,\frac{(\theta-\vartheta_i)(\theta-\theta_{i+1})^2}{(\Delta\theta_i)^3} \qquad (7)
$$

$$C_{i+1}(\theta) = \frac{(\theta-\theta_i)^2}{(\Delta\theta_i)^2} - 2\,\frac{(\theta-\theta_{i+1})(\theta-\theta_i)^2}{(\Delta\theta_i)^3} \tag{8}$$

$$D_i(\theta) = \frac{(\theta-\theta_i)(\theta-\theta_{i+1})^2}{(\Delta\theta_i)^2} \tag{9}$$

$$D_{i+1}(\theta) = \frac{(\theta-\theta_{i+1})(\theta-\theta_i)^2}{(\Delta\theta_i)^2} \tag{10}$$

and

$$\Delta\theta_i = \theta_{i+1} - \theta_i. \tag{11}$$

Evaluation of the interpolating functions and their derivatives at the knots yields the important results:

$$C_i(\theta_i) = C_{i+1}(\theta_{i+1}) = \frac{\partial D_i(\theta_i)}{\partial\theta} = \frac{\partial D_{i+1}(\theta_{i+1})}{\partial\theta} = 1 \tag{12}$$

$$C_i(\theta_{i+1}) = C_{i+1}(\theta_i) = \frac{\partial D_i(\theta_{i+1})}{\partial\theta} = \frac{\partial D_{i+1}(\theta_i)}{\partial\theta} = 0 \tag{13}$$

$$D_i(\theta_i) = D_{i+1}(\theta_i) = D_i(\theta_{i+1}) = D_{i+1}(\theta_{i+1}) = 0 \tag{14}$$

$$\frac{\partial C_i(\theta_i)}{\partial\theta} = \frac{\partial C_{i+1}(\theta_i)}{\partial\theta} = \frac{\partial C_i(\theta_{i+1})}{\partial\theta} = \frac{\partial C_{i+1}(\theta_{i+1})}{\partial\theta} = 0. \tag{15}$$

The last four equations can be used to show that the function $s(x,y)$ has the following properties: the values s, $\partial s/\partial x$, $\partial s/\partial y$, and $\partial^2 s/\partial x\partial y$ are equal to the corresponding values f, $\partial f/\partial x$, $\partial f/\partial y$, and $\partial^2 f/\partial x\partial y$ at each of the knots (x_i,y_j), (x_i,y_{j+1}), (x_{i+1},y_j), and (x_{i+1},y_{j+1}).

The first four terms in each bicubic polynomial, represented by eq 6, are known since $f(x,y)$ is given at every knot. The spline determination problem is then to estimate the remaining 12 terms in the equation of each bicubic polynomial. These unknowns are the values of $\partial f/\partial x$, $\partial f/\partial y$, and $\partial^2 f/\partial x\partial y$ at the four corners of every subrectangle. They are found by observing continuity and differentiability conditions at the knots. Since the joint edges of two adjacent subrectangles share common knots, the polynomials in each of these adjacent elements share common unknowns. This fact suggests the solution of simultaneous equations for the unknowns, and this is exactly the method used.

The algebra involved in obtaining the system of equations is quite cumbersome, although the procedure is relatively simple to explain. The procedure consists of four steps.

Step 1

The equation of the bicubic polynomial in each subrectangle is differentiated twice with respect to x. This yields a set of equations for $\partial^2 s(x,y)/\partial x^2$, one equation for each subrectangle of domain D.

A continuity requirement is imposed on the determination of $\partial^2 s/\partial x^2$ at each knot in D. Since a given knot usually belongs to more than one subrectangle, there is in general more than one equation for the determination of $\partial^2 s/\partial x^2$ at a knot. Continuity requires that all of these determinations be equal. Observing the continuity conditions along a single horizontal $y = y_j$ yields a complete set of equations for $\partial f/\partial x$ at each interior knot on that line. (The values of $\partial f/\partial x$ must be given at the end points of that line.) Each horizontal line is considered in succession until $\partial f/\partial x$ has been determined at every knot in D.

Let subscripts i and j be used to refer to the point (x_i, y_j); for example, $f_{i+1,j} = f(x_{i+1}, y_j)$. The systems of equations solved for the values $\partial f/\partial x$ along the horizontals can then be written:

$$\Delta x_{i+1} \frac{\partial f_{ij}}{\partial x} + 2(\Delta x_{i+1} + \Delta x_i) \frac{\partial f_{i+1,j}}{\partial x} + \Delta x_i \frac{\partial f_{i+2,j}}{\partial x} = 3\left[\frac{\Delta x_{i+1}}{\Delta x_i}(f_{i+1,j} - f_{i,j}) + \frac{\Delta x_i}{\Delta x_{i+2}}(f_{i+2,j} - f_{i+1,j})\right] \qquad (16)$$

where $1 \leq i \leq N-2$, $1 \leq j \leq M$, and $\Delta x_i = x_{i+1} - x_i$.

Step 2

A procedure similar to that used in Step 1 is used to determine $\partial f/\partial y$ at every knot in D. First, the equation of each bicubic polynomial is differentiated twice with respect to y, yielding a set of equations for $\partial^2 s(x,y)/\partial y^2$. The separate determinations of $\partial^2 s/\partial y^2$ at a knot must be equal. Observing this requirement along a single vertical line $x = x_i$ gives a complete set of equations for $\partial f/\partial y$ along that line.

Given the values of $\partial f/\partial y$ at the end points of each vertical line enables the solution of each set of simultaneous equations; thus the values $\partial f/\partial y$ at every knot are obtained. The systems of equations can be constructed from:

$$\Delta y_{j+1} \frac{\partial f_{i,j}}{\partial y} + 2(\Delta y_{j+1} + \Delta y_j) \frac{\partial f_{i,j+1}}{\partial y} + \Delta y \frac{\partial f_{i,j+2}}{\partial y} =$$

$$3\left[\frac{\Delta y_{j+1}}{\Delta y_j} (f_{i,j+1} - f_{i,j}) + \frac{\Delta y_j}{\Delta y_{j+1}} (f_{i,j+2} - f_{i,j+1})\right] \qquad (17)$$

where $1 \leq i \leq N$, $1 \leq j \leq M-2$, and $\Delta y_j = v_{j+1} - y_j$.

Step 3

The equations of the bicubic polynomials that are valid along the vertical boundaries of D are differentiated to produce equations for $\partial^3 s(x,y)/\partial x \partial y^2$. Along each of these vertical boundaries the value of $\partial^3 s/\partial x \partial y^2$ must be continuous at the knots. This results in two systems of equations, one for each boundary. When the values $\partial^2 f/\partial x \partial y$ are given at the four corners of D, these two systems can be solved for all other values of $\partial^2 f/\partial x \partial y$ on the vertical boundaries.

The equations are of the form:

$$\Delta y_{j+1} \frac{\partial^2 f_{i,j}}{\partial x \partial y} + 2(\Delta y_{j+1} + \Delta y_j) \frac{\partial^2 f_{i,j+1}}{\partial x \partial y} + \Delta y_j \frac{\partial^2 f_{i,j+2}}{\partial x \partial y} =$$

$$3\left[\frac{\Delta y_{j+1}}{\Delta y_j} \left(\frac{\partial f_{i,j+1}}{\partial x} - \frac{\partial f_{i,j}}{\partial x}\right) + \frac{\Delta y_j}{\Delta y_{j+1}} \left(\frac{\partial f_{i,j+2}}{\partial x} - \frac{\partial f_{i,j+1}}{\partial x}\right)\right] \qquad (18)$$

where $i = 1$ through N, and $1 \leq j \leq M-2$.

Step 4

The equations of all the bicubic polynomials are differentiated to give $\partial^3 s(x,y)/\partial x^2 \partial y$ in each subrectangle. Continuity of $\partial^3 s/\partial x^2 \partial y$ along each horizontal line leads to a set of simultaneous equations for $\partial f/\partial x \partial y$ at all the interior points of each line. The necessary values $\partial f/\partial x \partial y$ at the ends of each horizontal line were obtained in Step 3.

The equations to be solved can be written:

$$\Delta x_{i+1} \frac{\partial^2 f_{i,j}}{\partial x \partial y} + 2(\Delta x_{i+1} + \Delta x_i) \frac{\partial^2 f_{i+1,j}}{\partial x \partial y} + \Delta x_i \frac{\partial^2 f_{i+2,j}}{\partial x \partial y} =$$

$$3\left[\frac{\Delta x_{i+1}}{\Delta x_i} \left(\frac{\partial f_{i+1,j}}{\partial y} - \frac{\partial f_{i,j}}{\partial y}\right) + \frac{\Delta x_i}{\Delta x_{i+1}} \left(\frac{\partial f_{i+2,j}}{\partial y} - \frac{\partial f_{i+1,j}}{\partial y}\right)\right] \qquad (19)$$

where $1 \leq i \leq N-2$ and $1 \leq j \leq M$.

Since each part of the calculation procedure involves the solution of simultaneous tridiagonal equations, it is relatively easy to compute the

spline by Gaussian elimination. Moreover, if it is recognized that the tridiagonal matrices of Steps 1 and 4, as well as those of Steps 2 and 3, are identical, the number of Gauss elimination procedures can be cut in half. (This is equivalent to the statement that if $A\vec{x} = \vec{b}$ and $A\vec{y} = \vec{c}$ are to be solved for $\vec{x}$ and $\vec{y}$, then A^{-1} need be determined only once.)

Documentation

This section documents the two-dimensional spline interpolating package implemented in Fortran IV. Subroutines within the package are able to return interpolated values of a function and its first partial derivatives. The interpolating functions are defined within a domain covered by a rectangular grid in the x,y plane. Values of the function to be interpolated are known at the grid intersections on and within the boundary of the domain. Values of the first derivatives are known for grid intersections on the boundary, and values of the second derivative $\partial^2 f/\partial x \partial y$ are known at the four corners of the domain.

Using the complete interpolation package is a two-step process. First, the user creates a file containing raw input data and runs the CREATE program to create a binary file containing spline data. Employing the spline data, interpolating functions may then be called from the user's own Fortran programs like any other Fortran real-valued function. The raw input data file must contain the following information in the order shown using free-formatting conventions:

N, M

$x_1, x_2, x_3 \cdots x_N$

$y_1, y_2, y_3 \cdots y_M$

$f(x_1,y_1),\ f(x_2,y_1),\ f(x_3,y_1) \cdots \quad f(x_N,y_1)$

$\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots$

$f(x_1,y_M) \cdots \qquad\qquad f(x_N,y_M)$

$\frac{\partial f}{\partial x}(x_1,y_1) \cdots \qquad \frac{\partial f}{\partial x}(x_1,y_M)$

$\frac{\partial f}{\partial x}(x_N,y_1) \cdots \qquad \frac{\partial f}{\partial x}(x_N,y_M)$

$\frac{\partial f}{\partial y}(x_1,y_1) \cdots \qquad \frac{\partial f}{\partial y}(x_N,y_1)$

$$\frac{\partial f}{\partial y}(x_1,y_M) \cdots \qquad \frac{\partial f}{\partial y}(x_N,y_M)$$

$$\frac{\partial^2 f}{\partial x \partial y}(1,1),\ \frac{\partial^2 f}{\partial x \partial y}(x_N,1),\ \frac{\partial^2 f}{\partial x \partial y}(1,y_M),\ \frac{\partial^2 f}{\partial x \partial y}(x_N,y_M)$$

where N and M are integers denoting the number of vertical and horizontal lines in the grid. Though N and M must be integers, the other data may take any legal single-precision real value (N and M must both be ≥ 4). x_i and y_i need not be regularly spaced; the grid can be made finer in areas where greater resolution is required.

Two additional inputs, XYSC and FSCALE, allow for data scaling. The scaling parameters allow for more convenient input of raw data. Their function is most clearly explained by Table 1. If XYSC and FSCALE are omitted, their default value is 1.0.

Execution of CREATE generates the spline data file subsequently referenced by the interpolation functions. The CREATE program is designed to operate through the intermediary of an interactive terminal. It asks for the name of the raw data file and the name of an output file in which to store the binary spline data it generates. If all goes well, CREATE will terminate with the message 'normal termination.' Minor code revisions can easily be made to alter CREATE input/output functions. The CREATE program listing is provided in the appendix.

The interpolation functions SPLINE, DSDX, and DSDY return the value of the spline S and its first partial derivatives. Calls to the functions may be incorporated into Fortran code, as in the following examples evaluated at x = X1, y = Y1:

```
HEIGHT = SPLINE (X1,Y1,N)
XSLOPE = DSDX (X1,Y1,N)
YSLOPE = DSDY (X1,Y1,N)
GRADM = SQRT (DSDX (X1,Y1,N)**2 + DSDY (X1,Y1,N)**2)
```

Each of the interpolation functions requires the spline data file that CREATE produced. Before calling any of the functions, spline data must be read into a COMMON block by a single call to subroutine SPLOAD:

```
CALL SPLOAD (<infile>,n)
```

where <infile> is the Fortran unit number assigned to the spline data file. The parameter n specifies one of up to three separate spline data file locations in COMMON, i.e. n may be chosen by the user as equal to either 1,

Table 1. Function of XYSC and FSCALE.

input to CREATE	=	factor	×	data in file
x	=	XYSC	×	x
y	=	XYSC	×	y
$\frac{\partial f}{\partial x}$	=	$\frac{\text{FSCALE}}{\text{XYSC}}$	×	$\frac{\partial f}{\partial x}$
$\frac{\partial f}{\partial y}$	=	$\frac{\text{FSCALE}}{\text{XYSC}}$	×	$\frac{\partial f}{\partial y}$
$\frac{\partial^2 f}{\partial x \partial y}$	=	$\frac{\text{FSCALE}}{\text{XYSC} \times \text{XYSC}}$	×	$\frac{\partial^2 f}{\partial x \partial y}$

2, or 3. Thus, the same program code can call upon up to three spline functions.

SPLOAD, SPLINE, DSDX, and DSDY are listed in Appendix A, and a sample, called PROGRAM, demonstrates the use of these routines.

AVERAGE LINEAR VELOCITY ROUTINE

Description

The velocity field is an essential input to the calculation of fluid particle trajectory. For so-called laminar flow in porous media, this information is supplied by the average linear velocity, that is, an average velocity of the particles being carried along through the pores:

$$\vec{v}^* = - \frac{K}{\phi} \nabla h \tag{20}$$

where K is the medium hydraulic conductivity, ϕ is the effective porosity, and ∇h is the gradient of hydraulic head h.

Equation 20 presents a somewhat idealized view of fluid flow in porous media. The actual speed of particles moving through the void spaces ranges from near zero at the grain boundaries to many times the magnitude of $\vec{v}^*$ in the larger spaces between grains. This phenomenon accounts for some of the observed smearing of sharp solute fronts in porous media flows. It appears, however, that the effect is less consequential given only the approximate descriptions of the hydraulic head, hydraulic conductivity, and effective porosity fields that are normally available.

Documentation

Fortran program FG is written to supply the Cartesian components of a two-dimensional velocity field according to eq 20. Used in conjunction with a program called MAIN, FG requires spline data files for h (n=1), K (n=2), and ϕ (n=3). Listings of programs MAIN and FG are included in Appendix A.

FLOW-LINE DETERMINATION ROUTINES

Description

The equation of continuity for two-dimensional flow is:

$$\frac{\partial}{\partial x}\left(\rho v_x\right) + \frac{\partial}{\partial y}\left(\rho v_y\right) + \frac{\partial \rho}{\partial t} = 0 \tag{21}$$

where ρ is fluid density and t is time. Using the characteristics approach and assuming constant density, eq 21 becomes the linked system of ordinary differential equations:

$$v_x = \frac{dx}{dt} \qquad v_y = \frac{dy}{dt}$$

$$\frac{d\rho}{dt} = -\rho\left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}\right) = 0. \tag{22}$$

The first two equations define a flow line, while the last simply states the assumption that fluid density is constant along a flow line. The problem of determining the flow line becomes one of solving the joint equations

$$f(x,y,t) = \frac{dx}{dt} = v_x \qquad \frac{dy}{dt} = v_y = g(x,y,t) \tag{23}$$

for x(t) and y(t).

Two alternative methods for solving eq 23 are proposed.

Method 1

A fourth-order Runge-Kutta technique is used to start the procedure, which is continued by a more efficient predictor-corrector scheme. The numerical solution method starts at point (x_0,y_0) at time zero. Using a time step Δt, successive points (x_n, y_n) along the trajectory of the particle, which began at point (x_0, y_0), are obtained. The Runge-Kutta algorithm for accomplishing this is

$$x_{n+1} = x_n + \frac{1}{6}(a_1 + 2a_2 + 2a_3 + a_4) \tag{24}$$

$$y_{n+1} = y_n + \frac{1}{6}(b_1 + 2b_2 + 2b_3 + b_4) \tag{25}$$

where:

$$a_1 = \Delta t\, f(x_n, y_n, t_n) \tag{26}$$

$$b_1 = \Delta t\, g(x_n, y_n, t_n) \tag{27}$$

$$a_2 = \Delta t\, f(x_n + a_1/2,\ y_n + b_1/2,\ t_n + \Delta t/2) \tag{28}$$

$$b_2 = \Delta t\, g(x_n + a_1/2,\ y_n + b_1/2,\ t_n + \Delta t/2) \tag{29}$$

$$a_3 = \Delta t\, f(x_n + a_2/2,\ y_n + b_2/2,\ t_n + \Delta t/2) \tag{30}$$

$$b_3 = \Delta t\, g(x_n + a_2/2,\ y_n + b_2/2,\ t_n + \Delta t/2) \tag{31}$$

$$a_4 = \Delta t\, f(x_n + a_3,\ y_n + b_3,\ t_n + \Delta t) \tag{32}$$

$$b_4 = \Delta t\, g(x_n + a_3,\ y_n + b_3,\ t_n + \Delta t), \tag{33}$$

and f and g are defined in eq 23.

Runge-Kutta algorithms belong to the set of self-starting numerical solution methods. Self-starting means that the determination of all successive points (x_n, y_n) requires only the starting point (x_0, y_0). In other In other words, calculation of x_n and y_n depends only on the known values x_{n-1} and y_{n-1}. The set of nonself-starting methods requires the values (x_n, y_n) to be given at more than one point along the trajectory. For example, a fourth-order predictor-corrector algorithm, called Milne's method, requires the values x_0, x_1, x_2, x_3, y_0, y_1, y_2, and y_3 to calculate successive values of x_n and y_n.

In addition to the question of starting values, the efficiency of the calculation procedure is an important factor in selecting a numerical method. It turns out that Milne's algorithm is significantly more efficient than the Runge-Kutta method, although both are fourth-order accurate. Method 1 takes advantage of the Runge-Kutta self-starting feature and the efficiency of Milne's method. Given a starting point (x_0, y_0), the Runge-Kutta procedure is used to obtain (x_1, y_1), (x_2, y_2), and (x_3, y_3). At that stage the necessary starting values are available for Milne's method, which is then used to generate succeeding points.

Milne's predictor-corrector method consists of two steps. First, predicted estimates of x_{n+1} and y_{n+1} are calculated. Let these be denoted x^*_{n+1} and y^*_{n+1}. Second, the predicted values are corrected to obtain the final values x_{n+1} and y_{n+1} at the end of a time step. The algorithm is

$$x^*_{n+1} = x_{n-3} + \frac{4\Delta t}{3}\left[2x'_n - x'_{n-1} + 2x'_{n-2}\right] \tag{34}$$

$$y^*_{n+1} = y_{n-3} + \frac{4\Delta t}{3}\left[2y'_n - y'_{n-1} + 2y'_{n-2}\right]. \tag{35}$$

Then

$$x_{n+1} = x_{n-1} + \frac{\Delta t}{3}\left[x^{*\prime}_{n+1} + 4x'_n + x'_{n-1}\right] \tag{36}$$

$$y_{n+1} = y_{n-1} + \frac{\Delta t}{3}\left[y^{*\prime}_{n+1} + 4y'_n + y'_{n-1}\right] \tag{37}$$

where:

$$x'_n = f(x_n, y_n, t_n) \tag{38}$$

$$y'_n = g(x_n, y_n, t_n). \tag{39}$$

Consideration of the Runge-Kutta and Milne algorithms shows that Milne's method requires only two evaluations of f and g per time step, whereas Runge-Kutta requires four. This makes Milne's method more efficient.

Nonself-starting methods typically assume constant Δt, whereas self-starting methods allow for change of Δt at each time step. For problems in which the frequent change of Δt is desirable, exclusive use of a self-starting method, such as a Runge-Kutta algorithm, is advised.

Method 2

The algorithms of Method 1 do not permit the stepwise estimation of accuracy, nor do they allow for time-step modification in response to algorithm performance. Method 2, however, incorporates both of these features.

Consider the following Runge-Kutta algorithm from Sarafyan (1966):

$$x^{(4)}_{n+1} = x_n + \frac{1}{6}\left(a_1 + 4a_3 + a_4\right) \tag{40}$$

$$x_{n+1}^{(6)} = x_n + \frac{1}{336}\,(14a_1 + 35a_4 + 162a_5 + 125a_6) \tag{41}$$

$$y_{n+1}^{(4)} = y_n + \frac{1}{6}\,(b_1 + 4b_3 + b_4) \tag{42}$$

$$y_{n+1}^{(6)} = y_n + \frac{1}{336}\,(14b_1 + 35b_4 + 162b_5 + 125b_6) \tag{43}$$

$$a_1 = \Delta t\, f(x_n, y_n, t_n) \tag{44}$$

$$b_1 = \Delta t\, g(x_n, y_n, t_n) \tag{45}$$

$$a_2 = \Delta t\, f(x_n + a_1/2,\ y_n + b_1/2,\ t_n + \Delta t/2) \tag{46}$$

$$b_2 = \Delta t\, g(x_n + a_1/2,\ y_n + b_1/2,\ t_n + \Delta t/2) \tag{47}$$

$$a_3 = \Delta t\, f(x_n + a_1/4 + a_2/4,\ y_n + b_1/4 + b_2/4,\ t_n + \Delta t/2) \tag{48}$$

$$b_3 = \Delta t\, g(x_n + a_1/4 + a_2/4,\ y_n + b_1/4 + b_2/4,\ t_n + \Delta t/2) \tag{49}$$

$$a_4 = \Delta t\, f(x_n - a_2 + 2a_3,\ y_n - b_2 + 2b_3,\ t_n + \Delta t) \tag{50}$$

$$b_4 = \Delta t\, g(x_n - a_2 + 2a_3,\ y_n - b_2 + 2b_3,\ t_n + \Delta t) \tag{51}$$

$$a_5 = \Delta t\, f(x_n + 7a_1/27 + 10a_2/27 + a_4/27,\ y_n + 7b_1/27 + 10b_2/27 + b_4/27,\ t_n + 2\,\Delta t/3) \tag{52}$$

$$b_5 = \Delta t\, g(x_n + 7a_1/27 + 10a_2/27 + a_4/27,\ y_n + 7b_1/27 + 10b_2/27 + b_4/27,\ t_n + 2\,\Delta t/3) \tag{53}$$

$$\xi = x_n + \frac{1}{625}\,(28a_1 - 125a_2 + 546a_3 + 54a_4 - 378a_5) \tag{54}$$

$$\eta = y_n + \frac{1}{625}\,(28b_1 - 125b_2 + 546b_3 + 54b_4 - 378b_5) \tag{55}$$

$$a_6 = \Delta t\, f(\xi, \eta, t_n + \Delta t/5) \tag{56}$$

$$b_6 = \Delta t\, g(\xi, \eta, t_n + \Delta t/5). \tag{57}$$

As eqs 40 through 43 indicate, the Sarafyan algorithm is an embedded Runge-Kutta scheme, that is, the same information used to calculate the fifth-order result $x_{n+1}^{(6)}$ can be used to calculate an embedded fourth-order result $x_{n+1}^{(4)}$ with essentially no extra effort. When, after a given time

step Δt, $x_{n+1}^{(6)}$ and $x_{n+1}^{(4)}$ agree to m significant digits, $x_{n+1}^{(6)}$ must be accurate to that number of digits. This feature of the embedded algorithm is used to increase or decrease the time step so as to balance efficiency and accuracy.

Cartesian coordinates (x,y) along a flow line are calculated at points in time separated by time steps. The degree of detail desired for plotting a flow line will determine a selected maximum time step ΔT, chosen independently of accuracy considerations. For each interval $(t, t + \Delta T)$, Method 2 automatically determines a constant Δt that will guarantee that every successive result within the interval is accurate to a specified number of significant digits m. The rule for dividing the interval $(t, t + \Delta T)$ requires that

$$\Delta T = 2^k \Delta t \tag{58}$$

where k may equal 0, 1, 2, 3, or 4. When accuracy consistently exceeds m+1 significant digits throughout a time interval ΔT, the time step Δt is doubled (but never exceeds ΔT).

Documentation

The approaches of both Methods 1 and 2 are implemented in the form of the Fortran subroutines listed in Appendix A. The two subroutines are fully interchangeable, and operate in conjunction with the MAIN and FG programs.

MAIN serves to supply either method with necessary inputs. Execution of MAIN asks the user to give the file names of spline data files for h, K, and ϕ, as previously described. It then asks for an input file name containing data on time-step size, scale, flow region boundaries, and origins for the flow lines to be calculated. Finally, MAIN asks for a file name in which to output the computed points along the flow lines. The input file has the following free-format structure:

Δt	(or ΔT for Method 2); time step
XYSC;	scaling factor for x and y data
x_1, x_n, y_1, y_m;	flow boundaries
x_0, y_0;	origin of flow line 1
.	.
.	.
:	.
x_0, y_0;	origin of flow line Z

The scale factor multiplies all x_0, y_0 data and flow boundary data in the file. The MAIN program is designed to continue reading origins until it encounters an end-of-file. Calls from MAIN to either Method 1 or 2 subroutines include passing x_0, y_0 and Δt (ΔT). Either subroutine outputs flow-line point data to the specified file at intervals Δt (ΔT). Calls to subroutine FG originate from Method 1 or 2 routines.

FLOW-LINE PLOTTING

Description

A plotting routine, listed in Appendix A, can be used to display output from the method of characteristics solvers. Plot scale is set by the user at the time of plot execution. This scaling feature, used with transparent drawing films or tracing papers, enables the user to produce flow-line overlays at the scale of an appropriate base map (see Fig. 2).

Documentation

Program SPLOT is designed to be executed interactively. The user must provide the name of a file that contains output from either method of characteristics and a plot scale.

Coordinate data stored in the flow-line file are given at the actual base map scale, i.e. multiplied by scaling factor XYSC (see Documentation, above). That data may be in feet, meters, miles, etc. The scale factor for plotting (SCALE) has the interpretation:

1 inch of plotting paper = (SCALE) units of the base map

If coordinate data are expressed in feet, then SCALE = 2000 gives a plot for which 1 in. of paper equals 2000 ft.

Symbols are drawn showing the location of a fluid particle at time intervals whose duration is printed on the plot. The user can choose to have symbols drawn at every Nth point of the flow line by specifying N; all points are plotted, but only every Nth is given a symbol.

APPLICATION

The flow line calculation and plotting package is demonstrated by considering a potential groundwater contamination problem at the hypothetical site shown in Figure 5.

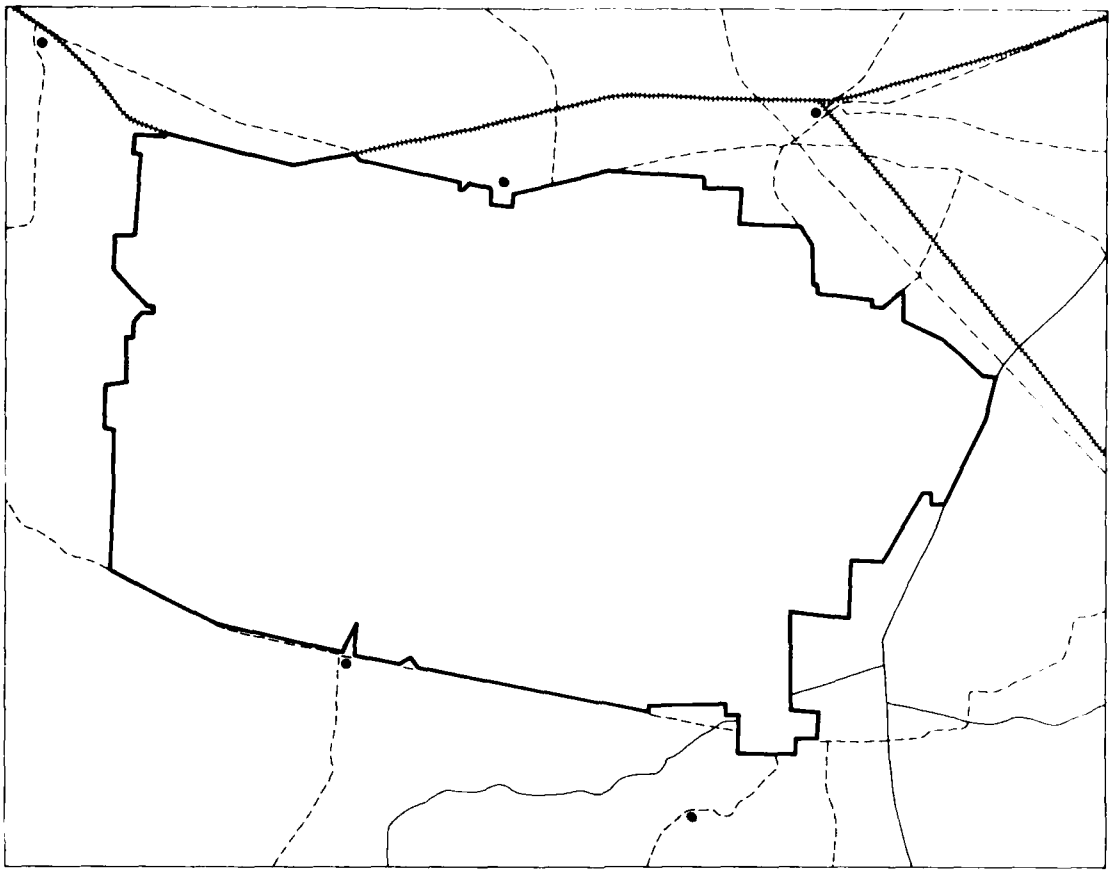


Figure 5. Base map for demonstration site.

Figure 6 is a contour map of steady water-table elevations (in feet above MSL) in the study area. A water-table contour map may be constructed by interpreting static water levels at a number of wells, by observing topographic and stream elevation data, and by considering the drainage behavior of the study region. The persistence of runoff after rainfall events is an indicator of the depth to the water table in the vicinity of streams; sustained baseflows generally indicate a high water table.

Often it can be assumed that although water table elevations may gradually change, the hydraulic gradient remains fairly constant (steady state or quasi-steady state). This may be a valid assumption over considerable lengths of time, up to many decades.

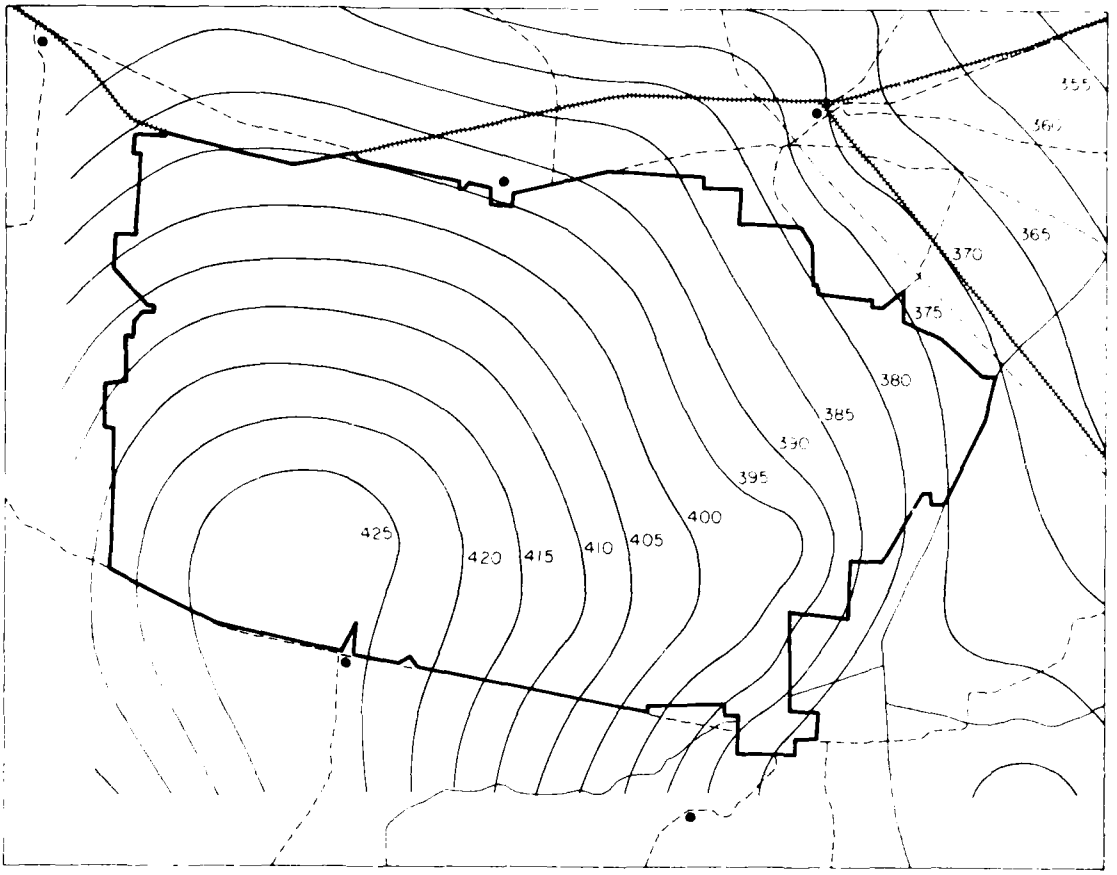


Figure 6. Water table contours in ft above MSL.

For the purpose of demonstration, hydraulic conductivity was chosen to vary over the study area from 30 ft/day in the south to 50 ft/day in the north. Effective porosity was selected as 0.1 throughout the study area. In actual practice, field surveys should be conducted to define clearly the spatial variability of these quantities.

Raw data files for head, conductivity, and effective porosity were prepared. In the case of hydraulic head, data were specified on a uniform grid $\Delta x = \Delta y = 2000$ ft. Far less detailed grids were required to describe K and ϕ. The three raw data files were analyzed by CREATE to generate spline data files.

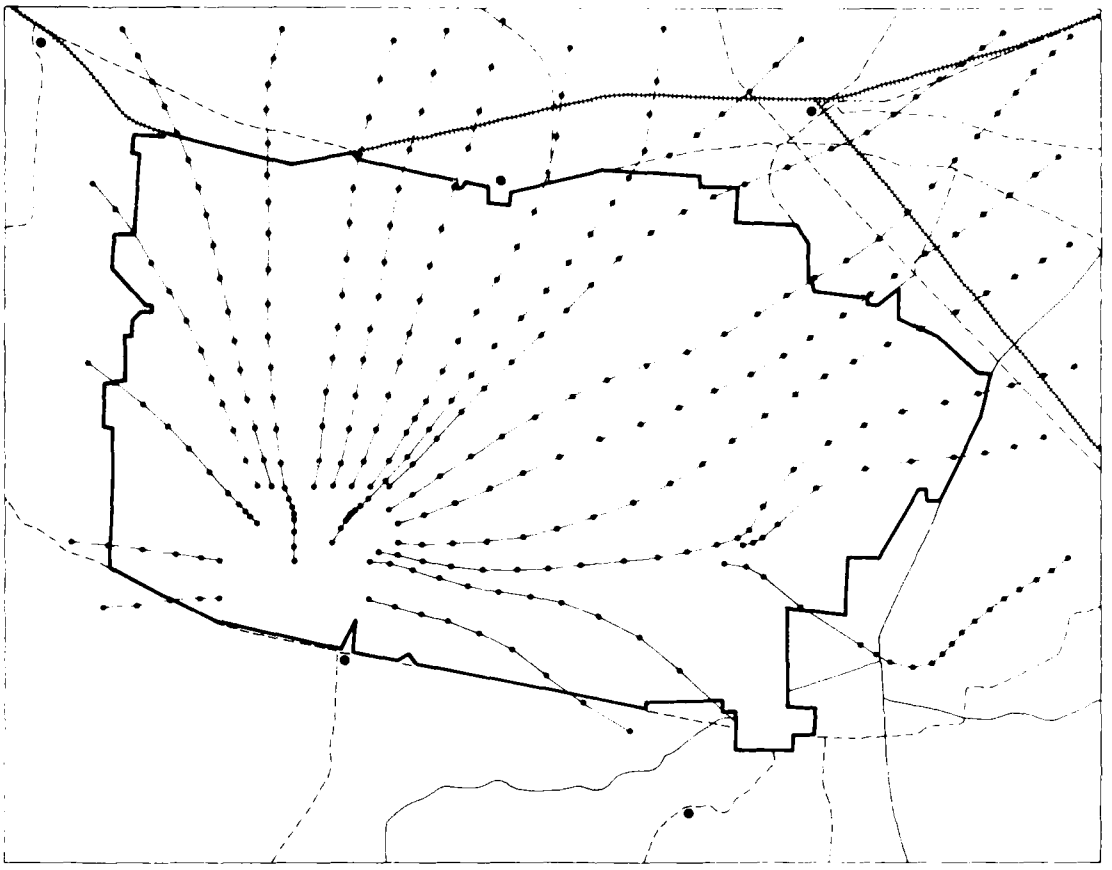


Figure 7. Stream line map.

Flow lines were calculated using Method 1, Δt = 365.25 days, and Method 2, ΔT = 2922 days. Occasionally, Method 2 required shorter time steps of 1461 and 730.5 days to maintain accuracy to three significant digits. Outputs of both methods were in very close agreement, comparing the values of x and y coordinates at time steps of 2922 days (the interval between outputs using Method 2). The results of Method 2 are plotted in Figure 7. Boxes around the data points locate fluid particles along stream lines at time intervals of 2922 days. An overlay of Figure 7 on the base map can be used to trace contaminant transport in the study area; the composite map is shown as Figure 8.

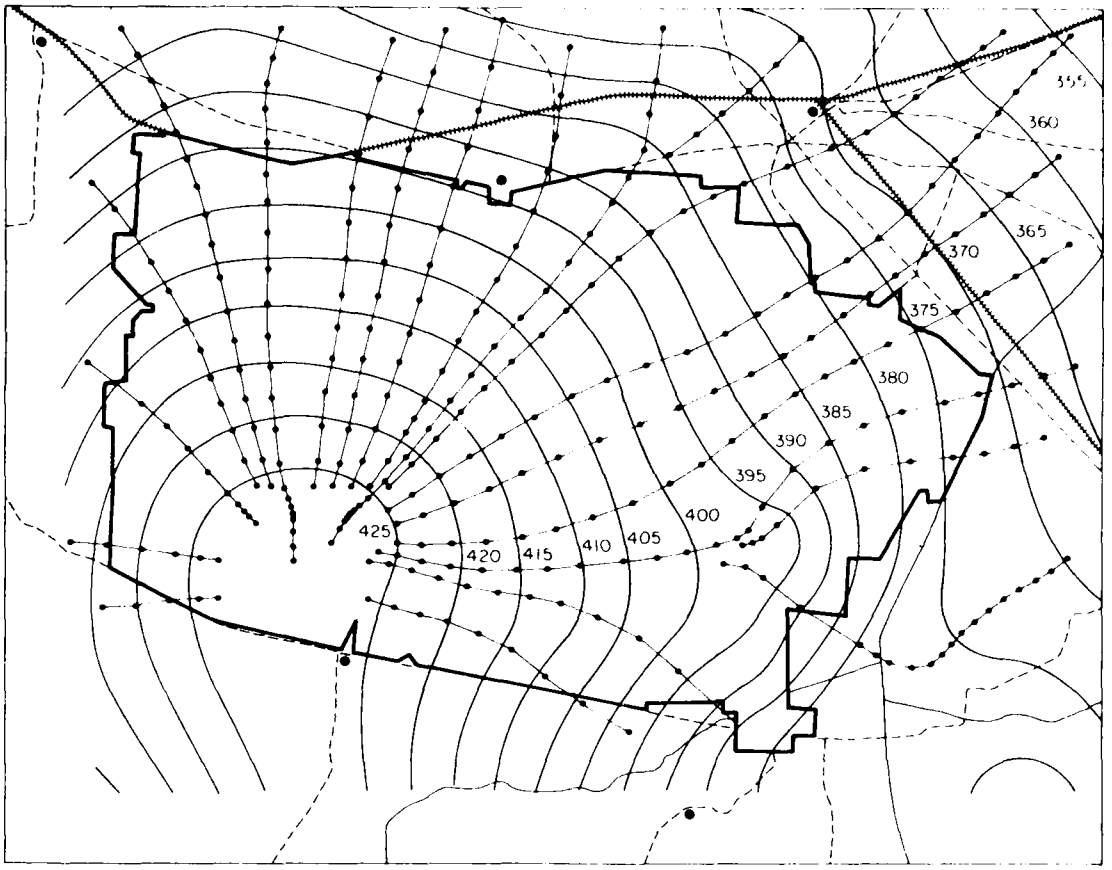


Figure 8. Composite flow net map.

SUMMARY

A set of Fortran computer programs for the calculation of flow lines in two-dimensional velocity fields has been described, documented, and demonstrated. The approach has been shown to have useful application to the prediction of contaminant transport in groundwater flow systems.

LITERATURE CITED

Daly, C.J. (1983) Calculation of advective mass transport in heterogeneous media, Transactions of the 28th Conference of Army Mathematicians, Army Research Office Report 83-1.

Rivlin, T.J. (1969) An Introduction to the Approximation of Functions, Blaisdell Publishing, 150 pp.

Sarafyan, D. (1966) Error estimation for Runge-Kutta methods through pseudo-iterative formulas, Louisiana State University, Technical Report 14, May.

Schultz, M.H. (1973) Spline Analysis. New York: Prentice Hall Publishers, 156 pp.

```
C       *****  PROGRAM CREATE  *****
C
C       This routine reads a file of raw data, analyses it, and
C       generates an output file of spline data.

        COMMON /SPL/ F(50,50),                          /* f(x,y)
       1             DFDX(50,50),                       /* df/dx (x,y)
       1             DFDY(50,50),                       /* df/dy (x,y)
       1             DFDXY(50,50),                      /* d2f/dxdy (x,y)
       1             X(50),                             /* x grid values
       1             Y(50),                             /* y grid values
       1             DELX(50),                          /* delta x along grid
       1             DELY(50)                           /* delta y along grid

        INTEGER*2 INFILE                /* logical unit # of input file
        INTEGER*2 OUTFIL                /* logical unit # of output file

        PARAMETER INFILE = 5             /* only file unit we open
        PARAMETER OUTFIL = 5

C       MNEMONIC CODES FOR CREEL FILE CALLS
        INTEGER*2 J$READ,J$WRIT,J$RDWR,J$CLOS,J$FRWD,J$RWND

        PARAMETER
       1     J$READ  = :1,       /* READ
       1     J$WRIT  = :2,       /* WRITE
       1     J$RDWR  = :3,       /* READ AND WRITE
       1     J$CLOS  = :4,       /* CLOSE FILE
       1     J$DELE  = :5,       /* DELETE FILE
       1     J$FRWD  = :6,       /* MOVE FORWARD ONE FILE (MAG TAPE)
       1     J$RWND  = :7        /* REWIND


C       Open input file
C       Get filename from user

        CALL GFILE$(INFILE,J$READ,'Spline data infile',18)


C       Get data from infile

        READ(INFILE,*) NX,NY                            /* x,y dimensions
        READ(INFILE,*) (X(I),I=1,NX)                    /* read x values
        READ(INFILE,*) (Y(I),I=1,NY)                    /* read y values
        READ(INFILE,*) ((F(I,J),I=1,NX),J=1,NY) /* f(x,y) at knots
        READ(INFILE,*) (DFDX(1,J),J=1,NY)       /* df/dx at left side
        READ(INFILE,*) (DFDX(NX,J),J=1,NY)      /* df/dx at right side
        READ(INFILE,*) (DFDY(I,1),I=1,NX)       /* df/dy at bottom
        READ(INFILE,*) (DFDY(I,NY),I=1,NX)      /* df/dy at top
        READ(INFILE,*) DFDXY(1,1),DFDXY(NX,1), /* d2f/dxdy at corners
       1               DFDXY(1,NY),DFDXY(NX,NY)

C       Read scaling factors from the file (if present)
        READ(INFILE,*,END=4)XYSC
        GO TO 1
4       XYSC=1.0
1       READ(INFILE,*,END=6)FSCALE
        GO TO 5
6       FSCALE=1.0

C       Close input file
5       CALL FILES$(INFILE,J$CLOS)
```

```
C       USE COORDINATE SCALE FACTOR TO SCALE INPUT X'S AND Y'S
C       AND FUNCTION SCALE FACTOR FOR F AND ITS DERIVATIVES
        DO 2 I=1,NX
2       X(I)=X(I)*XYSC
        DO 3 I=1,NY
3       Y(I)=Y(I)*XYSC
        DO 9 I=1,NX
        DO 9 J=1,NY
9       F(I,J)=F(I,J)*FSCALE
        DO 7 I=1,NX
        DFDY(I,1)=DFDY(I,1)*FSCALE/XYSC
7       DFDY(I,NY)=DFDY(I,NY)*FSCALE/XYSC
        DO 8 I=1,NY
        DFDX(1,I)=DFDX(1,I)*FSCALE/XYSC
8       DFDX(NX,I)=DFDX(NX,I)*FSCALE/XYSC
        XYSC2=XYSC*XYSC
        DFDXY(1,1)=DFDXY(1,1)*FSCALE/XYSC2
        DFDXY(1,NY)=DFDXY(1,NY)*FSCALE/XYSC2
        DFDXY(NX,1)=DFDXY(NX,1)*FSCALE/XYSC2
        DFDXY(NX,NY)=DFDXY(NX,NY)*FSCALE/XYSC2

C       Open output file
C       Get filename from user

        CALL GFILE$(OUTFIL,J$WRIT,'Output file name',16)

C       Call spline to generate matrix values
        CALL SPLNXY(NX,NY)

C       Write matrices to output file

        WRITE (OUTFIL) NX,NY
        WRITE (OUTFIL) ((F(I,J),I=1,NX),J=1,NY)
        WRITE (OUTFIL) ((DFDX(I,J),I=1,NX),J=1,NY)
        WRITE (OUTFIL) ((DFDY(I,J),I=1,NX),J=1,NY)
        WRITE (OUTFIL) ((DFDXY(I,J),I=1,NX),J=1,NY)
        WRITE (OUTFIL) (X(I),I=1,NX)
        WRITE (OUTFIL) (Y(I),I=1,NY)
        NXM1=NX-1
        NYM1=NY-1
        WRITE (OUTFIL) (DELX(I),I=1,NXM1)
        WRITE (OUTFIL) (DELY(I),I=1,NYM1)

C       Close output file

        CALL FILES$(OUTFIL,J$CLOS)                  /* close output file

C       Print message to terminal indicating normal termination.
        CALL TNOU ('normal termination',18)

999     CALL EXIT

        END


        SUBROUTINE SPLNXY(NX, NY)

C       BY C.J. DALY

        COMMON /SPL/ F(50,50), DFDX(50,50), DFDY(50,50), DFDXY(50,50),
       1 X(50), Y(50), DELX(50), DELY(50)

        DIMENSION TRIXU(50), TRIXD(50), TRIXL(50)
```

```
      DIMENSION TRIXU(50), TRIXD(50), TRIXL(50)
      DIMENSION RHS(50), DGTX(50), DGTY(50)

C     Compute coefficients for the spline.  Note that Gaussian
C     elimination is carried out for the tridiagonal matrices
C     before proceeding to Steps One through Four.

      NXM1 = NX - 1
      NYM1 = NY - 1
      NXM2 = NX - 2
      NYM2 = NY - 2
      NXM3 = NX - 3
      NYM3 = NY - 3

C     Compute delta x and delta y.
      DO 1 I = 1,NXM1
1     DELX(I) = X(I+1) - X(I)
      DO 2 I = 1,NYM1
2     DELY(I) = Y(I+1) - Y(I)

C     Fill tridiagonal matrices for all four Steps.
      DO 3 I = 1,NXM2
      TRIXU(I) = DELX(I)
      TRIXD(I) = (DELX(I+1) + DELX(I)) * 2
3     TRIXL(I) = DELX(I+1)
      DO 4 I = 1,NYM2
      TRIYU(I) = DELY(I)
      TRIYD(I) = (DELY(I+1) + DELY(I)) * 2
4     TRIYL(I) = DELY(I+1)

C     Perform pivoting of tridiagonal matrices, downward.
C     Save pivot values for all four Steps.
      DO 5 I = 1,NXM3
      TRIXL(I+1) = -TRIXL(I+1)/TRIXD(I)
5     TRIXD(I+1) = TRIXD(I+1) + TRIXL(I+1)*TRIXU(I)
      DO 6 I = 1,NYM3
      TRIYL(I+1) = -TRIYL(I+1)/TRIYD(I)
6     TRIYD(I+1) = TRIYD(I+1) + TRIYL(I+1)*TRIYU(I)

C     Perform pivoting of tridiagonal matrices, upward.
C     Save pivot values for all four Steps.
      DO 7 I = 1,NXM3
      II = NXM2 - I
7     TRIXU(II) = -TRIXU(II)/TRIXD(II+1)
      DO 8 I = 1,NYM3
      II = NYM2 - I
8     TRIYU(II) = -TRIYU(II)/TRIYD(II+1)

C     Compute and save ratios for efficiency.
      DO 9 I = 1,NYM2
9     DGTX(I) = DELX(I)/DELX(I+1)
      DO 10 I = 1,NYM2
10    DGTY(I) = DELY(I)/DELY(I+1)

C     Perform Step One.
      DO 11 J = 1,NY
      DO 12 I = 1,NXM2
      RHS(I) = 3.0 * ((F(I+1,J)-F(I,J))/DGTX(I) +
     1     (F(I+2,J)-F(I+1,J))*DGTX(I))
12    CONTINUE
      RHS(NXM2) = RHS(NXM2) - TRIXU(NXM2)*DFDX(NX,J)
      RHS(1) = RHS(1) - TRIXL(1)*DFDX(1,J)
      DO 13 I = 1,NXM3
```

```
13      RHS(I+1) = RHS(I+1) + TRIXL(I+1)*RHS(I)
        DO 14 I = 1,NXM3
        II = NXM2 - I
14      RHS(II) = RHS(II) + RHS(II+1)*TRIXU(II)
        DO 15 I = 2,NXM1
15      DFDX(I,J) = RHS(I-1)/TRIXD(I-1)
11      CONTINUE

C       Perform Step Two.
        DO 16 I = 1,NX
        DO 17 J = 1,NYM2
        RHS(J) = 3.0 * ((F(I,J+1)-F(I,J))/DQTY(J) +
     1                  (F(I,J+2)-F(I,J+1))*DQTY(J))
17      CONTINUE
        RHS(1) = RHS(1) - TRIYL(1)*DFDY(I,1)
        RHS(NYM2) = RHS(NYM2) - TRIYU(NYM2)*DFDY(I,NY)
        DO 18 J = 1,NYM3
18      RHS(J+1) = RHS(J+1) + RHS(J)*TRIYL(J+1)
        DO 19 J = 1,NYM3
        JJ = NYM2 - J
19      RHS(JJ) = RHS(JJ) + RHS(JJ+1)*TRIYU(JJ)
        DO 20 J = 2,NYM1
20      DFDY(I,J) = RHS(J-1) / TRIYD(J-1)
16      CONTINUE

C       Perform Step Three.
        DO 26 INDEX = 1,2
        IF (INDEX.EQ.1) I=1
        IF (INDEX.EQ.2) I=NX
        DO 21 J = 1,NYM2
21      RHS(J) = 3.0 * ((DFDX(I,J+1)-DFDX(I,J))/DQTY(J) +
     1          (DFDX(I,J+2)-DFDX(I,J+1))*DQTY(J))
        RHS(1) = RHS(1) - TRIYL(1)*DFDXY(I,1)
        RHS(NYM2) = RHS(NYM2) - TRIYU(NYM2)*DFDXY(I,NY)
        DO 22 J = 1,NYM3
22      RHS(J+1) = RHS(J+1) + TRIYL(J+1)*RHS(J)
        DO 23 J = 1,NYM3
        JJ = NYM2 - J
23      RHS(JJ) = RHS(JJ) + RHS(JJ+1)*TRIYU(JJ)
        DO 24 J = 2,NYM1
24      DFDXY(I,J) = RHS(J-1)/TRIYD(J-1)
26      CONTINUE

C       Perform Step Four.
        DO 28 J = 1,NY
        DO 29 I = 1,NXM2
29      RHS(I) = 3.0 * ((DFDY(I+1,J)-DFDY(I,J))/DQTX(I) +
     1          (DFDY(I+2,J)-DFDY(I+1,J))*DQTX(I))
        RHS(1) = RHS(1) - TRIXL(1)*DFDXY(1,J)
        RHS(NXM2) = RHS(NXM2) - TRIXU(NXM2)*DFDXY(NX,J)
        DO 30 I = 1,NXM3
30      RHS(I+1) = RHS(I+1) + TRIXL(I+1)*RHS(I)
        DO 31 I = 1,NXM3
        II = NXM2 - I
31      RHS(II) = RHS(II) + TRIXU(II)*RHS(II+1)
        DO 32 I = 2,NXM1
32      DFDXY(I,J) = RHS(I-1)/TRIXD(I-1)
28      CONTINUE

        RETURN

        END
```

```
C       Spline interpolating function package

C       *** SPLOAD ***

        SUBROUTINE SPLOAD (INFILE,K)
C       Reads spline data into one of three (K = 1,2, or 3)
C       locations in COMMON.

        COMMON/SPL/ NX(3),NY(3),        /* number of x,y indices
       1            F(50,50,3),         /* function values at interstices
       1            DFDX(50,50,3),      /* df/dx at interstices
       1            DFDY(50,50,3),      /* df/dy at same
       1            DFDXY(50,50,3),     /* df/dxdy at same
       1            X(50,3),            /* x values of vertical grid lines
       1            Y(50,3),            /* y values of horiz. grid lines
       1            DELX(50,3),         /* delta x along grid
       1            DELY(50,3)          /* delta y along grid


        INTEGER INFILE                  /* data file for  spline function


        REWIND INFILE

C       read data file into common
        IF(K.GT.3.OR.K.LT.1)K=1

        READ (INFILE) NX(K),NY(K)
        KX=NX(K)
        KY=NY(K)
        KXM1=KX-1
        KYM1=KY-1

        READ (INFILE) ((F(I,J,K),I=1,KX),J=1,KY)
        READ (INFILE) ((DFDX(I,J,K),I=1,KX),J=1,KY)
        READ (INFILE) ((DFDY(I,J,K),I=1,KX),J=1,KY)
        READ (INFILE) ((DFDXY(I,J,K),I=1,KX),J=1,KY)
        READ (INFILE) (X(I,K),I=1,KX)
        READ (INFILE) (Y(I,K),I=1,KY)
        READ (INFILE) (DELX(I,K),I=1,KXM1)
        READ (INFILE) (DELY(I,K),I=1,KYM1)

        REWIND INFILE

999     RETURN
        END


C       *** SPLINE ***


        FUNCTION SPLINE (X1,Y1,F)       /* Returns Real

C       Function spline(x,y) returns spline interpolating
C       function value at (x,y).

        COMMON /SPL/ NX(3),NY(3),       /* number of x,y indices
       1            F(50,50,3),         /* function values at interstices
       1            DFDX(50,50,3),      /* df/dx at interstices
       1            DFDY(50,50,3),      /* df/dy at same
       1            DFDXY(50,50,3),     /* df/dxdy at same
       1            X(50,3),            /* x values of vertical grid lines
```

```
     1              Y(50,3),          /* y values of horiz. grid lines
     1              DELX(50,3),       /* delta x along grid
     1              DELY(50,3)        /* delta y along grid

      REAL X1,Y1                     /* interpolating function arguments

      INTEGER I,J                    /* subrectangle indices


C     Define interpolating functions

      CX(X1,I,K) = (((X1-X(I+1,K))**2)/DELX(I,K)**2)+
     1             (2*((X1-X(I,K))*(X1-X(I+1,K))**2)/(DELX(I,K)**3))

      CX1(X1,I,K) = (((X1-X(I,K))**2)/DELX(I,K)**2)-
     1              (2*((X1-X(I+1,K))*(X1-X(I,K))**2)/(DELX(I,K)**3))

      CY(Y1,I,K) = (((Y1-Y(I+1,K))**2)/DELY(I,K)**2)+
     1             (2*((Y1-Y(I,K))*(Y1-Y(I+1,K))**2)/(DELY(I,K)**3))

      CY1(Y1,I,K) = (((Y1-Y(I,K))**2)/DELY(I,K)**2)-
     1              (2*((Y1-Y(I+1,K))*(Y1-Y(I,K))**2)/(DELY(I,K)**3))

      DX(X1,I,K) = (X1-X(I,K))*(X1-X(I+1,K))**2/DELX(I,K)**2
      DX1(X1,I,K) = (X1-X(I+1,K))*(X1-X(I,K))**2/DELX(I,K)**2
      DY(Y1,I,K) = (Y1-Y(I,K))*(Y1-Y(I+1,K))**2/DELY(I,K)**2
      DY1(Y1,I,K) = (Y1-Y(I+1,K))*(Y1-Y(I,K))**2/DELY(I,K)**2

C     Make sure the point is on the map
      IF(K.GT.3.OR.K.LT.1)K=1

      KX=NX(K)
      KY=NY(K)
      IF (X1.LT.X(1,K).OR.X1.GT.X(KX,K)) GOTO 777
      IF (Y1.LT.Y(1,K).OR.Y1.GT.Y(KY,K)) GOTO 777

C     Determine I and J

      NXM1 = KX - 1
      NYM1 = KY - 1

      DO 10 I = 1,NXM1
         IF (X1.LE.X(I+1,K)) GO TO 15
C     exit with X(I,K) <= X1 <= X(I+1,K)
10    CONTINUE

15    DO 20 J=1,NYM1
         IF (Y1.LE.Y(J+1,K)) GOTO 30
C     exit with Y(J,K) <= Y1 <= Y(J+1,K)
20    CONTINUE

C     Calculate spline interpolate

30    SPLINE = F(I,J,K)*CX(X1,I,K)*CY(Y1,J,K) +
     1         F(I,J+1,K)*CX(X1,I,K)*CY1(Y1,J,K) +
     1         F(I+1,J,K)*CX1(X1,I,K)*CY(Y1,J,K) +
     1         F(I+1,J+1,K)*CX1(X1,I,K)*CY1(Y1,J,K) +
     1         DFDX(I,J,K)*DX(X1,I,K)*CY(Y1,J,K) +
     1         DFDX(I,J+1,K)*DX(X1,I,K)*CY1(Y1,J,K) +
     1         DFDX(I+1,J,K)*DX1(X1,I,K)*CY(Y1,J,K) +
     1         DFDX(I+1,J+1,K)*DX1(X1,I,K)*CY1(Y1,J,K) +
     1         DFDY(I,J,K)*CX(X1,I,K)*DY(Y1,J,K) +
```

```
     1        DFDY(I,J+1,K)*CX(X1,I,K)*DY1(Y1,J,K) +
     1        DFDY(I+1,J,K)*CX1(X1,I,K)*DY(Y1,J,K) +
     1        DFDY(I+1,J+1,K)*CX1(X1,I,K)*DY1(Y1,J,K) +
     1        DFDXY(I,J,K)*DX(X1,I,K)*DY(Y1,J,K) +
     1        DFDXY(I,J+1,K)*DX(X1,I,K)*DY1(Y1,J,K) +
     1        DFDXY(I+1,J,K)*DX1(X1,I,K)*DY(Y1,J,K) +
     1        DFDXY(I+1,J+1,K)*DX1(X1,I,K)*DY1(Y1,J,K)

999    RETURN

C      ERROR: here if point out of bounds

777    WRITE (1,1010) X1,Y1
1010   FORMAT('SPLINE ARGUMENTS OUT OF BOUNDS: ',1P2E16.6)
       STOP

       END



C      *** DS/DX ***

       FUNCTION DSDX (X1,Y1,K)          /* returns real

C      Returns the partial w.r.t X of the spline interpolating function

       COMMON /SPL/ NX(3),NY(3),        /* number of x,y indices
     1              F(50,50,3),         /* function values at interstices
     1              DFDX(50,50,3),      /* df/dx at interstices
     1              DFDY(50,50,3),      /* df/dy at same
     1              DFDXY(50,50,3),     /* df/dxdy at same
     1              X(50,3),            /* x values of vertical grid lines
     1              Y(50,3),            /* y values of horiz. grid lines
     1              DELX(50,3),         /* delta x along grid
     1              DELY(50,3)          /* delta y along grid

       REAL X1,Y1                       /* function arguments

       INTEGER I,J                      /* subrectangle indices


C      Define interpolating sub-functions ( p for partials)

       PCX(X1,I,K) = 2*(X1-X(I+1,K))/DELX(I,K)**2 +
     1       2*(X1-X(I+1,K))**2/DELX(I,K)**3 +
     1       4*(X1-X(I,K))*(X1-X(I+1,K))/DELX(I,K)**3

       PCX1(X1,I,K) = 2*(X1-X(I,K))/DELX(I,K)**2 -
     1       2*(X1-X(I,K))**2/DELX(I,K)**3 -
     1       4*(X1-X(I,K))*(X1-X(I+1,K))/DELX(I,K)**3

       PDX(X1,I,K) = 2*(X1-X(I,K))*(X1-X(I+1,K))/DELX(I,K)**2 +
     1       (X1-X(I+1,K))**2/DELX(I,K)**2
       PDX1(X1,I,K) = 2*(X1-X(I,K))*(X1-X(I+1,K))/DELX(I,K)**2 +
     1       (X1-X(I,K))**2/DELY(I,K)**2


       CY(Y1,I,K) = (((Y1-Y(I+1,K))**2)/DELY(I,K)**2)+
     1            (2*((Y1-Y(I,K))*(Y1-Y(I+1,K))**2)/(DELY(I,K)**3))
```

```
      CY1(Y1,I,K) = (((Y1-Y(I,K))**2)/DELY(I,K)**2)-
     1            (2*((Y1-Y(I+1,K))*(Y1-Y(I,K))**2)/(DELY(I,K)**3))

      DY(Y1,I,K) = (Y1-Y(I,K))*(Y1-Y(I+1,K))**2/DELY(I,K)**2
      DY1(Y1,I,K) = (Y1-Y(I+1,K))*(Y1-Y(I,K))**2/DELY(I,K)**2

C     Make sure the point is on the map
      IF(K.GT.3.OR.K.LT.1)K=1

      KX=NX(K)
      KY=NY(K)
      IF (X1.LT.X(1,K).OR.X1.GT.X(KX,K)) GOTO 777
      IF (Y1.LT.Y(1,K).OR.Y1.GT.Y(KY,K)) GOTO 777

C     Determine I and J

      NXM1 = KX - 1
      NYM1 = KY - 1

      DO 10 I = 1,NXM1
         IF (X1.LE.X(I+1,K)) GOTO 15
C     exit with X(I,K) <= X1 <= X(I+1,K)
10    CONTINUE

15    DO 20 J=1,NYM1
         IF (Y1.LE.Y(J+1,K)) GOTO 30
C     exit with Y(J,K) <= Y1 <= Y(J+1,K)
20    CONTINUE

C     Calculate ds/dx interpolate

30    DSDX = F(I,J,K)*PCX(X1,I,K)*CY(Y1,J,K) +
     1       F(I,J+1,K)*PCX(X1,I,K)*CY1(Y1,J,K) +
     1       F(I+1,J,K)*PCX1(X1,I,K)*CY(Y1,J,K) +
     1       F(I+1,J+1,K)*PCX1(X1,I,K)*CY1(Y1,J,K) +
     1       DFDX(I,J,K)*PDX(X1,I,K)*CY(Y1,J,K) +
     1       DFDX(I,J+1,K)*PDX(X1,I,K)*CY1(Y1,J,K) +
     1       DFDX(I+1,J,K)*PDX1(X1,I,K)*CY(Y1,J,K) +
     1       DFDX(I+1,J+1,K)*PDX1(X1,I,K)*CY1(Y1,J,K) +
     1       DFDY(I,J,K)*PCX(X1,I,K)*DY(Y1,J,K) +
     1       DFDY(I,J+1,K)*PCX(X1,I,K)*DY1(Y1,J,K) +
     1       DFDY(I+1,J,K)*PCX1(X1,I,K)*DY(Y1,J,K) +
     1       DFDY(I+1,J+1,K)*PCX1(X1,I,K)*DY1(Y1,J,K) +
     1       DFDXY(I,J,K)*PDX(X1,I,K)*DY(Y1,J,K) +
     1       DFDXY(I,J+1,K)*PDX(X1,I,K)*DY1(Y1,J,K) +
     1       DFDXY(I+1,J,K)*PDX1(X1,I,K)*DY(Y1,J,K) +
     1       DFDXY(I+1,J+1,K)*PDX1(X1,I,K)*DY1(Y1,J,K)

999   RETURN

C     ERROR: here if point out of bounds

777   WRITE (1,1010) X1,Y1
1010  FORMAT('DSDX ARGUMENTS OUT OF BOUNDS: ',1P2E16.6)
      STOP

      END



C     *** DS/DY ***
```

```
      FUNCTION DSDY (X1,Y1,K)          /* returns real

C     Returns the partial w.r.t. Y of the spline interpolating function

      COMMON /SPL/ NX(3),NY(3),        /* number of x,y indices
     1             F(50,50,3),         /* function values at interstices
     1             DFDX(50,50,3),      /* df/dx at interstices
     1             DFDY(50,50,3),      /* df/dy at same
     1             DFDXY(50,50,3),     /* df/dxdy at same
     1             X(50,3),            /* x values of vertical grid lines
     1             Y(50,3),            /* y values of horiz. grid lines
     1             DELX(50,3),         /* delta x along grid
     1             DELY(50,3)          /* delta y along grid

      REAL X1,Y1                      /* function arguments

      INTEGER I,J                     /* subrectangle indices


C     Define interpolating sub-functions (p for partials)

      PCY(Y1,I,K) = 2*(Y1-Y(I+1,K))/DELY(I,K)**2 +
     1      2*(Y1-Y(I+1,K))**2/DELY(I,K)**3 +
     1      4*(Y1-Y(I,K))*(Y1-Y(I+1,K))/DELY(I,K)**3

      PCY1(Y1,I,K) = 2*(Y1-Y(I,K))/DELY(I,K)**2 -
     1      2*(Y1-Y(I,K))**2/DELY(I,K)**3 -
     1      4*(Y1-Y(I,K))*(Y1-Y(I+1,K))/DELY(I,K)**3

      PCY(Y1,I,K) = 2*(Y1-Y(I,K))*(Y1-Y(I+1,K))/DELY(I,K)**2 +
     1      (Y1-Y(I+1,K))**2/DELY(I,K)**2
      PCY1(Y1,I,K) = 2*(Y1-Y(I,K))*(Y1-Y(I+1,K))/DELY(I,K)**2 +
     1      (Y1-Y(I,K))**2/DELY(I,K)**2


      CX(X1,I,K) = (((X1-X(I+1,K))**2)/DELX(I,K)**2)+
     1            (2*((X1-X(I,K))*(X1-X(I+1,K))**2)/(DELX(I,K)**3))

      CX1(X1,I,K) = (((X1-X(I,K))**2)/DELX(I,K)**2)-
     1            (2*((X1-X(I+1,K))*(X1-X(I,K))**2)/(DELX(I,K)**3))

      DX(X1,I,K) = (X1-X(I,K))*(X1-X(I+1,K))**2/DELX(I,K)**2
      DX1(X1,I,K) = (X1-X(I+1,K))*(X1-X(I,K))**2/DELX(I,K)**2

C     Make sure the point is on the map
      IF(K.GT.3.OR.K.LT.1)K=1

      KX=NX(K)
      KY=NY(K)
      IF (X1.LT.X(1,K).OR.X1.GT.X(KX,K)) GOTO 777
      IF (Y1.LT.Y(1,K).OR.Y1.GT.Y(KY,K)) GOTO 777

C     Determine I and J

      NXM1 = KX - 1
      NYM1 = KY - 1

      DO 10 I = 1,NXM1
         IF (X1.LE.X(I+1,K)) GOTO 15
C     exit with X(I,K) <= X1 <= X(I+1,K)
10    CONTINUE
```

```
15      DO 20 J=1,NYM1
            IF (Y1.LE.Y(J+1,K)) GOTO 30
C       exit with Y(J,K) <= Y1 <= Y(J+1,K)
20      CONTINUE

C       Calculate ds/dy interpolate

30      DSDY = F(I,J,K)*CX(X1,I,K)*PCY(Y1,J,K) +
     1         F(I,J+1,K)*CX(X1,I,K)*PCY1(Y1,J,K) +
     1         F(I+1,J,K)*CX1(X1,I,K)*PCY(Y1,J,K) +
     1         F(I+1,J+1,K)*CX1(X1,I,K)*PCY1(Y1,J,K) +
     1         DFDX(I,J,K)*DX(X1,I,K)*PCY(Y1,J,K) +
     1         DFDX(I,J+1,K)*DX(X1,I,K)*PCY1(Y1,J,K) +
     1         DFDX(I+1,J,K)*DX1(X1,I,K)*PCY(Y1,J,K) +
     1         DFDX(I+1,J+1,K)*DX1(X1,I,K)*PCY1(Y1,J,K) +
     1         DFDY(I,J,K)*CX(X1,I,K)*PDY(Y1,J,K) +
     1         DFDY(I,J+1,K)*CX(X1,I,K)*PDY1(Y1,J,K) +
     1         DFDY(I+1,J,K)*CX1(X1,I,K)*PDY(Y1,J,K) +
     1         DFDY(I+1,J+1,K)*CX1(X1,I,K)*PDY1(Y1,J,K) +
     1         DFDXY(I,J,K)*DX(X1,I,K)*PDY(Y1,J,K) +
     1         DFDXY(I,J+1,K)*DX(X1,I,K)*PDY1(Y1,J,K) +
     1         DFDXY(I+1,J,K)*DX1(X1,I,K)*PDY(Y1,J,K) +
     1         DFDXY(I+1,J+1,K)*DX1(X1,I,K)*PDY1(Y1,J,K)

999     RETURN

C       ERROR: here if point out of bounds

777     WRITE (1,1010) X1,Y1
1010    FORMAT('DSDY ARGUMENTS OUT OF BOUNDS: ',1P2E16.6)
        STOP

        END
```

```
C       ***** PROGRAM *****

C       Sample program for Spline Interpolating Package
C       It returns function value and gradient at points chosen by user

C       MNEMONIC CODES FOR CREEL FILE CALLS
        INTEGER*2 J$READ,J$WRIT,J$RDWR,J$CLOS,J$FRWD,J$RWND

        PARAMETER
       1     J$READ = :1,       /* READ
       1     J$WRIT = :2,       /* WRITE
       1     J$RDWR = :3,       /* READ AND WRITE
       1     J$CLOS = :4,       /* CLOSE FILE
       1     J$DELE = :5,       /* DELETE FILE
       1     J$FRWD = :6,       /* MOVE FORWARD ONE FILE (MAG TAPE)
       1     J$RWND = :7        /* REWIND

        INTEGER INFILE                  /* fortran unit number of data file

        REAL X,Y,                       /* point we pass to spline
       1     HEIGHT,                    /* interpolated function value
       1     XSLOPE,                    /* ds/dx at x,y
       1     YSLOPE                     /* ds/dy at x,y

C       open the data file made by CREATE
        INFILE=5                        /* fortran unit # of data file

C       Get filename from user.
        CALL GFILE$(INFILE,J$READ,'SPLINE FILE',11)

C       load the spline data file
C       Data is arbitrarily loaded into location K=1 in COMMON.
        CALL SPLOAD(INFILE,1)

C       close data file
        CALL FILES$(INFILE,J$CLOS)

C       get point from user
10      CALL TNOUA('point to compute :',18)
        READ (1,*) X,Y

C       call interpolating functions

        HEIGHT = SPLINE(X,Y,1)          /* find function value at x,y
        XSLOPE = DSDX(X,Y,1)            /* find ds/dx
        YSLOPE = DSDY(X,Y,1)            /* find ds/dy

        PRINT 100,HEIGHT,XSLOPE,YSLOPE
100     FORMAT(1X,F10.2,2F10.5)

C       print answers on terminal

        GOTO 10         /* loop until user gets bored
        END
```

```
C       Main program for 2-D particle movement in a flow field.
C
C       MNEMONIC CODES FOR CRREL FILE CALLS
        INTEGER*2 J$READ,J$WRIT,J$RDWR,J$CLOS,J$FRWD,J$RWND

        PARAMETER
       1     J$READ = :1,      /* READ
       1     J$WRIT = :2,      /* WRITE
       1     J$RDWR = :3,      /* READ AND WRITE
       1     J$CLOS = :4,      /* CLOSE FILE
       1     J$DELE = :5,      /* DELETE FILE
       1     J$FRWD = :6,      /* MOVE FORWARD ONE FILE (MAG TAPE)
       1     J$RWND = :7       /* REWIND

        INTEGER SPFILE,OUTPUT

C       Points along a single flow line (XP,YP)
        DIMENSION XP(1000),YP(1000)

C       Boundaries of the flow region
        COMMON/BOUND/X1,XN,Y1,YM
        INFILE=5
        OUTPUT=6
        SPFILE=7
        PRINT 100
100     FORMAT(//1X,4HGIVE)

C       Read hydraulic head spline data file.
        CALL GFILE$(SPFILE,J$READ,'H SPLINE INFILE',15)
        CALL SPLOAD(SPFILE,1)
        CALL FILES$(SPFILE,J$CLOS)
        PRINT 100

C       Read hydraulic conductivity spline data file.
        CALL GFILE$(SPFILE,J$READ,'K SPLINE INFILE',15)
        CALL SPLOAD(SPFILE,2)
        CALL FILES$(SPFILE,J$CLOS)
        PRINT 100

C       Read effective porosity spline data file.
        CALL GFILE$(SPFILE,J$READ,'PSI SPLINE INFILE',17)
        CALL SPLOAD(SPFILE,3)
        CALL FILES$(SPFILE,J$CLOS)
        PRINT 100

C       Get filename for file containing delta t, scale, flow
C       region boundaries, and initial points of flow lines.
        CALL GFILE$(INFILE,J$READ,'POINT INPUT FILE',16)
        PRINT 100

C       Get filename for output of flow line points.
        CALL GFILE$(OUTPUT,J$WRIT,'STREAMLINE OUTPUT FILE',22)

        READ(INFILE,*)DELT
        READ(INFILE,*)XYSC
        READ(INFILE,*)X1,XN,Y1,YM

C       Scale input flow boundary data.
        X1=X1*XYSC
        XN=XN*XYSC
        Y1=Y1*XYSC
        YM=YM*XYSC
```

```
      WRITE(OUTPUT,1000)DELT
      WRITE(OUTPUT,1000)X1,XN,Y1,YM
1000  FORMAT(1X,4F10.2)

C     Read (X,Y), the beginning point of a flow line.
C     Scale input values.
2     READ(INFILE,*,END=1)X,Y
      X=X*XYSC
      Y=Y*XYSC

C     Compute flow line coordinates.
      CALL STREAM(OUTPUT,X,Y,DELT)

C     Go back to read initial point for another flow line.
      GO TO 2
1     CALL FILES$(INFILE,J$CLOS)
      CALL FILES$(OUTPUT,J$CLOS)
      STOP
      END
```

```
C       Average linear velocity calculation routine.

        SUBROUTINE FG(T,X,Y,F,G,IOUT)
        REAL K

C       K = hydraulic conductivity (isotropic).  COMMON location 2.
C       H = hydraulic head.  Spline data in COMMON location 1.
C       PSI = effective porosity.  COMMON location 3.

C       The flow region is  X1 < X < XN  and  Y1 < Y < YM.
        COMMON/BOUND/X1,XN,Y1,YM

C       Check to see if X and Y are within the flow region.
        IF(X.LT.X1.OR.X.GT.XN)GO TO 1
        IF(Y.LT.Y1.OR.Y.GT.YM)GO TO 1

C       Use spline files to compute the following.
C       (DHDX,DHDY) is the hydraulic gradient.
        DHDX=DSDX(X,Y,1)
        DHDY=DSDY(X,Y,1)
        K=SPLINE(X,Y,2)
        PSI=SPLINE(X,Y,3)

C       Darcy's law for average linear velocities.
C       F = x component of the average linear velocity.
C       G = y component of the average linear velocity.
        F=-(K*DHDX)/PSI
        G=-(K*DHDY)/PSI
        RETURN

C       IOUT = 1 signals the crossing of a flow boundary.
1       IOUT=1
        RETURN
        END
```

```
C       *****  METHOD 1 ALGORITHM  *****
        SUBROUTINE STREAM(OUTPUT,X,Y,DELT)
C
C   SOLVES F(X,Y,T) DC/DX  +  G(X,Y,T) DC/DY  +  DC/DT = 0
C   BY THE METHOD OF CHARACTERISTICS USING FOURTH ORDER
C   RUNGE-KUTTA AND PREDICTOR-CORRECTOR ALGORITHMS.
C
        INTEGER OUTPUT
        DIMENSION XNPM(3),YNPM(3),XNT(1001),YNT(1001)
        IOUT=0
C       WHEN IOUT IS RETURNED AS 1, THE CHARACTERISTIC WILL BE CROSSING
C       A BOUNDARY OF THE FLOW REGION.
        NPOINT=1
        DELT2=DELT/2.
        DELT3=DELT/3.
        DELT6=DELT/6.
        DELT43=4.*DELT3
        DELT83=8.*DELT3
C       INITIALIZE AT THE POINT OF ORIGIN OF THE CHARACTERISTIC
        XNT(1)=X
        YNT(1)=Y
        XN=X
        YN=Y
        TN=0.0
        CALL FG(TN,XN,YN,F1,G1,IOUT)
        IF(IOUT.NE.0)GO TO 1

C       PERFORM RUNGE-KUTTA FOR THE FIRST THREE POINTS TO GET STARTED
C       CLASSICAL (4,4) FORMULATION.
        DO 2 IT=1,3
        TN1=TN+DELT
        XVAL=XN+F1*DELT2
        YVAL=YN+G1*DELT2
        TVAL=TN+DELT2
        CALL FG(TVAL,XVAL,YVAL,F2,G2,IOUT)
        IF(IOUT.NE.0)GO TO 1

        XVAL=XN+F2*DELT2
        YVAL=YN+G2*DELT2
        CALL FG(TVAL,XVAL,YVAL,F3,G3,IOUT)
        IF(IOUT.NE.0)GO TO 1

        XVAL=XN+F3*DELT
        YVAL=YN+G3*DELT
        CALL FG(TN1,XVAL,YVAL,F4,G4,IOUT)
        IF(IOUT.NE.0)GO TO 1

        XN1=XN+DELT6*(F1+F4)+DELT3*(F2+F3)
        YN1=YN+DELT6*(G1+G4)+DELT3*(G2+G3)
        XNT(IT+1)=XN1
        YNT(IT+1)=YN1
        NPOINT=IT+1
        XN=XN1
        YN=YN1
        TN=TN1
        CALL FG(TN,XN,YN,F1,G1,IOUT)
        IF(IOUT.NE.0)GO TO 1

        XNPM(IT)=F1
2       YNPM(IT)=G1

C       USE PREDICTOR-CORRECTOR METHOD FOR THE REST OF THE CHARACTERISTIC
C       STANDARD FOURTH ORDER MILNE FORMULATION.
```

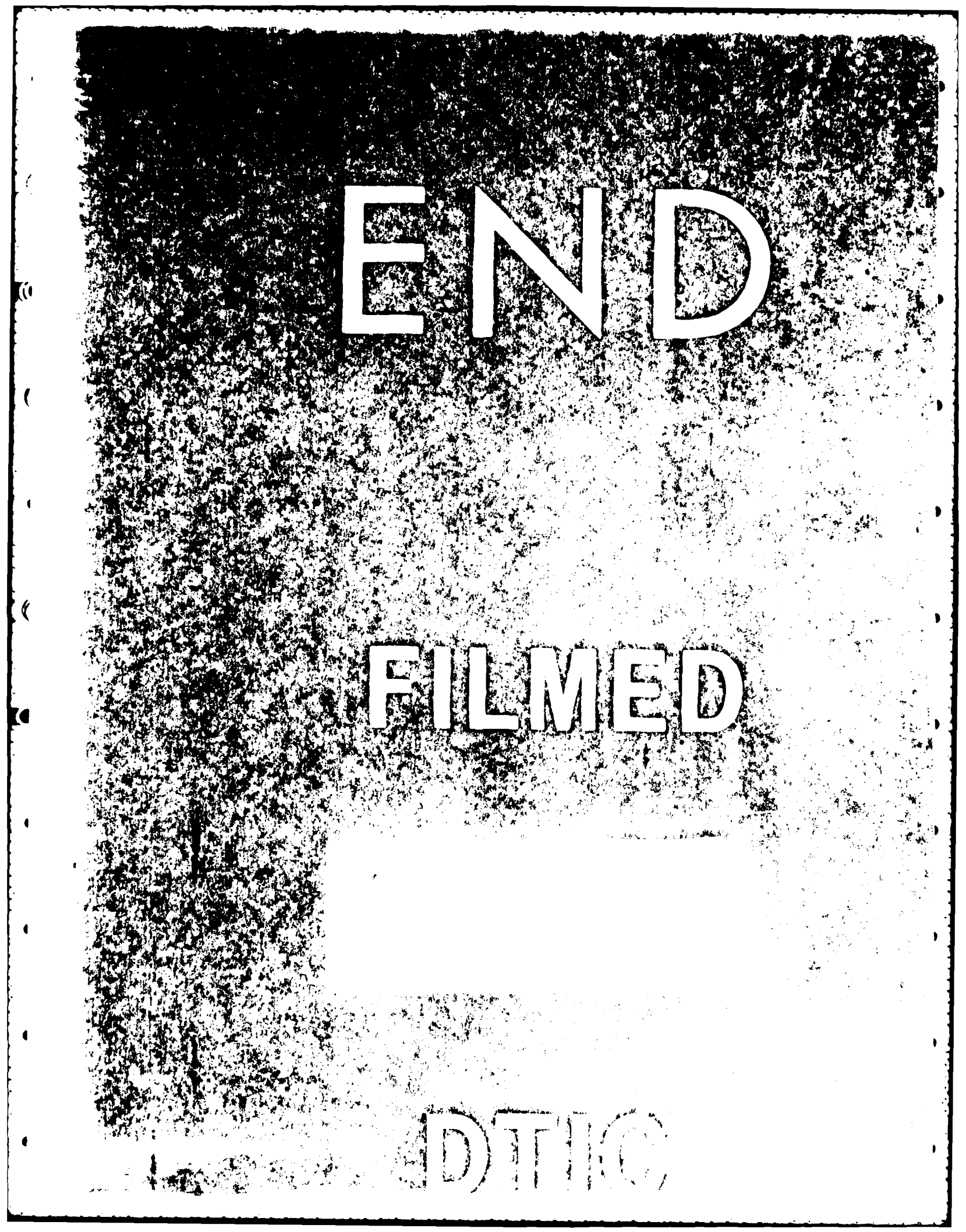
```
      DO 3 IT=4,1000
      TN1=TN+DELT
      XNP1=XNT(IT-3)+DELT83*(XNPM(3)+XNPM(1))-DELT43*XNPM(2)
      YNP1=YNT(IT-3)+DELT83*(YNPM(3)+YNPM(1))-DELT43*YNPM(2)
      CALL FG(TN1,XNP1,YNP1,XPRM,YPRM,IOUT)
      IF(IOUT.NE.0)GO TO 1

      XN1=XNT(IT-1)+DELT3*(XPRM+XNPM(2))+DELT43*XNPM(3)
      YN1=YNT(IT-1)+DELT3*(YPRM+YNPM(2))+DELT43*YNPM(3)

      XNT(IT+1)=XN1
      YNT(IT+1)=YN1
      NPOINT=IT+1
      XNPM(1)=XNPM(2)
      XNPM(2)=XNPM(3)
      YNPM(1)=YNPM(2)
      YNPM(2)=YNPM(3)
      CALL FG(TN1,XN1,YN1,XNPM(3),YNPM(3),IOUT)
      IF(IOUT.NE.0)GO TO 1

3     CONTINUE

C     THERE WILL BE NPOINTS ON THE CHARACTERISTIC LINE
1     WRITE(OUTPUT,1000)NPOINT
1000  FORMAT(1X,I5)
      WRITE(OUTPUT,1001)((XNT(I),YNT(I)),I=1,NPOINT)
1001  FORMAT(5(5X,2F10.2))
      RETURN
      END
```

```
C       *****  METHOD 2 ALGORITHM  *****

        SUBROUTINE STREAM(OUTPUT,X,Y,DELT)
C
C   SOLVES F(X,Y,T) DC/DX  +  G(X,Y,T) DC/DY  +  DC/DT = 0
C   BY THE METHOD OF CHARACTERISTICS USING AN EMBEDDED (5,6)
C   RUNGE-KUTTA FORM AND ACCURACY CHECK.
C   INVOLVES VARIABLE TIME STEP FEATURE.
C
        INTEGER OUTPUT
        DIMENSION XNT(1001),YNT(1001)
        PARAMETER TOLA=0.001
        PARAMETER TOLB=0.0001
        PARAMETER SMALL=1.E-10
        IOUT=0
C       WHEN IOUT IS RETURNED AS 1, THE CHARACTERISTIC WILL BE CROSSING
C       A BOUNDARY OF THE FLOW REGION.
        NPOINT=1
C       INITIALIZE AT THE POINT OF ORIGIN OF THE CHARACTERISTIC
        XNT(1)=X
        YNT(1)=Y
        XNP=X
        YNP=Y
        TNP=0.0

C       PERFORM RUNGE-KUTTA
        ISTEP=1
        DO 2 IT=1,1000
4       DT=DELT/ISTEP
        DT2=DT/2.
        DT4=DT/4.
        DT5=DT/5.
        DT6=DT/6.
        DT27=DT/27.
        DT336=DT/336.
        DT625=DT/625.
        DT23=2.*DT/3.

        XN=XNP
        YN=YNP
        TN=TNP
        KOUNT=0

C       within each major time step delta T, Runge-Kutta is
C       used at time steps delta t = delta T/ISTEP.
        DO 3 JT=1,ISTEP
        CALL FG(TN,XN,YN,F1,G1,IOUT)
        IF(IOUT.NE.0)GO TO 1

        XVAL=XN+DT2*F1
        YVAL=YN+DT2*G1
        TVAL=TN+DT2
        CALL FG(TVAL,XVAL,YVAL,F2,G2,IOUT)
        IF(IOUT.NE.0)GO TO 1

        XVAL=XN+DT4*(F1+F2)
        YVAL=YN+DT4*(G1+G2)
        TVAL=TN+DT2
        CALL FG(TVAL,XVAL,YVAL,F3,G3,IOUT)
        IF(IOUT.NE.0)GO TO 1

        XVAL=XN+DT*(2.*F3-F2)
        YVAL=YN+DT*(2.*G3-G2)
```

```
      TVAL=TN+DT
      CALL FG(TVAL,XVAL,YVAL,F4,G4,IOUT)
      IF(IOUT.NE.0)GO TO 1

      XVAL=XN+DT27*(7.*F1+16.*F2+F4)
      YVAL=YN+DT27*(7.*G1+16.*G2+G4)
      TVAL=TN+DT23
      CALL FG(TVAL,XVAL,YVAL,F5,G5,IOUT)
      IF(IOUT.NE.0)GO TO 1

      XVAL=XN+DT625*(28.*F1-125.*F2+546.*F3+54.*F4-378.*F5)
      YVAL=YN+DT625*(28.*G1-125.*G2+546.*G3+54.*G4-378.*G5)
      TVAL=TN+DT5
      CALL FG(TVAL,XVAL,YVAL,F6,G6,IOUT)
      IF(IOUT.NE.0)GO TO 1

      XN14=XN+DT6*(F1+4.*F3+F4)
      YN14=YN+DT6*(G1+4.*G3+G4)
      XN16=XN+DT336*(14.*F1+35.*F4+162.*F5+125.*F6)
      YN16=YN+DT336*(14.*G1+35.*G4+162.*G5+125.*G6)
      TN1=TN+DT

C     Begin calculation for accuracy check.
      XA6=ABS(XN16)
      YA6=ABS(YN16)
      IF(XA6.LT.SMALL)XA6=SMALL
      IF(YA6.LT.SMALL)YA6=SMALL
      CHECKX=ABS(XN16-XN14)/XA6
      CHECKY=ABS(YN16-YN14)/YA6
      IF(CHECKX.LT.TOLA.AND.CHECKY.LT.TOLA)GO TO 5

C     Here if accuracy is insufficient.  Go back and start
C     from the beginning of the major time step with a new
C     delta t equal to 1/2 the current delta t.
      ISTEP=ISTEP*2
      IF(ISTEP.LT.32)GO TO 4
      WRITE(1,1002)
1002  FORMAT(/1X,39HMAX TIME STEP TOO LARGE, ABORT)
      STOP

5     IF(CHECKX.LT.TOLB.AND.CHECKY.LT.TOLB)KOUNT=KOUNT+1
      XN=XN16
      YN=YN16
3     TN=TN1

      XNT(IT+1)=XN16
      YNT(IT+1)=YN16
      NPOINT=IT+1

C     If accuracy criteria was consistently surpassed throughout
C     the major time step, double delta t.
      IF(KOUNT.EQ.ISTEP)ISTEP=ISTEP/2
      IF(ISTEP.LT.1)ISTEP=1
      XNP=XN16
      YNP=YN16
      TNP=TN1
2     CONTINUE

C     THERE WILL BE NPOINTS ON THE CHARACTERISTIC LINE
1     WRITE(OUTPUT,1000)NPOINT
1000  FORMAT(1X,I5)
      WRITE(OUTPUT,1001)((XNT(I),YNT(I)),I=1,NPOINT)
1001  FORMAT(5(5X,2F10.2))

RETURN
END
```

```
C      *****  SFLOT PLOTTING ROUTINE  *****
C      ROUTINE FOR PLOTTING STREAMLINES.  CREATES OVERLAYS WHEN
C      USED WITH APPROPRIATE SCALE FACTORS.
C
C      MNEMONIC CODES FOR CREL FILE CALLS
       INTEGER*2 J$READ,J$WRIT,J$RDWR,J$CLOS,J$FRWD,J$RWND

       PARAMETER
      1     J$READ = :1,      /* READ
      1     J$WRIT = :2,      /* WRITE
      1     J$RDWR = :3,      /* READ AND WRITE
      1     J$CLOS = :4,      /* CLOSE FILE
      1     J$DELF = :5,      /* DELETE FILE
      1     J$FRWD = :6,      /* MOVE FORWARD ONE FILE (MAG TAPE)
      1     J$RWND = :7       /* REWIND
C
       DIMENSION X(1003),Y(1003),LABEL(10)
C
       INFILE=5
       CALL GFILE$(INFILE,J$READ,'GIVE STREAMLINE INPUT FILE',26)
       READ(INFILE,1000)DELT
       READ(INFILE,1000)X1,XN,Y1,YM
1000   FORMAT(1X,4F10.2)
C
C      INPUT SCALE, 1 INCH OF PAPER EQUALS (SCALE) FEET,METERS,MILES,....
C
       PRINT 1001
1001   FORMAT(/1X,11HINPUT SCALE)
       READ(1,*)SCALE
       PRINT 1004
1004   FORMAT(/1X,31HSYMBOL EVERY N TH POINT, GIVE N)
       READ(1,*)INCR
       TINCR=DELT*INCR

C      INITIATE PLOT MODE.
       CALL PLOTS(0)

C      DETERMINE FLOW REGION DIMENSIONS IN INCHES.
       XLONG=(XN-X1)/SCALE
       YLONG=(YM-Y1)/SCALE
C
C      DRAW OUTLINE OF FLOW REGION.
C
       CALL PLOT(0.,0.,-3)
       CALL PLOT(XLONG,0.,2)
       CALL PLOT(XLONG,YLONG,2)
       CALL PLOT(0.,YLONG,2)
       CALL PLOT(0.,0.,2)
C
C      READ IN, AND PLOT THE STREAMLINES ONE AT A TIME.
C
2      READ(INFILE,1002,END=1)NPOINT
1002   FORMAT(1X,I5)
       X(NPOINT+1)=X1
       X(NPOINT+2)=SCALE
       Y(NPOINT+1)=Y1
       Y(NPOINT+2)=SCALE
       READ(INFILE,1003)((X(I),Y(I)),I=1,NPOINT)
1003   FORMAT(5(5X,2F10.2))

C      PERFORM PLOTTING OF THE CURRENT FLOW LINE.
       CALL LINE(X,Y,NPOINT,1,INCR,6)
```

```
      GO TO 2
C
C     PUT REFERENCE MARKS AND ANNOTATION ON THE PLOT.
C
1     ENCODE(18,2000,LABEL)X1,Y1
2000  FORMAT(1H(,F7.1,1H,,F7.1,2H) )
      SIZE=.16
      XLOC=-.5
      YLOC=-.5
      CALL SYMBOL(XLOC,YLOC,SIZE,LABEL,0.,18)
      ENCODE(20,2001,LABEL)TINCR
2001  FORMAT(14HTIME INTERVAL=,F6.1)
      XLOC=XLONG/2.
      CALL SYMBOL(XLOC,YLOC,SIZE,LABEL,0.,20)
      ENCODE(18,2000,LABEL)XN,YM
      XLOC=XLONG-1.75
      YLOC=YLONG+.5
      CALL SYMBOL(XLOC,YLOC,SIZE,LABEL,0.,18)
      CALL FILES$(INFILE,J$CLOS)
      STOP
      END
```


END
FILMED
DTIC