THE UNIVERSITY OF ALBERTA

FINITE ELEMENT SOLUTION OF POLLUTANT CONSERVATION EQUATION AND ITS APPLICATION TO RIVER PLUMES

BY

SANDEEP C. SOLANKI

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF CIVIL ENGINEERING

EDMONTON, ALBERTA

FALL, 1988

ISBN 0-315-45643-4

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: SANDEEP C. SOLANKI

TITILE OF THESIS: FINITE ELEMENT SOLUTION OF THE POLLUTANT CONSERVATION EQUATION AND ITS APPLICATION TO RIVER PLUMES

DEGREE: MASTER OF SCIENCE

YEAR THIS DEGREE GRANTED: FALL 1988

Permanent Address:

827, Woodpark Way S.W.
Calgary, Alberta
Canada, T2W 2T9

Date: June 15, 1988

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled FINITE ELEMENT SOLUTION OF POLLUTANT CONSERVATION EQUATION AND ITS APPLICATION TO RIVER PLUMES submitted by SANDEEP C. SOLANKI in partial fulfilment of the requirements for the degree of MASTER OF SCIENCE in CIVIL ENGINEERING.

SUPERVISOR

DATE: June 2nd, 1988

ABSTRACT

A finite element formulation is used to solve the depth averaged pollutant conservation equation for a river. Two newly developed higher order upwinding elements are tested against standard methods. These new elements performed well in one and two dimensional purely advection tests. Their performance was a great deal better at higher Peclet numbers, in their ability to predict flows skewed to the mesh. The application of these new elements to both ideal and practical situations is restricted due to the large demands they make on computation effort.

The model was used to better understand the variation of eddy diffusivity on the characteristics of plumes. The performance of the model was adequate when it was applied to the solution of a plume in the Grand River, near Kitchener, Ontario. The application to the slug test was not as good, as the physics of the processes involved are not yet understood.

In conclusion the finite element application to iver plume was found to be an excellent tool due to its flexibility in incorporating arbitrary geometries, and its higher accuracy in a general case.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the encouragement and support provided by Dr. N. Rajaratnam.

The assistance provided by Dr. P. Steffler in the development and implementation of the programming and numerical modelling aspects of the thesis is greatly appreciated.

The encouragement given by Dr. P Steffler and Mr. S. Lovell in use of the latest computation hardware available to perform many aspects of this thesis is also acknowledged.

This thesis was financially supported by the Natural Sciences and Engineering Research Council by means of grants to Dr. N. Rajaratnam, and a scholarship from the Government of Alberta.

Table of Contents

Chapter Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

LIST OF SELECTED SYMBOLS

[J] - coordinate transformation matrix

[K] - stiffness matrix

[M] - mass matrix

A and B - coefficients used to fix the boundary condition

C - depth averaged concentration

c - instantaneous concentration

$\overline{C}$ - cross-sectional averaged concentration

$\overline{c}$ - time averaged concentration

C_i - nodal concentration

CUPG - Cubic Upwinding Petrov Galerkin Method

D_{ij} - the tensor of the molecular diffusion

$\overline{D_{ij}}$ - time averaged tensor of the molecular diffusion

ε_i - eddy diffusivity

E_{ij} - sum of dispersion and eddy diffusivity

E_x - eddy diffusivity & dispersion in the x direction

E_z - eddy diffusivity & dispersion in the z direction

ϕ - relative numerical dispersion

f_i - basis functions

Γ - the surface of the domain

g - acceleration due to gravity

γ - coefficients of the Fourier series Solution

h - mean depth

i - complex number $\sqrt{-1}$

k - roughness

K_{ij} - dispersion tensor

λ	-	wavelength of the perturbations
L2	-	L-2 norm
L2'	-	L-2 discrete norm
LBG	-	Linear Bubnov Galerkin Method
QUPG	-	Quadratic Upwinding Petrov Galerkin Method
L_i	-	length to where the slug occupies entire channel
m_i	-	metric coefficients to account for a curvilinear coordinate system
P	-	depth averaged source or sink term
p	-	source or sink term
$\overline{p}$	-	time averaged source or sink term
Pe	-	Peclet number
θ	-	implicit coefficient
q_c	-	cumulative discharge
R	-	hydraulic radius
ρ	-	Courant number
r	-	amplification component of the Fourier Mode
ρ_m	-	density
S_f	-	slope of the specific energy line
U	-	depth averaged velocity in x direction
$\overline{u'c'}$	-	time averaged velocity and concentration fluctuations
$\overline{\overline{u'c'}}$	-	depth and time averaged velocity in the x direction and concentration fluctuations
U_*	-	shear velocity

$\overline{U}$ - cross-sectional averaged velocity in x direction

$\overline{u_D c_D}$ - deviations from the cross-sectional averaged velocity and concentration

$\overline{u_d c_d}$ - deviations from the depth averaged x direction velocity and concentration

u_i - three coordinate velocities

$\overline{u}_i$ - time averaged coordinate velocities

V - depth averaged velocity in the z direction

$\overline{\overline{v'c'}}$ - depth and time averaged velocity in the z direction and concentration fluctuations

$\overline{v_d c_d}$ - deviations from the depth averaged z direction velocity and concentration

v_i - test functions

W - channel width

ω - wave speed

W' - distance from the closets bank to the center of the plume

X_i - nodal coordinate in the x direction

Z_i - nodal coordinate in the z direction

$\{F\}$ - force matrix

Ω - domain of the problem

1. INTRODUCTION

The urbanization of our society in the recent century has brought with it the immense problem of liquid waste disposal. A common form of disposal currently used is to dilute the waste in a large body of water, which would then with time decompose in the water. To design disposal systems on a river the engineer must understand the physical behavior of pollutants in rivers. This understanding requires the development of efficient and accurate methods of analysis. This thesis attempts to further the use of the finite element method as a vital and essential tool in the engineers ability to effectively predict the behavior of passive pollutants in rivers.

2. LITERATURE REVIEW

2.1 Depth averaged Pollutant Conservation Equation

This section presents the derivation of the depth averaged advection diffusion equation which describes the behavior of a passive pollutant in rivers. A brief introduction to the physical processes important in the modelling of plumes is also included.

2.1.1 Pollutant Conservation Equation

The behavior of a conservative passive pollutant in an incompressible laminar fluid flow can be shown to be described differentially by the expression,

$$\frac{\partial c}{\partial t} + u_i \frac{\partial c}{\partial x_i} = \frac{\partial}{\partial x_i} D_{ij}\left(\frac{\partial c}{\partial x_j}\right) + p \quad . \tag{2.1}$$

In this expression c is the concentration of the pollutant, u is the velocity of the fluid, D is the molecular diffusion coefficient and p represents a source or a sink of pollutant. The index i represents the summation over the three Cartesian coordinate as shown in Figure 2.1.

This equation is only applicable to laminar flow, but it also represents the conservation of pollutants in turbulent flows at an instant in time and space. The time and space interval over which this equation is valid is so small that, until new faster supercomputers are available, solution of this equation for a turbulent flow situation is impossible.

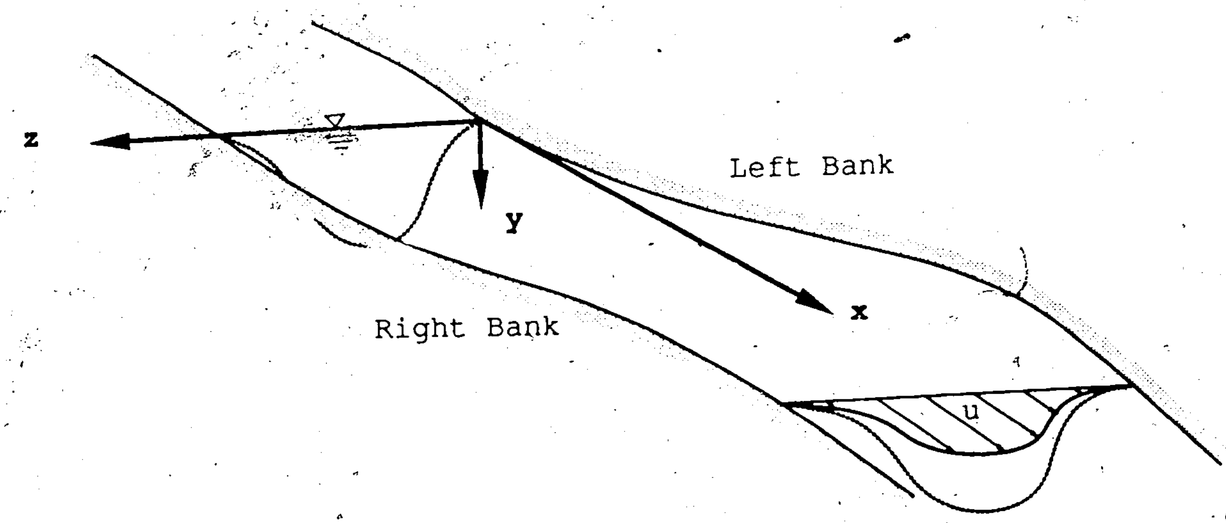


Figure 2.1 Definition Sketch with the Cartisian Coordinate System

In general this equation is time averaged to arrive at an equation applicable to turbulent flows. To time average Equation 2.1, the variable at an instant of time is expressed as a sum of its time averaged and fluctuating components, stated as $u_i = \overline{u}_i + u'$ and $c = \overline{c} + c'$. The time averaged Equation 2.2 is arrived at after substituting into Equation 2.1 these expressions and simplifying.

$$\frac{\partial \overline{c}}{\partial t} + \overline{u}_i \frac{\partial \overline{c}}{\partial x_i} = \frac{\partial}{\partial x_i}\left(- \overline{u_i'c'}\right) + \frac{\partial}{\partial x_i} \overline{D_{ij}}\left(\frac{\partial \overline{c}}{\partial x_i}\right) + \overline{p} \qquad (2.2)$$

In this equation the bars denote time averaged quantities defined as

$$\overline{(.)} = \frac{1}{T} \int_t^{t+T} (.) \quad dt, \qquad (2.3)$$

and the primes indicate fluctuating components of the respective quantities. The additional term on the right hand side $\overline{u_i'c'}$, represents the turbulent flux of fluctuating concentrations carried by the fluctuating velocity field. These terms describe the dominant mixing processes which dilute the effluent in the river.

There exists no general solution for Equation 2.2 for situations that are of concern to the engineer. The engineer is forced to devise an approximate solution for problems he is concerned with.

2.1.2 Depth Averaged Equations

A practical alternative that has been used in the past is to solve the depth averaged equation and apply it to situations where the mixing with respect to the depth is fully accomplished (Beltaos 1978, Fisher *et al* 1979, Harden *et al* 1979, Krishnappan *et al* 1983 and Sayre 1973). The depth averaged equation can be derived by integrating Equation 2.2 with respect to the depth and dividing the resulting equation by the depth. The resulting depth averaged equation is

$$\frac{\partial C}{\partial t} + U\frac{\partial C}{\partial x} + V\frac{\partial C}{\partial z} = \frac{\partial}{\partial x}\left(-\overline{\overline{u'c'}} - \overline{u_d c_d}\right) + \frac{\partial}{\partial y}\left(-\overline{\overline{v'c'}} - \overline{v_d c_d}\right) + P \qquad (2.4)$$

In this equation C, U, V, and P are the depth averaged quantities defined as $\frac{1}{h}\int_0^h (.)\,dy$. The depth averaging dispersion terms $\overline{u_d c_d}$ and $\overline{v_d c_d}$ represent the effect of the non-uniform velocity distribution with respect to the depth. The quantities u_d, v_d and c_d are the deviations from the average for each quantity. The terms $-\overline{\overline{u'c'}}$ and $-\overline{\overline{v'c'}}$ are the depth and time averaged turbulent fluctuations.

2.1.3 Diffusion Processes

The time and depth averaging results in two new terms, $-\overline{\overline{u'c'}}$ and $-\overline{\overline{v'c'}}$, that represent the mixing caused by the turbulent fluctuations. In his pioneering paper, G. I.

Taylor (1921) showed that momentum turbulent fluctuations can be modelled as a Fickian behavior, given that the turbulent time scale is much smaller than the time scale of the general flow. Similarly the fluctuating components in the time averaged pollutant conservation equation can also be modelled assuming Fickian behavior. This Fickian process implies that the fluctuation components can be modelled by a simple gradient law expression such as shown below (Elhdi *et al* 1984),

$$-\rho_m \overline{u_i'c'} = \varepsilon_{ij} \frac{\partial \bar{c}}{\partial x_j} \quad (2.5)$$

Where ε_i represents the eddy diffusivity in the three respective directions. From dimensional considerations it can be deduced that the eddy diffusivity is a product to a length scale and a velocity scale (Krishnappan and Lau 1977). The length scale used generally is either the depth or the width of the river and the velocity scale used is either the reach average velocity or the shear velocity $U_* \equiv \sqrt{gRS_f}$.

2.1.3.1 Transverse Diffusion

The most important factor governing the mixing process in a steady plume in a river is the transverse mixing of the plume in the fluid flow. This diffusive process governs the rate of lateral spreading of the plume.

Through the use of dimensional arguments a relationship that governs the behavior of ε_z, the transverse eddy diffusivity, was looked at in great detail by Krishnappan and Lau (1977). They concluded that the best fit to the experimental values documented is given by the relationship

$$\frac{\varepsilon_z}{U_* W} = f\left(\frac{U}{U_*}, \frac{W}{h}\right) \tag{2.6}$$

where W is the average width of the river and h is the reach average depth. Figure 2.2 shows a plot of this relationship as given by Krishnappan and Lau (1977). The original plot presented by Krishnappan and Lau shows that there is a strong dependence on friction factor. A convenient expression that accounts for this 'roughness' effect can be derived if the expression

$$\frac{U}{U_*} = 2.5 \ln\left(\frac{12R}{k}\right) \tag{2.7}$$

is used to incorporate the relationship between the hydraulic radius R ($\cong$ h for a river) and the conveyance coefficient $C_* \equiv \frac{U}{U_*}$. In this equation, k is the roughness height of the channel bed. A possible dimensionless relationship resulting from this is

$$\frac{\varepsilon_z}{UW} = f\left(\frac{k}{W}\right) \tag{2.8}$$

The experimental data is plotted again using this new relationship in Figure 2.3 . This plot suggests that an approximate value can be obtained from the linear regression line,

$$\frac{\varepsilon_z}{UW} = 0.0006538 + 0.0195\left(\frac{k}{W}\right). \tag{2.9}$$

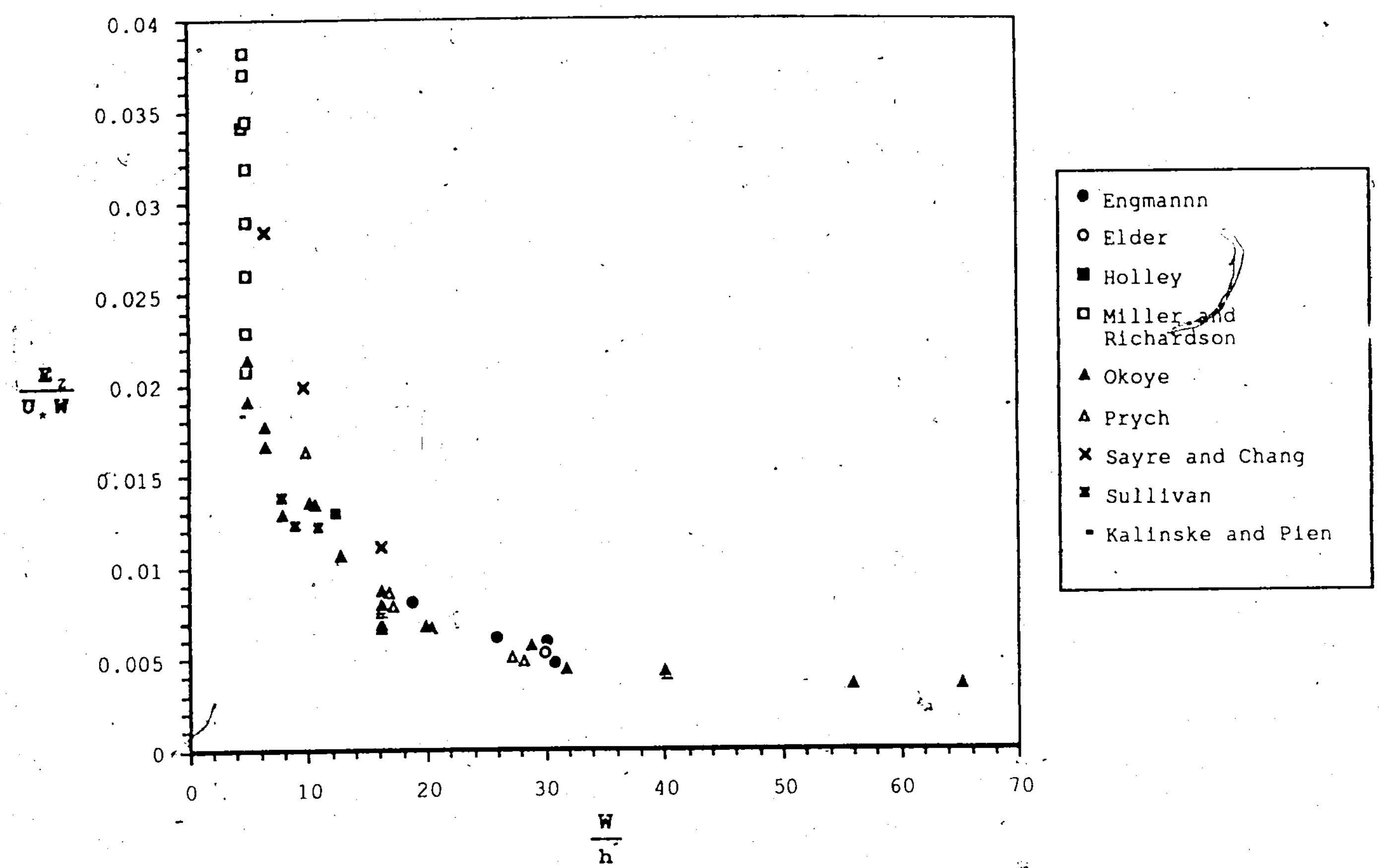


Figure 2.2 Relationship between transverse diffusivity and aspect ratio (modified from Krishnappan and Lau (1977))

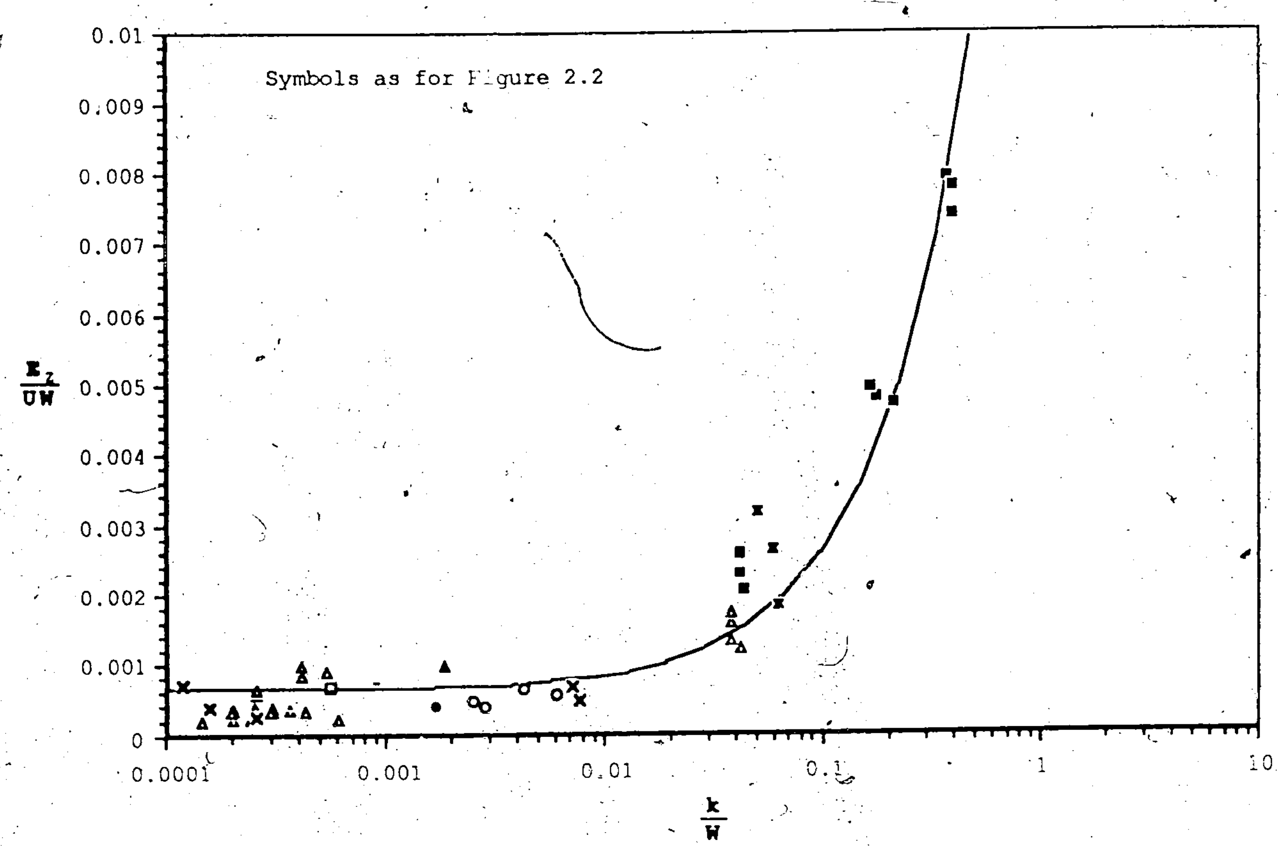


Figure 2.3 Relationship for Transverse Diffusivity

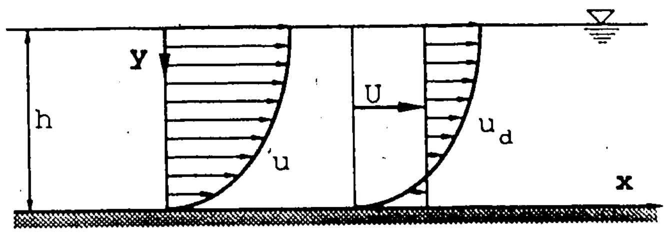
2.1.3.2 Longitudinal Diffusion

Longitudinal diffusivity can also be thought to follow a similar relationship as the transverse diffusivity. Even though this has not been studied in as extensively as the transverse diffusion, studies done by Sayre and Chang (1968) suggest that it is two to three times larger than the transverse diffusivity. The longitudinal diffusivity can be approximated in a similar manner by the expression

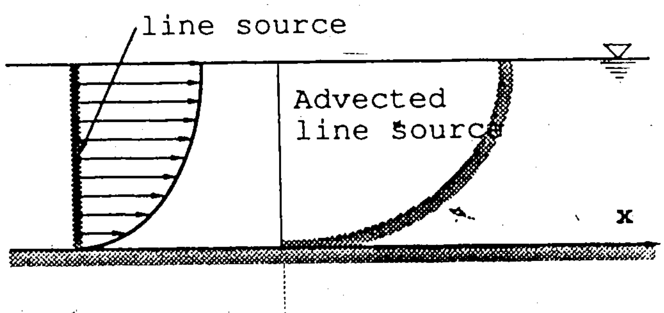
$$\frac{\varepsilon_x}{UW} = 0.0006538 + 0.0293 \left(\frac{k}{W} \right) \qquad (2.10)$$

2.1.4 Dispersion Processes

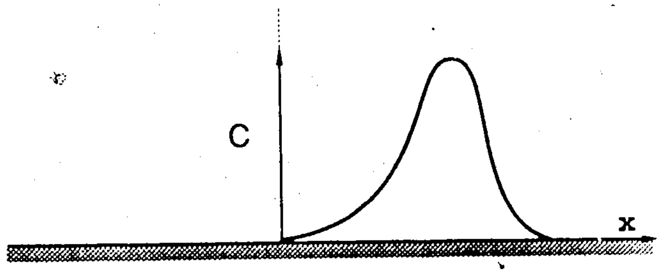
The depth averaged concentration equation hides the effects of any variation of properties in the vertical direction. One important distortion resulting from depth averaging is the smearing of pollutant in plan by the non-uniform vertical velocity distribution. Take the case of the release of a trace of pollutant in an infinitely wide channel, which has a logarithmic velocity distribution in the vertical. The tracer will be advected downstream at different velocities depending on its the distance above the bed. The result is that the cloud disperses in the horizontal plane, as illustrated in Figure 2.4. This dispersion manifests itself in the depth averaged concentration measurements, by showing a distinct skewness to the depth averaged concentration distribution. After a long



(a) Velocity Deviations for a logarithmic distribution.



(b) Advection of a line source by the logarithmic velocity distribution.



(c) Skewed distribution of the depth averaged concentration resulting from a non-uniform velocity distribution.

Figure 2.4 Effects of a logarithmic velocity distribution on the Depth averaged concentrations.

period of time this skewed distribution deteriorates into a Gaussian distribution. The effects of this behavior is incorporated by modelling the two dispersion terms , $\overline{u_d c_d}$ and $\overline{v_d c_d}$, which result from the depth averaging.

There exist two distinct regions of interest in the modelling of these dispersion terms. They consist of the initial developing region and a final 'steady' region.

2.1.4.1 Final 'Steady' Dispersion

In the final 'steady' stage of dispersion of the pollutant, it is believed that the variance of distribution grows linearly with time (Fisher *et al.* 1979). G. I. Taylor (1921) showed that if the variance can be thought of as growing linearly with time, that is

$$\frac{d\sigma}{dt} = \text{constant}, \tag{2.11}$$

then it follows that the process can be modelled as a Fickian behavior. Thus the dispersion terms can be modelled as,

$$\overline{u_d c_d} = - K_{xx} \frac{dC}{dx} - K_{xz} \frac{dC}{dz} \text{ and}$$
$$\overline{v_d c_d} = - K_{zx} \frac{dC}{dx} - K_{zz} \frac{dC}{dz}, \tag{2.12}$$

where K is defined by the following integrals,

$$K_{xx} = - \int u_d \int \frac{1}{\varepsilon} \int u_d \, dy \, dy \, dy,$$

$$K_{xz} = - \int u_d \int \frac{1}{\varepsilon} \int v_d \, dy \, dy \, dy,$$

$$K_{zx} = - \int v_d \int \frac{1}{\varepsilon} \int u_d \, dy \, dy \, dy \text{ and}$$

$$K_{zz} = - \int v_d \int \frac{1}{\varepsilon} \int v_d \, dy \, dy \, dy, \qquad (2.13)$$

In these integrals ε is the vertical turbulent eddy diffusion. Elder (1959) showed that for a logarithmic velocity distribution in the vertical direction in an infinitely wide channel, the corresponding expression for the dispersion coefficient in the x direction is

$$K_{xx} = 5.93 \, U_* \, h \qquad (2.14)$$

where U_* is the shear velocity and h is the depth.

This final region is defined to be after the cloud has travelled a characteristic length, after which the only dominant process is one dimensional dispersion. For a river this occurs after the slug occupies the full cross section. Fisher(1979) suggests that this occurs after a length,

$$L_i = \frac{1.8 \, W'^2 \, U}{h \, U_*}, \qquad (2.15)$$

where W' is the distance to the farthest bank from the center of the slug and U is the cross-sectional average velocity. Prior to this initial region, the use of a Fickian approximation is invalid.

2.1.4.2 Initial Transient Dispersion

In the initial region of the slug movement, there has not been any physically sound analysis performed. An attempt was made by S. Beltaos(1980) to define this region using an empirical fit to predict the behavior of the slug variance for the initial region. Though his method of analyses is quite firm, the use of an empirical relation that has no physical basis makes the analyses unattractive.

The lack of a sound predictive model for the physical behavior of dispersion limits the ability of numerical methods in predicting the measured values for river slug tests.

2.1.5 Modelled Equation

The depth averaged equation modelled after the simplification made for the dispersion and eddy diffusivity terms is

$$\frac{\partial C}{\partial t} + U_i \frac{\partial C}{\partial x_i} - \frac{\partial}{\partial x_i} E_{ij} \frac{\partial C}{\partial x_j} - P = 0 \qquad (2.16)$$

where E_{ij} is the total diffusion defined as

$$E_{ij} = \varepsilon_{ij} + K_{ij} \ . \qquad (2.17)$$

Equation 2.16 is applicable to situations where the plume is well mixed in the vertical direction and the longitudinal dispersion can be neglected.

2.2 Past Solution Techniques

The full three dimensional equation of conservation of a passive pollutant has no general solution for any situation. The solution techniques vary from the simple analytical solutions for idealized situations, to the more complicated finite element discrete solution of a general river reach. The strategies that have been employed by the engineer to solve the problem can be divided into two basic categories, namely analytical solutions and numerical solutions of simplified forms of Equation 2.1.

2.2.1 Analytical Solutions

Analytical solution may not be as precise in reflecting the behavior in a general river as some of the other numerical solutions, but they do provide an excellent tool in assessing initial investigations. Basically the analytical solutions can be categorized into two groups, namely one dimensional and two dimensional solutions.

2.2.1.1 One Dimensional Solutions

Generally the one dimensional solutions are used to predict the behavior of a slug test. In a slug test a large dose of pollutant is introduced into the river. The dilution of the slug due to diffusion and dispersion can be predicted using a one dimensional solution. Once the slug has occupied the entire channel width the dominant processes that are of interest are those of advection and longitudinal dispersion of the slug as it travels downstream.

To solve for this situation, the full equation has to be averaged over the cross section to arrive at the unsteady one dimension advection diffusion equation,

$$\frac{\partial \overline{C}}{\partial t} + \overline{U}\frac{\partial \overline{C}}{\partial x_i} = \frac{\partial}{\partial x_i}\overline{E}_x\left(\frac{\partial \overline{C}}{\partial x_i}\right) \tag{2.18}$$

In this equation $\overline{C}$, $\overline{U}$, and $\overline{E}_x$ are quantities averaged over the cross section. The averaged terms are defined as

$$\overline{C} = \int_0^W\int_0^h \overline{c}\; dy\; dz \tag{2.19}$$

and $\overline{E}_x$ is the longitudinal dispersion defined as

$$\overline{E}_x = \frac{-\rho\overline{u_D\;c_D}}{\frac{\partial \overline{C}}{\partial x}} \tag{2.20}$$

where u_D and c_D are the deviations from the cross-sectional average values. This dispersion due to the velocity profiles can only be modelled as a diffusive process using gradient laws if they behave in a Fickian manner. As mentioned earlier in Section 2.1.4.1, the distance after which this analysis is applicable is L_i.

The analytical solution to Equation 2.18 is

$$\overline{C} = \frac{M}{2\;A\sqrt{\pi\overline{Ex}\;t}}\;e^{\left(\frac{-(x-\overline{U}\;t)}{4\;\overline{Ex}\;t}\right)} \tag{2.21}$$

where M is the total quantity of tracer introduced into the river, and A is the average cross-sectional area.

2.2.1.2 Two Dimensional Solutions

Analytical solutions to problems based on solving the depth averaged equation similar to Equation 2.4 exist for steady plumes from line and point sources. Equation 2.4 is for a Cartesian coordinate system. However, no natural river is ever going to follow such an orthogonal coordinate system. To overcome this problem, Yotsukura and Sayre (1976) introduced a coordinate system fitted to the general shape of rivers, known as a body fitted curvilinear coordinate system, shown in Figure 2.5. The depth averaged equation in this coordinate system according to Yotsukura and Sayre can be expressed as,

$$m_x m_z \frac{\partial C}{\partial t} + \frac{\partial}{\partial x}\left(m_z UC\right) + \frac{\partial}{\partial z}\left(m_x VC\right) = \frac{\partial}{\partial x}\left(\frac{m_z}{m_x} E_x \frac{\partial C}{\partial x}\right) + \frac{\partial}{\partial z}\left(\frac{m_x}{m_z} E_z \frac{\partial C}{\partial z}\right) \qquad (2.22)$$

in which E_x and E_z are the sum of the turbulent diffusivity and dispersion coefficients, m_x and m_z are metric coefficients to account for the coordinate system transformation and all capitalized variables are depth averaged quantities.

The solution to Equation 2.22 may be obtained, but it would be of little benefit due to the large data base required. River velocity surveys seldom give V velocity measurements and for the purpose of preliminary analysis it has little effect. The solution generally used is a solution to a steady state situation where the longitudinal dispersion has little effect, since it is dependent on the longitudinal gradient which is very small. Applying these simplifications to Equation 2.22, the resulting equation can be stated as,

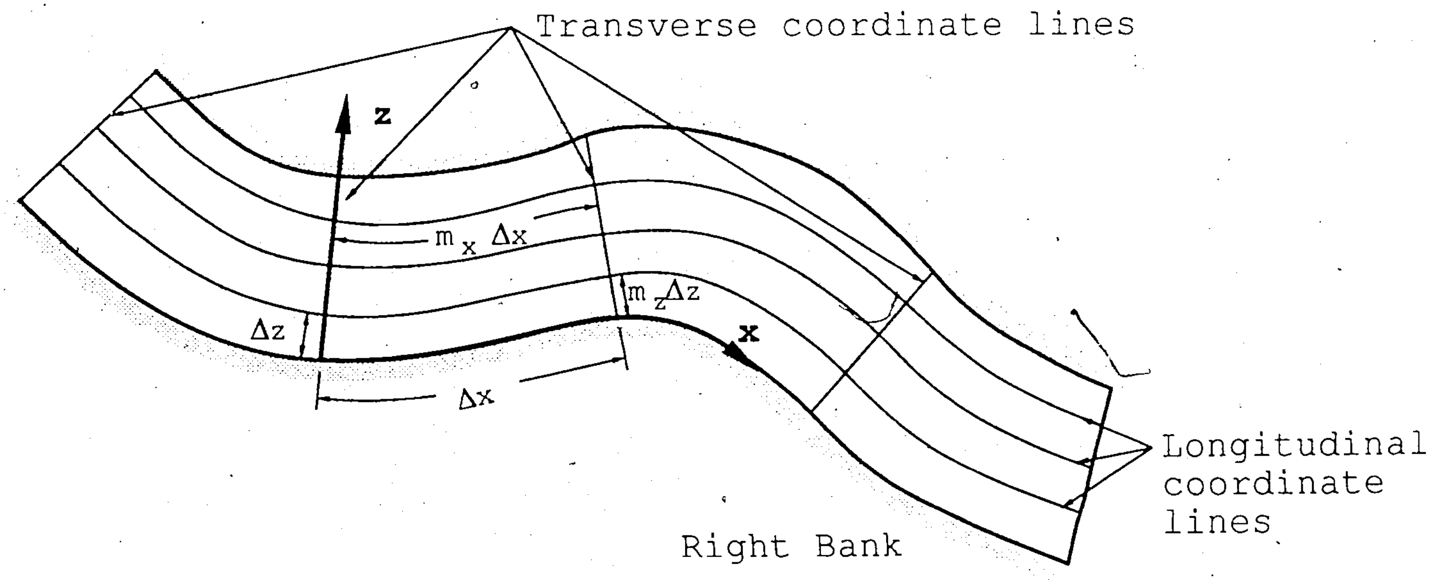


Figure 2.5 Curvilinear coordinate system

$$\frac{\partial}{\partial x}(m_z UC) = \frac{\partial}{\partial z}\left(\frac{m_x}{m_z} E_z \frac{\partial C}{\partial z}\right). \qquad (2.23)$$

Introducing the cumulative discharge concept defined as

$$q_c = \int_0^z m_z h U \, dz \qquad (2.24)$$

in place of z, Yotsukura and Sayre (1976) show that Equation 2.23 is the same as

$$\frac{\partial C}{\partial x} = \frac{\partial}{\partial q_c}\left(U h^2 m_z E_z \frac{\partial C}{\partial q_c}\right). \qquad (2.25)$$

The analytical solution of Equation 2.25 for a point source located at the bank, with constant E_z over the whole domain of the channel is

$$\frac{C}{C_\infty} = \frac{2}{\sqrt{2\pi\xi}}\left[e^{\left(\frac{-\eta^2}{2\xi}\right)} + \sum_{m=1}^{\infty}\left(e^{\left(\frac{-(2m-\eta)^2}{2\xi}\right)} + e^{\left(\frac{-(2m+\eta)^2}{2\xi}\right)}\right)\right] \qquad (2.26)$$

Where $C_\infty = \frac{M}{Q}$;

$\eta = \frac{q_c}{Q}$, the normalized cumulative discharge;

and $\xi = \frac{2Uh^2 m_z E_z x}{Q^2}$, a dimensionless distance.

A similar solution shown below exists for a line source extending from the bank to where the cumulative discharge, is equal to rQ,

$$\frac{C}{C_\infty} = \frac{1}{2r}\left[\text{erf}\left(\frac{r-\eta}{\sqrt{2\xi}}\right) + \text{erf}\left(\frac{r+\eta}{\sqrt{2\xi}}\right) + \sum_{n=1}^{\infty}\left\{\text{erf}\left(\frac{2n+r-\eta}{\sqrt{2\xi}}\right) + \text{erf}\left(\frac{2n+r+\eta}{\sqrt{2\xi}}\right) - \text{erf}\left(\frac{2n-r-\eta}{\sqrt{2\xi}}\right) - \text{erf}\left(\frac{2n+r-\eta}{\sqrt{2\xi}}\right)\right\}\right] \qquad (2.27)$$

where the variables are defined as for the point source equation above.

Both of these solutions are only applicable to idealized situations where the transverse diffusivity is constant over the domain and the flow consists of U velocity only. The domain of the solution would also have an average velocity that is not radically different between various longitudinal distances.

2.2.2 Numerical Solutions

The analytical solutions discussed earlier have restrictions, in that the velocity and eddy diffusivity are averaged over the domain and the solutions cannot account for any irregularities in the flow field. To find solutions that give an accurate picture of the behavior in a natural channel, a numerical solution is necessary.

There are many numerical schemes, both finite difference and finite element, that have been applied to solve the general depth averaged Equation 2.13. Schemes that have been used in the past have been tested by trying to find an approximate solution as close as possible to the exact solution of the simple hyperbolic equation

$$\frac{\partial C}{\partial t} + U\frac{\partial C}{\partial x} = 0 \qquad (2.28)$$

The exact solution to this equation is simply

$$C(t_{new}, x) = C(t_{old}, x - x_{\tau}), \qquad (2.29)$$

where $x_\tau = \int_0^t U(\tau)\,d\tau$.

2.2.2.1 Artificial Diffusion

The problem of solving Equation 2.28 using any one of the many numerical methods available, is that to get a stable solution, the method generally introduces artificial diffusion. To understand why it is that a numerical solution should exhibit some artificial diffusion for a stable solution, we must first look at a solution algorithm that shows no artificial diffusion. One such algorithm can be derived by using a theta implicit finite difference approximation for the time derivative and a centered finite difference approximation for the spatial derivative. After some simplifications, the resulting discrete form of Equation 2.28 is,

$$C_i^{n+1} + \frac{\rho}{2}\,\theta\left[C_{i+1}^{n+1} - C_{i-1}^{n+1}\right] = C_i^n - \frac{\rho}{2}\,(1-\theta)\left[C_{i+1}^n - C_{i-1}^n\right] \qquad (2.30)$$

where ρ is the Courant number defined as $\frac{U_i^{n+1}\Delta t}{\Delta x}$, and θ is an implicit coefficient to determine where the space derivative is evaluated. $\theta = 0.5$ for a Crank Nicholson approximation, $\theta = 0.0$ for an explicit algorithm and $\theta = 1.0$ for a fully implicit algorithm.

2.2.2.1.1 von Neumann Stability Analysis

The stability of any algorithm like Equation 2.30 can be explored using the von Neumann stability analysis (Lapidus and Pinder, 1982). This analysis is restricted to discrete forms of the equation which are a linear representation at an ordinary point.

The analysis essentially consists of looking at one mode of the solution as expressed by a Fourier series approximation. The general Fourier series solution to Equation 2.28 can be expressed as

$$C(x,t) = \sum_{m=-\infty}^{\infty} \gamma_m e^{(i\ \omega\Delta x)} \quad . \tag{2.31}$$

Where γ_m are coefficients that are dependent upon the initial conditions imposed on Equation 2.28, i is the complex number $\sqrt{-1}$ and ω is the wave number defined as $\frac{2\pi}{\lambda}$ in which λ is the wavelength of a particular mode of the solution. The series solution represents a continuous solution, where as for a difference formula the solution is only solved for discrete points. The solution at any one discrete point (node) is,

$$C_i^n \equiv C(i\Delta x, n\Delta t) = \gamma^n e^{(i\omega(i\Delta x))} \quad . \tag{2.32}$$

Substituting expressions analogous to 2.32 into a discrete form and simplifying usually results in γ being a complex function of the $\omega\Delta x$. The real and imaginary parts of the complex function can be looked at by observing the behavior of its magnitude and phase components. The magnitude r and relative phase ϕ are given by the general expressions

$$r = \sqrt{\gamma_{real}^2 + \gamma_{imaginary}^2} \text{ and} \tag{2.33}$$

$$\phi = \frac{\tan^{-1}\left(\frac{\gamma_{imaginary}}{\gamma_{real}}\right)}{\omega\Delta x\rho} . \tag{2.34}$$

To observe the behavior of these components, two plots are constructed, one of r verses N and another of ϕ verses N, where N is defined as $\frac{2\pi}{\omega\Delta x}$. Physically N defines the number of nodes used to represent an initial perturbation in the domain. The two plots physically represent the dissipation(r) and dispersion(ϕ) exhibited by the discrete representation of perturbation using N nodes.

A value of 1.0 for r and ϕ represent an ideal discrete algorithm which will propagate the perturbations at the exact speed(ϕ=1.0) with no modifications to their magnitude (r=1.0). Algorithms that exhibit r > 1.0 indicate an unstable solution since any perturbations would be amplified where as in the case for r < 1.0 they will damped out. For algorithms which exhibit values of ϕ < 1.0 show dispersion where waves of high frequency(small wavelength) will be traveling at speeds lower than the exact.

Substituting appropriate expressions analogous to Equation 2.31 into the discrete Equation 2.30 and simplifying using the identity

$$e^{(-i\omega\Delta x)} = \cos(\omega\Delta x) - i \sin(\omega\Delta x) \qquad (2.35)$$

results in the following expression for γ

$$\gamma = \frac{\gamma_{real} + i\,\gamma_{imaginary}}{\gamma_{denominator}} \qquad (2.36)$$

where

$$\gamma_{real} = 1 - \theta(1-\theta)\left(\rho \sin(\omega\Delta x)\right)^2,$$

$$\gamma_{imaginary} = -\rho\left(\sin(\omega\Delta x)\right) \text{ and}$$

$$\gamma_{denominator} = 1 + \left(\theta\,\rho \sin(\omega\Delta x)\right)^2$$

The corresponding expression for r and ϕ of the discrete form 2.30 can be interpreted as,

$$r = \sqrt{\left[\frac{\gamma_{real}}{\gamma_{denominator}}\right]^2 + \left[\frac{\gamma_{imaginary}}{\gamma_{denominator}}\right]^2} \qquad (2.37)$$

and

$$\phi = \frac{\tan^{-1}\left(\dfrac{\gamma_{imaginary}}{\gamma_{real}}\right)}{\omega\Delta x\rho} \qquad (2.38)$$

Figure 2.6 shows the two plots of and ϕ verses N for $\theta = 0.5$ with Courant numbers varying from 0.1 to 1.25. These plots show that this particular formulation exhibits no dissipation, but does produce some dispersion. The inability of algorithms such as the central difference algorithm to restrict dispersion is the principal reason why they fail.

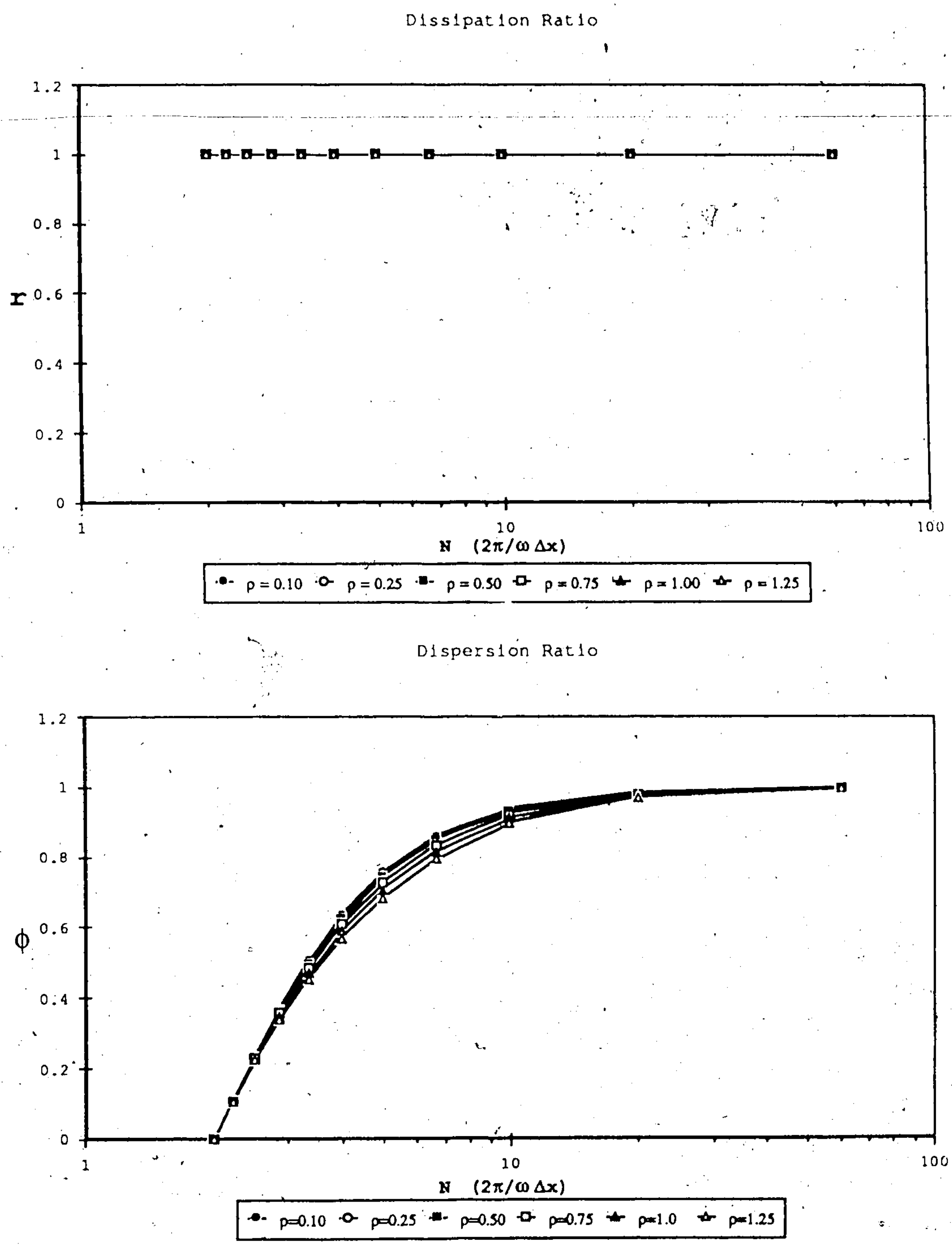


Figure 2.6 Dissipation and Dispersion for Central Difference

The effect of the excessive dispersion shown by the algorithm can be attested to the fact that as N approaches a value of 2, ϕ, which reflects the amount of dispersion, goes to zero. This implies that perturbations which can be defined by 2 nodes will not move at all. The result of this harsh restriction is clearly shown in Figure 2.7 which shows the results from using this algorithm for the advection of a gauss type wave. The propagated wave leaves a trail of oscillations behind due to this dispersion.

If a similar analysis is undertaken for a scheme which has some artificial diffusion, such as an algorithm based on an upwinded finite difference approximation for the advection term, the distortions shown by the central difference algorithm will disappear. The algorithm for such an upwinded approximation is

$$C_i^{n+1} + \rho\ \theta\left[C_i^{n+1} - C_{i-1}^{n+1}\right] = C_i^n - \rho\ (1-\theta)\left[C_i^n - C_{i-1}^n\right], \qquad (2.39)$$

and the corresponding terms for the von Neumann analysis that fit into Equations 2.38 and 2.39 are

$$\gamma_{real} = \Big(1 - \rho\ (1-\theta)(1-\cos(\omega\Delta x))\Big)\ \Big(1 + \rho\ \theta(1-\cos(\omega\Delta x))\Big)$$

$$- \theta(1-\theta)\Big(\rho\sin(\omega\Delta x)\Big)^2$$

$$\gamma_{imaginary} = \Big(1 - \rho\ (1-\theta)(1-\cos(\omega\Delta x))\Big)\ \theta\ \rho\Big(-\sin(\omega\Delta x)\Big)$$

$$+ \Big(1 + \rho\ \theta(1-\cos(\omega\Delta x))\Big)\ (1-\theta)\ \rho\Big(-\sin(\omega\Delta x)\Big)$$

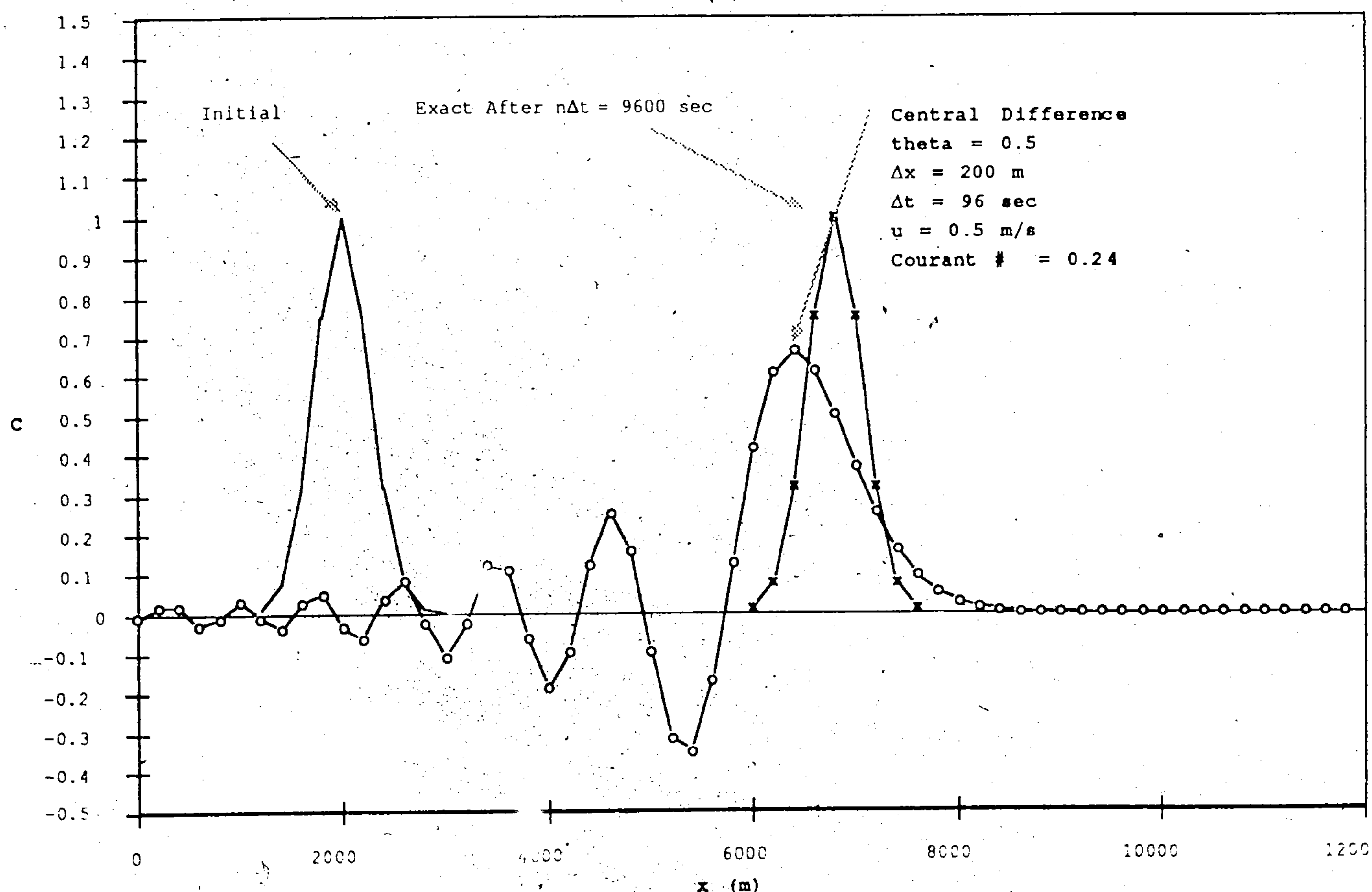


Figure 2.7 Advection of a Pulse using Central Difference

and

$$\gamma_{denominator} = \left(1 + \rho\,\theta\,(1-\cos(\omega\Delta x))\right)^2 + \left(\theta\,\rho\,\sin(\omega\Delta x)\right)^2 .$$

Figure 2.8 shows the plots for dissipation and dispersion the case when $\theta = 0.5$, a Crank Nicholson algorithm. The difference approximation has introduced artificial diffusion as illustrated by the fact that the dissipation(r) is less than one. However for cases other than $\rho = 1.0$, the algorithm shows dispersion greater than zero. This implies that any perturbations that may exist at the low frequencies will be dissipated as they are being advected. The dissipation produced by the algorithm can be seen in Figure 2.9 which shows the advection of a gauss wave at a $\rho = 0.24$ whose peak is reduced by sixty percent.

2.2.2.1.2 Taylor Series Analysis

The stability analysis discussed above shows that the upwinded schemes introduce dissipation in the solution. However to stipulate that this dissipation is in the form of diffusion, a further analysis must be undertaken. By looking at the Taylor Series representation of the discrete form, based on a general expression such as

$$C_{i+1}^{n} = C_{i}^{n} + \Delta x\frac{\partial C}{\partial x} + \frac{\Delta x^2}{2}\frac{\partial^2 C}{\partial x^2} + O(\Delta x)^3 + \ldots, \qquad (2.40)$$

the actual equation modelled by the discrete form can be deduced. The Taylor series representation of the upwinded

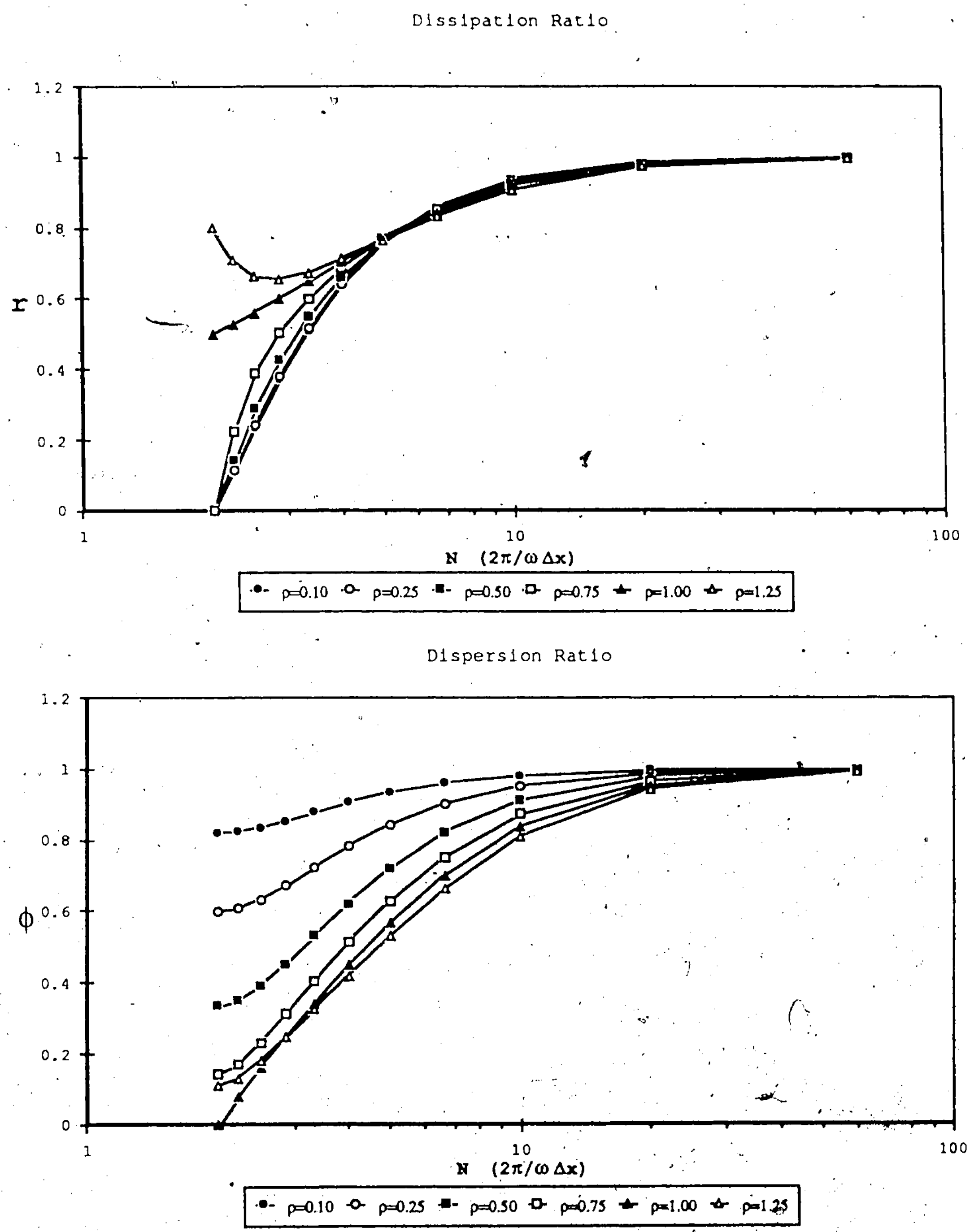


Figure 2.8 Dissipation and Dispersion for Upwinding Finite Difference

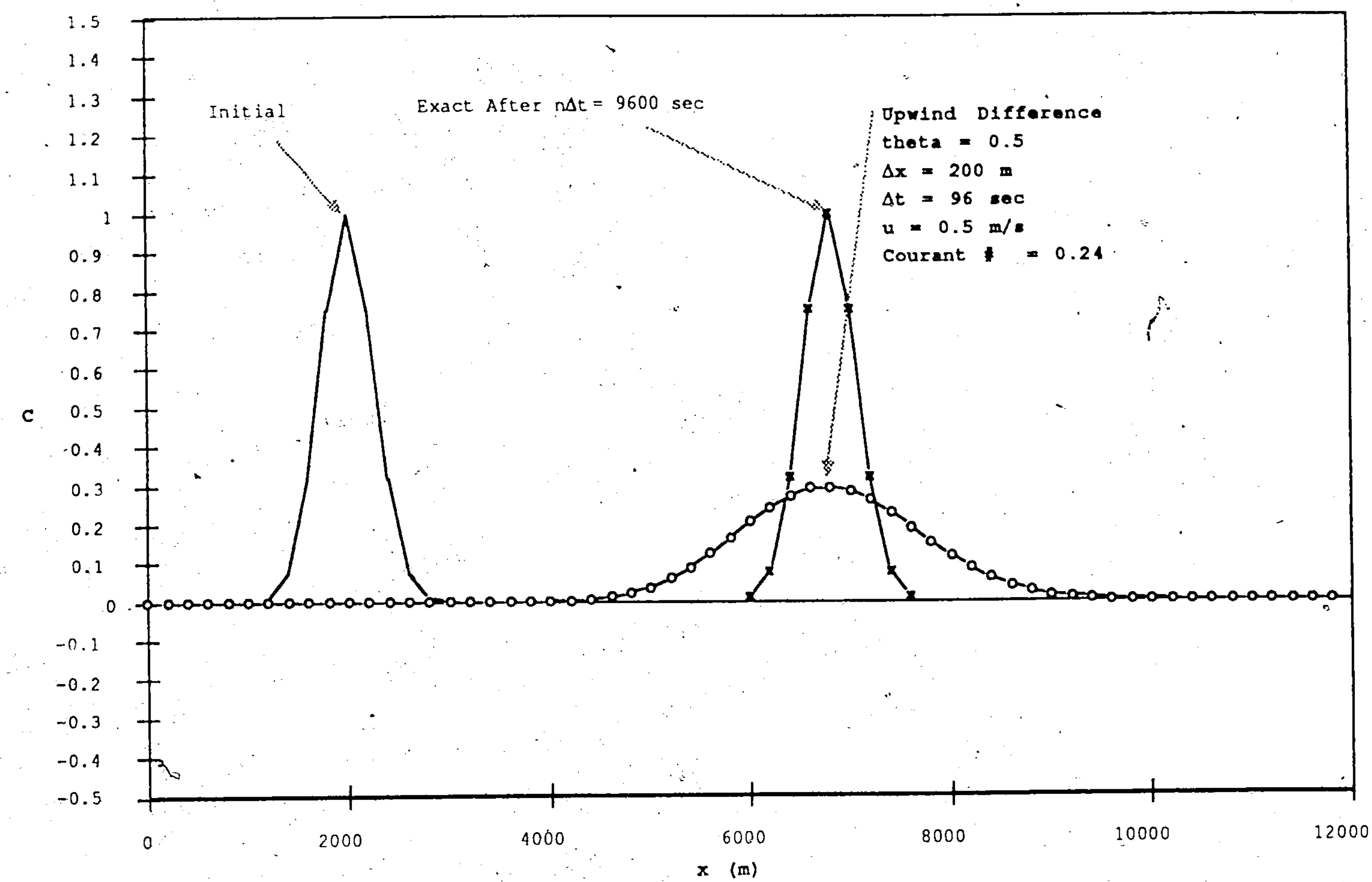


Figure 2.9 Advection of a Pulse using Upwinding FD

finite difference scheme, as given by Equation 2.39, shows that the equation modelled is actually

$$\frac{\partial C}{\partial t} + U\frac{\partial C}{\partial x} = \frac{U\Delta x}{2}\frac{\partial^2 C}{\partial x^2} + U\ (\Delta x)^2\frac{\partial^3 C}{\partial x^3}\ . \qquad (2.41)$$

From this equation it is evident that the modelled equation has an additional diffusion term which is dependent on the velocity and the discretization. Though this upwinded scheme shows that it has diffusion $\left(\text{ie } \frac{\partial^2 c}{\partial x^2}\right)$, upwinding schemes based on higher order approximations don't necessarily exhibit this artificial diffusion.

2.2.2.2 Popular Finite Difference Schemes

Many finite difference schemes that have been developed to overcome the difficulties presented by the advection diffusion equation. This section will discuss two such schemes that have been used in the analysis of the pollutant conservation equation as applied to natural rivers. One method that has been used in highly recognized programs, such as TRSMIX(G. Putz, 1984) and RIVMIX(Krishnappan and Lau, 1983), is the method developed by Stone and Brian(1963). Another method developed recently that has been used to simulate pollutant discharges through a network of rivers is the Holly-Preissmann method (Sauvaget 1985). Though these methods were used to solve two different problems, namely steady state transverse diffusion and unsteady longitudinal dispersion, they can be shown to be solving the same differential equation, with the aid of a transformation.

2.2.2.2.1 Stone and Brian Method

The method developed by Stone and Brian(1963) has been used to solve the problem of transverse diffusion of a river plume. Generally the process of transverse diffusion in a steady plume as cast into the depth averaged form for a curvilinear coordinate system is given by the equation

$$\frac{\partial}{\partial x}(m_z UC) + \frac{\partial}{\partial z}(m_x VC) = \frac{\partial}{\partial x}\left(\frac{m_z}{m_x} E_x \frac{\partial C}{\partial x}\right) + \frac{\partial}{\partial z}\left(\frac{m_x}{m_z} E_z \frac{\partial C}{\partial z}\right). \quad (2.42)$$

Since most cases involving field measurements of velocities, the V velocity is impossible to measure and the effects of longitudinal diffusion are only important over very long reaches, these terms are neglected in the analysis. The simplified equation as derived in detail by Krishnappan and Lau(1983) is then stated as

$$\frac{\partial}{\partial x}(m_z UC) = \frac{\partial}{\partial z}\left(\frac{m_x}{m_z} E_z \frac{\partial C}{\partial z}\right). \quad (2.43)$$

Investigators such as G. Putz(1984) and Krishnappan and Lau(1983) have gone on to introduce a coordinate transformations which results in the equation

$$\frac{\partial C}{\partial x} + V \frac{\partial C}{\partial \eta} = D\frac{\partial^2 C}{\partial \eta^2} \quad (2.44)$$

where

$$\eta = \frac{q_c}{Q} = \frac{1}{Q}\int_0^z m_z\, h\, U\, dz \quad (2.45)$$

$$V = \frac{1}{Q^2}\frac{\partial}{\partial\eta}\left(U\ h^2\ m_x\ E_z\right) \text{ and} \tag{2.46}$$

$$D = \left(\frac{U\ h^2\ m_x\ E_z}{Q^2}\right). \tag{2.47}$$

Equation 2.44 is now in the advection diffusion form, which is subsequently solved using the numerical algorithm presented by Stone and Brian.

To develop a sufficiently accurate algorithm for Equation 2.44 Stone and Brian employed a general difference equation which is dependent on weighting coefficients. The difference equation developed was

$$\frac{1}{\Delta x}\left[g(C_{i+1,j} - C_{i,j}) + \frac{\theta}{2}(C_{i+1,j-1} - C_{i,j-1}) + m(C_{i+1,j+1} - C_{i,j+1})\right] +$$

$$\frac{V_{i,j}}{\Delta\eta}\left[a(C_{i,j+1} - C_{i,j}) + \frac{\varepsilon}{2}(C_{i,j} - C_{i,j-1}) + b(C_{i+1,j+1} - C_{i+1,j}) +\right.$$

$$\left.d(C_{i+1,j} - C_{i+1,j-1})\right] = \frac{D_{i,j}}{2\Delta\eta^2}\left[(C_{i,j+1} - 2C_{i,j} + C_{i,j-1}) +\right.$$

$$\left.(C_{i+1,j+1} - 2C_{i+1,j} + C_{i+1,j-1})\right] \tag{2.48}$$

where the coefficients a, $\frac{\varepsilon}{2}$, b, and d are weighting coefficients to evaluate the derivative $\frac{\partial C}{\partial\eta}$ and g, $\frac{\theta}{2}$, and m are for evaluating the 'time' derivative $\frac{\partial C}{\partial x}$. For a meaningful approximation the coefficients have to satisfy the conditions

$$a + \frac{\varepsilon}{2} + b + d = 1.0 \text{ and} \tag{2.49}$$

$$g + \frac{\theta}{2} + m = 1.0. \tag{2.50}$$

The diffusion term $\frac{\partial^2 C}{\partial \eta^2}$ is approximated using a Crank-Nicholson difference formula.

Stone and Brian go through a series of analyses, including the von Neumann Stability analysis, to arrive at optimum values for the coefficients:

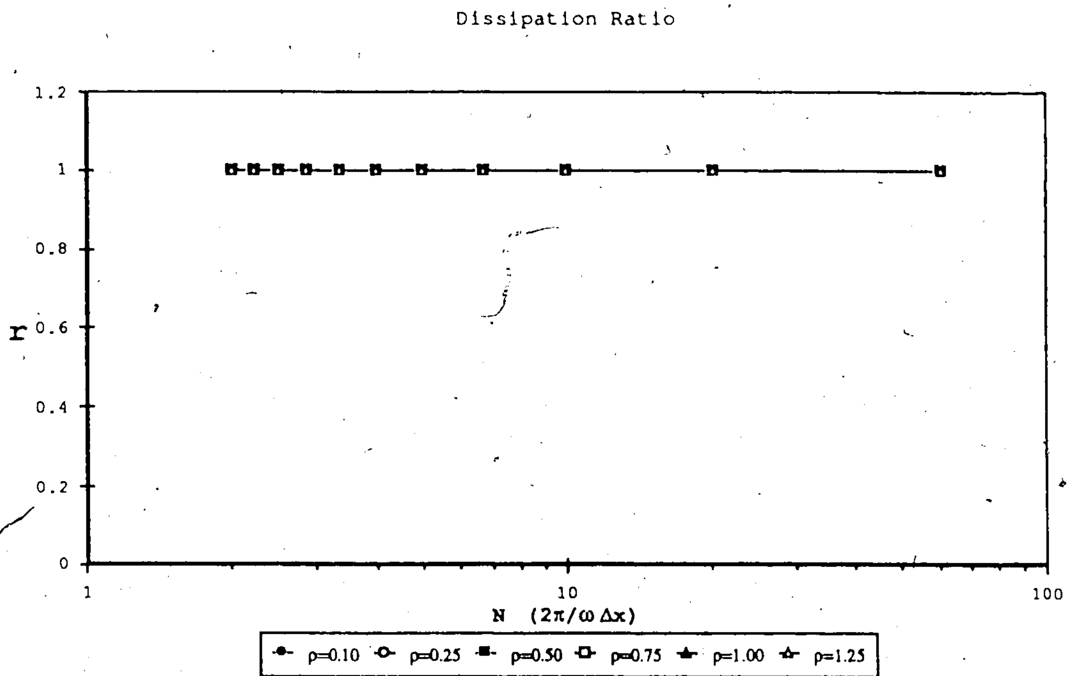
$$g = \frac{2}{3};\ m = \frac{\theta}{2} = \frac{1}{6};\ a = b = d = \frac{\varepsilon}{2} = \frac{1}{4}\ . \qquad (2.51)$$

The resulting simplified difference equation from these optimum coefficients is

$$C_{i+1,j-1}\left(\frac{1}{6\rho'} - \frac{1}{4} - \frac{1}{2Pe}\right) + C_{i+1,j}\left(\frac{2}{3\rho'} + \frac{1}{2Pe}\right) + C_{i+1,j+1}\left(\frac{1}{6\rho'} + \frac{1}{4} - \frac{1}{2Pe}\right)$$
$$+ C_{i,j-1}\left(\frac{-1}{6\rho'} - \frac{1}{4} - \frac{1}{2Pe}\right) + C_{i,j}\left(\frac{-2}{3\rho'} + \frac{1}{2Pe}\right) + C_{i,j+1}\left(\frac{-1}{6\rho'} + \frac{1}{4} - \frac{1}{2Pe}\right)$$
$$= 0.0 \qquad (2.52)$$

where ρ' is a numerical Courant number define as $\frac{V_i\ \Delta x}{\Delta \eta}$ and Pe is the Peclet number defined as $\frac{V_{i,j}\Delta x}{D}$.

Figure 2.10 shows the resulting plots for the dissipation and dispersion resulting from the von Neumann stability analysis. This algorithm, like the centered difference algorithm, shows no dissipation and a similar dispersion behavior. The solution of the advection of a gauss shaped wave is shown in Figure 2.11 for a ρ' of 0.24. Though this solution shows some oscillations, they are a great deal smaller than those observed for centered



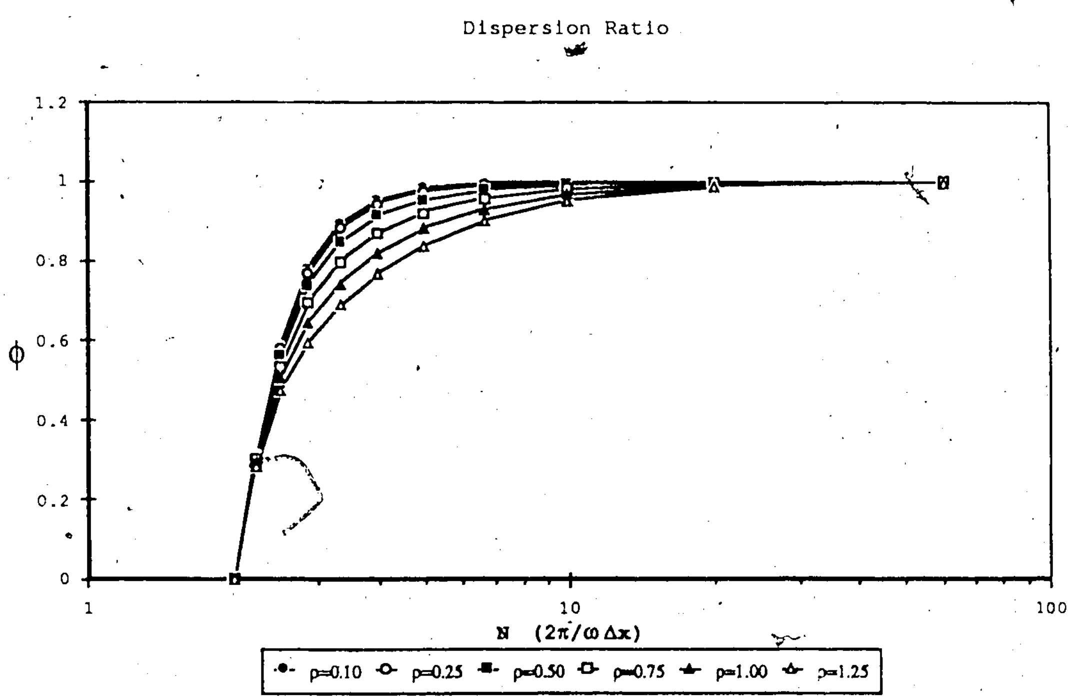


Figure 2.10 Dissipation and Dispersion for Stone and Brian

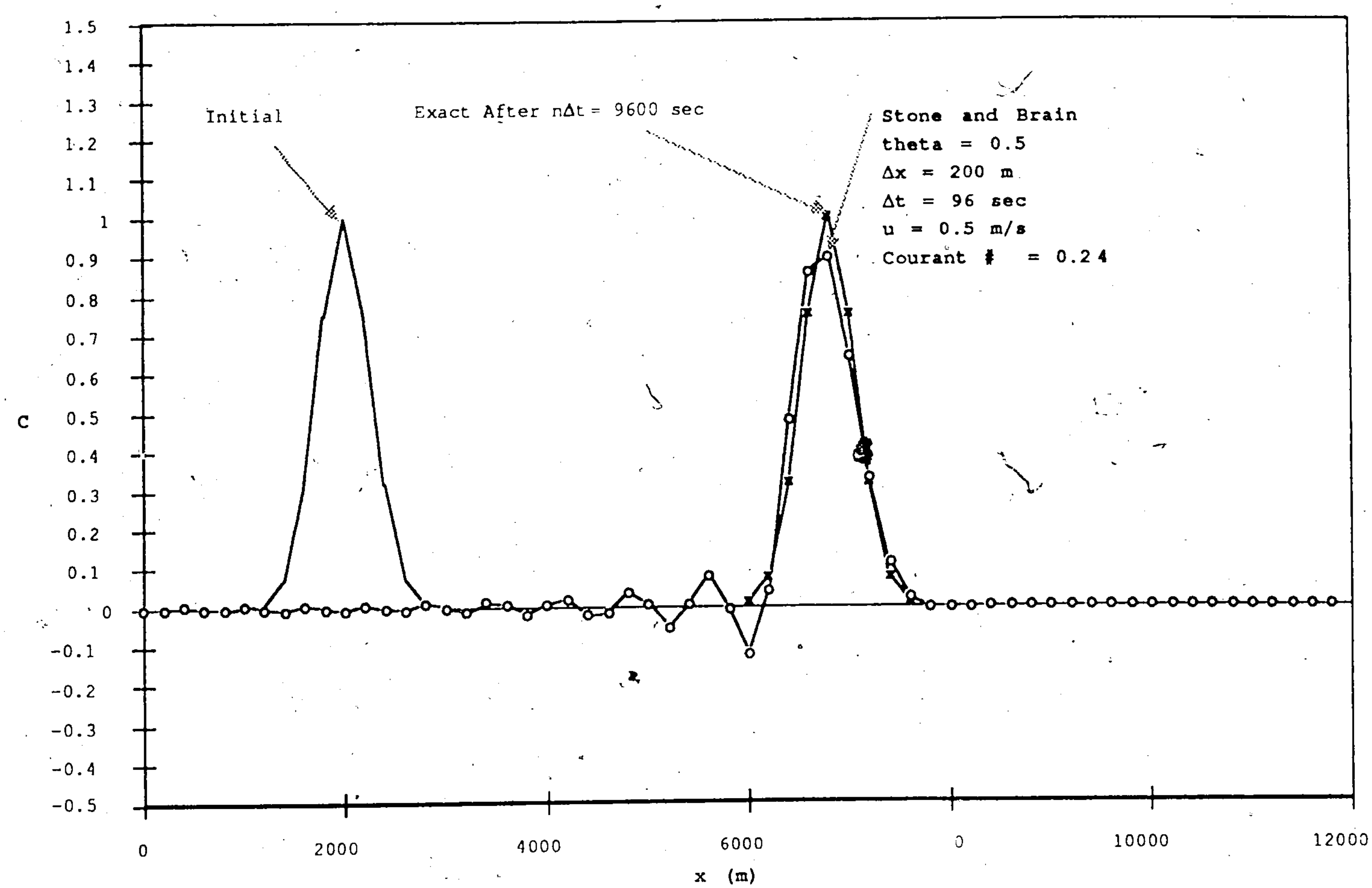


Figure 2.11 Advection of a Pulse using Stone and Brian

difference. A Taylor series analysis shows that this implicit algorithm models the equation

$$\frac{\partial c}{\partial x} + V\frac{\partial c}{\partial \eta} = D\frac{\partial^2 c}{\partial \eta^2} + O(\Delta\eta)^2 + O(\Delta x)^2. \tag{2.53}$$

This algorithm is satisfactory in most applications as the effective Courant number $\left(\frac{V\Delta x}{\Delta\eta}\right)$, used is very small.

2.2.2.2.2 The Holly-Preissmann Method

One of the newer methods to be applied to the solution of pollutant conservation in river systems is the Holly-Preissmann method (Sauvaget 1985). Basically their method involves the use of Hermite Polynomials to interpolate concentrations and concentration gradients, which are subsequently advected by the Characteristic method to the new time step. Due to the iterative nature of the solution algorithm, a simplified version for the case of constant velocity field was used to solve for the advection of the one dimensional pulse.

The results of this test is shown in Figure 2.12 for a D = 0.24, as was done for the other methods. The plot shows that though the pulse is advected at the right speed, there does appears to be some dissipation of the peak. There is also the persistence of a pair of small wiggles on either side attached to the pulse. These observations tend to indicate that though the solution algorithm shows very little dispersion, high frequency disturbances still produce

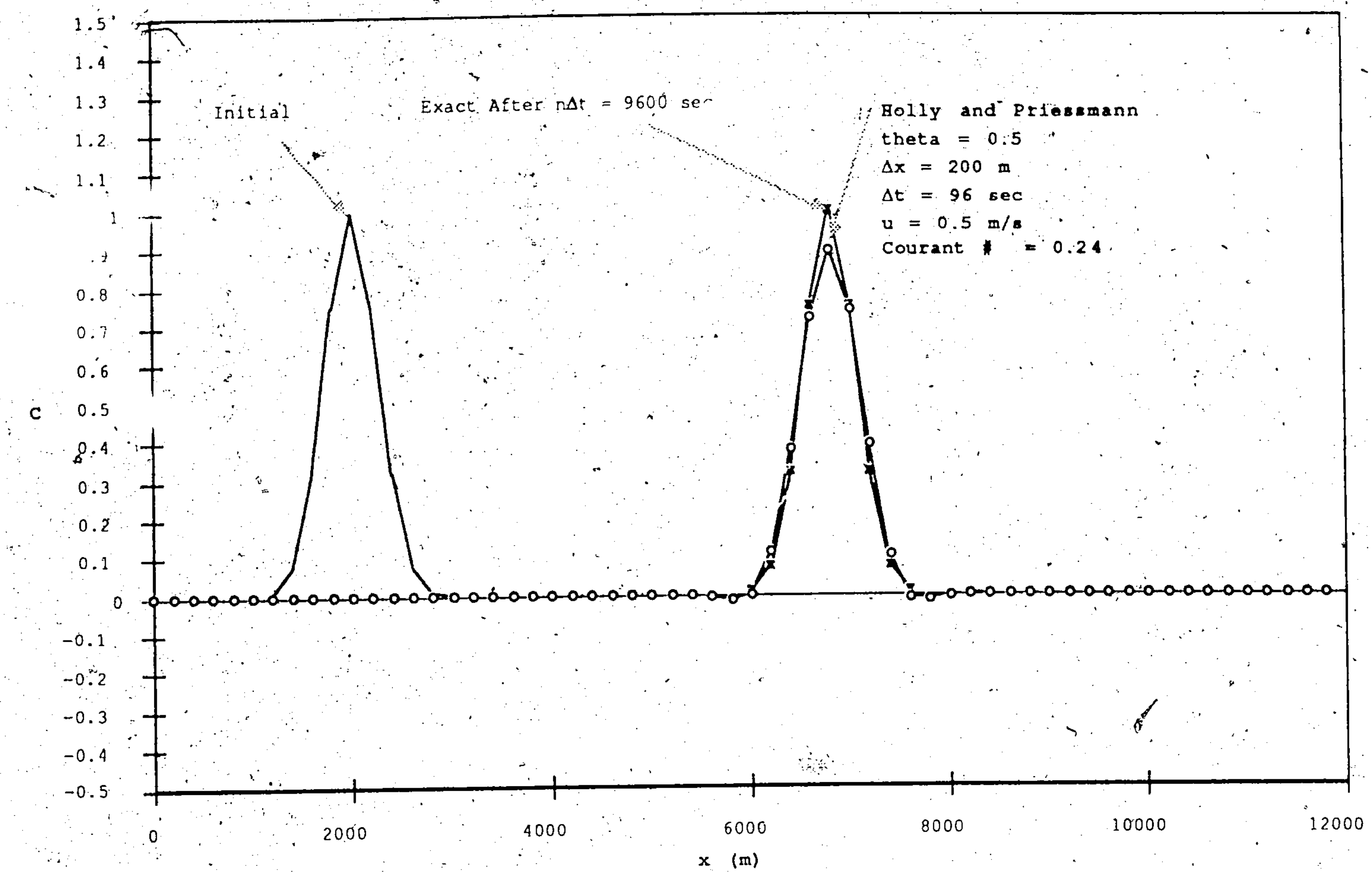


Figure 2.12 Advection of a Pulse using Holly and Preissmann

oscillations, even at this low a Courant number (0.24). There also appears to be some dissipation of the peak.

2.2.2.3 Finite Element Methods

Generally the finite element method has been restricted to use by the mathematicians due largely to its complexity. However recently a number of investigators have applied the Finite Element formulation to solve basic hydraulic problems. One of the first to apply this method to the conservation equation was Leikuhler *et al.* (1974). The formulation of the finite element consisted of the Method of Weighted Residuals using triangular elements. These linear triangular elements, at $\theta = 0.5$, give exactly the same results as for one dimensional advection of a gauss type wave using the Stone and Brian method discussed earlier. The results from their study indicate that the prospect of getting a viable solution using the Finite Element method seem to be very good.

3. FINITE ELEMENT METHOD

The advancement of Computational Fluid Mechanics is hindered by two basic restrictions. One limitation is the computing power available to perform the calculations and the other is the accuracy of the method, which is related to its complexity. Modelling fluid flow for practical applications requires that a balance be achieved between the degree of discretization and the complexity of the algorithm used. Due to these limitations the modeler has to develop new and generally more complicated models to accurately model the behavior in question.

The hydraulic engineer is further faced with the fundamental problem of irregular boundaries which do not conform to any orthogonal coordinate system. A method that has the ability to overcome the limitations of boundary fitted coordinate systems is the Finite Element Method (FEM). In the Finite Element Method the domain of a problem is discretized into a number of elements that can be of varying shape, size, orientation, and numerical accuracy. The discretized domain is subsequently formulated into a matrix which is solved for the desired unknowns.

The following is a general introduction to the FEM as applied to the depth averaged conservation equation (Equation 2.16). An introduction to the use of a new type of upwinding elements is presented. The flow chart of the program developed to implement these new elements is also discussed.

3.1 Method of Weighted Residuals

A literature review of the Finite Element Methods would realize a variety of different formulations that all result in the discretization of the domain of the problem into finite elements. Due to its simplicity and elegance, the general formulation of the method of Weighted Residuals was used for the present investigation.

Basically the premise of any finite element method as related to solving differential equations is to reduce the constraints of the problem so that a set of simple algebraic functions can describe the solution over a discrete element.

3.1.1 The Weak Statement

The first step in the Method of Weighted Residuals is to derive the weak statement of the governing equation. The method requires that the solution should be approximated over the finite element by a set of basis functions, so that

$$C(t,x,z) \cong \sum_{j=1}^{N} C_j(t)\ f_j(x,z), \qquad (3.1)$$

where $C_j(t)$ are the nodal values of the concentration as a function of time, $f_j(x,z)$ are a set of basis functions, and N is the number of unknowns.

If the basis functions are substituted into the original partial differential equation the result would be that a residual will exist. This residual would represent the error due to the approximate solution not being the exact solution.

The residual is weighted to a local domain specific to each element by multiplying the residual by a set of weight or test functions. This weight residual is then integrated over the domain to represent an average residual over the whole domain. The resulting integral can be stated as,

$$\int_{\Omega} v_i(x,z)\ L(C)\ d\Omega = \text{Global Residual} \qquad (3.2)$$

where $L(C)$ is the partial differential equation for the discrete representation given by 3.1, $v_i(x,z)$ are the test functions and Ω is the domain of the problem. The equations needed to solve for the discrete solution C_j are generated by forcing the Global Residual to be zero.

The advantage of such a formulation is its ability to reduce the restrictions on the basis functions. The original differential equation would generally have a second derivative. This would imply that for the basis functions to represent the solution, even over a single element, they would have to have continuous first derivative, meaning that the second derivative exists. This restriction would eliminate the use of simple linear functions, since they have a discontinuous first derivative. However with the use of a simplified weak statement, this restriction can be relaxed so as to allow broad range of possible functions.

The resulting integral equation 3.2 can be simplified by using the Green-Gauss theorem to remove this restrictions. The general Green-Gauss theorem can be stated as,

$$\int_{\Omega} \beta_i \frac{\partial \omega_k}{\partial x_j} d\Omega = \int_{\Gamma} \beta_i (\omega_k \circ n_j)\ d\Gamma - \int_{\Omega} \frac{\partial \beta_i}{\partial x_j} \circ \omega_k\ d\Omega \qquad (3.3)$$

where β_i are a set of scalar functions, ω_k are the basis functions, n_j is the normal vector, Ω is the domain of the problem, and Γ is the boundary portion of the domain.

The resulting simplified statement is termed the weak statement since the solution restrictions of the weak statement are less than those of the original differential equation. Originally the solution domain required that a second derivative of the functional of the solution must exist. The new statement requires that the first derivative of the functional need only be square integratable. Thus the solution domain of the original equation is a sub-set of the solution domain of the weak statement.

Applying this technique to the depth averaged concentration equation initially results in the integral equation shown below,

$$\iint_{\Omega_{x_i}} \left(\overset{(I)}{v_i \frac{\partial C}{\partial t}} + \overset{(II)}{v_i U_i \frac{\partial C}{\partial x_i}} - \overset{(III)}{v_i \frac{\partial}{\partial x_i} E_{ij} \frac{\partial C}{\partial x_j}} - \overset{(IV)}{v_i P} \right) dx_i = 0. \qquad (3.4)$$

where v_i are the test functions, and Ω_{x_i} is the domain in the two directions. By applying the Green-Gauss theorem to the third term, it can be simplified to the equation shown below,

$$-\iint_{\Omega_{x_i}} v_i \frac{\partial}{\partial x_i} E_{ij} \frac{\partial C}{\partial x_j}\ dx_i = -\int_{\Gamma_{x_i}} v_i \left(E_{ij} \frac{\partial C}{\partial x_j} \circ n_j\right)\ d\Gamma$$

$$+ \iint_{\Omega_{x_i}} \frac{\partial v_i}{\partial x_j} \circ E_{ij} \frac{\partial C}{\partial x_j} \, dx_i \quad . \qquad (3.5)$$

The boundary integral in the above equation is termed the natural boundary condition. The reason it is natural is that the flux boundary conditions can be incorporated in the original weak statement automatically. Thus there is no need to use any extra formulation for flux boundaries.

The simplified weak statement of Equation 2.16 is

$$\iint_{\Omega_{x_i}} \left(v_i \frac{\partial C}{\partial t} + v_i \, U_i \frac{\partial C}{\partial x_i} + \frac{\partial v_i}{\partial x_j} \circ E_{ij} \frac{\partial C}{\partial x_i} - v_i \, P \right) dx_i$$

$$- \int_{\Gamma_{x_i}} v_i \left(E_{ij} \frac{\partial C}{\partial x_i} \circ n_j \right) d\Gamma = 0 \qquad (3.6)$$

3.1.2 Semi-Discrete Form

The discrete representation expressed by the nodal values and the basis functions are substituted into the weak statement to derive the semi-discrete form of the equation shown below,

$$\int_{\Omega_x} \int_{\Omega_z} \Bigg[v_i \, f_j \frac{\partial C_j}{\partial t} + v_i \left(C_j \, U_j \frac{\partial f_j}{\partial x} + C_j \, V_j \frac{\partial f_j}{\partial z} \right)$$

$$+ \frac{\partial v_i}{\partial x} \left((E_{xx})_j C_j \frac{\partial f_j}{\partial x} + (E_{xz} \qquad \frac{\partial f_j}{\partial z} \right)$$

$$+ \frac{\partial v_i}{\partial z} \left((E_{zx})_j C_j \frac{\partial f_j}{\partial x} + (E_{zz})_j C_j \frac{\partial f_j}{\partial z} \right) + v_i \, P_j \Bigg] \, dz dx . \qquad (3.7)$$

This statement can be further simplified by performing the integral in two parts. The first integrate over each element, then to sum the element integrals over the whole domain. Each integral in the statement has an associated matrix as classified below:

the mass matrix:

$$[M] = \int_{\Omega_x}\int_{\Omega_z} v_i \, f_j \, dz dx;$$

the stiffness matrix:

$$[K] = \int_{\Omega_x}\int_{\Omega_z} v_i \left(U_j \frac{\partial f_j}{\partial x} + V_j \frac{\partial f_j}{\partial z}\right) + \frac{\partial v_i}{\partial x}\left((E_{xx})_j \frac{\partial f_j}{\partial x} + (E_{xz})_j \frac{\partial f_j}{\partial z} \right)$$

$$+ \frac{\partial v_i}{\partial z}\left((E_{zx})_j \frac{\partial f_j}{\partial x} + (E_{zz})_j \frac{\partial f_j}{\partial z} \right) dz dx \;;$$

and the force matrix: $[F] = \int_{\Omega_x}\int_{\Omega_z} v_i \, P_j \, dz dx.$

Thus the semi-discrete form can be simply expressed as

$$[M]\left\{\frac{dC_j}{dt}\right\} + [K]\left\{C_j\right\} + [F] = \{0\}. \qquad (3.8)$$

3.1.3 Coordinate Transformation

The finite element method has the advantage of being able to incorporate a mapping system into its basic formulation. In a general case, any arbitrary shape of domain can be discretized into a number of elements of different shape and size. Each element in the domain has to be mapped into a simple normalized local domain of a general elements, shown in Figure 3.1.

The mapping is accomplished by using a set of parametric functions such that the coordinates can be expressed as,

$$x = \sum_{i=0}^{Nn} X_i \; g_i(r,s) \text{ and } z = \sum_{i=0}^{Nn} Z_i \; g_i(r,s).$$

In these expressions X_i and Z_i are nodal coordinates, r and s represent the local coordinate system, g_i are a set of geometric transformation functions, and Nn is the number of nodes in the element.

The present formulation uses basis functions as the geometric functions. Using the above mapping method, the transformation from global coordinate system to local system is expressed by the relationship,

$$\begin{Bmatrix} \frac{\partial f_i}{\partial r} \\ \frac{\partial f_i}{\partial s} \end{Bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial f_j}{\partial r} X_j & \frac{\partial f_j}{\partial r} Z_j \\ \frac{\partial f_j}{\partial s} X_j & \frac{\partial f_j}{\partial s} Z_j \end{bmatrix}}_{[J]} \begin{Bmatrix} \frac{\partial f_i}{\partial x} \\ \frac{\partial f_i}{\partial z} \end{Bmatrix} \qquad (3.9)$$

However the transformation needed is usually the reverse, that is from local to global, which can be determined by the transformation expression shown below,

$$\begin{Bmatrix} \frac{\partial f_i}{\partial x} \\ \frac{\partial f_i}{\partial z} \end{Bmatrix} = [J]^{-1} \begin{Bmatrix} \frac{\partial f_i}{\partial r} \\ \frac{\partial f_i}{\partial s} \end{Bmatrix} \qquad (3.10)$$

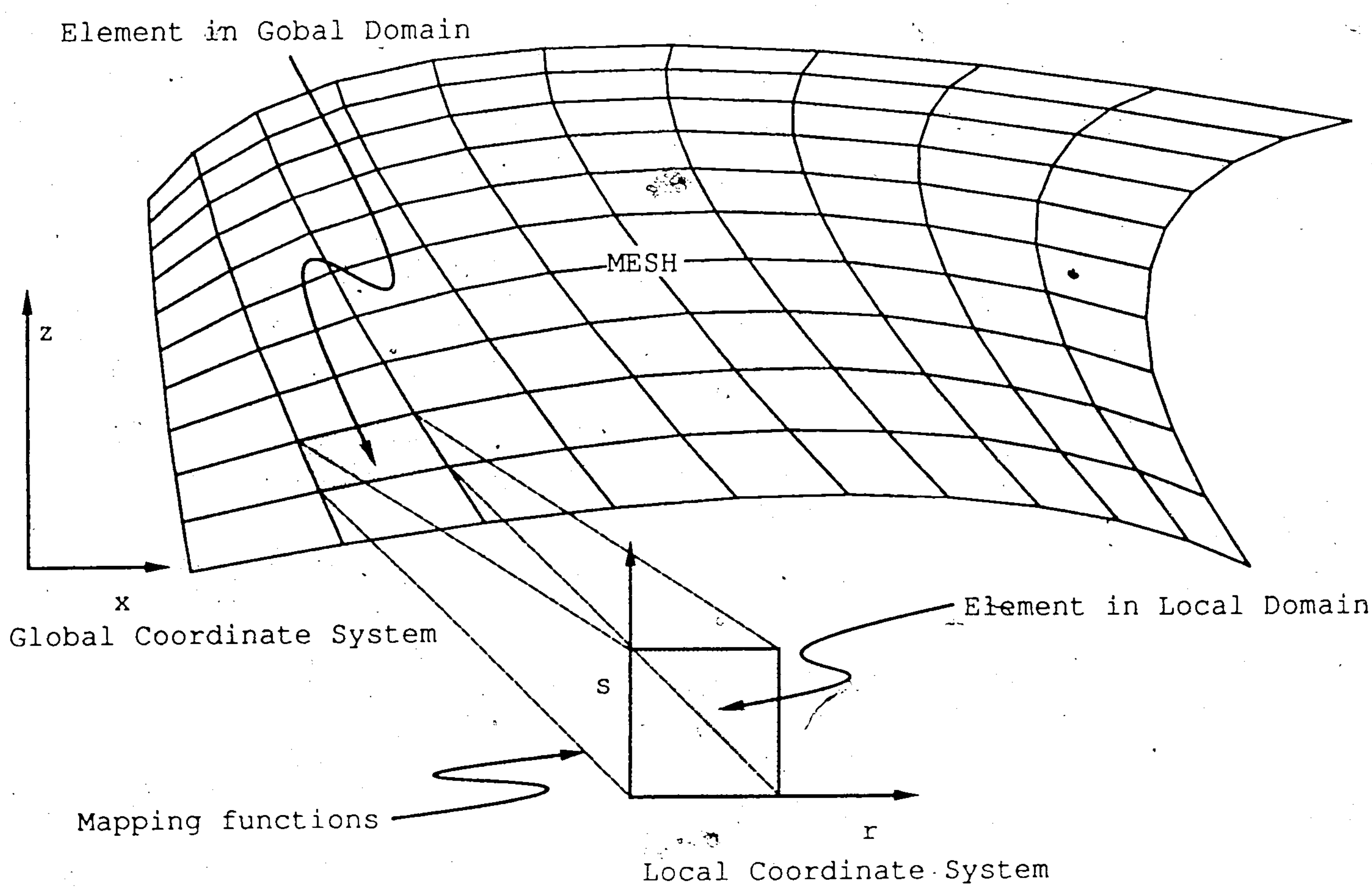


Figure 3.1 Coordinate Transformation Using Mapping Functions

3.1.4 Time Discretization

There are a number of different methods available to discretize the unsteady term in the partial differential equation. The most commonly used is the θ implicit method.

The θ implicit method applied to Equation 3.8 results in the discrete equation shown below,

$$[M]\left\{C_i^{n+1}\right\} - [M]\left\{C_i^{n}\right\} +$$

$$\theta\Delta t \ [K]\left\{C_i^{n+1}\right\} + (1-\theta)\Delta t \ [K]\left\{C_i^{n}\right\} + \Delta t\{F\} = \{0\} \qquad (3.11)$$

3.1.5 Boundary Conditions

The current thesis incorporates in its formulation a general method whereby a variety of boundary condition can be specified on any arbitrary boundary.

To incorporate any general boundary condition the boundary integral resulting from the integration by parts of the second derivative was substituted with an arbitrary penalty expression, as show below,

$$\frac{\partial C}{\partial n} = A\ C + B\ . \qquad (3.12)$$

By adjusting the values of the coefficients A and B, one can provide for a fixed flux boundary condition, fixed value boundary condition, or a combination of both.

The resulting expressions substituted into the weak form are

$$\int_{\Gamma} v_i C_j \frac{\partial f_j}{\partial n}\, d\Gamma = \int_{\Gamma} \left(A_j\; v_i C_j f_j - v_i\, B_j \right)\; d\Gamma = 0. \qquad (3.13)$$

In this equation *n* indicates that only the portion that is normal to the boundary of the integral survives.

3.1.6 Discrete Equation

In summary the discrete equation, for each element, coded into a program can be expressed as the following:

$$[M]\left\{C_i^{n+1} - C_i^{n}\right\} + \theta\Delta t\left([K_x] + [K_z]\right)\left\{C_i^{n+1}\right\}$$

$$+ (1-\theta)\Delta t\left([K_x] + [K_z]\right)\left\{C_i^{n}\right\} + \Delta t\;\{F\}$$

$$+ \Delta t\,[K_B]\left\{C_i^{n+1}\right\} + \Delta t\,\{F_B\} = \{0\} \qquad (3.14)$$

where,

$$[M] = \int_{\Omega_r}\int_{\Omega_s} v_i\; f_j\;\; [J]^{-1} ds dr\;;$$

$$[K_x] = \int_{\Omega_r}\int_{\Omega_s} v_i\; U_j\; [J_x]^{-1}\frac{\partial f_j}{\partial r} + [J_x]^{-1}\frac{\partial v_i}{\partial r}(E_{xx})_j[J_x]^{-1}\frac{\partial f_j}{\partial r}$$

$$+ [J_x]^{-1}\frac{\partial v_i}{\partial r}(E_{xz})_j[J_z]^{-1}\frac{\partial f_j}{\partial s}\;[J]^{-1}\; ds dr\;;$$

$$[K_z] = \int_{\Omega_r}\int_{\Omega_s} v_i\; V_j\; [J_z]^{-1}\frac{\partial f_j}{\partial s} + [J_z]^{-1}\frac{\partial v_i}{\partial s}(E_{zx})_j[J_x]^{-1}\frac{\partial f_j}{\partial r}$$

$$+ [J_z]^{-1}\frac{\partial v_i}{\partial s}(E_{zz})_j\;[J_z]^{-1}\frac{\partial f_j}{\partial s}\;[J]^{-1}\; ds dr\;;$$

$$\{F\} = \int_{\Omega_r}\int_{\Omega_s} v_i \; P_j \quad [J]^{-1} \, ds dr \; ;$$

$$[K_B] = \int_{\Gamma} A_j \; v_i f_j \; dn \quad \text{and}$$

$$\{F_B\} = \int_{\Gamma} v_i \; B_j \; dn \; .$$

In the above expressions the following convenience was used,

$$\begin{bmatrix} [J_x]^{-1} \\ [J_z]^{-1} \end{bmatrix} = \begin{bmatrix} [J_{xx}]^{-1} & [J_{xz}]^{-1} \\ [J_{zx}]^{-1} & [J_{zz}]^{-1} \end{bmatrix} \equiv [J]^{-1} \; .$$

3.1.7 The Solution Algorithm

The solution scheme for a general case of the finite element method proceeds as the following:

1) The weak statement is evaluated for each of the element using the appropriate set of functions (Equation 3.13);

2) The element matrices are assembled into a global matrix;

3) This global matrix of linear algebraic set of equations is solved using any particular method.

The global set of linear algebraic equations for the finite element formulation can be stated as,

$$[K']\left\{C_i^{n+1}\right\} = \{F'\}, \qquad (3.15)$$

in which,

$$[K'] = \sum_{i=1}^{N} \Big([M] + \theta\Delta t \left([K_x]+[K_z] \right) \Big) ,$$

$$\{F'\} = \sum_{i=1}^{N} \left((1-\theta)\,\Delta t \left([K_x]+[K_z] \right) \left\{C_i^n\right\} + \Delta t\{F\} \right)$$

$$+ \sum_{i=1}^{Nb} \left([K_B]\left\{C_i^n\right\} + \{F_B\} \right) .$$

Where N is the number of elements, and Nb is the number of boundary elements in the problem.

Though the above algorithm is well behaved, it does require a large block of memory to execute, even though Sky-lined storage scheme was used. In addition, the limits put on the available computational effort forced the addition of the alternating direction formulation discussed in some detail by Baker (Baker,1983).

3.2 Choice of Test and Basis Functions

The solution algorithm developed to this point is still incomplete since the test and basis functions have not been chosen. In the finite element method the choice of these functions have been categorized into two distinct groups. When the choice of the test and basis functions is the same then the formulation is termed the Bubnov Galerkin formulation. However if the test and basis functions are different then the method is termed a Petrov Galerkin formulation.

The Bubnov Galerkin formulation results in a square matrix, and for the particular case when linear elements are used its behavior is similar to the centered finite difference formulations.

The Petrov Galerkin formulation has generally been based on the use of higher order upwinding functions as test functions and simpler functions as basis functions. The test functions are generally developed by weighting the basis functions in an upwind manner.

Recently however a paper by Steffler (1988) presented, the use of higher order upwind functions as the basis functions. The reasoning behind this approach is that for convection dominating variables, the value at a new point should have a heavier dependence on the preceding points than the points in its surrounding. The result of this formulation is a scheme that reflects the behavior observed using the popular QUICK finite volume schemes (Leonard, 1979).

The current investigation will have two major purposes in mind. One is the judge the behavior of these new upwinding basis functions in a series of numerical tests, and the other is to observe their behavior in some simple practical problems. To under take such an investigation three different levels of accuracy were tested for the numerical tests.

3.2.1 Choice of Basis and Test Functions

The numerical tests were based on the following choices for the test and basis functions:

1) **LBG** : Linear Bubnov Galerkin,
   -uses linear test and basis function as stated below for the one dimensional case:

$$f(r) = \begin{Bmatrix} \frac{(1+r)}{2} \\ \frac{(1-r)}{2} \end{Bmatrix} \; ; \; r \in (-1,1) \tag{3.16}$$

2) **QUPG** : Quadratic Upwinding Petrov Galerkin,
   -uses quadratic basis functions as shown below for the one dimensional case, and the linear test functions 3.16,

$$f(r) = \begin{Bmatrix} \frac{(3+r)(1-r)}{4} \\ \frac{(1+r)(3+r)}{8} \\ \frac{-(1-r)(1+r)}{8} \end{Bmatrix} \; ; \; r \in (-1,1) \tag{3.17}$$

3) **CUPG** : Cubic Upwinding Petrov Galerkin,
   -uses cubic basis functions as shown below for the one dimensional case, and the linear test functions 3.16,

$$f(r) = \begin{Bmatrix} \frac{(5+r)(3+r)(1-r)}{16} \\ \frac{(1+r)(3+r)(5+r)}{48} \\ \frac{-(1-r)(1+r)(5+r)}{16} \\ \frac{-(1-r)(1+r)(3+r)}{48} \end{Bmatrix} ; \; r \in (-1,1) \qquad (3.18)$$

The surrounding elements that influence the general two dimensional element, as well as the specialized element near the boundaries, are shown for LBG, QUPG, and CUPG in Figures 3.2, 3.3 and 3.4 respectively.

3.2.2 The Program

The program was written originally by Dr. P. M. Steffler at the University of Alberta on an Apple Macintosh computer, using the C programming language. This version of the program was modified to include the solution for the depth average concentration in rivers and expanded to include the Cubic basis functions.

A general flow chart of the program is shown in Figures 3.5 to 3.9. This flow chart is meant to be a guide and aid in the use of the program in a user friendly environment. The flow chart indicates the flexibility incorporated into the program to solve problems posed in a variety of different ways. These range for the solution for plume in an idealized rectangular domain, to the laborious solution of a slug test in a general river reach.

Versions of the program were built for the Amdahl mainframe, IBM PC and Apple Macintosh machines. Two specialized versions of the program were developed, one general enough to use on any machine, and another based solely on the user friendly environment of the Apple Macintosh machine.

A listing of the general program is also included in Appendix 1. The program was written in the C language due to its ability to allow dynamic allocation of memory.

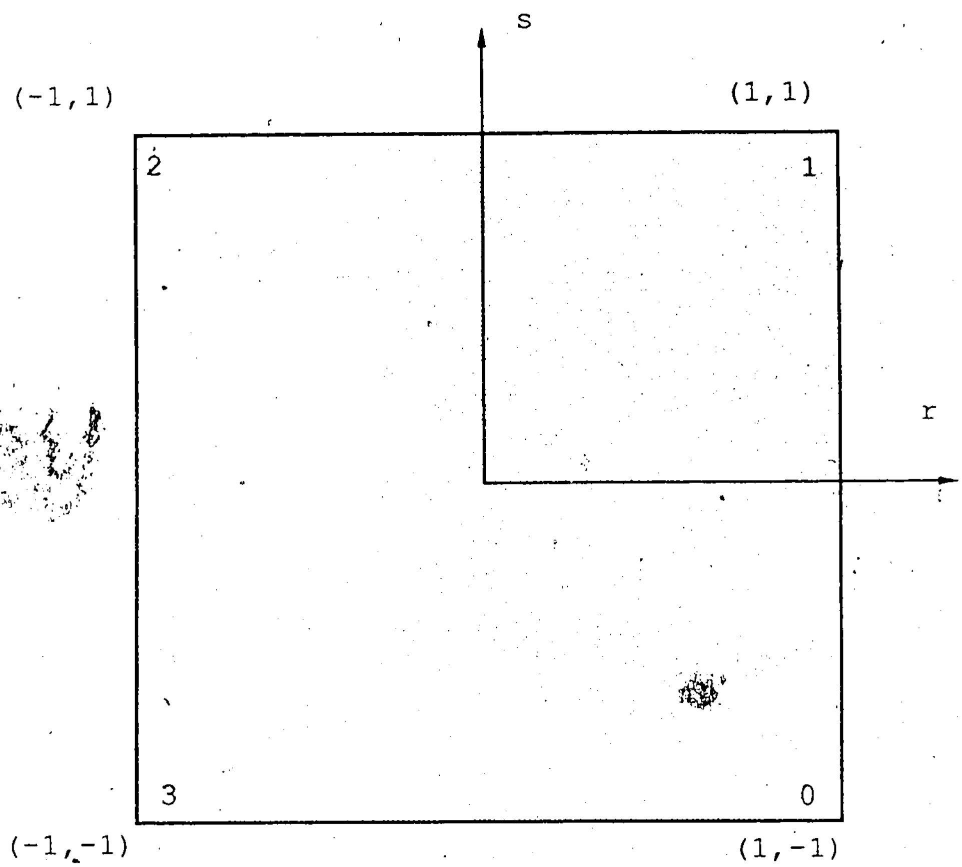
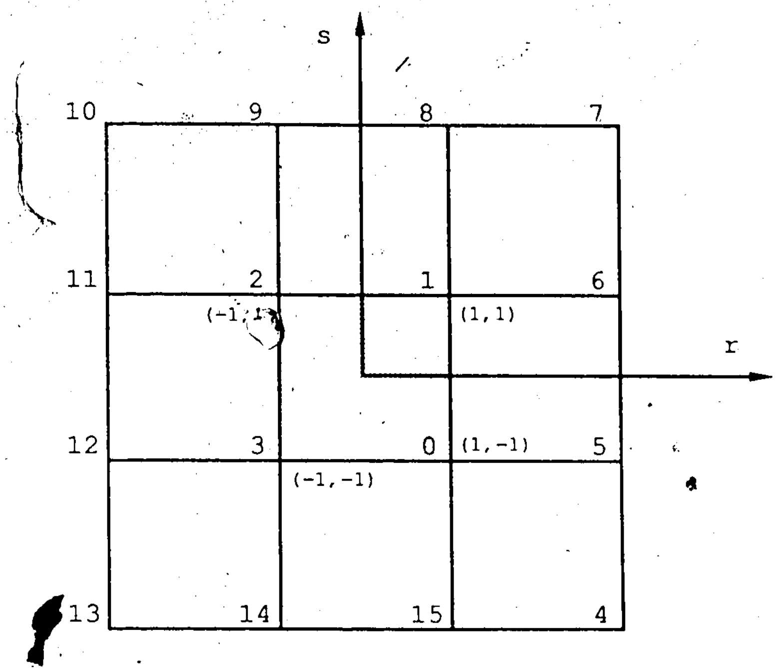
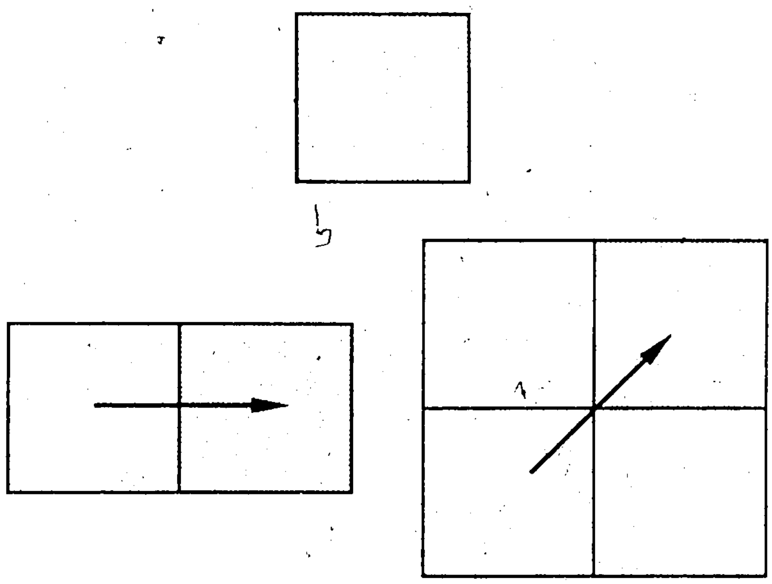


Figure 3.2 Local Domain of the Linear Rectangular Element

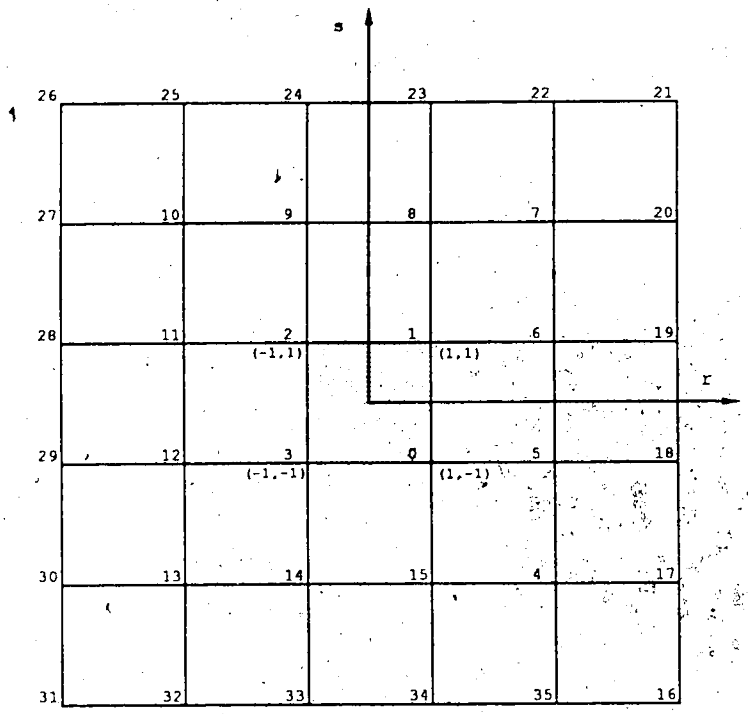


General area of influence for an element using Quadratic Upwinding Elements

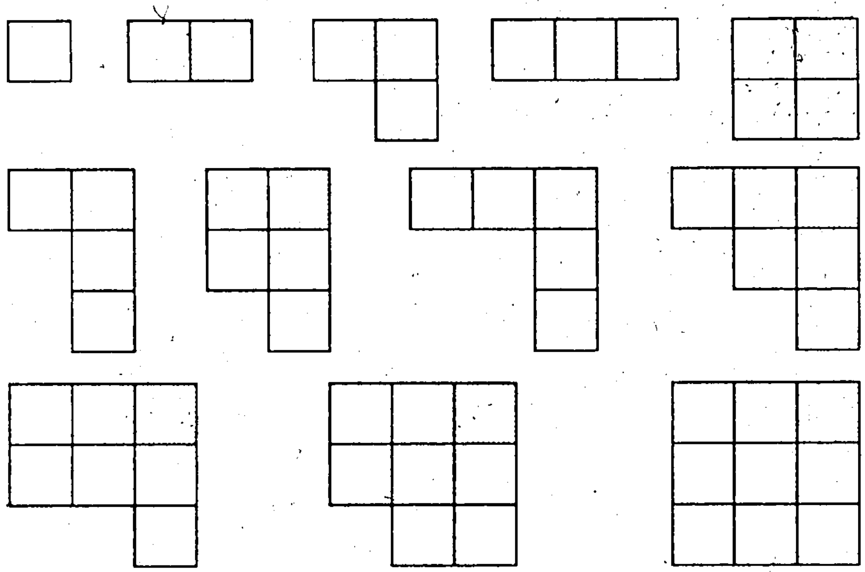


Subset of Possible Element Types for Quadratic Upwinding Elements

Figure 3.3 Local Domain of the Quadratic Upwinding Rectangular Element



General area of influence for an element using Cubic Upwinding Elements



Subset of Possible Element Types for Cubic Upwinding Elements

Figure 3.4 Local Domain of the Cubic Upwinding Rectangular Element

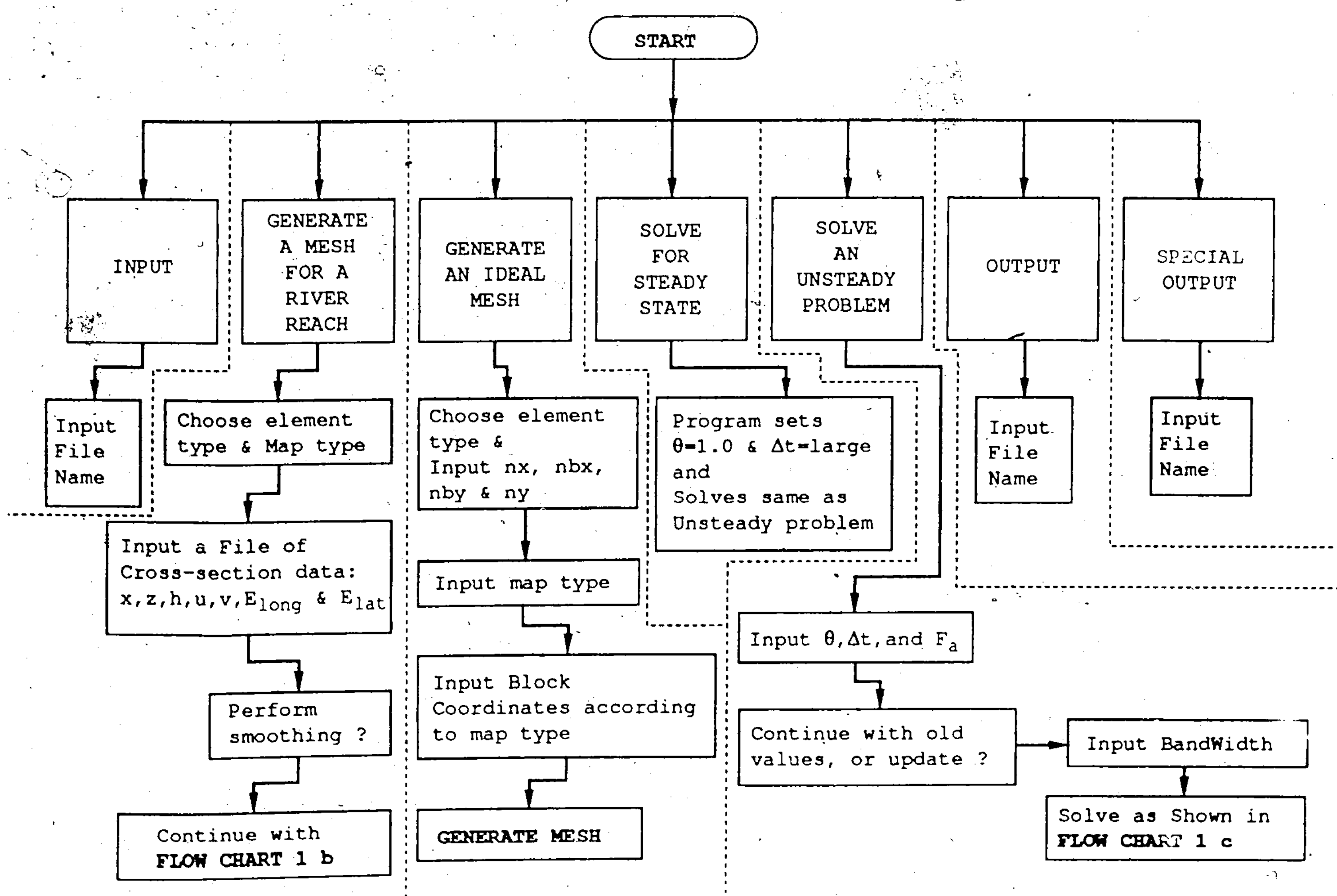


Figure 3.5 Flow Chart 1 a, A General Flow Chart of the Program CDFEM

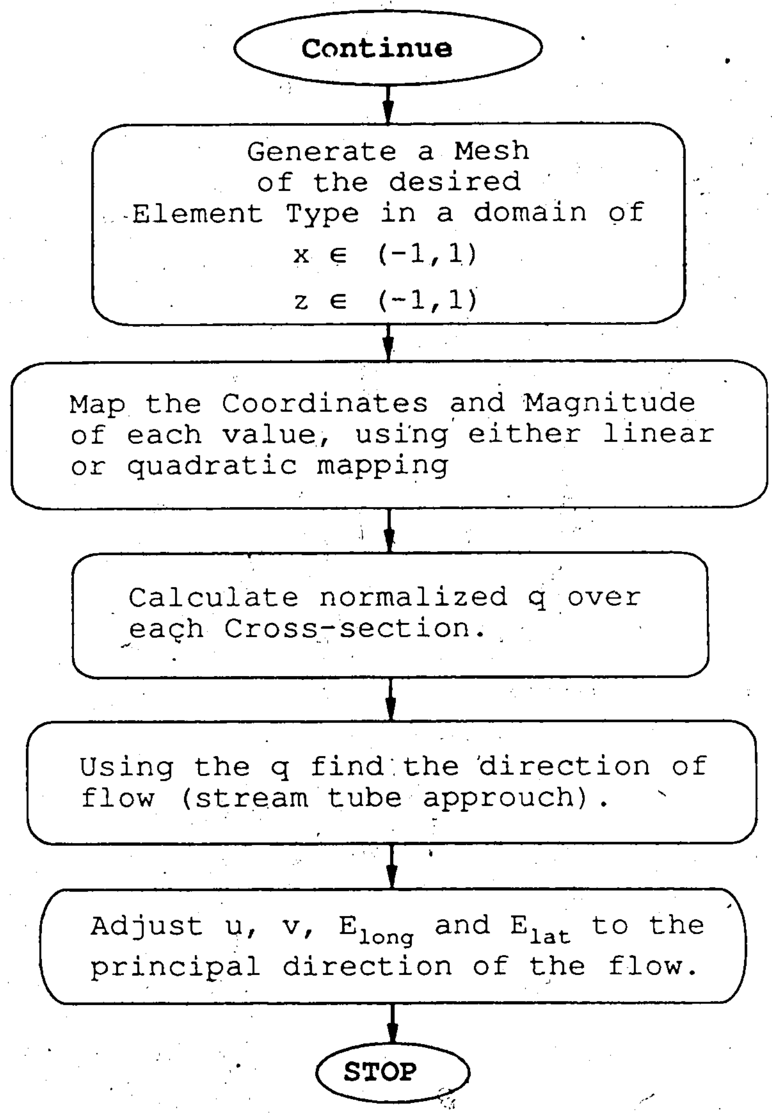


Figure 3.6 Flow Chart 1 b, Generate a Mesh fitted to a River Reach

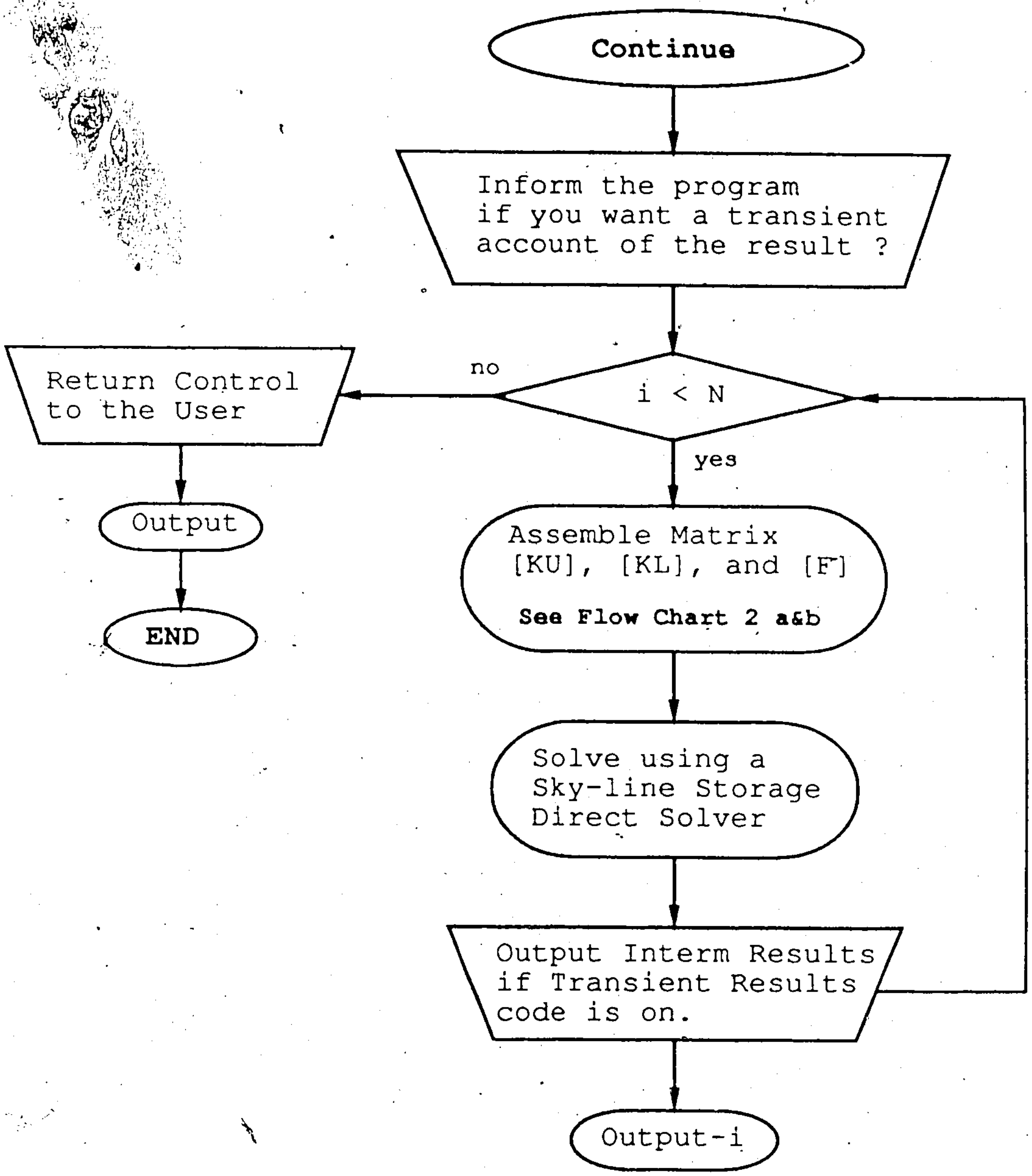


Figure 3.7 Flow Chart 1 c, The Programmed Solution Algorithm

Flow Chart 2a. Assembling of Global Matrices Part I

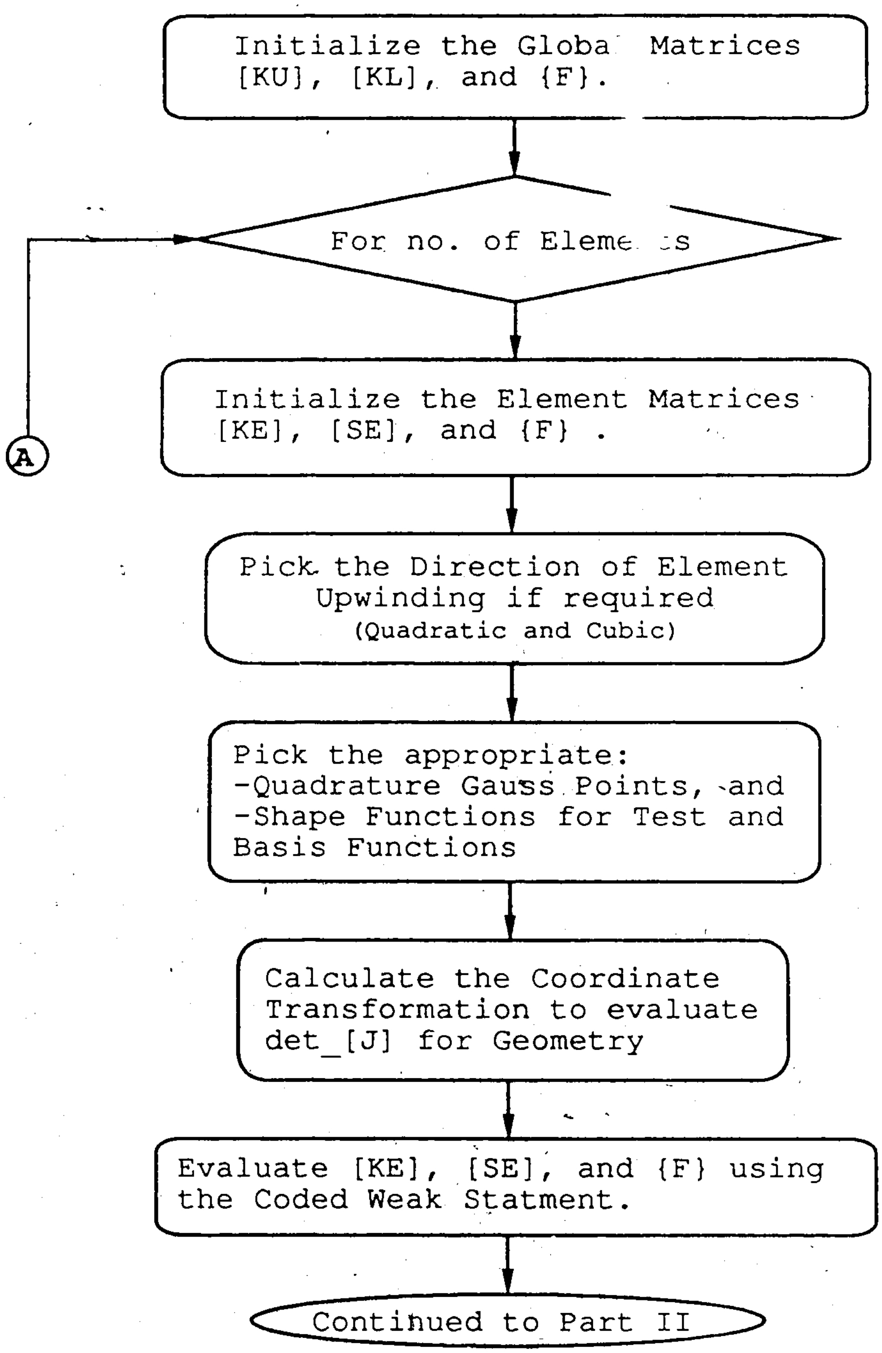


Figure 3.8 Flow Chart 2 a, Assembling of Global Matrices, Part I

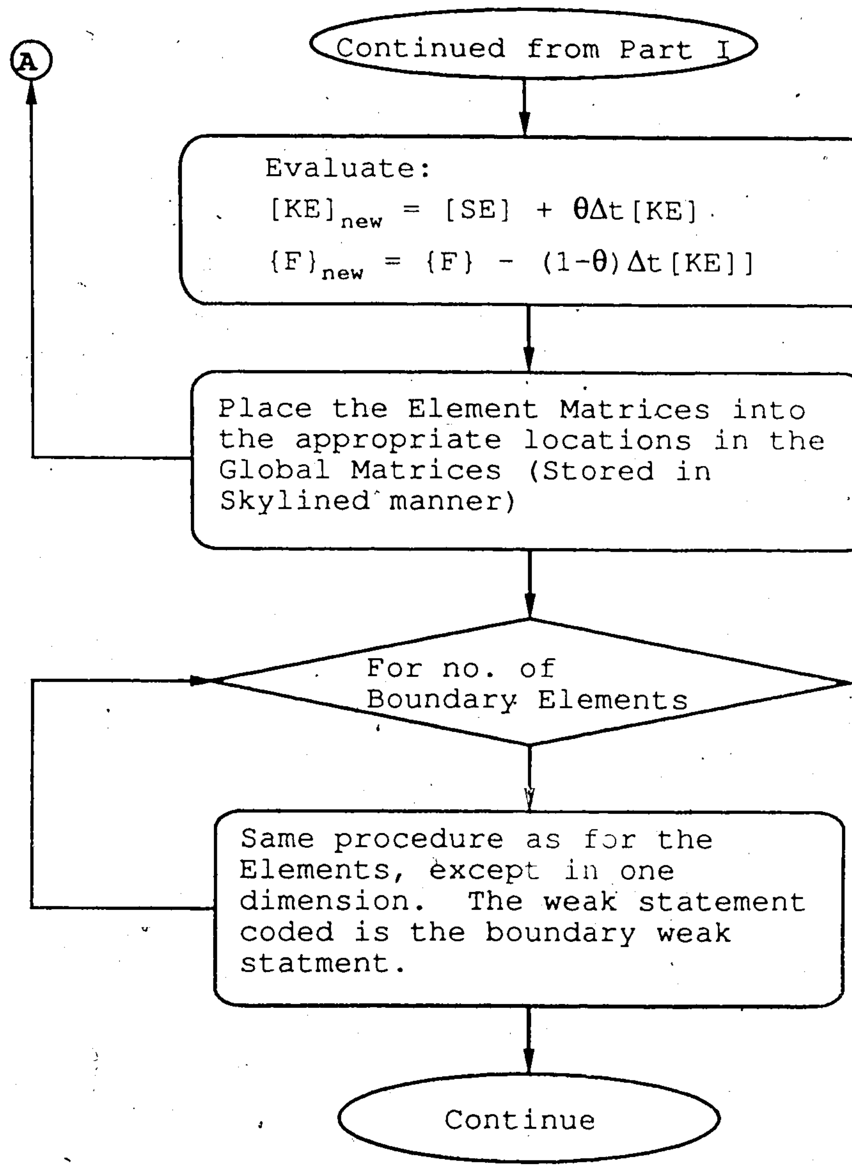


Figure 3.9 Flow Chart 2 b, Assembling of Global Matrices, Part II

4. NUMERICAL TESTS

The performance of the upwinding basis functions introduced earlier has not yet been fully evaluated. The use of the these new elements was evaluated by performing a set of standard numerical tests. A comparison is presented in this chapter between the upwinding basis functions elements and other commonly used methods, in the analysis of pollutant discharges in rivers.

4.1 One Dimensional Comparison

The first comparison performed on these new elements was to test their behavior against the finite difference and finite element methods mentioned in Section 2.2.2, using an one dimensional test.

One of the most important characteristic desired of any numerical method built to solve the advection diffusion equation is its ability to accurately predict the behavior of advection dominated flows. A problem that will test this behavior is the solution of the unsteady advection Equation 2.28. All three methods formulated here were tested to determine their behavior in the solution of this equation.

4.1.1 LBG

The use of linear test and basis functions in the solution of Equation 2.28, results in the discrete equation shown below,

$$\left(\frac{1}{6} - \frac{\theta\rho}{2}\right)C_{i-1}^{n+1} + \frac{2}{3}C_i^{n+1} + \left(\frac{1}{6} + \frac{\theta\rho}{2}\right)C_{i+1}^{n+1}$$

$$= \left(\frac{1}{6} + \frac{(1-\theta)\rho}{2}\right)C_{i-1}^{n} + \frac{2}{3}C_i^{n} + \left(\frac{1}{6} - \frac{(1-\theta)\rho}{2}\right)C_{i+1}^{n} \quad . \qquad (4.1)$$

The resulting Taylor series expansion of this discrete equation for $\theta = 0.5$ is

$$\frac{\partial c}{\partial t} + U\frac{\partial c}{\partial x} = O(\Delta x)^2 + O(\Delta t)^2. \qquad (4.2)$$

The Stone and Brian method discussed earlier in Section 2.2.2.2.1 is identical to this case when $\theta = 0.5$.

The dispersion and dissipation diagrams for this method are shown in Figure 4.1. It is apparent from the diagrams as well as from the Taylor Series expansion that this method shows no dissipation, but does show dispersion for the shorter wavelength perturbations. These small perturbations would persist in the solution, since there is no selective dissipation of them. The result of such an solution algorithm is the realization of oscillatory solutions.

This is shown in Figure 4.2, which is the solution for $\rho = 0.24$, of the advection of a Gaussian shaped pulse.

4.1.2 QUPG

The use of Quadratic basis functions and linear test functions results in the following discrete equation,

$$\left(\frac{-1}{24} + \frac{\theta\rho}{12}\right)C_{i-2}^{n+1} + \left(\frac{5}{24} - \frac{3\theta\rho}{4}\right)C_{i-1}^{n+1} + \left(\frac{17}{24} + \frac{\theta\rho}{4}\right)C_{i}^{n+1} + \left(\frac{1}{8} + \frac{5\theta\rho}{12}\right)C_{i+1}^{n+1}$$

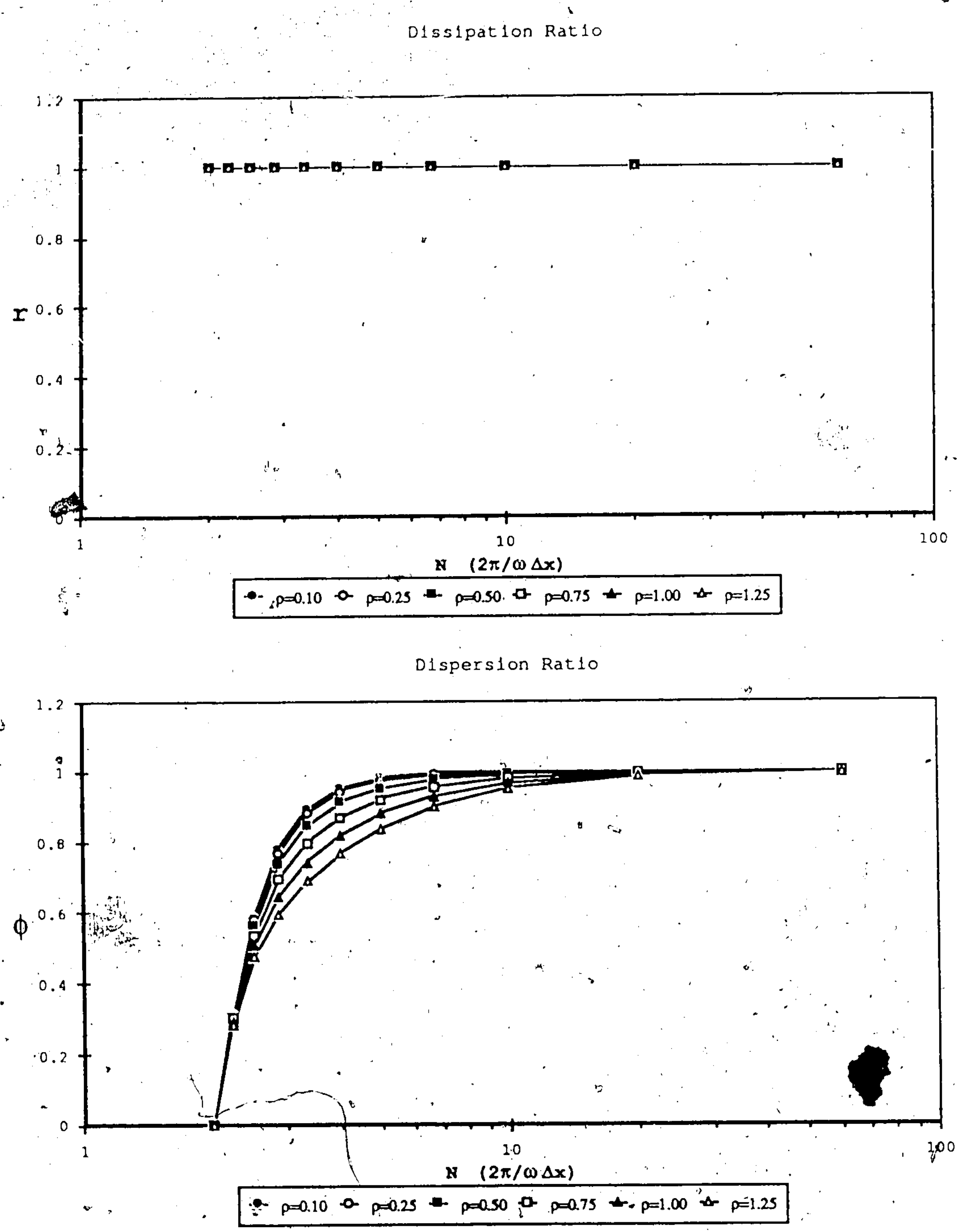


Figure 4.1 Dissipation and Dispersion for LBG

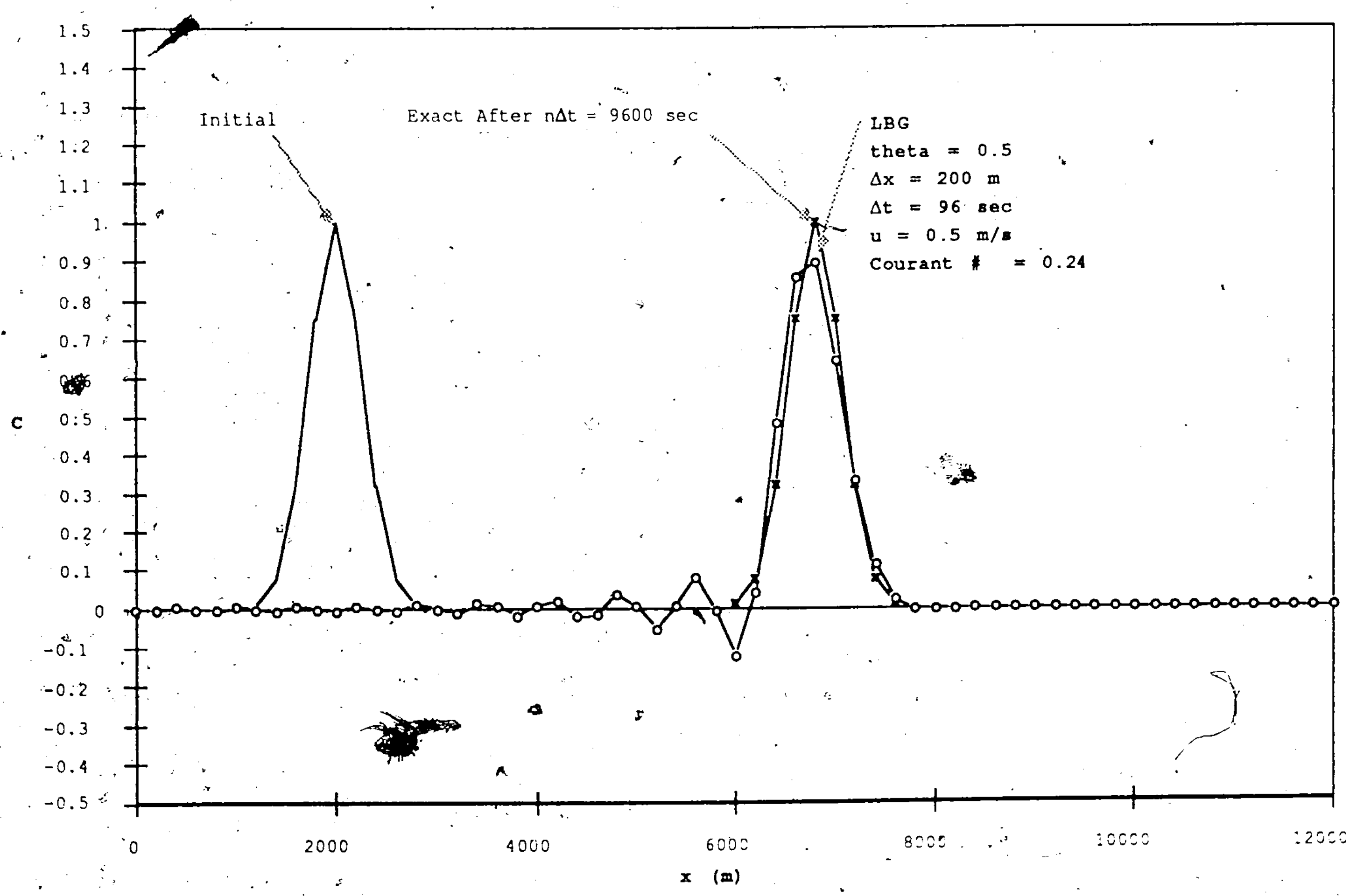


Figure 4.2 Advection of a Pulse using LBG

$$= \left(\frac{-1}{24} - \frac{(1-\theta)\rho}{12}\right)C_{i-2}^n + \left(\frac{5}{24} + \frac{3(1-\theta)\rho}{4}\right)C_{i-1}^n + \left(\frac{17}{24} - \frac{(1-\theta)\rho}{4}\right)C_i^n$$

$$+ \left(\frac{1}{8} - \frac{5(1-\theta)\rho}{12}\right)C_{i+1}^n \ . \qquad (4.3)$$

The modelled equation by this discrete equation is

$$\frac{\partial C}{\partial t} + U\frac{\partial C}{\partial x} + \frac{\Delta x^2}{12}\frac{\partial^2}{\partial x^2}\left(\frac{\partial C}{\partial t} + U\frac{\partial C}{\partial x}\right) + \frac{\Delta x^3}{24}\frac{\partial^3}{\partial x^3}\left(\frac{\partial C}{\partial t} + U\frac{\partial C}{\partial x}\right)$$

$$- \frac{\Delta x^4}{72}\frac{\partial^4}{\partial x^4}\left(\frac{\partial C}{\partial t} + U\frac{\partial C}{\partial x}\right) + \frac{\Delta x^4}{720}U\frac{\partial^5 C}{\partial x^5} + O(\Delta x^5) = 0, \qquad (4.4)$$

which can be seen to be fourth order accurate. The expansion of this equation would shows that no diffusion (that is the second derivative) is introduced by this equation. However there is some selective dissipation due to the higher order terms. The dispersion and dissipation diagrams for this method are shown in Figure 4.3. These diagrams show that though there is dispersion shown by the method, this dispersion is dissipated away. This implies that though the oscillation are present they disappear with time .

This is shown in the example solution of the advection of a Gaussian shape pulse at $\rho = 0.24$, illustrated in Figure 4.4. This solution shows the presence of a small wiggle trailing the pulse and the peak value is shown to be dissipated. However there is not the large amplified oscillations seen in the LBG method, as these are dissipated away as soon as they appear.

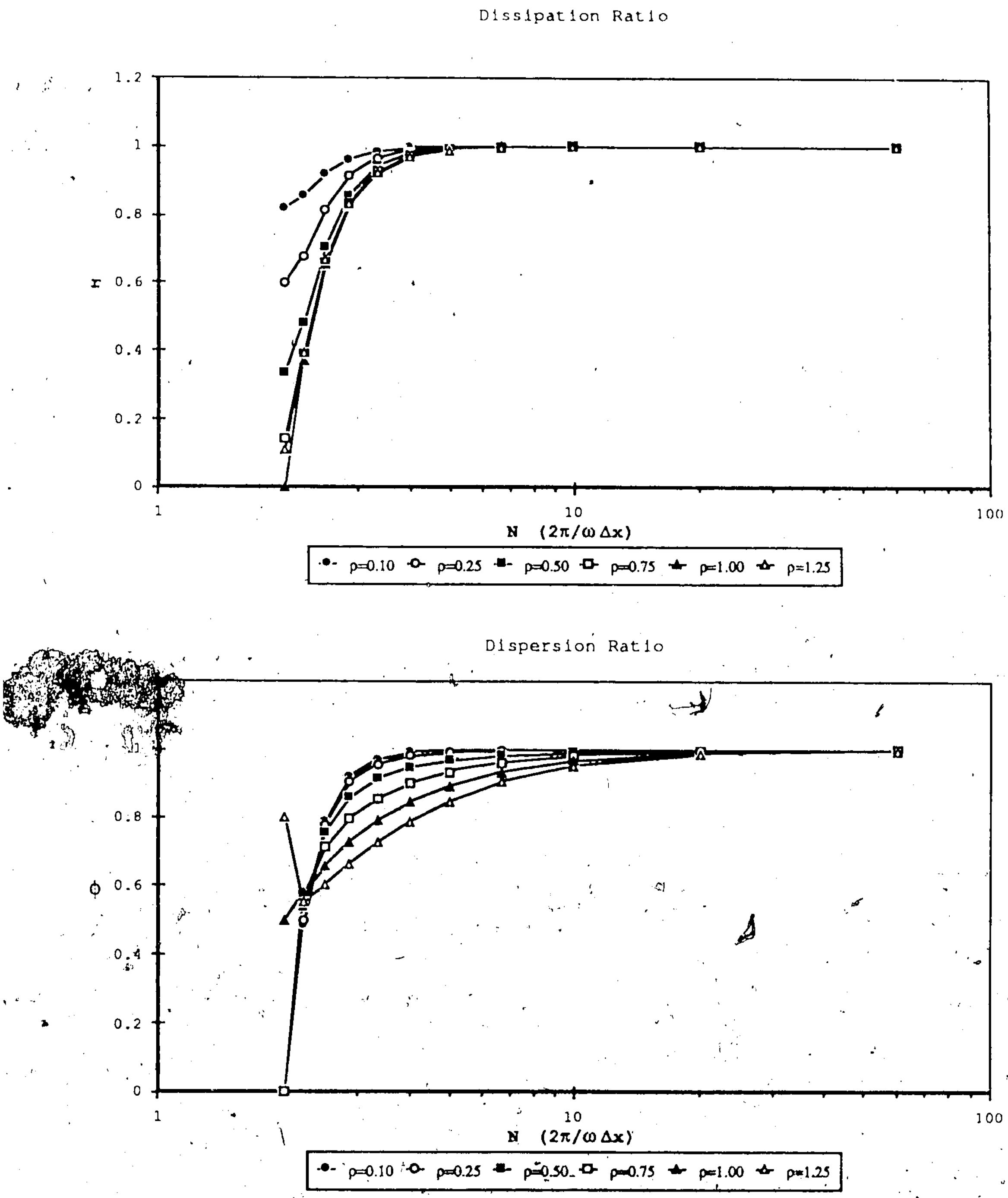


Figure 4.3 Dissipation and Dispersion for QUPG

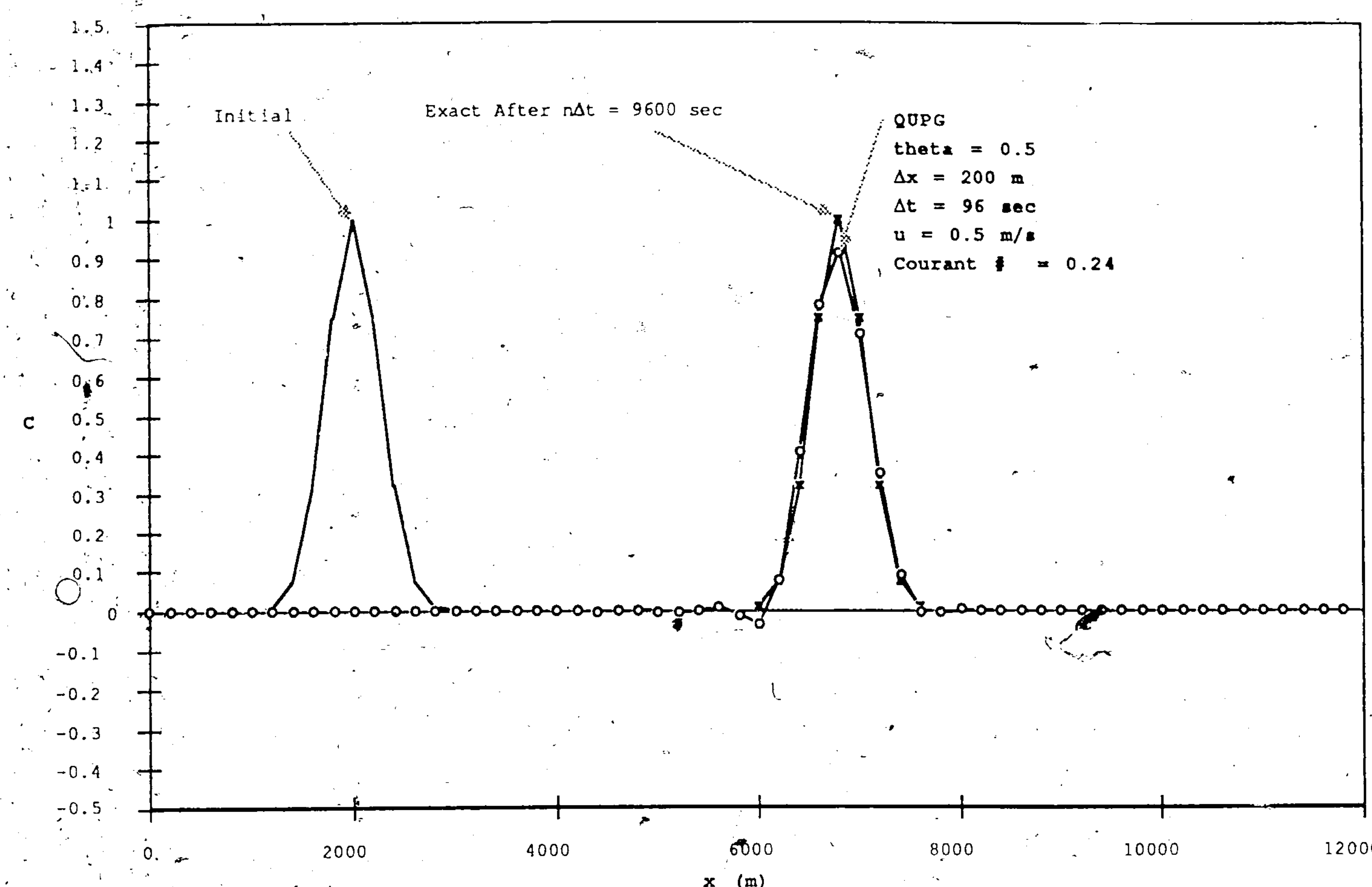


Figure 4.4 Advection of a Pulse using QUPG

4.1.3 CUPG

The discrete solution to Equation 2.10 at a particular node using Cubic basis functions and Linear test functions is represented by the equation,

$$\left(\frac{1}{45} - \frac{\theta\rho}{24}\right)C_{i-3}^{n+1} + \left(\frac{-4}{45} + \frac{\theta\rho}{4}\right)C_{i-2}^{n+1} + \left(\frac{13}{60} - \theta\rho\right)C_{i-1}^{n+1} + \left(\frac{67}{90} + \frac{5\theta\rho}{12}\right)C_{i}^{n+1}$$

$$+ \left(\frac{19}{180} + \frac{3\theta\rho}{8}\right)C_{i+1}^{n+1} = \left(\frac{1}{45} + \frac{(1-\theta)\rho}{24}\right)C_{i-3}^{n} + \left(\frac{-4}{45} - \frac{(1-\theta)\rho}{4}\right)C_{i-2}^{n}$$

$$+ \left(\frac{13}{60} + (1-\theta)\rho\right)C_{i-1}^{n} + \left(\frac{67}{90} - \frac{5(1-\theta)\rho}{12}\right)C_{i}^{n}$$

$$+ \left(\frac{19}{180} - \frac{3(1-\theta)\rho}{8}\right)C_{i+1}^{n} \qquad (4.5)$$

The modelled equation by this discrete equation is

$$\frac{\partial C}{\partial t} + U\frac{\partial C}{\partial x} + \frac{\Delta x^2}{12}\frac{\partial^2}{\partial x^2}\left(\frac{\partial C}{\partial t} + U\frac{\partial C}{\partial x}\right) + \frac{7\Delta x^4}{240}\frac{\partial^4}{\partial x^4}\left(\frac{\partial C}{\partial t} + U\frac{\partial C}{\partial x}\right)$$

$$- \frac{\Delta x^5}{45}\frac{\partial^5}{\partial x^5}\left(\frac{\partial C}{\partial t} + U\frac{\partial C}{\partial x}\right) + \frac{\Delta x^5}{720}\,U\,\frac{\partial^5 C}{\partial x^5} + O(\Delta x^6) = 0, \qquad (4.6)$$

which can be seen to be fifth order accurate. Again the expansion of this equation shows no diffusion and little dispersion. The dispersion and dissipation diagrams for this method are shown in Figure 4.5, which shows smaller dispersion and dissipation rates at the lower wavelengths than those by QUPG. A peculiarity of CUPG is that there appears to be an optimum value for the Courant number which would give the almost no dispersion ($0.25 < \rho < 0.5$). This shows an ability of the elements to accurately produce the exact solution at such a Courant number. One can also

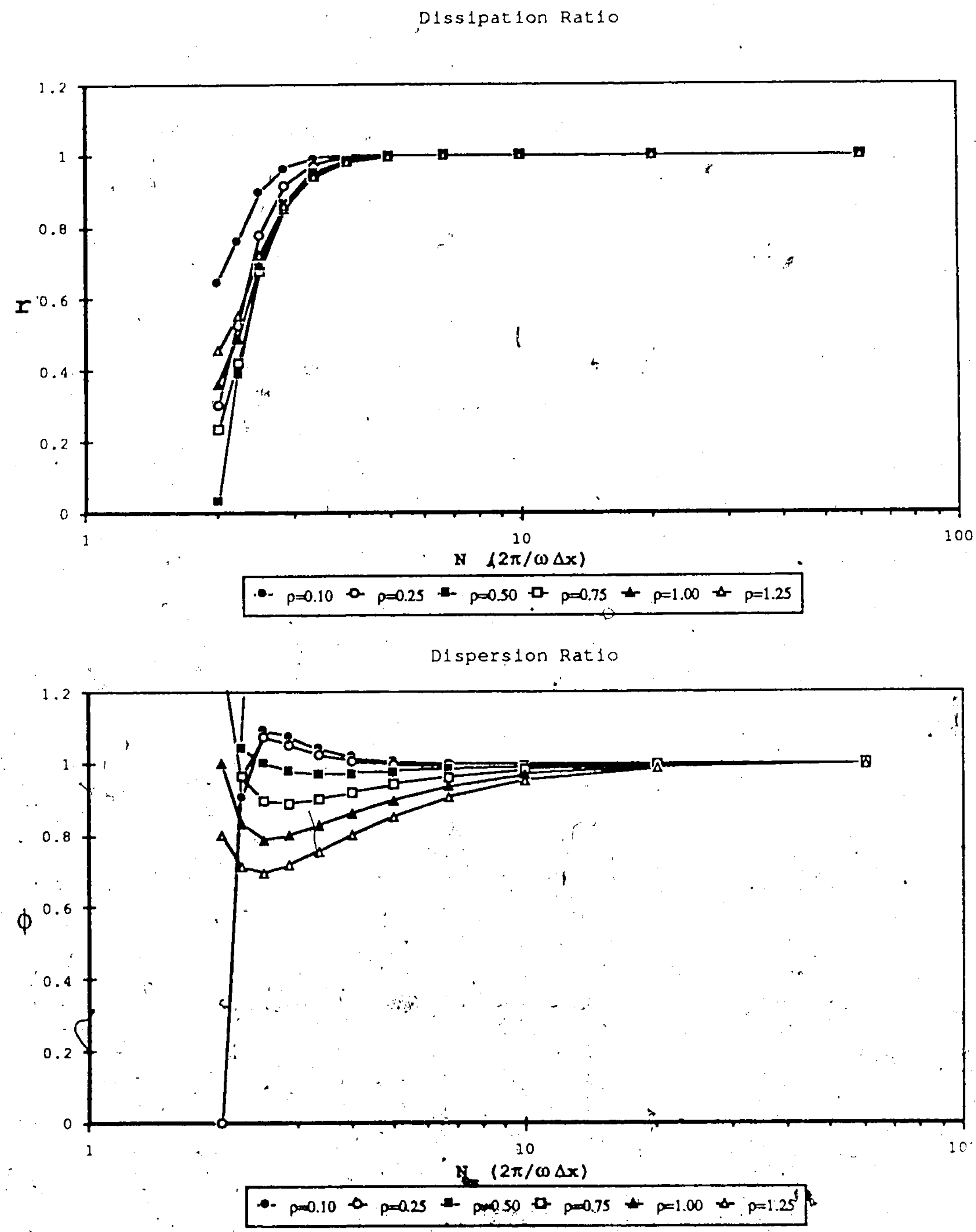


Figure 4.5 Dissipation and Dispersion for CUPG

observe that at Courant numbers less than 1.0, the small wavelength perturbations will be advected at phase speeds greater than exact. This implies the presence of wiggles ahead of the main pulse.

A plot of the solution for the advection of a Gaussian shaped pulse is shown in Figure 4.6 for $\rho = 0.24$. This illustration shows a peak value higher than the QUPG and the presence of a wiggle trailing the pulse, which is again smaller than that for QUPG. It also show the presence of a wiggle in front of the pulse due to the dispersion mentioned earlier.

An overall comparison of the these three and the four finite difference methods is shown in Figure 4.7. The indicators used to judge the behavior of the method are the L2 discrete norm, the maximum value, and the minimum value, each indicating the accuracy the dissipation and the dispersion respectively. The discrete L2 is defined as

$$L2' = \frac{\sum (C_i - C_{exact})^2}{\text{Total Mass}}, \tag{4.7}$$

which has a value of zero for an exact solution. The maximum value for the exact solution should be 1.0 and the minimum should be zero.

The semi-log bar chart of the L2' shows clearly that the best finite difference scheme at the Courant number of 0.24 is the Holly-Preissmann method (Sauvaget, 1985). While the

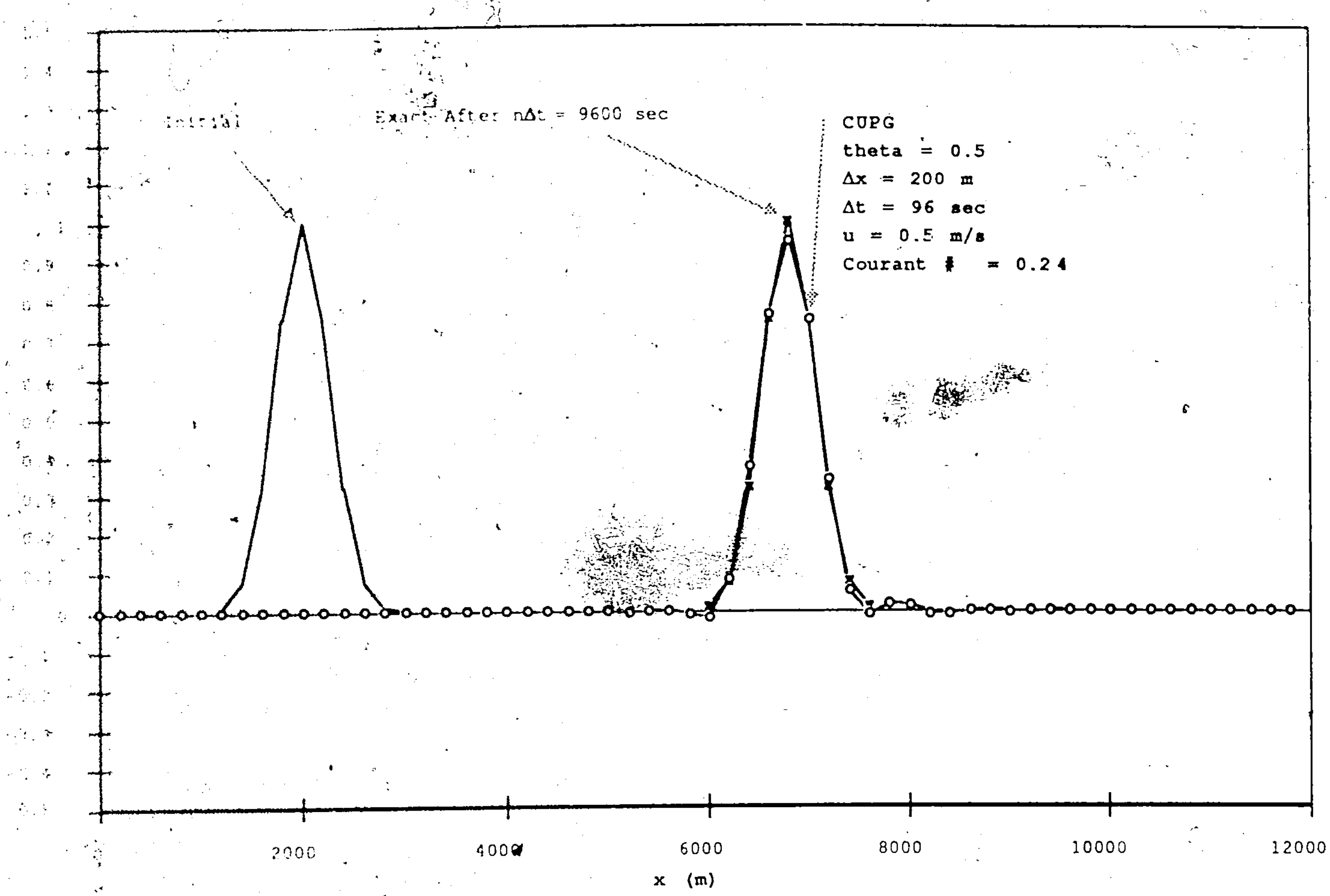
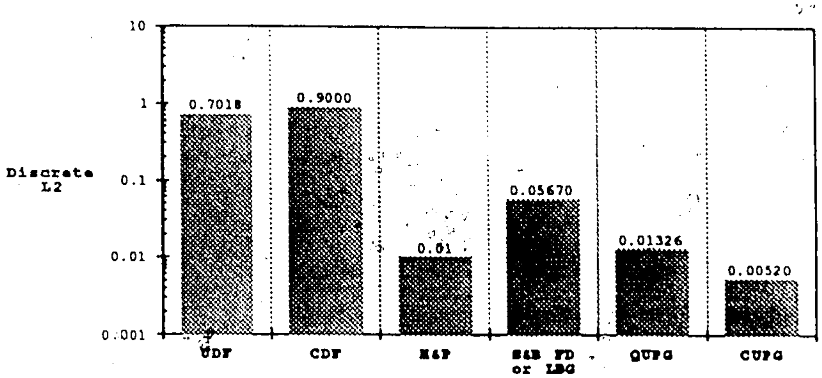
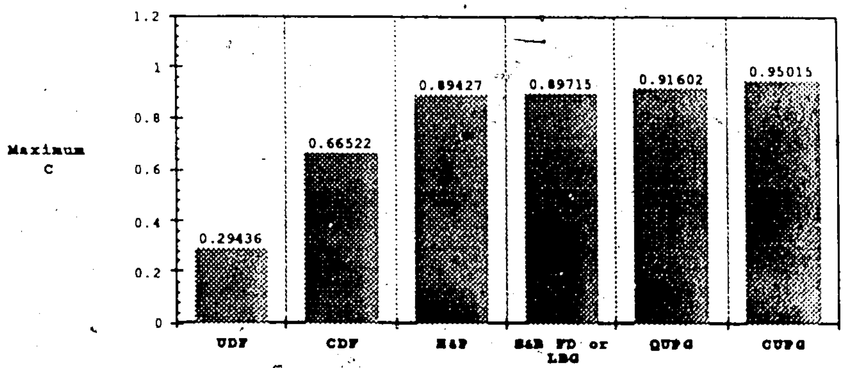


Figure 4.6 Advection of a Pulse using CUPG

Variation of the Discrete L2 norm (exact =0.0)
using different Methods Courant# = 0.24



Variation of Maximum Value (exact = 1.0)
using different Methods Courant# = 0.24



Variation of Minimum Value (exact = 0.0)
using different Methods Courant# = 0.24

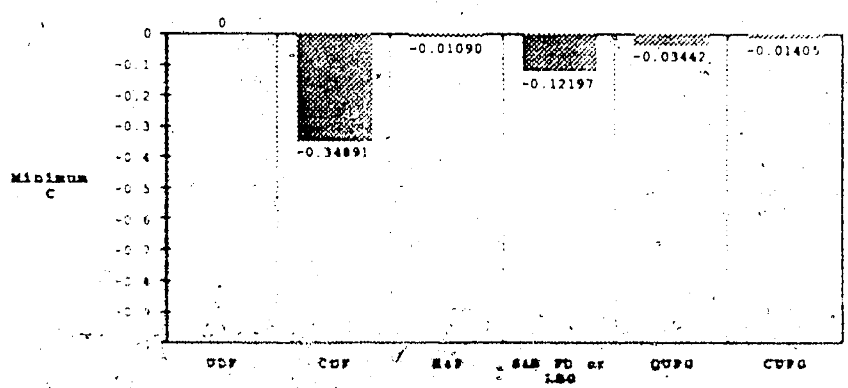


Figure 4.7 Comparison of One D tests

best finite element method is CUPG. The most accurate all of the methods tested is CUPG.

A similar conclusion can be derived for the charts of the maximum values. While for the minimum values, are the upwind finite difference method is the best, due to its artificial diffusion.

It should be noted that though a Courant number of 0.24 is representative of the type of value used in a general case, it does not represent the optimum choice for any particular method. However to model any general problem, which could conceivably have a range of different Courant numbers, the test can't be restricted to special Courant numbers. As a conclusion it is evident from the this analysis that the finite element method gives by far the best results in the one dimensional case.

4.2 Two Dimensional Tests

The previous analyses showed that the finite element method gives the best results in the one dimensional cases and since the scope of this thesis is to apply the finite element method to river problems, the remainder of the thesis will restrict itself to the evaluation of the two dimensional finite element method. Two two dimensional numerical test were performed to judge the advantages gained from new element types. The first was to look at the behavior of the new elements in simple advection equation in two dimensions. The second consisted of looking at the steady state behavior of a plume in skewed meshes.

4.2.1 Two Dimensional Advection

To look at the distortion resulting from a given choice of element types in a simple two dimensional advection, the equation

$$\frac{\partial c}{\partial t} + U\frac{\partial c}{\partial x} + V\frac{\partial c}{\partial y} = 0 \qquad (4.8)$$

was solved. The mesh size, velocity field and initial condition used for the test are as follows,

$$\Delta x = 200\ m;\ \Delta y = 200\ m;\ x \in (-3400,3400);\ y \in (-3400,3400);$$

$$U = \omega\, x;\ V = \omega\, y\ ;\ \omega = \frac{2\pi}{3000}\ ;$$

$$C_o(x,y) = e\left(-\frac{(x-x_o)^2+(y-y_o)^2}{2\ \sigma_o^2}\right)$$

where $x_o = 0.0$, $y_o = -1800.0$, and $\sigma_o = 264.0$. The boundary conditions used for this case were all fixed value of 0.0 on all the boundaries.

The velocity vectors and the contour plots, as well as other salient information, for the solutions using the three different element types for median Courant numbers of 1.9 and 0.19, are shown in Figure 4.8 and 4.9 respectively.

Table 4.1 summaries the indicators used to judge the behavior of the the methods for both the Courant numbers.

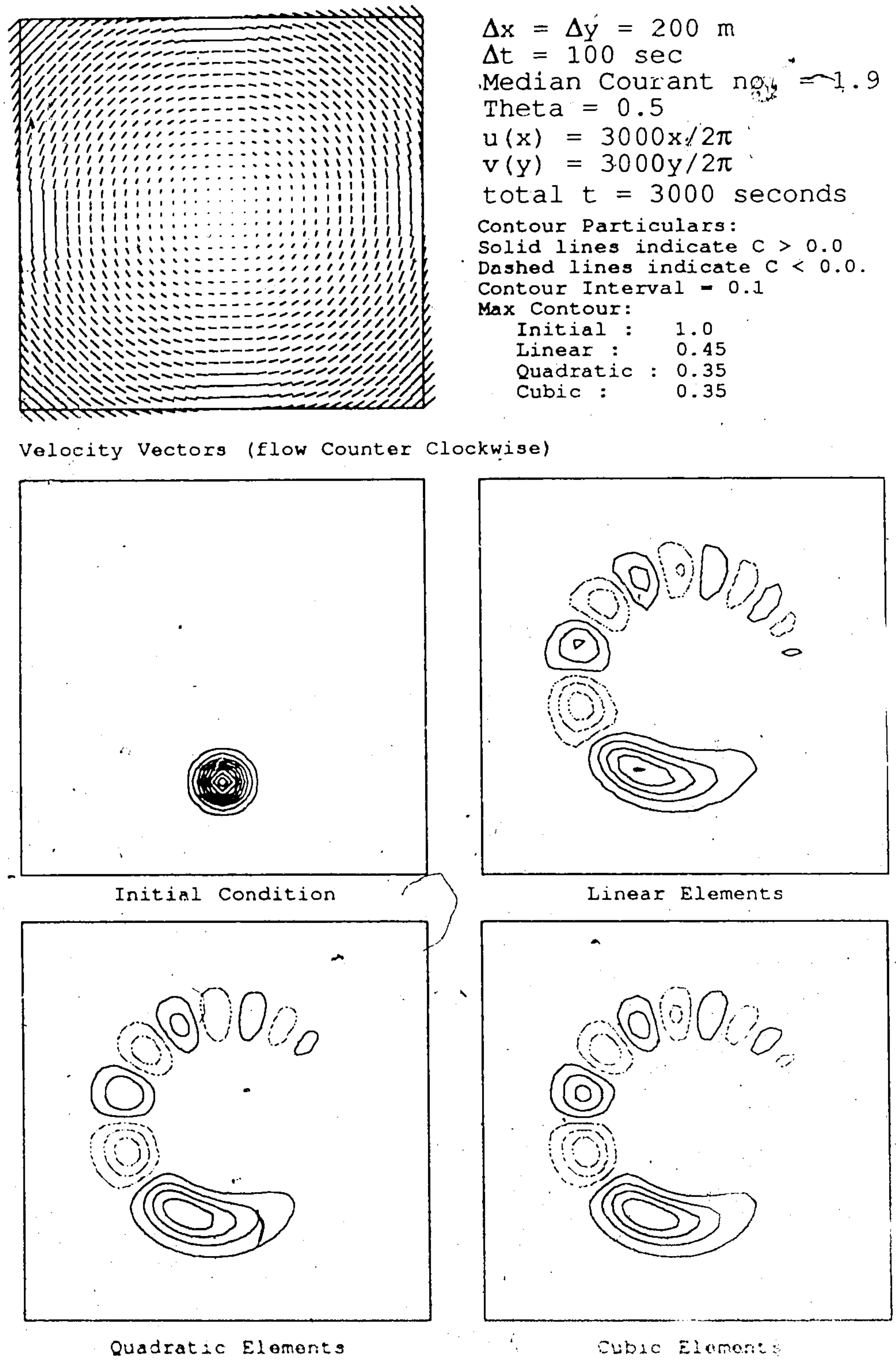
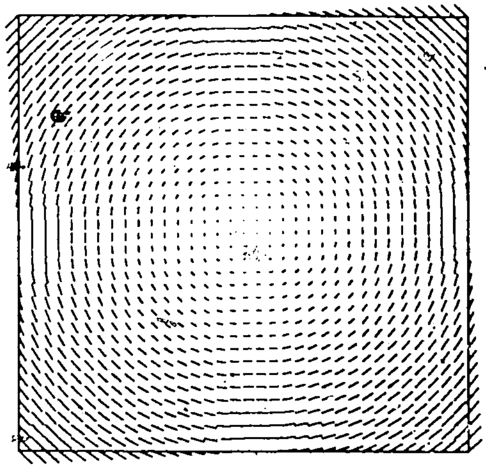


Figure 4.8 Circular Advection of a Gauss Hill at a Courant # ~ 1.9



$\Delta x = \Delta y = 200$ m
$\Delta t = 10$ sec
Median Courant no. = 0.19
Theta = 0.5
$u(x) = 3000x/2\pi$
$v(y) = 3000y/2\pi$
total t = 3000 seconds

Contour Particulars:
Solid lines indicate C > 0.0
Dashed lines indicate C < 0.0.
Contour Interval = 0.1
Max Contour:
Initial : 1.0
Linear : 0.75
Quadratic : 0.75
Cubic : 0.85

Velocity Vectors (flow Counter Clockwise)

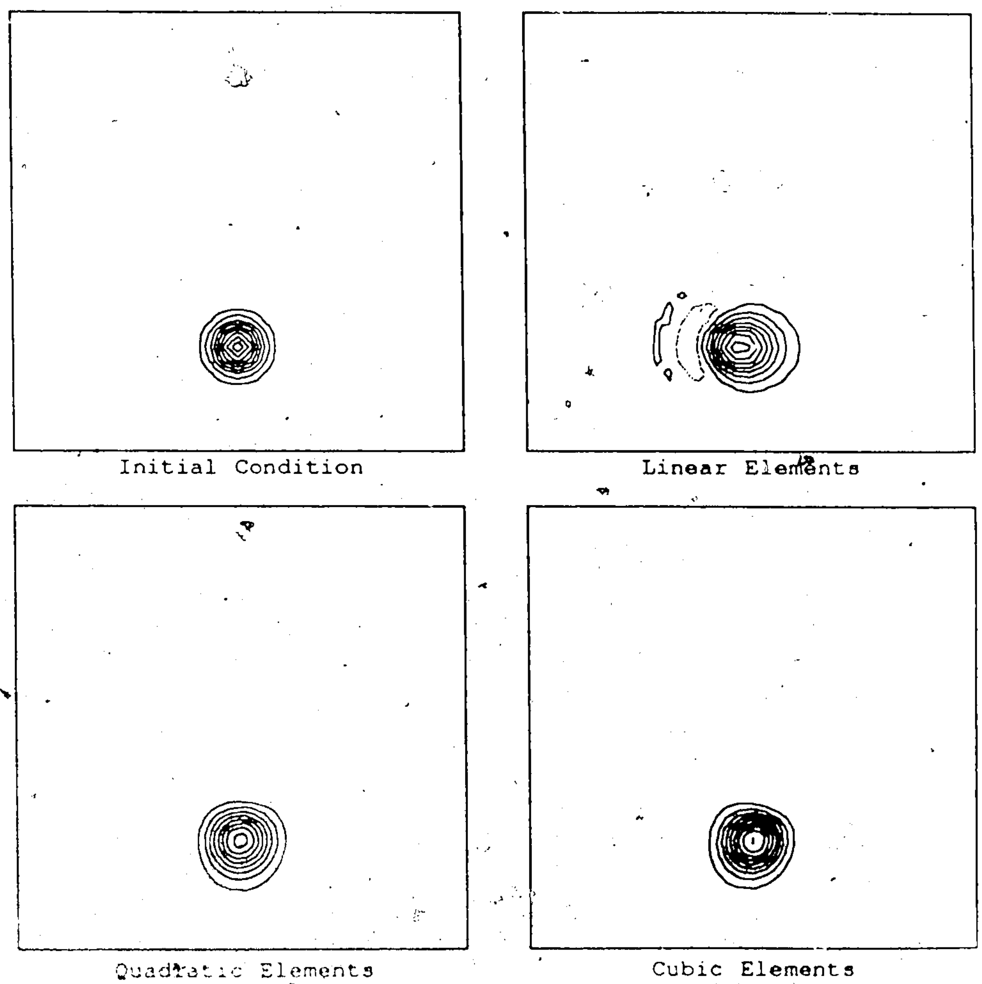


Figure 4.9 Circular Advection of a Gauss Hill at a Courant # ~ 0.19

C=1.9		LINEAR	QUADRATIC	CUBIC *
L-2 error Norm		0.0010900	0.0009489	0.0011117
Error in Peak		0.4633100	0.4254800	0.4517400
Error in Max neg.		-0.3127300	-0.3121500	-0.3227900
Error in Position				
of Peak	R	0.9938080	0.9162457	0.9493337
	Theta	0.9262082	0.9610104	0.9428999
Discrete L-2 Norm		5.64079E-06	4.76991E-06	5.53459E-06
Max Difference		0.6937200	0.6175700	0.6968200

C=0.19		LINEAR	QUADRATIC	CUBIC *
L-2 error Norm		0.0003307	0.0002116	0.0001494
Error in Peak		0.8229000	0.7799600	0.8526500
Error in Max neg.		-0.1207300	-0.0306500	-0.0364700
Error in Position				
of Peak	R	1.0061539	1.0000000	1.0000000
	Theta	0.9823884	1.0000000	1.0000000
Discrete L-2 Norm		1.76957E-06	1.05947E-06	7.38396E-07
Max Difference		0.2805892	0.2200400	0.1473500

* indicates the use of an iterative solution technique as discussed in 3.1.7

Table 4.1 Accuracy Measurements for the Advection of a Gauss Hill

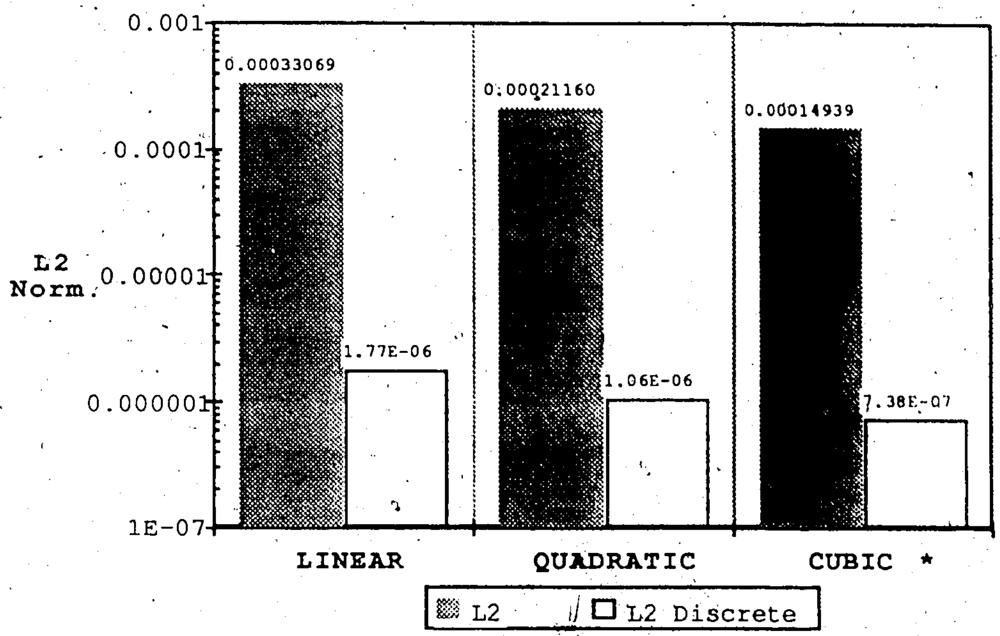
The indicators used to make the judgement are, the L2 error Norm defined by the expression

$$L2 = \frac{\int_{\Omega} (C_{num}-C_{exact})^2 \, d\Omega}{\int_{\Omega} C_{exact} \, d\Omega}; \qquad (4.9)$$

error in peak (Max), which should have an exact value of 1.0; error in minimum (Min), which should have an exact value of 0.0; error in the position as indicated by the value of the peak's radius and rotation compared to the exact solution; the discrete L2 norm as defined by Equation 4.7; and finally inf error, which is the maximum difference between the exact and the numerical solution.

It is apparent from the contour plots for the test at a Courant number of 1.9, that all three element types give a similar type of distorted solutions. This implies that at such a high Courant number the use of upwinding basis functions is of no added advantage. The same conclusions can be derived from poor performance shown by the bar charts of the error measures for the three methods shown in Figure 4.10.

At a median Courant number of 0.19, the solutions look to be very reasonable. LBG shows its wiggles behind the pulse, while QUPG shows a nice smooth solution. Also evident



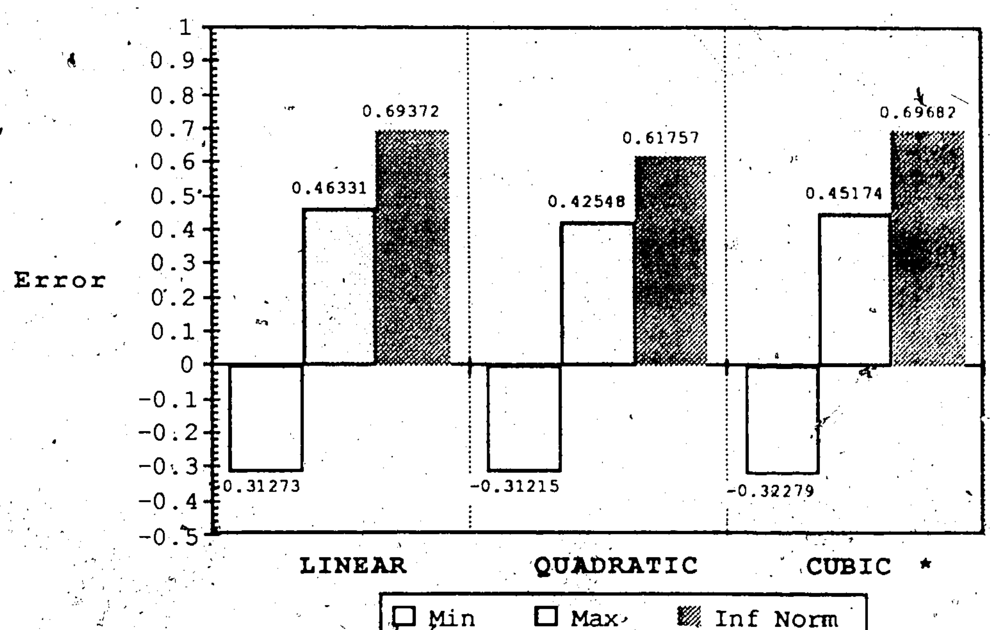


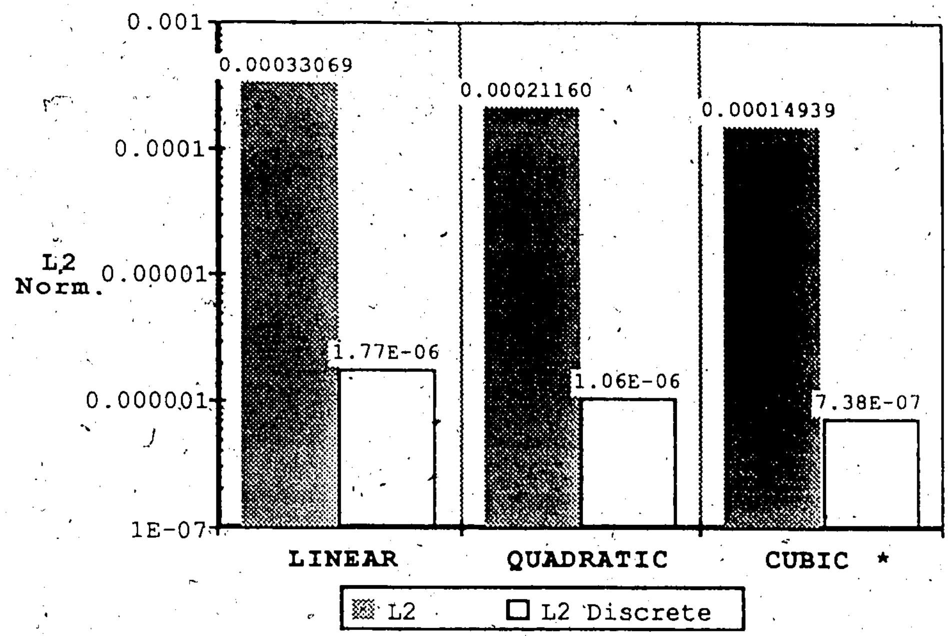
Figure 4.10 Error measures for 2-D advection Courant # 1.9

from the contour plots and the bar charts of the error measures in Figure 4.11, is that QUPG shows slightly more dissipation than CUPG, which is in agreement with results found for the one dimensional solution. The L2 norm seems to decrease in intervals of a third with each increase in order accuracy of the elements. That is it is reduced by a third from LBG to QUPG and a third form QUPG and CUPG. The inf norm reduces in a similar manner, expect it seems to decrease in 25% increments. There is an improvement in the size of wiggle (Min.) from LBG to QUPG (about 75%), however there only a slight improvement from QUPG to CUPG. The peak is preserved the best by CUPG. Generally the best performance is again given by CUPG.

4.2.2 Steady Plume in Skewed Meshes

One of the most important factor that indicate the applicability of a numerical method in practical situations is its ability to solve accurately in meshes that are skewed to the flow direction. A method that has this ability can accurately predict solutions in flows that are not aligned with the mesh such as circulations.

To compare the behavior of the new elements in a skewed mesh situation a steady problem was used. The problem consisted of a plume discharge into a body of water which had a constant velocity in a given direction.



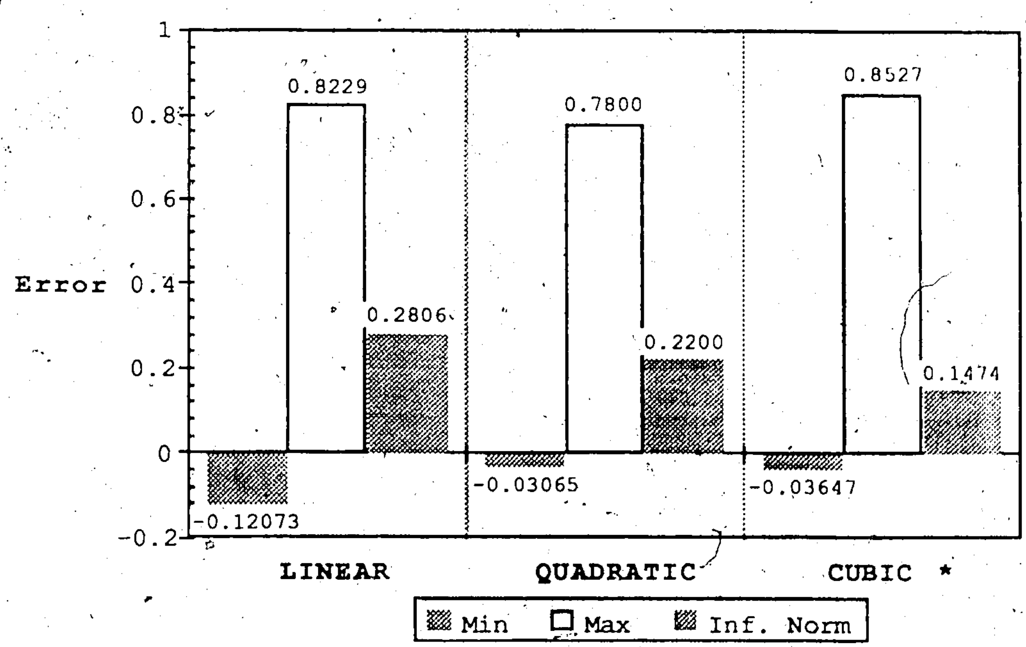


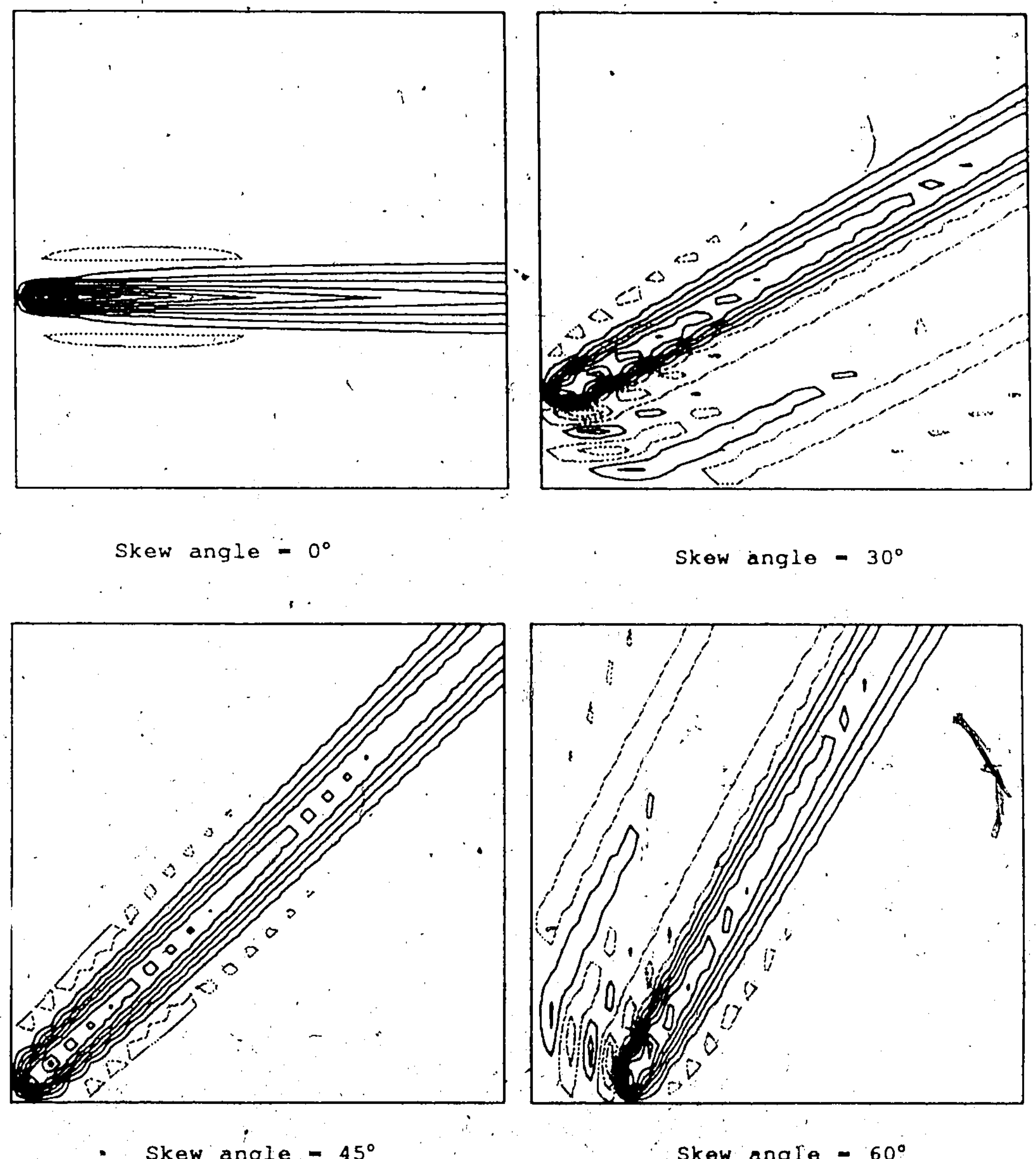
Figure 4.11 Error measures for 2 D advection Courant # 0.19

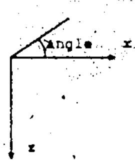
Two different cases were looked at to study this behavior. One in which the plume was solved for in four different directions skewed to the mesh, using the same method. While second case consisted of the use of the three element types to solve for a plume in a given direction of flow to the mesh.

4.2.2.1 Case 1

To infer the behavior to a natural channel situation, an 'infinitely' wide channel of constant depth of 2.0 m, slope of 0.0005, C_* of 25.0 and velocity of 2.475 m/s was used. Using a grid size of 1.0 m and both the longitudinal and the lateral diffusion given by $0.2hU_* \simeq 0.0391$, results in cell Peclet ($P_e \equiv \frac{U\Delta x}{E}$) number of 63.3. The flow was skewed to the mesh at angles of 0°, 30°, 45° an 60°.

The resulting concentration contours from using LBG are shown in Figure 4.12. The best solution is realized when the skew angle is zero and if the angle is changed, a similar solution should be obtained. However, since the LBG method prefers to have the flow in a favorable direction (ie 0° or 45°), there are oscillation produced. A summary of t salient features in presented in Table 4.2. Of interest in this case is the high ne ative concentrations for the 30° and 60° skew angle, which are in complete error. The test shows that LBG has a preference in the direction of the flow with respect to the mesh.





Δx = 1.0 m
Δy = 1.0 m
Slope = 0.0005
U = 2.475 m/s
Ezz = Exx = 0.2hU
depth = 2.0 m
C. = 25.0
Pe = 63.3

Contour Info:
Δ = 0.05
Dashed lines indicate values < 0.0
For more info. see Table 4.2

Figure 4.12 Solutions for different Skew angle Using LBG

Case 1
Solutions using Linear Elements at Different angles of Skew

Source = 1.0 g/s
$U = 2.475$ m/s
$E_{long} = E_{lat} = 0.2hU^* = 0.0391$
Slope = 0.0005
$C^* = 25.0$
depth = 2.0 m
$\Delta x = \Delta y = 1.0$ m
Cell Peclect no. = 63.3
All extremas are normalized by the Concentration at the source

Angle of Skew	Ratio of Cmax to C@source	Ratio of Cmin to C@source	Concentration at the Source
0°	1.2609	-0.0323	0.3841
30°	1.0000	-0.4247	0.3986
45°	1.0492	-0.0436	0.4061
60°	1.0000	-0.4247	0.3986

Case 2
Solutions for 60° Skewness using different elements

Source = 5.0 g/s
$U = 2.475$ m/s
$E_{long} = E_{lat} = 0.2hU^* = 0.0391$
Slope = 0.0005
$C^* = 25.0$
depth = 2.0 m
$\Delta x = \Delta y = 100.0$ m
Cell Peclect no. = 6330
All extremas are normalized by the Concentration at the source

Angle of Skew	Ratio of Cmax to C@source	Ratio of Cmin to C@source	Concentration at the Source
Linear	1.0787	-0.5218	204.5
Quadratic	1.1624	-0.3004	169.3
Cubic	1.2670	-0.1934	156.9

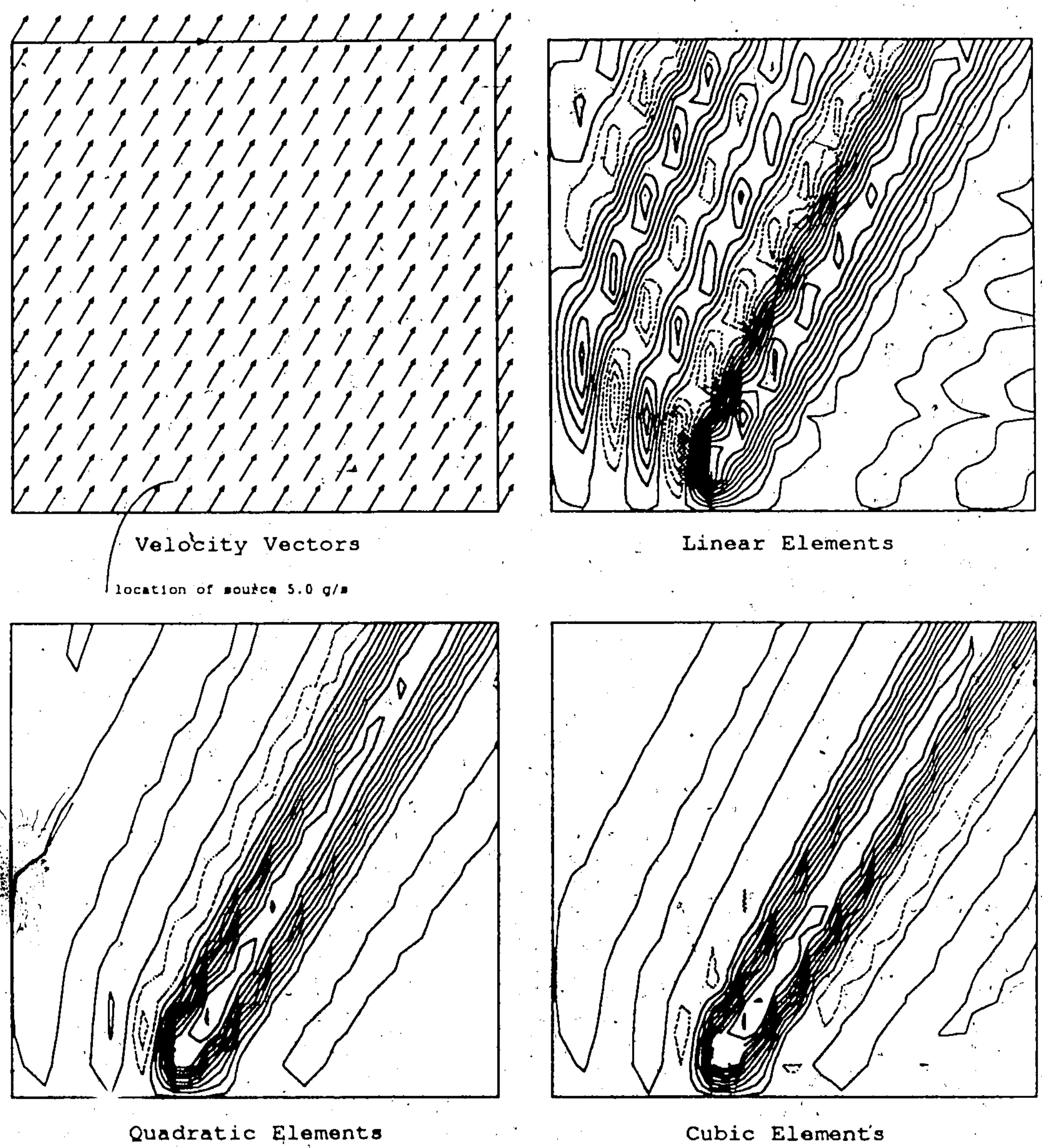
Table 4.2 Summary of Skew test, Case 1 and Case 2

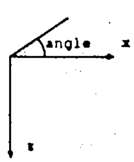
4.2.2.2 Case 2

The comparison of LBG to QUPG and CUPG in their ability to handle skewed mesh, was performed by looking at the solution given by each element type for a skew angle of 60°. The P_e was also hiked up so as to increase the difficulty each method has to resolve the solution. This was accomplished by increasing Δx and Δy to 100 m, which results in a 100 fold increase in the cell Peclet number to 6330. The strength of the source was also increased to 5.0 g/s so as to present a reasonable plume.

The contour plots of the resulting solutions are shown with the related details in Figure 4.13. Table 2b summaries the error measures used to judge the solutions. The maximum and minimum concentrations reach by each method indicate that LBG performs very poorly, as expected. However these indicator look considerable better for the case of CUPG. Thus the new elements seem to handle skewed meshes with far better success that the standard linear ones.

This conclusion can also be seen from Figure 4.14, which shows the concentration profile through the source in the y direction. From this plot it is evident that QUPG and CUPG both show one wiggle behind the core of the plume, while the rest of the domain is free from the large oscillations, which are so dominant in LBG.





Δx = 100.0 m
Δy = 100.0 m
Slope = 0.0005
U = 2.475 m/s
Ezz = Exx = 0.2hU.
dept 2.0 m
C. =)
Pe = 6330

Contour Info:
Δ = 20.0
Dashed lines
indicate
values < 0.0
For more info.
see Table 4.2

Figure 4.13 Solutions for different Methods for 60° skew angle, Pe = 6330

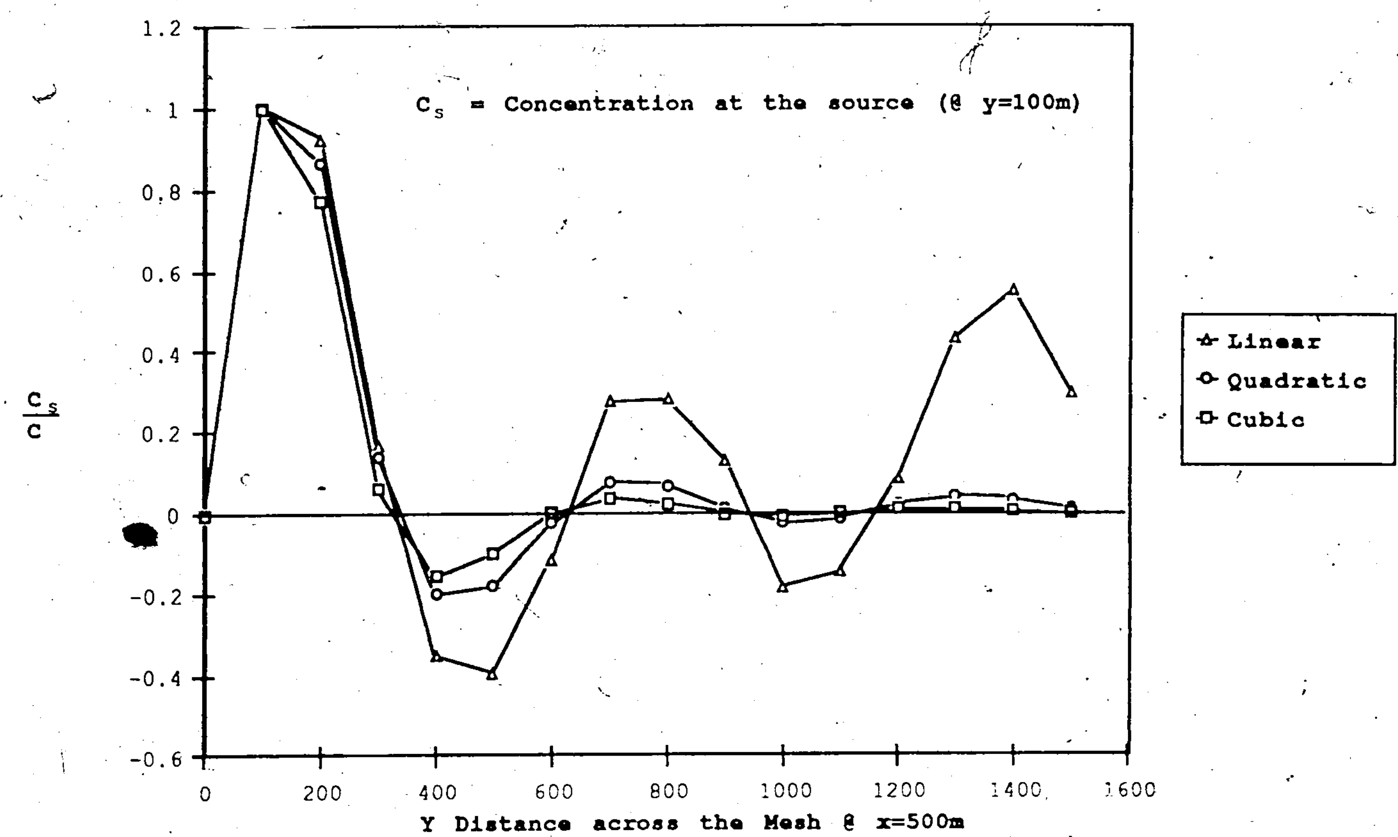


Figure 4.14 Concentration Profile across the Solutions for Skew test.

4.3 Conclusion of Numerical Tests

The tests performed here on the new elements indicate that their performance is better than the standard LBG method. However it should be noted that this increase in accuracy means an increase in the computational effort required. The use of QUPG and CUPG increases the computing time and the storage requirements for a given problem.

Figure 4.15 show that in the solution of a 20 by 20 mesh using banded storage, the storage required increases by a factor of 2 and 3 for the QUPG and CUPG elements respectively. The computational time on a Macintosh II machine increases by 3 and 6 times for the two element types. The use of a particular type of element must be weighted between the accuracy required and the computational resources available for a given situation.

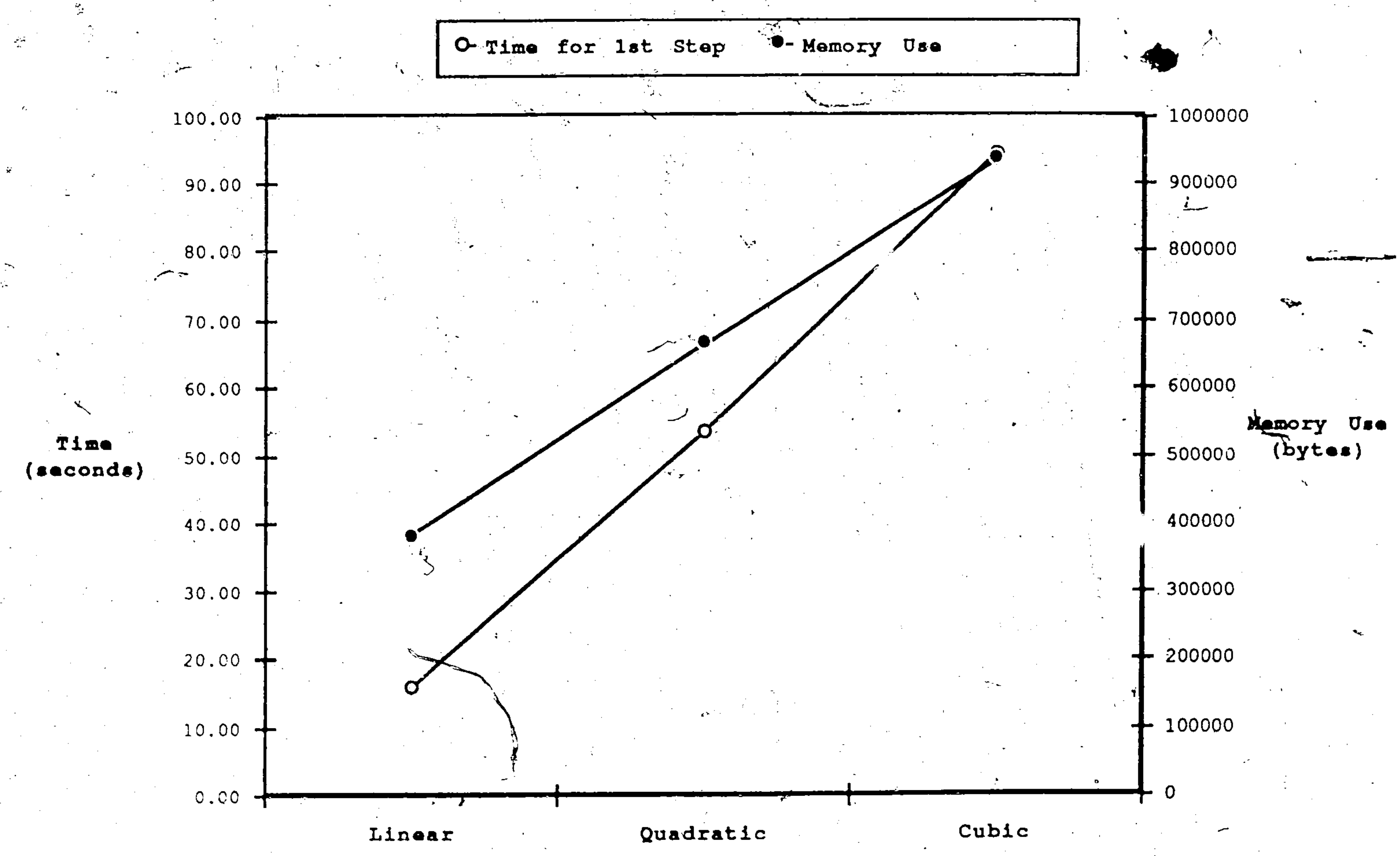


Figure 4.15 Computational effort required for the three methods

5. APPLICATIONS

Ideal and practical problems were solved using the numerical model to understand and reinforce the physical aspects of effluent discharges into natural channels. The idealized investigation consisted of studying the relationship between plume characteristics and modelling techniques used to access the variation of eddy diffusivity in a river channel. The second set of studies looks at the application of the numerical model to a practical river channel situations.

5.1 Variation of Eddy Diffusivity.

One of the hardest parameters to determine for a plume discharged into a river is its diffusive behavior. Generally it is believed that the simplifying assumption made, such as the use of constant eddy diffusivity is sufficiently accurate for practical analysis. However by making this assumption what important two dimensional characteristics are we ignoring ? To answer such a question two study problems were solved. One was to look at the discharge of a plume in the center of a channel having a cross-section shaped like a triangle. The other consisted of a side discharge into a trapezoidal shaped channel.

5.1.1 Center Discharges

The first investigation consisted of the discharge of a plume in the center of a channel with a triangular shaped cross-section. A triangular shape was chosen so as to exaggerate the effects, and to infer the behavior to a cross-section similar shape in the bends of rivers. The channel width was chosen to be 50 m and a maximum depth of 1.0 m was set on one side, resulting in an aspect ratio of 50. The length of the channel was chosen to be 8 km. The source of the plume was discharged into the river centered around the 27.5 m. The mesh used to model this plume was 20 elements long by 30 elements wide which set $\Delta x = 400$ m and $\Delta z = 1.67$ m. The slope of the channel was set to 0.0005 m/m. The resulting cell Peclet numbers were $\frac{40000}{h}$ and $\frac{278}{h}$ in the x and z direction respectively, where h is the depth.

The mesh was solved using the LBG formulation. There did not exist a need to use higher ordered elements since the discretization was well graded. As well the cell Peclet number were relatively small, particularly in the z direction, where the large changes are taking place. In addition the mesh was aligned to one of the preferred direc ons, so that the solution would not introduce any errc s due to mesh being skewed to the direction of flow.

The two solution were performed, one with the use of E_{zz} as a function of the depth, and another with E_{zz} being constant across the channel width. The E_{zz} as a function of the depth was given by the equation

$$E_{zz} = 0.15\, h\, U_* \tag{5.1}$$

and the average value given by the expression

$$\overline{E_{zz}} = \frac{1}{Na} \sum_{i=1}^{Na} (0.15 h U_*)_i \tag{5.2}$$

where Na is number of nodes across the channel(31). In both cases the value of E_{xx} was set by the expression,

$$E_{xx} = 0.25\, h\, U_* \,. \tag{5.3}$$

Figure 5.1 show the distributions of E_{xx}, E_{zz}, and U across the channel resulting from these conditions.

The resulting contours from the two solutions as well as a plot of the mesh and velocity vectors is shown in Figure 5.2. Figure 5.3 shows the spreading characteristics of the plume half width long the channel. From the contours plots as well as the concentration profiles, it is evident that the use of an average eddy diffusivity across the channel alters the spreading rate of the plume. The plot shows that in the case where an average diffusivity is used, the spreading is larger in the shallow portion of the channel, while in the deep portion the spreading is smaller. This is due to the distribution of the diffusivity across the channel. In the shallow portion $\overline{E_{zz}}$ is larger than E_{zz} and thus the spreading is larger. While in the deep portion $\overline{E_{zz}}$ is smaller than E_{zz} and therefore the plume spreads less. For comparison, the spreading rate of a point source given by the integral

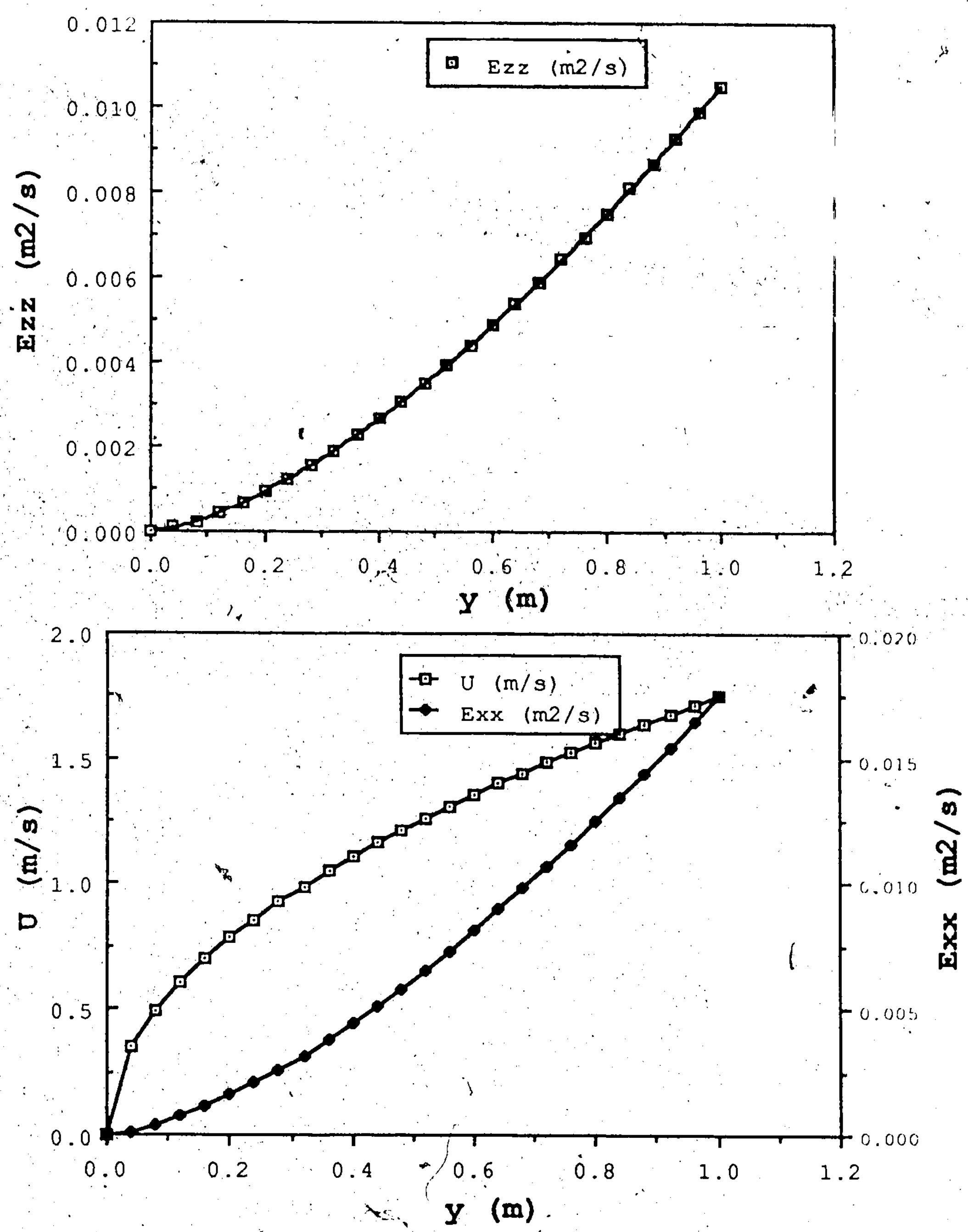


Figure 5.1 Variation of Exx, Ezz and U

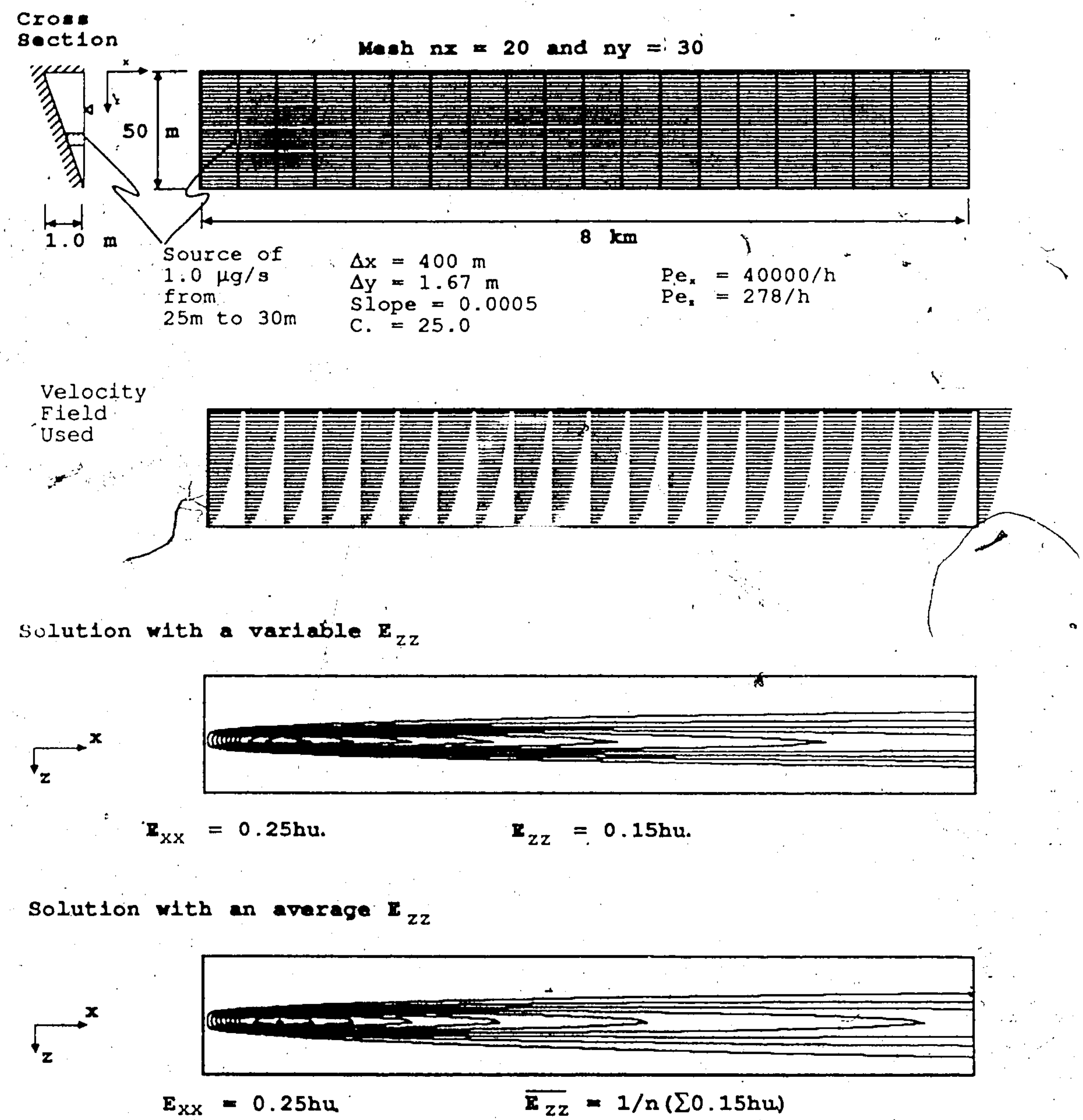


Figure 5.2 Results for Triangular Channel

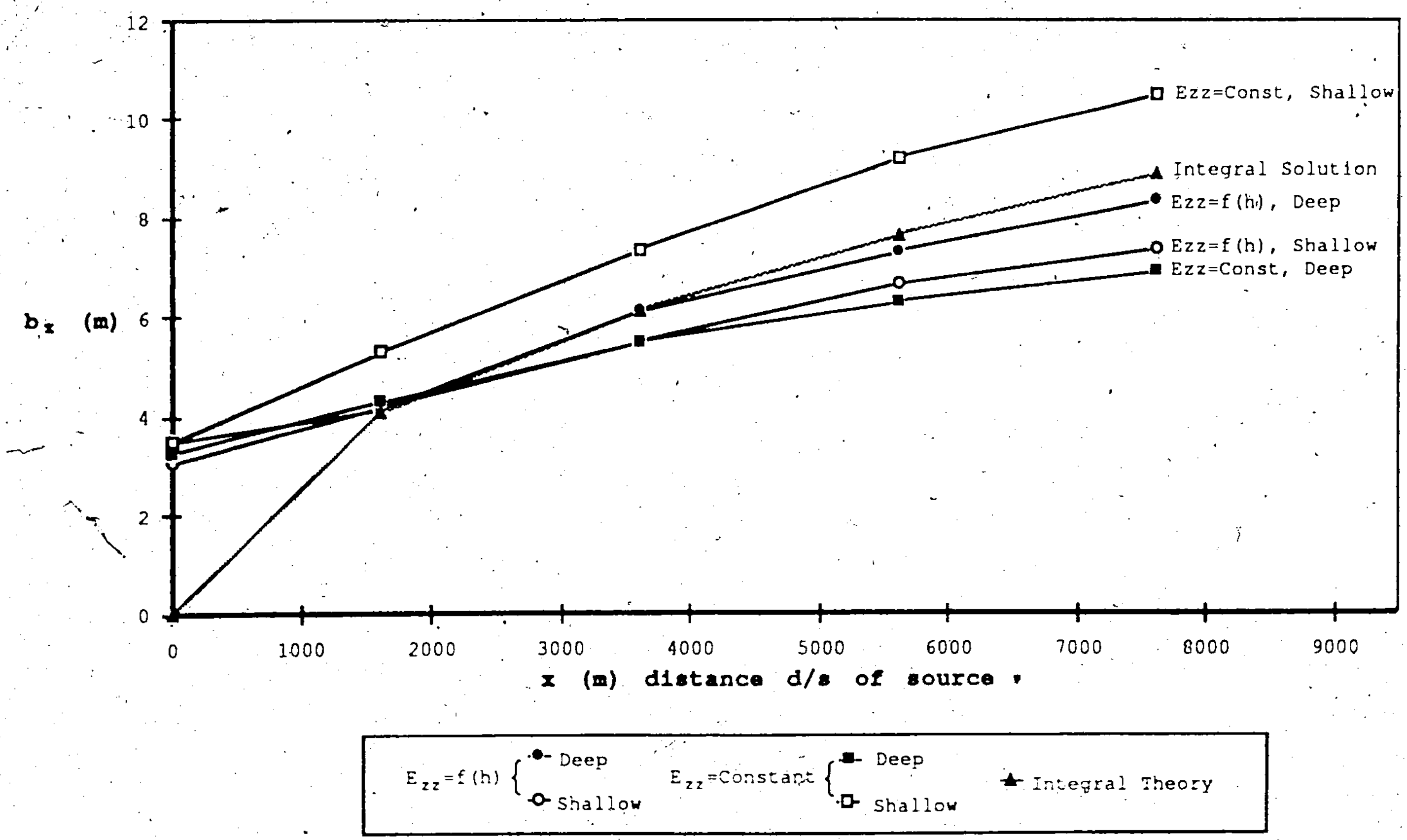


Figure 5.3 Plume Spreading Rates in a Triangular Cross-Section Channel

solution is also included in the plot (Rajaratnam, 1970). This solution fits nicely into the middle of the profiles, indicating its integral nature.

The longitudinal diffusion characteristics of the plume is shown in Figure 5.4, which shows a plot of the center line concentration along the channel. From this plot it is evident that the use of $\overline{E_{zz}}$ exaggerates the spreading rate, as indicated by the lower concentrations than those for the case of when E_{zz} was used.

In general, it can be concluded that for a plume situated in the center of the channel the use of $\overline{E_{zz}}$ across the channel will result in smearing of any anomalies that may exist due the the variation of the depth.

5.1.2 Side Discharges

The second situation studied was that of a side plume discharge into a channel with cross-section shaped like a trapezoid. The parameters used were the same as for the triangular channel. However the depth was kept at a value of 1.0 m for distance of 36.0 m from 3.0 m onward. The slope of the trapezoidal side was chosen as 3 horizontal to 1 vertical. The discharge was modelled by introducing a source of strength 1.0 μg/s over the sloping portion of the channel at a distance 160 m from the beginning of the left-hand side

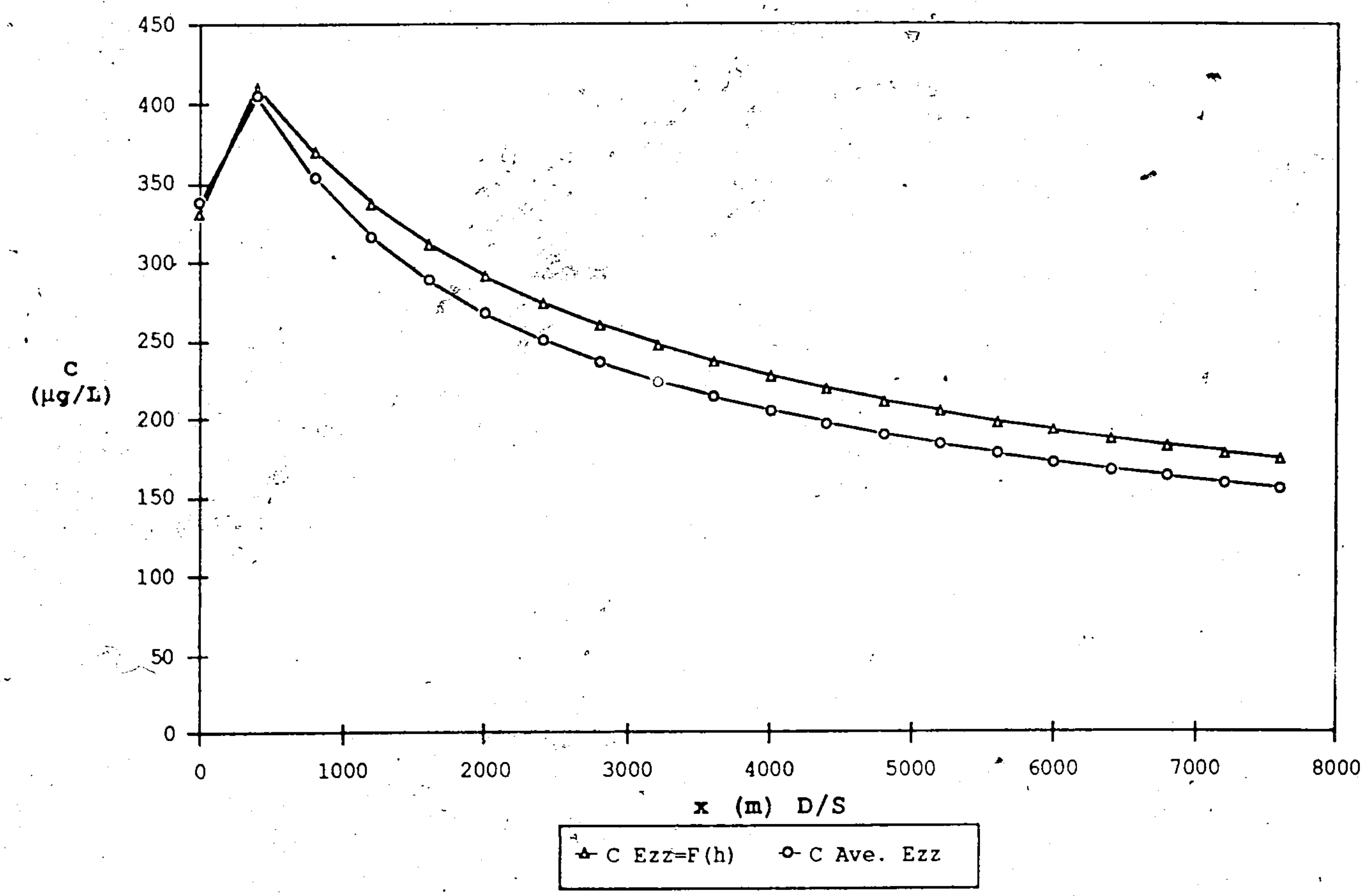


Figure 5.4 Concentration Profile in the Center of Plume Along the Channel

of the mesh. An illustration of the mesh, velocity vectors, and a summary of the salient features is shown in Figure 5.5.

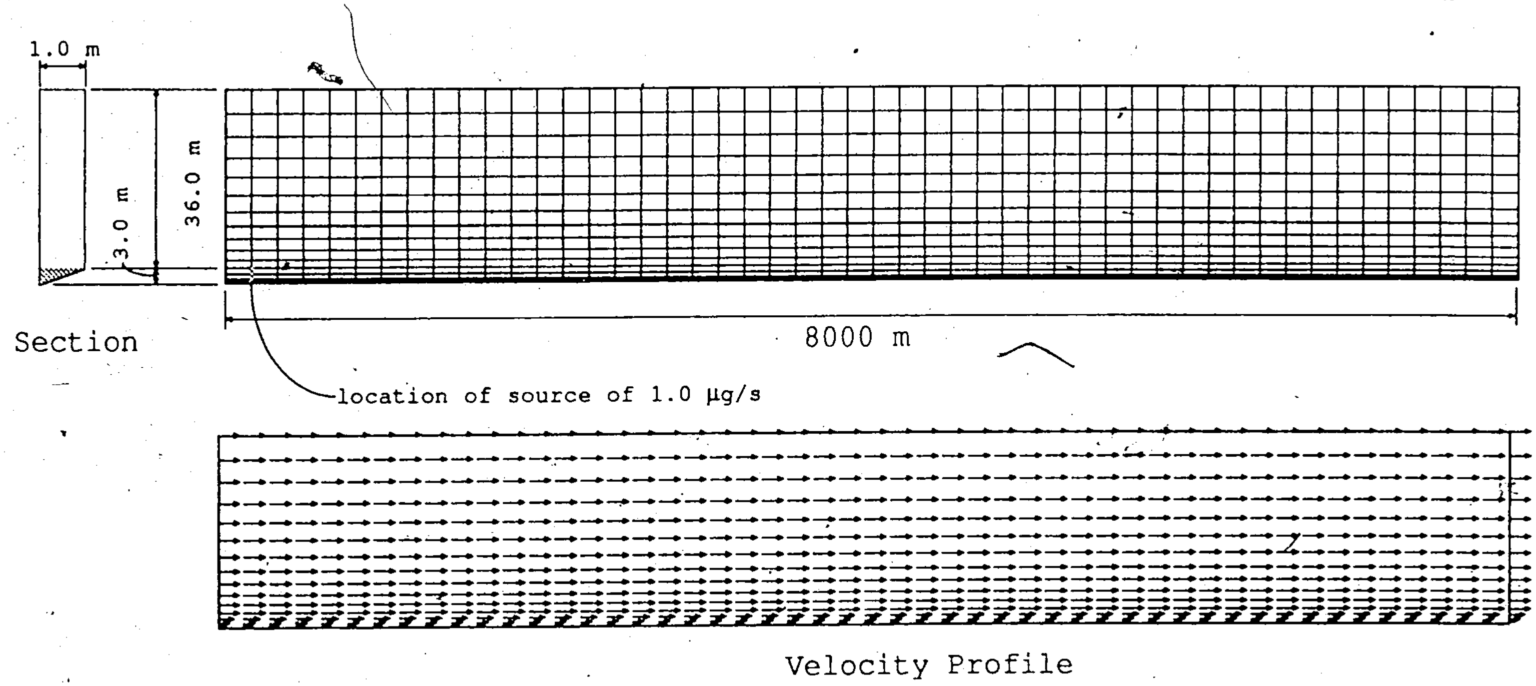
The problem was solved for the four different scenarios shown below,

(a)	E_{zz} = f(h)	E_{xx} = f(h)
(b)	E_{zz} = Constant	E_{xx} = f(h)
(c)	E_{zz} = Constant	E_{xx} = Constant
(d)	E_{zz} = Constant	E_{xx} = 0.0

Case (a) and (b) were used to observe the behavior due to the variation of E_{zz}, while (b), (c) and (d) were performed to observe the behavior due to E_{xx}.

The contour plots for each of the above case are shown in Figure 5.6. In addition a close-up of the contours for cases (a) and (b) are shown in Figure 5.7. A plot of the half-width(b_z) is also shown in Figure 5.8, and a plot of the concentration profile along the bank is shown in Figure 5.9.

These plots clearly show the difference between using a constant E_{zz} and a variable E_{zz}. As was found for the case of the center discharge earlier, the spreading rate of the plume is increased when an average value is used for E_{zz}. As a consequence the concentration on the bank is a great deal higher for case (a), than for case (b). The close-up of the contour plot shows another visual difference between the two cases. The use of a constant E_{zz} results in numerical result where the concentration contours are orthogonal to the



Δx min.= 0.24 m; max.= 5.0 m
Δy = 160.0 m
$E_{xx} = 0.25\ h\ U_*$
$E_{zz} = 0.15\ h\ U_*$
$U_* = 0.0700$ m/s
$S = 0.0005$
$h = 1.0$ m
$C_* = 25.0$

Figure 5.5 Mesh and Velocity Vectors

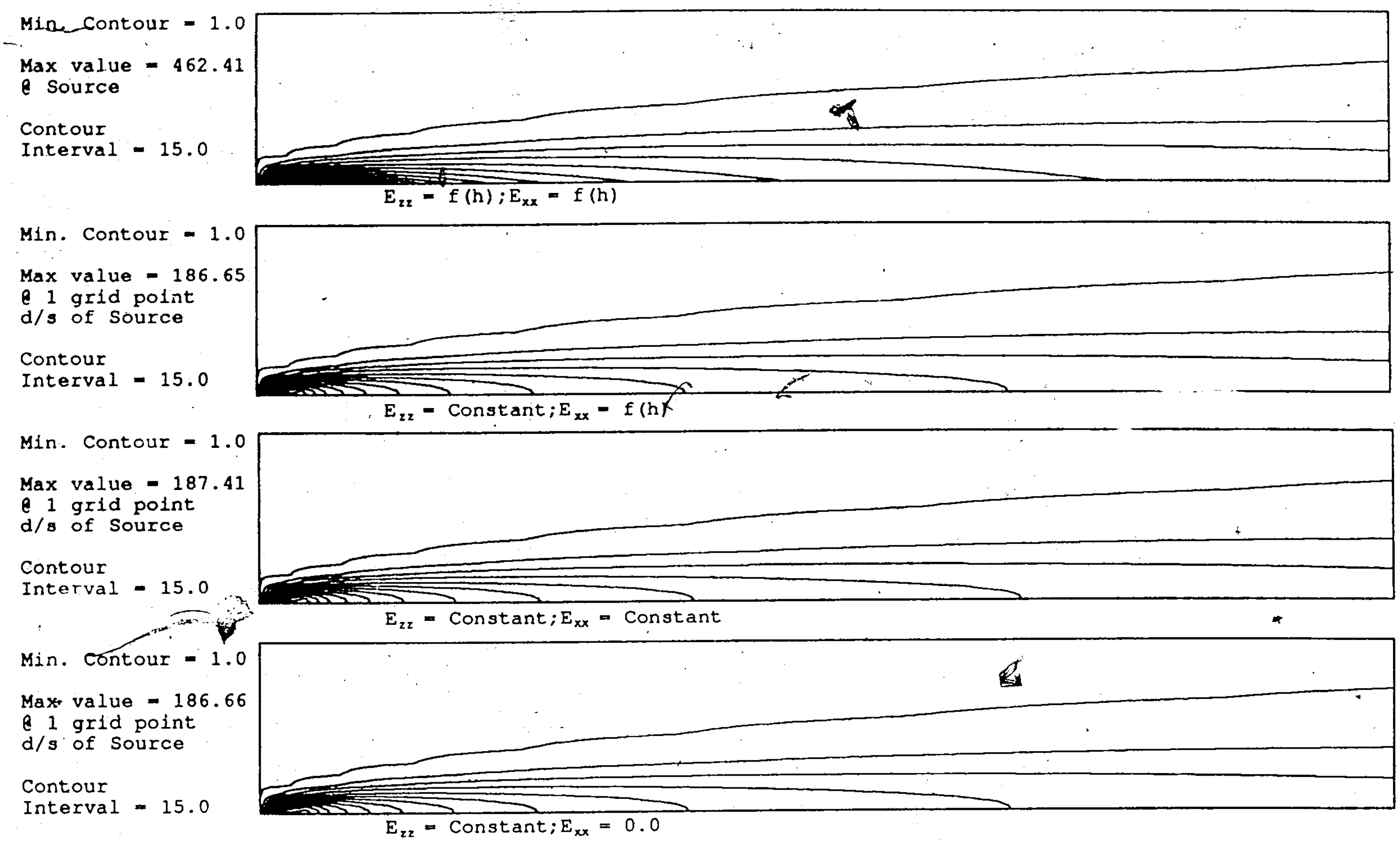
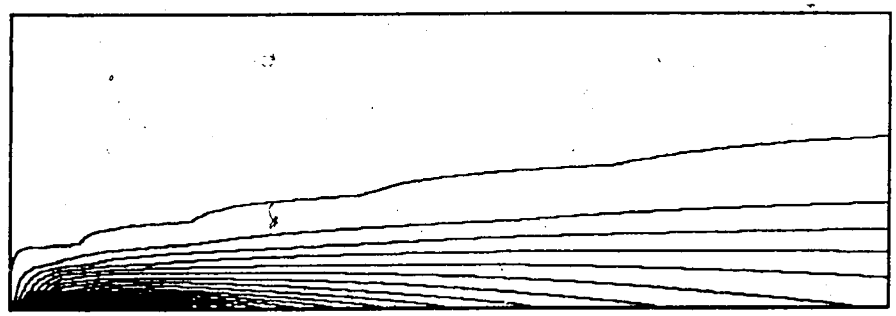


Figure 5.6 Contour Plots for Trapazoidal Shape Channel

Min. Contour = 1.0

Max value = 462.41
@ Source

Contour
Interval = 15.0



$E_{zz} = f(h); E_{xx} = f(h)$

Min. Contour = 1.0

Max value = 186.65
@ 1 grid point
d/s of Source

Contour
Interval = 15.0

$E_{zz} = \text{Constant}; E_{xx} = f(h)$

Figure 5.7 Close-up of the Contours

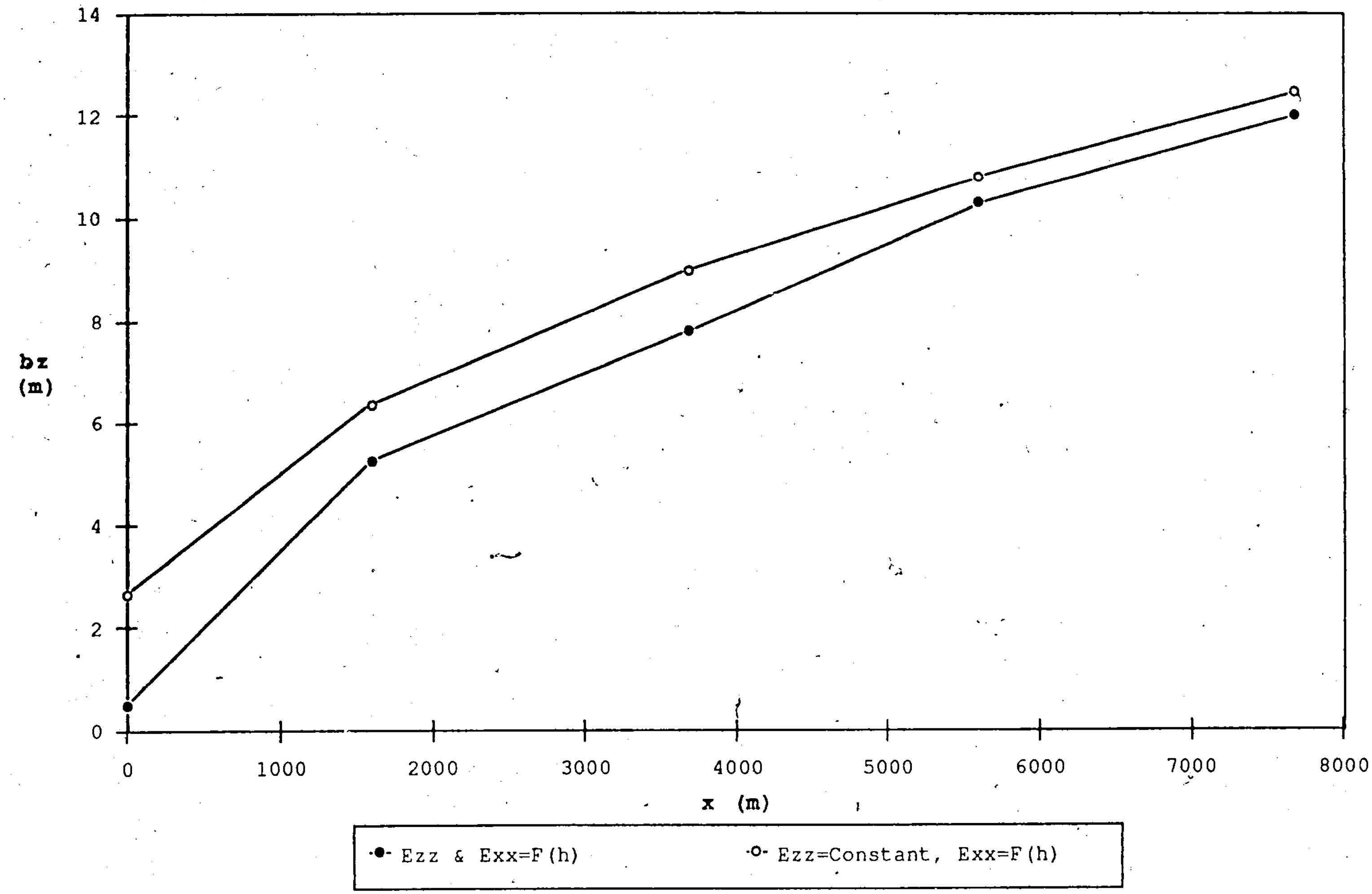


Figure 5.8 Variation of bz, the plume half width along the channel

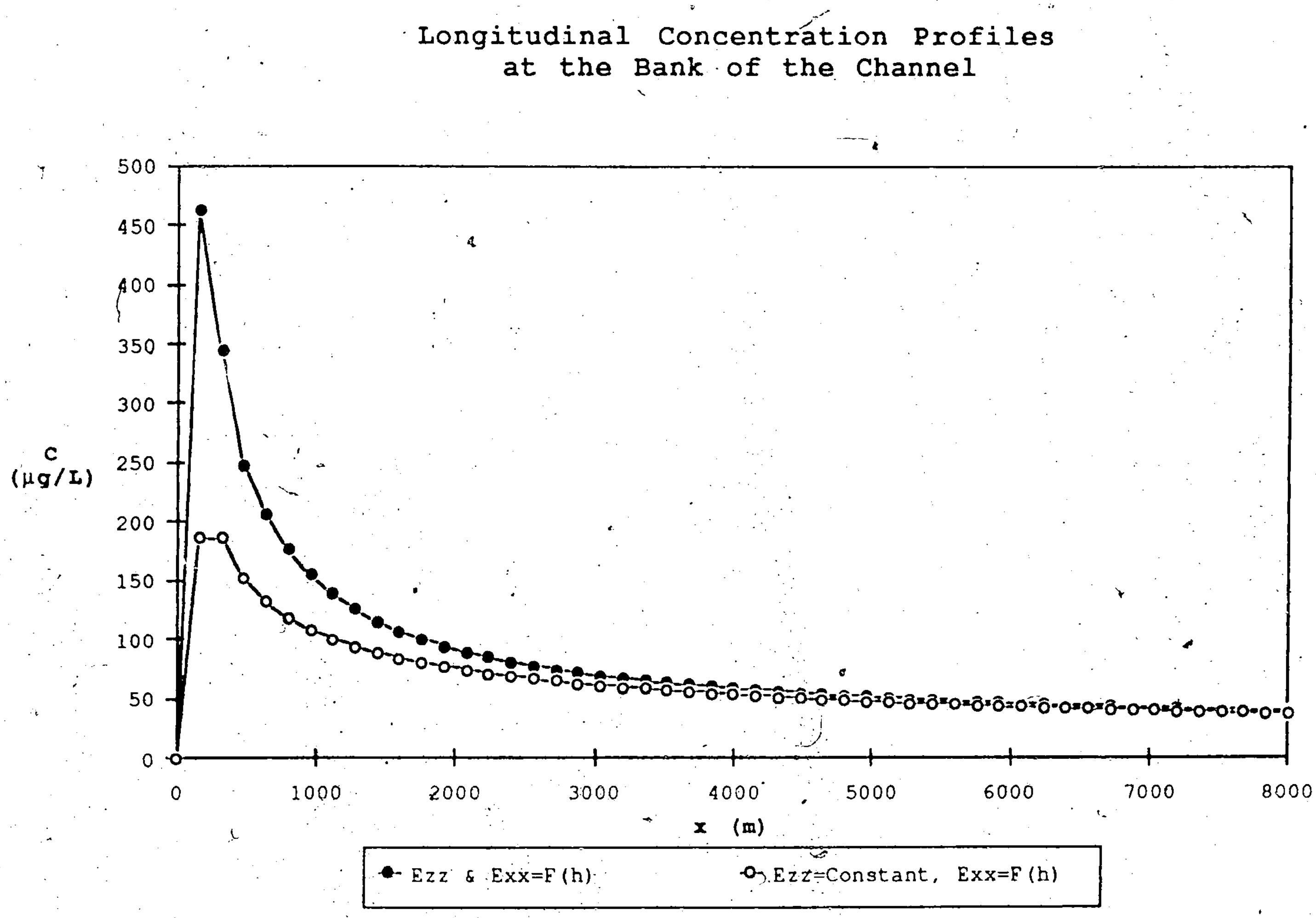


Figure 5.9 Concentration Profile along the Channel

boundary. However when E_{zz} was allowed to vary with the depth, which implies that E s zero on the bank, the resulting contours are no longer orthogonal to the bank.

It is evident from the longitudinal profile and from the spreading rate plot as well as the close-up, that the use of $\overline{E_{zz}}$ results in the diffusion of the concentration in the bank. This implies that when a constant E_{zz} is used the maximum concentrations predicted are generally on the non-conservative side.

The specification of the E_{xx} is generally believed to be unimportant in the solution. This was reinforced by the analyses performed here. It is clearly apparent from the contour plots that different models of E_{xx} results in very little difference in the solution. This is also shown in the plots of the concentration on the bank, in Figure 5.10.

5.2 Application to River Plume Discharges

A plume study was done to model a river reach surveyed by Krishnappan and Lau (1982) on the Grand River near Kitchener, Ontario, Canada, to judge the performance of the numerical model in real river situation . A sketch of the river reach plan view, as well as a rough sketch of the cross-sections where data was measured is show in Figure 5.11.

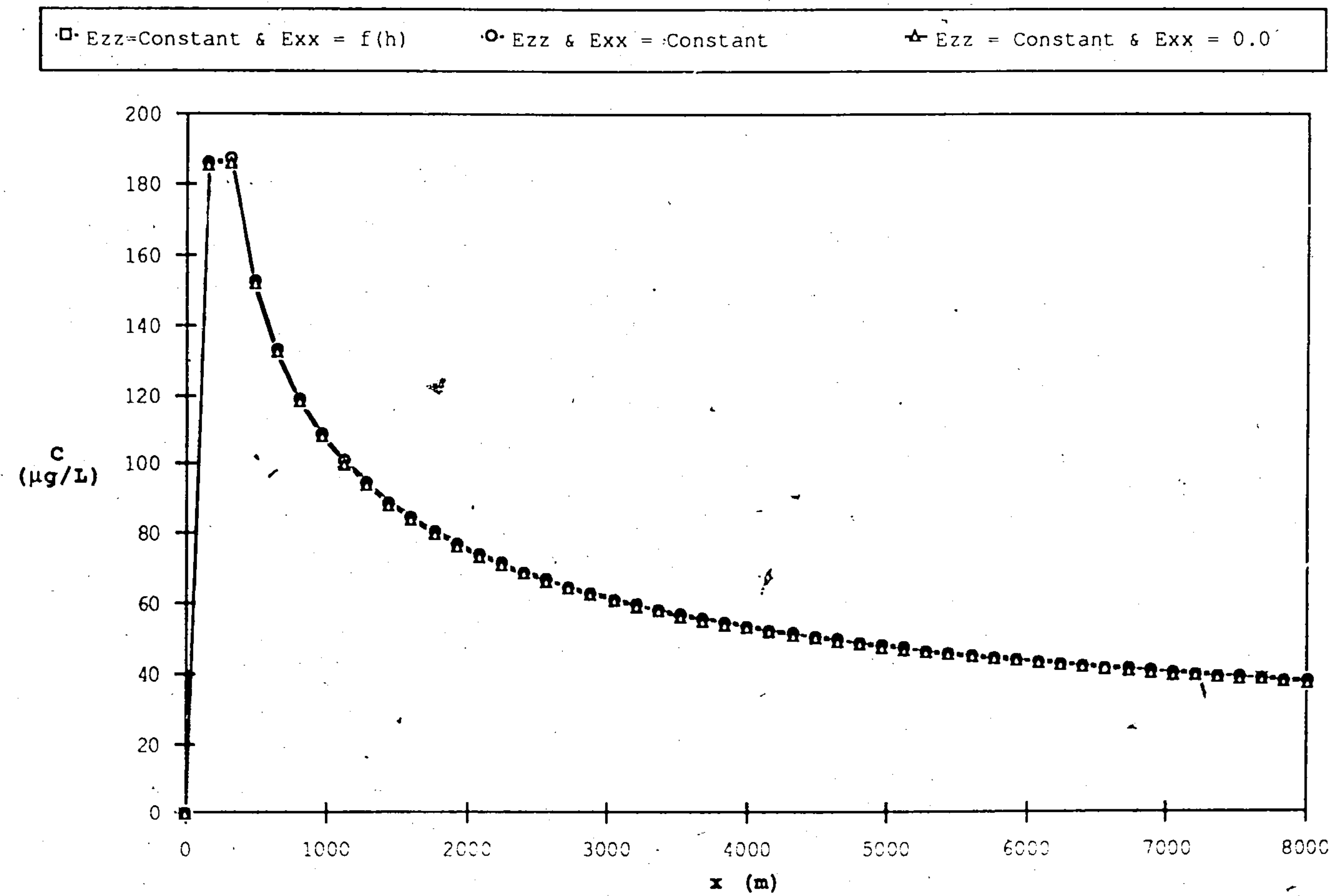


Figure 5.10 Longitudinal Concentration Profiles at the Bank of the Channel for cases b,c and d

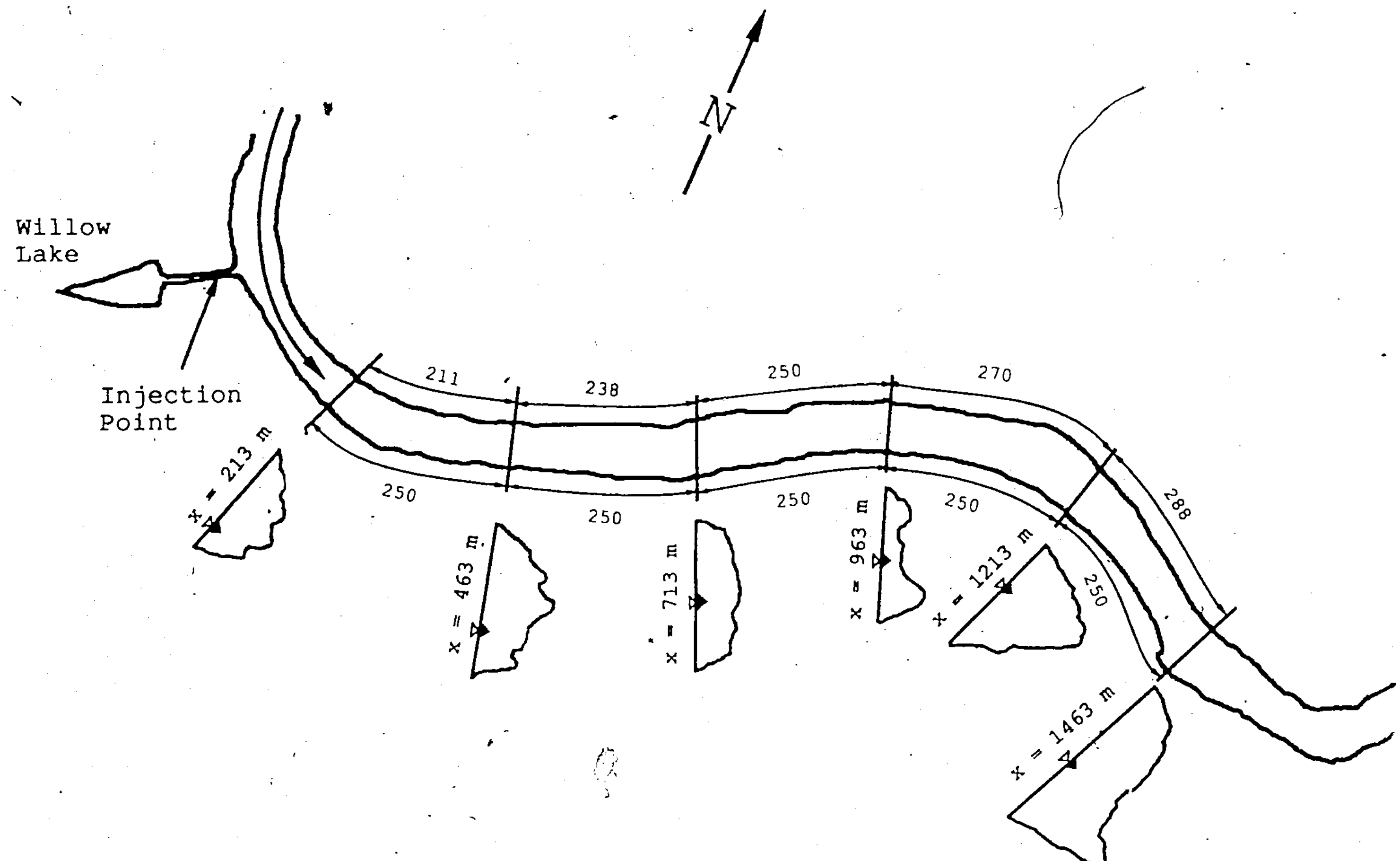


Figure 5.11 A Sketch of Grand River near Kitchener, Ontario, Canada (after Krishnappan and Lau 1982)

Krishnappan and Lau performed the plume test, where by a constant discharge of Rhodamine B dye was introduced into a tributary just downstream of Willow Lake. They subsequently measured the concentrations in the river at the cross-sections shown in Figure 5.11. They also measured the depth and a depth averaged velocity at each cross-section, at about 3.0 m intervals across the channel. The general river characteristics and pertinent data are reproduced in Table 5.1. The measured transverse diffusivity fitted the expression,

$$E_{zz} = 0.26 \; h \; U_* \qquad (5.4)$$

according to Krishnappan and Lau.

The above river reach was discretized into a 40 long by 20 wide elements mesh, using quadratic mapping it interpolate the intermediate elements. The resulting mesh and velocity vectors are plotted in Figure 5.12. The eddy diffusivities were experimented with until the best solution was obtained. The values settled upon were given by the expressions,

$$E_{zz} = 0.25 \; h \; U_* \text{ and} \qquad (5.5)$$

$$E_{xx} = 0.375 \; h \; U_* \; . \qquad (5.6)$$

The solution for the plume was undertaken by imposing the measured concentration distribution at $x = 213$ m as a given boundary condition on the left side of the mesh. The numerical model was used to solve for the steady state solution which resulted in the concentration contours shown

TRANSECT NUMBER	DISTANCE FROM SOURCE (m)	CHANNEL WIDTH W (m)	AVERAGE DEPTH H (m)	MEAN VELOCITY U (m/s)	MEAN SHEAR VELOCITY U* (m/s)	FLOW RATE Q (m /s)	NON-DIMENSIONAL CHEZY COEFFICENT C*
1	213	49.3	.47	0.30	0.067	10.84	4.51
2	463	60.0	0.57	0.29	0.074	10.03	3.92
3	713	60.0	0.42	0.42	0.063	10.60	6.67
4	963	55.7	0.28	0.70	0.052	10.08	13.46
5	1213	57.0	0.65	0.29	0.079	10.70	3.67
6	1463	79.5	0.55	0.23	0.072	10.29	3.19

FOR EACH CROSS-SECTION A REPRESENTATIVE U* WAS USED
DEPTH WAS ALLOWED TO VARY ACROSS THE CHANNEL AND THEREFORE EVERY WHERE

$$E_{zz} = 0.25\ U^*\ h$$
$$E_{xx} = 0.38\ U^*\ h$$

Table 5.1 Hydraulic Data for the Grand River

in Figure 5.12. The concentration distribution across the channel at each cross-section is shown in Figure 5.13. As a comparison to the concentration profiles were also plotted against the measured results in Figures 5.14 to 5.19.

The results from the numerical solution seem to be in agreement with the measured values. The discrepancies seen at x = 963 and x = 1213 are largely due to the fact that the channel reach is not straight. With a curved channel it is almost impossible to describe the behavior of the velocity field by any known method. The magnitude of the velocity vectors derived here are only representative at the given cross-sections. Due to the fact that the direction is not known, the velocity field used here is only a best estimate of the actual field. The model used here to interpolate the direction is only applicable to well behaved river reaches. In reaches such as this one where there is a substantial curvature the model has difficulties. The ability to accurately predict the measured concentration field is hampered by the lack of velocity direction information.

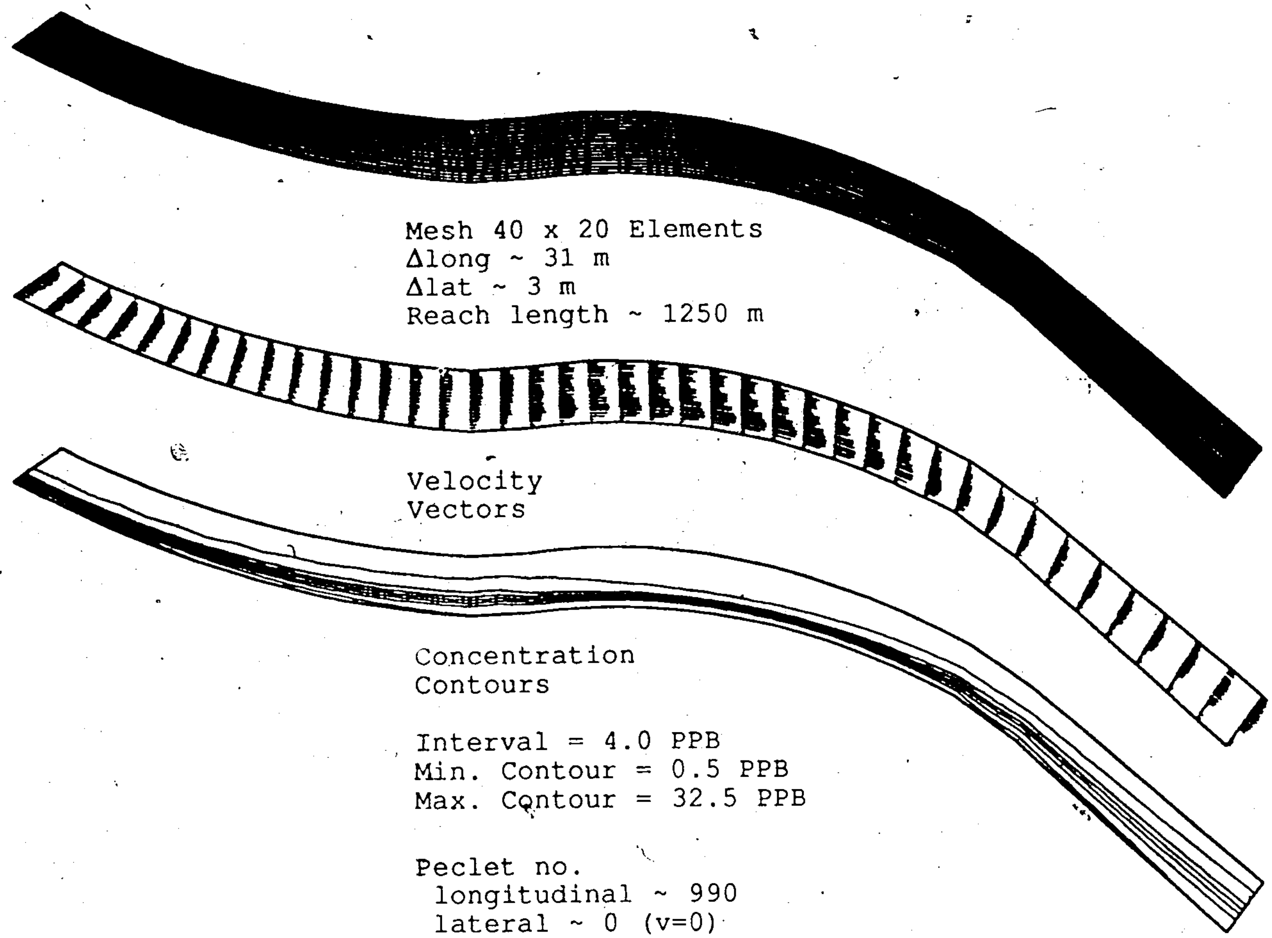


Figure 5.12 Mesh, Velocity, and Contours for Grand River

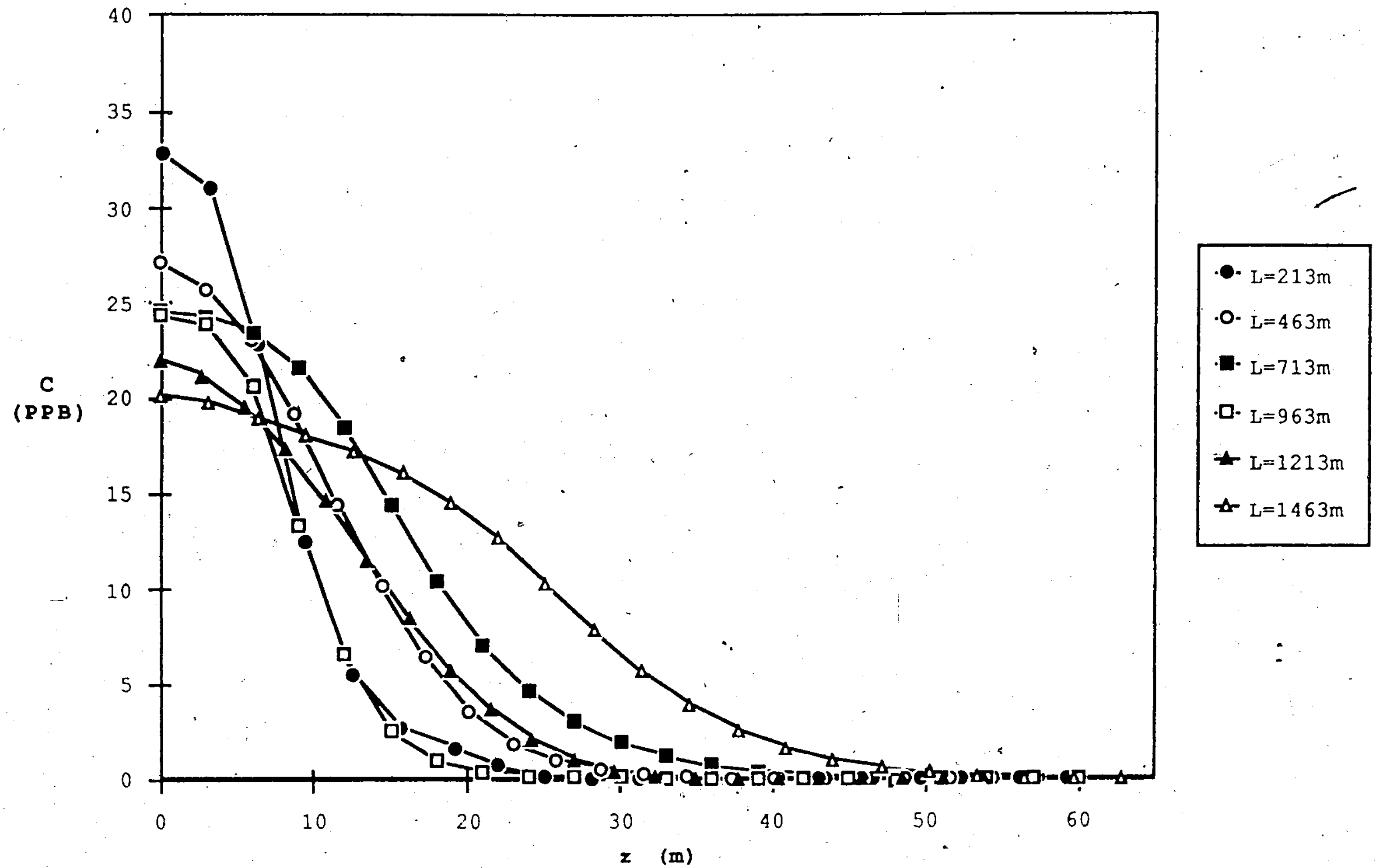


Figure 5.13 Concentration Profiles at Different Stations Grande River near Kitchener Ontario Canada

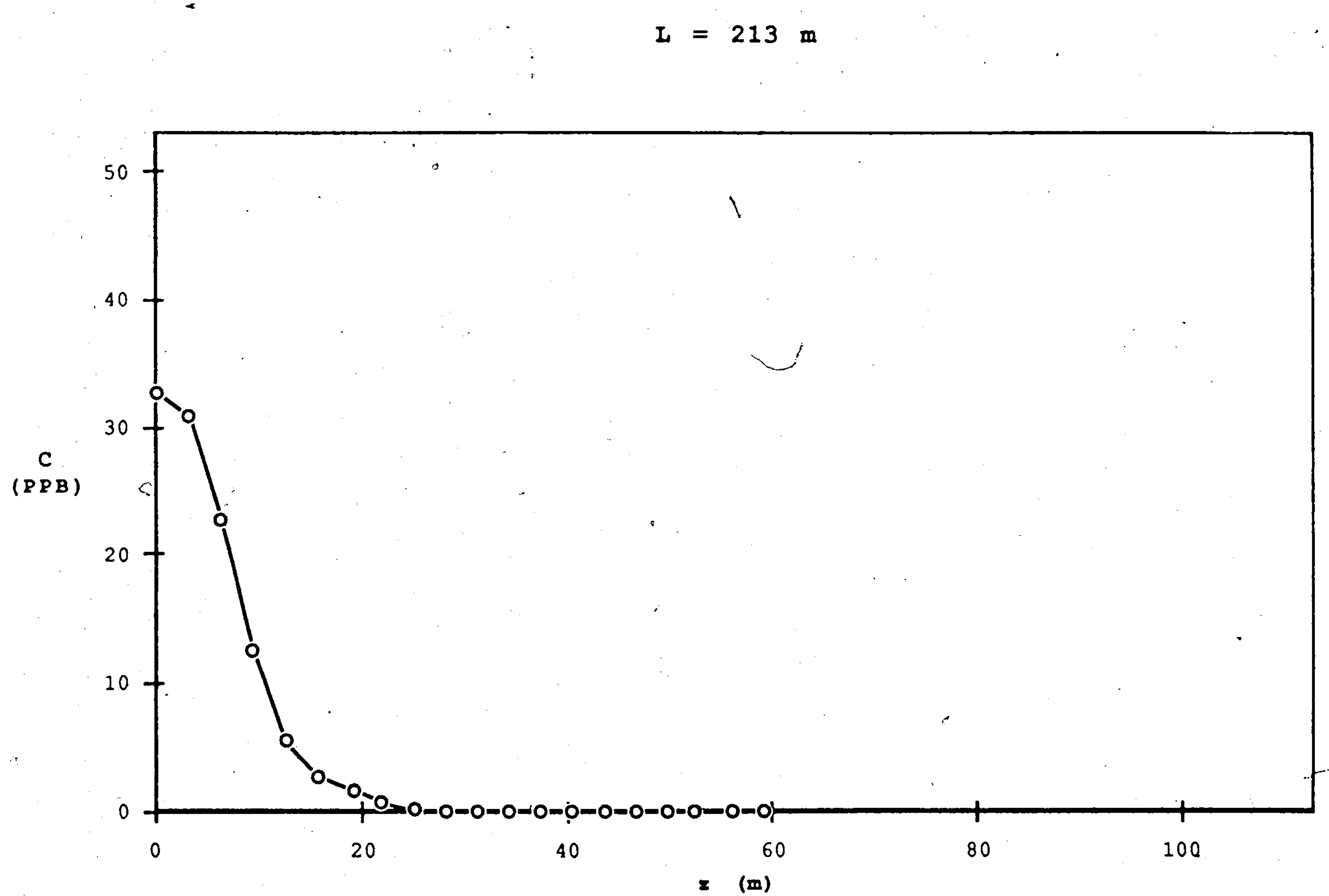


Figure 5.14 Concentration Profile at L = 213 m, Grand River

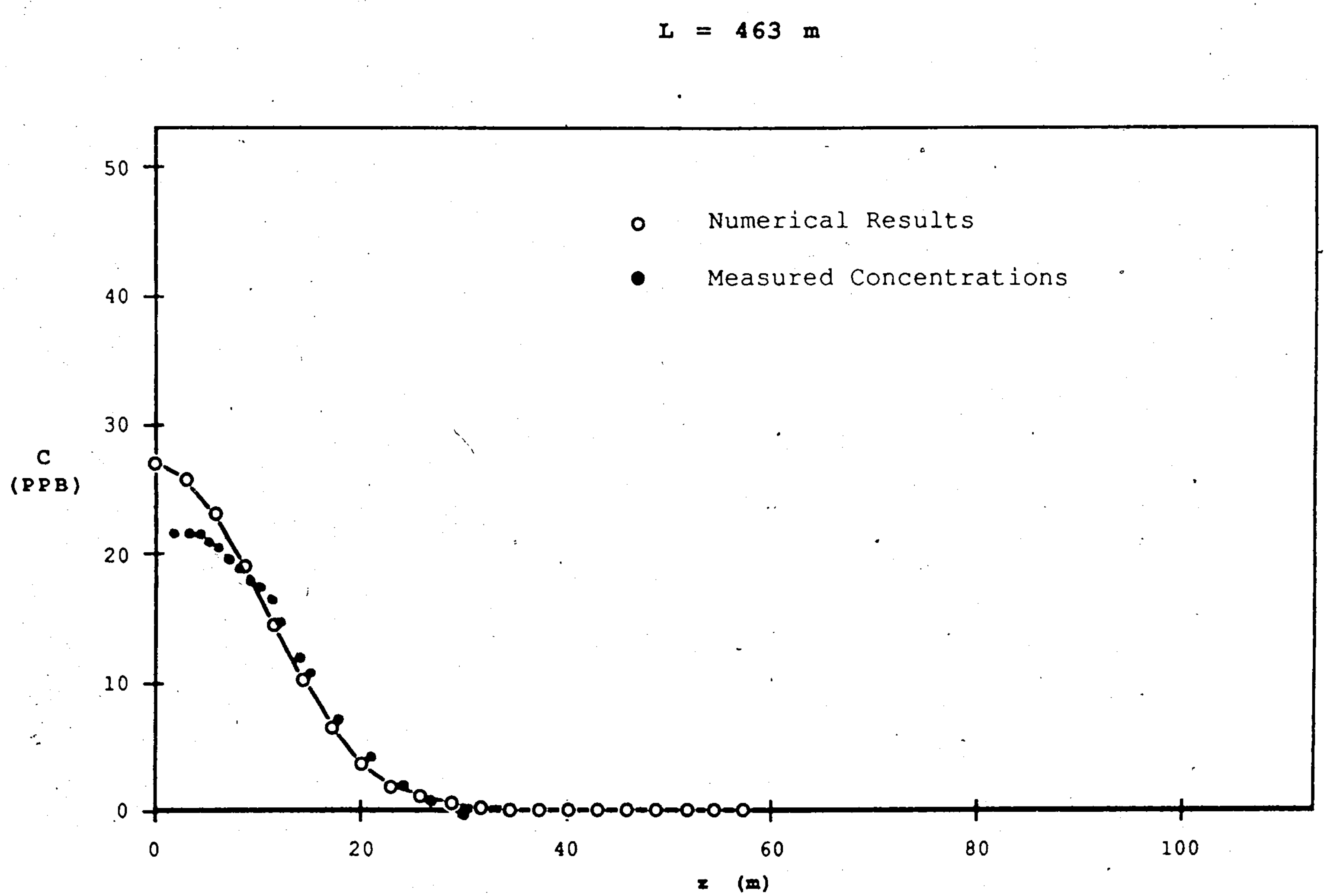


Figure 5.15 Concentration Profile at L = 463 m, Grand River

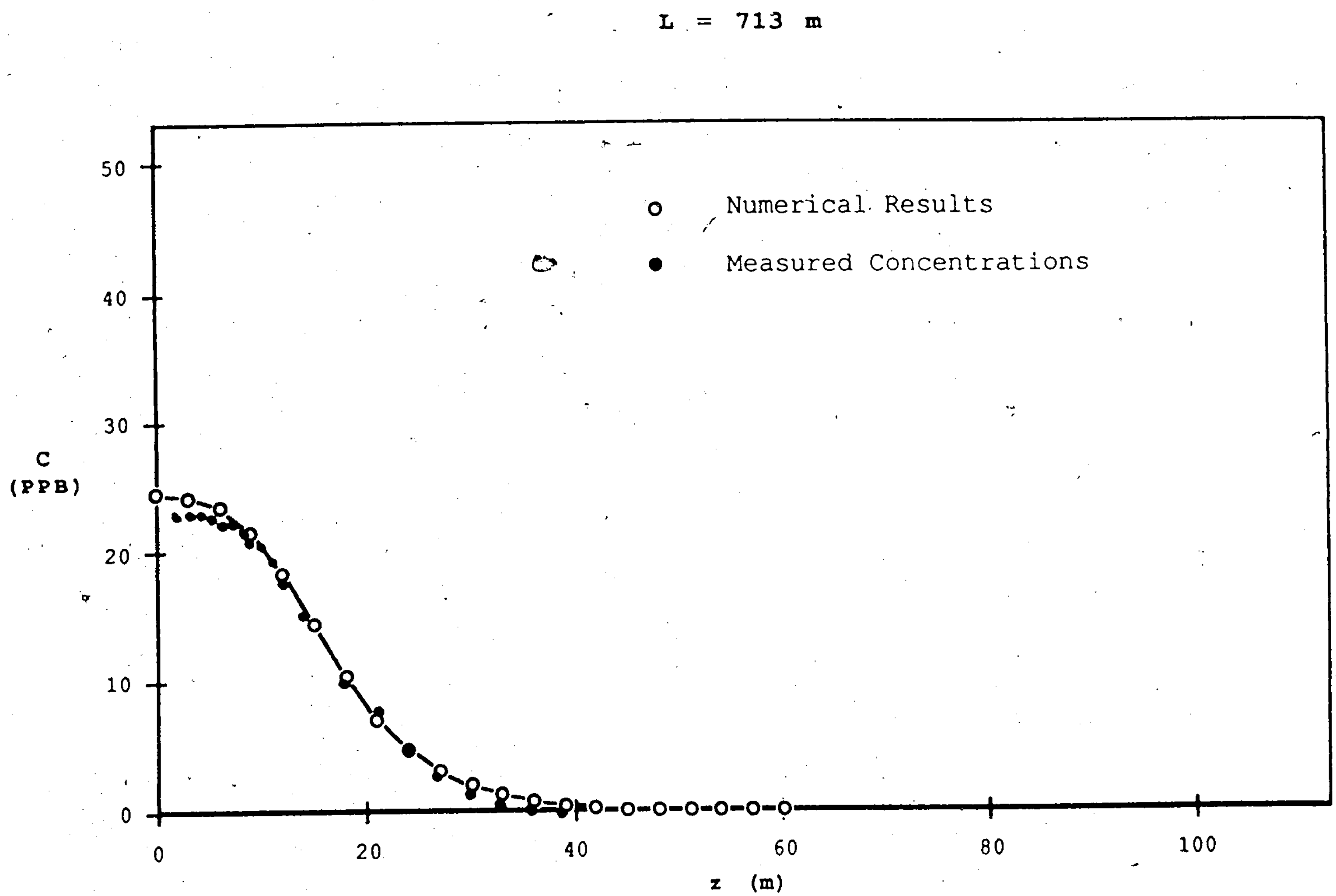


Figure 5.16 Concentration Profile at L = 713 m, Grand River

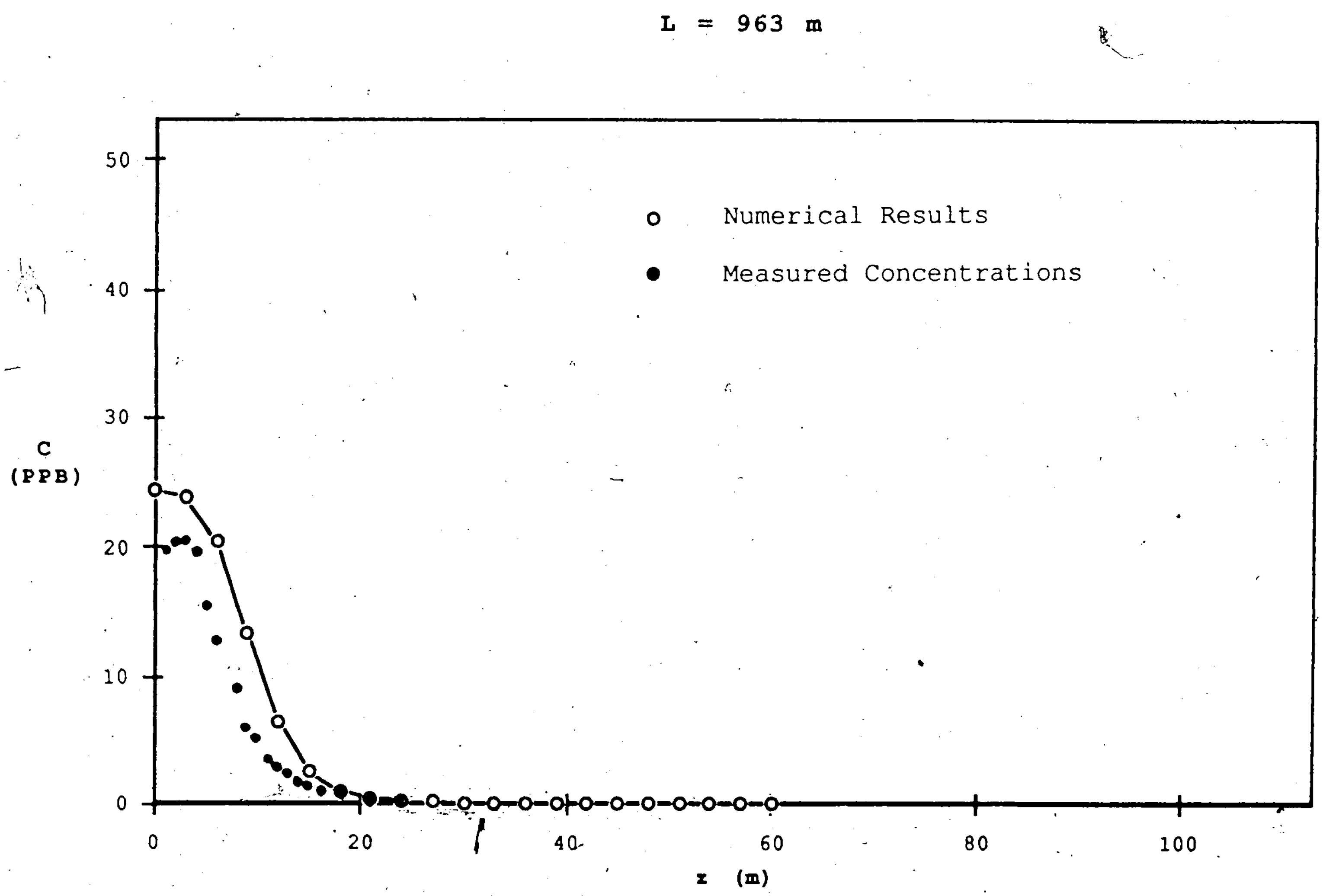


Figure 5.17 Concentration Profile at L = 963 m, Grand River

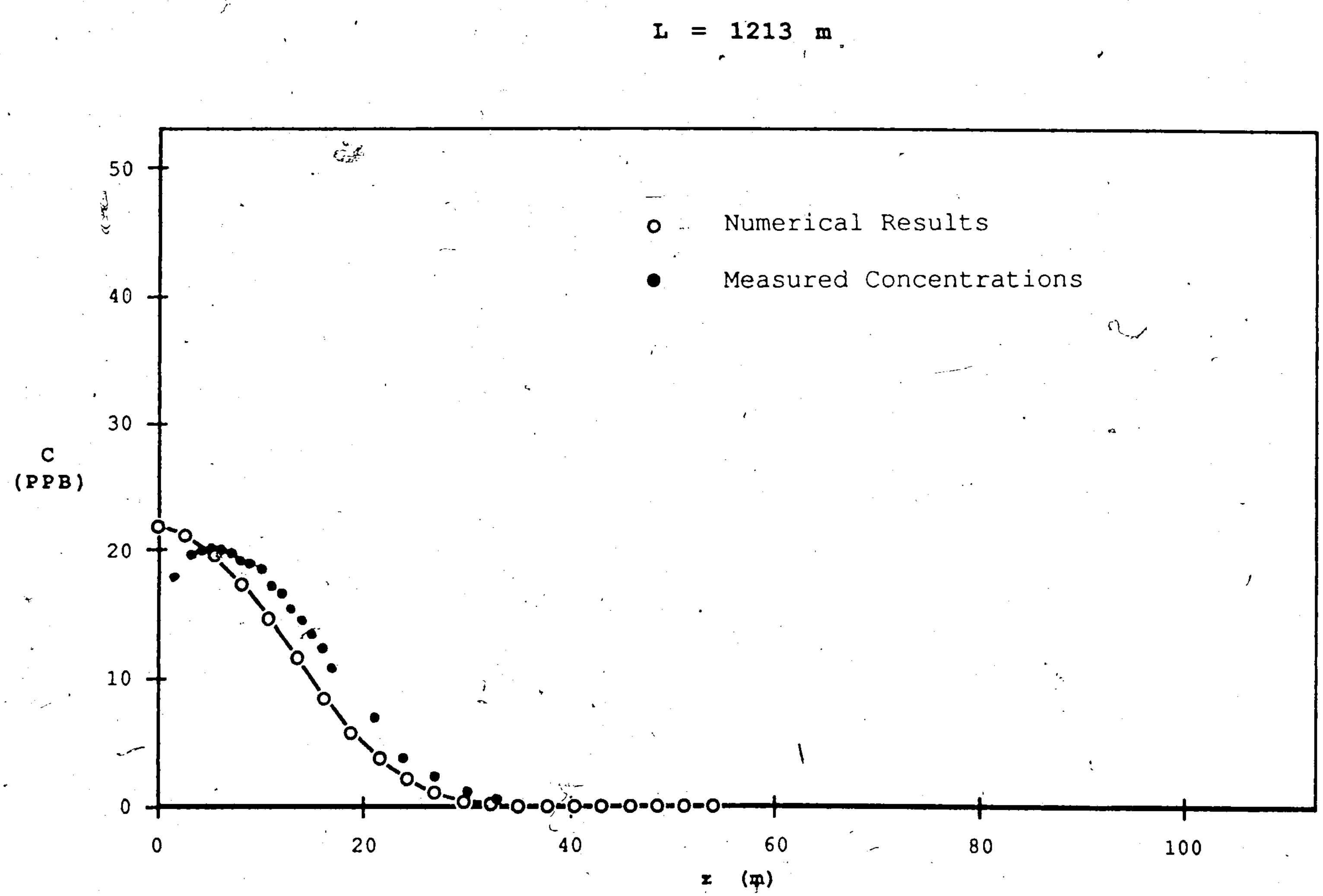


Figure 5.18 Concentration Profile at L = 1213 m, Grand River

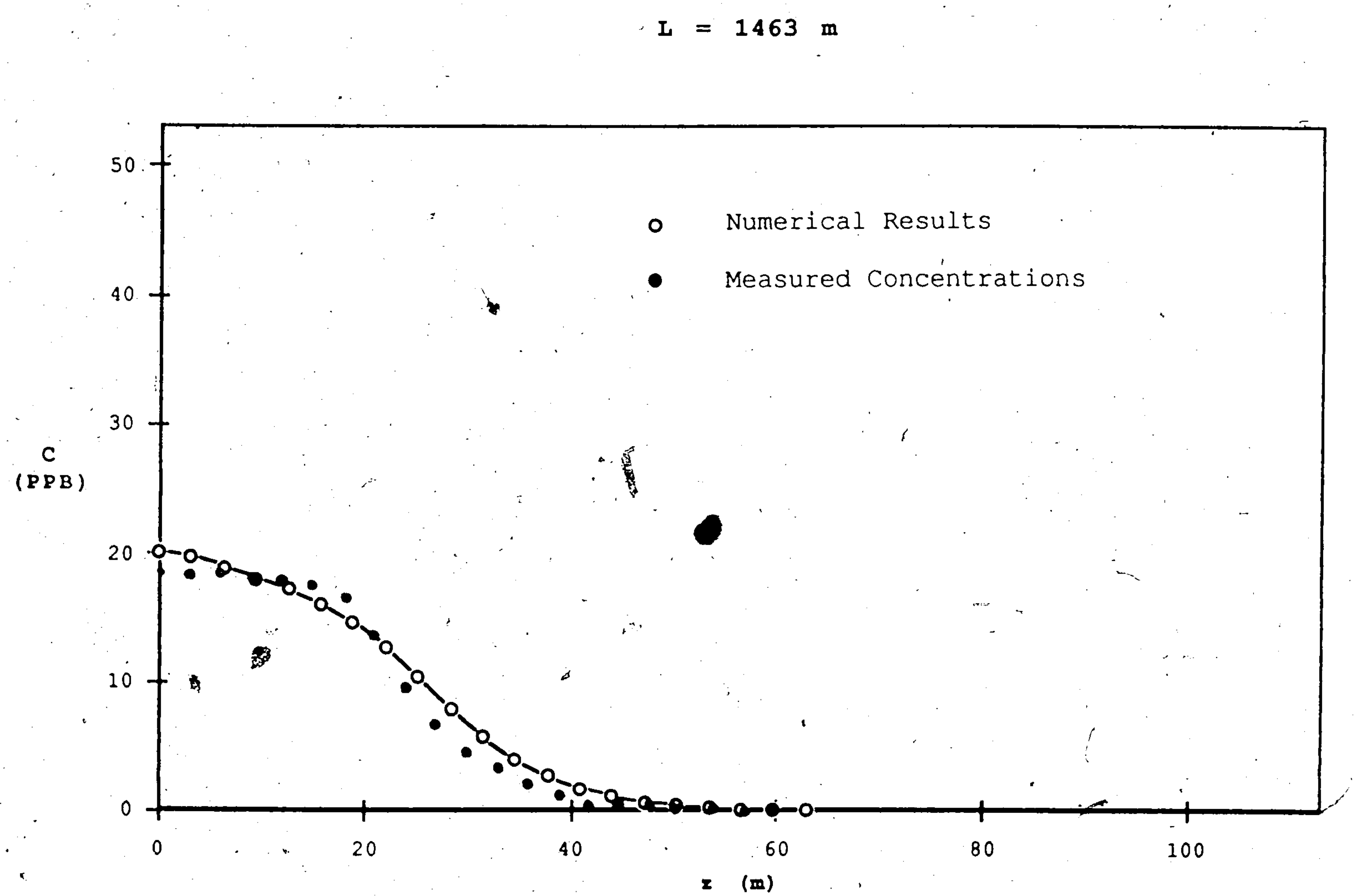


Figure 5.19 Concentration Profile at L = 1463 m, Grand River

5.2 Application to River Slug Tests

One of the most important concerns that the engineer is faced with is the accidental release of a contaminant into a river. The result of such a release is the possibility of large concentrations of pollutant entering the drinking water inlets situated on the river. To understand and predict the behavior, a 'slug test' is performed on most commercially used rivers. The ability to predict the behavior of such releases into the river by a numerical model is very desirable. However the physics of the processes involved in the predictive models is not yet fully understood.

An important component of the physics that is yet to be understood is the process of dispersion. Although there are models available such as the Elder model for logarithmic velocity profiles discussed in Section 2.1.4.1 (1959), they are limited to the final region. S. Beltaos(1980) also proposed a model to predict the dispersive behavior of a slug in the initial region which could reconstruct the skewed distribution found in reality. However his model basically amounted to an empirical fit to the processes observed, unrelated to any physical arguments.

This lack of physical understanding of the dispersion processes has made it impossible to reproduce the measured skewed distributions. However for the sake of application, the current finite element model was applied to such a case.

S. Beltaos at Alberta Research Council undertook a series of tests on the Athabasca River below Fort McMuarry. Among these was a test performed under an ice covered river reach. The detailed information on this particular test was reported in a Alberta Research Council Report (1978).

The reach considered in this report was discretized into a mesh 9 elements wide by 100 elements long. The discretized reach is shown in a plot in Figure 5.20. Included in Figure 5.20 is a plot of the depth averaged velocities derived from measurements and interpolation. A plot of the initial concentration slug is also shown in this figure.

A summary of the hydraulic data and the model used for the eddy diffusivity is shown in Table 5.2. Using the reach averaged velocity of 0.49 m/s, an average Δx = 118 m and a Courant number of 0.125, a time interval of 30 seconds was realized. An increased level of accuracy was needed due to the unsteady nature of the problem. Earlier investigations (Section 4.1) had shown that QUPG gave an adequately accurate solution to the unsteady advection of a pulse, thus QUPG was used to solve this problem. CUPG was not used due to a limitation on the memory available to solve the problem.

The contour plots for the results at times 100 min., 200 min., and 300 min., are shown in Figure 5.21. A plot of the history of the concentration across the channel at x = 6330 is shown in Figure 5.22.

The results show the potential of numerical solution to the unsteady problem in its ability to provide the engineer a wealth of data. The model allows the investigator to observe anomalies such as of pockets of relatively large concentrations attach to some banks as the one shown in Figure 5.21 on the left bank.

There exists a need to develop a physically sound analysis for the modelling of dispersion in rivers. Without this base, the advancement in the use of numerical solutions in aiding the engineer in assessing behavior of transient slugs in rivers is limited.

TRANSECT NUMBER	DISTANCE FROM SOURCE (m)	CHANNEL WIDTH W (m)	AVERAGE DEPTH H (m)	MEAN VELOCITY U (m/s)	E_long. Calcu. (m /s)	E_lat. Calcu. (m /s)
1	0	225.0	1.342	0.30	0.069	0.046
2	1900	166.5	2.327	0.35	0.135	0.090
3	3100	427.0	1.252	0.26	0.062	0.041
4	4300	288.0	1.555	0.33	0.073	0.049
5	6300	298.5	1.410	0.35	0.063	0.042
6	7800	130.0	2.595	0.41	0.168	0.112
7	9700	162.0	2.727	0.40	0.170	0.113
8	11800	288.0	1.836	0.28	0.091	0.061

AN AVERAGE U* WAS USED
DEPTH WAS ALLOWED TO VARY ACROSS THE CHANNEL AND THEREFORE EVERY WHERE

Slope = 0.00014
Average U* = 0.037 m/s
Ezz = 0.58 U* h
Exx = 0.87 U* h

Table 5.2 Hydraulic Data for the Athabasca River below Fort McMurray (according to S. Beltaos, 1978)

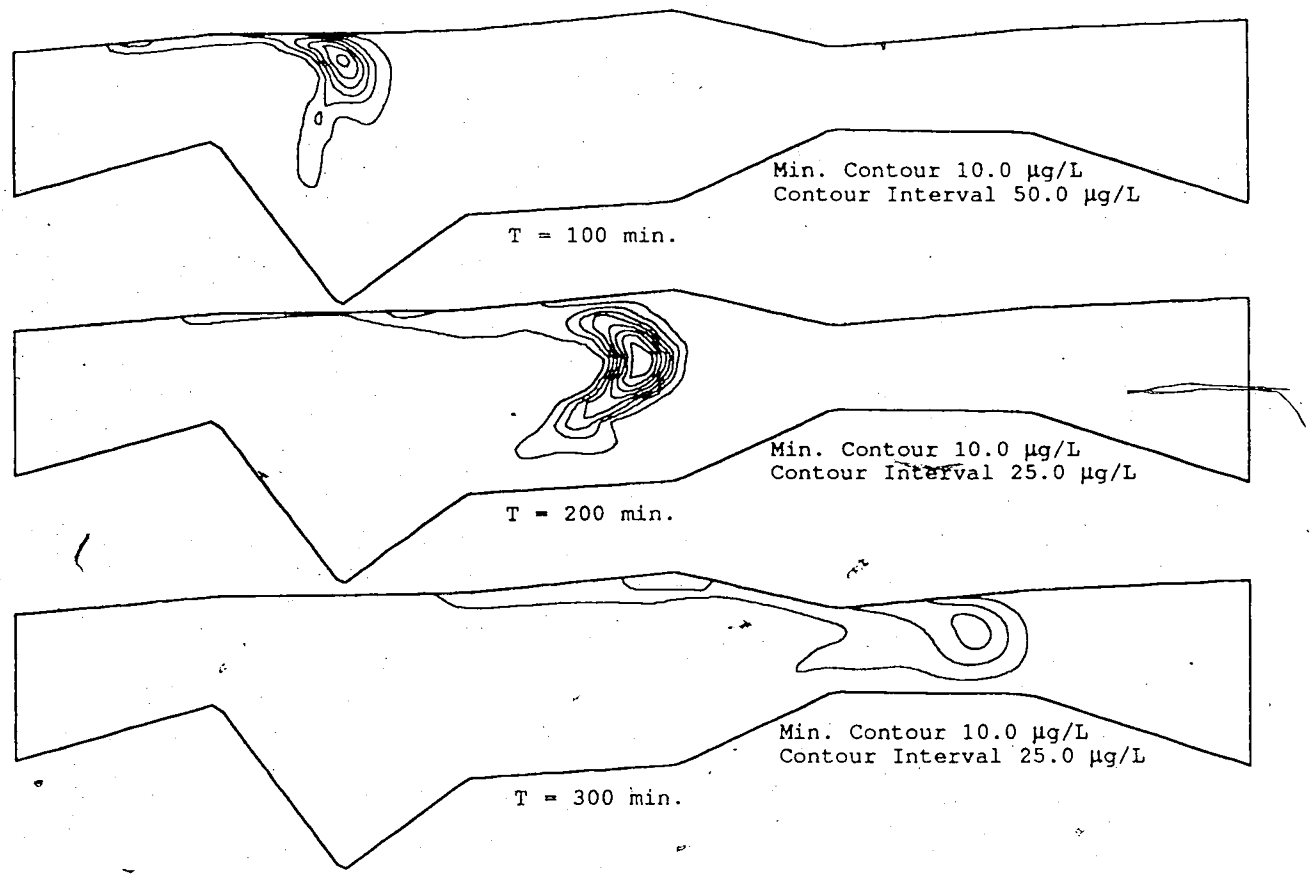


Figure 5.21 Concentration Contours for a Slug test

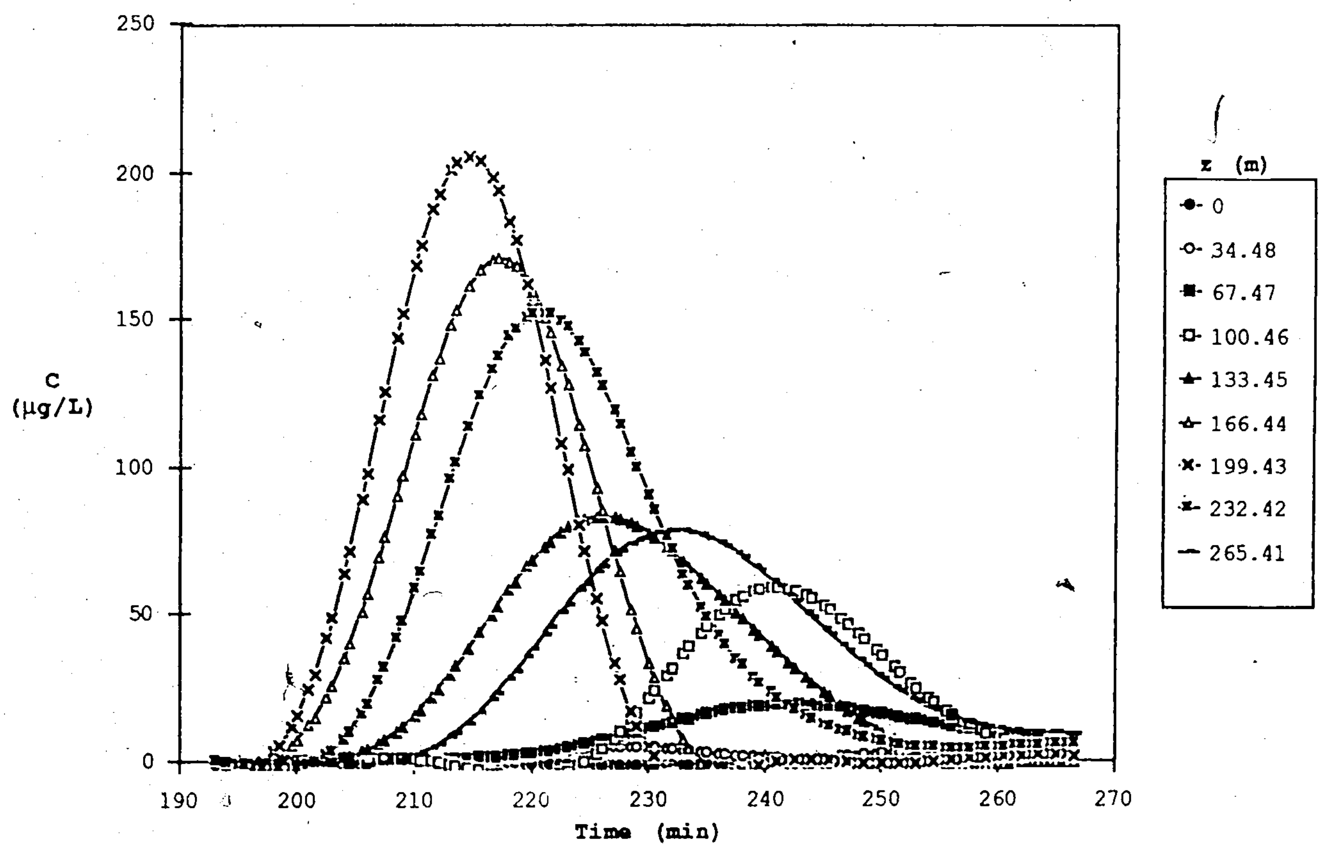


Figure 5.22 Computed Concentrations at x = 6360 m, Athabasca River

6. CONCLUSIONS

The use of the finite element method to solve the depth averaged pollutant equation gives the engineer an added degree of freedom in his ability to accurately present the picture of the behavior of pollutants in rivers.

The new upwinding elements, developed by Dr. P. M. Steffler, were found to be far superior than most of the standard finite difference schemes used in the modelling of the pollutant conservation equation. They were also found to be better than the standard linear finite elements. However the use of these elements has to be justified by a need to model flows with high Peclet numbers, as they require extra computational effort to solve.

The solutions of the two dimensional advection equation for a circular flow field, suggests that the elements are best suited for conditions where the direction of the flow is not known at the start. The ability of these elements to upwind in any desired direction is best realized by circulation problems. A situation like this exist in the modelling of effluent discharges in coastal waters, where the majority of the flow field is in circular motion and in an alternating direction. It is recommended that these new elements be applied to problems of outfall discharges in coastal waters.

The use of a variable eddy diffusivity in an idealized channel was found to produce a more realistic solutions than an average eddy diffusivity. The finite element model with the freedom to specify a variable eddy diffusivity, proved to be adequate in reproducing the measured concentrations for the Grand River reach.

The need to develop a firm physical understanding of the dispersion processes in a river slug test, was pointed out by the results of the numerical model applied to a 'slug' test. The solution presented by the model showed the large potential it has in providing the engineer with a wealth of information.

REFERENCES

Baker A. J., 1983, Finite Element Computational Fluid Mechanics. Hemisphere Publishing Corporation, New York, N.Y.

Beltaos S., 1978, Transverse Mixing in Natural Streams, Report No. SWE78-1. Alberta Research Council, Edmonton, Alberta.

Beltaos S., 1980, "Longitudinal Dispersion in Rivers". ASCE Journal of the Hydraulics Division. Vol. 106(HY10), pp. 1607-1625.

Elder J. W., 1959, "The dispersion of marked fluid in turbulent shear flow", Journal of Fluid Mechanics, Vol. 5, pp. 544-560.

Elhdi N., Harrington A., Hill I., Lau Y. L., Krishnappan B. G., 1984, "River mixing-A state-of-art report", Canadian Journal for Civil Engineer, Vol. 11, pp. 585-609.

Fisher H. B., List E. J., Koh R. C., Berger G. N., and Brooks N. H., 1979, Mixing in Inland and Coastal Waters. Academic Press, New York, N. Y.

Harden T. O. and Shen H. T., 1979, "Numerical simulation of mixing in natural rivers", ASCE Journal of the Hydraulic Division, Vol. 105(HY4), pp. 393-408.

Krishnappan B. G. and Lau Y. L., 1977, "Transverse mixing in meandering channels with varying bottom topography", Journal of Hydraulic Research, Vol 15(4), pp. 351-371.

Krishnappan B. G. and Lau Y. L., 1981, "Modelling transverse mixing in natural streams". ASCE Journal of the Hydraulic Division, Vol. 107(HY2), pp. 209-226.

Krishnappan B. G. and Lau Y. L., 1983, User's manual for model RIVMIX, National Water Research Institute, Environment Canada, Burlington, Ontario.

Lapidus L. and Pinder G.F., 1982, Numerical Solution of Partial Differential Equations in Science and Engineering, John Wiley and Sons, Toronto, Canada, 677 pages.

Leimkuhler W., Connor J., Wang J., Christodouluou G. and Sundgren S., 1975, "Two Dimensional Finite Element Dispersion Model". ASCE Symposium on Modeling Techniques, San Fransico, California, pp. 1467-1486.

Leonard B. P., 1979, "A Stable and Accurate convective modelling procedure based on quadratic Upstream Interpolation", J. Comp. Mtd. Appl. Mech. and Eng., Vol 19, pp. 59-98.

Putz G., 1984, TRSMIX Users Manual, Environmental Engineering Technical Report 84-2, University of Alberta, Edmonton, Alberta.

Rajaratnam N., 1976, Turbulent Jets, Developements in Water Science, Elsevier Scientific Publishing Company, New York, N. Y., 304 pages.

Sauvaget P., 1985, "Dispersion in Rivers and Coastal Waters - 2. Numerical Computation of Dispersion", Developments in Hydraulic Engineering, Vol 3, Elsevier Applied Science Publishers, New York, N.Y., pp. 39-78.

Sayre W. W. and Chang F. M., 1968, A Laboratory Investigation of the Open Channel Dispersion Process for Dissolved, Suspended and Floating Dispersants, U. S. Geological Survey Professional Paper 433-E.

Sayre W. W., 1973, "Natural mixing processes in Rivers", Environmental impact on Rivers, Edited by H. W. Shen, H. W. Shen, P. O. Box 606, Fort Collins, CO, U. S. A., Chapter 6.

Steffler P. M., 1988, "Upwind Basis Finite Elements for Convective Dominated Problems", To be Published in the International Journal for Numerical Methods in Fluids.

Stone H. L. and Brian P. L. T., 1963, "Numerical Solution of Convective Transport Problems", AIChE Journal, Vol. 9, pp. 681-688.

Taylor G. I., 1921, "Diffusion by Continuous Movements", the Proceedings of the London Mathematical Society, Vol. 20, pp. 196-212.

Yotsukura N. and Sayre W. W., 1976, "Transverse mixing in Natural Channels", Water Resources Research, Vol. 12(4), pp. 695-704.

APPENDIX 1

Program CDFEM Listing

Files Included Are

Page

fe.h

```
#include "math.h"
#include "stdio.h"

long FreeMem() ;      /*     NON STANDARD FUNCTION        */

char   *calloc(), *lmalloc() ; /*  NON STANDARD FUNCTION        */

#define  MNSF         36
#define  MRNSF        4
#define  MDOF         144
#define  MGPTS        16
#define  NDIMS        2
#define  NPARAMS      8
#define  NBPARAMS     2
#define  NVARS        1
#define  NKEQNS       20
#define  MNXS         40

struct shapefuncs {
       int     dof ;
       double f[MNSF] ;
       double        dfdr[MNSF] ;
       double dfds[MNSF] ;
       double dfdt[MNSF] ;
       } ;

struct govfuncs {
       int     dof ;
       double detJ ;
       double w ;
       double p[NPARAMS] ;
       double f[MNSF] ;
       double dfdx[MNSF] ;
       double dfdy[MNSF] ;
       double dfdz[MNSF] ;
       } ;

struct gausspts {
       double x ;
       double y ;
       double z ;
       double w ;
       } ;

struct node  {
       int           n ;
       int           i ;
       int           ui ;
       int           fxc    ;
       double        x[NDIMS]       ;
       double        u[NVARS]       ;
       float         p[NPARAMS]     ;
       float         f[NDIMS]       ;
       struct node   *nextnp        ;
       } ;

struct element  {
       int           n ;
       int           vtype ;
       int           gtype ;
       int           nnds ;
       struc         *nps[MNSF];
       float         p[NPARAMS] ;
```

```
        double          *matrices ;
        float           elinfo ;
        struct element        *nextelp ;
        } ;

struct belement  {
        int             n ;
        int             vtype ;
        int             gtype ;
        int             nnds ;
        struct node     *nps[MRNSF] ;
        double          p[NBPARAMS] ;
        int             bcs[NVARS] ;
        struct belement       *nextbelp ;
        } ;

struct control {
        int     trans ;
        int     dims ;
        int     vars ;
        int     params ;
        int     bparams ;
        int     Keqns[NKEQNS] ;
        int     nodes ;
        int     elms ;
        int     belms ;
        int     frnds ;
        int     ukns ;
        int     sym ;
        } ;

struct eqnset {
        double *Kp ;
        double *Lp ;
        double *Fp ;
        long   *diagp ;
        int    neqns ;
        } ;

struct pointers {
        struct node          *N ;
        struct element       *El ;
        struct belement *B ;
        struct node    **iptrs ;
        double          *S ;
        double          *M ;
        }   ;

struct elmpointers {
        double *K ;
        double *F ;
        double *S ;
        double *M ;
        int    n  ;
        }  ;

struct fxnodes {
        int    i ;
        int    j[MDOF] ;
        double k[MDOF]
        double f ;
        } ;
```

```
struct transient {
        int     nsteps ;
        double dtfac ;
        double t ;
        double dt ;
        double theta ;
        } ;

struct RegMesh {
        int             nx ;
        int             ny ;
        int             nbx ;
        int             nby ;
        int             elsinbx[MNSF] ;
        int             elsinby[MNSF] ;
        int             eltype ;
        int             maptype ;
        struct control        con ;
        struct blockmap       *firstmap ;
        } ;

struct blockmap {
        float   xnodes[MNSF] ;
        float   ynodes[MNSF] ;
        float   h[MNSF] ;
        float   u[MNSF] ;
        float   v[MNSF] ;
        float   E_x[MNSF] ;
        float   E_y[MNSF] ;
        float   E_xy[MNSF] ;
        float   E_yx[MNSF] ;
        } ;

struct cross_section {
        float                   x[MNXS] ;
        float                   z[MNXS] ;
        float                   h[MNXS] ;
        float                   u[MNXS] ;
        float                   v[MNXS] ;
        float                   E_long[MNXS] ;
        float                   E_lat[MNXS] ;
        struct cross_section         *next_x ;
        } ;
```

femain.c

```
#include        "fe.h"
#include        "time.h"



char  *strcat(), *strncat();

main()
{
        char            ain ;
        long            t_bytes ;
        int             nin, errcode, i ;
        long int        t1, t2 ;
        void            MaxApplZone() ;/*NON STANDARD FUNCTION */

        MaxApplZone() ;

        init_gp() ;

        printf(" Input command (s,t,i,n,m,l,b,o,p,t,c,r,y) and number\n") ;
        ain = 'b' ;

        while(ain != 'q'){
                time(&t1) ;
                t_bytes = FreeMem() ;/*    NON STANDARD FUNCTION */
                printf(" Total Memory Available =%ld\n",t_bytes) ;
                printf("> ");
                scanf(" %c %d",&ain,&nin) ;
                switch (ain) {

                        case 's' :
                                errcode = steady_state(nin,1) ;
                                break ;
                        case 'i' :
                                errcode = input(nin) ;
                                break ;
                        case 'n' :
                                errcode = test_mesh(nin) ;
                                break ;
                        case 'm' :
                                errcode = test_map(nin) ;
                                break ;
                        case 'l' :
                                errcode = list_vars(nin) ;
                                break ;
                        case 'b' :
                                errcode = set_bc(nin) ;
                                break ;
                        case 'o' :
                                errcode = output(nin) ;
                                break ;

                        case 'p' :
                                errcode = special_out(nin)
                                break ;
                        case 't' :
                                errcode = test_trans(nin) ;
                                break ;
                        case 'c' :
                                errcode = test_contour(nin) ;
                                break ;
                        case 'r' :
                                errcode = test_real_stream(nin) ;
```

```
                        break ;
                case 'y' :
                        errcode = test_analytical(nin) ;
                        break ;
                }
                time(&t2) ;     /*      NON STANDARD FUNCTION */
                printf(" Time = %s",ctime(&t2)) ;
                                /*      NON STANDARD FUNCTION */
        }
}

FILE    *get_fptr(code)
int             code ;

{
        FILE    *fp, *fopen() ;
        char    s[63] ;

        if(code == 1) {
                printf("\n Input Data File Name  ") ;
                scanf(" %s",s) ;
                fp = fopen(s,"r") ;
        }
        else
                fp = stdin ;
        return( fp ) ;
}

FILE    *put_fptr(code)
int             code ;

{
        FILE    *fp, *fopen() ;
        char    s[63] ;


        if(code == 1) {
                printf("\n Input Output File Name  ") ;
                scanf(" %s",s) ;
                fp = fopen(s,"w") ;
        }
        else if(code == 2)
                fp = fopen("CDFEM1.out","w") ;
        else if(code == 3)
                fp = fopen("CDFEM2.out","w") ;
        else if(code == 4)
                fp = fopen("CDFEM3.out","w") ;
        else if(code == 5)
                fp = fopen("CDFEM4.out","w") ;
        else if(code == 6)
                fp = fopen("CDFEM5.out","w") ;
        else if(code == 7)
                fp = fopen("CDFEM6.out","w") ;
        else
                fp = stdin ;

        return( fp ) ;
}

int             StartMes(astr,bstr,cstr,dstr)
char            *astr, *bstr, *cstr, *dstr ;

{
```

```
        printf("%s\n%s\n%s\n%s\n",astr,bstr,cstr,dstr) ;
        return(0);
}

int             UpDateMes(astr,bstr, cstr, dstr)
char            *astr, *bstr, *cstr, *dstr ;

{
        printf("%s\n", dstr) ;
        return(0);
}

int             EndMes()

{
        return(0);
}
int             InsertMes(astr)
char            *astr ;

{
        printf("%s\n",astr) ;
        return(0);
}
```

feio.c

```
#include "fe.h"
#include "ctype.h"

extern struct control           N ;
extern struct pointers          gp;
extern struct transient     tvals ;
extern struct RegMesh           Mesh ;

int                         Nndso, Nelmso, Nbelmso, Mesh_code ;

int         input(nin)
int         nin ;

{
        int     err,i, fer ;
        FILE    *fptr, *get_fptr() ;

        if((fptr = get_fptr(nin) ) != NULL) {
                if( (err = get_control(fptr)) == 0) {
                        put_control(stdout) ;
                        for(i=0;i<N.nodes;i++)
                                get_node(fptr,i) ;
                        for(i=0;i<N.elms;i++)
                                get_elm(fptr,i) ;
                        for(i=0;i<N.belms;i++)
                                get_belm(fptr,i) ;
                        if(fptr != stdin) fclose(fptr) ;
                                set_bvalues() ;
                }
                else
                        printf(" Control Variable Error %d\n",err) ;

                if((fer = ferror(fptr))!= NULL || (fer != 0) ){
                        printf(" Read error in routine Input \n Error ");
                        clearerr(fptr) ;
                        if(fptr != stdin) fclose(fptr) ;
                        return(fer) ;
                }
        }
        else {
                printf(" Unable to open that file\n") ;
                err = -1 ;
                return(err);
        }
}

int     get_control(f)
FILE    *f ;

{
        int     i, j, k, in, ii ;
        long    bigint ;
        char    *calloc(), *lmalloc() ; /*  NON STANDARD FUNCTION        */

        j = 0 ;
        for(i=0;i<1;i++) {
                j++ ;
                N.trans = get_int(f) ;
                if(N.trans < 0 || N.trans > 1) break ;
                j++ ;
                if(N.trans == 1)
                        get_trans(f) ;
                N.dims = get_int(f) ;
```

```
                        if(N.dims < 1 || N.dims > NDIMS) break ;
                        j++ ;
                        N.vars = get_int(f) ;
                        if(N.vars < 1 || N.vars > NVARS) break ;
                        j++ ;
                        for(ii=0;ii<((N.vars+1)*N.vars);ii++)
                                N.Keqns[ii] = get_int(f) ;
                        j++ ;
                        N.params = get_int(f) ;
                        if(N.params < 0 || N.params > NPARAMS) break ;
                        j++ ;
                        N.bparams = get_int(f) ;
                        if(N.bparams < 0){
                                Mesh_code = -1 ;
                                N.bparams *= -1 ;
                        }
                        else
                                Mesh_code = 1 ;
                        if(N.bparams < 0 || N.bparams > NBPARAMS) break ;
                        j++ ;
                        if(gp.N != NULL) {
                                clear_data() ;
                                free(gp.N) ;
                                free(gp.iptrs) ;
                        }
                        N.nodes = get_int(f) ;
                        Nndso = N.nodes ;
                        bigint = (5 * N.nodes)/4 * sizeof(struct node *) ;
                        if(N.nodes < 2 || (gp.iptrs = (struct node **) lmalloc(bigint))
        == NULL) break ;
                        bigint = (long int)N.nodes *(long int)sizeof(struct node) ;
                        if(N.nodes < 2 || (gp.N = (struct node *) lmalloc(bigint)) ==
        NULL) break ;/*         NON STANDARD FUNCTION          */
                        j++ ;
                        N.elms = get_int(f) ;
                        Nelmso = N.elms ;
                        bigint = (long int)N.elms * (long int)sizeof(struct element) ;
                        if(gp.El != NULL)
                                free(gp.El) ;
                        if(N.elms < 1 || (gp.El = (struct element *) lmalloc(bigint)) ==
        NULL) break ;/*         NON STANDARD FUNCTION          */
                        j++ ;
                        N.belms = get_int(f) ;
                        Nbelmso = N.belms ;
                        bigint =(long int) N.belms *(long int) sizeof(struct belement) ;
                        if(gp.B != NULL)
                                free(gp.B) ;
                        if(N.belms < 1 || (gp.B = (struct belement *) lmalloc( bigint))
        == NULL) break ;/*      NON STANDARD FUNCTION          */

                        if(Mesh_code == -1 )
                                get_mesh_info(f) ;

                        printf(" Memory left       \n",FreeMem()) ;/*      NON STANDARD
        FUNCTION        */

                        j = 0 ;
                }
                return(j) ;
        }

        int     get_mesh_info(f)
        FILE    *f ;
```

```
{
        Mesh.nx = get_int(f) ;
        Mesh.nbx = get_int(f) ;
        Mesh.ny = get_int(f) ;
        Mesh.nby = get_int(f) ;
        return(0) ;
}

int     get_trans(f)
FILE    *f ;
{
        double get_dbl() ;

        tvals.nsteps = get_int(f) ;
        tvals.dtfac = get_dbl(f) ;
        tvals.t = get_dbl(f) ;
        tvals.dt = get_dbl(f) ;
        tvals.theta = get_dbl(f) ;
        return(0) ;
}


int     clear_data()

{
        int             i ;
        struct node     *np1, *np2 ;
        struct element          *elp1, *elp2 ;
        struct belement *belp1, *belp2 ;

        np1 = gp.N ;
        for(i=0;i<N.nodes;i++) {
                np2 = np1->next[illegible]
                if(i >= Nndso)
                        free(np1) ;
                np1 = np2 ;
        }
        belp1 = gp.B ;
        for(i=0;i<N.belms;i++) {
                belp2 = belp1->nextbelp ;
                if(i >= Nbelmso)
                        free(belp1) ;
                belp1 = belp2 ;
        }
        elp1 = gp.El ;
        for(i=0;i<N.elms;i++) {
                elp2 = elp1->nextelp ;
                if(elp1->matrices != NULL)
                        free(elp1->matrices) ;
                if(i >= Nelmso)
                        free(elp1) ;
                elp1 = elp2 ;
        }
        return(0) ;
}

int     get_node(f,i)
FILE    *f ;
int     i ;

{
        int             j ;
        double          get_dbl() ;
```

```
	struct node	*nodep ;

	nodep = gp.N + i ;
	nodep->n = get_int(f)	;
	nodep->i = i ;
	nodep->fxc = 0 ;
	gp.iptrs[i] = nodep ;
	for(j=0;j<N.dims;j++)
		nodep->x[j] = get_dbl(f) ;
	for(j=0;j<N.params;j++)
		nodep->p[j] = get_dbl(f) ;
	for(j=0;j<N.vars;j++)
		nodep->u[j] = get_dbl(f) ;
	nodep->nextnp = nodep + 1 ;
	return(0) ;
}

int	get_elm(f,i)
FILE	*f ;

{
	int		j, jj, nnds, n, rc ;
	struct element	*elmntp ;
	struct node	*np ;

	elmntp = gp.El + i ;
	rc = 0 ;
	elmntp->n = get_int(f) ;
	elmntp->vtype = get_int(f);
	elmntp->gtype = get_int(f);
	elmntp->nnds =	nsf(elmntp->vtype) ;
	nnds = nsf(elmntp->gtype);
	if(nnds > elmntp->nnds) elmntp->nnds = nnds ;
	if(elmntp->vtype == 228)
		elmntp->nnds = 6 ;
	if(elmntp->vtype == 229)
		elmntp->nnds = 16 ;
	if(elmntp->vtype == 233)
		elmntp->nnds = 8 ;
	if(elmntp->vtype == 230)
		elmntp->nnds = 6 ;
	if(elmntp->vtype == 239)
		elmntp->nnds = 36 ;
	if(elmntp->vtype == 129)
		elmntp->nnds = 4 ;
	if(elmntp->vtype == 139)
		elmntp->nnds = 6 ;
	for(j=0;j<elmntp->nnds;j++) {
		if((n = get_int(f)) >= 0) {
			np = gp.N ;
			for(jj=0;jj<N.nodes;jj++) {
				if(np->n == n)
					break ;
				np = np->nextnp ;
			}
			if(np == NULL) {
				printf(" Non-existent node %d referred to in
element %d\n",n,elmntp->n) ;
				rc = -1 ;
			}
			elmntp->nps[j] = np ;
		}
		else
```

```
                        elmntp->nps[j] = NULL ;
        }
        for(j=0;j<N.params;j++)
                elmntp->p[j] = get_dbl(f) ;
        elmntp->matrices = NULL ;
        if(i < N.elms - 1)
                elmntp->nextelp = elmntp + 1 ;
        else
                elmntp->nextelp = NULL ;
        return(rc) ;
}

int     get_belm(f,i)
FILE    *f ;

{
        int             j, jj, nnds, n, rc ;
        struct belement         *belmntp ;
        struct node     *np;

        belmntp = gp.B + i ;
        rc = 0 ;
        belmntp->n = get_int(f) ;
        belmntp->vtype = get_int(f);
        belmntp->gtype = get_int(f);
        belmntp->nnds =  nsf(belmntp->vtype) ;
        nnds = nsf(belmntp->gtype);
        if(nnds > belmntp->nnds) belmntp->nnds = nnds ;
        for(j=0;j<belmntp->nnds;j++) {
                n = get_int(f) ;
                np = gp.N ;
                for(jj=0;jj<N.nodes;jj++) {
                        if(np->n == n)
                                break ;
                        np = np->nextnp ;
                }
                if(np == NULL) {
                        printf(" Non-existent node %d referred to in element
%d\n",n,belmntp->n) ;
                        rc = -1 ;
                }
                belmntp->nps[j] = np ;
        }
        for(j=0;j<N.bparams;j++)
                belmntp->p[j] = get_dbl(f) ;
        for(j=0;j<N.vars;j++) {
                belmntp->bcs[j] = get_int(f) ;
        }
        if(i < N.belms - 1)
                belmntp->nextbelp = belmntp + 1;
        else
                belmntp->nextbelp = NULL ;
        return(rc) ;
}

double          get_dbl(fptr)
FILE    *fptr ;

{
        char    s[25] ;
        int     n ;
        double atof();

```

```
        n = getnumstr(fptr,s) ;
        return( atof(s) ) ;
}

int     get_int(fptr)
FILE    *fptr ;

{
        char    s[25] ;
        int     n ;
        double          atof(), d;

        n = getnumstr(fptr,s) ;
        d = atof(s) ;
        return( (int)  d ) ;
}

int     getnumstr(fptr,s)
char    *s ;
FILE    *fptr ;

{
        register int  c ;
        register char *cs ;

        cs = s ;
        while( (c = getc(fptr)) != EOF) {
                if((isdigit(c) != 0) || (c == '.') || (c == '-') || (c == '+')) {
                        *cs++ = c ;
                        while( (c = getc(fptr)) != EOF) {
                                if((isdigit(c) != 0) || (c == '.') || (c == 'e') ||
(c == 'E') || (c == '-') || (c == '+') )
                                        *cs++ = c ;
                                else
                                        break ;
                        }
                        *cs = '\0' ;
                        break ;

        }
        return(cs - s) ;
}

int             set_bvalues()

{
        int             i, j ;
        struct node     *nptr ;

        j = 0 ;
        nptr = gp.N ;
        for(i=0;i<N.nodes;i++) {
                nptr->i = i ;
                gp.iptrs[i] = nptr ;
                nptr = nptr->nextnp ;
        }
        N.ukns = N.nodes * N.vars ;
/*      printf(" Number of unknowns = %d\n",N.ukns) ;                   */
        return(j) ;
}

int             output(nin)
int             nin ;
```

```
{
        int             err,i,fer ;
        FILE            *fptr, *put_fptr() ;
        struct node     *np ;
        struct element          *elp ;
        struct belement *belp ;

        if((fptr = put_fptr(nin) ) != NULL) {
                if( (err = put_control(fptr)) == 0) {
                        fprintf(fptr,"\n Node Information \n");
                        fprintf(fptr,"\n Node #, Coordinates, Parameters,
Variables\n\n");
                        np = gp.N ;
                        for(i=0;i<N.nodes;i++) {
                                put_node(fptr,i,np) ;
                                np = np->nextnp ;
                        }
                        fprintf(fptr,"\n Element Information \n");
                        fprintf(fptr,"\n Element #, vtype, gtype, nodes\n\n");
                        elp = gp.El ;
                        for(i=0;i<N.elms;i++) {
                                put_elm(fptr,i,elp) ;
                                elp = elp->nextelp ;
                        }
                        fprintf(fptr,"\n Boundary Element #, vtype, gtype, nodes,
boundary condition codes\n\n");
                        belp = gp.B ;
                        for(i=0;i<N.belms;i++) {
                                put_belm(fptr,i,belp) ;
                                belp = belp->nextbelp ;
                        }
                }
                else
                        printf(" Control Variable Error %d\n",err) ;
        }
        else
                printf(" Unable to open that file\n") ;

        if((fer = ferror(fptr))!= NULL || (fer != 0) ){
                printf(" Write error in routine Output \n Error ");
                clearerr(fptr) ;
                if(fptr != stdin) fclose(fptr) ;
                return(fer) ;
        }
        else {
                if(fptr != stdin) fclose(fptr) ;
                return(err);
        }
}

int     put_control(f)
FILE    *f ;

{
        int     i, ii ;

        fprintf(f," Transient analysis = %d\n",N.trans) ;
        if(N.trans == 1){
                fprintf(f," Number of Time Steps = %d\n",tvals.nsteps) ;
                fprintf(f," Delta t Acceleration Factor = %5.3f\n",tvals.dtfac) ;
                fprintf(f," Time = %7.5f\n",tvals.t) ;
                fprintf(f," Delta t = %7.5f\n", tvals.dt) ;
```

```
                fprintf(f," Theta = %5.3f\n", tvals.theta) ;
        }
        fprintf(f," Dimensions = %d\n",N.dims) ;
        fprintf(f," Number of Variables = %d\n",N.vars) ;
        fprintf(f," [K] governing equation numbers \n") ;
                for(i=0;i<N.vars;i++) {
                        for(ii=0;ii<N.vars;ii++) {
                                fprintf(f,"\t %d ",N.Keqns[i*(N.vars+1) + ii]) ;
                        }
                        fprintf(f,"\t\t %d \n",N.Keqns[i*(N.vars+1) + N.vars]) ;
                }
        fprintf(f," Number of Parameters = %d\n",N.params) ;
        fprintf(f," Number of Boundary Parameters = %d\n",Mesh_code*N.bparams) ;
        fprintf(f," Number of Nodes = %d\n",N.nodes) ;
        fprintf(f," Number of Elements = %d\n",N.elms) ;
        fprintf(f," Number of Boundary Elements = %d\n",N.belms) ;
        if(Mesh_code == -1){
                fprintf(f," Number of Element in X direction = %d\n",Mesh.nx) ;
                fprintf(f," Number of Blocks in X direction = %d\n",Mesh.nbx) ;
                fprintf(f," Number of Element in Y direction = %d\n",Mesh.ny) ;
                fprintf(f," Number of Blocks in Y direction = %d\n",Mesh.nby) ;
        }
        fprintf(f, " \n") ;

        return(0) ;
}

int             put_node(f,i,nodep)
FILE            *f ;
int             i ;
struct node     *nodep ;

{
        int     j ;

        put_int(f,nodep->n)  ;
        for(j=0;j<N.dims;j++)
                put_dbl(f,(double) nodep->x[j]) ;
        for(j=0;j<N.params;j++)
                put_dbl(f,(double) nodep->p[j]) ;
        for(j=0;j<N.vars;j++)
                put_dbl(f,(double) nodep->u[j]) ;
        fprintf(f," \n");
        return(0) ;
}

int             put_elm(f,i,elmntp)
int             i ;
FILE            *f ;
struct element          *elmntp ;

{
        int             j;

        put_int(f,elmntp->n) ;
        put_int(f,elmntp->vtype);
        put_int(f,elmntp->gtype);
        for(j=0;j<elmntp->nnds;j++) {
                if(elmntp->nps[j] != NULL)
                        put_int(f,(elmntp->nps[j])->n) ;
                else
                        put_int(f,-1) ;
        }
```

```
        for(j=0;j<N.params;j++)
                put_dbl(f,(double) elmntp->p[j]) ;
        fprintf(f," \n");
        return(0) ;
}

int             put_belm(f,i,belmntp)
int             i ;
FILE            *f ;
struct belement *belmntp ;

{
        int             j;

        put_int(f,belmntp->n) ;
        put_int(f,belmntp->vtype);
        put_int(f,belmntp->gtype);
        for(j=0;j<belmntp->nnds;j++)
                put_int(f,(belmntp->nps[j])->n) ;
        for(j=0;j<N.bparams;j++)
                put_dbl(f,(double) belmntp->p[j]) ;
        for(j=0;j<N.vars;j++)
                put_int(f,belmntp->bcs[j]) ;
        fprintf(f," \n");
        return(0) ;
}

int     put_dbl(fptr,d)
FILE    *fptr ;
double d;

{
        fprintf(fptr,"\t%14.5g",d) ;
        return(0) ;
}

int     put_int(fptr,i)
FILE    *fptr ;
int     i ;

{
        fprintf(fptr,"\t%d",i) ;
        return(0) ;
}
```

fecore.c

```
#include "fe.h"

#define KE(A,B)        ( *(ep.K + ep.n*A + B) )
#define SE(A,B)        ( *(ep.S + ep.n*A + B) )
#define ME(A,B)        ( *(ep.M + ep.n*A + B) )
#define FE(B)              ( *(ep.F + B) )
#define Min_mem        20000

struct control              N ;
struct pointers        gp;
struct RegMesh              Mesh ;
struct fxnodes              *first_fnp, *next_fnp ;
struct transient       tvals ;
int                    Nukns, Nrfixed,  Fac, *BFnodes ;
struct elmpointers     ep ;
struct eqnset               eqnsets[4] ;
int                    BandWidth , Max_iter, nx_s = 0, steady_code;
double                      uchange ;
float                  x_s[10] ;

extern double          defndp , defelp ;

int             init_gp()

{
        int     i ;

        gp.N = NULL ;
        gp.El = NULL ;
        gp.B = NULL ;
        gp.S = NULL ;
        gp.M = NULL ;
        for(i=0;i<4;i++) {
                eqnsets[i].Kp = NULL ;
                eqnsets[i].Fp = NULL ;
                eqnsets[i].diagp = NULL ;
                eqnsets[i].neqns = NULL ;
        }
        ep.K = NULL ;
        ep.F = NULL ;
        ep.S = NULL ;
        ep.M = NULL ;
        first_fnp = NULL ;
        Mesh.nx = 10 ;
        Mesh.ny = 10 ;
        Mesh.nbx = 1 ;
        Mesh.nby = 1 ;
        Mesh.eltype = 211 ;
        Mesh.maptype = 211 ;
        Mesh.firstmap = (struct blockmap *) calloc(24,sizeof(struct blockmap)) ;
        Mesh.firstmap->xnodes[0] = 100. ; Mesh.firstmap->ynodes[0] = 0.0 ;
        Mesh.firstmap->xnodes[1] = 100. ; Mesh.firstmap->ynodes[1] = 100. ;
        Mesh.firstmap->xnodes[2] = 0.0 ; Mesh.firstmap->ynodes[2] = 100. ;
        Mesh.firstmap->xnodes[3] = 0.0 ; Mesh.firstmap->ynodes[3] = 0.0 ;
        tvals.nsteps = 10 ;
        tvals.t = 0.0 ;
        tvals.dt = 10. ;
        tvals.theta = 0.5 ;
        tvals.dtfac = 1.0 ;
        N.sym = 1 ;
}

int             MkLapMesh(theMesh)
```

fecore.c

```
struct RegMesh *theMe    ;

{
        N.trans = theMesh->con.trans = 1 ;
        N.dims = theMesh->con.dims = 2 ;
        N.vars = theMesh->con.vars = 1 ;
        N.Keqns[0] = theMesh->con.Keqns[0] = 1 ;
        N.Keqns[1] = theMesh->con.Keqns[1] = 1 ;
        N.params = theMesh->con.params = 8 ;
        N.bparams = theMesh->con.bparams = 2 ;
        defndp = 0.00 ;
        defelp = 0.00 ;
        MakeMesh(theMesh) ;
        return(0) ;
}

int             steady_state(nvar, elcode)
int             nvar, elcode ;
{
        int     i ;
        char    t = 'n';

        N.trans = 1 ;
        tvals.dt = 1000000000000000. ;
        tvals.t += tvals.dt ;
        tvals.theta = 1.0 ;
        tvals.dtfac = 100. ;
        BandWidth = 1 ;
        steady_code = 1 ;

        printf(" Do you want to use 'banded storage'(faster)?\n") ;
        scanf(" %c",&t) ;
        if(t == 'y'){
                printf(" Input the Bandwidth of the Mesh ?\n  ->") ;
                scanf(" %d",&BandWidth) ;
                printf(" %d\n",BandWidth) ;
        }
        else
                printf(" Using skylined storage(slow!)\n") ;

        printf(" Now Solving\n") ;

        assemble(nvar,0,0,0) ;
        solve(nvar,1,0) ;
        printf(" Solved First step\n") ;
        tvals.t += tvals.dt ;
        assemble(nvar,0,0,0) ;
        solve(nvar,1,0) ;
        printf(" Done \n") ;

        return(0) ;
}

int             transient(nvar,elcode)
int             nvar,elcode ;

{
        int             int_code = 0 ,i, j, nin, k, n_code = -1, in, t_int;
        struct node     *np ;
        struct element        *elp ;
        FILE            *f, *f2, *put_fptr() ;
        char            t ;
        float           x, y ;
```

```
        double          x_p ;

        BandWidth = 1 ;
        steady_code = 0 ;
        N.trans = 1 ;
        nin = 1 ;

        printf(" Do you want to use 'banded storage'(faster)?\n  ->") ;
        scanf(" %c",&t) ;
        if(t == 'y'){
                printf(" Input the Bandwidth of the Mesh ?\n  ->") ;
                scanf(" %d",&BandWidth) ;
                printf(" %d\n",BandWidth) ;
        }
        else
                printf(" Using skylined storage(slow!)\n") ;


        t = 'n' ;

        if(tvals.theta == 0.5){
                printf(" Do you want to set up a Slug as an initial condition ?\n
->") ;
                scanf(" %c",&t) ;
                if(t == 'y')
                        set_slug() ;
                t = 'n' ;
                printf(" Do you want output at certain times (max 5)?\n  ->") ;
                scanf(" %c",&t) ;
                if(t == 'y'){
                        int_code = 1 ;
                        printf(" Input the interval between output?\n  ->") ;
                        scanf(" %d",&in) ;
                        printf(" Output will be generated every %d steps\n",in) ;
                }

                printf(" Do you want a time history of the results ?\n  ->") ;
                scanf(" %c",&t) ;
        }
        if(t == 'y') {
                f = put_fptr(1) ;
                printf(" Input the interval at which to output?\n  ->") ;
                scanf(" %d",t_int) ;
                printf(" Input the number of x stations where you want the
results (max 10)?\n  ->") ;
                scanf(" %d",&nx_s) ;
                printf(" Input the x_coordinates\n");
                for(k=0;k<nx_s;k++){
                        printf(" Input %d point?\n  ->",k) ;
                        scanf(" %f",&(x_s[k])) ;
                        printf(" Output will be generated at x = %f\n",x_s[k]) ;
                }
                fprintf(f,"Delta t =\t %7.5f\t", tvals.dt) ;
                fprintf(f," Theta =\t %5.3f", tvals.theta) ;
                fprintf(f," Time =\t %7.5f",tvals.t) ;
                fprintf(f,"\nNode # \t\t\t") ;

                for(k=0;k<nx_s;k++){
                        np = gp.N ;
                        for(j=0;j<N.nodes;j++){
                                if( np->x[0] == x_s[k] ){
                                        fprintf(f,"\t%d",np->n) ;
                                }
```

```
                        np = np->nextnp ;
                }
        }

        fprintf(f,"\nX \t\t\t") ;
        for(k=0;k<nx_s;k++){
                np = gp.N ;
                for(j=0;j<N.nodes;j++){
                        if(  np->x[0] == x_s[k] ){
                                fprintf(f,"\t%f",np->x[0]) ;
                        }
                        np = np->nextnp ;
                }
        }
        fprintf(f,"\nY \t\t\t") ;
        for(k=0;k<nx_s;k++){
                np = gp.N ;
                for(j=0;j<N.nodes;j++){
                        if(  np->x[0] == x_s[k] ){
                                fprintf(f,"\t%f",np->x[1]) ;
                        }
                        np = np->nextnp ;
                }
        }
        fprintf(f,"\nC init \t\t\t") ;
        for(k=0;k<nx_s;k++){
                np = gp.N ;
                for(j=0;j<N.nodes;j++){
                        if(  np->x[0] == x_s[k] ){
                                fprintf(f,"\t%f",np->u[0]) ;
                        }
                        np = np->nextnp ;
                }
        }
}


Mesh.eltype = (gp.El)->vtype ;

printf("\n Now Solving !\n") ;
for(i=0;i<tvals.nsteps;i++) {

        tvals.t += tvals.dt ;
        assemble(nvar,0,0,0) ;
        solve(nvar,1,0) ;

        k = (int) (tvals.t/tvals.dt) ;
        if(int_code == 1 && nin < 7){
                if((i+1) % in == 0){
                        nin += 1 ;
                        output(nin) ;
                }
        }

        if(t == 'y'  && i > 1 ){
                if ( i % t_int == 0)
                        put_results(f) ;
        }

        printf(" Time = %lf Finished %d of %d time steps \t Δu -
%g\n",tvals.t,i+1, tvals.nsteps, uchange ) ;
}
if(t == 'y')
```

```
                fclose(f) ;

        return(0) ;
}


int     set_slug()

{
        int             j ;
        float           x_loc, y_loc, sigma_x, sigma_y, mag ;
        struct node     *np ;

        printf(" Input the x and y coordinates for the location of the Slug
Center?\n  ->") ;
        scanf(" %f %f",&x_loc,&y_loc) ;
        printf(" Input the sigma_x and sigma_y of the distribution?\n  ->") ;
        scanf(" %f %f",&sigma_x,&sigma_y) ;
        printf(" Input the magnitude of the Maximum value @ Center of the
distribution?\n  ->") ;
        scanf(" %f",&mag) ;

        np = gp.N ;
        for(j=0;j<N.nodes;j++){
                np->u[0] =mag * exp(-( (np->x[0]-x_loc)*(np->x[0]-
x_loc)/(sigma_x*sigma_x)
                                + (np->x[1]-y_loc)*(np->x[1]-
y_loc)/(sigma_y*sigma_y)  )/2.) ;
                if(np->u[0] < 0.00000000001)
                        np->u[0] = 0 ;

                np = np->nextnp ;
        }
        printf(" Updated the Mesh !\n") ;
        return(0) ;
}

int put_results(f)
FILE *f ;
{
        int             j,k ;
        struct node     *np ;
        float           x , y  ;

        fprintf(f,"\n") ;
        fprintf(f," Time =\t%7.5f\t ",tvals.t) ;
        fprintf(f," C\t") ;
        for(k=0;k<nx_s;k++){
                np = gp.N ;
                for(j=0;j<N.nodes;j++){
                        if(  np->x[0] == x_s[k] ){
                                fprintf(f,"\t%f",np->u[0]) ;
                        }
                        np = np->nextnp ;
                }
        }
        return(0) ;
}



int             update_PGKe(gbfp,gtfp,nbf,ntf,np,vcode)
int             nbf, ntf, vcode ;
```

```
struct node   *np[] ;
struct govfuncs      *gbfp, *gtfp ;

{
        int                             err, n ;
        register int                    i, j, nk ;
        register struct govfuncs        *f, *v ;
        register double                         *Kep, *Sep, *Fep ;
        double                                  diff, conv;

        if(vcode<0)
                n = N.vars ;
        else
                n = 1 ;

        nk = ep.n ;
        f = gbfp ;
        v = gtfp ;
        Kep = ep.K ;
        Sep = ep.S ;
        Fep = ep.F ;

        i = ntf  ;
        while(--i >= 0) {
                j = nbf ;
                while(--j >= 0) {


                        *(Kep + nk*i + j) += (v->f[i] * (f->p[0] * f->dfdx[j] + f-
>p[1] * f->dfdy[j])
/* weak statement */                    + v->dfdx[i] * (f->p[2] * f->dfdx[j] + f-
>p[3] * f->dfdy[j])
                                        + v->dfdy[i] * (f->p[4] * f->dfdx[j] + f-
>p[5] * f->dfdy[j]) ) * f->detJ;

                        if(N.trans > 0)
                                *(Sep + nk*i + j) += (v->f[i] * f->f[j]) * f->detJ
;
                }

                *(ep.F + i) += ( f->p[6] * v->f[i] ) * f->detJ ;

        }
        return(0) ;
}

int                     update_BKe(elmntp,gbf,gtf,nbf,ntf,np,nvar,dS)
int                     nbf, ntf ;
double                  dS ;
struct node             *np[] ;
struct belement         *elmntp ;
struct shapefuncs       *gbf, *gtf ;

{
        int     i, j, n ;
        double BoundK(), BoundF() ;

        n = N.vars ;
        for(i=0;i<ntf;i++) {
                for(j=0;j<nbf;j++) {
                        KE(i*n,j*n) += BoundK(elmntp,i,j,gbf,gtf,dS) ;
                }
                FE(i*n) += BoundF(elmntp,i,gtf,dS) ;
```

```
        }
}

double                  BoundK(belp,i,j,fgp,ftp,dS)
int                     i,j ;
double                  dS ;
struct shapefuncs       *fgp, *ftp ;
struct belement         *belp ;

{
        if(belp->gtype == 1)
                return(belp->p[0]) ;
        return(belp->p[0]* fgp->f[i] * ftp->f[j] * dS ) ;
}

double                  BoundF(belp,i,fgp,dS)
int                     i ;
double                  dS ;
struct shapefuncs       *fgp ;
struct belement         *belp ;

{
        if(belp->gtype == 1)
                return( -belp->p[1]) ;
        return( -belp->p[1] * fgp->f[i] * dS ) ;
}



int             assemble(varnum,code,nset,elcode)
int             varnum, code, nset, elcode ;

{
        double                  *K_init(), *L_init(),*F_init() ;
        struct fxnodes                  *fn_init();
        struct element                  *elp, tel ;
        struct belement         *belp, tbel ;
        double                  *p ;
        int                     i,j,ns,ntf,istart,iend,ind  ;
        long                    lj ;
        char                    m1[255], m2[255], m3[255], m4[255] ;


        istart = 0 ;
        iend = N.elms ;
        if(N.trans == 0)
                tvals.dt = 1.0 ;
        uchange = 0.0 ;
        if(code< 2) {
                if(eqnsets[nset].Kp != NULL)
                        free(eqnsets[nset].Kp) ;
                if( (eqnsets[nset].Kp = K_init(varnum,&eqnsets[nset].diagp,code))
== NULL )
                        return(-1) ;


                if(N.sym != 0) {
                        if(eqnsets[nset].Lp != NULL )
                          free(eqnsets[nset].Lp) ;
                        if( (eqnsets[nset].Lp =
L_init(varnum,&eqnsets[nset].diagp,code)) == NULL )
                                return(-1) ;
                }
```

```

		eqnsets[nset].neqns = Nukns ;
		if(steady_code == 1){
			sprintf(m1," Assembling K matrix and F vector ");
			sprintf(m2," Number of equations to be solved = %d
",Nukns) ;
			sprintf(m3," Elements in K matrix - %ld
",*(eqnsets[nset].diagp+Nukns-1)+1) ;
			sprintf(m4," ") ;
			StartMes(m1,m2,m3,m4) ;
		}

		if(eqnsets[nset].Fp != NULL) {
			free(eqnsets[nset].Fp) ;
		}
		if( ( eqnsets[nset].Fp = F_init(Nukns) ) == NULL )
			return(-1) ;
	}
	else if (code == 2) {
		ind = 0 ;
		for(i=0;i<N.nodes;i++) {
			gp.iptrs[i]->ui = ind ;
			ind++ ;
		}
	}

	if(code < 1) {

		if( (ep.n = ep_init() ) == 0 )
			return(-1) ;
	}
	for(i=0;i<eqnsets[nset].neqns;i++)
		*(eqnsets[nset].Fp + i) = 0.0 ;

	elp = gp.El ;
	for(i=istart;i<iend;i++) {
		if( i % 10 == 0 && steady_code == 1 ) {
			sprintf(m4," Assembling Element # %d of %d ",i,N.elms) ;
			UpDateMes(m1,m2,m3,m4) ;
		}

		ns = get_PGKe(elp,&tel,varnum,elcode,&ntf) ;


	put_Ke(ns,ntf,&tel,varnum,eqnsets[nset].Kp,eqnsets[nset].Lp,eqnsets[nset
].Fp,eqnsets[nset].diagp,code) ;
		elp = elp->nextelp ;
	}
	if( steady_code == 1){
		sprintf(m4," Assembling Elements done !");
		UpDateMes(m1,m2,m3,m4);
	}

	belp = gp.B ;
	for(i=0;i<N.belms;i++) {
		ns = get_bKe(belp,&tbel,varnum,elcode,&ntf) ;
		if(ns > 0 )

	put_Ke(ns,ntf,belp,varnum,eqnsets[nset].Kp,eqnsets[nset].Lp,eqnset
].Fp,eqnsets[nset].diagp,code) ;
		belp = belp->nextbelp ;
	}
	if(steady_code == 1){
```

```
		sprintf(m4," Assembling Boundary Elements done !");
		UpDateMes(m1,m2,m3,m4);
	}

	if(code < 2)
		EndMes() ;

	return(0) ;
}


int		ep_init()

{
	int		i, ns, nsm ;
	struct element		*elp ;

	if(ep.K != NULL)
		free(ep.K) ;
	elp = gp.El ;
	nsm = 0 ;

	for(i=0;i<N.elms;i++) {
		nsm = ( (ns = nsf(elp->vtype)) > nsm ) ? ns : nsm ;
		elp = elp->nextelp ;
	}
	nsm *= N.vars ;
	if( (ep.K = (double *) calloc(nsm*nsm, sizeof(double)) ) == NULL )
		return(0) ;
	if(ep.F != NULL)
		free(ep.F) ;
	if( (ep.F = (double *) calloc(nsm, sizeof(double)) ) == NULL )
		return(0) ;
	if(N.trans > 0) {
		if(ep.S != NULL)
			free(ep.S) ;
		if( (ep.S = (double *) calloc(nsm*nsm, sizeof(double)) ) == NULL
)
			return(0) ;
	}
	if(N.trans > 1) {
		if(ep.M != NULL)
			free(ep.M) ;
		if( (ep.M = (double *) calloc(nsm*nsm, sizeof(double)) ) == NULL
)
		return(0) ;
	}
	return(nsm) ;
}


double		*K_init(nvar,thediagp,code)
int		nvar, code ;
long		*(*thediagp) ;

{
	int		nj, i, j, nf , ind;
	double			*kp, *dp ;
	unsigned	elsize ;
	long		bigint, nelem,*diagp, il ;

	nelem = 0 ;
	for(i=0;i<N.nodes;i++) {
```

```
                gp.iptrs[i]->ui = 9999 ;
                nelem++ ;
        }
        Nukns = nelem ;
        if(*thediagp != NULL)
                free(*thediagp) ;
        if((*thediagp = (long int *) calloc((int)nelem,sizeof(long))) == NULL)
                return(NULL) ;
        diagp = *thediagp ;
        *diagp = 0 ;
        ind = 0 ;
        diagp++ ;
        gp.iptrs[0]->ui = ind ;
        ind++ ;
        for(i=1;i<N.nodes;i++) {
                gp.iptrs[i]->ui = ind ;
                ind++ ;
                if(code == 0){
                        if( BandWidth == 1)
                                nf =  (mpd_i(i,nvar)  + 1) * N.vars ;
                        else {
                                if(ind < BandWidth)
                                        nf = ind ;
                                else
                                        nf = BandWidth;
                        }


                }
                else if(code == -1)
                        nf = 1 ;
                else if(code == 2) {
                        if(ind == 1)
                                nf = 1 ;
                        else if(ind == 2)
                                nf = 2 ;
                        else
                                nf = 3 ;
                }
                else
                        nf = 2 ;
                if(diagp != *thediagp)
                        *diagp = *(diagp-1) + nf ;
                diagp++ ;
        }
        nelem = *(diagp-1) + 1 ;
        elsize = sizeof(double) ;
        bigint =  nelem * (long) elsize ;
        if((kp =  dp = (double *) lmalloc(bigint))==NULL){           /* NON
STANDARD FUNCTION    */                           /* NON STANDARD FUNCTION CALL */
                printf(" Insufficent Memory avaliable") ;
                exit(0) ;
        }
        for(il=0;il<nelem;il++)
                *(dp++) = 0.0 ;
        return(kp) ;
}


int             mpd_i(ind,nvar)
int             ind , nvar;

{
        int                             i,j,n,mpd,td ;
```

```
        register struct element     *elmntp ;

        mpd = 0 ;

        if(N.dims == 1)
                return(3) ;
        elmntp = gp.El ;
        for(i=0;i<N.elms;i++) {
                n = -1 ;
                for(j=0;j<elmntp->nnds;j++) {
                        if(elmntp->nps[j] != NULL) {
                                if(gp.iptrs[ind]->n ==   (elmntp->nps[j])->n )
                                        n = gp.iptrs[ind]->n ;
                        }
                }
                if( n >= 0 ) {
                        for(j=0;j<elmntp->nnds;j++) {
                                if(elmntp->nps[j] != NULL)  {
                                        if((elmntp->nps[j])->n >= 0)
                                                td = d_i(ind,(elmntp->nps[j])->i) ;
                                        mpd = (mpd > td ) ? mpd : td ;
                                }
                        }
                }
                elmntp = elmntp->nextelp ;
        }
        return(mpd) ;
}

int             d_i(n1,n2)
int             n1, n2 ;

{
        return(n1 - n2) ;
}


double *L_init(nvar,thediagp,code)
int     nvar,code ;
long    *(*thediagp) ;

{
        int             nj, i, j, nf , ind;
        double                  *kp, *dp ;
        unsigned        elsize ;
        long            bigint, nelem, il, *diagp ;

        diagp = *thediagp ;
        for(i=0;i<N.nodes;i++) {
                if(isvarfixed(i,nvar) == 0) {
                        diagp++ ;
                }
        }

        nelem = *(diagp-1) + 1 ;
        elsize = sizeof(double) ;
        bigint =  nelem * (long) elsize  ;
        if((kp =  dp = (double *) lmalloc(bigint))==NULL){
/* NON STANDARD FUNCTION CALL */
                printf(" Insufficent Memory avaliable") ;
                exit(0) ;
        }
        for(il=0;il<nelem;il++)
```

```
		*(dp+i) = 0.0 ;
	return(kp) ;
}


double *F_init(num)
int		num ;

{
	double		*p ;
	unsigned	nelem, elsize ;

	nelem = num ;
	elsize = sizeof(double) ;
	p = (double *) calloc(nelem,elsize) ;
	return(p) ;
}



int		put_Ke(ns,nt,elmntp,nvar,Kp,Lp,Fp,diagp,code)
int		ns, code ;
double		*Kp,  *Lp,  *Fp ;
struct element		*elmntp ;
long		*diagp ;

{
	int		i, in, j, ii, jj, jn, frow, fcol, row, col, tj, isv ;
	struct node	*inp, *jnp ;
	struct fxnodes		*the_fnp ;

	for(i=0;i<nt;i++) {
		inp = elmntp->nps[i] ;
		frow = inp->i * N.vars ;
		for(j=0;j<ns;j++) {
			jnp = elmntp->nps[j] ;
			fcol = jnp->i * N.vars;
			for(ii=0;ii<N.vars;ii++) {
				row = frow + ii ;
				for(jj=0;jj<N.vars;jj++) {
					col = fcol + jj ;
					if( (col >= row) && ( code < 2 ) && ( code >
-1 ) ) {
						if(N.trans == 0)
							*(Kp + *(diagp+col) - col +
row ) += KE(i*N.vars+ii,j*N.vars+jj) ;
						else
							*(Kp + *(diagp+col) - col +
row ) += SE(i*N.vars+ii,j*N.vars+jj) ;
					}
					if( (col <= row) && (N.sym != 0 ) ) {
						if(N.trans == 0)
							*(Lp + *(diagp+row) - row +
col ) += KE(i*N.vars+ii,j*N.vars+jj) ;
						else
							*(Lp + *(diagp+row) - row +
col ) += SE(i*N.vars+ii,j*N.vars+jj) ;
					}
					if( (col == row) && ( code == -1 ) ) {
						if(N.trans == 0)
							*(Kp + r    ) +=
KE(i*N.vars+ii,j*N.vars+jj) ;
						else
```

```
                                                            *(Kp + row ) +=
SE(i*N.vars+ii,j*N.vars+jj) ;
                                        }
                                        if(  (code == 2 ) || code == -1 ) {
                                                *(Fp  + row ) += -
KE(i*N.vars+ii,j*N.vars+jj) * jnp->u[jj] ;
                                        }
                                }
                        }
                }
                for(ii=0;ii<N.vars;ii++) {
                        row = frow + ii ;
                        *(Fp + row) += FE(i*N.vars+ii) ;
                }
        }
}


int             get_PGKe(elmntp,theElp,nvar,code,ntf)
struct element          *elmntp, *theElp ;
int             nvar, code, *ntf ;

{
        struct gausspts         g[MGPTS] ;
        struct shapefuncs       bf, tf, *fgp;
        struct node             *np[MNSF] ;
        struct govfuncs         gbf, gtf ;
        int                     i, j, nbf, ni, nm, dir ;
        double                  tdt, mtdt ;
        double                  *fptr ;

        initm(ep.K,ep.n*ep.n) ;
        if(N.trans > 0)
                initm(ep.S,ep.n*ep.n) ;
        initm(ep.F,ep.n) ;

        ElPick(elmntp,theElp,ntf,&dir) ;
        nbf = theElp->nnds ;

        if(code > 0 || elmntp->matrices == NULL) {
                ni = get_gspts(theElp->vtype,g) ;
                for(j=0;j<theElp->nnds;j++)
                        np[j] = theElp->nps[j];

                for(i=0;i<ni;i++) {
                        get_shape(theElp->vtype,&g[i],&bf) ;
                        get_shape(theElp->gtype,&g[i],&tf) ;
                        get_PGgf(theElp,&gbf,&gtf,&bf,&tf,np,g[i].w) ;
                        update_PGKe(&gbf,&gtf,nbf,*ntf,np,nvar) ;
                }
                if(code != 2 && (FreeMem() > Min_mem)) {
                        if(elmntp->matrices != NULL)
                                free(elmntp->matrices) ;
                        nm = *ntf  * (nbf + 1) ;
                        if(N.trans > 0)
                                nm += *ntf * nbf ;
                        if((elmntp->matrices = (double *)calloc(nm
,sizeof(double)) ) == NULL)
                                return(-1) ;
                        fptr = elmntp->matrices ;
                        for(i=0;i<*ntf;i++)
                                for(j=0;j<nbf;j++) {
                                        *fptr = KE(i,j) ;
```

```
					fptr++ ;
				}
			for(i=0;i<*ntf;i++){
				*fptr = FE(i);
				fptr++ ;
			}
			if(N.trans > 0) {
				for(i=0;i<*ntf;i++) {
					for(j=0;j<nbf;j++) {
						*fptr = SE(i,j) ;
						fptr++ ;
					}
				}
			}
		}
	}
	else {
		fptr = elmntp->matrices ;
		for(i=0;i<*ntf;i++)
			for(j=0;j<nbf;j++) {
				KE(i,j) = *fptr ;
				fptr++ ;
			}
		for(i=0;i<*ntf;i++) {
			FE(i) = *fptr ;
			fptr++ ;
		}
		if(N.trans > 0) {
			for(i=0;i<*ntf;i++) {
				for(j=0;j<nbf;j++) {
					SE(i,j) = *fptr ;
					fptr++ ;
				}
			}
		}
	}
	if(N.trans > 0) {
		tdt = tvals.theta * tvals.dt ;
		mtdt = tvals.dt * (1.0 - tvals.theta) ;
		for(i=0;i<*ntf;i++) {
			FE(i) *= tvals.dt ;
			for(j=0;j<nbf;j++) {
				FE(i) += (SE(i,j) - mtdt * KE(i,j)) * (theElp-
>nps[j])->u[nvar] ;
				SE(i,j) += tdt * KE(i,j) ;
			}
		}
	}
	return(nbf) ;
}

int		ElPick(elmntp,theElp,ntfp,dirp)
struct element		*elmntp, *theElp ;
int		*ntfp, *dirp ;

{
	int	j;

	*dirp = 1 ;

	if(elmntp->vtype == 229)
		return(QUelPick(elmntp,theElp,ntfp,dirp)) ;
	else if(elmntp->vtype == 239)
```

```
                return(CU2DPick(elmntp,theElp,ntfp,dirp)) ;
        else if(elmntp->vtype == 233)
                return(CU_Stream_Pick(elmntp,theElp,ntfp,dirp)) ;



        *ntfp = nsf(elmntp->qtype) ;
        *theElp = *elmntp ;
        return(0) ;
}

int                     get_PGgf(elp,gbfp,gtfp,bfp,tfp,np,w)
struct element                  *elp ;
struct shapefuncs       *bfp, *tfp ;
struct node             *np[] ;
struct govfuncs         *gbfp,*gtfp ;
double                  w ;

{
        double J[NDIMS][NDIMS], get_J() ;
        int     i, j ;

        gbfp->dof = bfp->dof ;
        gtfp->dof = tfp->dof ;

        for(i=0;i<N.params;i++) {
                gbfp->p[i] = elp->p[i] ;
                for(j=0;j<gbfp->dof;j++)
                        gbfp->p[i] += bfp->f[j] * np[j]->p[i] ;
        }
        for(j=0;j<bfp->dof;j++)
                gbfp->f[j] = bfp->f[j]        ;
        for(j=0;j<tfp->dof;j++)
                gtfp->f[j] = tfp->f[j]        ;

        gbfp->detJ = get_J(np,bfp,J) ;
        gbfp->w = w ;
        invertJ(J,gbfp->detJ) ;

        for(j=0;j<bfp->dof;j++) {

                switch (N.dims) {

                        case 1 :
                                gbfp->dfdx[j] = J[0][0] * bfp->dfdr[j] ;
                                gbfp->dfdy[j] = 0.0 ;
                                gbfp->dfdz[j] = 0.0 ;
                                break ;

                        case 2 :
                                gbfp->dfdx[j] = J[0][0] * bfp->dfdr[j] + J[0][1] *
bfp->dfds[j] ;
                                gbfp->dfdy[j] = J[1][0] * bfp->dfdr[j] + J[1][1] *
bfp->dfds[j] ;
                                gbfp->dfdz[j] = 0.0 ;
                                break ;

                        case 3 :
                                gbfp->dfdx[j] = J[0][0] * bfp->dfdr[j] + J[0][1] *
bfp->dfds[j] + J[0][2] *bfp->dfdt[j] ;
                                gbfp->dfdy[j] = J[1][0] * bfp->dfdr[j] + J[1][1] *
bfp->dfds[j] + J[1][2] *bfp->dfdt[j] ;
```

```
                                        gbfp->dfdz[j] = J[2][0] * bfp->dfdr[j] + J[2][1] *
bfp->dfds[j] + J[2][2] *bfp->dfdt[j] ;
                                        break ;
                        }
                }

                for(j=0;j<tfp->dof;j++) {

                        switch (N.dims) {

                                case 1 :
                                        gtfp->dfdx[j] = J[0][0] * tfp->dfdr[j] ;
                                        gtfp->dfdy[j] = 0.0 ;
                                        gtfp->dfdz[j] = 0.0 ;
                                        break ;

                                case 2 :
                                        gtfp->dfdx[j] = J[0][0] * tfp->dfdr[j] + J[0][1] *
tfp->dfds[j] ;
                                        gtfp->dfdy[j] = J[1][0] * tfp->dfdr[j] + J[1][1] *
tfp->dfds[j] ;
                                        gtfp->dfdz[j] = 0.0 ;
                                        break ;

                                case 3 :
                                        gtfp->dfdx[j] = J[0][0] * tfp->dfdr[j] + J[0][1] *
tfp->dfds[j] + J[0][2] *tfp->dfdt[j] ;
                                        gtfp->dfdy[j] = J[1][0] * tfp->dfdr[j] + J[1][1] *
tfp->dfds[j] + J[1][2] *tfp->dfdt[j] ;
                                        gtfp->dfdz[j] = J[2][0] * tfp->dfdr[j] + J[2][1] *
tfp->dfds[j] + J[2][2] *tfp->dfdt[j] ;
                                        break ;
                        }
                }
                gbfp->detJ *= gbfp->w ;
                return(0) ;
}
```

feinitial.c

```
#include "fe.h"

/*
                p[0] = u
                p[1] = v
                p[2] = Exx
                p[3] = Exy
                p[4] = Eyx
                p[5] = Eyy
                p[6] = P

*/
extern struct transient tvals;
extern int          BandWidth ;

int             nodevalues(np)
struct node     *np ;

{
        np->p[0] = 1.0 ;
        np->p[1] = 0.0 ;
        np->p[2] = 0.0 ;
        np->p[3] = 0.0 ;
        np->p[4] = 0.0 ;
        np->p[5] = 0.0 ;
        np->p[6] = 0.0 ;

        return(0) ;
}

int             elvalues(elp)
struct element      *elp ;

{
        return(0) ;
}

int             bottomb(belp)
struct belement *belp ;

{
        belp->bcs[0] = 2 ;
        belp->p[0] = 0.0 ;
        belp->p[1] = 0.0 ;
        return(0) ;
}

int             topb(belp)
struct belement *belp ;

{
        belp->bcs[0] = 2 ;
        belp->p[0] = 0.0 ;
        belp->p[1] = 0.0 ;
        return(0) ;
}

int             leftb(belp)
struct belement *belp ;

{
        belp->bcs[0] = 3 ;
        belp->p[0] = 0.0 ;
```

```
        belp->p[1] = 0.0 ;
        return(0) ;
}

int             rightb(belp)
struct belement *belp ;

{
        belp->bcs[0] = 2 ;
        belp->p[0] = 0.0 ;
        belp->p[1] = 0.0 ;
        return(0) ;
}


double          uexact(nvar,x,y,z,p)
int             nvar ;
double          x, y, z, p ;

{
        double t ;

        t = exp(-((x-6800.0)*(x-6800.0))/(2.*264.*264.)) ;
        return(t) ;
}


int             update_p(np)
struct node     *np ;

{
        return(0) ;
```

fecom.c

```
#include "fe.h"

#define         KE(A,B)         ( *(ep.K + ep.n*A + B) )
#define         SE(A,B)         ( *(ep.S + ep.n*A + B) )
#define         ME(A,B)         ( *(ep.M + ep.n*A + B) )
#define         FE(B)           ( *(ep.F + B) )

extern struct control                N ;
extern struct pointers               gp;
extern struct RegMesh                Mesh ;
extern struct fxnodes                *first_fnp ;
extern int                   Nukns, steady_code ;
extern struct elmpointers    ep ;
extern struct transient              tvals ;
extern struct eqnset         eqnsets[4] ;
extern double                uchange ;


int             varindex(ndindex,var)
int             ndindex, var ;

{
        int     i, vi ;

        vi = gp.iptrs[ndindex]->ui ;
        for(i=0;i<var;i++) {
                if(isvarfixed(ndindex,i) == 0)
                        vi++ ;
        }
        return(vi) ;
}

int             isvarfixed(ndindex,var)
int             ndindex, var ;

{
        int     fixcode, rc ;

        rc = 0 ;
        fixcode = (1 << var) ;
        if( (fixcode & gp.iptrs[ndindex]->fxc) != 0)
                rc = -1 ;
        return(rc) ;
}

int             solve(nvar,code,nset)
int             code, nvar, nset ;

{
        int     i, j, ind, solvecode ;
        double *dp ;

        if(code > 1)
                solvecode = 1 ;
        else
                solvecode = code ;
        if(code < 0) {
                solvecode = 2 ;
                code = -code ;
        }
        if(steady_code == 1)
                printf(" Now solving ...") ;

```

```
        if((N.sym ==  0)   ||   (code == 3))
                actcol(eqnsets[nset].Kp,eqnsets[nset].Fp,eqnsets[nset].diagp,
eqnsets[nset].neqns, solvecode) ;
        else

        uactcl(eqnsets[nset].Kp,eqnsets[nset].Lp,eqnsets[nset].Fp,eqnsets[nset].
diagp, eqnsets[nset].neqns, solvecode) ;

        if(steady_code == 1)
                printf(" All done !\n") ;


        dp = eqnsets[nset].Fp ;
        if((code > 0) && (code != 3)) {
                if(nvar >= 0) {
                        ind = 0 ;
                /*      uchange = 0.0 ;         */
                        for(i=0;i<N.nodes;i++) {
                                if(code == 1) {
                                        uchange += fabs(gp.iptrs[i]->u[nvar] - *dp)
;

                                        gp.iptrs[i]->u[nvar] = *dp ;
                                }
                                if(code == 2) {
                                        *(eqnsets[nset+1].Fp + ind) =
*(eqnsets[nset].Fp + gp.iptrs[i]->i) ;
                                /*
                                        gp.iptrs[i]->p[4] = *(eqnsets[nset].Fp +
gp.iptrs[i]->i) ;

                                }
                                if(code == 4) {
                                /*
                                        printf(" %f  %f\n",gp.iptrs[i]->u[nvar],
*dp);
                                */
                                        uchange += fabs(*dp) ;
                                        gp.iptrs[i]->u[nvar] += *dp ;
                                }
                                dp++;
                                ind++ ;
                        }
                }
                else {
                        for(i=0;i<N.nodes;i++) {
                                for(j=0;j<N.vars;j++) {
                                        if(code == 1)
                                                gp.iptrs[i]->u[j] = *dp ;
                                        if(code == 2)
                                                gp.iptrs[i]->u[j] += *dp ;
                                        dp++;
                                }
                        }
                }
        }
        return(0) ;
}

int             actcol(Kp,Fp,diagp,neqns,code)
double          *Kp, *Fp ;
long            *diagp ;
int             neqns, code ;

```

```
{
        int     i, j, nf, id, n, ni, nj ;
        double dot(), *ip, *idp, t ;
        char   m1[255], m2[255], m3[255], m4[255] ;

        if(steady_code == 1){
                sprintf(m1," Now factoring equation # ") ;
                sprintf(m2," 0 ") ;
                sprintf(m3," ") ;
                sprintf(m4," ") ;
                StartMes(m1,m2,m3,m4) ;
        }


        for(j=1;j<neqns;j++) {

                if( (j % 10 == 0) && (steady_code == 1)) {
                        sprintf(m4,"  %d ",j) ;
                        UpDateMes(m1,m2,m3,m4) ;
                }

                nf = *(diagp+j) - *(diagp+j-1) ;
                if(code < 2) {
                        ip = Kp + *(diagp+j-1) + 1 ;
                        id = j - nf + 1 ;

                        for(i=1;i<nf;i++) {
                                n = ((nj = i-1) < (ni = *(diagp+id) - *(diagp+id-1)
- 1)) ? nj : ni ;
                                idp = Kp + *(diagp+id++) ;
                                *ip++ -= dot(ip-n-1,idp-n,n) ;
                        }
                        ip = Kp + *(diagp+j-1) + 1 ;
                        id = j - nf + 1 ;
                        idp = Kp + *(diagp+j) ;
                        for(i=1;i<nf;i++) {
                                t = *ip ;
                                *ip /= *(Kp + *(diagp+id++)) ;
                                *idp -= t * *ip++ ;
                        }
                }
                ip = Kp + *(diagp+j-1) + 1 ;
                id = j - nf + 1 ;
                if(code > 0)
                        *(Fp + j) -= dot(ip,Fp+id,nf-1) ;
        }
        if(code == 0) {
                EndMes() ;
                return(0) ;
        }
        for(j=0;j<neqns;j++)
                *(Fp + j) /= *(Kp + *(diagp+j)) ;

        for(j=neqns-1;j>-1;j--) {
                if(j == 0)
                        nf = 1;
                else
                        nf = *(diagp+j) - *(diagp+j-1) ;
                for(i=1;i<nf;i++) {
                        *(Fp + j - nf + i) -= *(Fp + j) * *(Kp + *(diagp+j-1) + i)

                }
        }
```

```
        if(code < 2)
                EndMes() ;
        return(0) ;
}

int             uactcl(Kp;Lp,Fp,diagp,neqns,code)
double          *Kp, *Lp, *Fp ;
long            *diagp ;
int             neqns, code ;

{
        int     i, j, nf, id, n, ni, nj ;
        double dot(), *ip, *iLp, *idp, t ;
        char    m1[255], m2[255], m3[255], m4[255] ;

        if(steady_code == 1) {
                sprintf(m1," Now factoring equation # ") ;
                sprintf(m2," 0 ") ;
                sprintf(m3," ") ;
                sprintf(m4," ") ;
                StartMes(m1,m2,m3,m4) ;
        }

        for(j=1;j<neqns;j++) {

                if( (j % 10 == 0) && (steady_code == 1)) {
                        sprintf(m4,"  %d ",j) ;
                        UpDateMes(m1,m2,m3,m4) ;
                }

                nf = *(diagp+j) - *(diagp+j-1) ;
                if(code < 2) {
                        iLp = Lp + *(diagp+j-1) + 1 ;
                        id = j - nf + 1 ;

                        for(i=0;i<nf-1;i++) {
                                n = ((nj = i) < (ni = *(diagp+id) - *(diagp+id-1) -
1)) ? nj : ni ;
                                idp = Kp + *(diagp+id) ;
                                id++ ;
                                *iLp = (*iLp - dot(iLp-n,idp-n,n))/ *idp ;
                                iLp++ ;
                        }
                        ip = Kp + *(diagp+j-1) + 1 ;
                        id = j - nf + 1 ;

                        for(i=0;i<nf;i++) {
                                n = ((nj = i) < (ni = *(diagp+id) - *(diagp+id-1) -
1)) ? nj : ni ;
                                idp = Lp + *(diagp+id) ;
                                id++ ;
                                *ip -= dot(ip-n,idp-n,n) ;
                                ip++ ;
                        }
                }
                iLp = Lp + *(diagp+j-1) + 1 ;
                id = [illegible] - nf + 1 ;
                if(code > 0)
                        *(Fp+[illegible]) -= dot(iLp,Fp+[illegible],nf-1) ;

        [illegible](code [illegible]) {
                EndMes() ;
```

```
                return(0) ;
        }
        for(j=neqns-1;j>-1;j--) {
                *(Fp + j) /= *(Kp + *(diagp+j)) ;
                if(j == 0)
                        nf = 1;
                else
                        nf = *(diagp+j) - *(diagp+j-1) ;
                for(i=1;i<nf;i++) {
                        *(Fp + j - nf + i) -= *(Fp + j) * *(Kp + *(diagp+j-1) + i)

                }
        }
        if(code < 2)
                EndMes() ;
        return(0) ;
}


double dot(p1,p2,n)
double *p1, *p2 ;
int     n ;

{
        register double      t=0.0 ;

        while(n-- > 0) {
                        t += *p1++ * *p2++ ;
        }
        return(t) ;
}

int                     get_Derivs(elmntp,pt,nvar,ddx,ddy,ddz)
struct element                  *elmntp ;
struct gausspts         *pt ;
int                     nvar ;
double                  *ddx, *ddy, *ddz ;

{
        struct shapefuncs       fv;
        struct node             *np[MNSF] ;
        struct govfuncs                 gf ;
        int                     i, j, ns, ni, nm ;

        ns = nsf(elmntp->vtype) ;
        for(j=0;j<elmntp->nnds;j++)
                np[j] = elmntp->nps[j] ;
        get_shape(elmntp->vtype,pt,&fv) ;
        get_gf(elmntp,&gf,&fv,&fv,np,pt->w) ;
        *ddx = *ddy = *ddz = 0.0 ;
        for(j=0;j<elmntp->nnds;j++) {
                *ddx += gf.dfdx[j] * np[j]->u[nvar] ;
                *ddy += gf.dfdy[j] * np[j]->u[nvar] ;
                *ddz += gf.dfdz[j] * np[j]->u[nvar] ;
        }
        *ddx *= gf.p[0] ;
        *ddy *= gf.p[0] ;
        *ddz *= gf.p[0] ;
        return(ns) ;


{                       get_gf(elp,gfp,fvp,fgp,np,w)
struct element                  *elp ;
```

```
struct shapefuncs     *fvp, *fgp ;
struct node           *np[MNSF] ;
struct govfuncs             *gfp ;
double                w ;

{
        double J[NDIMS][NDIMS], get_J() ;
        int     i, j ;

        gfp->dof = fvp->dof ;

        for(i=0;i<N.params;i++) {
              gfp->p[i] = elp->p[i] ;
              for(j=0;j<fvp->dof;j++)
                    gfp->p[i] += fvp->f[j] * np[j]->p[i] ;
        }
        for(j=0;j<fvp->dof;j++)
              gfp->f[j] = fvp->f[j]  ;

        gfp->detJ = get_J(np,fgp,J) ;
        gfp->w = w ;
        invertJ(J,gfp->detJ) ;

        for(j=0;j<fvp->dof;j++) {

              switch (N.dims) {

                    case 1 :
                          gfp->dfdx[j] = J[0][0] * fvp->dfdr[j] ;
                          gfp->dfdy[j] = 0.0 ;
                          gfp->dfdz[j] = 0.0 ;
                          break ;

                    case 2 :
                          gfp->dfdx[j] = J[0][0] * fvp->dfdr[j] + J[0][1] *
fvp->dfds[j] ;
                          gfp->dfdy[j] = J[1][0] * fvp->dfdr[j] + J[1][1] *
fvp->dfds[j] ;
                          gfp->dfdz[j] = 0.0 ;
                          break ;

                    case 3 :
                          gfp->dfdx[j] = J[0][0] * fvp->dfdr[j] + J[0][1] *
fvp->dfds[j] + J[0][2] *fvp->dfdt[j] ;
                          gfp->dfdy[j] = J[1][0] * fvp->dfdr[j] + J[1][1] *
fvp->dfds[j] + J[1][2] *fvp->dfdt[j] ;
                          gfp->dfdz[j] = J[2][0] * fvp->dfdr[j] + J[2][1] *
fvp->dfds[j] + J[2][2] *fvp->dfdt[j] ;
                          break ;
              }
        }
        gfp->detJ *= gfp->w ;
        return(0) ;
}

double                get_J(np,fgp,J)
struct shapefuncs     *fgp ;
struct node           *np[MNSF] ;
double                J[NDIMS][NDIMS] ;


        int
```

```
	for(i=0;i<N.dims;i++)
		for(j=0;j<N.dims;j++)
			J[i][j] = 0.0 ;

	for(i=0;i<fgp->dof;i++)
		J[0][0] += fgp->dfdr[i] * np[i]->x[0] ;
	if(N.dims == 1)
		return( J[0][0] ) ;

	for(i=0;i<fgp->dof;i++) {
		J[0][1] += fgp->dfdr[i] * np[i]->x[1] ;
		J[1][0] += fgp->dfds[i] * np[i]->x[0] ;
		J[1][1] += fgp->dfds[i] * np[i]->x[1] ;
	}
	if(N.dims == 2)
		return( J[0][0] * J[1][1] - J[0][1] * J[1][0] ) ;

	for(i=0;i<fgp->dof;i++) {
		J[0][2] += fgp->dfdr[i] * np[i]->x[2] ;
		J[1][2] += fgp->dfds[i] * np[i]->x[2] ;
		J[2][2] += fgp->dfdt[i] * np[i]->x[2] ;
		J[2][0] += fgp->dfdt[i] * np[i]->x[0] ;
		J[2][1] += fgp->dfdt[i] * np[i]->x[1] ;
	}
	return( J[0][0] * (J[1][1] * J[2][2] - J[1][2] * J[2][1] )
	    -  J[0][1] * (J[1][0] * J[2][2] - J[2][0] * J[1][2] )
		+   J[0][2] * (J[1][0] * J[2][1] - J[1][1] * J[2][1] ) ) ;
}

int		invertJ(J,detJ)
double		J[NDIMS][NDIMS], detJ ;

{
	double K[NDIMS][NDIMS] ;
	int	i,j ;

	for(i=0;i<N.dims;i++)
		for(j=0;j<N.dims;j++)
			K[i][j] = J[i][j] ;

	switch (N.dims) {

		case 1 :
			J[0][0] = 1.0 / K[0][0] ;
			break ;

		case 2 :
			J[0][0] =  K[1][1] / detJ ;
			J[0][1] = -K[0][1] / detJ ;
			J[1][0] = -K[1][0] / detJ ;
			J[1][1] =  K[0][0] / detJ ;
			break ;

		case 3 :
			J[0][0] = ( K[1][1] * K[2][2] - K[1][2] * K[2][1] ) / detJ
;
			J[0][1] = ( K[0][2] * K[2][1] - K[0][1] * K[2][2] ) / detJ
;
			J[0][2] = ( K[0][1] * K[1][2] - K[1][1] * K[0][2] ) / detJ
;
			J[1][0] = ( K[1][2] * K[2][0] - K[1][0] * K[2][2] ) / detJ
;
```

```
                        J[1][1] = ( K[0][0] * K[2][2] - K[0][2] * K[2][0] ) / detJ ;
                        J[1][2] = ( K[1][0] * K[0][2] - K[0][0] * K[1][2] ) / detJ ;
                        J[2][0] = ( K[1][0] * K[2][1] - K[1][1] * K[2][0] ) / detJ ;
                        J[2][1] = ( K[0][1] * K[2][0] - K[0][0] * K[2][1] ) / detJ ;
                        J[2][2] = ( K[0][0] * K[1][1] - K[0][1] * K[1][0] ) / detJ ;
                        break ;
        }
        return(0) ;
}


int                     get_bKe(elmntp,theElp,nvar,code,ntf)
struct belement                 *elmntp, *theElp ;
int                     nvar, code, *ntf ;

{

        struct gausspts         g[MGPTS] ;
        struct shapefuncs       fv, ft, *fgp ;
        struct node             *np[MNSF] ;
        struct govfuncs                 gf, gtf ;
        int                     i, j, ig, ns, ni, dir ;
        float                   *fptr ;
        double                  J00, J01, dS, tdt, mtdt ;

        if(nvar < 0)
                ns =  nsf(elmntp->vtype) * N.vars ;
        else
                ns =  nsf(elmntp->vtype) ;
        initm(ep.K,ep.n*ep.n) ;
        if(N.trans > 0)
                initm(ep.S,ep.n*ep.n) ;
        initm(ep.F,ep.n) ;
        *theElp = *elmntp ;
        if(code > -1 ) {
                ni = get_gspts(elmntp->vtype,g) ;
                for(j=0;j<elmntp->nnds;j++)
                        np[j] = elmntp->nps[j] ;

                for(ig=0;ig<ni;ig++) {
                        get_shape(elmntp->vtype,&g[ig],&fv) ;
                        get_shape(elmntp->gtype,&g[ig],&ft) ;
                        if(nvar < 0)
                                *ntf = ft.dof * N.vars ;
                        else
                                *ntf = ft.dof ;
                        if((elmntp->bcs[nvar] == 3) && (nvar >= 0)) {
                                for(i=0;i<*ntf;i++) {
                                        KE(i,i) = 1.0e16 ;
                                        FE(i) = 1.0e16 * np[i]->u[nvar] ;
                                }
                                break ;
                        }

                        J00 = J01 = 0.0 ;
                        fgp = &fv ;
                        for(i=0;i<fgp->dof;i++) {
                                J00 += fgp->dfdr[i] * np[i]->x[0] ;
```

```
                                J01 += fgp->dfdr[i] * np[i]->x[1] ;
                        }
                        dS = g[ig].w * sqrt(J00 * J00 + J01 * J01) ;
                        update_BKe(elmntp,&fv,&ft,ns,*ntf,np,nvar,dS)
                }
        }
        if(N_trans > 0) {
                tdt = tvals.theta * tvals.dt ;
                mtdt = tvals.dt * (1.0 - tvals.theta) ;
                for(i=0;i<*ntf;i++) {
                        FE(i) *= tvals.dt ;
                        for(j=0;j<ns;j++) {
                                FE(i) += (-mtdt * KE(i,j)) * (elmntp->nps[j])-
>u[nvar] ;
                                SE(i,j) = tdt * KE(i,j) ;
                        }
                }
        }
        return(ns) ;
}


int     initm(K,n)
double *K ;
int     n ;

{

        register int  i ;

        for(i=0;i<n;i++)
                        *(K++) = 0.0 ;
        return(0) ;
}


int             NdinEl(n,elpp)
int             n ;
struct element         *(*elpp) ;

{
        int     i, rc;
        struct element          *selp ;

        rc = 0 ;
        selp = *elpp ;
        while(selp != NULL) {
                for(i=0;i<selp->nnds;i++) {
                        if(n == (selp->nps[i])->n) {
                                rc = 1 ;
                                *elpp = selp ;
                                break ;
                        }
                }
                if(rc == 1)
                        return(i) ;
                selp = selp->nextelp ;
        }
        return(-1) ;
}


int             get_cline(elp,cval,riv,nvar,tol,xl,yl
```

```
int             niv, nvar ;
double          xl[], yl[], tol, cval ;
struct element        *elp ;

{
        int                     i, j, nipts, rc ;
        double                  x, y, Dx, Dy, dfdr, dfds ;
        struct node             *np[MNSF] ;
        struct shapefuncs       sf ;
        struct gausspts                 xg ;

        if(niv == 0)
                tol = -1.0 ;
        nipts = isvalinel(elp,np,cval,nvar,tol,&dfdr,&dfds,xl,yl) ;
        switch (nipts) {
                case    0 :
                        break ;
                case 2 :
                        Dx = xl[1] - xl[0] ;
                        Dy =  yl[1] - yl[0] ;
                        xl[niv+1] = xl[1] ;
                        yl[niv+1] = yl[1] ;
                        for(i=1;i<=niv;i++) {
                                xl[i] = xl[i-1] + (xl[niv+1] - xl[i-1]) * 1 / (niv
- i + 2) ;
                                yl[i] = yl[i-1] + (yl[niv+1] - yl[i-1]) * 1 / (niv
- i + 2) ;
                                find_2Dloc(np,nvar+N.dims,elp-
>gtype,cval,tol,Dx,Dy,dfdr,dfds,&xl[i],&yl[i]) ;
                        }
                        xg.z = 0.0 ;
                        xg.w = -1.0 ;
                        for(i=0;i<=niv+1;i++) {
                                xg.x = xl[i] ;
                                xg.y = yl[i] ;
                                rc = get_shape(elp->gtype,&xg,&sf) ;
                                xl[i] = yl[i] = 0.0 ;
                                for(j=0;j<sf.dof;j++) {
                                        xl[i] += sf.f[j] * np[j]->x[0] ;
                                        yl[i] += sf.f[j] * np[j]->x[1] ;
                                }
                        }
                        break ;
        }

        return(nipts) ;
}

int             isvalinel(elp,np,cval,nvar,tol,dfdr,dfds,xl,yl)
int             nvar ;
double          xl[], yl[], tol, cval, *dfdr, *dfds ;
struct node     *np[] ;
struct element        *elp ;

{
        int             j, k, nbps, etyp, ng, next, prev, ncpts, aj[MNSF],
ak[MNSF] ;
        double          Dx, Dy, r, s, t, f, ff, fs, ft ;
        struct gausspts       g[MGPTS] ;

        nbps = nbounds(elp->gtype) ;
        etyp = elp->gtype ;
        for(j=0;j<elp->nnds;j++)
```

```
			np[j] =  elp->nps[j] ;
		prev = (cval <= np[nbps-1]->u[nvar]) ? 1 : -1 ;
		ncpts = 0 ;
		for(j=0;j<nbps;j++) {
			next = (cval <= np[j]->u[nvar]) ? 1 : -1 ;
			if(next != prev) {
				aj[ncpts] = j ;
				if( j == 0 )
					ak[ncpts] = nbps-1 ;
				else
					ak[ncpts] = j - 1 ;
				ncpts++ ;
			}
			prev = next ;
		}
		if((ncpts > 0) && (tol > 0.0) ) {
			ng = get_gspts(etyp,g) ;
			*dfdr = *dfds = 0.0 ;
			for(j=0;j<ng;j++) {
				r = g[j].x ;
				s = g[j].y ;
				t = g[j].z ;
				get_value(np,nvar,etyp,r,s,t,2,&f,&fr,&fs,&ft) ;
				*dfdr += fr ;
				*dfds += fs ;
			}
			*dfdr /= ng ;
			*dfds /= ng ;
		}
		for(j=0;j<ncpts;j++) {

		GetIPoint(aj[j],ak[j],np,etyp,cval,nvar,&xl[j],&yl[j],&Dx,&Dy) ;
				if(tol > 0.0)

		find_2Dloc(np,nvar+Ndims,etyp,cval,tol,Dx,Dy,*dfdr,*dfds,&xl[j],&yl[j])
;
		}
		return(ncpts) ;
	}

	int		GetIPoint(j,k,np,eltype,u,n,xp,yp,Dx,Dy)
	double		*xp, *yp, *Dx, *Dy, u ;
	int		j, k, n, eltype;
	struct node	*np[] ;

	{
		double x1, y1, x2, y2 ;

		get_nlcs(eltype,k,&x1,&y1) ;
		get_nlcs(eltype,j,&x2,&y2) ;
		*xp = x1 + (u - np[k]->u[n])/(np[j]->u[n] - np[k]->u[n])*(x2 - x1) ;
		*yp = y1 + (u - np[k]->u[n])/(np[j]->u[n] - np[k]->u[n])*(y2 - y1) ;
		*Dx = y2 - y1 ;
		*Dy = x1 - x2 ;
		return(0) ;
	}

	int		find_2Dloc(np,nvar,eltype,fc,tol,Dr,Ds,dfdr,dfds,r,s)
	double		fc, tol, Dr, Ds, dfdr, dfds, *r, *s ;
	int		eltype, nvar ;
	struct node	*np[] ;

	{
```

```
	int	i, eq, imax = 100 ;
	double t, f, ddr, dds, dfdt, df, dr, ds, sl ;

	printf(" in find_2Dloc...") ;

	t = 0.0 ;
	if(fabs(Dr) >= fabs(Ds)) {
		eq = 2 ;
		sl = -Ds/Dr ;
	}
	else {
		eq = 1 ;
		sl = -Dr/Ds ;
	}
	for(i=0;i<imax;i++) {
		if( get_value(np,nvar,eltype,*r,*s,t,2,&f,&ddr,&dds,&dfdt)  != 0)
			return(-1);
		if(fabs(df = f - fc) < tol) {
			printf(" fc %f f %f df %f tol %f\n",fc,f,df,tol);
			break ;
		}
		if(eq == 2) {
			ds = df / (ddr*sl + dds) ;
			dr = sl*ds ;
		}
		else {
			dr = df / (ddr + sl*dds) ;
			ds = sl * dr ;
		}
		if(fabs(dr) > 0.5 || fabs(ds) > 0.5) {
			if(eq == 2) {
				ds = df / (dfdr*sl + dfds) ;
				dr = sl*ds ;
			}
			else {
				dr = df / (dfdr + sl*dfds) ;
				ds = sl * dr ;
			}
		}
		if(fabs(dr) > 0.5 || fabs(ds) > 0.5)
			return(-2) ;
		*r -= dr ;
		*s -= ds ;
		printf(" %d ",i) ;
	}
	printf(" ...out\n") ;
	return(i) ;
}

int		get_value(np,nvar,eltype,r,s,t,code,f,dfdr,dfds,dfdt)
double		*f, *dfdr, *dfds, *dfdt, r, s, t ;
int		eltype, nvar, code ;
struct node	*np[] ;

{
	int			i, rc ;
	struct shapefuncs	sf ;
	struct gausspts			x ;

	x.x = r ;
	x.y = s ;
	x.z = t ;
	x.w = 0.0 ;
```

```
	if(code == 0)
		x.w = -1.0 ;
	rc = get_shape(eltype,&x,&sf) ;

	switch (code) {
		case  3 :
			*dfdt = 0.0 ;
			for(i=0;i<sf.dof;i++)
				*dfdt += sf.dfdt[i] * np[i]->x[nvar] ;
		case 2 :
			*dfds = 0.0 ;
			for(i=0;i<sf.dof;i++)
				*dfds += sf.dfds[i] * np[i]->x[nvar] ;
		case 1 :
			*dfdr = 0.0 ;
			for(i=0;i<sf.dof;i++)
				*dfdr += sf.dfdr[i] * np[i]->x[nvar] ;
		case 0 :
			*f = 0.0 ;
			for(i=0;i<sf.dof;i++)
				*f += sf.f[i] * np[i]->x[nvar] ;
	}
	return(rc) ;
}
```

fetest.c

```
#include "fe.h"

#define KE(A,B)        ( *(ep.K + ep.n*A + B) )
#define SE(A,B)        ( *(ep.S + ep.n*A + B) )
#define ME(A,B)        ( *(ep.M + ep.n*A + B) )
#define FE(B)                ( *(ep.F + B) )
#define TRUE           -1

extern struct control              N ;
extern struct pointers             gp;
extern struct RegMesh              Mesh ;
extern struct fxnodes              *first_fnp ;
extern int                         Nukns ;
extern struct elmpointers    ep ;
extern struct transient            tvals ;
extern struct RegMesh              Mesh ;
extern unsigned long               d_time ;


int            list_vars(nvar)
int            nvar ;

{
        int         i, j ;
        struct node   *np ;
        FILE    *fp, *put_fptr() ;

        fp = put_fptr(1)   ;

        fprintf(fp," Node\t Value \n\n");

        for(i=0;i<N.nodes;i++)  {

                        for(j=0;j<N.dims;j++)
                                put_dbl(fp,(double) gp.iptrs[i]->x[j]) ;
                        for(j=0;j<N.vars;j++)
                                put_dbl(fp,(double) gp.iptrs[i]->u[j]) ;
                        fprintf(fp,"\n") ;

        }
        printf("ok after variables\n") ;
        fclose(fp) ;
        return ;
}

int            test_contour(code)
int            code ;

{
        int    nelm,niv,nvar, ni, i ;
        double tol, xl[10], yl[10], cval ;
        struct element        *elp ;

        printf(" Input el#, #iv, #var, value, tol \n") ;
        scanf(" %d %d %d %lf %lf",&nelm,&niv,&nvar,&cval,&tol) ;
        elp = gp.El ;
        for(i=0;i<N.elms;i++) {
                if(elp->n == nelm)
                        break ;
                elp = elp->nextelp ;
        }
        ni = get_cline(elp,cval,niv,nvar,tol,xl,yl) ;
        printf(" number of intersections = %d\n",ni) ;
```

```
		if(ni == 2) {
			printf(" contour line\n") ;
			for(i=0;i<niv+2;i++)
				printf(" %f  %f \n",xl[i],yl[i]) ;
		}
		return(0) ;
}


int		test_trans(nvar)
int		nvar ;

{
		int	i, nt ;

		printf(" Time = %lf \n", tvals.t) ;
		printf(" Input theta, delta t and magnification factor \n") ;
		scanf(" %lf %lf %lf",&tvals.theta,&tvals.dt,&tvals.dtfac) ;
		printf(" Input number of time steps \n") ;
		scanf(" %d",&tvals.nsteps) ;
		transient(0,1) ;
		return(0) ;
}

int		test_mesh(eltype)
int		eltype ;

{
		int				i, ni ;
		struct element			*elp ;

		Mesh.eltype = eltype ;
		printf(" Input nx, nbx, ny, nby \n") ;
		scanf(" %d %d %d %d",&Mesh.nx,&Mesh.nbx,&Mesh.ny,&Mesh.nby) ;
		ni = MkLapMesh(&Mesh) ;
		printf(" return code = %d\n",ni) ;
		return(0) ;
}


int		test_map(maptype)
int		maptype ;

{
		int				i, n, k, l, numrows, numcols ;
		struct blockmap		*theblock ;

		Mesh.maptype = maptype ;
		theblock = Mesh.firstmap ;
		for(k=0;k<Mesh.nby;k++) {
			for(l=0;l<Mesh.nbx;l++) {
				printf(" Input macro element nodes x,y coord. pairs\n") ;
				n = nsf(maptype) ;
				for(i=0;i<n;i++) {
					scanf(" %f %f",&(theblock->xnodes[i]),&(theblock-
>ynodes[i])) ;
				}
				map_mesh(theblock,Mesh.maptype,k,l) ;
				theblock++ ;
			}
		}
		return(0) ;
}
```

```
int             set_bc(code)
int             code ;

{
        int                             bestart, beend,i, j, bctype ;
        double                  bvalue ;
        struct belement         *belp ;

        while (TRUE) {
                printf(" Input first, last boundary elements, code and value\n")
;
                scanf(" %d %d %d %lf",&bestart,&beend,&bctype,&bvalue) ;
                if(bestart <= 0)
                        break ;
                belp = gp.B ;
                for(i=0;i<N.belms;i++) {
                        if(belp->n == bestart)
                                break ;
                        belp = belp->nextbelp ;
                }
                do {
                        if(bctype == 0) {
                                belp->bcs[0] = 0 ;
                                belp->bcs[1] = 1 ;
                                for(j=0;j<belp->nnds;j++) {
                                        (belp->nps[j])->u[1] = bvalue ;
                                }
                        }
                        else {
                                belp->bcs[0] = 1 ;
                                belp->bcs[1] = 0 ;
                                for(j=0;j<belp->nnds;j++) {
                                        (belp->nps[j])->u[0] = bvalue ;
                                }
                        }
                        if(belp->n == beend)
                                break;
                        belp = belp->nextbelp ;
                }
                while(belp->n <= N.belms) ;
        }
        return(0) ;
}


int     test_real_stream(eltype)
int eltype ;
{
        int                                     i, j, k, l, numrows, numcols, ni,
n_x, n_z;
        int                                     maptype, get_int(), is, iend, off_set

        int                                     actual_nbx ;
        struct node                     *np, *node_u ;
        struct blockmap                 *theblock ;
        FILE                            *f, *get_fptr(),*put_fptr() ;
        struct cross_section            *x_all,  *x_c, *x_n, *x_n_n ;
        double                                  get_dbl() ;
        double                                  diag_l, dx_ave, dy_ave, co, si,
E_long, E_lat ;
        double                                  x1 , x0, y1, y0, x2, x3, y2, y3;
        double                                  constant = 100.0 ;
```

```
        char                        smooth_code = 'n' ;

        printf( " Input the type of mapping desired (211, 221)\n->");
        scanf(" %d",&maptype) ;
        if(maptype == 211){
                off_set  = 0 ;
                printf(" Using linear mapping\n");
        }
        else {
                printf(" Using quadratic mapping\n");
                off_set = 1 ;
        }
        printf(" Do   you want smoothing ? \n  ->") ;
        scanf(" %c",&smooth_code) ;

        f = get_fptr(1) ;

        Mesh.eltype = eltype ;
        Mesh.maptype = maptype ;
        Mesh.nx = get_int(f) ;
        actual_nbx = get_int(f) ;
        if((x_all = (struct cross_section *) lmalloc(
        (long int)(actual_nbx+1)*sizeof(struct cross_section) )) == NULL) /*
        NON STANDARD FUNCTION         */
                return(-1)  ;
        if( actual_nbx % 2 == 0 && maptype == 221)
                Mesh.nbx = actual_nbx / 2 ;
        else
                Mesh.nbx = actual_nbx ;
        Mesh.ny = get_int(f) ;
        Mesh.nby = get_int(f) ;
        free(Mesh.firstmap);
        Mesh.firstmap = (struct blockmap *)calloc(Mesh.nbx*Mesh.nby
                                                            ,sizeof(struct
blockmap)) ;

        printf(" Generation a mesh with %d by %d blocks\n",Mesh.nbx,Mesh.nby) ;
        ni = MkLapMesh(&Mesh) ;

        if(ni != 0)
                exit(0) ;

        if(maptype == 221 && (actual_nbx % 2 != 0))
                ni =First_map(&Mesh) ;

        if(maptype == 221){
                if(actual_nbx % 2 == 0)
                        off_set = 0 ;
                else {
                        Mesh.nbx /= 2 ;
                        off_set = 1 ;
                        Mesh.nbx += off_set ;
                }
        }

        n_x = actual_nbx + 1 ;
        n_z = Mesh.nby + 1 ;
        for(i=0;i<n_x;i++){
                x_c = x_all + i ;
                for(j=0;j<n_z;j++){
                        x_c->x[j] = get_dbl(f)+constant ;
                        x_c->z[j] = get_dbl(f)+constant ;
                        x_c->h[j] = get_dbl(f)+constant ;
```

```
                x_c->u[j] = get_dbl(f)+constant ;
                x_c->v[j] = get_dbl(f)+constant ;
                x_c->E_long[j] = get_dbl(f)+constant ;
                x_c->E_lat[j] = get_dbl(f)+constant;
            }
            x_c->next_x = x_c + 1;
        }

        fclose(f) ;
        Mesh.maptype = maptype ;
        theblock = Mesh.firstmap ;
        x_c = x_all ;


        for(k=0;k<Mesh.nbx;k++) {
            x_n = x_c->next_x ;

            if(k > (Mesh.nbx-2) && off_set == 1)
                    Mesh.maptype = 211 ;

            printf(" mapping type = %d\n",Mesh.maptype) ;

            for(l=0;l<Mesh.nby;l++) {
                switch(Mesh.maptype){
                    case  211 :
                        theblock->xnodes[0] = x_n->x[l] ;
                        theblock->xnodes[1] = x_n->x[l+1] ;
                        theblock->xnodes[2] = x_c->x[l+1] ;
                        theblock->xnodes[3] = x_c->x[l] ;

                        theblock->ynodes[0] = x_n->z[l] ;
                        theblock->ynodes[1] = x_n->z[l+1] ;
                        theblock->ynodes[2] = x_c->z[l+1] ;
                        theblock->ynodes[3] = x_c->z[l] ;

                        theblock->h[0] = x_n->h[l] ;
                        theblock->h[1] = x_n->h[l+1] ;
                        theblock->h[2] = x_c->h[l+1] ;
                        theblock->h[3] = x_c->h[l] ;

                        theblock->u[0] = x_n->u[l] ;
                        theblock->u[1] = x_n->u[l+1] ;
                        theblock->u[2] = x_c->u[l+1] ;
                        theblock->u[3] = x_c->u[l] ;

                        theblock->v[0] = x_n->v[l] ;
                        theblock->v[1] = x_n->v[l+1] ;
                        theblock->v[2] = x_c->v[l+1] ;
                        theblock->v[3] = x_c->v[l] ;

                        theblock->E_x[0] = x_n->E_long[l] ;
                        theblock->E_x[1] = x_n->E_long[l+1] ;
                        theblock->E_x[2] = x_c->E_long[l+1] ;
                        theblock->E_x[3] = x_c->E_long[l] ;

                        theblock->E_y[0] = x_n->E_lat[l] ;
                        theblock->E_y[1] = x_n->E_lat[l+1] ;
                        theblock->E_y[2] = x_c->E_lat[l+1] ;
                        theblock->E_y[3] = x_c->E_lat[l] ;

                        theblock->E_xy[0] = constant ;
                        theblock->E_xy[1] = constant ;
                        theblock->E_xy[2] = constant ;
```

```
                theblock->E_xy[3] = constant ;

                theblock->E_yx[0] = constant ;
                theblock->E_yx[1] = constant ;
                theblock->E_yx[2] = constant ;
                theblock->E_yx[3] = constant ;
                break ;

        case 221 :
                x_n_n = x_n->next_x ;

                theblock->xnodes[0] = x_n_n->x[l] ;
                theblock->xnodes[1] = (x_n_n->x[l]+x_n_n-
>x[l+1])/2. ;
                theblock->xnodes[2] = x_n_n->x[l+1] ;
                theblock->xnodes[3] = x_n->x[l+1] ;
                theblock->xnodes[4] = x_c->x[l+1] ;
                theblock->xnodes[5] = (x_c->x[l+1] + x_c-
>x[l])/2. ;
                theblock->xnodes[6] = x_c->x[l] ;
                theblock->xnodes[7] = x_n->x[l] ;

                theblock->ynodes[0] = x_n_n->z[l] ;
                theblock->ynodes[1] = (x_n_n->z[l]+x_n_n-
>z[l+1])/2. ;
                theblock->ynodes[2] = x_n_n->z[l+1] ;
                theblock->ynodes[3] = x_n->z[l+1] ;
                theblock->ynodes[4] = x_c->z[l+1] ;
                theblock->ynodes[5] = (x_c->z[l+1] + x_c-
>z[l])/2. ;
                theblock->ynodes[6] = x_c->z[l] ;
                theblock->ynodes[7] = x_n->z[l] ;

                theblock->h[0] = x_n_n->h[l] ;
                theblock->h[1] = (x_n_n->h[l]+x_n_n-
>h[l+1])/2. ;
                theblock->h[2] = x_n_n->h[l+1] ;
                theblock->h[3] = x_n->h[l+1] ;
                theblock->h[4] = x_c->h[l+1] ;
                theblock->h[5] = (x_c->h[l+1] + x_c-
>h[l])/2. ;
                theblock->h[6] = x_c->h[l] ;
                theblock->h[7] = x_n->h[l] ;

                theblock->u[0] = x_n_n->u[l] ;
                theblock->u[1] = (x_n_n->u[l]+x_n_n-
>u[l+1])/2. ;
                theblock->u[2] = x_n_n->u[l+1] ;
                theblock->u[3] = x_n->u[l+1] ;
                theblock->u[4] = x_c->u[l+1] ;
                theblock->u[5] = (x_c->u[l+1] + x_c-
>u[l])/2. ;
                theblock->u[6] = x_c->u[l] ;
                theblock->u[7] = x_n->u[l] ;

                theblock->v[0] = x_n_n->v[l] ;
                theblock->v[1] = (x_n_n->v[l]+x_n_n-
>v[l+1])/2. ;
                theblock->v[2] = x_n_n->v[l+1] ;
                theblock->v[3] = x_n->v[l+1] ;
                theblock->v[4] = x_c->v[l+1] ;
                theblock->v[5] = (x_c->v[l+1] + x_c-
>v[l])/2. ;
```

```
                                theblock->v[6] = x_c->v[l] ;
                                theblock->v[7] = x_n->v[l] ;

                                theblock->E_x[0] = x_n_n->E_long[l] ;
                                theblock->E_x[1] = (x_n_n->E_long[l]+x_n_n-
>E_long[l+1])/2. ;
                                theblock->E_x[2] = x_n_n->E_long[l+1] ;
                                theblock->E_x[3] = x_n->E_long[l+1] ;
                                theblock->E_x[4] = x_c->E_long[l+1] ;
                                theblock->E_x[5] = (x_c->E_long[l+1] + x_c-
>E_long[l])/2. ;
                                theblock->E_x[6] = x_c->E_long[l] ;
                                theblock->E_x[7] = x_n->E_long[l] ;

                                theblock->E_y[0] = x_n_n->E_lat[l] ;
                                theblock->E_y[1] = (x_n_n->E_lat[l]+x_n_n-
>E_lat[l+1])/2. ;
                                theblock->E_y[2] = x_n_n->E_lat[l+1] ;
                                theblock->E_y[3] = x_n->E_lat[l+1] ;
                                theblock->E_y[4] = x_c->E_lat[l+1] ;
                                theblock->E_y[5] = (x_c->E_lat[l+1] + x_c-
>E_lat[l])/2. ;
                                theblock->E_y[6] = x_c->E_lat[l] ;
                                theblock->E_y[7] = x_n->E_lat[l] ;

                                theblock->E_xy[0] = constant ;
                                theblock->E_xy[1] = constant ;
                                theblock->E_xy[2] = constant ;
                                theblock->E_xy[3] = constant ;
                                theblock->E_xy[4] = constant ;
                                theblock->E_xy[5] = constant ;
                                theblock->E_xy[6] = constant ;
                                theblock->E_xy[7] = constant ;

                                theblock->E_yx[0] = constant ;
                                theblock->E_yx[1] = constant ;
                                theblock->E_yx[2] = constant ;
                                theblock->E_yx[3] = constant ;
                                theblock->E_yx[4] = constant ;
                                theblock->E_yx[5] = constant ;
                                theblock->E_yx[6] = constant ;
                                theblock->E_yx[7] = constant ;

                                break ;
                        }

                map_stream(theblock,Mesh.maptype,l,k) ;
                theblock++ ;
        }

    if(maptype == 221)
        x_c = x_n->next_x ;
    else
        x_c = x_c->next_x ;
    }

    np = gp.N ;
    for(i=0;i<N.nodes;i++){
        np->x[0] -= constant ;
        np->x[1] -= constant ;
        np->p[0] -= constant ;
        np->p[1] -= constant ;
        np->p[2] -= constant ;
```

```
                np->p[3] -= constant ;
                np->p[4] -= constant ;
                np->p[5] -= constant ;
                np->p[7] -= constant ;
                np = np->nextnp ;
        }

        ni = Get_vel_vec() ;

        if( smooth_code == 'y'){
                printf(" Performing Smoothing !\n") ;
                for(i=0;i<5;i++)
                        Smooth_Mesh(actual_nbx,Mesh.ny) ;
        }

        free(Mesh.firstmap);

        return(0) ;
}

int             Smooth_Mesh(nx,ny)
int             nx, ny ;
{
        register int  i,j,k ;
        int                     index, i_1,i_2,i_3,i_4 ;

        for(i=2;i< nx;i++){
                for(j=2;j< ny;j++){
                        index = (i-1) * (ny+1) + j - 1 ;
                        i_1 = i*(ny+1) + j - 1 ;
                        i_2 = (i-1)  * (ny+1) + j ;
                        i_3 = (i-2)*(ny+1) + j - 1 ;
                        i_4 = (i-1) * (ny+1) + j - 2 ;

                        gp.iptrs[index]->x[0] = (   gp.iptrs[i_1]->x[0] / 2.
                                                +      gp.iptrs[i_2]->x[0]
                                                +      gp.iptrs[i_3]->x[0] / 2.
                                                +      gp.iptrs[i_4]->x[0] ) / 3.0 ;
                        gp.iptrs[index]->x[1] = (   gp.iptrs[i_1]->x[1]
                                                +      gp.iptrs[i_2]->x[1] / 2.
                                                +      gp.iptrs[i_3]->x[1]
                                                +      gp.iptrs[i_4]->x[1] / 2. ) /
3.0 ;

                        for(k=0;k<N.params;k++){
                                gp.iptrs[index]->p[k] = (   gp.iptrs[i_1]->p[k]
                                                        +      gp.iptrs[i_2]->p[k]
                                                        +      gp.iptrs[i_3]->p[k]
                                                        +      gp.iptrs[i_4]->p[k] ) /
4.0 ;
                        }
                }
        }
        return(0) ;
}

int     Get_vel_vec()
{
        int                                     i, j, k, l, ni, n_x, n_z, nm ;
        int                                     get_int(), is, iend, off_set,
index,n1,n2 ;
        register struct node        *np1, *np2 ;
```

```
        double                               diag_l, dx_ave, dy_ave, co, si,
E_long, E_lat ;
        double                               x1 , x0, y1, y0, x2, x3, y2, y3,
q_old;
        register struct element     *elp ;
        double                      tcl, xl[10], yl[10], cval ;


        for(i=1;i<=(Mesh.nx+1);i++){
                q_old = 0.0 ;
                for(j=1;j<=(Mesh.ny);j++){
                        index = (i - 1) * (Mesh.ny+1) + j - 1;
                        np1 = gp.N ;
                        for(k=0;k<N.nodes;k++){
                                if(np1->n == (index+1))
                                        break ;
                                else
                                        np1 = np1->nextnp ;
                        }
                        np2 = np1->nextnp ;

                        x1 = np1->x[0];
                        x2 = np2->x[0];
                        y1 = np1->x[1];
                        y2 = np2->x[1];
                        diag_l = sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)) ;
                        q_old += (np1->p[0]+np2->p[0])
                                *diag_l*(np1->p[7]+np2->p[7])/4.0 ;

                        np2->u[0] = q_old ;
                }
        }
        for(i=1;i<=(Mesh.nx+1);i++){
                q_old = 0.0 ;
                iend = (i-1) *( Mesh.ny+1) + Mesh.ny ;
                np2 = gp.N ;
                for(k=0;k<N.nodes;k++){
                        if(np2->n==(iend+1))
                                break ;
                        else
                                np2 = np2->nextnp ;
                }
                for(j=1;j<=(Mesh.ny+1);j++){
                        index = (i - 1) *(Mesh.ny+1) + j - 1;
                        np1 = gp.N ;
                        for(k=0;k<N.nodes;k++){
                                if(np1->n==(index+1))
                                        break;
                                else
                                        np1 = np1->nextnp ;
                        }
                        np1->u[0] /= np2->u[0] ;

                }
        }
        printf(" Finished Obtaining q\n") ;

        np1 = gp.N ;
        for(k=0;k<N.nodes;k++){
                np1->ui = 0 ;
                elp = gp.El ;

                for(i=0;i<N.elms;i++){
```

```
                        for(j=0;j<elp->nnds;j++){
                                if((elp->nps[j])->n == np1->n)
                                        np1->ui += 1 ;
                        }
                        elp = elp->nextelp ;
                }
                np1 = np1->nextnp ;
        }


        tol = 0.0001 ;
        elp = gp.El ;
        for(i=0;i<N.elms;i++){
                for(j=0;j<4;j++){
                        ni = get_cline(elp,(elp->nps[j])->u[0],0,0,tol,x1,y1) ;
                        dx_ave = x1[1] - x1[0] ;
                        dy_ave = y1[1] - y1[0] ;
                        if((fabs(dx_ave) > tol && fabs(dy_ave) > tol)){
                                switch(ni){
                                        case 2:
                                                (elp->nps[j])->p[6] +=
atan(dy_ave/dx_ave) ;
                                        break ;
                                }
                        }
                }
                elp = elp->nextelp ;
        }
        is = Mesh.ny + 1 ;
        iend = N.nodes - Mesh.ny -1 ;

        np2 = gp.N ;
        for(i=0;i<iend;i++){
                if(i > is && np2->ui != 1)
                        np2->p[6] /= 2. ;
                np2 = np2->nextnp ;
        }
        printf(" Finished Obtaining the angles\n") ;



        np2 = gp.N ;
        for(i=0;i<N.nodes;i++){
                co = cos(np2->p[6]) ;
                si = sin(np2->p[6]) ;

                E_long = np2->p[0] ;
                E_lat = np2->p[1] ;

                np2->p[0] = E_long * co - E_lat  * si ;
                np2->p[1] = E_long * si + E_lat  * co ;

                E_long = np2->p[2] ;
                E_lat = np2->p[5] ;

                np2->p[2] = E_long *co*co + E_lat *si*si ;
                np2->p[3] = np2->p[4] = (E_long - E_lat) *si*co ;
                np2->p[5] = E_long *si*si + E_lat *co*co ;

                np2->u[0] = 0.0 ;
                np2->p[6] = 0.0 ;

                np2 = np2->nextnp ;
```

```
        }
        return(0) ;
}

int                      map_stream(theblock,maptype,rownum,colnum)
int                      maptype, rownum, colnum ;
struct blockmap      *theblock ;

{
        struct node          *np ;
        struct gausspts      g ;
        struct shapefuncs    sf ;
        int                         i, j ;

        np = gp.N ;
        g.z = 0.0 ;
        g.w = -1.0 ;
        for(i=0;i<N.nodes;i++) {
                g.x = np->x[0] - 2.0 * colnum ;
                g.y = np->x[1] - 2.0 * rownum ;
                if(get_shape(maptype,&g,&sf) == 0) {
                        np->x[0] = 0.0 ;
                        np->x[1] = 0.0 ;
                        np->p[0] = 0.0 ;
                        np->p[1] = 0.0 ;
                        np->p[2] = 0.0 ;
                        np->p[3] = 0.0 ;
                        np->p[4] = 0.0 ;
                        np->p[5] = 0.0 ;
                        np->p[7] = 0.0 ;

                        for(j=0;j<sf.dof;j++) {
                                np->x[0] += sf.f[j] * theblock->xnodes[j] ;
                                np->x[1] += sf.f[j] * theblock->ynodes[j] ;
                                np->p[0] += sf.f[j] * theblock->u[j] ;
                                np->p[1] += sf.f[j] * theblock->v[j] ;
                                np->p[2] += sf.f[j] * theblock->E_x[j] ;
                                np->p[3] += sf.f[j] * theblock->E_xy[j] ;
                                np->p[4] += sf.f[j] * theblock->E_yx[j] ;
                                np->p[5] += sf.f[j] * theblock->E_y[j] ;
                                np->p[7] += sf.f[j] * theblock->h[j] ;
                        }
                }
                np = np->nextnp ;
        }
        return(0) ;
}

int             First_map(theMesh)
struct RegMesh *theMesh ;
{
        int                         i, n, k, l, numrows, numcols, new_nbx,
off_set ;
        struct blockmap      *theblock ;
        float                ks, ls ;
        struct                      node    *nodep ;

        theblock = (struct blockmap *)calloc((theMesh->nbx*theMesh-
>nby),sizeof(struct blockmap)) ;

        new_nbx = theMesh->nbx/2 ;

        if(new_nbx*2 == theMesh->nbx)
```

```
			return(0) ;
		else
			off_set = 1 ;

		for(k=0;k<theMesh->nbx;k++) {
			for(l=0;l<theMesh->nby;l++) {

						ks = 100 + (float) k ;
						ls = (float) l ;

						theblock->xnodes[0] = ks    ;
						theblock->xnodes[1] = ks   ;
						theblock->xnodes[2] = ks - 1. ;
						theblock->xnodes[3] = ks - 1. ;

					if(k == (theMesh->nbx-1)){
						theblock->xnodes[0] = ks + 1. ;
						theblock->xnodes[1] = ks + 1. ;
						theblock->xnodes[2] = ks - 1. ;
						theblock->xnodes[3] = ks - 1.   ;
					}

						theblock->ynodes[0] = 100. + 2.*ls -1. ;
						theblock->ynodes[1] = 100. +2.*ls +1. ;
						theblock->ynodes[2] = 100. +2.*ls +1. ;
						theblock->ynodes[3] = 100. +2.*ls -1. ;

				map_mesh(theblock,211,l,k) ;
				theblock++ ;
			}
		}

		nodep = gp.N ;
		for(i=0;i<N.nodes;i++){
			nodep->x[0] -= 100.0 ;
			nodep->x[1] -= 100.0 ;
			nodep = nodep->nextnp ;
		}
		free(theblock) ;
		return(0) ;
}


int	test_analytical(nvar)
int	nvar ;
{
		int		i, j, k, d_n ;
		double eta, Q, neta, w, w_m_n, w_p_n ;
		double		erf(), value, first_x ;
		struct belement		*belp ;
		struct node	*nodep ;


		printf(" ||||||||||				WARNING
||||||||||| \n\n") ;
		printf(" |||| You are about to solve for the Analytical Solution||||\n")
;
		printf("The rest of this program will treat p[2] as the \n") ;
		print(" non-dimensional cummulative discharge !\n") ;


		print(" Please input the discharge and the fraction for the Line
source?\n->") ;
```

```
        scanf(" %lf %lf", &Q, &w) ;
        printf(" Discharge = %lf and w = %lf\n",Q,w) ;

        d_n = -1000000 ;
        belp = gp.B ;
        for(i=0;i<N.belms;i++) {
                if((belp->nps[0]->n - belp->nps[1]->n) > d_n)
                        d_n = (belp->nps[0]->n - belp->nps[1]->n) ;
                belp = belp->nextbelp ;
        }

        first_x = gp.iptrs[d_n]->x[0] - gp.iptrs[0]->x[0] ;

        printf(" source at x = %lf\n",first_x) ;


        nodep = gp.N ;
        for(i=0;i<N.nodes;i++){

                if( i % 10 == 0.0 )
                        printf(" Finished %d nodes\n",i) ;

                neta = 4.*nodep->p[5] * (nodep->x[0]-first_x)/ Q/Q ;


                if(neta <= 0.0)
                        goto SKIP ;

                eta = sqrt(fabs(neta)) ;
                w_m_n = w - nodep->p[0] ;
                w_p_n = w + nodep->p[0] ;

                printf(" for node %d, neta = %lf, eta = %lf, w_m_n = %lf, w_p_n =
%lf\n",
                                i, neta, eta, w_m_n, w_p_n) ;
/*                      printf(" value = %lf \n",value) ;
*/
                value = erf(w_m_n / eta) + erf(w_p_n / eta) ;
                for(j=1;j<3;j++) {
                        value += ( erf((2*j + w_m_n) / eta) + erf((2*j + w_p_n)) /
eta ) ;
                        value -= ( erf((2*j - w_p_n) / eta) + erf((2*j - w_m_n)) /
eta) ;
                }

                nodep->u[0] = value / 2. / w ;
SKIP :
                nodep = nodep->nextnp ;
        }
        return(0) ;
}

double          erf(x)
double          x ;
{
        double          v, result, fact = 1.0, delta ;

        delta = 0.0001 ;

        v = 0.0 ;
        if(x < 0.0){
                fact = -1.0 ;
                x *= fact ;
```

```
	}

	result = 0.0 ;
	while( v <= x ){
		result +=2/1.772453851*delta*(exp(-v*v)+exp(-
(v+delta)*(v+delta)))/2. ;
		delta *= 1.001 ;
		v += delta ;
	}
	return(result*fact) ;
}


int	special_out(nin)
int nin;
{
	int			err,i,dir ;
	FILE			*fptr, *put_fptr() ;
	struct node	*np ;
	struct element		*elp ;
	float		val;
	char		l_dir;

	if ( (fptr = put_fptr(1)) != NULL){
		printf("Please input the direction along which the output is
desired!");

		printf("\n >");
		scanf(" %c",&l_dir);
		if (l_dir == 'x'){
			printf("\n Input the y coordinate.");
			dir = 1;
		}
		else {
			printf("\n Input the x coordinate.");
			dir = 0;
		}
		printf("\n >");
		scanf(" %f",&val);
		np = gp.N;
		for (i=0;i<N.nodes;i++){
			if (np->x[dir] == val){
				if (nin == 1){
					printf(" %f\t%f\t%f\n",np->x[0], np-
>x[1],np->u[0]);
				}
				fprintf(fptr," %f\t%f\t%20.14g\n",np->x[0], np-
>x[1],np->u[0]);
			}
			np = np->nextnp;
		}
		if (nin == 1 || nin == 2){
			fclose(fptr);
			return(1);
		}
		return(1);
	}
	else {
		printf("Couldn't open the specified file");
		return(-1);
	}
}
```

fe_nupf.c

```
#include "fe.h"


extern struct control       N ;

int             QU1DPick(elmntp,theElp,ntfp,dirp)
struct element        *elmntp, *theElp ;
int             *ntfp, *dirp ;

{
        int             j;
        double          q01 ;
        struct node     *np[MNSF] ;

        theElp->gtype = 111 ;
        *ntfp = 2;
        for(j=0;j<elmntp->nnds;j++)
                np[j] = elmntp->nps[j] ;

        for(j=0;j<N.params;j++)
                theElp->p[j] = elmntp->p[j] ;

/*
        q01 = np[0]->p[0] * (np[1]->x[0]-np[0]->x[0])
                + np[0]->p[1] *(np[1]->x[1]-np[0]->x[1]) ;

*/
        q01 = (np[0]->p[0]+np[1]->p[0])/2*(np[1]->x[0]-np[0]->x[0])
                        + (np[0]->p[1]+np[1]->p[1])/2*(np[1]->x[1]-np[0]->x[1]) ;

        if (q01 >= 0.0) {
                if(elmntp->nps[2] == NULL ){
                        theElp->vtype = theElp->gtype = 111 ;
                        theElp->nnds = 2 ;
                        theElp->nps[0] = elmntp->nps[0] ;
                        theElp->nps[1] = elmntp->nps[1] ;
                        return(0) ;
                }
                else {
                        theElp->vtype = 129 ;
                        theElp->nnds = 3 ;
                        theElp->nps[0] = elmntp->nps[0] ;
                        theElp->nps[1] = elmntp->nps[1] ;
                        theElp->nps[2] = elmntp->nps[2] ;
                        return(1) ;
                }
        }
        else {
                if(elmntp->nps[3] == NULL ){
                        theElp->vtype = theElp->gtype = 111 ;
                        theElp->nnds = 2 ;
                        theElp->nps[0] = elmntp->nps[0] ;
                        theElp->nps[1] = elmntp->nps[1] ;
                        return(0) ;
                }
                else  {
                        theElp->vtype = 128 ;
                        theElp->gtype = 111 ;
                        theElp->nnds = 3 ;
                        theElp->nps[0] = elmntp->nps[0] ;
                        theElp->nps[1] = elmntp->nps[1] ;
                        theElp->nps[2] = elmntp->nps[3] ;
                        return(1) ;
```

```
			}
		}
	}


	int		CU1DPick(elmntp,theElp,ntfp,dirp)
	struct element		*elmntp, *theElp ;
	int		*ntfp, *dirp ;

	{
		int		j;
		double		q01 ;
		struct node	*np[MNSF] ;

		theElp->gtype = 111 ;
		*ntfp = 2;
		for(j=0;j<elmntp->nnds;j++)
			np[j] = elmntp->nps[j] ;

		for(j=0;j<N.params;j++)
			theElp->p[j] = elmntp->p[j] ;

		q01 = (np[0]->p[0]+np[1]->p[0])/2*(np[1]->x[0]-np[0]->x[0])
				+ (np[0]->p[1]+np[1]->p[1])/2*(np[1]->x[1]-np[0]->x[1]) ;

		if (q01 >= 0.0) {
			if((elmntp->nps[2] == NULL) || (elmntp->nps[3] == NULL) ){
				theElp->vtype = theElp->gtype = 111 ;
				theElp->nnds = 2 ;
				theElp->nps[0] = elmntp->nps[0] ;
				theElp->nps[1] = elmntp->nps[1] ;
				return(0) ;
			}
			else {
				theElp->vtype = 139 ;
				theElp->gtype = 111 ;
				*ntfp = 2;
				theElp->nnds = 4 ;
				theElp->nps[0] = elmntp->nps[0] ;
				theElp->nps[1] = elmntp->nps[1] ;
				theElp->nps[2] = elmntp->nps[2] ;
				theElp->nps[3] = elmntp->nps[3] ;
				return(1) ;
			}
		}
		else {
			if((elmntp->nps[4] == NULL) || (elmntp->nps[5] == NULL) ){
				theElp->vtype = theElp->gtype = 111 ;
				theElp->nnds = 2 ;
				theElp->nps[0] = elmntp->nps[0] ;
				theElp->nps[1] = elmntp->nps[1] ;
				return(0) ;
			}
			else {
				theElp->vtype = 138 ;
				theElp->gtype = 111 ;
				*ntfp = 2;
				theElp->nnds = 4 ;
				theElp->nps[0] = elmntp->nps[0] ;
				theElp->nps[1] = elmntp->nps[1] ;
				theElp->nps[2] = elmntp->nps[4] ;
				theElp->nps[3] = elmntp->nps[5] ;
				return(1) ;
```

```
                )
        )
/*      return(-1) ;  */
}


int             QUelPick(elmntp,theElp,ntfp,dirp)
struct element          *elmntp, *theElp ;
int             *ntfp, *dirp ;

{
        int             j;
        double          q01, q12, q23, q30 ;
        struct node     *np[MNSF] ;

        theElp->gtype = 219 ;
        *ntfp = 4;
        for(j=0;j<elmntp->nnds;j++)
                np[j] = elmntp->nps[j] ;

        for(j=0;j<N.params;j++)
                theElp->p[j] = elmntp->p[j] ;

        q01 = (np[0]->p[0]+np[1]->p[0])/2*(np[0]->x[1]-np[1]->x[1])
                        - (np[0]->p[1]+np[1]->p[1])/2*(np[0]->x[0]-np[1]->x[0]) ;
        q12 = (np[1]->p[0]+np[2]->p[0])/2*(np[1]->x[1]-np[2]->x[1])
                        - (np[1]->p[1]+np[2]->p[1])/2*(np[1]->x[0]-np[2]->x[0]) ;
        q23 = (np[2]->p[0]+np[3]->p[0])/2*(np[2]->x[1]-np[3]->x[1])
                        - (np[2]->p[1]+np[3]->p[1])/2*(np[2]->x[0]-np[3]->x[0]) ;
        q30 = (np[3]->p[0]+np[0]->p[0])/2*(np[3]->x[1]-np[0]->x[1])
                        - (np[3]->p[1]+np[0]->p[1])/2*(np[3]->x[0]-np[0]->x[0]) ;

        if((q23>=q01) && (q30>=q12)) {
                if((elmntp->nps[14] == NULL) || (elmntp->nps[15] == NULL) ){
                        if((elmntp->nps[11] == NULL) || (elmntp->nps[12] == NULL)
){
                                theElp->vtype = theElp->gtype = 211 ;
                                theElp->nnds = 4 ;
                                theElp->nps[0] = elmntp->nps[0] ;
                                theElp->nps[1] = elmntp->nps[1] ;
                                theElp->nps[2] = elmntp->nps[2] ;
                                theElp->nps[3] = elmntp->nps[3] ;
                                return(0) ;
                        }
                        else {
                                theElp->vtype = 228 ;
                                theElp->nnds = 6 ;
                                theElp->nps[0] = elmntp->nps[0] ;
                                theElp->nps[1] = elmntp->nps[1] ;
                                theElp->nps[2] = elmntp->nps[2] ;
                                theElp->nps[3] = elmntp->nps[3] ;
                                theElp->nps[4] = elmntp->nps[11] ;
                                theElp->nps[5] = elmntp->nps[12] ;
                                return(1) ;
                        }
                }
                else {
                        if((elmntp->nps[11] == NULL) || (elmntp->nps[12] == NULL)
){
                                theElp->vtype = 228 ;
                                theElp->nnds = 6 ;
                                theElp->nps[0] = elmntp->nps[1] ;
                                theElp->nps[1] = elmntp->nps[2] ;
```

```
				theElp->nps[2] = elmntp->nps[3] ;
				theElp->nps[3] = elmntp->nps[0] ;
				theElp->nps[4] = elmntp->nps[14] ;
				theElp->nps[5] = elmntp->nps[15] ;
				return(1) ;
			}
			else if (elmntp->nps[13] == NULL) {
				theElp->vtype = 224 ;
				theElp->nnds = 8 ;
				theElp->nps[0] = elmntp->nps[0] ;
				theElp->nps[1] = elmntp->nps[1] ;
				theElp->nps[2] = elmntp->nps[2] ;
				theElp->nps[3] = elmntp->nps[3] ;
				theElp->nps[4] = elmntp->nps[11] ;
				theElp->nps[5] = elmntp->nps[12] ;
				theElp->nps[7] = elmntp->nps[14] ;
				theElp->nps[8] = elmntp->nps[15] ;
				return(2) ;
			}
			else {
				theElp->vtype = 229 ;
				theElp->nnds = 9 ;
				theElp->nps[0] = elmntp->nps[0] ;
				theElp->nps[1] = elmntp->nps[1] ;
				theElp->nps[2] = elmntp->nps[2] ;
				theElp->nps[3] = elmntp->nps[3] ;
				theElp->nps[4] = elmntp->nps[11] ;
				theElp->nps[5] = elmntp->nps[12] ;
				theElp->nps[6] = elmntp->nps[13] ;
				theElp->nps[7] = elmntp->nps[14] ;
				theElp->nps[8] = elmntp->nps[15] ;
				return(3) ;
			}
		}
	}
	if((q23>=q01) && (q30<q12)) {
		if((elmntp->nps[11] == NULL) || (elmntp->nps[12] == NULL) ){
			if((elmntp->nps[8] == NULL) || (elmntp->nps[9] == NULL) ){
				theElp->vtype = 211 ;
				theElp->nnds = 4 ;
				theElp->nps[0] = elmntp->nps[0] ;
				theElp->nps[1] = elmntp->nps[1] ;
				theElp->nps[2] = elmntp->nps[2] ;
				theElp->nps[3] = elmntp->nps[3] ;
				return(0) ;
			}
			else {
				theElp->vtype = 228 ;
				theElp->nnds = 6 ;
				theElp->nps[0] = elmntp->nps[3] ;
				theElp->nps[1] = elmntp->nps[0] ;
				theElp->nps[2] = elmntp->nps[1] ;
				theElp->nps[3] = elmntp->nps[2] ;
				theElp->nps[4] = elmntp->nps[8] ;
				theElp->nps[5] = elmntp->nps[9] ;
				return(1) ;
			}
		}
		else {
			if((elmntp->nps[8] == NULL) || (elmntp->nps[9] == NULL) ){
				theElp->vtype = 228 ;
				theElp->nnds = 6 ;
				theElp->nps[0] = elmntp->nps[0] ;
```

```
                                theElp->nps[1] = elmntp->nps[1] ;
                                theElp->nps[2] = elmntp->nps[2] ;
                                theElp->nps[3] = elmntp->nps[3] ;
                                theElp->nps[4] = elmntp->nps[11] ;
                                theElp->nps[5] = elmntp->nps[12] ;
                                return(1) ;
                        }
                        else if (elmntp->nps[10] == NULL) {
                                theElp->vtype = 224 ;
                                theElp->nnds = 8 ;
                                theElp->nps[0] = elmntp->nps[3] ;
                                theElp->nps[1] = elmntp->nps[0] ;
                                theElp->nps[2] = elmntp->nps[1] ;
                                theElp->nps[3] = elmntp->nps[2] ;
                                theElp->nps[4] = elmntp->nps[8] ;
                                theElp->nps[5] = elmntp->nps[9] ;
                                theElp->nps[7] = elmntp->nps[11] ;
                                theElp->nps[8] = elmntp->nps[12] ;
                                return(2) ;
                        }
                        else {
                                theElp->vtype = 229 ;
                                theElp->nnds = 9 ;
                                theElp->nps[0] = elmntp->nps[3] ;
                                theElp->nps[1] = elmntp->nps[0] ;
                                theElp->nps[2] = elmntp->nps[1] ;
                                theElp->nps[3] = elmntp->nps[2] ;
                                theElp->nps[4] = elmntp->nps[8] ;
                                theElp->nps[5] = elmntp->nps[9] ;
                                theElp->nps[6] = elmntp->nps[10] ;
                                theElp->nps[7] = elmntp->nps[11] ;
                                theElp->nps[8] = elmntp->nps[12] ;
                                return(3) ;
                        }
                }
        }
        if((q23<q01) && (q30<q12)) {
                if((elmntp->nps[8] == NULL) || (elmntp->nps[9] == NULL) ){
                        if((elmntp->nps[5] == NULL) || (elmntp->nps[6] == NULL) ){
                                theElp->vtype = 211 ;
                                theElp->nnds = 4 ;
                                theElp->nps[0] = elmntp->nps[0] ;
                                theElp->nps[1] = elmntp->nps[1] ;
                                theElp->nps[2] = elmntp->nps[2] ;
                                theElp->nps[3] = elmntp->nps[3] ;
                                return(0) ;
                        }
                        else {
                                theElp->vtype = 228 ;
                                theElp->nnds = 6 ;
                                theElp->nps[0] = elmntp->nps[2] ;
                                theElp->nps[1] = elmntp->nps[3] ;
                                theElp->nps[2] = elmntp->nps[0] ;
                                theElp->nps[3] = elmntp->nps[1] ;
                                theElp->nps[4] = elmntp->nps[5] ;
                                theElp->nps[5] = elmntp->nps[6] ;
                                return(1) ;
                        }
                }
                else {
                        if((elmntp->nps[5] == NULL) || (elmntp->nps[6] == NULL) ){
                                theElp->vtype = 228 ;
                                theElp->nnds = 6 ;
```

```
                theElp->nps[0] = elmntp->nps[3] ;
                theElp->nps[1] = elmntp->nps[0] ;
                theElp->nps[2] = elmntp->nps[1] ;
                theElp->nps[3] = elmntp->nps[2] ;
                theElp->nps[4] = elmntp->nps[8] ;
                theElp->nps[5] = elmntp->nps[9] ;
                return(1) ;
            }
            else if (elmntp->nps[7] == NULL) {
                theElp->vtype = 224 ;
                theElp->nnds = 8 ;
                theElp->nps[0] = elmntp->nps[2] ;
                theElp->nps[1] = elmntp->nps[3] ;
                theElp->nps[2] = elmntp->nps[0] ;
                theElp->nps[3] = elmntp->nps[1] ;
                theElp->nps[4] = elmntp->nps[5] ;
                theElp->nps[5] = elmntp->nps[6] ;
                theElp->nps[6] = elmntp->nps[8] ;
                theElp->nps[7] = elmntp->nps[9] ;
                return(2) ;
            }
            else {
                theElp->vtype = 229 ;
                theElp->nnds = 9 ;
                theElp->nps[0] = elmntp->nps[2] ;
                theElp->nps[1] = elmntp->nps[3] ;
                theElp->nps[2] = elmntp->nps[0] ;
                theElp->nps[3] = elmntp->nps[1] ;
                theElp->nps[4] = elmntp->nps[5] ;
                theElp->nps[5] = elmntp->nps[6] ;
                theElp->nps[6] = elmntp->nps[7] ;
                theElp->nps[7] = elmntp->nps[8] ;
                theElp->nps[8] = elmntp->nps[9] ;
                return(3) ;
            }
        }
    }
    if((q23<q01) && (q30>=q12)) {
        if((elmntp->nps[5] == NULL) || (elmntp->nps[6] == NULL) ){
            if((elmntp->nps[14] == NULL) || (elmntp->nps[15] == NULL)
){
                theElp->vtype = 211 ;
                theElp->nnds = 4 ;
                theElp->nps[0] = elmntp->nps[0] ;
                theElp->nps[1] = elmntp->nps[1] ;
                theElp->nps[2] = elmntp->nps[2] ;
                theElp->nps[3] = elmntp->nps[3] ;
                return(0) ;
            }
            else {
                theElp->vtype = 228 ;
                theElp->nnds = 6 ;
                theElp->nps[0] = elmntp->nps[1] ;
                theElp->nps[1] = elmntp->nps[2] ;
                theElp->nps[2] = elmntp->nps[3] ;
                theElp->nps[3] = elmntp->nps[0] ;
                theElp->nps[4] = elmntp->nps[14] ;
                theElp->nps[5] = elmntp->nps[15] ;
                return(1) ;
            }
        }
        else {
```

```
                        if((elmntp->nps[14] == NULL) || (elmntp->nps[15] == NULL)
){
                                theElp->vtype = 228 ;
                                theElp->nnds = 6 ;
                                theElp->nps[0] = elmntp->nps[2] ;
                                theElp->nps[1] = elmntp->nps[3] ;
                                theElp->nps[2] = elmntp->nps[0] ;
                                theElp->nps[3] = elmntp->nps[1] ;
                                theElp->nps[4] = elmntp->nps[5] ;
                                theElp->nps[5] = elmntp->nps[6] ;
                                return(1) ;
                        }
                        else if (elmntp->nps[4] == NULL) {
                                theElp->vtype = 224 ;
                                theElp->nnds = 8 ;
                                theElp->nps[0] = elmntp->nps[1] ;
                                theElp->nps[1] = elmntp->nps[2] ;
                                theElp->nps[2] = elmntp->nps[3] ;
                                theElp->nps[3] = elmntp->nps[0] ;
                                theElp->nps[4] = elmntp->nps[14] ;
                                theElp->nps[5] = elmntp->nps[15] ;
                                theElp->nps[6] = elmntp->nps[5] ;
                                theElp->nps[7] = elmntp->nps[6] ;
                                return(2) ;
                        }
                        else {
                                theElp->vtype = 229 ;
                                theElp->nnds = 9 ;
                                theElp->nps[0] = elmntp->nps[1] ;
                                theElp->nps[1] = elmntp->nps[2] ;
                                theElp->nps[2] = elmntp->nps[3] ;
                                theElp->nps[3] = elmntp->nps[0] ;
                                theElp->nps[4] = elmntp->nps[14] ;
                                theElp->nps[5] = elmntp->nps[15] ;
                                theElp->nps[6] = elmntp->nps[4] ;
                                theElp->nps[7] = elmntp->nps[5] ;
                                theElp->nps[8] = elmntp->nps[6] ;
                                return(3) ;
                        }
                }
        }
        return(-1) ;
}

int             CU_Stream_Pick(elmntp,theElp,ntfp,dirp)
struct          element         *elmntp, *theElp ;
int             *ntfp, *dirp ;
{
                theElp->vtype = 233 ;
                        theElp->nnds = 8 ;
                        theElp->nps[0] = elmntp->nps[0] ;
                        theElp->nps[1] = elmntp->nps[1] ;
                        theElp->nps[2] = elmntp->nps[2] ;
                        theElp->nps[3] = elmntp->nps[3] ;
                        theElp->nps[4] = elmntp->nps[4] ;
                        theElp->nps[5] = elmntp->nps[5] ;
                        theElp->nps[6] = elmntp->nps[6] ;
                        theElp->nps[7] = elmntp->nps[7] ;

                if (elmntp->nps[5] == NULL || elmntp->nps[6] == NULL){
                        theElp->vtype = 230;
                        theElp->nnds = 6 ;
                        theElp->nps[0] = elmntp->nps[0] ;
```

```
                        theElp->nps[1] = elmntp->nps[1] ;
                        theElp->nps[2] = elmntp->nps[2] ;
                        theElp->nps[3] = elmntp->nps[3] ;
                        theElp->nps[4] = elmntp->nps[4] ;
                        theElp->nps[5] = elmntp->nps[7] ;
                }

                if(elmntp->nps[4] == NULL || elmntp->nps[7] == NULL){
                        theElp->vtype = 211;
                        theElp->nnds = 4 ;
                        theElp->nps[0] = elmntp->nps[0] ;
                        theElp->nps[1] = elmntp->nps[1] ;
                        theElp->nps[2] = elmntp->nps[2] ;
                        theElp->nps[3] = elmntp->nps[3] ;
                }

                return(-1) ;
}

int             CU2DPick(elmntp,theElp,ntfp,dirp)
struct element          *elmntp, *theElp ;
int             *ntfp, *dirp ;

{
        int     j;
        double  q01, q12, q23, q30 ;
        struct node     *np[MNSF] ;
        int node_hands[MNSF] ;

        theElp->gtype = 211 ;
        *ntfp = 4;
        for(j=0;j<elmntp->nnds;j++)
                np[j] = elmntp->nps[j] ;

        for(j=0;j<N.params;j++)
                theElp->p[j] = elmntp->p[j] ;

        q01 = (np[0]->p[0]+np[1]->p[0])/2*(np[0]->x[1]-np[1]->x[1])
                        - (np[0]->p[1]+np[1]->p[1])/2*(np[0]->x[0]-np[1]->x[0]) ;
        q12 = (np[1]->p[0]+np[2]->p[0])/2*(np[1]->x[1]-np[2]->x[1])
                        - (np[1]->p[1]+np[2]->p[1])/2*(np[1]->x[0]-np[2]->x[0]) ;
        q23 = (np[2]->p[0]+np[3]->p[0])/2*(np[2]->x[1]-np[3]->x[1])
                        - (np[2]->p[1]+np[3]->p[1])/2*(np[2]->x[0]-np[3]->x[0]) ;
        q30 = (np[3]->p[0]+np[0]->p[0])/2*(np[3]->x[1]-np[0]->x[1])
                        - (np[3]->p[1]+np[0]->p[1])/2*(np[3]->x[0]-np[0]->x[0]) ;

        if((q23>=q01) && (q30>=q12)) {
                node_hands[0] = 0 ;
                node_hands[1] = 1 ;
                node_hands[2] = 2 ;
                node_hands[3] = 3 ;
                node_hands[4] = 11 ;
                node_hands[5] = 28 ;
                node_hands[6] = 29 ;
                node_hands[7] = 12 ;
                node_hands[8] = 30 ;
                node_hands[9] = 13 ;
                node_hands[10] = 14 ;
                node_hands[11] = 15 ;
                node_hands[12] = 31 ;
                node_hands[13] = 32 ;
                node_hands[14] = 33 ;
                node_hands[15] = 34 ;
```

```
	}
	if((q23>=q01) && (q30<q12)) {
		node_hands[0] = 3 ;
		node_hands[1] = 0 ;
		node_hands[2] = 1 ;
		node_hands[3] = 2 ;
		node_hands[4] = 8 ;
		node_hands[5] = 23 ;
		node_hands[6] = 24 ;
		node_hands[7] = 9 ;
		node_hands[8] = 25 ;
		node_hands[9] = 10 ;
		node_hands[10] = 11 ;
		node_hands[11] = 12 ;
		node_hands[12] = 26 ;
		node_hands[13] = 27 ;
		node_hands[14] = 28 ;
		node_hands[15] = 29 ;
	}
	if((q23<q01) && (q30<q12)) {
		node_hands[0] = 2 ;
		node_hands[1] = 3 ;
		node_hands[2] = 0 ;
		node_hands[3] = 1 ;
		node_hands[4] = 5 ;
		node_hands[5] = 18 ;
		node_hands[6] = 19 ;
		node_hands[7] = 6 ;
		node_hands[8] = 20 ;
		node_hands[9] = 7 ;
		node_hands[10] = 8 ;
		node_hands[11] = 9 ;
		node_hands[12] = 21 ;
		node_hands[13] = 22 ;
		node_hands[14] = 23 ;
		node_hands[15] = 24 ;
	}
	if((q23<q01) && (q30>=q12)) {
		node_hands[0] = 1 ;
		node_hands[1] = 2 ;
		node_hands[2] = 3 ;
		node_hands[3] = 0 ;
		node_hands[4] = 14 ;
		node_hands[5] = 33 ;
		node_hands[6] = 34 ;
		node_hands[7] = 15 ;
		node_hands[8] = 35 ;
		node_hands[9] = 4 ;
		node_hands[10] = 5 ;
		node_hands[11] = 6 ;
		node_hands[12] = 16 ;
		node_hands[13] = 17 ;
		node_hands[14] = 18 ;
		node_hands[15] = 19 ;
	}
	reall_nodes_CU2D( elmntp, theElp, node_hands) ;

	return(-1) ;
}


int		reall_nodes_CU2D(elmntp, theElp, n_h)
struct element		*elmntp, *theElp ;
```

```
int             n_h[MNSF] ;
{

        if((elmntp->nps[n_h[12]] == NULL)){
                theElp->vtype = 238 ;
                theElp->nnds = 15 ;
                theElp->nps[0] = elmntp->nps[n_h[0]] ;
                theElp->nps[1] = elmntp->nps[n_h[1]] ;
                theElp->nps[2] = elmntp->nps[n_h[2]] ;
                theElp->nps[3] = elmntp->nps[n_h[3]] ;
                theElp->nps[4] = elmntp->nps[n_h[4]] ;
                theElp->nps[5] = elmntp->nps[n_h[5]] ;
                theElp->nps[6] = elmntp->nps[n_h[6]] ;
                theElp->nps[7] = elmntp->nps[n_h[7]] ;
                theElp->nps[8] = elmntp->nps[n_h[8]] ;
                theElp->nps[9] = elmntp->nps[n_h[9]] ;
                theElp->nps[10] = elmntp->nps[n_h[10]] ;
                theElp->nps[11] = elmntp->nps[n_h[13]] ;
                theElp->nps[12] = elmntp->nps[n_h[14]] ;
                theElp->nps[13] = elmntp->nps[n_h[15]] ;
                theElp->nps[14] = elmntp->nps[n_h[11]] ;

        }
        else {
                theElp->vtype = 239 ;
                theElp->nnds = 16 ;
                theElp->nps[0] = elmntp->nps[n_h[0]] ;
                theElp->nps[1] = elmntp->nps[n_h[1]] ;
                theElp->nps[2] = elmntp->nps[n_h[2]] ;
                theElp->nps[3] = elmntp->nps[n_h[3]] ;
                theElp->nps[4] = elmntp->nps[n_h[4]] ;
                theElp->nps[5] = elmntp->nps[n_h[5]] ;
                theElp->nps[6] = elmntp->nps[n_h[6]] ;
                theElp->nps[7] = elmntp->nps[n_h[7]] ;
                theElp->nps[8] = elmntp->nps[n_h[8]] ;
                theElp->nps[9] = elmntp->nps[n_h[9]] ;
                theElp->nps[10] = elmntp->nps[n_h[10]] ;
                theElp->nps[11] = elmntp->nps[n_h[12]] ;
                theElp->nps[12] = elmntp->nps[n_h[13]] ;
                theElp->nps[13] = elmntp->nps[n_h[14]] ;
                theElp->nps[14] = elmntp->nps[n_h[15]] ;
                theElp->nps[15] = elmntp->nps[n_h[11]] ;

        }
        if ((elmntp->nps[n_h[13]] == NULL)){
                theElp->vtype = 2371 ;
                theElp->nnds = 14 ;
                theElp->nps[0] = elmntp->nps[n_h[0]] ;
                theElp->nps[1] = elmntp->nps[n_h[1]] ;
                theElp->nps[2] = elmntp->nps[n_h[2]] ;
                theElp->nps[3] = elmntp->nps[n_h[3]] ;
                theElp->nps[4] = elmntp->nps[n_h[4]] ;
                theElp->nps[5]   elmntp->nps[n_h[5]] ;
                theElp->nps[6] = elmntp->nps[n_h[6]] ;
                theElp->nps[7] = elmntp->nps[n_h[7]] ;
                theElp->nps[8] = elmntp->nps[n_h[8]] ;
                theElp->nps[9] = elmntp->nps[n_h[9]] ;
                theElp->nps[10] = elmntp->nps[n_h[10]] ;
                theElp->nps[11] = elmntp->nps[n_h[14]] ;
                theElp->nps[12] = elmntp->nps[n_h[15]] ;
                theElp->nps[13] = elmntp->nps[n_h[11]] ;

        }
```

```
                if ((elmntp->nps[n_h[8]] == NULL)){
                        theElp->vtype = 2372 ;
                        theElp->nnds = 14 ;
                        theElp->nps[0] = elmntp->nps[n_h[0]] ;
                        theElp->nps[1] = elmntp->nps[n_h[1]] ;
                        theElp->nps[2]      mntp->nps[n_h[2]] ;
                        theElp->nps[3]      mntp->nps[n_h[3]] ;
                        theElp->nps[4]      mntp->nps[n_h[11]] ;
                        theElp->nps[5]     lmntp->nps[n_h[15]] ;
                        theElp->nps[6] = elmntp->nps[n_h[14]] ;
                        theElp->nps[7] = elmntp->nps[n_h[10]] ;
                        theElp->nps[8] = elmntp->nps[n_h[13]] ;
                        theElp->nps[9] = elmntp->nps[n_h[9]] ;
                        theElp->nps[10] = elmntp->nps[n_h[7]] ;
                        theElp->nps[11] = elmntp->nps[n_h[6]] ;
                        theElp->nps[12] = elmntp->nps[n_h[5]] ;
                        theElp->nps[13] = elmntp->nps[n_h[4]] ;

                }
                if ((elmntp->nps[n_h[14]] == NULL) || (elmntp->nps[n_h[15]] ==
NULL)){
                        theElp->vtype = 2361;
                        theElp->nnds = 12;
                        theElp->nps[0] = elmntp->nps[n_h[0]] ;
                        theElp->nps[1] = elmntp->nps[n_h[1]] ;
                        theElp->nps[2] = elmntp->nps[n_h[2]] ;
                        theElp->nps[3] = elmntp->nps[n_h[3]] ;
                        theElp->nps[4] = elmntp->nps[n_h[4]] ;
                        theElp->nps[5] = elmntp->nps[n_h[5]] ;
                        theElp->nps[6] = elmntp->nps[n_h[6]] ;
                        theElp->nps[7] = elmntp->nps[n_h[7]] ;
                        theElp->nps[8] = elmntp->nps[n_h[8]] ;
                        theElp->nps[9] = elmntp->nps[n_h[9]] ;
                        theElp->nps[10] = elmntp->nps[n_h[10]] ;
                        theElp->nps[11] = elmntp->nps[n_h[11]] ;

                }
                if ((elmntp->nps[n_h[5]] == NULL) || (elmntp->nps[n_h[6]] ==
NULL)){
                        theElp->vtype = 2363;
                        theElp->nnds = 12;
                        theElp->nps[0] = elmntp->nps[n_h[0]] ;
                        theElp->nps[1] = elmntp->nps[n_h[1]] ;
                        theElp->nps[2] = elmntp->nps[n_h[2]] ;
                        theElp->nps[3] = elmntp->nps[n_h[3]] ;
                        theElp->nps[4] = elmntp->nps[n_h[11]] ;
                        theElp->nps[5] = elmntp->nps[n_h[15]] ;
                        theElp->nps[6] = elmntp->nps[n_h[14]] ;
                        theElp->nps[7] = elmntp->nps[n_h[10]] ;
                        theElp->nps[8] = elmntp->nps[n_h[13]] ;
                        theElp->nps[9] = elmntp->nps[n_h[9]] ;
                        theElp->nps[10] = elmntp->nps[n_h[7]] ;
                        theElp->nps[11] = elmntp->nps[n_h[4   ;

                }
                if ((elmntp->nps[n_h[8]] == NULL) && ((elmntp->nps[n_h[14]] ==
NULL)

        || (elmntp->nps[n_h[15]] == NULL))){
                        theElp->vtype = 2351;
                        theElp->nnds = 11 ;
                        theElp->nps[0] = elmntp->nps[n_h[0]] ;
                        theElp->nps[1] = elmntp->nps[n_h[1]] ;
```

```
                    theElp->nps[2] = elmntp->nps[n_h[2]] ;
                    theElp->nps[3] = elmntp->nps[n_h[3]] ;
                    theElp->nps[4] = elmntp->nps[n_h[4]] ;
                    theElp->nps[5] = elmntp->nps[n_h[5]] ;
                    theElp->nps[6] = elmntp->nps[n_h[6]] ;
                    theElp->nps[7] = elmntp->nps[n_h[7]] ;
                    theElp->nps[8] = elmntp->nps[n_h[9]] ;
                    theElp->nps[9] = elmntp->nps[n_h[10]] ;
                    theElp->nps[10] = elmntp->nps[n_h[11]] ;

            }
            if ((elmntp->nps[n_h[13]] == NULL) && ((elmntp->nps[n_h[5]] ==
NULL)
                        || (elmntp->nps[n_h[6]] == NULL))) {
                    theElp->vtype = 2353;
                    theElp->nnds = 11 ;
                    theElp->nps[0] = elmntp->nps[n_h[0]] ;
                    theElp->nps[1] = elmntp->nps[n_h[1]] ;
                    theElp->nps[2] = elmntp->nps[n_h[2]] ;
                    theElp->nps[3] = elmntp->nps[n_h[3]] ;
                    theElp->nps[4] = elmntp->nps[n_h[11]] ;
                    theElp->nps[5] = elmntp->nps[n_h[15]] ;
                    theElp->nps[6] = elmntp->nps[n_h[14]] ;
                    theElp->nps[7] = elmntp->nps[n_h[10]] ;
                    theElp->nps[8] = elmntp->nps[n_h[9]] ;
                    theElp->nps[9] = elmntp->nps[n_h[7]] ;
                    theElp->nps[10] = elmntp->nps[n_h[4]] ;
            }
            if((elmntp->nps[n_h[8]] == NULL) && (elmntp->nps[n_h[13]] ==
NULL)){
                    theElp->vtype = 2362 ;
                    theElp->nnds = 13 ;
                    theElp->nps[0] = elmntp->nps[n_h[0]] ;
                    theElp->nps[1] = elmntp->nps[n_h[1]] ;
                    theElp->nps[2] = elmntp->nps[n_h[2]] ;
                    theElp->nps[3] = elmntp->nps[n_h[3]] ;
                    theElp->nps[4] = elmntp->nps[n_h[4]] ;
                    theElp->nps[5] = elmntp->nps[n_h[5]] ;
                    theElp->nps[6] = elmntp->nps[n_h[6]] ;
                    theElp->nps[7] = elmntp->nps[n_h[7]] ;
                    theElp->nps[8] = elmntp->nps[n_h[9]] ;
                    theElp->nps[9] = elmntp->nps[n_h[10]] ;
                    theElp->nps[10] = elmntp->nps[n_h[14]] ;
                    theElp->nps[11] = elmntp->nps[n_h[15]] ;
                    theElp->nps[12] = elmntp->nps[n_h[11]] ;

            }
            if((elmntp->nps[n_h[9]] == NULL)){
                    theElp->vtype = 2352 ;
                    theElp->nnds = 12 ;
                    theElp->nps[0] = elmntp->nps[n_h[0]] ;
                    theElp->nps[1] = elmntp->nps[n_h[1]] ;
                    theElp->nps[2] = elmntp->nps[n_h[2]] ;
                    theElp->nps[3] = elmntp->nps[n_h[3]] ;
                    theElp->nps[4] = elmntp->nps[n_h[4]] ;
                    theElp->nps[5] = elmntp->nps[n_h[5]] ;
                    theElp->nps[6] = elmntp->nps[n_h[6]] ;
                    theElp->nps[7] = elmntp->nps[n_h[7]] ;
                    theElp->nps[8] = elmntp->nps[n_h[10]] ;
                    theElp->nps[9] = elmntp->nps[n_h[14]] ;
                    theElp->nps[10] = elmntp->nps[n_h[15]] ;
                    theElp->nps[11] = elmntp->nps[n_h[11]] ;
            }
```

```
		else if(((elmntp->nps[n_h[6]] == NULL)    || (elmntp->nps[n_h[5]] == NULL))
			&& ((elmntp->nps[n_h[14]] == NULL) || (elmntp->nps[n_h[15]] == NULL))){
			theElp->vtype = 232 ;
			theElp->nnds = 9 ;
			theElp->nps[0] = elmntp->nps[n_h[0]] ;
			theElp->nps[1] = elmntp->nps[n_h[1]] ;
			theElp->nps[2] = elmntp->nps[n_h[2]] ;
			theElp->nps[3] = elmntp->nps[n_h[3]] ;
			theElp->nps[4] = elmntp->nps[n_h[4]] ;
			theElp->nps[5] = elmntp->nps[n_h[7]] ;
			theElp->nps[6] = elmntp->nps[n_h[9]] ;
			theElp->nps[7] = elmntp->nps[n_h[10]] ;
			theElp->nps[8] = elmntp->nps[n_h[11]] ;

		}
		if(((elmntp->nps[n_h[14]] == NULL) || (elmntp->nps[n_h[15]] == NULL))
		 && (elmntp->nps[n_h[9]] == NULL)){
			theElp->vtype = 2341 ;
			theElp->nnds = 10 ;
			theElp->nps[0] = elmntp->nps[n_h[0]] ;
			theElp->nps[1] = elmntp->nps[n_h[1]] ;
			theElp->nps[2] = elmntp->nps[n_h[2]] ;
			theElp->nps[3] = elmntp->nps[n_h[3]] ;
			theElp->nps[4] = elmntp->nps[n_h[4]] ;
			theElp->nps[5] = elmntp->nps[n_h[5]] ;
			theElp->nps[6] = elmntp->nps[n_h[6]] ;
			theElp->nps[7] = elmntp->nps[n_h[7]] ;
			theElp->nps[8] = elmntp->nps[n_h[10]] ;
			theElp->nps[9] = elmntp->nps[n_h[11]] ;

			if((elmntp->nps[n_h[6]] == NULL)   || (elmntp->nps[n_h[5]] == NULL)){
				theElp->vtype = 2311 ;
				theElp->nnds = 8 ;
				theElp->nps[0] = elmntp->nps[n_h[0]] ;
				theElp->nps[1] = elmntp->nps[n_h[1]] ;
				theElp->nps[2] = elmntp->nps[n_h[2]] ;
				theElp->nps[3] = elmntp->nps[n_h[3]] ;
				theElp->nps[4] = elmntp->nps[n_h[4]] ;
				theElp->nps[5] = elmntp->nps[n_h[7]] ;
				theElp->nps[6] = elmntp->nps[n_h[10]] ;
				theElp->nps[7] = elmntp->nps[n_h[11]] ;
			}
		}
		if(((elmntp->nps[n_h[6]] == NULL)   || (elmntp->nps[n_h[5]] == NULL))
		 && (elmntp->nps[n_h[9]] == NULL)){
			theElp->vtype = 2342 ;
			theElp->nnds = 10 ;
			theElp->nps[0] = elmntp->nps[n_h[0]] ;
			theElp->nps[1] = elmntp->nps[n_h[1]] ;
			theElp->nps[2] = elmntp->nps[n_h[2]] ;
			theElp->nps[3] = elmntp->nps[n_h[3]] ;
			theElp->nps[4] = elmntp->nps[n_h[11]] ;
			theElp->nps[5] = elmntp->nps[n_h[15]] ;
			theElp->nps[6] = elmntp->nps[n_h[14]] ;
			theElp->nps[7] = elmntp->nps[n_h[10]] ;
			theElp->nps[8] = elmntp->nps[n_h[7]] ;
			theElp->nps[9] = elmntp->nps[n_h[4]] ;
```

```
                }
                if((elmntp->nps[n_h[10]] == NULL) || (elmntp->nps[n_h[11]] ==
NULL)){
                        theElp->vtype = 233 ;
                        theElp->nnds = 8 ;
                        theElp->nps[0] = elmntp->nps[n_h[0]] ;
                        theElp->nps[1] = elmntp->nps[n_h[1]] ;
                        theElp->nps[2] = elmntp->nps[n_h[2]] ;
                        theElp->nps[3] = elmntp->nps[n_h[3]] ;
                        theElp->nps[4] = elmntp->nps[n_h[4]] ;
                        theElp->nps[5] = elmntp->nps[n_h[5]] ;
                        theElp->nps[6] = elmntp->nps[n_h[6]] ;
                        theElp->nps[7] = elmntp->nps[n_h[7]] ;

                        if ((elmntp->nps[n_h[6]] == NULL)  || (elmntp->nps[n_h[5]]
== NULL)){
                                theElp->vtype = 230;
                                theElp->nnds = 6 ;
                                theElp->nps[0] = elmntp->nps[n_h[0]] ;
                                theElp->nps[1] = elmntp->nps[n_h[1]] ;
                                theElp->nps[2] = elmntp->nps[n_h[2]] ;
                                theElp->nps[3] = elmntp->nps[n_h[3]] ;
                                theElp->nps[4] = elmntp->nps[n_h[4]] ;
                                theElp->nps[5] = elmntp->nps[n_h[7]] ;
                        }
                }

                if((elmntp->nps[n_h[4]] == NULL) || (elmntp->nps[n_h[7]] ==
NULL)){
                        theElp->vtype = 233 ;
                        theElp->nnds = 8 ;
                        theElp->nps[0] = elmntp->nps[n_h[1]] ;
                        theElp->nps[1] = elmntp->nps[n_h[2]] ;
                        theElp->nps[2] = elmntp->nps[n_h[3]] ;
                        theElp->nps[3] = elmntp->nps[n_h[0]] ;
                        theElp->nps[4] = elmntp->nps[n_h[10]] ;
                        theElp->nps[5] = elmntp->nps[n_h[14]] ;
                        theElp->nps[6] = elmntp->nps[n_h[15]] ;
                        theElp->nps[7] = elmntp->nps[n_h[11]] ;

                        if ((elmntp->nps[n_h[14]] == NULL) || (elmntp-
>nps[n_h[15]] == NULL)){
                                theElp->vtype = 230;
                                theElp->nnds = 6 ;
                                theElp->nps[0] = elmntp->nps[n_h[1]] ;
                                theElp->nps[1] = elmntp->nps[n_h[2]] ;
                                theElp->nps[2] = elmntp->nps[n_h[3]] ;
                                theElp->nps[3] = elmntp->nps[n_h[0]] ;
                                theElp->nps[4] = elmntp->nps[n_h[10]] ;
                                theElp->nps[5] = elmntp->nps[n_h[11]] ;
                        }
                }
                if(((elmntp->nps[n_h[10]] == NULL) || (elmntp->nps[n_h[11]] ==
NULL))
                        && ((elmntp->nps[n_h[7]] == NULL)  || (elmntp->nps[n_h[4]]
== NULL))){

                        theElp->vtype = 211;
                        theElp->nnds = 4 ;
                        theElp->nps[0] = elmntp->nps[n_h[0]] ;
                        theElp->nps[1] = elmntp->nps[n_h[1]] ;
                        theElp->nps[2] = elmntp->nps[n_h[2]] ;
                        theElp->nps[3] = elmntp->nps[n_h[3]] ;
```

```
                }

        return(-1) ;
}
```

fegauss.c

```
#include "fe.h"


int                   get_gspts(i_type,g)
int                   i_type ;
struct gausspts              g[] ;

{
        int n ;

        switch (i_type) {

                case 1 :
                        n = gs_pt(g) ;
                        break ;

                case 111 :
                        n = gs_1D2(g) ;
                        break ;

                case 119 :
                        n = qu_1D2(g) ;
                        break ;

                case 118 :
                        n = qu_1D2r(g) ;
                        break ;

                case 128 :
                        n = gs_1D2(g) ;
                        break ;

                case 129 :
                        n = gs_1D2(g) ;
                        break ;

                case 121 :
                        n = gs_1D3(g) ;
                        break ;

                case 131 :
                        n = gs_1D3(g) ;
                        break ;

                case 138 :
                        n = gs_1D3(g) ;
                        break ;

                case 139 :
                        n = gs_1D3(g) ;
                        break ;

                case 149 :
                        n = gs_1D3(g) ;
                        break ;

                case 159 :
                        n = gs_1D3(g) ;
                        break ;

                case 169 :
                        n = gs_1D5(g) ;
                        break ;
```

```
		case 179 :
			n = gs_1D5(g) ;
			break ;

		case 175 :
			n = gs_1D5(g) ;
			break ;

		case 210 :
			n = gs_2t2(g) ;
			break ;

		case 220 :
			n = gs_2t2(g) ;
			break ;

		case 211 :
			n = gs_2D2(g) ;
			break ;

		case 216 :
			n = gs_2D2(g) ;
			break ;

		case 221 :
			n = gs_2D2(g) ;
			break ;

		case 226 :
			n = gs_2D2(g) ;
			break ;

		case 222 :
			n = gs_2D2(g) ;
			break ;

		case 227 :
			n = gs_2D2(g) ;
			break ;

		case 219 :
			n = qu_2D2(g) ;
			break ;

		case 228 :
			n = qu_2D3(g) ;
			break ;

		case 229 :
			n = qu_2D3(g) ;
			break ;

		case 231 :
			n = gs_2D3(g) ;
			break ;

		case 230 :
			n = gs_2D2(g) ;
			break ;

		case 2311 :
			n = gs_2D2(g) ;
```

```
            break ;

        case 232 :
            n = gs_2D2(g) ;
            break ;

        case 233 :
            n = gs_2D3(g) ;
            break ;

        case 2341 :
            n = gs_2D3(g) ;
            break ;

        case 2342 :
            n = gs_2D3(g) ;
            break ;

        case 2351 :
            n = gs_2D3(g) ;
            break ;

        case 2353 :
            n = gs_2D3(g) ;
            break ;

        case 2352 :
            n = gs_2D3(g) ;
            break ;

        case 2361 :
            n = gs_2D3(g) ;
            break ;

        case 2363 :
            n = gs_2D3(g) ;
            break ;

        case 2362 :
            n = gs_2D3(g) ;
            break ;

        case 2371 :
            n = gs_2D3(g) ;
            break ;

        case 2372 :
            n = gs_2D3(g) ;
            break ;

        case 238 :
            n = gs_2D3(g) ;
            break ;

        case 239 :
            n = gs_2D3(g) ;
            break ;

        case 235 :
            n = gs_2D2(g) ;
            break ;

        case 911 :
```

```
                        n = gs_1D5(g) ;
                        break ;

                case 918 :
                        n = gs_qu1D5r(g) ;
                        break ;

                case 919 :
                        n = gs_qu1D5(g) ;
                        break ;

                case 921 :
                        n = gs_1D5(g) ;
                        break ;

                case 929 :
                        n = gs_qu1D5(g) ;
                        break ;

                case 922 :
                        n = gs_2D4(g) ;
                        break ;

                default :
                        n = 0 ;
                        break ;
        }
        return(n) ;
}

int                     gs_pt(g)
struct gausspts                 g[] ;

{
        g[0].x =  0.0 ;
        g[0].y =  0.0 ;
        g[0].z =  0.0 ;
        g[0].w =  1.0 ;

        return(1) ;
}

int                     gs_1D1(g)
struct gausspts                 g[] ;

{
        g[0].x =  0.0 ;
        g[0].y =  0.0 ;
        g[0].z =  0.0 ;
        g[0].w =  2.0 ;

        return(1) ;
}

int                     gs_1D2(g)
struct gausspts                 g[] ;

{
        g[0].x = -0.577350269189626 ;
        g[0].y =  0.0 ;
        g[0].z =  0.0 ;
        g[0].w =  1.0 ;

```

```
        g[1].x =  0.577350269189626 ;
        g[1].y =  0.0 ;
        g[1].z =  0.0 ;
        g[1].w =  1.0 ;

        return(2) ;
}

int                     qu_1D2(g)
struct gausspts                 g[] ;

{
        g[0].x =  0.788675134594813 ;
        g[0].y =  0.0 ;
        g[0].z =  0.0 ;
        g[0].w =  0.5 ;

        g[1].x =  0.211324865405187 ;
        g[1].y =  0.0 ;
        g[1].z =  0.0 ;
        g[1].w =  0.5 ;

        return(2) ;
}

int                     qu_1D2r(g)
struct gausspts                 g[] ;

{
        g[0].x =  -0.788675134594813 ;
        g[0].y =  0.0 ;
        g[0].z =  0.0 ;
        g[0].w =  0.5 ;

        g[1].x =  -0.211324865405187 ;
        g[1].y =  0.0 ;
        g[1].z =  0.0 ;
        g[1].w =  0.5 ;

        return(2) ;
}

int                     gs_1D3(g)
struct gausspts                 g[] ;

{
        g[0].x = -0.774596669241483 ;
        g[0].y =  0.0 ;
        g[0].z =  0.0 ;
        g[0].w =  0.555555555555556 ;

        g[1].x =  0.0 ;
        g[1].y =  0.0 ;
        g[1].z =  0.0 ;
        g[1].w =  0.888888888888889 ;

        g[2].x =  0.774596669241483 ;
        g[2].y =  0.0 ;
        g[2].z =  0.0 ;
        g[2].w =  0.555555555555556 ;

        return(3) ;
}
```

```
int                        gs_1D5(g)
struct gausspts                  g[] ;

{
        g[0].x = -0.906179845938664 ;
        g[0].y =  0.0 ;
        g[0].z =  0.0 ;
        g[0].w =  0.236926885056189 ;

        g[1].x = -0.538469310105683 ;
        g[1].y =  0.0 ;
        g[1].z =  0.0 ;
        g[1].w =  0.478628670499366;

        g[2].x =  0.0 ;
        g[2].y =  0.0 ;
        g[2].z =  0.0 ;
        g[2].w =  0.568888888888889 ;

        g[3].x =  0.538469310105683 ;
        g[3].y =  0.0 ;
        g[3].z =  0.0 ;
        g[3].w =  0.478628670499366 ;

        g[4].x =  0.906179845938664 ;
        g[4].y =  0.0 ;
        g[4].z =  0.0 ;
        g[4].w =  0.236926885056189 ;

        return(5) ;
}

int                        gs_qu1D5(g)
struct gausspts                  g[] ;

{
        g[0].x =  0.046910077030668 ;
        g[0].y =  0.0 ;
        g[0].z =  0.0 ;
        g[0].w =  0.118463442528945 ;

        g[1].x =  0.2307653449471585 ;
        g[1].y =  0.0 ;
        g[1].z =  0.0 ;
        g[1].w =  0.239314335249683;

        g[2].x =  0.5 ;
        g[2].y =  0.0 ;
        g[2].z =  0.0 ;
        g[2].w =  0.284444444444445 ;

        g[3].x =  0.7692346550528415 ;
        g[3].y =  0.0 ;
        g[3].z =  0.0 ;
        g[3].w =  0.239314335249683 ;

        g[4].x =  0.953089922969332 ;
        g[4].y =  0.0 ;
        g[4].z =  0.0 ;
        g[4].w =  0.118463442528945 ;

        return(5) ;
```

```
}

int                         gs_qu1D5r(g)
struct gausspts                     g[] ;

{
        g[0].x = -0.046910077030668 ;
        g[0].y =  0.0 ;
        g[0].z =  0.0 ;
        g[0].w =  0.118463442528945 ;

        g[1].x = -0.2307653449471585 ;
        g[1].y =  0.0 ;
        g[1].z =  0.0 ;
        g[1].w =  0.239314335249683;

        g[2].x = -0.5 ;
        g[2].y =  0.0 ;
        g[2].z =  0.0 ;
        g[2].w =  0.284444444444445 ;

        g[3].x = -0.7692346550528415 ;
        g[3].y =  0.0 ;
        g[3].z =  0.0 ;
        g[3].w =  0.239314335249683 ;

        g[4].x = -0.953089922969332 ;
        g[4].y =  0.0 ;
        g[4].z =  0.0 ;
        g[4].w =  0.118463442528945 ;

        return(5) ;
}

int                         gs_2t1(g)
struct gausspts                     g[] ;

{
        g[0].x =  0.3333333333333 ;
        g[0].y =  0.3333333333333 ;
        g[0].z =  0.0 ;
        g[0].w =  0.5 ;

        return(1) ;
}

int                         gs_2D1(g)
struct gausspts                     g[] ;

{
        g[0].x =  0.0 ;
        g[0].y =  0.0 ;
        g[0].z =  0.0 ;
        g[0].w =  4.0 ;

        return(1) ;
}

int                         gs_2t2(g)
struct gausspts                     g[] ;

{
        g[0].x =  0.5 ;
```

```
        g[0].y =  0.5 ;
        g[0].z =  0.0 ;
        g[0].w =  0.166666666666667 ;

        g[1].x =  0.0 ;
        g[1].y =  0.5 ;
        g[1].z =  0.0 ;
        g[1].w =  0.166666666666667 ;

        g[2].x =  0.5 ;
        g[2].y =  0.0 ;
        g[2].z =  0.0 ;
        g[2].w =  0.166666666666667 ;

        return(3) ;
}

int                     gs_2D2(g)
struct gausspts              g[] ;

{
        g[0].x =  0.577350269189626 ;
        g[0].y = -0.577350269189626 ;
        g[0].z =  0.0 ;
        g[0].w =  1.0 ;

        g[1].x =  0.577350269189626 ;
        g[1].y =  0.577350269189626 ;
        g[1].z =  0.0 ;
        g[1].w =  1.0 ;

        g[2].x = -0.577350269189626 ;
        g[2].y =  0.577350269189626 ;
        g[2].z =  0.0 ;
        g[2].w =  1.0 ;

        g[3].x = -0.577350269189626 ;
        g[3].y = -0.577350269189626 ;
        g[3].z =  0.0 ;
        g[3].w =  1.0 ;

        return(4) ;
}


int                     qu_2D2(g)
struct gausspts              g[] ;

{
        g[0].x =  0.7886751345 ;
        g[0].y =  0.2113248655 ;
        g[0].z =  0.0 ;
        g[0].w =  0.25 ;

        g[1].x =  0.7886751345 ;
        g[1].y =  0.7886751345 ;
        g[1].z =  0.0 ;
        g[1].w =  0.25 ;

        g[2].x =  0.2113248655 ;
        g[2].y =  0.7886751345 ;
        g[2].z =  0.0 ;
        g[2].w =  0.25 ;
```

```
        g[3].x =  0.2113248655 ;
        g[3].y =  0.2113248655 ;
        g[3].z =  0.0 ;
        g[3].w =  0.25 ;

        return(4) ;
}

int                    gs_2D3b2(g)
struct gausspts             g[] ;

{
        g[0].x =  0.774596669241483 ;
        g[0].y = -0.577350269189626 ;
        g[0].z =  0.0 ;
        g[0].w =  0.555555555555556 ;

        g[1].x =  0.774596669241483 ;
        g[1].y =  0.577350269189626 ;
        g[1].z =  0.0 ;
        g[1].w =  0.555555555555556 ;

        g[2].x =  0.0 ;
        g[2].y = -0.577350269189626 ;
        g[2].z =  0.0 ;
        g[2].w =  0.888888888888889 ;

        g[3].x =  0.0 ;
        g[3].y =  0.577350269189626 ;
        g[3].z =  0.0 ;
        g[3].w =  0.888888888888889 ;

        g[4].x = -0.774596669241483 ;
        g[4].y = -0.577350269189626 ;
        g[4].z =  0.0 ;
        g[4].w =  0.555555555555556 ;

        g[5].x = -0.774596669241483 ;
        g[5].y =  0.577350269189626 ;
        g[5].z =  0.0 ;
        g[5].w =  0.555555555555556 ;

        return(6) ;
}

int                    gs_2D3(g)
struct gausspts             g[] ;

{
        g[0].x =  0.774596669241483 ;
        g[0].y = -0.774596669241483 ;
        g[0].z =  0.0 ;
        g[0].w =  0.3086419 ;

        g[1].x =  0.774596669241483 ;
        g[1].y =  0.0 ;
        g[1].z =  0.0 ;
        g[1].w =  0.4938271 ;

        g[2].x =  0.774596669241483 ;
        g[2].y =  0.774596669241483 ;
        g[2].z =  0.0 ;
```

```
        g[2].w =  0.3086419 ;

        g[3].x =  0.0 ;
        g[3].y =  0.774596669241483 ;
        g[3].z =  0.0 ;
        g[3].w =  0.4938271 ;

        g[4].x =  0.0 ;
        g[4].y =  0.0 ;
        g[4].z =  0.0 ;
        g[4].w =  0.7901233 ;

        g[5].x =  0.0 ;
        g[5].y = -0.774596669241483 ;
        g[5].z =  0.0 ;
        g[5].w =  0.4938271 ;

        g[6].x = -0.774596669241483 ;
        g[6].y = -0.774596669241483 ;
        g[6].z =  0.0 ;
        g[6].w =  0.3086419 ;

        g[7].x = -0.774596669241483 ;
        g[7].y =  0.0 ;
        g[7].z =  0.0 ;
        g[7].w =  0.4938271 ;

        g[8].x = -0.774596669241483 ;
        g[8].y =  0.774596669241483 ;
        g[8].z =  0.0 ;
        g[8].w =  0.3086419 ;

        return(9) ;
}
int                     qu_2D3(g)
struct gausspts                 g[] ;

{
        g[0].x =  0.112701665379259 ;
        g[0].y =  0.112701665379259 ;
        g[0].z =  0.0 ;
        g[0].w =  0.077160475 ;

        g[1].x =  0.112701665379259 ;
        g[1].y =  0.5 ;
        g[1].z =  0.0 ;
        g[1].w =  0.12345678 ;

        g[2].x =  0.112701665379259 ;
        g[2].y =  0.887298334621741 ;
        g[2].z =  0.0 ;
        g[2].w =  0.077160475 ;

        g[3].x =  0.5 ;
        g[3].y =  0.112701665379259 ;
        g[3].z =  0.0 ;
        g[3].w =  0.12345678 ;

        g[4].x =  0.5 ;
        g[4].y =  0.5 ;
        g[4].z =  0.0 ;
        g[4].w =  0.197530825 ;
```

```
        g[5].x =  0.5 ;
        g[5].y =  0.887298334621741 ;
        g[5].z =  0.0 ;
        g[5].w =  0.12345678 ;

        g[6].x =  0.887298334621741 ;
        g[6].y =  0.112701665379259 ;
        g[6].z =  0.0 ;
        g[6].w =  0.077160475 ;

        g[7].x =  0.887298334621741 ;
        g[7].y =  0.5 ;
        g[7].z =  0.0 ;
        g[7].w =  0.12345678 ;

        g[8].x =  0.887298334621741 ;
        g[8].y =  0.887298334621741 ;
        g[8].z =  0.0 ;
        g[8].w =  0.077160475 ;

        return(9) ;
}

int                     gs_2D4(g)
struct gausspts              g[] ;

{
        g[0].x =  0.861136311594053 ;
        g[0].y = -0.861136311594053 ;
        g[0].z =  0.0 ;
        g[0].w =  0.1210029932 ;

        g[1].x =  0.861136311594053 ;
        g[1].y =  -0.339981043584856 ;
        g[1].z =  0.0 ;
        g[1].w =  0.2268518518 ;

        g[2].x =  0.861136311594053 ;
        g[2].y =  0.339981043584856 ;
        g[2].z =  0.0 ;
        g[2].w =  0.2268518518 ;

        g[3].x =  0.861136311594053 ;
        g[3].y =  0.861136311594053 ;
        g[3].z =  0.0 ;
        g[3].w =  0.1210029932 ;

        g[4].x =  0.339981043584856 ;
        g[4].y =  0.861136311594053 ;
        g[4].z =  0.0 ;
        g[4].w =  0.2268518518 ;

        g[5].x =  -0.339981043584856 ;
        g[5].y =  0.861136311594053 ;
        g[5].z =  0.0 ;
        g[5].w =  0.2268518518 ;

        g[6].x = -0.861136311594053 ;
        g[6].y = 0.861136311594053 ;
        g[6].z =  0.0 ;
        g[6].w =  0.1210029932 ;

        g[7].x = -0.861136311594053 ;
```

```
        g[7].y =  0.339981043584856 ;
        g[7].z =  0.0 ;
        g[7].w =  0.2268518518 ;

        g[8].x = -0.861136311594053 ;
        g[8].y = -0.339981043584856 ;
        g[8].z =  0.0 ;
        g[8].w =  0.2268518518 ;

        g[9].x = -0.861136311594053 ;
        g[9].y = -0.861136311594053 ;
        g[9].z =  0.0 ;
        g[9].w =  0.1210029932 ;

        g[10].x = -0.339981043584856 ;
        g[10].y = -0.861136311594053 ;
        g[10].z =  0.0 ;
        g[10].w =  0.2268518518 ;

        g[11].x = 0.339981043584856 ;
        g[11].y = -0.861136311594053 ;
        g[11].z =  0.0 ;
        g[11].w =  0.2268518518 ;

        g[12].x = 0.339981043584856 ;
        g[12].y = -0.339981043584856 ;
        g[12].z =  0.0 ;
        g[12].w =  0.4252933019 ;

        g[13].x = 0.339981043584856 ;
        g[13].y = 0.339981043584856 ;
        g[13].z =  0.0 ;
        g[13].w =  0.4252933019 ;

        g[14].x = -0.339981043584856 ;
        g[14].y = 0.339981043584856 ;
        g[14].z =  0.0 ;
        g[14].w =  0.4252933019 ;

        g[15].x = -0.339981043584856 ;
        g[15].y = -0.339981043584856 ;
        g[15].z =  0.0 ;
        g[15].w =  0.4252933019 ;


        return(16) ;
}
```

```
#include "[illegible]e.h"


extern struct control  N ;


int                        get_shape(el_type,x,psf)  ,
int                        el_type ;
struct gausspts                  *x ;
struct shapefuncs          *psf ;

{
        int     err ;
        switch (el_type) {

                case 210 :
                        err = shap_2t1(x,psf)  ;
                        break ;

                case 211 :
                        err = shap_2s1(x,psf)  ;
                        break ;

                case 216 :
                        err = shap_2s1(x,psf)  ;
                        break ;

                case 220 :
                        err = shap_2t2(x,psf)  ;
                        break ;

                case 221 :
                        err = shap_2s2(x,psf)  ;
                        break ;

                case 226 :
                        err = shap_2s2(x,psf)  ;
                        break ;

                case 222 :
                        err = shap_212(x,psf)  ;
                        break ;

                case 227 :
                        err = shap_212(x,psf)  ;
                        break ;

                case 228 :
                        err = shqu6_212(x,psf)  ;
                        break ;

                case 229 :
                        err = shqu9_212(x,psf)  ;
                        break ;

                case 224 :
                        err = shqu8_212(x,psf)  ;
                        break ;

                case 219 :
                        err = shqu_211(x,psf)  ;
                        break ;

```

```
                case 231 :
                        err = shap_2s3(x,psf) ;
                        break ;

                case 111 :
                        err = shap_1s1(x,psf) ;
                        break ;

                case 121 :
                        err = shap_1s2(x,psf) ;
                        break ;

                case 128 :
                        err = shcu3r_1s2(x,psf) ;
                        break ;

                case 129 :
                        err = shcu3_1s2(x,psf) ;
                        break ;

                case 139 :
                        err = shapcu_1s3(x,psf) ;
                        break ;

                case 138 :
                        err = shapcur_1s3(x,psf) ;
                        break ;

                case 131 :
                        err = shap_1s3(x,psf) ;
                        break ;

                case 1 :
                        err = shap_pt(x,psf) ;
                        break ;

                case 230 :
                        err = shcu2_2x1(x,psf) ;
                        break ;

                case 2311 :
                        err = shcu3_2x2(x,psf) ;
                        break ;

                case 232 :
                        err = shcu4_2x2(x,psf) ;
                        break ;

                case 233 :
                        err = shcu3_3x1(x,psf) ;
                        break ;

                case 2341 :
                        err = shcu4_3x2(x,psf) ;
                        break ;

                case 2342 :
                        err = sh4_3x2b(x,psf) ;
                        break ;

                case 2351 :
                        err = shcu5_3x2(x,psf) ;
                        break ;
```

```
                case 2353 :
                        err = sh5_3x2b(x,psf) ;
                        break ;

                case 2352 :
                        err = shcu5_3x3(x,psf) ;
                        break ;

                case 2361 :
                        err = shcu6_3x2(x,psf) ;
                        break ;

                case 2363 :
                        err = sh6_3x2b(x,psf) ;
                        break ;

                case 2362 :
                        err = shcu6_3x3(x,psf) ;
                        break ;

                case 2371 :
                        err = shcu7_3x3(x,psf) ;
                        break ;

                case 2372 :
                        err = sh7_3x3b(x,psf) ;
                        break ;

                case 238 :
                        err = shcu8_3x3(x,psf) ;
                        break ;

                case 239 :
                        err = shcu9_3x3(x,psf) ;
                        break ;

                case 235 :
                        err = hermite(x,psf) ;
                        break ;
        }
        return(err) ;
}


int     nsf(el_type)
int     el_type ;

{
        struct gausspts         x ;
        struct shapefuncs       psf ;
        int                     err ;

        x.x = 100.0 ;
        x.y = 100.0 ;
        x.z = 100.0 ;
        err = get_shape(el_type,&x,&psf) ;
        return(err) ;
}


int     nbounds(el_type)
int     el_type ;
```

```
{
        int     err ;

        err = 0 ;
        switch (el_type) {

                case 210 :
                        err = 3 ;
                        break ;

                case 211 :
                        err = 4 ;
                        break ;

                case 216 :
                        err = 4 ;
                        break ;

                case 220 :
                        err = 6 ;
                        break ;

                case 221 :
                        err = 8 ;
                        break ;

                case 222 :
                        err = 8 ;
                        break ;

                case 226 :
                        err = 8 ;
                        break ;

                case 227 :
                        err = 8 ;
                        break ;

                case 224 :
                        err = 4 ;
                        break ;

                case 228 :
                        err = 4 ;
                        break ;

                case 229 :
                        err = 4 ;
                        break ;

                case 239 :
                        err = 4 ;
                        break ;

                case 230 :
                        err = 4 ;
                        break ;

                case 2311 :
                        err = 4 ;
                        break ;
```

```
                case 232 :
                        err = 4 ;
                        break ;

                case 233 :
                        err = 4 ;
                        break ;

                case 2341 :
                        err = 4 ;
                        break ;

                case 2342 :
                        err = 4 ;
                        break ;

                case 2351 :
                        err = 4 ;
                        break ;

                case 2353 :
                        err = 4 ;
                        break ;

                case 2352 :
                        err = 4 ;
                        break ;

                case 2361 :
                        err = 4 ;
                        break ;

                case 2363 :
                        err = 4 ;
                        break ;

                case 2362 :
                        err = 4 ;
                        break ;

                case 2371 :
                        err = 4 ;
                        break ;

                case 2372 :
                        err = 4 ;
                        break ;

                case 238 :
                        err = 4 ;
                        break ;

                case 231 :
                        err = 12 ;
                        break ;

                case 111 :
                        err = 2 ;
                        break ;

                case 118 :
                        err = 2 ;
                        break ;
```

```
            case 119 :
                err = 2 ;
                break ;

            case 128 :
                err = 2 ;
                break ;

            case 129 :
                err = 2 ;
                break ;

            case 139 :
                err = 2 ;
                break ;

            case 138 :
                err = 2 ;
                break ;

            case 121 :
                err = 2 ;
                break ;

            case 131 :
                err = 2 ;
                break ;

            case 1 :
                err = 1 ;
                break ;

            case 235 :
                err = 4 ;
                break ;
        }
        return(err) ;
}
```

festuff1.c

```
#include "fe.h"


extern struct control N ;
int    shap_2t1(x,p)
struct gausspts            *x ;
struct shapefuncs     *p ;

{
        double r,s,mr,ms,pr,ps,t ;

        r = x->x ;
        s = x->y ;
        t = 1.0 - r - s ;
        if( (r<-0.0001) || (r>1.0001) || (s<-0.00001) || (s>1.00001) || (t<-
0.00001) ) return(3) ;

        p->dof = 3 ;

        p->f[0] = r ;
        p->f[1] = s ;
        p->f[2] = t ;

        if(x->w < 0.0)
                return(0) ;

        p->dfdr[0] =  1.0 ;
        p->dfdr[1] =  0.0 ;
        p->dfdr[2] = -1.0 ;

        p->dfds[0] =  0.0 ;
        p->dfds[1] =  1.0 ;
        p->dfds[2] = -1.0 ;

        return(0) ;
}


int    shap_2s1(x,p)
struct gausspts            *x ;
struct shapefuncs     *p ;

{
        double r,s,mr,ms,pr,ps,t ;

        r = x->x ;
        s = x->y ;
        if( (r<-1.00001) || (r>1.00001) || (s<-1.00001) || (s>1.00001) )
return(4) ;

        t = 0.25 ;
        mr = t * (1.0 - r) ;
        ms = 1.0 - s ;
        pr = t * (1.0 + r) ;
        ps = 1.0 + s ;

        p->dof = 4 ;

        p->f[0] = pr * ms ;
        p->f[1] = pr * ps ;
        p->f[2] = mr * ps ;
        p->f[3] = mr * ms ;

```

```
        if(x->w < 0.0)
                return(0) ;
        ms *= t ;
        ps *= t ;

        p->dfdr[0] =  ms ;
        p->dfdr[1] =  ps ;
        p->dfdr[2] = -ps ;
        p->dfdr[3] = -ms ;

        p->dfds[0] = -pr ;
        p->dfds[1] =  pr ;
        p->dfds[2] =  mr ;
        p->dfds[3] = -mr ;

        return(0) ;
}

int     shcu3_ls2(x,p)
struct  gausspts                *x ;
struct  shapefuncs      *p ;

{
        double r, mr, pr, rp3, t, tt ;

        r = x->x ;
        if( (r<-1.0) || (r>1.0) ) return(3) ;

        t = 0.125 ;
        tt = 0.25 ;
        mr = 1.0 - r ;
        pr = 1.0 + r ;
        rp3 = 3.0 + r ;

        p->dof = 3 ;

        p->f[0] = tt * rp3 * mr ;
        p->f[1] = t * rp3 * pr ;
        p->f[2] = -t * pr * mr ;

        if(x->w < 0.0)
                return(0) ;

        p->dfdr[0] = -0.5 * pr ;
        p->dfdr[1] = tt *(2.0+r)   ;
        p->dfdr[2] = tt * r ;

/*      p->d2fdr[0] = -0.5 ;
        p->d2fdr[1] = tt   ;
        p->d2fdr[2] = tt ;
*/
        return(0) ;
}
int     shapcu_ls3(x,p)
struct  gausspts                *x ;
struct  shapefuncs      *p ;

{
        double r,rm,rp,rp3,rp5,t ;

        r = x->x ;
        if( (r<-1.0) || (r>1.0) ) return(4) ;

```

```
        t = 1.0/48.0  ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
        rp3 = r + 3.0 ;
        rp5 = r + 5.0 ;

        p->dof = 4 ;

        p->f[0] =  3.0*t * rm * rp3 * rp5 ;
        p->f[1] =  t * rp * rp3 * rp5 ;
        p->f[2] = -3.0*t * rp * rm * rp5 ;
        p->f[3] = t * rp * rm * rp3 ;

        if(x->w < 0.0)
              return(0) ;

        p->dfdr[0] = -3.0 * t * ( 7.0 + (14.0 + 3.0*r)*r) ;
        p->dfdr[1] = t * ( 23.0 + r * (18.0 + 3.0*r))  ;
        p->dfdr[2] = -3.0 * t * ( 1.0 - r*(10.0 +3.0*r)) ;
        p->dfdr[3] = t * ( 1.0 -3.0*r*(2.0 + r)) ;

        return(0) ;
}
int     shcu3r_1s2(x,p)
struct gausspts           *x ;
struct shapefuncs    *p ;

{
        double r,mr,pr,t ;

        r = x->x ;
        if( (r<-1.0) || (r>1.0) ) return(3) ;

        t = 0.125 ;
        mr = 1.0 - r ;
        pr = 1.0 + r ;

        p->dof = 3 ;

        p->f[0] = t *(3.0- r) * mr ;
        p->f[1] = 2*t* (3.0- r) * pr ;
        p->f[2] = -t * pr * mr ;

        if(x->w < 0.0)
              return(0) ;

        p->dfdr[0] = -2.0*t * (2.0-r) ;
        p->dfdr[1] = 4.0 * t*mr  ;
        p->dfdr[2] = 2.0*t * r ;

        return(0) ;
}int    shapcur_1s3(x,p)
struct gausspts           *x ;
struct shapefuncs    *p ;

{
        double r,rm,rp,rp3,rp5,t ;

        r = x->x ;
        if( (r<-1.0) || (r>1.0) ) return(4) ;

        t = 1.0/48.0  ;
        rm = 1.0 - r ;
```

```
        rp = 1.0 + r ;
        rp3 = 3.0 - r ;
        rp5 = 5.0 - r;

        p->dof = 4 ;

        p->f[0] = t * rm * rp3 * rp5 ;
        p->f[1] =  3.0 * t * rp * rp3 * rp5 ;
        p->f[2] = -3.0*t * rp * rm * rp5 ;
        p->f[3] = t * rp * rm * rp3 ;

        if(x->w < 0.0)
            return(0) ;

        p->dfdr[0] = t * ( -23.0 + (6.0 - r)*r*3.0) ;
        p->dfdr[1] = 3.0 * t * (7.0 - r * (14.0 - 3.0*r))  ;
        p->dfdr[2] = 3.0 * t * ( 1.0 + r*(10.0 -3.0*r)) ;
        p->dfdr[3] = t * ( -1.0 -3.0*r*(2.0 - r)) ;

        return(0) ;
}


int     shap_2t2(x,p)
struct  gausspts            *x ;
struct  shapefuncs      *p ;

{
        double r,s,mr,ms,pr,ps,t ;

        r = x->x ;
        s = x->y ;
        t = 1.0 - r - s ;
        if( (r<-0.00001) || (r>1.00001) || (s<-0.00001) || (s>1.00001) || (t<-
0.00001) ) return(6) ;

        p->dof = 6 ;

        p->f[0] = r * (2.0*r - 1.0) ;
        p->f[1] = 4.0 * r * s ;
        p->f[2] = s * (2.0*s - 1.0) ;
        p->f[3] = 4.0 * s * t ;
        p->f[4] = t * (2.0*t - 1.0) ;
        p->f[5] = 4.0 * r * t ;

        if(x->w < 0.0)
            return(0) ;

        p->dfdr[0] =  4.0 * r - 1.0 ;
        p->dfdr[1] =  4.0 * s ;
        p->dfdr[2] =  0.0 ;
        p->dfdr[3] = -4.0 * s ;
        p->dfdr[4] = -4.0 * t + 1.0 ;
        p->dfdr[5] =  4.0 * (t - r) ;

        p->dfds[0] =  0.0 ;
        p->dfds[1] =  4.0 * r ;
        p->dfds[2] =  4.0 * s - 1.0 ;
        p->dfds[3] =  4.0 * (t - s) ;
        p->dfds[4] = -4.0 * t + 1.0 ;
        p->dfds[5] = -4.0 * r ;

        return(0) ;
```

```
}


int     shap_2s2(x,p)
struct gausspts             *x ;
struct shapefuncs     *p ;

{
        double r,s,mr,ms,pr,ps,t,tr,ts,tprps,tmrps,tprms,tmrms ;
        double trms,trps,tsmr,tspr,tps,tms,tpr,tmr ;

        r = x->x ;
        s = x->y ;
        if( (r<-1.00001) || (r>1.00001) || (s<-1.00001) || (s>1.00001) )
              return(8) ;

        t = 0.25 ;
        mr = 1.0 - r ;
        ms = 1.0 - s ;
        pr = 1.0 + r ;
        ps = 1.0 + s ;
        tprms = t * pr * ms ;
        tprps = t * pr * ps ;
        tmrps = t * mr * ps ;
        tmrms = t * mr * ms ;

        p->dof = 8 ;

        p->f[0] = tprms * (r - ps) ;
        p->f[2] = tprps * (r - ms) ;
        p->f[4] = tmrps * (s - pr) ;
        p->f[6] = tmrms * (-s - pr) ;
        tprms *= 2.0 ;
        tmrps *= 2.0 ;
        p->f[1] = tprms * ps ;
        p->f[3] = tmrps * pr ;
        p->f[5] = tmrps * ms ;
        p->f[7] = tprms * mr ;

        if(x->w < 0.0)
              return(0) ;

        ts = 2.0 * s ;
        tr = 2.0 * r ;
        trms = tr - s ;
        trps = tr + s ;
        tsmr = ts - r ;
        tspr = ts + r ;
        tps = t * ps ;
        tms = t * ms ;
        tpr = t * pr ;
        tmr = t * mr ;

        p->dfdr[0] = tms * (trms) ;
        p->dfdr[2] = tps * (trps) ;
        p->dfdr[4] = tps * (trms) ;
        p->dfdr[6] = tms * (trps) ;
        p->dfdr[1] =  0.5 * ms * ps ;
        p->dfdr[3] = -r * ps ;
        p->dfdr[5] = -(p->dfdr[1]) ;
        p->dfdr[7] = -r * ms ;

        p->dfds[0] = tpr * (tsmr) ;
```

```
        p->dfds[2] = tpr * (tspr) ;
        p->dfds[4] = tmr * (tsmr) ;
        p->dfds[6] = tmr * (tspr) ;
        p->dfds[1] = -s * pr ;
        p->dfds[3] = 0.5 * pr * mr ;
        p->dfds[5] = -s * mr ;
        p->dfds[7] = - (p->dfds[3]) ;

        return(0) ;
}

int     shap_212(x,p)
struct  gausspts            *x ;
struct  shapefuncs     *p ;

{
        double r,s,mr,ms,pr,ps,t,tr,ts,pmr,pms ;

        r = x->x ;
        s = x->y ;
        if( (r<-1.00001) || (r>1.00001) || (s<-1.00001) || (s>1.00001) )
              return(9) ;

        t = 0.25 ;
        tr = 2.0 * r ;
        ts = 2.0 * s ;
        mr = 1.0 - r ;
        ms = 1.0 - s ;
        pr = 1.0 + r ;
        ps = 1.0 + s ;
        pmr = pr * mr ;
        pms = ps * ms ;

        p->dof = 9 ;

        p->f[0] = -t * pr * ms * r * s ;
        p->f[2] = t * pr * ps * r * s ;
        p->f[4] = -t * r * mr * s * ps ;
        p->f[6] = t * r * mr * s * ms ;
        p->f[1] = 0.5 * r * pr * pms ;
        p->f[3] = 0.5 * pmr * s * ps ;
        p->f[5] = -0.5 * r * mr * pms ;
        p->f[7] = -0.5 * pmr * s * ms ;
        p->f[8] = pmr * pms ;

        if(x->w < 0.0)
              return(0) ;

        p->dfdr[0] = -t * s * ms * (tr + 1.0) ;
        p->dfdr[2] = t * s * ps * (1.0 + tr) ;
        p->dfdr[4] = -t * s * ps * (1.0 - tr) ;
        p->dfdr[6] = t * s * ms * (1.0 - tr) ;
        p->dfdr[1] =  0.5 * pms * (1.0 + tr) ;
        p->dfdr[3] = -r * s * ps ;
        p->dfdr[5] = -0.5 * (1.0 - tr) * pms ;
        p->dfdr[7] =  r * s * ms ;
        p->dfdr[8] = -2.0 * r * pms ;

        p->dfds[0] = -t * r * pr * (1.0 - ts) ;
        p->dfds[2] = t * r * pr * (1.0 + ts) ;
        p->dfds[4] = -t * r * mr * (1.0 + ts) ;
        p->dfds[6] = t * r * mr * (1.0 - ts) ;
        p->dfds[1] = -s * r * pr ;
```

```
        p->dfds[3] = 0.5 * pmr * (1.0 + ts) ;
        p->dfds[5] = s * r * mr ;
        p->dfds[7] = -0.5 * pmr * (1.0 - ts) ;
        p->dfds[8] = -2.0 * s * pmr ;

        return(0) ;
}
int     shap_2s3(x,p)
struct gausspts          *x ;
struct shapefuncs    *p ;

{
        double r,s,mr,ms,pr,ps,m3r,p3r,m3s,p3s,r2,mr2,s2,ms2,frs,t,nt ;
        double temp, mr2p3r,mr2m3r,ms2p3s,ms2m3s ;

        r = x->x ;
        s = x->y ;
        if( (r<-1.00001) || (r>1.00001) || (s<-1.00001) || (s>1.00001) )
              return(12) ;

        t = 0.03125 ;
        nt = 0.28125 ;
        p3r = 1.0 + 3.0 * r ;
        m3r = 1.0 - 3.0 * r ;
        p3s = 1.0 + 3.0 * s ;
        m3s = 1.0 - 3.0 * s ;
        mr = 1.0 - r ;
        ms = 1.0 - s ;
        pr = 1.0 + r ;
        ps = 1.0 + s ;
        r2 = r * r ;
        s2 = s * s ;
        mr2 = nt * (1.0 - r2) ;
        ms2 = nt * (1.0 - s2) ;
        frs = t * (-10 + 9 * (r2 + s2)) ;
        ms2m3s = ms2 * m3s ;
        ms2p3s = ms2 * p3s ;
        mr2m3r = mr2 * m3r ;
        mr2p3r = mr2 * p3r ;

        p->dof = 12 ;

        p->f[0] = pr * ms * frs ;
        p->f[3] = pr * ps * frs ;
        p->f[6] = mr * ps * frs ;
        p->f[9] = mr * ms * frs ;
        p->f[1] = pr * ms2m3s ;
        p->f[2] = pr * ms2p3s ;
        p->f[4] = ps * mr2p3r ;
        p->f[5] = ps * mr2m3r ;
        p->f[7] = mr * ms2p3s ;
        p->f[8] = mr * ms2m3s ;
        p->f[10] = ms * mr2m3r ;
        p->f[11] = ms * mr2p3r ;

        if(x->w < 0.0)
              return(0) ;

        nt *= 2.0 ;
        r *= nt ;
        s *= nt ;
        m3s *= -s ;
        p3s *= -s ;
```

```
        m3r *= -r ;
        p3r *= -r ;
        ms2 *= 3.0 ;
        mr2 *= 3.0 ;

        temp = frs + pr * r ;
        p->dfdr[0] = ms * temp ;
        p->dfdr[3] = ps * temp ;
        temp = mr * r - frs ;
        p->dfdr[6] = ps * temp ;
        p->dfdr[9] = ms * temp ;

        p->dfdr[1] =  ms2m3s ;
        p->dfdr[2] =  ms2p3s ;
        p->dfdr[4] =  ps * (p3r + mr2) ;
        p->dfdr[5] =  ps * (m3r - mr2) ;
        p->dfdr[7] = -ms2p3s ;
        p->dfdr[8] = -ms2m3s ;
        p->dfdr[10] =  ms * (m3r - mr2) ;
        p->dfdr[11] =  ms * (p3r + mr2) ;

        temp = frs + ps * s ;
        p->dfds[3] = pr * temp ;
        p->dfds[6] = mr * temp ;
        temp = ms * s - frs ;
        p->dfds[0] = pr * temp ;
        p->dfds[9] = mr * temp ;

        p->dfds[1] =  pr * (m3s - ms2) ;
        p->dfds[2] =  pr * (p3s + ms2) ;
        p->dfds[4] =  mr2p3r ;
        p->dfds[5] =  mr2m3r ;
        p->dfds[7] =  mr * (p3s + ms2) ;
        p->dfds[8] =  mr * (m3s - ms2) ;
        p->dfds[10] =  -mr2m3r ;
        p->dfds[11] =  -mr2p3r ;

        return(0) ;
}
int     shap_1s1(x,p)
struct gausspts           *x ;
struct shapefuncs     *p ;

{
        double r,mr,pr,t ;

        r = x->x ;
        if( (r<-1.0) || (r>1.0) ) return(2) ;

        t = 0.5 ;
        mr = 1.0 - r ;
        pr = 1.0 + r ;

        p->dof = 2 ;

        p->f[0] =  t * mr ;
        p->f[1] =  t * pr ;

        if(x->w < 0.0)
                return(0) ;

        p->dfdr[0] = -0.5 ;
        p->dfdr[1] =  0.5 ;
```

```
        return(0) ;
}
int     shap_ls2(x,p)
struct gausspts             *x ;
struct shapefuncs     *p ;

{
        double r,mr,pr,t ;

        r = x->x ;
        if( (r<-1.0) || (r>1.0) ) return(3) ;

        t = 0.5 ;
        mr = 1.0 - r ;
        pr = 1.0 + r ;

        p->dof = 3 ;

        p->f[0] = -t * r * mr ;
        p->f[1] = pr * mr ;
        p->f[2] = t * pr * r ;

        if(x->w < 0.0)
                return(0) ;

        p->dfdr[0] = -t * (1.0 - 2.0*r) ;
        p->dfdr[1] = -2.0 * r   ;
        p->dfdr[2] = t * (1.0 + 2.0*r) ;

        return(0) ;
}
int     shap_ls3(x,p)
struct gausspts             *x ;
struct shapefuncs     *p ;

{
        double r,rm,rp,r3m,r3p,t ;

        r = x->x ;
        if( (r<-1.0) || (r>1.0) ) return(4) ;

        t = 9.0/144.0  ;
        rm = r - 1.0 ;
        rp = 1.0 + r ;
        r3m = 3.0 * r - 1.0 ;
        r3p = 3.0 * r + 1.0 ;

        p->dof = 4 ;

        p->f[0] = -t * rm * r3p * r3m ;
        p->f[1] = 9.0 * t * rp * rm * r3m ;
        p->f[2] = -9.0 * t * rp * rm * r3p ;
        p->f[3] = t * rp * r3p * r3m ;

        if(x->w < 0.0)
                return(0) ;

        p->dfdr[0] = -t * ( r3p * r3m + 3.0*rm*r3m + 3.0*rm*r3p) ;
        p->dfdr[1] = 9.0 * t * ( 3.0*rp * rm + rm*r3m + rm*r3p)  ;
        p->dfdr[2] = -9.0 * t * ( 3.0*rp * rm + rm*r3p + rp*r3p) ;
        p->dfdr[3] = t * ( r3p * r3m + 3.0*rm*r3m + 3.0*rm*r3p) ;
```

```
        return(0) ;
}
int     shap_pt(x,p)
struct gausspts             *x ;
struct shapefuncs     *p ;

{
        double r;

        r = x->x ;
        if( (r<-1.0) || (r>1.0) ) return(1) ;

        p->dof = 1 ;
        p->f[0] =  1 ;
        p->dfdr[0] = 0.0 ;

        return(0) ;
}
int     hermite(x,p)
struct gausspts             *x ;
struct shapefuncs     *p ;

{
/*
        double r,s,mr,ms,pr,ps,t,tr,ts ;

        r = x->x ;
        s = x->y ;
        if( (r<-1.00001) || (r>1.00001) || (s<-1.00001) || (s>1.00001) )
              return(4) ;

        t = 1.0/16.0 ;
        tr = 2.0 * r ;
        ts = 2.0 * s ;
        mr = r - 1.0 ;
        ms = s - 1.0 ;
        pr = 1.0 + r ;
        ps = 1.0 + s ;

        p->dof = 12 ;

        p->f[0] = t * pr * pr * (r - 2.0) * ms * ms * (-s - 2.0) ;
        p->f[3] = t * pr * pr * (r - 2.0) * ps * ps * (s - 2.0) ;
        p->f[6] = t * mr * mr * (-r - 2.0) * ps * ps * (s - 2.0) ;
        p->f[9] = t * mr * mr * (-r - 2.0) * ms * ms * (-s - 2.0) ;
        p->f[1] = -t * pr * pr * (r - 1.0) * ms * ms * (-s - 2.0) ;
        p->f[4] = -t * pr * pr * (r - 1.0) * ps * ps * (s - 2.0) ;
        p->f[7] = t * mr * mr * (-r - 1.0) * ps * ps * (s - 2.0) ;
        p->f[10] = t * mr * mr * (-r - 1.0) * ms * ms * (-s - 2.0) ;
        p->f[2] = t * pr * pr * (r - 2.0) * ms * ms * (-s - 1.0) ;
        p->f[5] = -t * pr * pr * (r - 2.0) * ps * ps * (s - 1.0) ;
        p->f[8] = -t * mr * mr * (-r - 2.0) * ps * ps * (s - 1.0) ;
        p->f[11] = t * mr * mr * (-r - 2.0) * ms * ms * (-s - 1.0) ;

        if(x->w < 0.0)
              return(0) ;

        p->dfdr[0] = t * pr * (3*r - 3.0) * ms * ms * (-s - 2.0) ;
        p->dfdr[3] = t * pr * (3*r - 3.0) * ps * ps * (s - 2.0) ;
        p->dfdr[6] = t * mr * (-3*r - 3.0) * ps * ps * (s - 2.0) ;
        p->dfdr[9] = t * mr * (-3*r - 3.0) * ms * ms * (-s - 2.0) ;
        p->dfdr[1] = -t * pr * (3*r - 1.0) * ms * ms * (-s - 2.0) ;
        p->dfdr[4] = -t * pr * (3*r - 1.0) * ps * ps * (s - 2.0) ;
```

```
        p->dfdr[7] = t * mr * (-3*r - 1.0) * ps * ps * (s - 2.0) ;
        p->dfdr[10] = t * mr * (-3*r - 1.0) * ms * ms * (-s - 2.0) ;
        p->dfdr[2] = t * pr * (3*r - 3.0) * ms * ms * (-s - 1.0) ;
        p->dfdr[5] = -t * pr * (3*r - 3.0) * ps * ps * (s - 1.0) ;
        p->dfdr[8] = -t * mr * (-3*r - 3.0) * ps * ps * (s - 1.0) ;
        p->dfdr[11] = t * mr * (-3*r - 3.0) * ms * ms * (-s - 1.0) ;

        p->dfds[0] = t * pr * pr * (r - 2.0) * ms * (-3*s - 3.0) ;
        p->dfds[3] = t * pr * pr * (r - 2.0) * ps * (3*s - 3.0) ;
        p->dfds[6] = t * mr * mr * (-r - 2.0) * ps * (3*s - 3.0) ;
        p->dfds[9] = t * mr * mr * (-r - 2.0) * ms * (-3*s - 3.0) ;
        p->dfds[1] = -t * pr * pr * (r - 1.0) * ms * (-3*s - 3.0) ;
        p->dfds[4] = -t * pr * pr * (r - 1.0) * ps * (3*s - 3.0) ;
        p->dfds[7] = t * mr * mr * (-r - 1.0) * ps * (3*s - 3.0) ;
        p->dfds[10] = t * mr * mr * (-r - 1.0) * ms * (-3*s - 3.0) ;
        p->dfds[2] = t * pr * pr * (r - 2.0) * ms * (-3*s - 1.0) ;
        p->dfds[5] = -t * pr * pr * (r - 2.0) * ps * (3*s - 1.0) ;
        p->dfds[8] = -t * mr * mr * (-r - 2.0) * ps * (3*s - 1.0) ;
        p->dfds[11] = t * mr * mr * (-r - 2.0) * ms * (-3*s - 1.0) ;

        */

        return(0) ;
}



int     shqu_211(x,p)
struct gausspts             *x ;
struct shapefuncs    *p ;

{
        double r,s,mr,ms ;

        r = x->x ;
        s = x->y ;
        if( (r<-1.00001) || (r>1.00001) || (s<-1.00001) || (s>1.00001) )
return(4) ;

        mr = 1.0 - r ;
        ms = 1.0 - s ;

        p->dof = 4 ;

        p->f[0] = r * ms ;
        p->f[1] = r * s ;
        p->f[2] = mr * s ;
        p->f[3] = mr * ms ;

        if(x->w < 0.0)
            return(0) ;

        p->dfdr[0] =  ms ;
        p->dfdr[1] =  s ;
        p->dfdr[2] = -s ;
        p->dfdr[3] = -ms ;

        p->dfds[0] = -r ;
        p->dfds[1] =  r ;
        p->dfds[2] =  mr ;
        p->dfds[3] = -mr ;

        return(0) ;
```

```
}


int     shqu9_212(x,p)
struct gausspts              *x ;
struct shapefuncs     *p ;

{
        double r,s,mr,ms,pr,ps,t,tr,ts,pmr,pms ;

        r = x->x ;
        s = x->y ;
        if( (r<-1.00001) || (r>1.00001) || (s<-1.00001) || (s>1.00001) )
return(9) ;

        t = 0.25 ;
        tr = 2.0 * r ;
        ts = 2.0 * s ;
        mr = 1.0 - r ;
        ms = 1.0 - s ;
        pr = 1.0 + r ;
        ps = 1.0 + s ;
        pmr = pr * mr ;
        pms = ps * ms ;

        p->dof = 9 ;

        p->f[8] = -t * pr * ms * r * s ;
        p->f[1] = t * pr * ps * r * s ;
        p->f[4] = -t * r * mr * s * ps ;
        p->f[6] = t * r * mr * s * ms ;
        p->f[0] = 0.5 * r * pr * pms ;
        p->f[2] = 0.5 * pmr * s * ps ;
        p->f[5] = -0.5 * r * mr * pms ;
        p->f[7] = -0.5 * pmr * s * ms ;
        p->f[3] = pmr * pms ;

        if(x->w < 0.0)
                return(0) ;

        p->dfdr[8] = -t * s * ms * (tr + 1.0) ;
        p->dfdr[1] = t * s * ps * (1.0 + tr) ;
        p->dfdr[4] = -t * s * ps * (1.0 - tr) ;
        p->dfdr[6] = t * s * ms * (1.0 - tr) ;
        p->dfdr[0] =  0.5 * pms * (1.0 + tr) ;
        p->dfdr[2] = -r * s * ps ;
        p->dfdr[5] = -0.5 * (1.0 - tr) * pms ;
        p->dfdr[7] =  r * s * ms ;
        p->dfdr[3] = -2.0 * r * pms ;

        p->dfds[8] = -t * r * pr * (1.0 - ts) ;
        p->dfds[1] = t * r * pr * (1.0 + ts) ;
        p->dfds[4] = -t * r * mr * (1.0 + ts) ;
        p->dfds[6] = t * r * mr * (1.0 - ts) ;
        p->dfds[0] = -s * r * pr ;
        p->dfds[2] = 0.5 * pmr * (1.0 + ts) ;
        p->dfds[5] = s * r * mr ;
        p->dfds[7] = -0.5 * pmr * (1.0 - ts) ;
        p->dfds[3] = -2.0 * s * pmr ;

        return(0) ;
}
```

```
int     shqu8_212(x,p)
struct gausspts             *x ;
struct shapefuncs      *p ;

{
        double r,s,mr,ms,pr,ps,t,tr,ts,pmr,pms ;

        r = x->x ;
        s = x->y ;
        if( (r<-1.00001) || (r>1.00001) || (s<-1.00001) || (s>1.00001) )
return(8) ;

        t = 0.25 ;
        tr = 2.0 * r ;
        ts = 2.0 * s ;
        mr = 1.0 - r ;
        ms = 1.0 - s ;
        pr = 1.0 + r ;
        ps = 1.0 + s ;
        pmr = pr * mr ;
        pms = ps * ms ;

        p->dof = 8 ;

        p->f[7] = -0.5 * ms * r * s ;
        p->f[1] = t * pr * ps * r * s ;
        p->f[4] = -0.5 * r * mr * s ;
        p->f[0] = 0.5 * r * pr * pms ;
        p->f[2] = 0.5 * pmr * s * ps ;
        p->f[5] = -0.5 * r * mr * ms ;
        p->f[6] = -0.5 * mr * s * ms ;
        p->f[3] = pmr * pms ;

        if(x->w < 0.0)
                return(0) ;

        p->dfdr[7] = -0.5 * ms * s  ;
        p->dfdr[1] = t * s * ps * (1.0 + tr) ;
        p->dfdr[4] = -0.5 * s * (1.0 - tr) ;
        p->dfdr[0] =  0.5 * pms * (1.0 + tr) ;
        p->dfdr[2] = -r * s * ps ;
        p->dfdr[5] = -0.5 * (1.0 - tr) * ms ;
        p->dfdr[6] =  0.5 * s * ms ;
        p->dfdr[3] = -2.0 * r * pms ;

        p->dfds[7] = 0.5 * r * (1.0 - ts) ;
        p->dfds[1] = t * r * pr * (1.0 + ts) ;
        p->dfds[4] = -0.5 * r * mr ;
        p->dfds[0] = -s * r * pr ;
        p->dfds[2] = 0.5 * pmr * (1.0 + ts) ;
        p->dfds[5] = 0.5 * r * mr ;
        p->dfds[6] = -0.5 * mr * (1.0 - ts) ;
        p->dfds[3] = -2.0 * s * pmr ;

        return(0) ;
}

int     shqu6_212(x,p)
struct gausspts             *x ;
struct shapefuncs      *p ;

{
```

```
        double r,s,mr,ms,pr,ps,t,tr,ts,pmr,pms ;

        r = x->x ;
        s = x->y ;
        if( (r<-1.00001) || (r>1.00001) || (s<-1.00001) || (s>1.00001) )
                return(6) ;

        t = 0.25 ;
        tr = 2.0 * r ;
        ts = 2.0 * s ;
        mr = 1.0 - r ;
        ms = 1.0 - s ;
        pr = 1.0 + r ;
        ps = 1.0 + s ;
        pmr = pr * mr ;
        pms = ps * ms ;

        p->dof = 6 ;

        p->f[1] = 0.5 * pr * r * s ;
        p->f[4] = -0.5 * r * mr * s ;
        p->f[0] = 0.5 * r * pr * ms ;
        p->f[2] = pmr * s  ;
        p->f[5] = -0.5 * r * mr * ms ;
        p->f[3] = pmr * ms ;

        if(x->w < 0.0)
                return(0) ;

        p->dfdr[1] =  0.5 * s * (1.0 + tr) ;
        p->dfdr[4] = -0.5 * s * (1.0 - tr) ;
        p->dfdr[0] =  0.5 * ms * (1.0 + tr) ;
        p->dfdr[2] = -2.0 * r * s ;
        p->dfdr[5] = -0.5 * (1.0 - tr)  * ms ;
        p->dfdr[3] = -2.0 * r * ms ;

        p->dfds[1] = 0.5 * pr * r ;
        p->dfds[4] = -0.5 * r * mr ;
        p->dfds[0] = -0.5 * r * pr ;
        p->dfds[2] = pmr ;
        p->dfds[5] = 0.5 * r * mr ;
        p->dfds[3] = -pmr ;

        return(0) ;
}
```

festuff2.c

```
/*	the shape functions for the cubic upwinding functions   239	*/
#include "fe.h"


extern struct control  N ;



int	shcu2_2x1(x,p)
struct gausspts		*x;
struct shapefuncs	*p;

{
	double r, s, t, rm, sm, rp, sp, rp3;

	r = x->x ;
	s = x->y ;
	if ( (r<-3.00001) || (r>1.00001) || (s<-1.00001) || (s>1.00001) )
return(6) ;

	t = 0.125 ;
	rm = 1.0 - r ;
	rp = 1.0 + r ;
	sp = 1.0 + s ;
	sm = 1.0 - s ;
	rp3 = 3.0 + r ;

	p->dof = 6 ;

	p->f[3] = t * rp3 * sm * rm ;
	p->f[0] = t / 2.0 * rp3 * rp * sm ;
	p->f[1] = t / 2.0 * rp * sp * rp3 ;
	p->f[2] = t * rm * sp * rp3 ;
	p->f[4] = t / -2.0 * rp * sp * rm ;
	p->f[5] = t / -2.0 * rp * sm * rm ;

	if (x->w < 0.0)
		return(0) ;

	p->dfdr[3] = -2.0 * t * sm * rp ;
	p->dfdr[0] = t * sm * ( r + 2.0 ) ;
	p->dfdr[1] = t * sp * ( r + 2.0 ) ;
	p->dfdr[2] = -2.0 * t * sp * rp ;
	p->dfdr[4] = t * sp * r ;
	p->dfdr[5] = t * sm * r ;

	p->dfds[3] = -t * rp3 * rm ;
	p->dfds[0] = -t / 2.0 * rp3 * rp ;
	p->dfds[1] = t / 2.0 * rp3 * rp ;
	p->dfds[2] = t * rp3 * rm ;
	p->dfds[4] = -t / 2.0 * rp * rm ;
	p->dfds[5] = t / 2.0 * rp * rm ;

	return (0);
}


int	shcu3_2x2(x,p)
struct gausspts		*x;
struct shapefuncs	*p;

{
	double r, s, t, rm, sm, rp, sp, rp3, sp3;
```

```
        r = x->x ;
        s = x->y ;
        if ( (r<-3.00001) || (r>1.00001) || (s<-3.00001) || (s>1.00001) )
return(8) ;

        t = 0.0625 ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
        sp = 1.0 + s ;
        sm = 1.0 - s ;
        rp3 = 3.0 + r ;
        sp3 = 3.0 + s ;

        p->dof = 8 ;

        p->f[3] = t * rp3 * sm * rm * sp3 ;
        p->f[0] = t / 2.0 * rp3 * rp * sm * sp3 ;
        p->f[1] = t / 4.0 * rp * sp * rp3 * sp3 ;
        p->f[2] = t / 2.0 * rm * sp * rp3 * sp3 ;
        p->f[4] = -t * rp * sp * rm ;
        p->f[5] = -t * rp * sm * rm ;
        p->f[6] = -t * sm * rm * sp ;
        p->f[7] = -t * rp * sm * sp ;


        if (x->w < 0.0)
                return(0) ;

        p->dfdr[3] = -2.0 * t * sm * rp * sp3;
        p->dfdr[0] =     * sm * ( r + 2.0 ) * sp3;
        p->dfdr[1] = t / 2.0 * sp * ( r + 2.0 ) * sp3;
        p->dfdr[2] = -t * sp * rp * sp3;
        p->dfdr[4] = 2. * t * sp * r ;
        p->dfdr[5] = 2. * t * sm * r ;
        p->dfdr[6] = t * sm * sp;
        p->dfdr[7] = -t * sm * sp;


        p->dfds[3] = -2.0 * t * sp * rp3 * rm ;
        p->dfds[0] = -t * sp * rp3 * rp ;
        p->dfds[1] = t / 2.0 * ( s + 2.0 ) * rp3 * rp ;
        p->dfds[2] = t    ( s + 2.0 ) * rp3 * rm ;
        p->dfds[4] = -t * rp * rm ;
        p->dfds[5] = t * rp * rm ;
        p->dfds[6] = t * 2.0 * s * rm ;
        p->dfds[7] = t * 2.0 * s * rp ;

        return (0);
}

int     shcu4_2x2(x,p)
struct gausspts      *x;
struct shapefuncs    *p;

{
        double r, s, t, rm, sm, rp, sp, rp3, sp3;

        r = x->x ;
        s = x->y ;
        if ( (r<-3.00001) || (r>1.00001) || (s<-3.00001) || (s>1.00001) )
return(9) ;
```

```
        t = 0.0625 ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
        sp = 1.0 + s ;
        sm = 1.0 - s ;
        rp3 = 3.0 + r ;
        sp3 = 3.0 + s ;

        p->dof = 9 ;

        p->f[3] = t * rp3 * sm * rm * sp3 ;
        p->f[0] = t / 2.0 * rp3 * rp * sm * sp3 ;
        p->f[1] = t / 4.0 * rp * sp * rp3 * sp3 ;
        p->f[2] = t / 2.0 * rm * sp * rp3 * sp3 ;
        p->f[4] = t / -4.0 * rp * sp * rm * sp3 ;
        p->f[5] = t / -2.0 * rp * sm * rm * sp3 ;
        p->f[7] = t / -2.0 * rp3 * sm * rm * sp ;
        p->f[8] = t / -4.0 * rp * sm * rp3 * sp ;
        p->f[6] = t / 4.0 * rp * sm * rm * sp ;


        if (x->w < 0.0)
                return(0) ;

        p->dfdr[3] = -2.0 * t * sm * rp * sp3;
        p->dfdr[0] = t * sm * ( r + 2.0 ) * sp3;
        p->dfdr[1] = t / 2.0 * sp * ( r + 2.0 ) * sp3;
        p->dfdr[2] = -t * sp * rp * sp3;
        p->dfdr[4] = t / 2.0 * sp * r * sp3;
        p->dfdr[5] = t * sm * r * sp3;
        p->dfdr[7] = t * sm * rp * sp;
        p->dfdr[8] = -t / 2.0 * sm * ( r + 2.0 ) * sp;
        p->dfdr[6] = -t / 2.0 * sm * r * sp;


        p->dfds[3] = -2.0 * t * sp * rp3 * rm ;
        p->dfds[0] = -t * sp * rp3 * rp ;
        p->dfds[1] = t / 2.0 * ( s + 2.0 ) * rp3 * rp ;
        p->dfds[2] = t * ( s + 2.0 ) * rp3 * rm ;
        p->dfds[4] = -t / 2.0 * ( s + 2.0 ) * rp * rm ;
        p->dfds[5] = t * sp * rp * rm ;
        p->dfds[7] = t * s * rp3 * rm ;
        p->dfds[8] = t / 2.0 * s * rp * rp3 ;
        p->dfds[6] = -t / 2.0 * s * rp * rm ;

        return (0);
}

int     shcu3_3x1(x,p)
struct gausspts       *x;
struct shapefuncs     *p;

{
        double r, s, t, rm, sm, rp, sp, rp3, rp5, r2p8 ;

        r = x->x ;
        s = x->y ;
        if ( (r<-5.00001) || (r>1.00001) || (s<-1.00001) || (s>1.00001) )
return(8) ;

        t = 1.0 / 32. ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
```

```
        sp = 1.0 + s ;
        sm = 1.0 - s ;
        rp3 = 3.0 + r ;
        rp5 = 5.0 + r ;

        p->dof = 8 ;

        p->f[3] = t * rp3 * sm * rm * rp5 ;
        p->f[0] = t / 3.0 * rp3 * rp * sm * rp5 ;
        p->f[1] = t / 3.0 * rp * sp * rp3 * rp5 ;
        p->f[2] = t * rm * sp * rp3 * rp5 ;
        p->f[4] = -t * rp * sp * rm * rp5 ;
        p->f[5] = t / 3.0 * rp * sp * rm * rp3 ;
        p->f[6] = t / 3.0 * rp * sm * rm * rp3 ;
        p->f[7] = -t * rp * sm * rm * rp5 ;

        if (x->w < 0.0)
              return(0) ;

        r2p8 = 2.0 * r + 8.0 ;

        p->dfdr[3] = t * sm * ( (-rp5 * rp3) + rm * r2p8 ) ;
        p->dfdr[0] = t / 3.0 * sm * ( ( rp5 * rp3) + rp * r2p8 ) ;
        p->dfdr[1] = t / 3.0 * sp * ( ( rp5 * rp3) + rp * r2p8 ) ;
        p->dfdr[2] = t * sp * ( (-rp5 * rp3) + rm * r2p8 ) ;
        p->dfdr[4] = -t * sp * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[5] = t / 3.0 * sp * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[6] = t / 3.0 * sm * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[7] = -t * sm * ( -2.0 * r * rp5 + rp * rm ) ;

        p->dfds[3] = -t * rp3 * rm * rp5 ;
        p->dfds[0] = -t / 3.0 * rp3 * rp * rp5 ;
        p->dfds[1] = t / 3.0 * rp * rp3 * rp5 ;
        p->dfds[2] = t * rm * rp3 * rp5 ;
        p->dfds[4] = -t * rp * rm * rp5 ;
        p->dfds[5] = t / 3.0 * rp * rm * rp3 ;
        p->dfds[6] = -t / 3.0 * rp * rm * rp3 ;
        p->dfds[7] = t * rp * rm * rp5 ;

        return (0);
}

int     shcu4_3x2(x,p)
struct  gausspts        *x;
struct  shapefuncs      *p;


        double r, s, t, rm, sm, rp, sp, rp3, rp5, r2p8, sp3 ;

        r = x->x ;
        s = x->y ;
        if ( (r<-5.00001) || (r>1.00001) || (s<-3.00001) || (s>1.00001) )
return(10) ;

        t = 1.0 / 64. ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
        sp = 1.0 + s ;
        sm = 1.0 - s ;
        rp3 = 3.0 + r ;
        rp5 = 5.0 + r ;
        sp3 = 3.0 + s ;

```

```
        p->dof = 10 ;

        p->f[3] = t * rp3 * sm * rm * rp5 * sp3 ;
        p->f[0] = t / 3.0 * rp3 * rp * sm * rp5 * sp3 ;
        p->f[1] = t / 6.0 * rp * sp * rp3 * rp5 * sp3 ;
        p->f[2] = t / 2.0 * rm * sp * rp3 * rp5 * sp3 ;
        p->f[4] = -2.0 * t * rp * sp * rm * rp5 ;
        p->f[5] = 2.0 * t / 3.0 * rp * sp * rm * rp3 ;
        p->f[6] = 2.0 * t / 3.0 * rp * sm * rm * rp3 ;
        p->f[7] = -2.0 * t * rp * sm * rm * rp5 ;
        p->f[8] = -4.0 * t * sp * rm * sm ;
        p->f[9] = -4.0 * t * sp * rp * sm ;

        if (x->w < 0.0)
                return(0) ;

        r2p8 = 2.0 * r + 8.0 ;

        p->dfdr[3] = t * sm * sp3 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfdr[0] = t / 3.0 * sm * sp3 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[1] = t / 6.0 * sp * sp3 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[2] = t / 2.0 * sp * sp3 * ( (-rp5 * rp3) + rm * r2p8 ) ;
        p->dfdr[4] = -2.0 * t * sp * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[5] = 2.0 * t / 3.0 * sp * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[6] = 2.0 * t / 3.0 * sm * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[7] = -2.0 * t * sm * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[8] = 4.0 * t * sm * sp ;
        p->dfdr[9] = -4.0 * t * sm * sp ;

        p->dfds[3] = -t * 2.0 * rp3 * sp * rm * rp5 ;
        p->dfds[0] = -t * 2.0 / 3.0 * rp3 * rp * sp * rp5 ;
        p->dfds[1] = t / 3.0 * rp * rp3 * rp5 * ( s + 2.0 ) ;
        p->dfds[2] = t * rm * rp3 * rp5 * ( s + 2.0 ) ;
        p->dfds[4] = -2.0 * t * rp * rm * rp5 ;
        p->dfds[5] = 2.0 * t / 3.0 * rp * rm * rp3 ;
        p->dfds[6] = -2.0 * t / 3.0 * rp * rm * rp3 ;
        p->dfds[7] = 2.0 * t * rp * rm * rp5 ;
        p->dfds[8] = 8.0 * t * s * rm ;
        p->dfds[9] = 8.0 * t * s * rp ;

        return (0);
}

int     sh4_3x2b(x,p)
struct gausspts       *x;
struct shapefuncs     *p;

{
        double r, s, t, rm, sm, rp, sp, rp3, rp5, r2p8, sp3 ;

        r = x->y ;
        s = x->x ;
        if ( (r<-5.00001) || (r>1.00001) || (s<-3.00001) || (s>1.00001) )
return(10) ;

        t = 1.0 / 64. ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
        sp = 1.0 + s ;
        sm = 1.0 - s ;
        rp3 = 3.0 + r ;
        rp5 = 5.0 + r ;
        sp3 = 3.0 + s ;
```

```
        p->dof = 10 ;

        p->f[3] = t * rp3 * sm * rm * rp5 * sp3 ;
        p->f[2] = t / 3.0 * rp3 * rp * sm * rp5 * sp3 ;
        p->f[1] = t / 6.0 * rp * sp * rp3 * rp5 * sp3 ;
        p->f[0] = t / 2.0 * rm * sp * rp3 * rp5 * sp3 ;
        p->f[4] = -2.0 * t * rp * sp * rm * rp5 ;
        p->f[5] = 2.0 * t / 3.0 * rp * sp * rm * rp3 ;
        p->f[6] = 2.0 * t / 3.0 * rp * sm * rm * rp3 ;
        p->f[7] = -2.0 * t * rp * sm * rm * rp5 ;
        p->f[8] = -4.0 * t * sp * rm * sm ;
        p->f[9] = -4.0 * t * sp * rp * sm ;

        if (x->w < 0.0)
                return(0);

        r2p8 = 2.0 * r + 8.0 ;

        p->dfds[3] = t * sm * sp3 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfds[2] = t / 3.0 * sm * sp3 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfds[1] = t / 6.0 * sp * sp3 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfds[0] = t / 2.0 * sp * sp3 * ( (-rp5 * rp3) + rm * r2p8 ) ;
        p->dfds[4] = -2.0 * t * sp * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfds[5] = 2.0 * t / 3.0 * sp * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfds[6] = 2.0 * t / 3.0 * sm * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfds[7] = -2.0 * t * sm * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfds[8] = 4.0 * t * sm * sp ;
        p->dfds[9] = -4.0 * t * sm * sp ;

        p->dfdr[3] = -t * 2.0 * rp3 * sp * rm * rp5 ;
        p->dfdr[2] = -t * 2.0 / 3.0 * rp3 * rp * sp * rp5 ;
        p->dfdr[1] = t / 3.0 * rp * rp3 * rp5 * ( s + 2.0 ) ;
        p->dfdr[0] = t * rm * rp3 * rp5 * ( s + 2.0 ) ;
        p->dfdr[4] = -2.0 * t * rp * rm * rp5 ;
        p->dfdr[5] = 2.0 * t / 3.0 * rp * rm * rp3 ;
        p->dfdr[6] = -2.0 * t / 3.0 * rp * rm * rp3 ;
        p->dfdr[7] = 2.0 * t * rp * rm * rp5 ;
        p->dfdr[8] = 8.0 * t * s * rm ;
        p->dfdr[9] = 8.0 * t * s * rp ;

        return (0);
}

int     shcu5_3x2(x,p)
struct gausspts      *x;
struct shapefuncs    *p;

{
        double r, s, t, rm, sm, rp, sp, rp3, rp5, r2p8, sp3 ;

        r = x->x ;
        s = x->y ;
        if ( (r<-5.00001) || (r>1.00001) || (s<-3.00001) || (s>1.00001) )
return(11) ;

        t = 1.0 / 64. ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
        sp = 1.0 + s ;
        sm = 1.0 - s ;
        rp3 = 3.0 + r ;
        rp5 = 5.0 + r ;
```

```
        sp3 = 3.0 + s ;

        p->dof = 11 ;

        p->f[3] = t * rp3 * sm * rm * rp5 * sp3 ;
        p->f[0] = t / 3.0 * rp3 * rp * sm * rp5 * sp3 ;
        p->f[1] = t / 6.0 * rp * sp * rp3 * rp5 * sp3 ;
        p->f[2] = t / 2.0 * rm * sp * rp3 * rp5 * sp3 ;
        p->f[4] = -t / 2.0 * rp * sp * rm * rp5 * sp3 ;
        p->f[5] = 2.0 * t / 3.0 * rp * sp * rm * rp3 ;
        p->f[6] = 2.0 * t / 3.0 * rp * sm * rm * rp3 ;
        p->f[7] = -t * rp * sm * rm * rp5 * sp3 ;
        p->f[8] = t * rp * sp * rm * sm ;
        p->f[9] = -2.0 * t * rp3 * sp * rm * sm ;
        p->f[10] = -t * rp3 * sp * rp * sm ;

        if (x->w < 0.0)
                return(0) ;

        r2p8 = 2.0 * r + 8.0 ;

        p->dfdr[3] = t * sm * sp3 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfdr[0] = t / 3.0 * sm * sp3 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[1] = t / 6.0 * sp * sp3 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[2] = t / 2.0 * sp * sp3 * ( (-rp5 * rp3) + rm * r2p8 ) ;
        p->dfdr[4] = -t / 2.0 * sp * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[5] = 2.0 * t / 3.0 * sp * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[6] = 2.0 * t / 3.0 * sm * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[7] = -t * sm * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[8] = t * sp * sm * ( -2.0 * r ) ;
        p->dfdr[9] = -t * 2.0 * sm * sp * ( -2.0 * rp ) ;
        p->dfdr[10] = -t * sp * sm * ( 2.0 * r + 4.0 ) ;

        p->dfds[3] = -t * 2.0 * rp3 * sp * rm * rp5 ;
        p->dfds[0] = -t * 2.0 / 3.0 * rp3 * rp * sp * rp5 ;
        p->dfds[1] = t / 3.0 * rp * rp3 * rp5 * ( s + 2.0 ) ;
        p->dfds[2] = t * rm * rp3 * rp5 * ( s + 2.0 ) ;
        p->dfds[4] = -t * rp * rm * rp5 * ( s + 2.0 ) ;
        p->dfds[5] = 2.0 * t / 3.0 * rp * rm * rp3 ;
        p->dfds[6] = -2.0 * t / 3.0 * rp * rm * rp3 ;
        p->dfds[7] = 2.0 * t * rp * rm * rp5 * sp ;
        p->dfds[8] = -2.0 * t * rp * s * rm ;
        p->dfds[9] = 4.0 * t * rp3 * s * rm ;
        p->dfds[10] = 2.0 * t * rp3 * s * rp ;

        return (0);
}

int     sh5_3x2b(x,p)
struct gausspts       *x;
struct shapefuncs     *p;

{
        double r, s, t, rm, sm, rp, sp, rp3, rp5, r2p8, sp3 ;

        r = x->y ;
        s = x->x ;
        if ( (r<-5.00001) || (r>1.00001) || (s<-3.00001) || (s>1.00001) )
return(11) ;

        t = 1.0 / 64. ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
```

```
        sp = 1.0 + s ;
        sm = 1.0 - s ;
        rp3 = 3.0 + r ;
        rp5 = 5.0 + r ;
        sp3 = 3.0 + s ;

        p->dof = 11 ;

        p->f[3] = t * rp3 * sm * rm * rp5 * sp3 ;
        p->f[2] = t / 3.0 * rp3 * rp * sm * rp5 * sp3 ;
        p->f[1] = t / 6.0 * rp * sp * rp3 * rp5 * sp3 ;
        p->f[0] = t / 2.0 * rm * sp * rp3 * rp5 * sp3 ;
        p->f[4] = -t / 2.0 * rp * sp * rm * rp5 * sp3 ;
        p->f[5] = 2.0 * t / 3.0 * rp * sp * rm * rp3 ;
        p->f[6] = 2.0 * t / 3.0 * rp * sm * rm * rp3 ;
        p->f[7] = -t * rp * sm * rm * rp5 * sp3 ;
        p->f[8] = t * rp * sp * rm * sm ;
        p->f[9] = -2.0 * t * rp3 * sp * rm * sm ;
        p->f[10] = -t * rp3 * sp * rp * sm ;

        if (x->w < 0.0)
              return(0) ;

        r2p8 = 2.0 * r + 8.0 ;

        p->dfds[3] = t * sm * sp3 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfds[2] = t / 3.0 * sm * sp3 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfds[1] = t / 6.0 * sp * sp3 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfds[0] = t / 2.0 * sp * sp3 * ( (-rp5 * rp3) + rm * r2p8 ) ;
        p->dfds[4] = -t / 2.0 * sp * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfds[5] = 2.0 * t / 3.0 * sp * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfds[6] = 2.0 * t / 3.0 * sm * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfds[7] = -t * sm * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfds[8] = t * sp * sm * ( -2.0 * r ) ;
        p->dfds[9] = -t * 2.0 * sm * sp * ( -2.0 * rp ) ;
        p->dfds[10] = -t * sp * sm * ( 2.0 * r + 4.0 ) ;

        p->dfdr[3] = -t * 2.0 * rp3 * sp * rm * rp5 ;
        p->dfdr[2] = -t * 2.0 / 3.0 * rp3 * rp * sp * rp5 ;
        p->dfdr[1] = t / 3.0 * rp * rp3 * rp5 * ( s + 2.0 ) ;
        p->dfdr[0] = t * rm * rp3 * rp5 * ( s + 2.0 ) ;
        p->dfdr[4] = -t * rp * rm * rp5 * ( s + 2.0 ) ;
        p->dfdr[5] = 2.0 * t / 3.0 * rp * rm * rp3 ;
        p->dfdr[6] = -2.0 * t / 3.0 * rp * rm * rp3 ;
        p->dfdr[7] = 2.0 * t * rp * rm * rp5 * sp ;
        p->dfdr[8] = -2.0 * t * rp * s * rm ;
        p->dfdr[9] = 4.0 * t * rp3 * s * rm ;
        p->dfdr[10] = 2.0 * t * rp3 * s * rp ;

        return (0);
}
```

```
/*      the shape functions for the cubic upwinding functions   239
#include "fe.h"


extern struct control  N ;

int     shcu6_3x2(x,p)
struct gausspts      *x;
struct shapefuncs    *p;

{
        double r, s, t, rm, sm, rp, sp, rp3, rp5, r2p8, sp3 ;

        r = x->x ;
        s = x->y ;
        if ( (r<-5.00001) || (r>1.00001) || (s<-3.00001) || (s>1.00001) )
return(12) ;

        t = 1.0 / 64. ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
        sp = 1.0 + s ;
        sm = 1.0 - s ;
        rp3 = 3.0 + r ;
        rp5 = 5.0 + r ;
        sp3 = 3.0 + s ;

        p->dof = 12 ;

        p->f[3] = t * rp3 * sm * rm * rp5 * sp3 ;
        p->f[0] = t / 3.0 * rp3 * rp * sm * rp5 * sp3 ;
        p->f[1] = t / 6.0 * rp * sp * rp3 * rp5 * sp3 ;
        p->f[2] = t / 2.0 * rm * sp * rp3 * rp5 * sp3 ;
        p->f[4] = -t / 2.0 * rp * sp * rm * rp5 * sp3 ;
        p->f[5] = t / 6.0 * rp * sp * rm * rp3 * sp3;
        p->f[6] = t / 3.0 * rp * sm * rm * rp3 * sp3;
        p->f[7] = -t * rp * sm * rm * rp5 * sp3;
        p->f[8] = -t / 6.0 * rp3 * sp * rp * rm * sm ;
        p->f[9] = t / 2.0 * rp * sp * rm * rp5 * sm ;
        p->f[10] = -t / 2.0 * rp3 * sp * rm * rp5 * sm ;
        p->f[11] = -t / 6.0 * rp3 * sp * rp * rp5 * sm ;

        if (x->w < 0.0)
              return(0) ;

        r2p8 = 2.0 * r + 8.0 ;

        p->dfdr[3] = t * sm * sp3 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfdr[0] = t / 3.0 * sm * sp3 * ( (-rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[1] = t / 6.0 * sp * sp3 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[2] = t / 2.0 * sp * sp3 * ( (-rp5 * rp3) + rm * r2p8 ) ;
        p->dfdr[4] = -t / 2.0 * sp * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[5] = t / 6.0 * sp * sp3 * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[6] = t / 3.0 * sm * sp3 * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[7] = -t * sm * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[8] = -t / 6.0 * sp * sm * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[9] = t / 2.0 * sp * sm * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[10] = -t / 2.0 * sm * sp * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfdr[11] = -t / 6.0 * sp * sm * ( ( rp5 * rp3 ) + rp * r2p8 ) ;

        p->dfds[3] = -t * 2.0 * rp3 * sp * rm * rp5 ;
        p->dfds[0] = -t * 2.0 / 3.0 * rp3 * rp * sp * rp5 ;
        p->dfds[1] = t / 3.0 * rp * rp3 * rp5 * ( s + 2.0 ) ;
```

```
        p->dfds[2] = t * rm * rp3 * rp5 * ( s + 2.0 ) ;
        p->dfds[4] = -t * rp * rm * rp5 * ( s + 2.0 ) ;
        p->dfds[5] = t / 3.0 * rp * rm * rp3 * ( s + 2.0 ) ;
        p->dfds[6] = -2.0 * t / 3.0 * rp * sp * rm * rp3 ;
        p->dfds[7] = 2.0 * t * rp * rm * rp5 * sp ;
        p->dfds[8] = t / 3.0 * rp * s * rm * rp3 ;
        p->dfds[9] = -t * rp * s * rm * rp5 ;
        p->dfds[10] = t * rp3 * s * rm * rp5 ;
        p->dfds[11] = t / 3.0 * rp3 * s * rp * rp5 ;

        return (0);
}
int     sh6_3x2b(x,p)
struct gausspts      *x;
struct shapefuncs    *p;

{
        double r, s, t, rm, sm, rp, sp, rp3, rp5, r2p8, sp3 ;

        r = x->y ;
        s = x->x ;
        if ( (r<-5.00001) || (r>1.00001) || (s<-3.00001) || (s>1.00001) )
return(2) ;

        t = 1.0 / 64. ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
        sp = 1.0 + s ;
        sm = 1.0 - s ;
        rp3 = 3.0 + r ;
        rp5 = 5.0 + r ;
        sp3 = 3.0 + s ;

        p->dof = 12 ;

        p->f[3] = t * rp3 * sm * rm * rp5 * sp3 ;
        p->f[2] = t / 3.0 * rp3 * rp * sm * rp5 * sp3 ;
        p->f[1] = t / 6.0 * rp * sp * rp3 * rp5 * sp3 ;
        p->f[0] = t / 2.0 * rm * sp * rp3 * rp5 * sp3 ;
        p->f[4] = -t / 2.0 * rp * sp * rm * rp5 * sp3 ;
        p->f[5] = t / 6.0 * rp * sp * rm * rp3 * sp3;
        p->f[6] = t / 3.0 * rp * sm * rm * rp3 * sp3;
        p->f[7] = -t * rp * sm * rm * rp5 * sp3;
        p->f[8] = -t / 6.0 * rp3 * sp * rp * rm * sm ;
        p->f[9] = t / 2.0 * rp * sp * rm * rp5 * sm ;
        p->f[10] = -t / 2.0 * rp3 * sp * rm * rp5 * sm ;
        p->f[11] = -t / 6.0 * rp3 * sp * rp * rp5 * sm ;

        if (x->w < 0.0)
            return(0) ;

        r2p8 = 2.0 * r + 8.0 ;

        p->dfds[3] = t * sm * sp3 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfds[2] = t / 3.0 * sm * sp3 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfds[1] = t / 6.0 * sp * sp3 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfds[0] = t / 2.0 * sp * sp3 * ( (-rp5 * rp3) + rm * r2p8 ) ;
        p->dfds[4] = -t / 2.0 * sp * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfds[5] = t / 6.0 * sp * sp3 * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfds[6] = t / 3.0 * sm * sp3 * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfds[7] = -t * sm * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfds[8] = -t / 6.0 * sp * sm * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfds[9] = t / 2.0 * sp * sm * ( -2.0 * r * rp5 + rp * rm ) ;
```

```
        p->dfds[10] = -t / 2.0 * sm * sp * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfds[11] = -t / 6.0 * sp * sm * ( (-rp5 * rp3 ) + rp * r2p8 ) ;

        p->dfdr[3] = -t * 2.0 * rp3 * sp * rm * rp5 ;
        p->dfdr[2] = -t * 2.0 / 3.0 * rp3 * rp * sp * rp5 ;
        p->dfdr[1] = t / 3.0 * rp * rp3 * rp5 * ( s + 2.0 ) ;
        p->dfdr[0] = t * rm * rp3 * rp5 * ( s + 2.0 ) ;
        p->dfdr[4] = -t * rp * rm * rp5 * ( s + 2.0 ) ;
        p->dfdr[5] = t / 3.0 * rp * rm * rp3 * ( s + 2.0 ) ;
        p->dfdr[6] = -2.0 * t / 3.0 * rp * sp * rm * rp3 ;
        p->dfdr[7] = 2.0 * t * rp * rm * rp5 * sp ;
        p->dfdr[8] = t / 3.0 * rp * s * rm * rp3 ;
        p->dfdr[9] = -t * rp * s * rm * rp5 ;
        p->dfdr[10] = t * rp3 * s * rm * rp5 ;
        p->dfdr[11] = t / 3.0 * rp3 * s * rp * rp5 ;

        return (0);
}

int     shcu5_3x3(x,p)
struct gausspts       *x;
struct shapefuncs     *p;

{
        double r, s, t, rm, sm, rp, sp, rp3, rp5, r2p8, sp3, sp5, s2p8 ;

        r = x->x ;
        s = x->y ;
        if ( (r<-5.00001) || (r>1.00001) || (s<-5.00001) || (s>1.00001) )
return(12) ;

        t = 1.0 / 256. ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
        sp = 1.0 + s ;
        sm = 1.0 - s ;
        rp3 = 3.0 + r ;
        rp5 = 5.0 + r ;
        sp3 = 3.0 + s ;
        sp5 = 5.0 + s ;

        p->dof = 12 ;

        p->f[3] = t * rp3 * sm * rm * rp5 * sp3 * sp5 ;
        p->f[0] = t / 3.0 * rp3 * rp * sm * rp5 * sp3 * sp5 ;
        p->f[1] = t / 9.0 * rp * sp * rp3 * rp5 * sp3 * sp5 ;
        p->f[2] = t / 3.0 * rm * sp * rp3 * rp5 * sp3 * sp5 ;
        p->f[4] = -8.0 * t * rp * sp * rm * rp5 ;
        p->f[5] = 8.0 * t / 3.0 * rp * sp * rm * rp3  ;
        p->f[6] = 8.0 * t / 3.0 * rp * sm * rm * rp3  ;
        p->f[7] = -8.0 * t * rp * sm * rm * rp5  ;
        p->f[8] = -8.0 * t * sp * rm * sm * sp5 ;
        p->f[11] = -8.0 * t * sp * rp * sm * sp5 ;
        p->f[9] = 8.0 * t / 3.0 * sp * rm * sm * sp3 ;
        p->f[10] = 8.0 * t / 3.0 * sp * rp * sm * sp3 ;

        if (x->w < 0.0)
              return(0) ;

        r2p8 = 2.0 * r + 8.0 ;
        s2p8 = 2.0 * s + 8.0 ;

```

```
        p->dfdr[3] = t * sm * sp3 * sp5 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfdr[0] = t / 3.0 * sm * sp3 * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[1] = t / 9.0 * sp * sp3 * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[2] = t / 3.0 * sp * sp3 * sp5 * ( (-rp5 * rp3) + rm * r2p8 ) ;
        p->dfdr[4] = -8.0 * t * sp * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[5] = 8.0 * t / 3.0 * sp * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[6] = 8.0 * t / 3.0 * sm * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[7] = -8.0 * t * sm * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[8] = 8.0 * t * sm * sp * sp5 ;
        p->dfdr[11] = -8.0 * t * sp * sm * sp5 ;
        p->dfdr[10] = 8.0 * t / 3.0 * sp * sm * sp3 ;
        p->dfdr[9] = -8.0 * t / 3.0 * sm * sp * sp3 ;

        p->dfds[3] = t * rp3 * rm * rp5 * ( -( sp5 * sp3 ) + sm * s2p8 ) ;
        p->dfds[0] = t / 3.0 * rp3 * rp * rp5 * ( -( sp5 * sp3 ) + sm * s2p8 ) ;
        p->dfds[1] = t / 9.0 * rp * rp3 * rp5 * ( ( sp5 * sp3 ) + sp * s2p8 ) ;
        p->dfds[2] = t / 3.0 * rm * rp3 * rp5 * ( ( sp5 * sp3 ) + sp * s2p8 ) ;
        p->dfds[4] = -8.0 * t * rp * rm * rp5 ;
        p->dfds[5] = 8.0 * t / 3.0 * rp * rm * rp3 ;
        p->dfds[6] = -8.0 * t / 3.0 * rp * rm * rp3 ;
        p->dfds[7] = 8.0 * t * rp * rm * rp5 ;
        p->dfds[8] = -8.0 * t * rm * ( -2.0 * s * sp5 + sp * sm ) ;
        p->dfds[11] = -8.0 * t * rp * ( -2.0 * s * sp5 + sp * sm ) ;
        p->dfds[9] = 8.0 * t / 3.0 * rm * ( -2.0 * s * sp3 + sp * sm ) ;
        p->dfds[10] = 8.0 * t / 3.0 * rp * ( -2.0 * s * sp3 + sp * sm ) ;

        return (0);
}

int     shcu6_3x3(x,p)
struct gausspts     *x;
struct shapefuncs   *p;

{
        double r, s, t, rm, sm, rp, sp, rp3, rp5, r2p8, sp3, sp5, s2p8 ;

        r = x->x ;
        s = x->y ;
        if ( (r<-5.00001) || (r>1.00001) || (s<-5.00001) || (s>1.00001) )
return(13) ;

        t = 1.0 / 256. ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
        sp = 1.0 + s ;
        sm = 1.0 - s ;
        rp3 = 3.0 + r ;
        rp5 = 5.0 + r ;
        sp3 = 3.0 + s ;
        sp5 = 5.0 + s ;

        p->dof = 13 ;

        p->f[3] = t * rp3 * sm * rm * rp5 * sp3 * sp5 ;
        p->f[0] = t / 3.0 * rp3 * rp * sm * rp5 * sp3 * sp5 ;
        p->f[1] = t / 9.0 * rp * sp * rp3 * rp5 * sp3 * sp5 ;
        p->f[2] = t / 3.0 * rm * sp * rp3 * rp5 * sp3 * sp5 ;
        p->f[4] = -2.0 * t * rp * sp * rm * rp5 * sp3 ;
        p->f[5] = 8.0 * t / 3.0 * rp * sp * rm * rp3 ;
        p->f[6] = 8.0 * t / 3.0 * rp * sm * rm * rp3 ;
        p->f[7] = -4.0 * t * rp * sm * rm * rp5 * sp3 ;
        p->f[8] = 4.0 * t * rp * sp * rm * sm ;
        p->f[9] = -4.0 * t * rp3 * sp * rm * sm * sp5 ;
```

```
        p->f[10] = 8.0 * t / 3.0 * sp * rm * sm * sp3 ;
        p->f[11] = 8.0 * t / 3.0 * sp * rp * sm * sp3 ;
        p->f[12] = -2.0 * t * rp3 * sp * rp   sm * sp5 ;

        if (x->w < 0.0)
              return(0) ;

        r2p8 = 2.0 * r + 8.0 ;
        s2p8 = 2.0 * s + 8.0 ;


        p->dfdr[3] = t * sm * sp3 * sp5 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfdr[0] = t / 3.0 * sm * sp3 * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[1] = t / 9.0 * sp * sp3 * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[2] = t / 3.0 * sp * sp3 * sp5 * ( (-rp5 * rp3) + rm * r2p8 ) ;
        p->dfdr[4] = -2.0 * t * sp * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[5] = 8.0 * t / 3.0 * sp * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[6] = 8.0 * t / 3.0 * sm * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[7] = -4.0 * t * sm * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[8] = 4.0 * t * sm * sp * ( -2.0 * r ) ;
        p->dfdr[9] = -4.0 * t * sm * sp * sp5 * (-2.0 * rp ) ;
        p->dfdr[10] = -8.0 * t / 3.0 * sm * sp * sp3 ;
        p->dfdr[11] = 8.0 * t / 3.0 * sp * sm * sp3  ;
        p->dfdr[12] = -4.0 * t * sp * sm * sp5 * ( r + 2.0 ) ;

        p->dfds[3] = t * rp3 * rm * rp5 * ( -( sp5 * sp3 ) + sm * s2p8 ) ;
        p->dfds[0] = t / 3.0 * rp3 * rp * rp5 * ( -( sp5 * sp3 ) + sm * s2p8 ) ;
        p->dfds[1] = t / 9.0 * rp * rp3 * rp5 * ( ( sp5 * sp3 ) + sp * s2p8 ) ;
        p->dfds[2] = t / 3.0 * rm * rp3 * rp5 * ( ( sp5 * sp3 ) + sp * s2p8 ) ;
        p->dfds[4] = -2.0 * t * rp * rm * rp5 * ( 2.0 * s + 4.0 ) ;
        p->dfds[5] = 8.0 * t / 3.0 * rp * rm * rp3 ;
        p->dfds[6] = -8.0 * t / 3.0 * rp * rm * rp3 ;
        p->dfds[7] = -4.0 * t * rp * rm * rp5 * ( -2.0 * sp ) ;
        p->dfds[8] = 4.0 * t * rp * rm * ( -2.0 * s ) ;
        p->dfds[9] = -4.0 * t * rp3 * rm * ( -2.0 * s * sp5 + sp * sm ) ;
        p->dfds[10] = 8.0 * t / 3.0 * rm * ( -2.0 * s * sp3 + sp * sm ) ;
        p->dfds[11] = 8.0 * t / 3.0 * rp * ( -2.0 * s * sp3 + sp * sm ) ;
        p->dfds[12] = -2.0 * t * rp3 * rp * ( -2.0 * s * sp5 + sp * sm ) ;

        return (0);
}
```

```
/*      the shape functions for the cubic upwinding functions   239           */
#include "fe.h"


extern struct control  N ;

int     shcu7_3x3(x,p)
struct gausspts      *x;
struct shapefuncs    *p;

{
        double r, s, t, rm, sm, rp, sp, rp3, rp5, r2p8, sp3, sp5, s2p8 ;

        r = x->x ;
        s = x->y ;
        if ( (r<-5.00001) || (r>1.00001) || (s<-5.00001) || (s>1.00001) )
return(14) ;

        t = 1.0 / 256. ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
        sp = 1.0 + s ;
        sm = 1.0 - s ;
        rp3 = 3.0 + r ;
        rp5 = 5.0 + r ;
        sp3 = 3.0 + s ;
        sp5 = 5.0 + s ;

        p->dof = 14 ;

        p->f[3] = t * rp3 * sm * rm * rp5 * sp3 * sp5 ;
        p->f[0] = t / 3.0 * rp3 * rp * sm * rp5 * sp3 * sp5 ;
        p->f[1] = t / 9.0 * rp * sp * rp3 * rp5 * sp3 * sp5 ;
        p->f[2] = t / 3.0 * rm * sp * rp3 * rp5 * sp3 * sp5 ;
        p->f[4] = -2.0 * t * rp * sp * rm * rp5 * sp3 ;
        p->f[5] = 2.0 * t / 3.0 * rp * sp * sp3 * rm * rp3 ;
        p->f[6] = 4.0 * t / 3.0 * rp * sm * sp3 * rm * rp3 ;
        p->f[7] = -4.0 * t * rp * sm * rm * rp5 * sp3 ;
        p->f[8] = -2.0 / 3.0 * t * rp * sp * rm * sm * rp3 ;
        p->f[9] = 2.0 * t * rp * sp * rm * sm * rp5 ;
        p->f[10] = -t * rp3 * sp * rm * rp5 * sm * sp5 ;
        p->f[11] = 8.0 * t / 3.0 * sp * rm * sm * sp3 ;
        p->f[12] = 8.0 * t / 3.0 * sp * rp * sm * sp3 ;
        p->f[13] = -t / 3.0 * rp3 * rp5 * sp * rp * sm * sp5 ;

        if (x->w < 0.0)
                return(0) ;

        r2p8 = 2.0 * r + 8.0 ;
        s2p8 = 2.0 * s + 8.0 ;

        p->dfdr[3] = t * sm * sp3 * sp5 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfdr[0] = t / 3.0 * sm * sp3 * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[1] = t / 9.0 * sp * sp3 * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[2] = t / 3.0 * sp * sp3 * sp5 * ( (-rp5 * rp3) + rm * r2p8 ) ;
        p->dfdr[4] = -2.0 * t * sp * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[5] = 2.0 * t / 3.0 * sp * sp3 * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[6] = 4.0 * t / 3.0 * sm * sp3 * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[7] = -4.0 * t * sm * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[8] = -2.0 / 3.0 * t * sp * sm * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[9] = 2.0 * t * sm * sp * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[10] = -t * sm * sp * sp5 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfdr[11] = -8.0 * t / 3.0 * sm * sp * sp3 ;
```

```
	p->dfdr[12] = 8.0 * t / 3.0 * sp * sm * sp3  ;
	p->dfdr[13] = -t / 3.0 * sp * sm * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;

	p->dfds[3] = t * rp3 * rm * rp5 * ( -( sp5 * sp3 ) + sm * s2p8 ) ;
	p->dfds[0] = t / 3.0 * rp3 * rp * rp5 * ( -( sp5 * sp3 ) + sm * s2p8 ) ;
	p->dfds[1] = t / 9.0 * rp * rp3 * rp5 * ( ( sp5 * sp3 ) + sp * s2p8 ) ;
	p->dfds[2] = t / 3.0 * rm * rp3 * rp5 * ( ( sp5 * sp3 ) + sp * s2p8 ) ;
	p->dfds[4] = -2.0 * t * rp * rm * rp5 * ( 2.0 * s + 4.0 ) ;
	p->dfds[5] = 2.0 * t / 3.0 * rp * rm * rp3 * ( 2.0 * s + 4.0 ) ;
	p->dfds[6] = 4.0 * t / 3.0 * rp * rm * rp3 * (-2.0 * sp );
	p->dfds[7] = -4.0 * t * rp * rm * rp5 * ( -2.0 * sp ) ;
	p->dfds[8] = -2.0 / 3.0 * t * rp * rm * rp3 * ( -2.0 * s) ;
	p->dfds[9] = 2.0 * t * rp * rm * rp5 * ( -2.0 * s ) ;
	p->dfds[10] = -t * rp3 * rm * rp5 * ( -2.0 * s * sp5 + sp * sm ) ;
	p->dfds[11] = 8.0 * t / 3.0 * rm * ( -2.0 * s * sp3 + sp * sm ) ;
	p->dfds[12] = 8.0 * t / 3.0 * rp * ( -2.0 * s * sp3 + sp * sm ) ;
	p->dfds[13] = -t / 3.0 * rp3 * rp * rp5 * ( -2.0 * s * sp5 + sp * sm ) ;

	return (0);
}

int	sh7_3x3b(x,p)
struct gausspts		*x;
struct shapefuncs	*p;

{
	double r, s, t, rm, sm, rp, sp, rp3, rp5, r2p8, sp3, sp5, s2p8 ;

	r = x->y ;
	s = x->x ;
	if ( (r<-5.00001) || (r>1.00001) || (s<-5.00001) || (s>1.00001) )
return(14) ;

	t = 1.0 / 256. ;
	rm = 1.0 - r ;
	rp = 1.0 + r ;
	sp = 1.0 + s ;
	sm = 1.0 - s ;
	rp3 = 3.0 + r ;
	rp5 = 5.0 + r ;
	sp3 = 3.0 + s ;
	sp5 = 5.0 + s ;

	p->dof = 14 ;

	p->f[3] = t * rp3 * sm * rm * rp5 * sp3 * sp5 ;
	p->f[2] = t / 3.0 * rp3 * rp * sm * rp5 * sp3 * sp5 ;
	p->f[1] = t / 9.0 * rp * sp * rp3 * rp5 * sp3 * sp5 ;
	p->f[0] = t / 3.0 * rm * sp * rp3 * rp5 * sp3 * sp5 ;
	p->f[4] = -2.0 * t * rp * sp * rm * rp5 * sp3 ;
	p->f[5] = 2.0 * t / 3.0 * rp * sp * sp3 * rm * rp3 ;
	p->f[6] = 4.0 * t / 3.0 * rp * sm * sp3 * rm * rp3 ;
	p->f[7] = -4.0 * t * rp * sm * rm * rp5 * sp3 ;
	p->f[8] = -2.0 / 3.0 * t * rp * sp * rm * sm * rp3 ;
	p->f[9] = 2.0 * t * rp * sp * rm * sm * rp5 ;
	p->f[10] = -t * rp3 * sp * rm * rp5 * sm * sp5 ;
	p->f[11] = 8.0 * t / 3.0 * sp * rm * sm * sp3 ;
	p->f[12] = 8.0 * t / 3.0 * sp * rp * sm * sp3 ;
	p->f[13] = -t / 3.0 * rp3 * rp5 * sp * rp * sm * sp5 ;

	if (x->w < 0.0)
		return(0) ;

```

```
        r2p8 = 2.0 * r + 8.0 ;
        s2p8 = 2.0 * s + 8.0 ;

        p->dfds[3] = t * sm * sp3 * sp5 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfds[2]   t / 3.0 * sm * sp3 * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfds[1] = t / 9.0 * sp * sp3 * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfds[0] = t / 3.0 * sp * sp3 * sp5 * ( (-rp5 * rp3) + rm * r2p8 ) ;
        p->dfds[4] = -2.0 * t * sp * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfds[5] = 2.0 * t / 3.0 * sp * sp3 * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfds[6] = 4.0 * t / 3.0 * sm * sp3 * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfds[7] = -4.0 * t * sm * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfds[8] = -2.0 / 3.0 * t * sp * sm * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfds[9] = 2.0 * t * sm * sp * ( -2.0 * r * rp5 + r  * rm ) ;
        p->dfds[10] = -t * sm * sp * sp5 * ( (-rp5 * rp3 ) +   * r2p8 ) ;
        p->dfds[11] = -8.0 * t / 3.0 * sm * sp * sp3 ;
        p->dfds[12] = 8.0 * t / 3.0 * sp * sm * sp3  ;
        p->dfds[13] = -t / 3.0 * sp * sm * sp5 * ( ( rp5 * rp   + rp * r2p8 ) ;

        p->dfdr[3] = t * rp3 * rm * rp5 * ( -( sp5 * sp3 ) +     s2p8 ) ;
        p->dfdr[2] = t / 3.0 * rp3 * rp * rp5 * ( -( sp5 * sp    + sm * s2p8 ) ;
        p->dfdr[1] = t / 9.0 * rp * rp3 * rp5 * ( ( sp5 * sp3   + sp * s2p8 ) ;
        p->dfdr[0] = t / 3.0 * rm * rp3 * rp5 * ( ( sp5 * sp3 ) + sp * s2p8 ) ;
        p->dfdr[4] = -2.0 * t * rp * rm * rp5 * ( 2.0 * s + 4.0 ) ;
        p->dfdr[5] = 2.0 * t / 3.0 * rp * rm * rp3 * ( 2.0 * s + 4.0 ) ;
        p->dfdr[6] = 4.0 * t / 3.0 * rp * rm * rp3 * (-2.0 * sp );
        p->dfdr[7] = -4.0 * t * rp * rm * rp5 * ( -2.0 * sp ) ;
        p->dfdr[8] = -2.0 / 3.0 * t * rp * rm * rp3 * ( -2.0 * s) ;
        p->dfdr[9] = 2.0 * t * rp * rm * rp5 * ( -2.0 * s ) ;
        p->dfdr[10] = -t * rp3 * rm * rp5 * ( -2.0 * s * sp5 + sp * sm ) ;
        p->dfdr[11] = 8.0 * t / 3.0 * rm * ( -2.0 * s * sp3 + sp * sm ) ;
        p->dfdr[12] = 8.0 * t / 3.0 * rp * ( -2.0 * s * sp3 + sp * sm ) ;
        p->dfdr[13] = -t / 3.0 * rp3 * rp * rp5 * ( -2.0 * s * sp5 + sp * sm ) ;

        return (0);
}

int     shcu8_3x3(x,p)
struct gausspts      *x;
struct shapefuncs    *p;

{
        double r, s, t, rm, sm, rp, sp, rp3, rp5, r2p8, sp3, sp5, s2p8 ;

        r = x->x ;
        s = x->y ;
        if ( (r<-5.00001) || (r>1.00001) || (s<-5.00001) || (s>1.00001) )
return(15) ;

        t = 1.0 / 256. ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
        sp = 1.0 + s ;
        sm = 1.0 - s ;
        rp3 = 3.0 + r ;
        rp5 = 5.0 + r ;
        sp3 = 3.0 + s ;
        sp5 = 5.0 + s ;

        p->dof = 15 ;



        p->f[3] = t * rp3 * sm * rm * rp5 * sp3 * sp5 ;
```

```
        p->f[0] = t / 3.0 * rp3 * rp * sm * rp5 * sp3 * sp5 ;
        p->f[1] = t / 9.0 * rp * sp * rp3 * rp5 * sp3 * sp5 ;
        p->f[2] = t / 3.0 * rm * sp * rp3 * rp5 * sp3 * sp5 ;
        p->f[4] = -t / 3.0 * rp * sp * rm * rp5 * sp3 * sp5 ;
        p->f[5] = 2.0 * t / 3.0 * rp * sp * sp3 * rm * rp3 ;
        p->f[6] = 4.0 * t / 3.0 * rp * sm * sp3 * rm * rp3 ;
        p->f[7] = -t * rp * sm * rm * rp5 * sp3 * sp5 ;
        p->f[8] = -2.0 / 3.0 * t * rp * sp * rm * sm * rp3 ;
        p->f[9] = t * rp * sp * rm * sm * sp5 * rp5 ;
        p->f[10] = -t * rp3 * sp * rm * rp5 * sm * sp5 ;
        p->f[11] = -2.0 / 3.0 * t * sp * rm * rp * sm * sp3 ;
        p->f[12] = 4.0 * t / 3.0 * sp * rp3 * rm * sm * sp3 ;
        p->f[13] = 2.0 * t / 3.0 * sp * rp3 * rp * sm * sp3 ;
        p->f[14] = -t / 3.0 * rp3   rp5 * sp * rp * sm * sp5 ;

        if (x->w < 0.0)
                return(0) ;
        r2p8 = 2.0 * r + 8.0 ;
        s2p8 = 2.0 * s + 8.0 ;

        p->dfdr[3] = t * sm * sp3 * sp5 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfdr[0] = t / 3.0 * sm * sp3 * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[1] = t / 9.0 * sp * sp3 * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[2] = t / 3.0 * sp * sp3 * sp5 * ( (-rp5 * rp3) + rm * r2p8 ) ;
        p->dfdr[4] = -t / 3.0 * sp5 * sp * sp3 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[5] = 2.0 * t / 3.0 * sp3 * sp * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[6] = 4.0 * t / 3.0 * sp3 * sm * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[7] = -t * sm * sp3 * sp5 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[8] = -2.0 / 3.0 * t * sp * sm * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[9] = t * sm * sp * sp5 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[10] = -t * sm * sp * sp5 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfdr[11] = -2.0 / 3.0 * t * sm * sp * sp3 * ( -2.0 * r ) ;
        p->dfdr[12] = -2.0 * t / 3.0 * sm * sp * sp3 * rp ;
        p->dfdr[13] = 2.0 * t / 3.0 * sp * sm * sp3 * ( 2.0 * r + 4.0 ) ;
        p->dfdr[14] = -t / 3.0 * sp * sm * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;

        p->dfds[3] = t * rp3 * rm * rp5 * ( -( sp5 * sp3 ) + sm * s2p8 ) ;
        p->dfds[0] = t / 3.0 * rp3 * rp * rp5 * ( -( sp5 * sp3 ) + sm * s2p8 ) ;
        p->dfds[1] = t / 9.0 * rp * rp3 * rp5 * ( ( sp5 * sp3 ) + sp * s2p8 ) ;
        p->dfds[2] = t / 3.0 * rm * rp3 * rp5 * ( ( sp5 * sp3 ) + sp * s2p8 ) ;
        p->dfds[4] = -t / 3.0 * rp * rm * rp5 * ( ( sp5 * sp3 ) + sp * s2p8 ) ;
        p->dfds[5] = 2.0 * t / 3.0 * rp * rm * rp3 * ( 2.0 * s + 4.0 ) ;
        p->dfds[6] = 4.0 * t / 3.0 * rp * rm * rp3 * ( -2.0 * sp ) ;
        p->dfds[7] = -t * rp * rm * rp5 * ( -( sp5 * sp3 ) + sm * s2p8 ) ;
        p->dfds[8] = -2.0 / 3.0 * t * rp * rm * rp3 * ( -2.0 * s) ;
        p->dfds[9] = t * rp * rm * rp5 * ( -2.0 * s * sp5 + sp * sm ) ;
        p->dfds[10] = -t * rp3 * rm * rp5 * ( -2.0 * s * sp5 + sp * sm ) ;
        p->dfds[11] = -2.0 / 3.0 * t * rm * rp * ( -2.0 * s * sp3 + sp * sm ) ;
        p->dfds[12] = 4.0 * t / 3.0 * rm * rp3 * ( -2.0 * s * sp3 + sp * sm ) ;
        p->dfds[13] = 2.0 * t / 3.0 * rp * rp3 * ( -2.0 * s * sp3 + sp * sm ) ;
        p->dfds[14] = -t / 3.0 * rp3 * rp * rp5 * ( -2.0 * s * sp5 + sp * sm ) ;

        return (0);
}
```

```
/*      the shape functions for the cubic upwinding functions   239          */
#include "fe.h"


extern struct control  N ;

int     shcu9_3x3(x,p)
struct gausspts      *x;
struct shapefuncs    *p;

{
        double r, s, t, rm, sm, rp, sp, rp3, rp5, r2p8, sp3, sp5, s2p8 ;

        r = x->x ;
        s = x->y ;
        if ( (r<-5.00001) || (r>1.00001) || (s<-5.00001) || (s>1.00001) )
return(16) ;

        t = 1.0 / 256. ;
        rm = 1.0 - r ;
        rp = 1.0 + r ;
        sp = 1.0 + s ;
        sm = 1.0 - s ;
        rp3 = 3.0 + r ;
        rp5 = 5.0 + r ;
        sp3 = 3.0 + s ;
        sp5 = 5.0 + s ;

        p->dof = 16 ;

        p->f[3] = t * rp3 * sm * rm * rp5 * sp3 * sp5 ;
        p->f[0] = t / 3.0 * rp3 * rp * sm * rp5 * sp3 * sp5 ;
        p->f[1] = t / 9.0 * rp * sp * rp3 * rp5 * sp3 * sp5 ;
        p->f[2] = t / 3.0 * rm * sp * rp3 * rp5 * sp3 * sp5 ;
        p->f[4] = -t / 3.0 * rp * sp * rm * rp5 * sp3 * sp5 ;
        p->f[5] = t / 9.0 * rp * sp * rm * rp3 * sp3 * sp5 ;
        p->f[6] = t / 3.0 * rp * sm * rm * rp3 * sp3 * sp5 ;
        p->f[7] = -t * rp * sm * rm * rp5 * sp3 * sp5 ;
        p->f[8] = -t / 3.0 * rp * sm * rm * rp3 * sp * sp5 ;
        p->f[9] = t * rp * sp * rm * rp5 * sm * sp5 ;
        p->f[10] = -t * rp3 * sp * rm * rp5 * sm * sp5 ;
        p->f[11] = t / 9.0 * rp3 * sp * rp * rm * sm * sp3 ;
        p->f[12] = -t / 3.0 * rp * sp * rm * rp5 * sm * sp3 ;
        p->f[13] = t / 3.0 * rp3 * sp * rm * rp5 * sm * sp3 ;
        p->f[14] = t / 9.0 * rp3 * sp * rp * rp5 * sm * sp3 ;
        p->f[15] = -t / 3.0 * rp3 * sp * rp * rp5 * sm * sp5 ;

        if (x->w < 0.0)
              return(0) ;

        r2p8 = 2.0 * r + 8.0 ;
        s2p8 = 2.0 * s + 8.0 ;


        p->dfdr[3] = t * sm * sp3 * sp5 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfdr[0] = t / 3.0 * sm * sp3 * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[1] = t / 9.0 * sp * sp3 * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[2] = t / 3.0 * sp * sp3 * sp5 * ( (-rp5 * rp3) + rm * r2p8 ) ;
        p->dfdr[4] = -t / 3.0 * sp * sp3 * sp5 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[5] = t / 9.0 * sp * sp3 * sp5 * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[6] = t / 3.0 * sm * sp3 * sp5 * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[7] = -t * sm * sp3 * sp5 * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[8] = -t / 3.0 * sm * sp * sp5 * ( -2.0 * r * rp3 + rp * rm ) ;
```

```
        p->dfdr[9] = t * sm * sp5 * sp * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[10] = -t * sm * sp * sp5 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfdr[11] = t / 9.0 * sm * sp * sp3 * ( -2.0 * r * rp3 + rp * rm ) ;
        p->dfdr[12] = -t / 3.0 * sm * sp3 * sp * ( -2.0 * r * rp5 + rp * rm ) ;
        p->dfdr[13] = t / 3.0 * sm * sp * sp3 * ( (-rp5 * rp3 ) + rm * r2p8 ) ;
        p->dfdr[14] = t / 9.0 * sp * sm * sp3 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;
        p->dfdr[15] = -t / 3.0 * sp * sm * sp5 * ( ( rp5 * rp3 ) + rp * r2p8 ) ;

        p->dfds[3] = t * rp3 * rm * rp5 * ( -( sp5 * sp3 ) + sm * s2p8 ) ;
        p->dfds[0] = t / 3.0 * rp3 * rp * rp5 * ( -( sp5 * sp3 ) + sm * s2p8 ) ;
        p->dfds[1] = t / 9.0 * rp * rp3 * rp5 * ( ( sp5 * sp3 ) + sp * s2p8 ) ;
        p->dfds[2] = t / 3.0 * rm * rp3 * rp5 * ( ( sp5 * sp3 ) + sp * s2p8 ) ;
        p->dfds[4] = -t / 3.0 * rp * rm * rp5 * ( ( sp5 * sp3 ) + sp * s2p8 ) ;
        p->dfds[5] = t / 9.0 * rp * rm * rp3  * ( ( sp5 * sp3 ) + sp * s2p8 ) ;
        p->dfds[6] = t / 3.0 * rp * rm * rp3  * ( -( sp5 * sp3 ) + sm * s2p8 ) ;
        p->dfds[7] = -t * rp * rm * rp5  * ( -( sp5 * sp3 ) + sm * s2p8 ) ;
        p->dfds[8] = -t / 3.0 * rp * rm * rp3 * ( -2.0 * s * sp5 + sp * sm ) ;
        p->dfds[9] = t * rp * rm * rp5 * ( -2.0 * s * sp5 + sp * sm ) ;
        p->dfds[10] = -t * rp3 * rm * rp5 * ( -2.0 * s * sp5 + sp * sm ) ;
        p->dfds[11] = t / 9.0 * rp3 * rp * rm * ( -2.0 * s * sp3 + sp * sm ) ;
        p->dfds[12] = -t / 3.0 * rp * rm * rp5 * ( -2.0 * s * sp3 + sp * sm ) ;
        p->dfds[13] = t / 3.0 * rp3 * rm * rp5 * ( -2.0 * s * sp3 + sp * sm ) ;
        p->dfds[14] = t / 9.0 * rp3 * rp * rp5 * ( -2.0 * s * sp3 + sp * sm ) ;
        p->dfds[15] = -t / 3.0 * rp3 * rp * rp5 * ( -2.0 * s * sp5 + sp * sm ) ;

        return (0);

}
```

Example Input File for a River Mesh Generation

```
40      5       20      20
x       z       h       u       v       Elong   Elat
212.9   459.9   0       0       0       0.000   0.000
213     460     0.06    0.064   0       0.004   0.002
215.55  461.75  0.31    0.356   0       0.020   0.014
218.1   463.5   0.51    0.456   0       0.026   0.017
220.65  465.25  0.55    0.481   0       0.027   0.018
223.2   467     0.48    0.436   0       0.025   0.017
225.75  468.75  0.6     0.436   0       0.025   0.017
228.3   471.1   0.66    0.483   0       0.028   0.018
230.85  472.25  0.73    0.512   0       0.029   0.020
233.4   474     0.58    0.504   0       0.029   0.019
235.95  475.75  0.63    0.49    0       0.028   0.019
238.5   477.5   0.48    0.523   0       0.030   0.020
241.05  479.25  0.54    0.473   0       0.027   0.018
243.6   481     0.53    0.511   0       0.029   0.019
246.15  482.75  0.42    0.488   0       0.028   0.019
248.7   484.5   0.4     0.439   0       0.025   0.017
251.25  486.25  0.74    0.333   0       0.019   0.013
253.8   488     0.15    0.178   0       0.010   0.007
256.35  488.8   0       0.      0       0.000   0.000

258.9   491.5   0       0       0       0.000   0.000
261.45  493.25  0       0       0       0.000   0.000
x       z       h       u       v       Elong   Elat
463     360     0       0       0       0.000   0.000
463.6   362.75  0.3     0.184   0       0.013   0.009
464.2   365.5   0.56    0.25    0       0.017   0.012
464.8   368.25  0.5     0.205   0       0.014   0.010
465.4   371     0.6     0.273   0       0.019   0.013
466     373.75  0.71    0.194   0       0.013   0.009
466.6   376.5   0.7     0.328   0       0.023   0.015
467.2   379.25  0.7     0.31    0       0.022   0.014
467.8   382     0.74    0.337   0       0.023   0.016
468.4   384.75  0.8     0.284   0       0.020   0.013
469     387.5   0.74    0.333   0       0.023   0.015
469.6   390.25  0.7     0.339   0       0.024   0.016
470.2   393     0.62    0.334   0       0.023   0.015
470.8   395.75  0.7     0.333   0       0.023   0.015
471.4   398.5   0.61    0.354   0       0.025   0.016
472     401.25  0.6     0.328   0       0.023   0.015
472.6   404     0.575   0.298   0       0.021   0.014
473.2   406.75  0.51    0.302   0       0.021   0.014
473.8   409.5   0.415   0.247   0       0.017   0.011
474.4   412.25  [illegible] 0.184 0     0.013   0.009
475     415     0       0       0       0.000   0.000
x       z       h       u       v       Elong   Elat
713     320     0       0       0       0.000   0.000
713     323     0.24    0.16    0       0.011   0.007
713     326     0.25    0.233   0       0.016   0.011
713     329     0.36    0.264   0       0.018   0.012
713     332     0.43    0.306   0       0.021   0.014
713     335     0.47    0.434   0       0.030   0.020
713     338     0.42    0.413   0       0.029   0.019
713     341     0.48    0.384   0       0.027   0.018
713     344     0.48    0.397   0       0.028   0.018
713     347     0.5     0.464   0       0.032   0.022
713     350     0.44    0.462   0       0.032   0.021
713     353     0.47    0.518   0       0.036   0.024
713     356     0.47    0.462   0       0.032   0.021
713     359     0.51    0.514   0       0.036   0.024
713     362     0.5     0.491   0       0.034   0.023
713     365     0.51    0.514   0       0.036   0.024
```

64	713	368	0.52	0.518	0	0.036	0.024
65	713	371	0.49	0.525	0	0.037	0.024
66	713	374	0.46	0.464	0	0.032	0.022
67	713	377	0.3	0.356	0	0.025	0.017
68	713	380	0	0	0	0.000	0.000
69	x	z	h	u	v	Elong	Elat
70	963	317	0	0	0	0.000	0.000
71	963	320	0.09	0.107	0	0.007	0.005
72	963	323	0.33	0.72	0	0.046	0.031
73	963	326	0.4	0.904	0	0.058	0.039
74	963	329	0.5	0.965	0	0.062	0.042
75	963	332	0.53	0.956	0	0.062	0.041
76	963	335	0.46	0.876	0	0.057	0.038
77	963	338	0.38	0.802	0	0.052	0.035
78	963	341	0.29	0.796	0	0.051	0.034
79	963	344	0.23	0.519	0	0.034	0.022
80	963	347	0.17	0.634	0	0.041	0.027
81	963	350	0.21	0.605	0	0.039	0.026
82	963	353	0.26	0.619	0	0.040	0.027
83	963	356	0.2	0.365	0	0.024	0.016
84	963	359	0.13	0.307	0	0.020	0.013
85	963	362	0.11	0.551	0	0.036	0.024
86	963	365	0.19	0.495	0	0.032	0.021
87	963	368	0.29	0.507	0	0.033	0.022
88	963	371	0.3	0.657	0	0.042	[illegible]
89	963	374	0.29	0.313	0	0.020	[illegible]
90	963	377	0	0	0	0.000	0.000
91	x	z	h	u	v	Elong	Elat
92	1213	220	0	0	0	0.000	0.000
93	1214	222.5	0.21	0.144	0	0.010	0.006
94	1215	225	0.35	0.223	0	0.015	0.010
95	1216	227.5	0.46	0.224	0	0.015	0.010
96	1217	230	0.49	0.252	0	0.017	0.011
97	1218	232.5	0.55	0.31	0	0.020	0.014
98	1219	235	0.63	0.334	0	0.022	0.015
99	1220	237.5	0.74	0.326	0	0.022	0.014
100	1221	240	0.79	0.315	0	0.021	0.014
101	1222	242.5	0.84	0.339	0	0.022	0.015
102	1223	245	0.92	0.339	0	0.022	0.015
103	1224	247.5	1.03	0.328	0	0.022	0.014
104	1225	250	1	0.311	0	0.021	0.014
105	1226	252.5	1	0.31	0	0.020	0.014
106	1227	255	1	0.246	0	0.016	0.011
107	1228	257.5	0.83	0.256	0	0.017	0.011
108	1229	260	0.64	0.233	0	0.015	0.010
109	1230	262.5	0.6	0.114	0	0.008	0.005
110	1231	265	0.4	0.039	0	0.003	0.002
111	1232	267.5	0	0	0	0.000	0.000
112	1233	270	0	0	0	0.000	0.000
113	x	z	h	u	v	Elong	Elat
114	1463	0	0	0	0	0.000	0.000
115	1464.9	2.5	0.3	0.129	0	0.012	0.008
116	1466.8	5	0.41	0.145	0	0.013	0.009
117	1468.7	7.5	0.64	0.046	0	0.004	0.003
118	1470.6	10	0.68	0.031	0	0.003	0.002
119	1472.5	12.5	0.7	0.038	0	0.004	0.002
120	1474.4	15	0.92	0.038	0	0.004	0.002
121	1476.3	17.5	0.92	0.269	0	0.025	0.017
122	1478.2	20	0.74	0.31	0	0.029	0.019
123	1480.1	22.5	0.57	0.334	0	0.031	0.021
124	1482	25	0.52	0.391	0	0.036	0.024
125	1483.9	27.5	0.55	0.36	0	[illegible]	0.022
126	1485.8	30	0.57	0.412	0	0.038	0.025

127	1487.7	32.5	0.58	0.422	0	0.039	0.026
128	1489.6	35	0.58	0.397	0	0.037	0.024
129	1491.5	37.5	0.61	0.359	0	0.033	0.022
130	1493.4	40	0.59	0.368	0	0.034	0.023
131	1495.3	42.5	0.6	0.294	0	0.027	0.018
132	1497.2	45	0.58	0.344	0	0.032	0.021
133	1499.1	47.5	0.56	0.284	0	0.026	0.017
134	1501	50	0.57	0.235	0	0.022	0.014

Example Output of a Mesh

Transient analysis = 1
Number of Time Steps = 10
Delta t Acceleration Factor = 1.000
Time = 0.00000
Delta t = 10.00000
Theta = 0.500
Dimensions = 2
Number of Variables = 1
[K] governing equation numbers
1 1
Number of Parameters = 8
Number of Boundary Parameters = -2
Number of Nodes = 231
Number of Elements = 200
Number of Boundary Elements = 60
Number of Element in X direction = 10
Number of Blocks in X direction = 5
Number of Element in Y direction = 20
Number of Blocks in Y direction = 20

Node Information

Node#, Coordinates, Parameters, Variables

1	212.85	459.90	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	213.00	460.00	0.059417	-0.023791	0.0037216	-0.00068979	-0.00068979	0.0022751	0.0000	0.059998	0.0000
3	215.55	461.75	0.33073	-0.13175	0.019175	-0.0020616	-0.0020616	0.014821	0.0000	0.31000	0.0000
4	218.10	463.50	0.42486	-0.16561	0.024813	-0.0030464	-0.0030464	0.018186	0.0000	0.51000	0.0000
5	220.65	465.25	0.44921	-0.17198	0.025850	-0.0030060	-0.0030060	0.019149	0.0000	0.55000	0.0000
6	223.20	467.00	0.40760	-0.15478	0.023993	-0.0026562	-0.0026562	0.018007	0.0000	0.48000	0.0000
7	225.75	468.75	0.40772	-0.15446	0.023997	-0.0026515	-0.0026515	0.018003	0.0000	0.60000	0.0000
8	228.30	471.08	0.45167	-0.[illegible]113	0.026744	-0.0033139	-0.0033139	0.019253	0.0000	0.66000	0.0000
9	230.85	472.25	0.47975	-0.17885	0.027901	-0.0029466	-0.0029466	0.021095	0.0000	0.73000	0.0000
10	233.40	474.00	0.47279	-0.17459	0.027799	-0.0032503	-0.0032503	0.020197	0.0000	0.58000	0.0000
11	235.95	475.75	0.45998	-0.16887	0.026931	-0.0029126	-0.0029126	0.020067	0.0000	0.63000	0.0000
12	238.50	477.50	0.49128	-0.17936	0.028822	-0.0032222	-0.0032222	0.021173	0.0000	0.48000	0.0000
13	241.05	479.25	0.44441	-0.16195	0.025945	-0.0028961	-0.0028961	0.019053	0.0000	0.54000	0.0000
14	243.60	481.00	0.48040	-0.17417	0.027837	-0.0032049	-0.0032049	0.020159	0.0000	0.53000	0.0000
15	246.15	482.75	0.45902	-0.16565	0.026963	-0.0028745	-0.0028745	0.020035	0.0000	0.42000	0.0000
16	248.70	484.50	0.41272	-0.14962	0.024072	-0.0025643	-0.0025643	0.017928	0.0000	0.40000	0.0000
17	251.25	486.25	0.31378	-0.11150	0.018325	-0.0018920	-0.0018920	0.013673	0.0000	0.74000	0.0000

Example Output of a Mesh

18	253.80	488.00	0.17010	-0.052456	0.0097417	-0.00084436	-0.00084436	0.0072642	0.0000	0.15000	0.0000
19	256.35	488.76	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
20	258.90	491.50	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
21	261.45	493.25	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
22	337.93	409.95	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
23	338.30	411.38	0.11519	-0.045900	0.0080883	-0.0010310	-0.0010310	0.0059116	0.0000	0.18000	0.0000
24	339.87	413.62	0.28182	-0.11130	0.017753	-0.0018767	-0.0018767	0.013742	0.0000	0.43500	0.0000
25	341.45	415.87	0.30794	-0.12004	0.019147	-0.0021996	-0.0021996	0.014361	0.0000	0.50500	0.0000
26	343.03	418.13	0.35211	-0.13470	0.022038	-0.0025003	-0.0025003	0.016459	0.0000	0.57500	0.0000
27	344.60	420.38	0.29462	-0.11148	0.018246	-0.0019849	-0.0019849	0.013752	0.0000	0.59500	0.0000
28	346.17	422.63	0.35748	-0.13467	0.023007	-0.0026403	-0.0026403	0.016994	0.0000	0.64999	0.0000
29	347.75	425.17	0.37121	-0.13935	0.023890	-0.0029621	-0.0029621	0.017111	0.0000	0.68000	0.0000
30	349.32	427.13	0.39794	-0.14779	0.025031	-0.0026119	-0.0026119	0.018968	0.0000	0.73500	0.0000
31	350.90	429.38	0.36987	-0.13577	0.023489	-0.0027494	-0.0027494	0.017008	0.0000	0.69000	0.0000
32	352.48	431.63	0.38665	-0.14084	0.024502	-0.0027332	-0.0027332	0.017994	0.0000	0.68500	0.0000
33	354.05	433.87	0.40532	-0.14654	0.025960	-0.0028785	-0.0028785	0.019038	0.0000	0.59000	0.0000
34	355.62	436.12	0.37971	-0.13652	0.024028	-0.0027084	-0.0027084	0.017469	0.0000	0.58000	0.0000
35	357.20	438.37	0.39729	-0.14226	0.024978	-0.0028573	-0.0028573	0.018021	0.0000	0.61500	0.0000
36	358.78	440.62	0.39660	-0.14123	0.025491	-0.0028450	-0.0028450	0.018515	0.0000	0.51500	0.0000
37	360.35	442.87	0.36144	-0.12819	0.023108	-0.0025213	-0.0025213	0.016893	0.0000	0.50000	0.0000
38	361.92	445.12	0.29765	-0.10462	0.019290	-0.0020335	-0.0020335	0.014219	0.0000	0.65750	0.0000
39	363.50	447.37	0.22682	-0.078443	0.014968	-0.0015460	-0.0015460	0.011033	0.0000	0.33000	0.0000
40	365.07	449.13	0.11687	-0.039922	0.0081859	-0.00091716	-0.00091716	0.0058141	0.0000	0.20750	0.0000
41	366.65	451.87	0.087132	-0.029514	0.0062945	-0.00060739	-0.00060739	0.0047071	0.0000	0.099998	0.0000
42	368.23	454.12	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
43	463.00	360.00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
44	463.60	362.75	0.17756	-0.048264	0.012725	-0.0010119	-0.0010119	0.0092778	0.0000	0.30000	0.0000
45	464.20	365.50	0.24142	-0.064943	0.016661	-0.0012536	-0.0012536	0.012338	0.0000	0.56000	0.0000
46	464.80	368.25	0.19819	-0.052391	0.013739	-0.00098776	-0.00098776	-0.010263	0.0000	0.50000	0.0000
47	465.40	371.00	0.26415	-0.068974	0.018614	-0.0014659	-0.0014659	0.013383	0.0000	0.60000	0.0000
48	466.00	373.75	0.18778	-0.048723	0.012748	-0.00097186	-0.00097186	0.0092548	0.0000	0.71000	0.0000
49	466.60	376.50	0.31759	-0.081990	0.022503	-0.0019370	-0.0019370	0.015499	0.0000	0.70000	0.0000
50	467.20	379.25	0.30034	-0.076770	0.021512	-0.0019202	-0.0019202	0.014491	0.0000	0.70000	0.0000
51	467.80	382.00	0.32675	-0.082490	0.022583	-0.0016622	-0.0016622	0.016418	0.0000	0.74000	0.0000
52	468.40	384.75	0.27556	-0.068689	0.019587	-0.0016419	-0.0016419	0.013410	0.0000	0.80000	0.0000
53	469.00	387.50	0.32332	-0.079722	0.022544	-0.0018603	-0.0018603	0.015458	0.0000	0.74000	0.0000
54	469.60	390.25	0.32934	-0.080338	0.023553	-0.0018426	-0.0018426	0.016448	0.0000	0.70000	0.0000
55	470.20	393.00	0.32464	-0.078523	0.022560	-0.0018288	-0.0018288	0.015442	0.0000	0.62000	0.0000
56	470.80	395.75	0.32373	-0.078028	0.022563	-0.0018231	-0.0018231	0.015439	0.0000	0.70000	0.0000
57	471.40	398.50	0.34424	-0.082533	0.024512	-0.0020411	-0.0020411	0.016488	0.0000	0.610[illegible]	0.0000
58	472.00	401.25	0.31904	-0.076142	0.022571	-0.0018071	-0.0018071	0.015431	0.0000	0.60000	0.0000
59	472.60	404.00	0.28986	-0.069173	0.020626	-0.0015814	-0.0015814	0.014377	0.0000	0.57500	0.0000

Example Output of a Mesh

60	473.20	406.75	0.29385	-0.069713	0.020631	-0.0015731	-0.0015731	0.014373	0.0000	0.51000	0.0000
61	473.80	409.50	0.24035	-0.056928	0.016680	-0.0013449	-0.0013449	0.011320	0.0000	0.41500	0.0000
62	474.40	412.25	0.17876	-0.043575	0.012776	-0.00091983	-0.00091983	0.0092269	0.0000	0.20000	0.0000
63	475.00	415.00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
64	588.00	340.00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
65	588.30	342.87	0.16994	-0.026590	0.011906	-0.00061058	-0.00061058	0.0080988	0.0000	0.27000	0.0000
66	588.60	345.75	0.23879	-0.036083	0.016391	-0.00073827	-0.00073827	0.011617	0.0000	0.40500	0.0000
67	588.90	348.62	0.23200	-0.034190	0.015893	-0.00072079	-0.[illegible]72079	0.011108	0.0000	0.43000	0.0000
68	589.20	351.50	0.28644	-0.042010	0.019867	-0.00093327	-0.00093327	0.013641	0.0000	0.51500	0.0000
69	589.50	354.38	0.31064	-0.045842	0.021351	-0.0010104	[illegible]0.0010104	0.014653	0.0000	0.59000	0.0000
70	589.80	357.25	0.36657	-0.053859	0.025811	-0.0012948	-0.0012948	0.017189	0.0000	0.56000	0.0000
71	590.10	360.13	0.34347	-0.049394	0.024333	-0.0011986	-0.0011986	0.016171	0.0000	0.59000	0.0000
72	590.40	363.00	0.36342	-0.051188	0.025340	-0.0011749	-0.0011749	0.017164	0.0000	0.61000	0.0000
73	590.70	365.87	0.37048	-0.051204	0.[illegible]	-0.0011527	-0.0011527	0.017661	0.0000	0.65000	0.0000
74	591.00	368.75	0.39389	-0.053466	0.027332	-0.0012660	-0.[illegible]012660	0.018177	0.0000	0.59000	0.0000
75	591.30	371.62	0.42472	-0.056763	0.029831	-0.0013133	-0.0013133	0.020180	0.0000	0.58500	0.0000
76	591.60	374.50	0.39457	-0.052152	0.027341	-0.0012339	-0.0012339	0.018168	0.0000	0.54501	0.0000
77	591.90	377.37	0.41985	-0.055478	0.029331	-0.0012990	-0.0012990	0.019672	0.0000	0.60500	0.0000
78	592.20	380.25	0.41886	-0.055374	0.029331	-0.0012996	-0.0012996	0.019673	0.0000	0.55500	0.0000
79	592.50	383.12	0.41734	-0.055397	0.029330	-0.0013047	[illegible]0013047	0.019674	0.0000	0.55500	0.0000
80	592.80	386.00	0.40436	-0.054383	0.028335	-0.0012[illegible]	[illegible]012548	0.019174	0.0000	0.54750	0.0000
81	593.10	388.88	0.40967	-0.056192	0.028822	-0.0013[illegible]	[illegible]013466	0.019190	0.0000	0.50000	0.0000
82	593.40	391.75	0.35207	-0.049281	0.024344	-0.0010977	[illegible]010977	0.016656	0.0000	0.43750	0.0000
83	593.70	394.62	0.26732	-0.037961	0.018886	-0.00083579	[illegible]00083579	0.013119	0.0000	0.25000	0.0000
84	594.00	397.50	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
85	713.00	320.00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
86	713.00	323.00	0.15946	-0.013136	0.010975	-0.00032711	-0.00032711	0.0070307	0.0000	0.24000	0.0000
87	713.00	326.00	0.23219	-0.019429	0.015964	-0.00041525	-0.00041525	0.011036	0.0000	0.25000	0.0000
88	713.00	329.00	0.26301	-0.022840	0.017953	-0.00051685	-0.00051685	0.012046	0.0000	0.36000	0.0000
89	713.00	332.00	0.30475	-0.027664	0.020946	-0.00063058	-0.00063058	0.014057	0.0000	0.43000	0.0000
90	713.00	335.00	0.43193	-0.042317	0.029904	-0.00097062	-0.00097062	0.020092	0.0000	0.47000	0.0000
91	713.00	338.00	0.41074	-0.043145	0.028890	-0.0010392	-0.0010392	0.019106	0.0000	0.42000	0.0000
92	713.00	341.00	0.38159	-0.042985	0.026888	-0.0010014	-0.0010014	0.018111	0.0000	0.48000	0.0000
93	713.00	344.00	0.39441	-0.045296	0.027870	-0.0011337	-0.0011337	0.018128	0.0000	0.48000	0.0000
94	713.00	347.00	0.46110	-0.051808	0.031873	-0.0011090	-0.0011090	0.022128	0.0000	0.50000	0.0000
95	713.00	350.00	0.45927	-0.050175	0.031868	-0.0011869	-0.0011869	0.021133	0.0000	0.44000	0.0000
96	713.00	353.00	0.51519	-0.053859	0.035873	-0.0012410	-0.0012410	0.024132	0.0000	0.47000	0.0000
97	713.00	356.00	0.45971	-0.045958	0.031889	-0.0010882	-0.0010882	0.021113	0.0000	0.47000	0.0000
98	713.00	359.00	0.51162	-0.049456	0.035892	-0.0011494	-0.0011494	0.024113	0.0000	0.51000	0.0000
99	713.00	362.00	0.48892	-0.045145	0.033904	-0.0010066	-0.0010066	0.023096	0.0000	0.50000	0.0000
100	713.00	365.00	0.51207	-0.044465	0.035913	-0.0010343	-0.0010343	0.024092	0.0000	0.51000	0.0000
101	713.00	368.00	0.51631	-0.041815	0.035925	-0.00096562	-0.00096562	0.024080	0.0000	0.52000	0.0000

Example Output of a Mesh

102	713.00	371.00	0.52346	-0.040251-	0.036926	-0.00099380	-0.00099380	0.024078	0.0000	0.49000	0.0000
103	713.00	374.00	0.46264	-0.035495	0.031939	-0.00076233	-0.00076233	0.022062	0.0000	0.46000	0.0000
104	713.00	377.00	0.35491	-0.027912	0.024952	-0.00062555	-0.00062555	0.017047	0.0000	0.30000	0.0000
105	713.00	380.00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
106	838.00	318.50	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
107	838.00	321.50	0.13350	-0.000841	0.0090026	-1.8891e-05	-1.8891e-05	0.0060045	0.0000	0.16500	0.0000
108	838.00	324.50	0.47644	-0.007756	0.030996	-0.00016266	-0.00016266	0.021006	0.0000	0.29000	0.0000
109	838.00	327.50	0.58375	-0.017088	0.037991	-0.00036551	-0.00036551	0.025516	0.0000	0.38000	0.0000
110	838.00	330.50	0.63510	-0.022560	0.041487	-0.00047908	-0.00047908	0.028017	0.0000	0.46500	0.0000
111	838.00	333.50	0.69435	-0.030092	0.045969	-0.00067029	-0.00067029	0.030531	0.0000	0.50000	0.0000
112	838.00	336.50	0.64347	-0.036446	0.042953	-0.00081842	-0.00081842	0.028550	0.0000	0.44000	0.0000
113	838.00	339.50	0.59152	-0.041971	0.039440	-0.00091783	-0.00091783	0.026570	0.0000	0.43000	0.0000
114	838.00	342.50	0.59445	-0.049460	0.039412	-0.0011159	-0.0011159	0.026094	0.0000	0.38500	0.0000
115	838.00	345.50	0.48935	-0.045938	0.032901	-0.0010230	-0.0010230	0.022099	0.0000	0.36501	0.0000
116	838.00	348.50	0.54561	-0.051202	0.036390	-0.0011625	-0.0011625	0.024111	0.0000	0.30500	0.0000
117	838.00	351.50	0.55957	-0.046495	0.037420	-0.0010319	-0.0010319	0.025087	0.0000	0.34000	0.0000
118	838.00	354.50	0.53900	-0.040295	0.035936	-0.00089220	-0.00089220	0.024069	0.0000	0.36501	0.0000
119	838.00	357.50	0.43836	-0.031623	0.029955	-0.00071782	-0.00071782	0.020056	0.0000	0.35500	0.0000
120	838.00	360.50	0.39818	-0.025601	0.026963	-0.00057595	-0.00057595	0.018042	0.0000	0.31500	0.0000
121	838.00	363.50	0.53190	-0.025318	0.035976	-0.00056995	-0.00056995	0.024029	0.0000	0.31001	0.0000
122	838.00	366.50	0.50623	-0.016519	0.033992	-0.00037478	-0.00037478	0.022519	0.0000	0.35500	0.0000
123	838.00	369.50	0.51591	-0.009980	0.034999	-0.00023208	-0.00023208	0.023007	0.0000	0.39000	0.0000
124	838.00	372.50	0.56045	-0.00768	0.037000	-0.00016443	-0.00016443	0.025004	0.0000	0.38000	0.0000
125	838.00	375.50	0.33447	-0.004462	0.022498	-0.00010004	-0.00010004	0.015001	0.0000	0.29501	0.0000
126	838.00	378.50	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
127	963.00	317.00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
128	963.00	320.00	0.10510	-0.020091	0.0069330	-0.00037005	-0.00037005	0.0050680	0.0000	0.089996	0.0000
129	963.00	323.00	0.70687	-0.13690	0.045455	-0.0027999	-0.0027999	0.031540	0.0000	0.33000	0.0000
130	963.00	326.00	0.88737	-0.17260	0.057306	-0.0035604	-0.0035604	0.039694	0.0000	0.40000	0.0000
131	963.00	329.00	0.94726	-0.18416	0.061268	-0.0037460	-0.0037460	0.042728	0.0000	0.50000	0.0000
132	963.00	332.00	0.93826	-0.18330	0.061225	-0.0039511	-0.0039511	0.041772	0.0000	0.53000	0.0000
133	963.00	335.00	0.85952	-0.16914	0.056291	-0.0035990	-0.0035990	0.038710	0.0000	0.46000	0.0000
134	963.00	338.00	0.78656	-0.15660	0.051354	-0.0032552	-0.0032552	0.035652	0.0000	0.38000	0.0000
135	963.00	341.00	0.78007	-0.15846	0.050329	-0.0033176	-0.0033176	0.034670	0.0000	0.29000	0.0000
136	963.00	344.00	0.50786	-0.10693	0.033487	-0.0024180	-0.0024180	0.022512	0.0000	0.23000	0.0000
137	963.00	347.00	0.61964	-0.13421	0.040373	-0.0028964	-0.0028964	0.027628	0.0000	0.17000	0.0000
138	963.00	350.00	0.59116	-0.12868	0.038413	-0.0027019	-0.0027019	0.026589	0.0000	0.21000	0.0000
139	963.00	353.00	0.60364	-0.13704	0.039364	-0.0028068	-0.0028068	0.027638	0.0000	0.26000	0.0000
140	963.00	356.00	0.35337	-0.091385	0.023500	-0.0019400	-0.0019400	0.016501	0.0000	0.20000	0.0000
141	963.00	359.00	0.29800	-0.073794	0.019592	-0.0016324	-0.0016324	0.013405	0.0000	0.13000	0.0000
142	963.00	362.00	0.53808	-0.11863	0.035447	-0.0025232	-0.0025232	0.024558	0.0000	0.11000	0.0000
143	963.00	365.00	0.48406	-0.10349	0.031517	-0.0022476	-0.0022476	0.021484	0.0000	0.19000	0.0000

Example Output of a Mesh

144	963.00	368.00	0.49602	-0.10495	0.032526	-0.0022264	-0.0022264	0.022474	0.0000	0.29000	0.0000
145	963.00	371.00	0.64343	-0.13281	0.041428	-0.0027717	-0.0027717	0.028572	0.0000	0.30000	0.0000
146	963.00	374.00	0.30685	-0.061737	0.019724	-0.0013528	-0.0013528	0.013273	0.0000	0.29000	0.0000
147	963.00	377.00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
148	1088.0	268.50	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
149	1088.5	271.25	0.11676	-0.046019	0.0080950	-0.0010254	-0.0010254	0.0058973	0.0000	0.14999	0.0000
150	1089.0	274.00	0.44088	-0.16996	0.029208	-0.0033570	-0.0033570	0.021794	0.0000	0.34000	0.0000
151	1089.5	276.75	0.52802	-0.19821	0.035018	-0.0039461	-0.0039461	0.025987	0.0000	0.43000	0.0000
152	1090.0	279.50	0.57116	-0.20988	0.037952	-0.0042064	-0.0042064	0.028050	0.0000	0.49500	0.0000
153	1090.5	282.25	0.59538	-0.21496	0.039443	-0.0043134	-0.0043134	0.029054	0.0000	0.54000	0.0000
154	1091.0	285.00	0.57069	-0.20083	0.038072	-0.0040709	-0.0040709	0.027937	0.0000	0.54500	0.0000
155	1091.5	287.75	0.53326	-0.18365	0.035678	-0.0038475	-0.0038475	0.025831	0.0000	0.56000	0.0000
156	1092.0	290.50	0.52592	-0.17884	0.034759	-0.0036580	-0.0036580	0.025246	0.0000	0.54000	0.0000
157	1092.5	293.25	0.40621	-0.13798	0.027017	-0.0028927	-0.0028927	0.019484	0.0000	0.53500	0.0000
158	1093.0	296.00	0.45950	-0.15980	0.030368	-0.0032593	-0.0032593	0.022130	0.0000	0.54500	0.0000
159	1093.5	298.75	0.43892	-0.15806	0.029297	-0.0033465	-0.0033465	0.021209	0.0000	0.62000	0.0000
160	1094.0	301.50	0.43615	-0.16124	0.029300	-0.0032531	-0.0032531	0.021703	0.0000	0.63000	0.0000
161	1094.5	304.25	0.31585	-0.11890	0.021[illegible]	-0.0023093	-0.0023093	0.015869	0.0000	0.60000	0.0000
162	1095.0	307.00	0.25791	-0.099681	0.017218	-0.0020165	-0.0020165	0.012780	0.0000	0.56500	0.0000
163	1095.5	309.75	0.37344	-0.15282	0.025213	-0.0031556	-0.0031556	0.018793	0.0000	0.47000	0.0000
164	1096.0	312.50	0.33599	-0.14001	0.022316	-0.0028389	-0.0028389	0.016686	0.0000	0.41500	0.0000
165	1096.5	315.25	0.28648	-0.11976	0.019459	-0.0024896	-0.0024896	0.014545	0.0000	0.44500	0.0000
166	1097.0	318.00	0.32130	-0.13369	0.021392	-0.0026600	-0.0026600	0.016106	0.0000	0.35001	0.0000
167	1097.5	320.75	0.14452	-0.060056	0.0094799	-0.0012382	-0.0012382	0.0070148	0.0000	0.14500	0.0000
168	1098.0	323.50	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
169	1213.0	220.00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
170	1214.0	222.50	[illegible]	-0.074927	0.0089177	-0.0017798	-0.0017798	0.0070812	0.0000	0.21000	0.0000
171	1215.0	225.00	0.19253	-0.11643	0.013661	-0.0022128	-0.0022128	0.011340	0.0000	0.35000	0.0000
172	1216.0	227.50	0.19239	-0.11473	0.013688	-0.0021983	-0.0021983	0.011313	0.0000	0.46000	0.0000
173	1217.0	230.00	0.21732	-0.12758	0.015461	-0.0026181	-0.0026181	0.012539	0.0000	0.49000	0.0000
174	1218.0	232.50	0.26856	-0.15484	0.018501	-0.0025949	-0.0025949	0.015496	0.0000	0.55000	0.0000
175	1219.0	235.00	0.29010	-0.16552	0.020283	-0.0030147	-0.0030147	0.016719	0.0000	0.63000	0.0000
176	1220.0	237.50	0.28340	-0.16111	0.020048	-0.0034385	-0.0034385	0.015955	0.0000	0.74000	0.0000
177	1221.0	240.00	0.27398	-0.15544	0.019298	-0.0030060	-0.0030060	0.015705	0.0000	0.79000	0.0000
178	1222.0	242.50	0.29484	-0.16729	0.020298	-0.0030061	-0.0030061	0.016705	0.0000	0.84000	0.0000
179	1223.0	245.00	0.29452	-0.16786	0.020286	-0.0030131	-0.0030131	0.016717	0.0000	0.92000	0.0000
180	1224.0	247.50	0.28461	-0.16304	0.020026	-0.0034519	-0.0034519	0.015977	0.0000	1.0300	0.0000
181	1225.0	250.00	0.269[illegible]	-0.15507	0.019262	-0.0030271	-0.0030271	0.015741	0.0000	1.0000	0.0000
182	1226.0	252.50	0.26841	-0.15509	0.018496	-0.0025977	-0.0025977	0.015501	0.0000	1.0000	0.0000
183	1227.0	255.00	0.21242	-0.12408	0.014728	-0.0021764	-0.0021764	0.012273	0.0000	1.0000	0.0000
184	1228.0	257.50	0.22057	-0.12993	0.015453	-0.0026225	-0.0026225	0.012546	0.0000	0.83000	0.0000
185	1229.0	260.00	0.20067	-0.11841	0.013709	-0.0021871	-0.0021871	0.011293	0.0000	0.64000	0.0000

Example Output of a Mesh

186	1230.0	262.50	0.098543	-0.057314	0.0072434	-0.0013064	-0.0013064	0.0057571	0.0000	0.60000	0.0000
187	1231.0	265.00	0.034803	-0.017604	0.0027947	-0.00040254	-0.00040254	0.0022025	0.0000	0.40000	0.0000
188	1232.0	267.50	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
189	1233.0	270.00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
190	1338.0	110.00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
191	1339.5	112.50	0.10247	-0.090175	0.0092568	-0.0019827	-0.0019827	0.0087486	0.0000	0.25500	0.0000
192	1340.9	115.00	0.13921	-0.12184	0.012051	-0.0022271	-0.0022271	0.011455	0.0000	0.38000	0.0000
193	1342.3	117.50	0.10241	-0.087970	0.0082255	-0.0014820	-0.0014820	0.0077734	0.0000	0.55000	0.0000
194	1343.8	120.00	0.10834	-0.091026	0.0085530	-0.0017247	-0.0017247	0.0079494	0.0000	0.58500	0.0000
195	1345.3	122.50	0.13398	-0.11103	0.010373	-0.0019641	-0.0019641	0.0096309	0.0000	0.62500	0.0000
196	1346.7	125.00	0.14314	-0.11878	0.011165	-0.0022120	-0.0022120	0.010335	0.0000	0.77500	0.0000
197	1348.2	127.50	0.22868	-0.19030	0.020232	-0.0039350	-0.0039350	0.018778	0.0000	0.83000	0.0000
198	1349.6	130.00	0.24051	-0.19953	0.021537	-0.0041765	-0.0041765	0.019967	0.0000	0.76500	0.0000
199	1351.0	132.50	0.25953	-0.21419	0.023061	-0.0041724	-0.0041724	0.021449	0.0000	0.70500	0.0000
200	1352.5	135.00	0.28221	-0.23148	0.025183	-0.0046613	-0.0046613	0.023324	0.0000	0.72000	0.0000
201	1354.0	137.50	0.26667	-0.21731	0.023713	-0.0046515	-0.0046515	0.021796	0.0000	0.79000	0.0000
202	1355.4	140.00	0.28090	-0.22756	0.025540	-0.0048922	-0.0048922	0.023464	0.0000	0.78500	0.0000
203	1356.8	142.50	0.28516	-0.22943	0.025770	-0.0046392	-0.0046392	0.023737	0.0000	0.79000	0.0000
204	1358.3	145.00	0.25131	-0.20052	0.023002	-0.0043890	-0.0043890	0.021004	0.0000	0.79000	0.0000
205	1359.8	147.50	0.24111	-0.19085	0.021728	-0.0041360	-0.0041360	0.019776	0.0000	0.72000	0.0000
206	1361.2	150.00	0.23648	-0.18541	0.021454	-0.0038823	-0.0038823	0.019546	0.0000	0.61500	0.0000
207	1362.7	152.50	0.16102	-0.12526	0.015238	-0.0029099	-0.0029099	0.0[illegible]761	0.0000	0.60000	0.0000
208	1364.1	155.00	0.15105	-0.11772	0.015236	-0.0029075	-0.0029075	0.013771	0.0000	0.49001	0.0000
209	1365.5	157.50	0.11127	-0.088215	0.011263	-0.0021913	-0.0021913	0.010236	0.0000	0.28000	0.0000
210	1367.0	160.00	0.091371	-0.073876	0.0094213	-0.0019546	-0.0019546	0.0085841	0.0000	0.28500	0.0000
211	1463.0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
212	1464.9	2.5000	0.096943	-0.085103	0.010261	-0.0019821	-0.0019821	0.0097432	0.0000	0.30000	0.0000
213	1466.8	5.0000	0.10912	-0.095480	0.011267	-0.0019812	-0.0019812	0.010736	0.0000	0.41000	0.0000
214	1468.7	7.5000	0.034810	-0.030067	0.0035707	-0.00049441	-0.00049441	0.0034254	0.0000	0.64000	0.0000
215	1470.6	10.000	0.023712	-0.019966	0.0025837	-0.00049243	-0.00049243	0.0024135	0.0000	0.68000	0.0000
216	1472.5	12.500	0.029907	-0.023447	0.0032369	-0.00097057	-0.00097057	0.0027598	0.0000	0.70000	0.0000
217	1474.4	15.000	0.030590	-0.022548	0.0032941	-0.00095470	-0.00095470	0.0027026	0.0000	0.92000	0.0000
218	1476.3	17.500	0.20868	-0.16974	0.021815	-0.0039178	-0.0039178	0.020185	0.0000	0.92000	0.0000
219	1478.2	20.000	0.23912	-0.19728	0.024949	-0.0049100	-0.0049100	0.023048	0.0000	0.74000	0.0000
220	1480.1	22.500	0.25829	-0.21175	0.026981	-0.0049002	-0.0049002	0.025021	0.0000	0.57000	0.0000
221	1482.0	25.000	0.30337	-0.24667	0.031227	-0.0058743	-0.0058743	0.028778	0.0000	0.52000	0.0000
222	1483.9	27.500	0.28032	-0.22587	0.028669	-0.0053712	-0.0053712	0.026331	0.0000	0.55000	0.0000
223	1485.8	30.000	0.32181	-0.25726	0.032933	-0.0063407	-0.0063407	0.030070	0.0000	0.57000	0.0000
224	1487.7	32.500	0.33016	-0.26282	0.033959	-0.0063347	-0.0063347	0.031044	0.0000	0.58000	0.0000
225	1489.6	35.000	0.31157	-0.24604	0.032009	-0.0063231	-0.0063231	0.028995	0.0000	0.58000	0.0000
226	1491.5	37.500	0.28274	-0.22122	0.028823	-0.0053355	-0.0053355	0.026178	0.0000	0.61000	0.0000
227	1493.4	40.000	0.29034	-0.22611	0.029846	-0.0053295	-0.0053295	0.027153	0.0000	0.59000	0.0000

Example Output of a Mesh

228	1495.0	42.500	0.23240	-0.18007	0.023623	-0.0043587	-0.0043587	0.021375	0.0000	0.60000	0.0000
229	1497.2	45.000	0.27148	-0.21127	0.027851	-0.0053285	-0.0053285	0.025150	0.0000	0.58000	0.0000
230	1499.2	47.500	0.22312	-0.17571	0.022555	-0.0043759	-0.0043759	0.020444	0.0000	0.56000	0.0000
231	1501.0	50.000	0.18164	-0.14911	0.018781	-0.0039249	-0.0039249	0.017222	0.0000	0.57000	0.0000

Element Information

Element #, vtype, gtype, nodes

1	211	211	22	23	2	1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	211	211	23	24	3	2	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
3	211	211	24	25	4	3	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
4	211	211	25	26	5	4	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
5	211	211	26	27	6	5	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
6	211	211	27	28	7	6	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
7	211	211	28	29	8	7	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
8	211	211	29	30	9	8	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
9	211	211	30	31	10	9	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
10	211	211	31	32	11	10	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
11	211	211	32	33	12	11	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
12	211	211	33	34	13	12	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
13	211	211	34	35	14	13	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
14	211	211	35	36	15	14	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
15	211	211	36	37	16	15	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
16	211	211	37	38	17	16	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
17	211	211	38	39	18	17	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
18	211	211	39	40	19	18	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
19	211	211	40	41	20	19	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
20	211	211	41	42	21	20	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
21	211	211	43	44	23	22	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
22	211	211	44	45	24	23	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
23	211	211	45	46	25	24	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
24	211	211	46	47	26	25	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
25	211	211	47	48	27	26	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
26	211	211	48	49	28	27	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
27	211	211	49	50	29	28	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
28	211	211	50	51	30	29	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
29	211	211	51	52	31	30	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
30	211	211	52	53	32	31	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
31	211	211	53	54	33	32	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
32	211	211	54	55	34	33	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
33	211	211	55	56	35	34	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Example Output of a Mesh

34	211	211	56	57	36	35	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
35	211	211	57	58	37	36	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
36	211	211	58	59	38	37	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
37	211	211	59	60	39	38	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
38	211	211	60	61	40	39	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
39	211	211	61	62	41	40	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
40	211	211	62	63	42	41	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
41	211	211	64	65	44	43	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
42	211	211	65	66	45	44	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
43	211	211	66	67	46	45	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
44	211	211	67	68	47	46	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
45	211	211	68	69	48	47	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
46	211	211	69	70	49	48	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
47	211	211	70	71	50	49	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
48	211	211	71	72	51	50	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
49	211	211	72	73	52	51	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
50	211	211	73	74	53	52	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
51	211	211	74	75	54	53	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
52	211	211	75	76	55	54	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
53	211	211	76	77	56	55	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
54	211	211	77	78	57	56	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
55	211	211	78	79	58	57	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
56	211	211	79	80	59	58	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
57	211	211	80	81	60	59	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
58	211	211	81	82	61	60	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
59	211	211	82	83	62	61	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
60	211	211	83	84	63	62	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
61	211	211	85	86	65	64	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
62	211	211	86	87	66	65	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
63	211	211	87	88	67	66	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
64	211	211	88	89	68	67	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
65	211	211	89	90	69	68	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
66	211	211	90	91	70	69	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
67	211	211	91	92	71	70	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
68	211	211	92	93	72	71	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
69	211	211	93	94	73	72	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
70	211	211	94	95	74	73	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
71	211	211	95	96	75	74	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
72	211	211	96	97	76	75	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
73	211	211	97	98	77	76	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
74	211	211	98	99	78	77	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
75	211	211	99	100	79	78	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Example Output of a Mesh

76	211	211	100	101	80	79	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
77	211	211	101	102	81	80	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
78	211	211	102	103	82	81	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
79	211	211	103	104	83	82	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
80	211	211	104	105	84	83	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
81	211	211	106	107	86	85	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
82	211	211	107	108	87	86	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
83	211	211	108	109	88	87	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
84	211	211	109	110	89	88	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
85	211	211	110	111	90	89	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
86	211	211	111	112	91	90	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
87	211	211	112	113	92	91	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
88	211	211	113	114	93	92	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
89	211	211	114	115	94	93	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
90	211	211	115	116	95	94	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
91	211	211	116	117	96	95	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
92	211	211	117	118	97	96	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
93	211	211	118	119	98	97	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
94	211	211	119	120	99	98	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
95	211	211	120	121	100	99	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
96	211	211	121	122	101	100	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
97	211	211	122	123	102	101	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
98	211	211	123	124	103	102	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
99	211	211	124	125	104	103	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
100	211	211	125	126	105	104	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
101	211	211	127	128	107	106	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
102	211	211	128	129	108	107	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
103	211	211	129	130	109	108	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
104	211	211	130	131	110	109	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
105	211	211	131	132	111	110	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
106	211	211	132	133	112	111	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
107	211	211	133	134	113	112	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
108	211	211	134	135	114	113	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
109	211	211	135	136	115	114	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
110	211	211	136	137	116	115	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
111	211	211	137	138	117	116	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
112	211	211	138	139	118	117	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
113	211	211	139	140	119	118	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
114	211	211	140	141	120	119	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
115	211	211	141	142	121	120	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
116	211	211	142	143	122	121	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
117	211	211	143	144	123	122	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Example Output of a Mesh

118	211	211	144	145	124	123	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
119	211	211	145	146	125	124	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
120	211	211	146	147	126	125	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
121	211	211	148	149	128	127	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
122	211	211	149	150	129	128	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
123	211	211	150	151	130	129	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
124	211	211	151	152	131	130	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
125	211	211	152	153	132	131	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
126	211	211	153	154	133	132	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
127	211	211	154	155	134	133	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
128	211	211	155	156	135	134	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
129	211	211	156	157	136	135	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
130	211	211	157	158	137	136	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
131	211	211	158	159	138	137	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
132	211	211	159	160	139	138	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
133	211	211	160	161	140	139	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
134	211	211	161	162	141	140	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
135	211	211	162	163	142	141	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
136	211	211	163	164	143	142	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
137	211	211	164	165	144	143	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
138	211	211	165	166	145	144	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
139	211	211	166	167	146	145	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
140	211	211	167	168	147	146	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
141	211	211	169	170	149	148	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
142	211	211	170	171	150	149	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
143	211	211	171	172	151	150	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
144	211	211	172	173	152	151	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
145	211	211	173	174	153	152	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
146	211	211	174	175	154	153	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
147	211	211	175	176	155	154	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
148	211	211	176	177	156	155	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
149	211	211	177	178	157	156	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
150	211	211	178	179	158	157	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
151	211	211	179	180	159	158	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
152	211	211	180	181	160	159	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
153	211	211	181	182	161	160	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
154	211	211	182	183	162	161	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
155	211	211	183	184	163	162	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
156	211	211	184	185	164	163	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
157	211	211	185	186	165	164	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
158	211	211	186	187	166	165	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
159	211	211	187	188	167	166	0.0000	0.0000	[illegible]	[illegible]	0.0000	0.0000	0.0000	0.0000

Example Output of a Mesh

160	211	211	188	189	168	167	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
161	211	211	190	191	170	169	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
162	211	211	191	192	171	170	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
163	211	211	192	193	172	171	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
164	211	211	193	194	173	172	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
165	211	211	194	195	174	173	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
166	211	211	195	196	175	174	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
167	211	211	196	197	176	175	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
168	211	211	197	198	177	176	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
169	211	211	198	199	178	177	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
170	211	211	199	200	179	178	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
171	211	211	200	201	180	179	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
172	211	211	201	202	181	180	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
173	211	211	202	203	182	181	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
174	211	211	203	204	183	182	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
175	211	211	204	205	184	183	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
176	211	211	205	206	185	184	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
177	211	211	206	207	186	185	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
178	211	211	207	208	187	186	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
179	211	211	208	209	188	187	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
180	211	211	209	210	189	188	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
181	211	211	211	212	191	190	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
182	211	211	212	213	192	191	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
183	211	211	213	214	193	192	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
184	211	211	214	215	194	193	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
185	211	211	215	216	195	194	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
186	211	211	216	217	196	195	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
187	211	211	217	218	197	196	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
188	211	211	218	219	198	197	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
189	211	211	219	220	199	198	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
190	211	211	220	221	200	199	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
191	211	211	221	222	201	200	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
192	211	211	222	223	202	201	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
193	211	211	223	224	203	202	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
194	211	211	224	225	204	203	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
195	211	211	225	226	205	204	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
196	211	211	226	227	206	205	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
197	211	211	227	228	207	206	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
198	211	211	228	229	208	207	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
199	211	211	229	230	209	208	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
200	211	211	230	231	210	209	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Example Output of a Mesh

Boundary Element #, vtype, gtype, nodes, boundary condition codes

1	111	111	1	22	0.0000	0.0000	2
2	111	111	22	43	0.0000	0.0000	2
3	111	111	43	64	0.0000	0.0000	2
4	111	111	64	85	0.0000	0.0000	2
5	111	111	85	106	0.0000	0.0000	2
6	111	111	106	127	0.0000	0.0000	2
7	111	111	127	148	0.0000	0.0000	2
8	111	111	148	169	0.0000	0.0000	2
9	111	111	169	190	0.0000	0.0000	2
10	111	111	190	211	0.0000	0.0000	2
11	111	111	211	212	0.0000	0.0000	2
12	111	111	212	213	0.0000	0.0000	2
13	111	111	213	214	0.0000	0.0000	2
14	111	111	214	215	0.0000	0.0000	2
15	111	111	215	216	0.0000	0.0000	2
16	111	111	216	217	0.0000	0.0000	2
17	111	111	217	218	0.0000	0.0000	2
18	111	111	218	219	0.0000	0.0000	2
19	111	111	219	220	0.0000	0.0000	2
20	111	111	220	221	0.0000	0.0000	2
21	111	111	221	222	0.0000	0.0000	2
22	111	111	222	223	0.0000	0.0000	2
23	111	111	223	224	0.0000	0.0000	2
24	111	111	224	225	0.0000	0.0000	2
25	111	111	225	226	0.0000	0.0000	2
26	111	111	226	227	0.0000	0.0000	2
27	111	111	227	228	0.0000	0.0000	2
28	111	111	228	229	0.0000	0.0000	2
29	111	111	229	230	0.0000	0.0000	2
30	111	111	230	231	0.0000	0.0000	2
31	111	111	231	210	0.0000	0.0000	2
32	111	111	210	189	0.0000	0.0000	2
33	111	111	189	168	0.0000	0.0000	2
34	111	111	168	147	0.0000	0.0000	2
35	111	111	147	126	0.0000	0.0000	2
36	111	111	126	105	0.0000	0.0000	2
37	111	111	105	84	0.0000	0.0000	2
38	111	111	84	63	0.0000	0.0000	2
39	111	111	63	42	0.0000	0.0000	2
40	111	111	42	21	0.0000	0.0000	2

Example Output of a Mesh

[illegible]	111	111	21	20	0.0000	0.0000	3
42	111	111	20	19	0.0000	0.0000	3
43	111	111	19	18	0.0000	0.0000	3
44	111	111	18	17	0.0000	0.0000	3
45	111	111	17	16	0.0000	0.0000	3
46	111	111	16	15	0.0000	0.0000	3
47	111	111	15	14	0.0000	0.0000	3
48	111	111	14	13	0.0000	0.0000	3
49	111	111	13	12	0.0000	0.0000	3
50	111	111	12	11	0.0000	0.0000	3
51	111	111	11	10	0.0000	0.0000	3
52	111	111	10	9	0.0000	0.0000	3
53	111	111	9	8	0.0000	0.0000	3
54	111	111	8	7	0.0000	0.0000	3
55	111	111	7	6	0.0000	0.0000	3
56	111	111	6	5	0.0000	0.0000	3
57	111	111	5	4	0.0000	0.0000	3
58	111	111	4	3	0.0000	0.0000	3
59	111	111	3	2	0.0000	0.0000	3
60	111	111	2	1	0.0000	0.0000	3