CHEMICAL TRANSPORT BY THREE-DIMENSIONAL GROUNDWATER FLOWS

Prepared by

D.K. Babu
G.F. Pinder
A. Niemi
D.P. Ahlfeld
S.A. Stothoff

84-WR-3

Revised June 1997

Contents

Chapter 1

INTRODUCTION

1.1 Purpose of Manual

This manual describes the theory and use of the Princeton Transport Code (**PTC**). The first three chapters will introduce the method used in solving the partial differential equations which describe groundwater flow and contaminant transport, while Chapter 4 consists of documentation for **PTC**. The manual is designed for the practicing hydrologist rather than for the theoretician. Accordingly, we will omit detailed discussions of equation development and error analysis, and refer the interested reader to the numerical methods and water resources literature.

1.2 Governing Equations

PTC uses the following system of partial differential equations to represent groundwater flow described by hydraulic head, h,

$$\frac{\partial}{\partial x}\left(K_{xx}\frac{\partial h}{\partial x}\right)+\frac{\partial}{\partial y}\left(K_{yy}\frac{\partial h}{\partial y}\right)+\frac{\partial}{\partial z}\left(K_{zz}\frac{\partial h}{\partial z}\right)-S\frac{\partial h}{\partial t}+Q=0 \tag{1.1}$$

groundwater velocity components,

$$V_x=-K_{xx}\frac{\partial h}{\partial x},\qquad V_y=-K_{yy}\frac{\partial h}{\partial y},\qquad V_z=-K_{zz}\frac{\partial h}{\partial z} \tag{1.2}$$

and contaminant transport described by concentration, c,

$$\begin{aligned}\frac{\partial}{\partial x}\left[D_{xx}\frac{\partial c}{\partial x}+D_{xy}\frac{\partial c}{\partial y}+D_{xz}\frac{\partial c}{\partial z}\right]+\frac{\partial}{\partial y}\left[D_{yx}\frac{\partial c}{\partial x}+D_{yy}\frac{\partial c}{\partial y}+D_{yz}\frac{\partial c}{\partial z}\right]\\ +\frac{\partial}{\partial z}\left[D_{zx}\frac{\partial c}{\partial x}+D_{zy}\frac{\partial c}{\partial y}+D_{zz}\frac{\partial c}{\partial z}\right]-\left[V_x\frac{\partial c}{\partial x}+V_y\frac{\partial c}{\partial y}+V_z\frac{\partial c}{\partial z}\right]\\ +Q(c^w-c)-\theta[1+E(c)]\left(\frac{\partial c}{\partial t}\right)=0\end{aligned} \tag{1.3}$$

These equations are derived from conservation of mass principles and Darcy's Law.

The definitions of various symbols introduced above will be explained in the following chapters. The solution of these equations proceeds in the following sequence: first solve for the hydraulic heads h from (1.1); next, calculate the Darcy velocities V_x, V_y, V_z from (1.2); and finally, solve (1.3) for the contaminant concentration c.

1.3 The PTC Solution Algorithm

Solving the system of equations (1.1)–(1.3) for complex physical systems generally requires application of numerical methods. For field scale systems, the computational effort involved in solving a numerical discretization of these three-dimensional equations is large. **PTC** employs a unique splitting algorithm for solving the fully three-dimensional equations which reduces the computational burden significantly.

The algorithm involves discretizing the domain into approximately parallel horizontal layers. Within each layer a finite element discretization [Pinder and Gray, 1977] is used allowing for accurate representation of irregular domains. The layers are connected vertically by a finite difference discretization. This hybrid coupling of the finite element and finite difference methods provides the opportunity to apply the splitting procedure. During a given time iteration, all the computations are divided into two steps. In the first step all the horizontal finite element discretizations are solved independently of each other. In the second step, the vertical equations which link the layers are solved.

The splitting procedure and its advantages will be presented in greater detail in the next chapter.

Chapter 2

THREE DIMENSIONAL GROUNDWATER FLOW

2.1 Governing Equation

PTC determines the flow characteristics of a groundwater system by solving for the hydraulic head via the following partial differential equation

$$\frac{\partial}{\partial x}\left(K_{xx}\frac{\partial h}{\partial x}\right)+\frac{\partial}{\partial y}\left(K_{yy}\frac{\partial h}{\partial y}\right)+\frac{\partial}{\partial z}\left(K_{zz}\frac{\partial h}{\partial z}\right)-S\frac{\partial h}{\partial t} + \sum_{i=1}^{r} Q_i\delta(x-x_i)\delta(y-y_i)\delta(z-z_i)=0 \qquad (2.1)$$

where, using a fundamental set of units denoted by length [L], time [T], and mass [M],

- h is hydraulic head [L],
- K_{xx} is the hydraulic conductivity in the x horizontal direction [LT^{-1}],
- K_{yy} is the hydraulic conductivity in the y horizontal direction [LT^{-1}],
- K_{yy} is the hydraulic conductivity in the z (vertical) direction [LT^{-1}],
- S is the specific storage coefficient [L^{-1}],
- Q_i is the source/sink term at location i [L^3T^{-1}] (e.g. pumps; positive values imply injection),
- $\delta(\ )$ is the Dirac delta function,
- r is the number of source sink points.

For convenience, the last term in (2.1) will be abbreviated as Q.

The governing equation (2.1) is solved numerically by **PTC** using finite element and finite difference methods. In the following sections, we introduce each method and apply it to (2.1).

2.2 Application of the Finite Element Method

The splitting scheme used by **PTC** to solve (2.1) involves approximating the terms of (2.1) containing x and y derivatives using a finite element method. Finite elements in the horizontal plane are widely used (see, for example, Pinder and Gray, 1977).

The finite element method assumes that there exists an infinite sum of functions that will exactly represent the solution to the partial differential equation describing groundwater flow. A finite approximate form of the series is

$$h \sim \hat{h} = \sum_{i=1}^{N} h_i(z,t) w_i(x,y) \qquad (2.2)$$

where

h is hydraulic head [L],
$\hat{h}$ is the series approximation to h [L],
h_i is an undetermined coefficient [L],
w_i is a basis (or interpolating) function (dimensionless), and
N is the number of nodes in the finite element mesh.

The series approximation (2.2) provides an exact representation as N approaches infinity ($\hat{h}$ approaches h). By a careful selection of the basis functions w_i, the undetermined coefficients h_i become the head values at nodes with coordinates (x, y, z). One key to the computational efficiency of the finite element method is the use of piecewise continuous basis functions which are nonzero over only a small subarea of the total domain. While many different types of basis functions may be used [Lapidus and Pinder, 1982], **PTC** uses piecewise linear basis functions between adjacent finite element nodes.

The finite element method proceeds by noting that, although the differential operator L (e.g. equation 2.1) operating on h is equal to zero, when L operates on the approximating function an error is introduced. In mathematical notation, we rewrite (2.1) as

$$\mathrm{L}(h) = 0 \qquad (2.3)$$

while

$$\mathrm{L}(\hat{h}) = R \qquad (2.4)$$

where R is the residual error.

To solve (2.2) using the finite element method we attempt to minimize the residual R. We accomplish this by first considering a complete set of functions w_j. If we now force the residual R to be orthogonal to all possible values of w_j we are, in fact, forcing R to zero and thereby obtaining a solution to (2.2). Expressing this another way

$$\mathrm{L}(\hat{h}) = \mathrm{L}(h) \qquad \text{when } R = 0. \qquad (2.5)$$

PTC uses the same set of functions for the weighting functions w_j as for the basis functions w_i; this procedure is called Galerkin's method. Accordingly, w_i and w_j are used interchangably throughout the remainder of this work.

Unfortunately, the condition expressed by (2.5) can be achieved only as N approaches infinity and computers can deal only with finite sets of numbers. We are forced, therefore, to consider a finite subset of values w_i, $i = 1, 2, \ldots, N$, which generally makes our solution approximate rather than exact. Recalling the definition of orthogonal functions these N conditions can be expressed as

$$\iint_{\Omega} \mathrm{L}(\hat{h}) w_i \, dx \, dy = 0 \qquad i = 1, 2, \ldots, N \tag{2.6}$$

where the domain of integration Ω covers the entire horizontal cross section of the flow region. Introducing the definition of (2.2) we obtain, for each weighting function w_i,

$$\iint_{\Omega} \left[\frac{\partial}{\partial x}\left(K_{xx} \frac{\partial \hat{h}}{\partial x} \right) + \frac{\partial}{\partial y}\left(K_{yy} \frac{\partial \hat{h}}{\partial y} \right) + \frac{\partial}{\partial z}\left(K_{zz} \frac{\partial \hat{h}}{\partial z} \right) - S \frac{\partial \hat{h}}{\partial t} + Q \right] w_i \, dx \, dy = 0 \tag{2.7}$$

Following the standard procedure for integration in the two-dimensional case, the x and y terms of equation (2.7) can be integrated using Green's Theorem, producing

$$\begin{aligned} \iint_{\Omega} \left[K_{xx} \frac{\partial \hat{h}}{\partial x} \frac{\partial w_i}{\partial x} + K_{yy} \frac{\partial \hat{h}}{\partial y} \frac{\partial w_i}{\partial y} \right] dx \, dy - \int_{\sigma} \left[K_{xx} \frac{\partial \hat{h}}{\partial x} l_x + K_{yy} \frac{\partial \hat{h}}{\partial y} l_y \right] w_i \, d\sigma \\ - \iint_{\Omega} \left[\frac{\partial}{\partial z}\left(K_{zz} \frac{\partial \hat{h}}{\partial z} \right) \right] w_i \, dx \, dy + \iint_{\Omega} \left[S \frac{\partial \hat{h}}{\partial t} - Q \right] w_i \, dx \, dy = 0 \end{aligned} \tag{2.8}$$

where l_x and l_y are the direction cosines between the normal to the cross-sectional boundary σ (the $d\sigma$ represents a small length along this boundary) and the x and y coordinate axes, respectively. Substituting (2.2) into (2.8) completes our use of the finite element method for discretizing (2.1)

$$\begin{aligned} \iint_{\Omega} \Bigg[& K_{xx} \left(\sum_{j=1}^{N} h_j \frac{\partial w_j}{\partial x} \right) \frac{\partial w_i}{\partial x} + K_{yy} \left(\sum_{j=1}^{N} h_j \frac{\partial w_j}{\partial y} \right) \frac{\partial w_i}{\partial y} \\ & - \frac{\partial}{\partial z}\left(K_{zz} \frac{\partial}{\partial z} \sum_{j=1}^{N} h_j w_j \right) w_i + S \frac{\partial}{\partial t}\left(\sum_{j=1}^{N} h_j w_j \right) w_i - Q w_i \Bigg] \, dx \, dy \\ & - \int_{\sigma} \left[K_{xx} \frac{\partial \hat{h}}{\partial x} l_x + K_{yy} \frac{\partial \hat{h}}{\partial y} l_y \right] w_i \, d\sigma = 0 \end{aligned} \tag{2.9}$$

where $i = 1, 2, \ldots, N$. A formal substitution for the last term in (2.9) was not made because the quantity in brackets is, in fact, the flux across the vertical boundary σ of the horizontal region. This term thus represents a horizontal flux boundary condition.

In the horizontal plane, at a given time level, the finite element discretization summarized in (2.9) provides us with N equations in the N unknown coefficients (heads) defined in (2.2). The finite element method has provided the means to transform the derivatives of the unknown heads with respect to x and y to derivatives of the known basis functions. In the next section, the application of the finite difference method to discretize the derivatives with respect to z and time is described.

As an aid to the mass balance calculations, all of these equations are multiplied by the layer thickness within the code. This is an equivalent formulation.

2.3 Application of the Finite Difference Method

Introducing matrix notation, where boldface capital letters represent square matrices and boldface lowercase letters represent column vectors, we rewrite (2.9) in matrix form:

$$\mathbf{A}\mathbf{h} + \mathbf{B}\frac{\partial \mathbf{h}}{\partial t} - \mathbf{v} + \mathbf{f} = 0 \tag{2.10}$$

where $\mathbf{A}$ and $\mathbf{B}$ are $(N \times N)$ matrices and $\mathbf{h}$, $\partial\mathbf{h}/\partial t$, $\mathbf{v}$ and $\mathbf{f}$ are column vectors of length N. The elements of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{v}$ and $\mathbf{f}$ are

$$A_{ij} = \iint_{\Omega} \left[K_{xx} \frac{\partial w_i}{\partial x} \frac{\partial w_j}{\partial x} + K_{yy} \frac{\partial w_i}{\partial y} \frac{\partial w_j}{\partial y} \right] dx\, dy \tag{2.11a}$$

$$B_{ij} = \iint_{\Omega} S w_i w_j \, dx\, dy \tag{2.11b}$$

$$f_i = -\iint_{\Omega} Q w_i \, dx\, dy - \int_{\sigma} \left[K_{xx} \frac{\partial \hat{h}}{\partial x} l_x + K_{yy} \frac{\partial \hat{h}}{\partial y} l_y \right] w_i \, d\sigma \tag{2.11c}$$

$$v_i = \sum_{j=1}^{N} \left[\iint_{\Omega} \frac{\partial}{\partial z} \left(K_{zz} \frac{\partial h_j}{\partial z} \right) w_i w_j \, dx\, dy \right] \tag{2.11d}$$

where, as discussed earlier, $\mathbf{f}$ contains known boundary conditions.

2.3.1 The Vertical Derivative

The central feature of this computer code is the use of a central differencing scheme for the space derivatives in the z direction in (2.11d). The vertical discretization is accomplished by requiring that the horizontal finite element meshes be replicated in layers with nodes stacked one above the other (see Figure 2.1). This means that in the vertical direction a one-dimensional finite difference equation can be used to approximate (2.11d). Using k as

the vertical index, with $k = 1$ as the bottom layer, this approximation written in matrix form yields:

$$\mathbf{v} \cong \mathbf{C}_k^+(\mathbf{h}_{k+1} - \mathbf{h}_k) - \mathbf{C}_k^-(\mathbf{h}_k - \mathbf{h}_{k-1}) \tag{2.12}$$

where the harmonic mean of adjacent layer properties is used to define the elements of $\mathbf{C}_k^+$, the vertical term between layer k and layer $k+1$, and $\mathbf{C}_k^-$, the vertical term between layer k and layer $k-1$

$$C_{ij;k}^{\pm} = \iint_{\Omega} \frac{2}{\Delta z_k[(\Delta z/K_{zz})_{k\pm1} + (\Delta z/K_{zz})_k]}\, w_i w_j\, dx\, dy \tag{2.13}$$

where (Δz_k) is the thickness of the kth layer at the point of approximation. The harmonic mean gives the most realistic quantities in the heterogeneous situations normally encountered.

Substituting (2.12) into (2.10) produces the following expression for a typical kth layer:

$$\mathbf{A}_k\mathbf{h}_k + \mathbf{B}_k\frac{\partial \mathbf{h}_k}{\partial t} - \Big[\mathbf{C}_k^+(\mathbf{h}_{k+1} - \mathbf{h}_k) - \mathbf{C}_k^-(\mathbf{h}_k - \mathbf{h}_{k-1})\Big] + \mathbf{f}_k = 0 \tag{2.14}$$

where $\mathbf{h}_k$ represents the vector $\mathbf{h}$ of heads in the kth layer, $k = 1, 2, \ldots, M$, and M is the number of layers in the z direction.

2.3.2 The Time Derivative

Our experience indicates that an implicit backward difference approximation of the time derivative provides the most accurate solution to groundwater flow problems for a given cost. In the backward difference representation, a first order correct scheme is used to approximate the time derivative and the spatial derivatives are written at the new time level.

Applying this scheme to the time derivative in (2.14) yields, for each layer,

$$\begin{aligned} \mathbf{A}_k\mathbf{h}_k^{(t+\Delta t)} &+ \frac{(\mathbf{B}_D)_k}{\Delta t}\Big[\mathbf{h}_k^{(t+\Delta t)} - \mathbf{h}_k^t\Big] \\ &- \Big[\mathbf{C}_k^+(\mathbf{h}_{k+1} - \mathbf{h}_k) - \mathbf{C}_k^-(\mathbf{h}_k - \mathbf{h}_{k-1})\Big]^{t+\Delta t} + \mathbf{f}_k^t = 0 \end{aligned} \tag{2.15}$$

Equation (2.15) is the complete discretization of (2.1), and provides us with $M \times N$ equations in the N unknowns in (2.2) over each of the M layers. Computationally efficient solution of (2.15) is the focus of the next section.

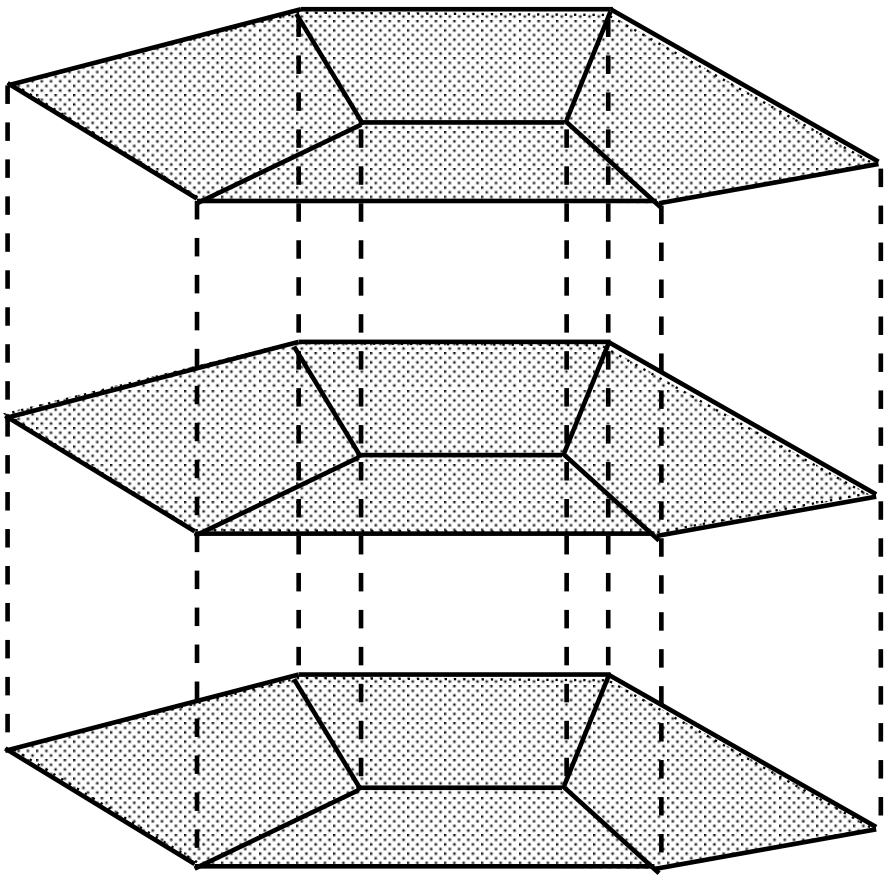


Figure 2.1: Schematic of horizontal finite element meshes stacked atop one another, producing the three-dimensional discretization.

2.4 Solving the Discretized Equations

2.4.1 Matrix Lumping

Computational efficiency and reduced storage requirements can be met by lumping matrices [Pinder and Gray, 1977]. In the lumping of a matrix, the terms in each row are summed up, and the resulting sum delegated to the diagonal position of that row. It may be seen that for the storage matrix $\mathbf{B}$ (with terms $\iint S w_i w_j \, dx \, dy$), and for vertical conductivity matrices $\mathbf{C}$ (with terms defined by (2.13)), the diagonal elements predominate over the off-diagonal elements. It would, therefore, appear that lumping of these matrices introduces only minor errors into the solution.

Thus, the lumped $\mathbf{C}$ and $\mathbf{B}$ matrices become, notationally, $\mathbf{C}_D$ and $\mathbf{B}_D$, with diagonal elements given by

$$(C_i)_k^{\pm} = \sum_{i=1}^{N} C_{ij;k}^{\pm} \tag{2.16a}$$

$$(B_i)_k = \sum_{i=1}^{N} B_{ij;k} \tag{2.16b}$$

where $C_{ij;k}^{\pm}$ is the (i,j)th entry in the original matrix $\mathbf{C}_k^{\pm}$ and $B_{ij;k}$ is the (i,j)th entry in the original matrix $\mathbf{B}_k$.

All non-diagonal elements of $\mathbf{B}_D$ and $\mathbf{C}_D$ are defined as zero. Note that by lumping of the vertical conductivity matrices, horizontal coupling between nodes in (2.12) has been removed, producing a tridiagonal matrix equation at each node.

2.4.2 The Splitting Algorithm

To simplify discussion of this algorithm, we introduce the new notation

$$\mathbf{P}[\mathbf{C}_D; \mathbf{h}] = \mathbf{C}_k^{+}(\mathbf{h}_{k+1} - \mathbf{h}_k) - \mathbf{C}_k^{-}(\mathbf{h}_k - \mathbf{h}_{k-1}) \tag{2.17}$$

where the lumped form of the matrices has been used. Using the lumped form and substituting (2.17) into (2.15) yields, for layers $k = 1, 2, \ldots, M$,

$$\mathbf{A}_k \mathbf{h}_k^{(t+\Delta t)} + \frac{(\mathbf{B}_D)_k}{\Delta t}\left[\mathbf{h}_k^{(t+\Delta t)} - \mathbf{h}_k^{t}\right] - \mathbf{P}[\mathbf{C}_D; \mathbf{h}]^{(t+\Delta t)} + \mathbf{f}_k^{t} = 0 \tag{2.18}$$

The splitting algorithm is motivated by the recognition that solving (2.18) as a single matrix equation presents an enormous computational burden for field scale problems. To mitigate this burden, **PTC** uses a novel two-step splitting approach. In the first step, the z derivative terms are assumed known and the remaining terms are solved for an intermediate solution ($\mathbf{h}_k^{(t+\Delta t)^*}$).

For a representative layer k, the matrix equation for this first step is:

STEP 1: $(k = 1, 2, \ldots, M)$

$$\mathbf{A}_k \mathbf{h}_k^{(t+\Delta t)^*} + \frac{(\mathbf{B}_D)_k}{\Delta t}\left[\mathbf{h}_k^{(t+\Delta t)^*} - \mathbf{h}_k^t\right] - \mathbf{P}[\mathbf{C}_D; \mathbf{h}]^t + \mathbf{f}_k^t = 0 \qquad (2.19)$$

By lagging the vertical derivative in (2.19) the horizontal layer equations have been decoupled. Solving (2.19) then consists of solving M independent horizontal finite element systems.

The next step in the procedure retains the interim solution $\mathbf{h}_k^{(t+\Delta t)^*}$ in the $\mathbf{A}$ term, but evaluates the z derivatives at the new time step. The matrix equations for the second stage are thus:

STEP 2: $(k = 1, 2, \ldots, M)$

$$\mathbf{A}_k \mathbf{h}_k^{(t+\Delta t)^*} + \frac{(\mathbf{B}_D)_k}{\Delta t}\left[\mathbf{h}_k^{(t+\Delta t)} - \mathbf{h}_k^t\right] - \mathbf{P}[\mathbf{C}_D; \mathbf{h}]^{(t+\Delta t)} + \mathbf{f}_k^t = 0 \qquad (2.20)$$

A simple algebraic manipulation of the above equations leads to substantial savings in computational effort. Equation (2.20) can be rewritten by subtracting (2.20) from (2.19), yielding

$$\mathbf{P}[\mathbf{C}_D; \mathbf{h}]^{(t+\Delta t)} - \frac{(\mathbf{B}_D)_k}{\Delta t}\mathbf{h}_k^{(t+\Delta t)} - \mathbf{P}[\mathbf{C}_D; \mathbf{h}]^t + \frac{(\mathbf{B}_D)_k}{\Delta t}\mathbf{h}_k^{(t+\Delta t)^*} = 0 \qquad (2.21)$$

Due to the decoupling introduced by the matrix lumping the solution of (2.21) involves a single tridiagonal system for each node in the horizontal plane (i.e. N independent tridiagonal systems).

2.4.3 The Final Form

Rewriting the matrix equations for each step of the **PTC** splitting algorithm with known quantities on the right hand side, we have

STEP 1: $(k = 1, 2, \ldots, M)$

$$\left(\mathbf{A}_k + \frac{(\mathbf{B}_D)_k}{\Delta t}\right)\mathbf{h}_k^{(t+\Delta t)^*} = -\mathbf{f}_k^t + \mathbf{P}[\mathbf{C}_D; \mathbf{h}]^t + \frac{(\mathbf{B}_D)_k}{\Delta t}\mathbf{h}_k^t \qquad (2.22)$$

STEP 2: $(k = 1, 2, \ldots, M)$

$$\frac{(\mathbf{B}_D)_k}{\Delta t}\mathbf{h}_k^{(t+\Delta t)} - \mathbf{P}[\mathbf{C}_D; \mathbf{h}]^{(t+\Delta t)} = \frac{(\mathbf{B}_D)_k}{\Delta t}\mathbf{h}_k^{(t+\Delta t)^*} - \mathbf{P}[\mathbf{C}_D; \mathbf{h}]^t \qquad (2.23)$$

The combined algorithm consists of solving M systems of equations of dimension N (Step 1) and N tridiagonal systems of equations (Step 2) to complete one time step in the $M \times N$ unknowns.

2.5 Boundary Conditions

PTC accommodates three types of boundary conditions on the flow equation. The three types are Dirichlet (specified head), Neumann (specified flux) and third type (leakage). Boundary conditions may be changed by the user during the course of a simulation as described in Chapter 4. The default boundary condition for all boundaries is zero flux. This implies a confined aquifer with impermeable bottom and sides. By selecting alternate boundary conditions throughout the domain, the user can represent other hydrologic conditions.

2.5.1 Specified Head

Constant head boundaries are defined by specifying nodes and the head values which are fixed at those nodes. The code automatically factors out those rows and columns in the coefficient matrix associated with those nodes. Inasmuch as constant-head nodes are thus effectively eliminated from our matrix equation, we will be left with $(N \times M - N_c)$ equations in $(N \times M - N_c)$ unknowns, with N_c being the total number of constant-head boundary nodes in the flow domain.

2.5.2 Specified Flux

The finite element method provides a very simple means of specifying flux boundary conditions. Upon application of Green's Theorem in (2.8) a boundary term arises. This term can be rewritten as:

$$-\int_{\sigma}\left[K_{xx}\frac{\partial\hat{h}}{\partial x}l_x + K_{yy}\frac{\partial\hat{h}}{\partial y}l_y\right]w_i\,d\sigma = -\int_{\sigma} q_n w_i\,d\sigma \qquad (2.24)$$

where q_n is the normal flux across a unit area (length $\times$ height) of the vertical boundary σ. When the flux q_n is assumed constant along an element face of length L the integration of (2.24) will give the nodal allocations indicated in Figure 2.2. Thus, the user need only specify the volumetric flux at a node to represent the flux across a boundary. Fluxes due to pumping are introduced mathematically in the same way (see equation 2.11c). That is, the user specifies a volumetric flux at a node. To facilitate introduction of uniform infiltration the code allows for specification of a nodal infiltration flux. Areal integration to obtain volumetric flux is performed automatically by the code.

2.5.3 Third Type

Leakage boundary conditions are introduced by performing a substitution in (2.11c)

$$Q = k_L(h^t_{j,L} - h^t_{j,k}) \qquad (2.25)$$

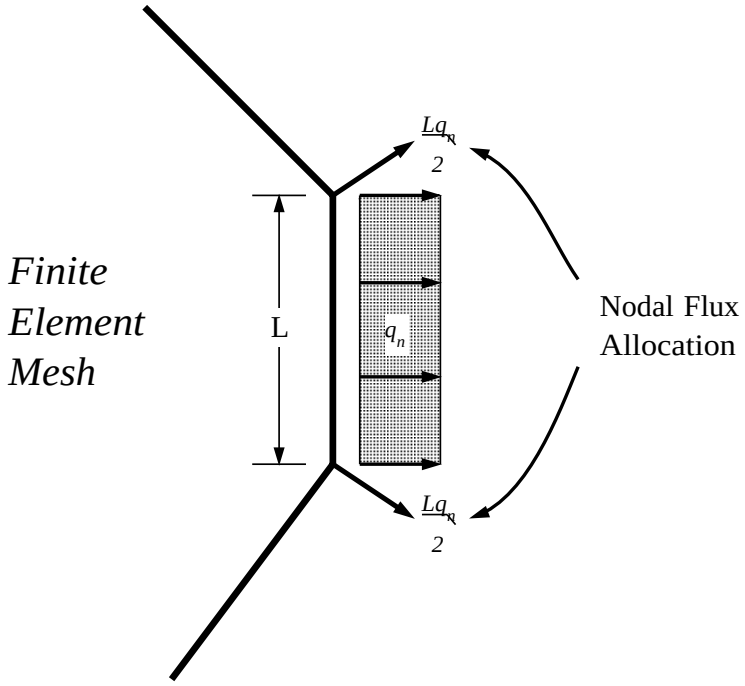


Figure 2.2: Nodal distribution of flux q across an element of length L. For linear elements, each node is equally weighted.

where $h^t_{j,k}$ is the unknown head at node j in layer k at time t; $h^t_{j,L}$ is the corresponding head at the leakage reference point (e.g., elevation of a lake); and k_L is the leakage conductance (hydraulic conductivity divided by distance). The conductance term is integrated over area and so implicitly represents a vertical leakage. However, by appropriate definition of the user specified constant, k_L, any directional leakage can be defined.

2.6 Water Table Conditions

Imposition of water table or unconfined aquifer boundary conditions requires the introduction of two boundary conditions applied to the water level in the top layer.

A schematic of the situation involving a moving water table, subject to infiltration of magnitude $R(x, y, t)$, is indicated in Figure 2.3. The fixed elevation of the upper boundary of the flow domain would be given by $z = z_{M+1}(x, y, t)$ and the actual elevation of the water table is denoted by $z = z_{WT}(x, y, t)$.

The first boundary condition states that the water level in the top layer (h_M) defines the aquifer thickness in the top layer ($z_{WT} - z_M$). Accordingly, at each node we require

$$(z_{WT})_i = (h_M)_i \tag{2.26}$$

It is assumed in the present work that the water table is always situated within the top layer ($k = M$). Thus, when the program computes the heads in the top layer, it checks if the nodal values of h_M is within the range of elevations given by

$$z_M \leq h_M \leq z_{M+1} \tag{2.27}$$

PTC issues a message if condition (2.27) is violated. It prints out the nodal location, the top thickness, etc., relevant to this violation. Execution is then terminated. Note that (2.27) must also be satisfied by the initial head conditions. Since z_{WT} defines the top layer thickness, which is present in the coefficient matrices, (2.26) introduces nonlinearity into the solution of the flow system of equations. This nonlinearity is accommodated by iterating on the second step of the split algorithm. The iterations are performed at a single time step, with all thickness related properties in (2.23) updated. When **h** changes by less than EPSILN, a given tolerance value, the iterations are terminated and the calculations are advanced to the next time step.

The second water table boundary condition describes the transient response of the water table to infiltration. Let S_y denote the specific yield near the water table. Then the equations relevant to this situation are given by

$$S_y \frac{\partial h}{\partial t} + K_z \frac{\partial h}{\partial z} = R \tag{2.28}$$

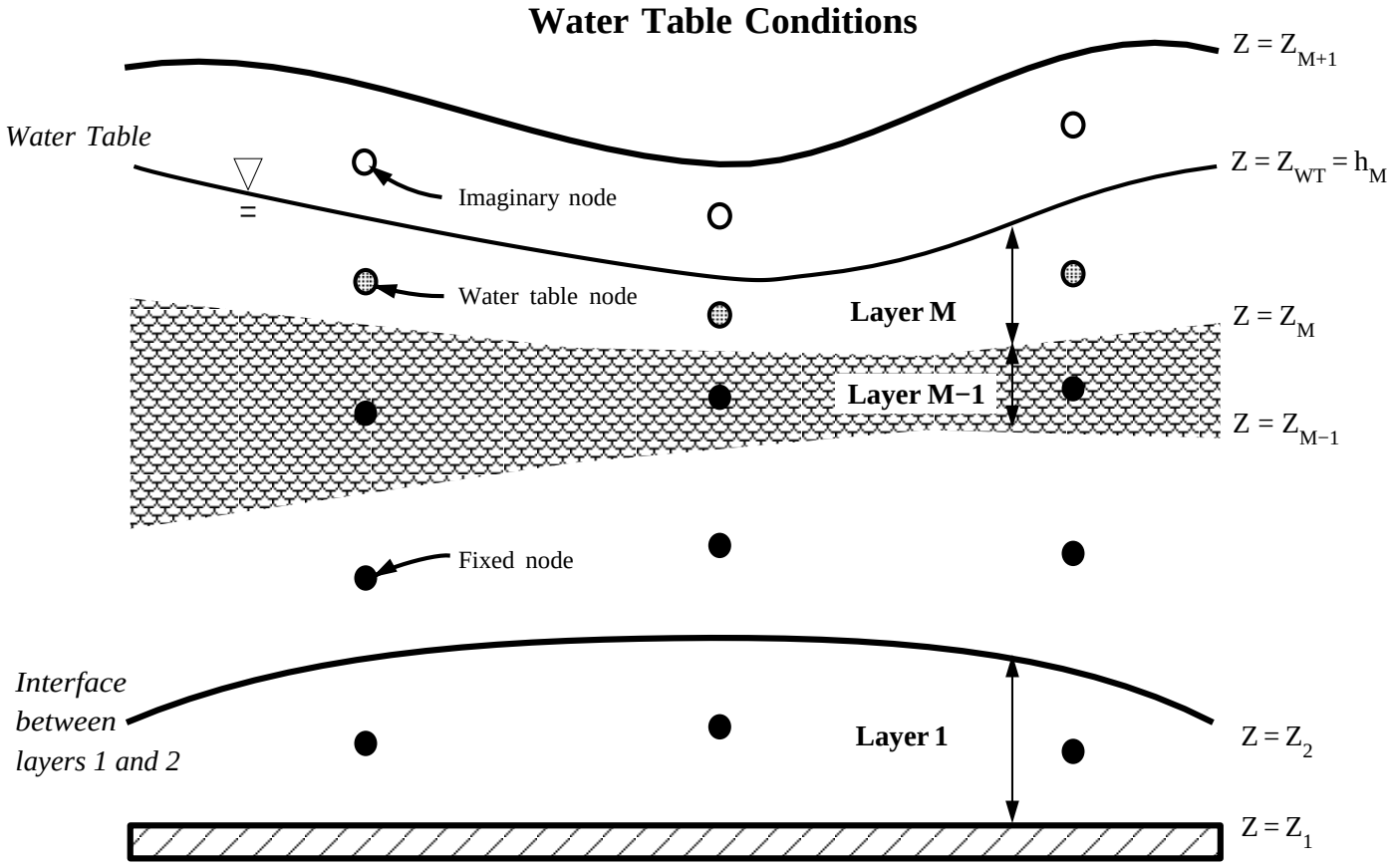


Figure 2.3: Water table and precipitation conditions at the top boundary. The surfaces denoted by $z = z_1$, $z = z_2$, etc., are fixed boundaries in space, and define layers. **Note**: the transient water table $z = z_{WT}$ must satisfy $z_M \leq z_{WT} \leq z_{M+1}$.

Utilizing the imaginary nodes of layer $(M+1)$, a finite difference version of (2.28) may be written as

$$\mathbf{h}_{M+1} = \mathbf{h}_M + \mathbf{q}_I{}^t - \mathbf{W}\frac{\partial \mathbf{h}_M}{\partial t} \tag{2.29}$$

where, for each node i of the top layer $(k = M)$, S_{y_i} is the porosity of the aquifer, R_i is the net vertical infiltration, Δz_i is the layer thickness, and $(K_z)_i$ is the vertical hydraulic conductivity

$$\mathbf{q}_I{}^t = \begin{Bmatrix} \Delta z_1 R_1/(K_z)_1 \\ \vdots \\ \Delta z_i R_i/(K_z)_i \\ \vdots \\ \Delta z_N R_N/(K_z)_N \end{Bmatrix} \tag{2.30}$$

$$\mathbf{W} = \begin{bmatrix} \Delta z_1 S_{y_1}/(K_z)_1 & 0 & \dots & 0 \\ 0 & \Delta z_2 S_{y_2}/(K_z)_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \Delta z_N S_{y_N}/(K_z)_N \end{bmatrix} \tag{2.31}$$

Equation (2.29) is employed as the boundary condition for equations (2.22) and (2.23), when dealing with the top layer $k = M$. The terms continuing the time derivatives $(\mathbf{W}\partial \mathbf{h}_M/\partial t)$ are entered as implicit (unknown) variables into equation (2.22) and (2.23), for the top layer only. The extended version of these equations is thus given as follows:

STEP 1: *(Modified with Water Table Conditions $(k = M)$)*

$$\Big(\mathbf{A}_k + \frac{(\mathbf{B}_D)_k}{\Delta t} + \frac{\mathbf{W}}{\Delta t}\Big)\mathbf{h}_k^{(t+\Delta t)^*} = \Big(\frac{(\mathbf{B}_D)_k}{\Delta t} + \frac{\mathbf{W}}{\Delta t}\Big)\mathbf{h}_k^t + \mathbf{P}[\mathbf{C}_D;\mathbf{h}]^t - \mathbf{f}_k^t + \mathbf{q}_I{}^t \tag{2.32}$$

STEP 2: *(Modified with Water Table Conditions $(k = M)$)*

$$\begin{aligned} \Big[\frac{(\mathbf{B}_D)_k}{\Delta t} + \frac{\mathbf{W}}{\Delta t}\Big]\mathbf{h}_k^{(t+\Delta t)} &- \mathbf{P}[\mathbf{C}_D;\mathbf{h}]^{(t+\Delta t)} + \mathbf{q}_I{}^{(t+\Delta t)} \\ &= \Big[\frac{(\mathbf{B}_D)_k}{\Delta t} + \frac{\mathbf{W}}{\Delta t}\Big]\mathbf{h}_k^{(t+\Delta t)^*} - \mathbf{P}[\mathbf{C}_D;\mathbf{h}]^t + \mathbf{q}_I{}^t \end{aligned} \tag{2.33}$$

2.7 Analytical Integration Procedures

All integrations occurring in **PTC** are performed by employing analytical integration procedures. This has resulted in considerable savings of computational effort.

Two simplifications made this improvement possible: first, the finite elements used here consist exclusively of linear quadrilaterals formed by four (4) corner nodes; second, the

aquifer properties (K_{xx}, K_{yy}, K_{zz}, S) are assumed to remain constant over an element. These properties may, of course, change from element to element.

The following brief description of the analytical integration procedure is to provide a sense of continuity for the general reader. Algebraic details and estimates of errors may be found in Babu and Pinder (1984b). The accompanying computer code incorporates several of the formulae given in this reference.

Consider a typical integral given by (2.11b)

$$B_{ij} = \iint_{\Omega} S w_i w_j \, dx \, dy$$

Noting that this integral over the entire cross section equals the sum of integrals over individual quadrilateral elements (e), the above integral is expressed as

$$\begin{aligned} B_{ij} &\equiv \sum_{(e)} \left(\iint_{(e)} S_{(e)} w_i w_j \, dx \, dy \right) \\ &= \sum_{(e)} \left(S_{(e)} \iint_{(e)} w_i w_j \, dx \, dy \right) \end{aligned} \tag{2.34}$$

since the storage S is assumed to remain constant over the given element (e).

The integration in the global (x, y) coordinate system is transformed into integration in the local (ξ,η) system via the well known transformation

$$\iint_{(e)} w_i w_j \, dx \, dy = \int_{-1}^{+1}\int_{-1}^{+1} w_i w_j \det J \, d\xi \, d\eta \tag{2.35}$$

where det J is the determinant of the Jacobian of the transformation

$$\det J \equiv \begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{vmatrix} \tag{2.36}$$

It is shown in Babu and Pinder (1984b) that, with ξ_i, η_i, α_i, β_i, α_0, and β_0 constants depending on element (e),

$$\begin{aligned} w_i &\sim (1 + \xi\xi_i)(1 + \eta\eta_i) \\ \det J &\sim (1 + \alpha_0\xi + \beta_0\eta) \\ \frac{\partial w_i}{\partial x} &\sim \frac{(1 + \alpha_i\xi + \beta_i\eta)}{\det J} \qquad \text{and so on.} \end{aligned} \tag{2.37}$$

Substitution of (2.37) into (2.35) yields integrals of the form

$$\int_{-1}^{+1}\int_{-1}^{+1} \Big[(1 + \alpha_0\xi + \beta_0\eta)(1 + \xi\xi_i)(1 + \eta\eta_i)(1 + \xi\xi_j)(1 + \eta\eta_j) \Big] \, d\xi \, d\eta \tag{2.38}$$

This integral can be evaluated in a straightforward manner. Results such as (2.38) above, combined with element parameters such as S in (2.34), are summed over all elements to derive the values of B_{ij}. Analogous procedures are applied to other integrals occurring in all finite element equations.

If the element (e) happens to be a rectangle (or a parallelogram), then $\alpha_0 = \beta_0 \equiv 0$, giving det $J \equiv 1.0$ in (2.37). This introduces further simplications into (2.38). Therefore, different subroutines in **PTC** perform the appropriate integrations.

2.8 Mass Balance of Fluid Flow

The governing equations described in the previous sections on groundwater flow are based in part upon the principle of the conservation of mass. Due to the approximations inherent in the numerical solution of the governing equations, exact mass balance is not possible; however, conscientious time and space discretization can produce an adequate mass balance. **PTC** provides the capability for performing a mass balance check which is independent of the solution of the governing equations. This is accomplished by directly calculating the mass flux across the boundary and the change of mass within the model domain. The net flux across the domain boundaries must equal the change of mass within the domain for mass balance to be satisfied.

PTC calculates a volume balance as a surrogate for a mass balance. The two approaches are equivalent, as the fluid density is assumed to be constant. The change in volume of fluid at a node is equal to the change in head at the node, multiplied by the area associated with the node (equal to one quarter of the area of each of the elements adjoining that node) and the storage coefficient associated with the node. The sum of these nodal changes in volume is the total change in volume.

The volumetric flux across the boundary has a number of components. These include specified volumetric fluxes, either at boundaries or at well points, computed fluxes which result from leakage from adjoining water units, and fluxes associated with specified head nodes. The finite element equation corresponding to a node contains a term which describes the volumetric flux at that node (the second integral in equation (2.8)). This equation is discarded for specified head nodes when solving the flow equations. Given the head solution for the entire domain, this equation can be used to back-substitute for the flux corresponding to the specified head node.

Details of output from the mass balance computations are provided in Chapter 4.

Chapter 3

THREE DIMENSIONAL CONTAMINANT TRANSPORT

Chapter 2 described in some detail the procedure for computing the transient hydraulic head $h(x, y, z, t)$ by solving the flow equation (1.1). The present section briefly outlines analogous procedures for solving the contaminant transport equation (1.3).

3.1 Governing Equations

At the outset, Darcy velocities V_x, V_y and V_z are computed as element-averaged quantities. The transport equation (1.3) is

$$\begin{aligned}\frac{\partial}{\partial x}\Big[D_{xx}\frac{\partial c}{\partial x}+D_{xy}\frac{\partial c}{\partial y}+D_{xz}\frac{\partial c}{\partial z}\Big]+\frac{\partial}{\partial y}\Big[D_{yx}\frac{\partial c}{\partial x}+D_{yy}\frac{\partial c}{\partial y}+D_{yz}\frac{\partial c}{\partial z}\Big]\\ +\frac{\partial}{\partial z}\Big[D_{zx}\frac{\partial c}{\partial x}+D_{zy}\frac{\partial c}{\partial y}+D_{zz}\frac{\partial c}{\partial z}\Big]+Q(c^w-c)\\ -\Big(V_x\frac{\partial c}{\partial x}+V_y\frac{\partial c}{\partial y}+V_z\frac{\partial c}{\partial z}\Big)-\theta[1+E(c)]\Big(\frac{\partial c}{\partial t}\Big)=0.\end{aligned} \tag{3.1}$$

The dispersion terms in (3.1) are defined (following Burnett and Frind [1987]):

$$\begin{aligned}D_{xx}&=(\alpha_L V_x^2+\alpha_T V_y^2+\alpha_V V_z^2)/V+D_M\\ D_{yy}&=(\alpha_T V_x^2+\alpha_L V_y^2+\alpha_V V_z^2)/V+D_M\\ D_{zz}&=(\alpha_V V_x^2+\alpha_V V_y^2+\alpha_L V_z^2)/V+D_M\\ D_{yx}&=D_{xy}=(\alpha_L-\alpha_T)V_xV_y/V\\ D_{yz}&=D_{zy}=(\alpha_L-\alpha_V)V_yV_z/V\\ D_{zx}&=D_{xz}=(\alpha_L-\alpha_V)V_zV_x/V\end{aligned} \tag{3.2}$$

and the remaining terms are

D_M is the molecular diffusion coefficient, generally small [L^2/T],
α_L is the longitudinal dispersivity [L],
α_T is the horizontal transverse dispersivity [L],
α_V is the vertical transverse dispersivity [L],
V is the magnitude of the velocity vector [L/T] ($V \equiv \sqrt{V_x^2 + V_y^2 + V_z^2}$),
c is the chemical concentration at the point (x, y, z) at time t [M/L^3],
θ is the porosity of the aquifer [dimensionless],
$E(c)$ is the function representing chemical adsorption properties (see Section 3.5),
Q is the (source/sink) strength of pumping [1/T] ($Q \equiv Q_i \delta(x-x_i)\delta(y-y_i)\delta(z-z_i)$),
Q_i is the volumetric injection/discharge rate [L^3/T] at point (x_i, y_i, z_i),
c^w is the concentration of the pumped fluid at point (x_i, y_i, z_i),
$\delta()$ is the Dirac delta function.

For all cases of withdrawal by a pump ($Q_i \leq 0$), we assume that the concentration of the withdrawn (fluid) water c^w at the pump equals the concentration of the surrounding ambient water c. Thus, at all discharging pumps, the term $Q(c^w - c) \equiv 0$ in (3.1). The code therefore retains the terms with Q in (3.1) only when the pump is injecting ($Q_i > 0$) the solute, with concentration c^w representing the concentration of the injection fluid.

3.2 Solution Technique

The solution procedure is identical to that described in Chapter 2 for solving for head (x, y, z, t). We form the Galerkin finite element equations by first multiplying the left hand side of (3.1) with the linear basis function $w_i(x, y)$, $i = 1, 2, \ldots, N$, and integrating by parts (by Green's theorem) over the (x, y) cross section of the flow domain.

This results in the following system of N equations for N nodes in each layer

$$\begin{aligned}
&\iint_\Omega \left[D_{xx}\frac{\partial c}{\partial x} + D_{xy}\frac{\partial c}{\partial y} + D_{xz}\frac{\partial c}{\partial z} \right] \frac{\partial w_i}{\partial x} + \left(D_{yx}\frac{\partial c}{\partial x} + D_{yy}\frac{\partial c}{\partial y} + D_{yz}\frac{\partial c}{\partial z} \right) \frac{\partial w_i}{\partial y} \Big] \, dx\, dy \\
&\quad + \iint_\Omega \left[V_x\frac{\partial c}{\partial x} + V_y\frac{\partial c}{\partial y} + V_z\frac{\partial c}{\partial z} + \theta[1 + E(c)]\frac{\partial c}{\partial t} \right] w_i \, dx\, dy + Q(c - c^w)_i \qquad (3.3) \\
&\quad = \int_\sigma \left(D_n \frac{\partial c}{\partial n} \right) w_i \, d\sigma + \iint_\Omega \left[\frac{\partial}{\partial z} \left(D_{zx}\frac{\partial c}{\partial x} + D_{zy}\frac{\partial c}{\partial y} + D_{zz}\frac{\partial c}{\partial z} \right) \right] w_i \, dx\, dy
\end{aligned}$$

where $(c - c^w)_i$ represents the concentration difference between ambient and injected fluid at node i. The integral on σ denotes the dispersive flux normal to the boundary of the cross section of the flow domain.

For the concentration c itself, the following approximation is introduced

$$c \equiv c(x,y,z,t) \sim \sum_{i=1}^{N} c_i(z,t) w_i(x,y) \tag{3.4}$$

so that c_i denotes the concentration at node i.

The next step follows the procedure for the flow equation. We substitute (3.4) into (3.3); use finite differences for the vertical and time derivatives; employ lumping of matrices (as described in Section 2.3.1) that contain the time derivatives as well as the derivatives in the vertical coordinate z; and finally, transfer all terms with z derivatives to the right hand side. The resulting matrix equations have the following structure:

$$\begin{aligned} \mathbf{A}_k \mathbf{c}_k + \mathbf{B}_k \frac{\partial \mathbf{c}_k}{\partial t} + \mathbf{Q}_k (\mathbf{c}_k - \mathbf{c}_k^w) \\ = \mathbf{f}_k + [\mathbf{D}_k^+ (\mathbf{c}_{k+1} - \mathbf{c}_k) - \mathbf{D}_k^- (\mathbf{c}_k - \mathbf{c}_{k-1})] - \mathbf{M}_k \frac{\partial \mathbf{c}_k}{\partial z} \end{aligned} \tag{3.5}$$

The matrices introduced in (3.5) have the following definitions (for a typical layer k)

$$\begin{aligned} \mathbf{A}_k &\equiv \iint_\Omega \Big[D_{xx} \frac{\partial w_i}{\partial x} \frac{\partial w_j}{\partial x} + D_{yy} \frac{\partial w_i}{\partial y} \frac{\partial w_j}{\partial y} + D_{xy} \Big(\frac{\partial w_i}{\partial x} \frac{\partial w_j}{\partial y} + \frac{\partial w_i}{\partial y} \frac{\partial w_j}{\partial x} \Big) \\ &\quad + \Big(V_x \frac{\partial w_j}{\partial x} + V_y \frac{\partial w_j}{\partial y} \Big) w_i \Big]_k \, dx\, dy \\ \mathbf{B}_k &\equiv \iint_\Omega \theta [1 + E(c)]_k w_i \, dx\, dy \\ \mathbf{D}_k^{\pm} &= \iint_\Omega \frac{2 (D_{zz})_k (D_{zz})_{k\pm1}}{\Delta z_k [(\Delta z_{k\pm1})(D_{zz})_k + \Delta z_k (D_{zz})_{k\pm1}]} \, w_i \, dx\, dy \\ \mathbf{M}_k &\equiv \iint_\Omega \Big[V_z w_i + D_{zx} \frac{\partial w_i}{\partial x} + D_{zy} \frac{\partial w_i}{\partial y} \Big]_k \, dx\, dy \\ \mathbf{f}_k &\equiv \int_\sigma \Big(D_n \frac{\partial c}{\partial n} \Big)_k w_i \, d\sigma \end{aligned} \tag{3.6}$$

$$\mathbf{Q}_k \equiv \begin{bmatrix} Q_1 \delta(z - z_N) & 0 & \dots & 0 \\ 0 & Q_2 \delta(z - z_N) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & Q_N \delta(z - z_N) \end{bmatrix}_k \quad \mathbf{c}_k \equiv \begin{Bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{Bmatrix}_k \quad \mathbf{c}_k^w \equiv \begin{Bmatrix} c_1^w \\ c_2^w \\ \vdots \\ c_N^w \end{Bmatrix}_k$$

The $\mathbf{B}_k$, $\mathbf{D}_k^{\pm}$, and $\mathbf{M}$ matrices are lumped diagonal matrices and the $\mathbf{A}_k$ matrix is banded nonsymmetric. The $\mathbf{f}_k$ vector contains the dispersive flux across the vertical boundary σ of the horizontal region Ω for layer k, and the $\mathbf{Q}_k$ diagonal matrix contains pumping/injection

rates. The vector of nodal concentrations of the ambient fluid is denoted by $\mathbf{c}_k$ and the vector of concentrations at the pumps is denoted by $\mathbf{c}_k^w$.

A well known technique called upstream weighting is introduced when evaluating the convective terms containing V_x and V_y in $\mathbf{A}_k$ of (3.5) or (3.6). Such weighting on w_i is known to enhance the accuracy of the solution. Therefore, we define the derivative $\partial c/\partial z$ in (3.5) in terms of nodes situated upstream in the vertical direction. Thus,

$$\frac{\partial c}{\partial z} = \begin{cases} 2(\mathbf{c}_{k+1} - \mathbf{c}_k)/[\Delta z_k + (\Delta z)_{k+1}] & \text{if } V_z < 0 \\ 2(\mathbf{c}_k - \mathbf{c}_{k-1})/[\Delta z_k + (\Delta z)_{k-1}] & \text{if } V_z > 0 \end{cases} \tag{3.7}$$

3.3 Final Form

It is now obvious that the chemical transport equation (3.5) very much resembles the flow equation (2.14). As a result of the lumping procedure, the transport equation for layer k ($k = 1, 2, \ldots, M$) is given by

$$(\mathbf{A} + \mathbf{Q})_k \mathbf{c}_k + \mathbf{B}_k \frac{\partial \mathbf{c}_k}{\partial t} = \mathbf{f}_k + \mathbf{Q}_k \mathbf{c}_k^w + \mathbf{U}(\mathbf{c}_k) \tag{3.8}$$

where the components Q_i of the diagonal matrix $\mathbf{Q}$ of pump rates are absent from both sides for all discharging pumps ($Q_i < 0 \rightarrow Q_i \equiv 0$), and where

$$\mathbf{U}(\mathbf{c}_k) \equiv \begin{cases} \mathbf{D}_k^-(\mathbf{c}_{k-1} - \mathbf{c}_k) + \left(\mathbf{D}_k^+ - \dfrac{2\mathbf{M}_k}{\Delta z_k + \Delta z_{k+1}}\right)(\mathbf{c}_{k+1} - \mathbf{c}_k) & \text{if } V_z \le 0 \\ \mathbf{D}_k^+(\mathbf{c}_{k+1} - \mathbf{c}_k) + \left(\mathbf{D}_k^- + \dfrac{2\mathbf{M}_k}{\Delta z_k + \Delta z_{k-1}}\right)(\mathbf{c}_{k-1} - \mathbf{c}_k) & \text{if } V_z \ge 0 \end{cases} \tag{3.9}$$

The same two-step time splitting procedure summarized for the flow equation in (2.22) and (2.23) is applied to (3.8), leading finally to the solution algorithm for the transport problem of (3.1).

STEP 1: $k = 1, 2, \ldots, M$

$$\left(\mathbf{A}_k + \mathbf{Q}_k + \frac{\mathbf{B}_k}{\Delta t}\right)\mathbf{c}_k^* = \mathbf{f}_k^t + \mathbf{Q}_k \mathbf{c}_k^w + \frac{\mathbf{B}_k}{\Delta t}\mathbf{c}_k^t + \mathbf{U}(\mathbf{c}_k^t) \tag{3.10}$$

STEP 2: $k = 1, 2, \ldots, M$

$$\mathbf{A}_k \mathbf{c}_k^* + \left(\mathbf{Q}_k + \frac{\mathbf{B}_k}{\Delta t}\right)\mathbf{c}_k^{(t+\Delta t)} - \mathbf{U}(\mathbf{c}_k^{(t+\Delta t)}) = \mathbf{f}_k^t + \mathbf{Q}_k \mathbf{c}_k^w + \frac{\mathbf{B}_k}{\Delta t}\mathbf{c}_k^t \tag{3.11}$$

$$\left(\mathbf{Q}_k + \frac{\mathbf{B}_k}{\Delta t}\right)\mathbf{c}_k^{(t+\Delta t)} - \mathbf{U}(\mathbf{c}_k^{(t+\Delta t)}) = \left(\mathbf{Q}_k + \frac{\mathbf{B}_k}{\Delta t}\right)\mathbf{c}_k^* - \mathbf{U}(\mathbf{c}_k^t) \tag{3.11a}$$

For computational reasons, the code works with a slightly modified version of (3.11). The unknown variable considered in (3.11) will be the difference $(\mathbf{c}_{\mathrm{DIFF}}) \equiv (\mathbf{c}^{(t+\Delta t)} - \mathbf{c}^{t})$ for all layers. This change adds to the efficiency of the numerical scheme, and also permits the use of quite large time steps (Δt).

$$\left(\frac{\mathbf{B}_k}{\Delta t} - \mathbf{U}\right)\left(\mathbf{c}_k^{(t+\Delta t)} - \mathbf{c}_k^{t}\right) = \frac{\mathbf{B}_k}{\Delta t}\left(\mathbf{c}^{*} - \mathbf{c}_k^{t}\right) \qquad \text{(3.11b)}$$

3.4 Boundary Conditions

PTC accommodates the specification of two types of boundary conditions on the transport equation; specified nodal concentration and specified convective flux. The boundary conditions are zero dispersive flux on the vertical side boundaries and zero total contaminant flux on the top and bottom boundaries.

Specified nodal concentrations are defined by the user as described in Chapter 4. Specified convective fluxes are defined by indicating a concentration associated with the volumetric fluid fluxes in the program input. Similarly, indirectly specified convective flux is introduced through the concentration associated with the leakage fluid in the third type flow boundary conditions.

3.5 Equilibrium or Adsorption Isotherms

The term adsorption is employed whenever some of the chemical, during its transport via the moving fluid, adheres to the soil grains. The following discussion is taken from Geraghty and Miller (1979).

It is assumed that adsorption takes place instantaneously, implying that the solute in the fluid is in equilibrium with the adsorped component. This means that there exists an equilibrium or adsorption isotherm of the form

$$I \equiv I(c) \qquad \text{(3.14)}$$

The following options are presented in this model:

Linear isotherm: $I = \alpha_1 c$,
Freundlich isotherm: $I = \alpha_2 c^{\alpha_3}$,
Langmuir isotherm: $I = \alpha_4 c/(1 + \alpha_5 c)$,

where the isotherms mentioned in (3.14) and the $E(c)$ function introduced in equations (3.1) and (3.2) are related by

$$E(c) \equiv \frac{\rho}{\theta}\frac{\mathrm{d}I(c)}{\mathrm{d}c}. \qquad \text{(3.15)}$$

In this, ρ is the bulk density of the solid matrix and θ is the porosity. $I(c)$ in (3.14) is representative of the mass of the adsorped material, and $\mathrm{d}I/\mathrm{d}c$ is a distribution coefficient. A more general discussion of these isotherms may be found in Perry's Chemical Engineering Handbook (1963).

The linear isotherm,

$$\frac{\mathrm{d}I(c)}{\mathrm{d}c} = \alpha \qquad (\alpha \ \text{ constant}), \tag{3.16}$$

is the simplest case as far as computations are concerned. This type of isotherm is often used in modelling adsorption of radioactive species.

The Freundlich isotherm,

$$\frac{\mathrm{d}I(c)}{\mathrm{d}c} = \beta c^{\gamma} \qquad (\beta, \gamma \ \text{ constants}), \tag{3.17}$$

is used to model adsorption in dilute solutions with low concentrations, to model adsorption of organic chemicals, and to model adsorption cases wherein the identity of the solute remains unknown (Treybal, 1968). Note: For this isotherm, if $\gamma < 0$ and $c \to 0$, then $E(c) \to \infty$. In such situations, care must be exercised to prevent exponential overflow in the computer code.

The Langmuir isotherm is

$$\frac{\mathrm{d}I(c)}{\mathrm{d}c} = \frac{\mu\delta}{(1+\delta c)^2} \qquad (\mu, \delta \ \text{ constants}), \tag{3.18}$$

where δ is the adsorption equilibrium constant, and μ is the mass of solute adsorped per unit mass of the adsorbent when all active sites for adsorption are covered. Since its introduction in 1916, this type of isotherm has been widely used in view of its versatile applicability to many adsorption processes (Smith, 1970).

From the computational point of view, the linear isotherm in (3.16) requires no extra effort. However, for both the Freundlich (3.17) and Langmuir (3.18) isotherms, the matrices $\mathbf{A}_k$ defined in (3.6) must be recomputed for every time step. This means that the overall computational burden is substantially increased in the solution procedures for the transport problem.

The code uses the following general analytical expression in the evaluation of adsorption terms:

$$E(c) \equiv \frac{\alpha c^{\beta}}{(1+\gamma c)^2}. \tag{3.19}$$

From this general expression are derived the three special cases, linear ($\beta = \gamma = 0$), Freundlich ($\gamma = 0$), and Langmuir ($\beta = 0$). The user supplies the values of α, β and γ whenever the adsorption option is exercised. Note: α must include bulk density ρ when specified in the data input to **PTC**! Porosity θ is included internally to the code.

3.6 Mass Balance of Contaminants

PTC performs a mass balance on the computed concentration solution. The mass balance is performed in much the same way as that described for flow in Section 2.8.

The notable addition is that both convective and dispersive mass fluxes must be accounted for. Convective flux occurs at nodes at which fluid crosses the boundary. This fluid flux is computed as described in Section 2.8. The convective flux then is simply the product of the fluid flux and the contaminant concentration of the fluid.

Dispersive mass flux is accounted for by deriving an expression for dispersive boundary flux. This is done by a second application of Green's Theorem to the finite element integral equation (3.3). With this new form the dispersive flux at nodes can be computed based on the computed concentrations. The results of this mass balance analysis are output as described in Chapter 4.

References

1. Babu, D.K., G.F. Pinder and D.K. Sunada, "A Three-dimensional hybrid finite element-finite difference scheme for groundwater simulation", **Proc. 10th IMACS World Congress on System Simulation and Scientific Computing.** pp 292-294, 1982.
2. Babu, D.K. and G.F. Pinder, "A finite element-finite difference alternating direction algorithm for three dimensional groundwater transport", **Adv. in Water Resources 7**, pp 116-119, 1984a.
3. Babu, D.K. and G.F. Pinder, "Analytical integration formulae for linear isoparametric finite elements", **Intl. Journal of Num. Methods in Engineering 20**, pp 1153-1166, 1984b.
4. Burnett, R.D. and E.O. Frind, "Simulation of Contaminant Transport in Three Dimension 2. Dimensionality Effects", **Water Resour. Res., 23**(4), pp 695-705, 1987.
5. Geraghty and Miller, Inc., "Interim Report for Groundwater Module. Groundwater Model documentation", prepared for Arthur D. Little, Inc. 1979.
6. Lapidus, L. and G.F. Pinder, "Numerical Solution of Partial Differential Equations in Science and Engineering", John Wiley, New York, 1982.
7. Perry, J., "Chemical Engineers' Handbook", 4th Edition, McGraw-Hill, New York, pp 16.5-16.11, 1963.
8. Pinder, G.F. and W.G. Gray, "Finite Element Simulation in Surface and Subsurface Hydrology", Academic Press, New York, 1977.
9. Smith, J.M., "Chemical Engineering Kinetics", 2nd Edition, McGraw-Hill, New York, pp 291-293, 1970.
10. Treybal, R.E., "Mass Transfer Operations", 2nd Edition, McGraw-Hill, New York, pp 506-508, 1968.

Chapter 4

USING PTC

4.1 Introduction

The Princeton Transport Code (**PTC**) is the result of contributions by many individuals at Princeton University. A two-dimensional code for groundwater flow and contaminant transport was originally developed by G. F. Pinder and W. G. Gray. The extension of the code to three space dimensions was carried out by D. Krishna Babu who also substituted Gaussian quadrature with analytical integration of basis functions. Auli Niemi revised the code and adapted it to the IBM-PC. Additional boundary condition capability and internal checking of the computed solution for mass balance were incorporated by David Ahlfeld. Stuart Stothoff implemented the command-driven input/output structure, inspired by a similar structure developed by Roger Page, and developed the data verification, time step generation, and graphics output routines.

4.2 Program Capabilities

PTC is a hybrid finite element/finite difference groundwater flow and contaminant transport simulator. Present capabilities include:

- Two dimensional simulations
- Fully three dimensional simulations
- Transient groundwater flow
- Transient contaminant transport
- Transient boundary conditions
- Time step generation capabilities
- Saturated confined flow
- Saturated water table flow
- Linear, Freundlich and Langmuir adsorption isotherms
- Mass balancing capabilities for groundwater flow

- Mass balancing capabilities for contaminant transport
- Specified head conditions at any node
- Specified groundwater flux conditions at any node
- Specified groundwater leakage conditions at any node
- Specified concentration conditions at any node
- Specified convective flux conditions at any node
- Specified contaminant leakage conditions at any node
- Specified rainfall conditions at every element in the top layer
- Piecewise constant parameters by element
- Quadrilateral and triangular elements
- Analytic elemental integrations
- Input of parameters by element or by node
- Command-driven and user-defined input structure
- User-requested data echoing
- Data editing capabilities
- Data looping capabilities
- Data generation capabilities
- Data verification capabilities
- Output for graphics packages

4.3 PTC Revisions

The Princeton Transport Code is continually undergoing revision, both minor and major. As these modifications are made, the input structure may change. The following denote revisions in the release of January of the indicated year.

New or modified from 1992 to 1993 are:

- memory is partitioned at run-time
- arrays are passed as arguments, not in common
- vertical transverse dispersion is separate from horizontal transverse dispersion
- adsorption input is defined by elements
- **setfdon/setfdoff** sets flow dimensioning
- **setmdon/setmdoff** sets mass transport dimensioning
- **setdfrac** sets fraction of nodes considered dirichlet
- **setsson/setssoff** sets specified steady state
- **drgparam/draparam** includes vertical transverse dispersivity

Previous revisions include:

- triangular elements [1992]
- major simplification of numeric code structure [1992]

- third-type boundary condition input redefined [1992]
- data generation/checking [1990]
- command language [1989]
- graphics output facilities [1989]
- transient and third-type boundary condition [1988]
- mass balance facilities [1987]
- IBM PC version [1986]

4.4 Program Structure

PTC is written in standard FORTRAN 77, with one known exception, explained in the section on dimensioning and compilation.

The program is divided into four program modules, comprising two modules with subroutines and two modules which are included into the other modules. Two of the modules are directly related to operating **PTC**, and the other two are standalone modules which are devoted to providing a standardized set of input commands.

The program modules are listed as they appear on the distribution disk, with a description of the subroutines:

4.4.1 Module PTC.FOR

This module contains all **PTC**-specific routines. Many of the major routines are listed below.

PTC	Main program for **PTC**.
PRESET	Initialization of file information, print information, and variables used in calculations.
PSETLG	Set dimensions and allocate arrays.
INTADM	Assure coefficient arrays allocated.
INTDPT	Initialize a pointer to an array.
INTPNT	Set up a pointer description.
FINISH	Standard end point for execution.
PTCIN	Command center for all **PTC** input.
RGENIN	Command center for distributed variable input.
GEOMAK	Generate calculated input values for geometry and mesh.
UNKMAK	Generate calculated input values for stresses and initial conditions.
GEOCHK	Verify validity of geometry and mesh.
SIMCHK	Verify problem dimensioning.
PARMAK	Generate calculated input values for parameters.
GNTIME	Generate initial time step size.
GNTEND	Generate final time from time step parameters.

PTCOUT	Command center for all **PTC** output.
SOLOUT	Solution output for time-varying solution values.
DBAND	Triangular decomposition by Cholesky method.
SBAND	Back substitution for Cholesky method.
TRID	Tridiagonal matrix solver using Thomas's algorithm.
SOLVE	Non-symmetric matrix solver using Gaussian elimination.
SAVMAT	Fast input/output for solution matrices.
PTCGRF	Command center for all **PTC** graphical output.
PGRVAL	Center for graphics output of parameter values.
SIM	Command center for time stepping.
FLOW	Solution of the flow equation at a time step.
BM2VM	Convert a banded matrix to compressed storage.
VM2BM	Convert a compressed matrix to banded form.
FLASM1	Assembly of the horizontal flow coefficient matrix for a layer.
FLASM2	Reassembly of the horizontal flow coefficient matrix for a layer.
FLHORZ	Solution of the horizontal flow equation at a time step.
FLVERT	Solution of the vertical flow equation at a time step.
WTUPDT	Update information dependent on the water table.
MASS	Solution of the transport equation at a time step.
MSHORZ	Solution of the horizontal mass transport equation at a time step.
MSVERT	Solution of the vertical mass transport equation at a time step.
MSASM1	Assembly of the horizontal transport coefficient matrix for a layer.
MSASM2	Incorporation of retardation in the horizontal transport coefficient matrix for a layer .
MSASM3	Solve horizontal flow equations for a layer.
ADSORB	Adsorption isotherm calculation.
STNNPE	Determine if an element is a triangle or quadrilateral.
FHAFRC	Accumulate flux terms in horizontal flow equations.
MHAFRC	Accumulate flux terms in horizontal transport equations.
ASBDMT	Assemble elemental coefficients into a banded matrix.
MKHFLO	Make horizontal coefficient matrix generation for flow.
MKHCON	Make horizontal coefficient matrix generation for transport.
UDHFLO	Make horizontal coefficient update for flow.
MKEVEL	Make elemental velocities from head field.
NQDET	Make coefficients for quadrilateral integrations.
NQUWT	Make upstream weighting coefficients for quadrilaterals.
NQSYM	Do symmetric quadrilateral integrals.
NQ2SYM	Do more quadrilateral integrals.
NQ1DER	Do unsymmetric quadrilateral integrals.
NQ1DU	Do more unsymmetric quadrilateral integrals.

NRSYM	Do symmetric rectangle integrals.
NR2SYM	Do more rectangle integrals.
NR1DER	Do unsymmetric rectangle integrals.
NR1DU	Do more unsymmetric rectangle integrals.
NTDET	Make coefficients for triangle integrations.
NTUWT	Make upstream weighting coefficients for triangles.
NTSYM	Do symmetric triangle integrals.
NT2SYM	Do more triangle integrals.
NT1DER	Do unsymmetric triangle integrals.
NT1DU	Do more unsymmetric triangle integrals.
ELAREA	Analytical area integration for quadrilateral elements.
STFDIR	Store elemental coefficients for flow mass balance.
STMDIR	Store elemental coefficients for transport mass balance.
MASSBL	Command center for mass balance calculations.
PMACCM	Accumulation routine for mass balance calculations.

4.4.2 Module IOTOOLS.FOR

This module contains a package of miscellaneous routines, designed to handle most of the low level input/output requirements independent of the main program. There are nearly forty routines included, of which only the main or most frequently used routines are described below.

CMNDDR	Command center for command input interpretation and file manipulations.
CMARPR	Argument prompting, for terminal input.
CMF1PR	List-directed integer and character input.
CMF2PR	Formatted character input.
CMSLRD	List-directed input for all data types, with specified number of values.
CMULRD	List-directed input for all data types, with number of values specified as part of input.
IGEN	Data generation for integer arrays.
RGEN	Data generation for single and double precision arrays.
RDGEN	Data generation for all data types.
RCLEAR	Array zeroing for single and double precision arrays.
ICLEAR	Array zeroing for integer arrays.
IOSET	Initialization of IOTOOLS variables, file units, and **cfile** information.

4.4.3 Module PTCCOM

This module contains parameter statements, dimension statements, and common block declarations for the PTC.FOR module. This is placed inline at compilation time at every place INCLUDE 'PTCCOM' statement is found. If this module is altered, module PTC.FOR must be recompiled and relinked.

4.4.4 Module INCOMM

This module contains parameter statements, dimension statements, and common block declarations for the IOTOOLS.FOR module. This is placed inline at compilation time at every place INCLUDE 'INCOMM' statement is found. If this module is altered, module IOTOOLS.FOR must be recompiled and relinked.

4.5 Dimensioning, Modification, and Compilation

The **PTC** code is supplied as a set of FORTRAN modules, which must be compiled and linked before the code may be run. It is expected that minor modifications to the code will be required due to differing problem sizes, differing output needs, and differing compiler requirements. This section discusses the modifications which may be required.

4.5.1 Compile-time Program Dimensioning

The **PTC** code partitions one large array into problem-dependent chunks at run-time. This allows for the code to be compiled once, and run for differing problems without recompilation. It is recommended that the array is dimensioned as large as the particular computer can fit into memory. It may take a little experimentation to find the largest practical dimension for a particular computer.

Dimensioning of the array is done in one place, in PTCCOM. The array is automatically distributed to all other subroutines during compilation via the "INCLUDE" statement.

4.5.2 Code Modification

Compiler-Dependent Modification

All of the dimension statements and common blocks are grouped into two separate program modules, and included at compilation time using the non-ANSI standard "INCLUDE" statement. Most compilers accept some form of this statement; however, if your compiler does not accept the statement, it is necessary to replace all occurrences of the INCLUDE statement with an appropriate copy of the included file prior to compilation. For such compilers, it is

good strategy to maintain a pristine copy of the original files and only alter a copy of the files.

WARNING: some older versions of the MICROSOFT compiler for the IBM-PC support the INCLUDE statement in a slightly different format than that indicated in this documentation. Use of the MICROSOFT compiler may force the alteration of the "INCLUDE" statements at each location they appear in the code.

In the PTCCOM file, allowance is also provided for a space-saving procedure, which may be desirable on machines with small amounts of memory. Floating point numbers can be changed from double to single precision, thus using half of the storage space, with a resulting loss of accuracy. The code may run faster in single precision mode; however, it is highly recommended that double precision be retained wherever feasible to maintain accuracy. These options are indicated at the top of the PTCCOM module.

In addition, the format of OPEN statements, used for file definition, differs among varying compilers, and these may need to be tailored to a compiler. For example, certain compilers do not accept OPEN statements, and alternative methods of specifying designated files are required. The user must consult the compiler documentation in these instances.

Terminal-Dependent Modification

Unfortunately, there is no standard way that the terminal is treated in FORTRAN; thus interaction with **PTC** through the terminal is compiler dependent and operating system dependent. A parameter, `zopsys`, is defined in the INCOMM module in order to handle peculiarities in a graceful manner. The available options are listed in the INCOMM module – if none of these seem appropriate, define `zopsys` to be a blank string.

The initial **cfile** for **PTC** may be specified as a default file, typically "ptc.run", or the capabilities of DOS and unix operating systems may be exploited for terminal input redirection. With this capability, the file for input is signified by the string "`< filename`", where `filename` is the file to be used.

The redirection capability is rather useful when several data sets are used, since copying files into the default file is avoided, but there is no standard way in FORTRAN for forced interaction with the terminal when input and output redirection is in effect. Alternatives have been coded, using the `zopsys` flag, but it may be necessary to directly define the forced terminal input and output unit numbers, using the compiler documentation as a reference. These variables are specified in the IOSET subroutine in the IOTOOLS module.

If input redirection is desired, the variable `yttred` in the PRESET subroutine should be set to true; otherwise it should be set to false.

PTC has historically been run with `zopsys` blank and `yttred` off, and this is the way it is configured for distribution.

Graphics-Dependent Modification

The PTCGRF and PGRVAL subroutines are provided to output information to a graphics file, including mesh and nodal function values. Any commercially available graphics package should be able to use the output from the subroutines, given proper formats from the data set. If the output is to be tailored to a particular graphics package, the output from PTCGRF and PGRVAL may be tailored to this package by changing the marked write statements.

4.5.3 Compilation

PTC has to date been successfully compiled under IBM FORTVS, UNIX f77 (on DEC, SUN, and Silicon Graphics workstations), and Ryan-McFarland Professional FORTRAN for the IBM-PC, all using Princeton facilities. There have also been rumors that compilation has successfully occurred on a Macintosh, on a Cray supercomputer, and using the MICROSOFT compiler on an IBM-PC; these have not been verified by any of the authors of the code.

Each compiler has slightly different standards for warning and error messages. In general, warning messages may be ignored, particularly messages about unused variables. Warning messages about inconsistent subroutine usage should be checked, but can usually be ignored – some of the IOTOOLS routines have been known to set off this type of message.

The authors of the code are always interested in hearing of strange messages. If a compiler issues an error message, or a warning message that seems important, please let us know. If you find a fix for the message, this is also of great interest.

4.6 Model Input and Verification Issues

There are several steps involved in creation and verification of a model of a hydrogeologic regime, some of which must be repeated time after time. One must have available a simulator which is capable of modelling the regime of interest. One must create a conceptual model which represents reality to the suitable level of abstraction. One must provide this model to the simulator without error in translation. And one must interpret the output from the simulator.

Procedures for creation of a suitable conceptual model is a topic beyond the scope of this text. However, assuming that **PTC** is capable of simulating the selected conceptual model, there are a number of steps which may be followed to ensure that this conceptual model is correctly simulated. This section discusses issues involved in providing information to **PTC** and retrieving information from **PTC**.

4.6.1 Parameter Definition

Solving a groundwater problem using a numerical code requires the specification of many site-specific parameters. These parameters can, in theory, be specified either for each node or

for each element. Because the numerical solution requires slightly less computational effort when parameters are assumed constant over an element, and since all integrations in this code are performed analytically, these parameters are assumed to be specified as constants over each element. In the event that a user decides to specify these parameters at each node, the code automatically computes the arithmetic averages of these values over each element and uses the calculated average as the actual input data.

4.6.2 Node Numbering

Careful examination of the finite element equations used either for flow or for transport will reveal that the coefficient matrix created by these equations will have a banded structure.

The bandwidth of non-zero elements in the coefficient matrices plays a significant role in the amount of computational effort required to solve (2.20), and it is important to minimize this quantity. The bandwidth is a function of the maximum difference between nodal numbers occurring on the same element. Defining the worst difference as the maximum of these differences over the entire mesh, the full bandwidth is equal to one plus twice the worst difference, and the half bandwidth is one plus the worst difference. One soon learns from experience that the minimum bandwidth is generally obtained by numbering sequentially in the direction of the smallest number of elements in the model.

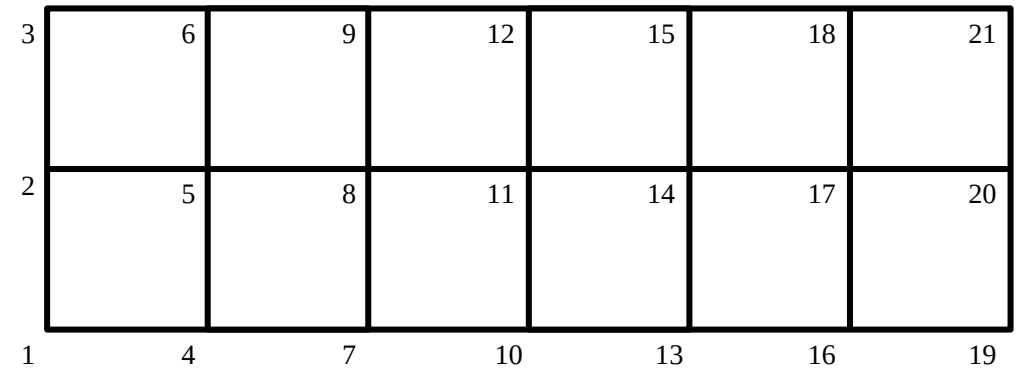
To illustrate this point, reference is made to Figure 4.1, where a mesh consisting of 12 elements and 21 nodes is numbered in two ways. In case A, numbering across the smallest mesh dimension, the half-bandwidth is five as dictated by the maximum nodal difference of eight. In case B, numbering across the largest mesh dimension, the half-bandwidth is nine.

When the incidence generation option is used, node numbering always increases in sweeps along the x axis; however, an internal renumbering occurs so that the equation numbering is optimal. When inputting a mesh, it is possible to explicitly specify the internal renumbering, perhaps using some bandwidth minimizer.

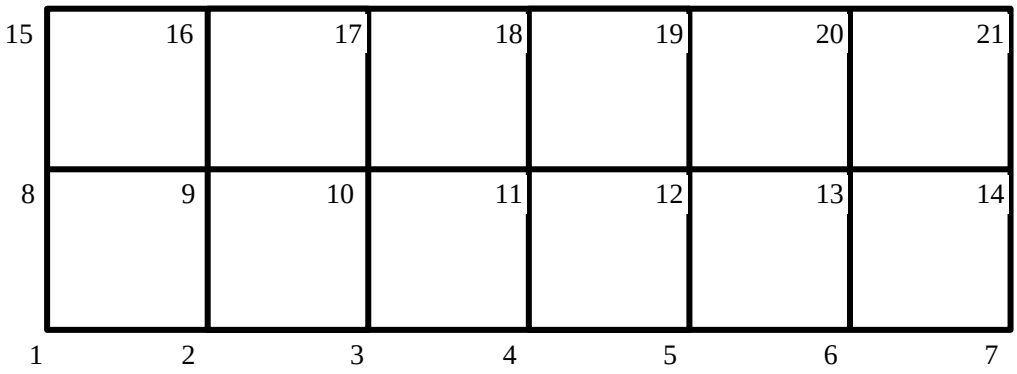
4.6.3 PTC Error Checking

A great number of pieces of information must be input into **PTC**, and inevitably some of the information is incorrectly or inappropriately specified in early stages of model creation. Accordingly, it is exceedingly important to verify all of the input to **PTC**, and a number of methods for this may be pursued.

PTC is able to help with certain basic error checking tasks. **PTC** checks that dimensions and flags are compatible with the input data whenever appropriate. **PTC** also performs a series of checks prior to running *SIM*, including simple mesh verification and parameter validity checks. These last checks make sure that each element is valid geometrically, that all layers have positive thickness, and that each material property is positive. When running water table simulations, every time step the water table is checked to make sure it lies fully



Half bandwidth is 5



Half bandwidth is 9

Figure 4.1: Two identical element configurations with markedly different bandwidths, dictated by the choice of nodal numbering scheme.

within the top layer. If any of these checks are violated, the offending element or node is flagged in the output and execution is halted.

These checks are useful for finding gross errors in input, but obviously only provide a minimum assurance. There is no assurance that elements are hooked up correctly, that parameters are physically reasonable, or that applied stresses are in the correct location. Thus, the checks are only a starting point for a complete verification effort.

PTC echoes the input of any parameter upon request, as many times as requested, which allows for cross-checking of the parameter against expected input at several points during the course of data input. This is again useful for finding gross input errors, but can be extremely tedious in general.

4.6.4 Graphical Error Checking

It has been found that graphical examination of the mesh, parameters, initial conditions, and solutions is a most effective method for verifying the input of the conceptual model. It may be immediately evident that a well is in the wrong location upon examination of a contour plot of head, for example. Similarly, a contour map of the parameters will often reveal incorrect specifications.

With this observation in mind, **PTC** provides methodology for outputting the parameters and solutions to a file in a format suitable for plotting.

Since it is impossible to appropriately prespecify the correct format for every plotting package, the actual write statements responsible for outputting to the file may need to be modified before compilation in order to be suitable for an individual application. These write statements are confined to the subroutines PTCGRF and PGRVAL in module PTCOUT. The usage of the graphics commands are explained in a separate section.

4.6.5 Mass Balance Error Checking

PTC provides mass balance capabilities, both for flow and transport simulations. The mass balance output provides an excellent indication of possible time and space discretization errors, as a poor choice of time step or grid size will lead to a poor mass balance.

Usage and interpretation of the mass balance facilities is provided in a separate section.

4.7 Input/Output Philosophy

PTC is a command-driven package, and is rather flexible in input/output structure. The input structure is designed to be like a programming language itself, as the usage of the program may vary widely among different applications. Accordingly, **PTC** provides powerful facilities for specifying input and requesting output. Note that all output must be specifically requested; no output will be generated without a request.

A number of files are addressed simultaneously when running **PTC**. In order to keep the files straight, each file is referred to using a unit number. **PTC** uses an independent variable to maintain the unit number for each potential input and output file, but these variables may point to the same file. The *OPEN* command is used to let **PTC** know which file a given unit points to.

The independent routines contained in the program module "IOTOOLS" and the associated file "INCOMM" are used as a filter between **PTC** and the data input. These routines act as a black box to the main portion of **PTC**, handling all of the low-level details of basic file manipulation, command screening, and data preprocessing. All input enters the program through one of these routines.

The "IOTOOLS" routines have two input units and two output units, which are specified independently from the remainder of **PTC**. These units may be changed and restored easily. Accordingly, input to **PTC** may be from any file or combination of files. Note that interactive input and output is treated as a special case of a file, enabling prompting for arguments and data.

The "IOTOOLS" input files are termed the **cfile** and the **dfile**. The **cfile** handles commands and arguments to the commands. The **dfile** handles data input, where the distributed data facilities are used, as described in the section on distributed data.

The "IOTOOLS" output files are termed the **cecho** file and the **efile** file. The **cecho** file echoes commands as input (default is no echo). The **efile** is the error output file (default is no error output), where irregularities in the commands or data are noted as they occur. These irregularities are from the standpoint of the "IOTOOLS" package, which are strictly read-related; **PTC** performs higher level model assembly and checking prior to simulations. It is a good idea to reset these immediately upon starting execution.

PTC maintains an independent list of dedicated units for various output functions. Each of these units may point to any file, again with the terminal being merely a special case. Up to six files at once may receive output, as there are dedicated units available for output of flow solutions, transport solutions, graphical output, flow mass balance output, transport mass balance output, and formatted output.

As few as two files may be used for all of the input and output combined, as each dedicated unit may point to the same file, or an entire string of files may be addressed in sequence.

Every variable which has a value for each node or element (distributed variables) may be specified in a number of ways, and each specification may be combined with other specifications. For example, a parameter may be specified over the entire model, layerwise, elementwise, and nodewise. When specifying parameters elementwise and nodewise, data generation may be used for repetitious patterns of specification; or a parameter may be specified for specific elements.

All parameters may have a scaling factor applied upon input. As **PTC** assumes that all units are self consistent, such a scale is indispensible in converting from one set of units to another. For example, rainfall data may be converted from in/year to m/sec merely by

applying the appropriate scale factor.

All data input may be read without a specified format (list-directed). In addition, distributed variables may be read with a specified format.

Commenting within the input files is allowed.

4.8 Input Files

PTC expects input from two files, which are be termed the **cfile** and the **dfile**. The **cfile** is the file from which commands and arguments to commands are read, and the **dfile** is the file from which large blocks of data are read. Of course, these files need not have different unit numbers, and the same file may be read for all input information. In general, the **dfile** is only used when the distributed data facilities are in use, and is explained in the section on distributed data facilities. The separate **dfile** unit allows large blocks of data to be kept in their own separate (independently named) file. This is most useful when independent data is generated for each element, perhaps with some kriging algorithm, and it would be more convenient to maintain several sets of data for each parameter.

PTC starts with both of these files set to unit 1, which has a default name of "`PTC.RUN`". In order to change this default file name, it is necessary to change the OPEN statement in subroutine PRESET, and recompile the program.

In order to change standard input files during runtime, the commands *CFILE*, *DFILE*, *CRETURN*, *DRETURN*, *TERM*, and *GO* are useful. As might be expected, these commands manipulate which unit the commands and data are to be read from. In addition, the commands *DFILE* and *DFROMC* specify the format in which the data is expected. The use of *CFILE* and *DFILE* commands assumes that the appropriate file has been previously opened using an *OPEN* command.

In order to have interactive input, the commands *TERM* and *GO* are provided. In **term** mode, the terminal is the expected source of commands and data. In addition, prompting for arguments is provided and entering a blank line reissues the previous command. The *GO* command ends **term** mode, and the previous **cfile** and **dfile** specifications are resumed.

Commands which manipulate files include the FORTRAN calls *OPEN*, *CLOSE*, and *REWIND*, with related calls *BACKREC* and *SKIPREC*. Using these commands, any desired file may be manipulated, and it may be accessed as often as desired.

Usage of all of these commands is specified in the section describing IOTOOLS commands.

4.9 Output Files

PTC outputs information as requested by the user. Each component of the program which generates output sends the output to a file as directed by a dedicated variable unit number. For example, the mass balance output for flow simulations is sent to a unit number specified

by the variable NUOMBF. If this unit number is less than 1, output is suppressed for that particular function. The default unit for all output is unit 6, which is the unit number reserved for terminal output. Note that unit 5 is reserved for terminal input, and may not be used for output. Units 15 through 19 inclusive are reserved for fast matrix storage, and unit 91 is reserved for a looping manipulation file; these must not be used as output units.

The file which output is sent to must be explicitly opened using the *OPEN* command prior to output being generated, and the unit variable for the output must be set to correspond to this file. The only exception is terminal units 5 and 6, which must not be opened. If the unit variable is set to some unopened file, the operating system will create a default file (typically "fort.3" or some variant).

Each of the variable unit numbers may be independently set to the same unit, which will result in all output being sent to the file the unit points to. Default unit numbers are initialized in the PRESET subroutine. These may be altered by the user and the program recompiled, or the unit numbers may be reset during execution.

There are three classes of output files.

One class outputs information periodically during the simulation procedure, including solutions and mass balances. The commands for setting the frequency of output and naming the file to be output to are described in the sections on OUTPUT CONTROL and GRAPHICS OUTPUT COMMANDS.

Another class is used for querying **PTC** before and after simulations on the present state of the variables. The commands for querying for this information are described in the sections on OUTPUT QUERY COMMANDS and GRAPHICS OUTPUT COMMANDS.

The last class is used for fast input/output during simulations when minimizing storage requirements, such as occurs for large problems on small computers. This last class generates binary scratch files during runtime. The scratch files are only opened if incore option is turned off using the *RDCONTRL* command; these files should not exist following execution, but other output must not go to these files. Accordingly, the unit numbers specified for these files must remain out of bounds for all other unit variables. The default unit numbers are units 15 through 19 inclusive; these are set in subroutine PRESET.

The dedicated unit variables for output are as following:

4.9.1 Unit Variables for Simulation Output

NUOHD	Formatted solution output for head. This is used for output from FLWOUT during flow simulations.
NUOCC	Formatted solution output for concentration. This is used for output from MASOUT during transport simulations.
NUOMBF	Formatted mass balance output. This is used for output from MASSBL during flow simulations.

`NUOMBM` Formatted mass balance output. This is used for output from MASSBL during transport simulations.

`NUOGRF` Formatted graphics output. Query commands prefixed with 'GR' go to this unit, as well as graphics output during flow and transport simulations.

4.9.2 Unit Variables for Query Output

`NUODAT` Formatted data output. Query commands prefixed with 'WR' or 'EC' go to this unit, as well as error and warning messages.

4.9.3 Unit Variables for Fast Input/Output

`NUSCF1` Scratch file for flow simulations.

`NUSCF2` Scratch file for flow simulations.

`NUSCTB` Scratch file for water table flow simulations.

`NUSCM1` Scratch file for transport simulations.

`NUSCM2` Scratch file for transport simulations.

4.10 Command Structure

Most **PTC** commands stand alone, as for flag setting commands or query commands, or require small amounts of additional information, such as a unit number or time step information. On the other hand, creating a large finite element model necessarily requires the specification of an assortment of values for each node and each element.

Use of the "IOTOOLS" package provides a uniform structure for all input. In addition, the RGENIN routine, which uses the "IOTOOLS" package, is a higher level command parsing routine providing a uniform structure for input of all variables which must be defined for each element or each node (distributed variables).

4.10.1 Command Protocol

The "IOTOOLS" package expects certain protocols to be followed in interpreting commands and associated input, and will complain or do unexpected things if not humored. **PTC** performs input processing external to the "IOTOOLS" package, but maintains the same set of protocols for consistency's sake.

A command is actually a sequence of cards, including an optional sequence of comment cards, a command card, perhaps a series of argument cards, and perhaps a series of distributed data cards. The comment cards, command card, and argument cards must all

reside in the **cfile**. The distributed data cards must exist in the **dfile**. If the **dfile** is the same as the **cfile**, distributed data cards occur directly after the argument cards.

Sample command sequences are provided in a separate section below to illustrate the procedures.

Comments

Comments are signaled by a "*" (star) in the first column of a card, when a command card would otherwise be read. This card is ignored. In addition, any space on a command card or argument card following the complete command or set of arguments is valid for commenting. Further, for "distributed data" with a specified number of groups per card, space after the final group on each card is valid for commenting.

WARNING: attempting to put comments before the end of the sequence of arguments may result in a FORTRAN error message and crash **PTC**.

Command Words

A command word is a string of eight left-justified characters, INCLUDING ANY TRAILING BLANKS, on a card. Any characters after the eighth are ignored, allowing room for comments. For some compilers and operating systems, a tab character may not be interpreted as a blank, resulting in mysterious interpretation problems.

Command words are converted to a uniform case before interpretation, so command words may be any combination of upper and lower case.

If a command word or a comment is expected, and the first eight characters of a card do not fit the description of either a comment or a command word, the word is flagged as it is echoed but otherwise it is ignored. Accordingly, if an input command is not found or an incorrect amount of data is found, the first eight characters of each card will be flagged until a correct command is found.

WARNING: if a command card is encountered while variables from a previous command remain to be specified, the program is unable to interpret the situation properly and may crash, depending on the compiler. In such cases, using the data echoing facilities will provide clues to the cause of the malfunction.

Arguments

On the card(s) following the command, a number of arguments may be required, depending on the particular command, specifying such information as layer number and parameter values. All arguments are input list-directed, and so may be spread over several cards.

Note that trailing arguments may remain unchanged by inserting a slash "/" instead of the first of these trailing arguments.

Distributed Data Lines

Following all of the arguments, data input may be required, depending on the command, where "data" is used in a specific sense. "Data" refers to only the information processed by the distributed data facilities. The distributed data facilities allow a variety of input structures, ranging from a specified input value for each node or element through global specification and interpolated input.

Description of the distributed data routines is found below, in the distributed data section.

PTC only uses the distributed data facilities for input of distributed variables, and only when requested.

4.11 "IOTOOLS" Commands

This section describes the specification of all requests handled using the "IOTOOLS" package. These requests include file manipulation and data preprocessing requests.

In this and subsequent sections, commands are italicized (i.e. *OPEN*); any corresponding arguments are in a typewriter font (i.e. `MUNIT`).

4.11.1 Run Commands

STOP	Exit **PTC**.
QUIT	Exit **PTC**.
END	Exit **PTC**.
TERM	Switch to terminal control (unit 5 input, unit 6 output).
GO	Resume **cfile** control with same **cfile** and **dfile** that were in effect prior to *TERM* command.

4.11.2 File Manipulation Commands

OPEN	Open a file.
`MUNIT`	Unit number for the file.
`ZFILE`	File name (enclosed in quotes).
CLOSE	Close a file.
`MUNIT`	Unit number for the file.
REWIND	Rewind a file.
`MUNIT`	Unit number for the file.
BACKREC	Backspace a specified number of records in a file.
`MUNIT`	Unit number for the file.
`NREC`	Number of records to be backspaced.

SKIPREC	Skip a specified number of records in a file.
MUNIT	Unit number for the file.
NREC	Number of records to be skipped.

For each of the above commands, if the input file unit number is 0, the current **cfile** unit number is used.

4.11.3 Unit Manipulation Commands

One of the strengths of the "IOTOOLS" package is the ability to "program" the data set. The unit manipulation commands act something like subroutines, the BACKREC and SKIPREC commands can act something like goto statements, and the DO/OD commands (presented in another section) provide looping capability.

In order to simulate the action of subroutines, **PTC** maintains information on the nesting of each level of **cfile**. Current depth of nesting is 20, which is controlled by the parameter MNSUB in the INCOMM module. If this is not sufficient, increase MNSUB, recompile the IOTOOLS module, and relink **PTC**.

CFILE	Specify unit number for command input.
MUNIT	Unit number for command input file.

This command has the effect of changing the current **cfile** ("calling a subroutine") to MUNIT. The unit which is referred to must have been previously opened. The current **cfile** is returned to with the first *CRETURN* command in the new **cfile**. Note that a *CFILE*/*CRETURN* sequence, followed by another *CFILE* to the same unit, will cause execution to resume below the *CRETURN* command in the new file, unless file manipulation commands have been performed on the new unit in the interim.

DFILE	Specify unit number and format for data input (distributed data facility usage only). This command has the effect of changing the current **dfile**.
MUNIT	Unit number for data input file.
ZFMT	Data input format, enclosed in quotes. If the first character is a "*" (star), list-directed input is assumed.
NGPL	Number of data groups per line. Specifying a positive number on list-directed reads (ZFMT is a "*") will create a space for comments on each line, after the last group on the line. A negative value on list-directed reads means that distributed data is read as a stream; data generation must be off and no comments are allowed until following the last input value.

For each of the above commands, if the input file unit number is 0, the current **cfile** unit number is used.

DFROMC	Specify format for data input, and use the current command file unit for the data file unit (distributed data facility usage only).
`ZFMT`	Data input format, enclosed in quotes (see *DFILE* description).
`NGPL`	Number of data groups per line (see *DFILE* description).
EFILE	Specify unit number for "IOTOOLS" error messages.
`MUNIT`	Unit number for file (default no output).
CRETURN	Return to the **cfile** from which the last *CFILE* command was issued. This has the effect of making a file like a subroutine.
DRETURN	Return to the **dfile** from which the last *DFILE* or *DFROMC* command was issued.

4.11.4 Input Data Manipulation Commands

INDEXON	Indexing enabled for distributed data and index integer expected.
INDEXOFF	Indexing disabled for distributed data and index integer not expected.
DGENON	Data generation enabled in distributed data routines and data generation integer active.
DGENOFF	Data generation disabled in distributed data routines (data generation integer disabled).
SCALE	Specify scale factor, and enable data scaling.
`DSCALE`	Scale factor applied to all subsequent real numbers input with scaling enabled. The default value is 1.0.
SCALEON	Enable data scaling with the (previously defined) scale factor.
SCALEOFF	Disable data scaling.

4.12 Simulation Variables

This section describes the input of variables which control model simulations. Each of these variables is completely specified from the **cfile**.

Each of these variables is specified using the appropriate command word, followed by an argument card.

In the following sections, a command word or portion of a command word is italicized and in capital letters. The associated arguments are indented below the command.

WARNING: the problem dimensions must be among the first variables which are specified, as input of the distributed variables depends on this prior specification. Array allocation flags MUST be set before even the problem dimensions are specified. Array dimensioning flags configure memory to allow flow or transport; if the flags are off, memory is released for other purposes.

RDTITLE	Problem title.

ZTITLE	A character string, no more than 80 characters, enclosed in quotes.
SETFDON	Set flow array dimensioning on.
SETFDOFF	Set flow array dimensioning off.
SETMDON	Set mass transport array dimensioning on.
SETMDOFF	Set mass transport array dimensioning off.
SETDFRAC	Set maximum fraction of nodes that will be considered dirichlet nodes.
DIRFRC	Fraction of nodes in the most extreme layer.
RDDIMS	Problem dimensions.
NND	Number of nodes in problem.
NEL	Number of elements in problem.
NLY	Number of layers in problem.
RDCONTRL	Simulation option control (a 1 turns the option on, a 0 turns it off).
NFLOW	Flow calculation request (always on when mass balance required).
NVEL	Velocity calculation request (on if transport is on).
NCONC	Transport calculation request.
NCORE	Incore calculation request (only turn off if **PTC** is compiled with the limited storage option).
NMB	Mass balance calculation request.
RDDEBUG	Debug option control (a 1 turns the option on, a 0 turns it off).
NMAT	Full matrix print request.
RDWTABLE	Water table option control (a 1 turns the option on, a 0 turns it off).
NTABLE	Water table calculation request.
NITER	Maximum number of water table iterations in a time step (input but not used if water table calculation off).
EPSILN	Water table convergence criterion (input but not used if water table calculation off).
RDWEIGHT	Upstream weighting value.
UPVALUE	A value between 0.0 and 1.0, with 0.0 implying no upstream weighting.
RDDIFUSN	Molecular diffusion value.
DIFUSN	A (small) value, used in transport simulations, which is only important where velocity is negligible.
SIM	Perform data checks and begin the simulation specified by the current variable set.

4.12.1 Time Marching Control

Since the time scales for flow and transport simulation are often quite different, **PTC** maintains two separate sets of time stepping parameters. At any particular point in time, the only information which is needed for the simulator is the actual size of the time step. Since

the size of the time step should be directly related to the change in the solution during the time step, larger time steps may be taken as transient behavior dies out.

The strategy which has been adopted attempts to minimize the cost of creating and solving systems of equations as much as possible, since this is the largest portion of the computational burden. **PTC** divides the system matrix into a space-dependent portion and a time-dependent portion. These portions are generated and stored, and neither need be recalculated until the boundary conditions change. Each time the time step changes, the two portions are combined into a single system matrix. This system of equations is inverted, and the resulting matrix is stored. Until the time step changes size again, the solution for each time step is obtained by backsubstituting into the inverted matrix, which is a relatively inexpensive operation.

Of these steps, inverting the system matrix is by far the largest burden. Accordingly, it is advantageous to run the problem for several constant time steps, thereby not having to invert the system matrix, then changing the time step; this can then be repeated as necessary.

With the observation that the cost of inverting the transport matrix is substantially greater than the cost of inverting the flow matrix, the flow time step size is chosen as the primary time unit. The transport time step size is treated as a fraction of the flow time step size, with some number of transport steps per flow step; however, a separate set of time accumulation variables are maintained, which keep track of flow time and transport time from start of **PTC** and start of *SIM*.

In order to allow the time step to vary, **PTC** requires information on the initial time step size, the number of steps the time step size is valid for, and the increase in time step size after these steps. Either the initial time step size may be specified, or it may be calculated from the time step change information and a total length of time for the simulation.

If the time step becomes too large, perhaps in an attempt to reach steady state, the quality of the solution deteriorates. Accordingly, there is user-specifed cutoff number of steps beyond which the time step will not increase in size. In addition, a "steady-state" rate of change of head with respect to time may be specifed. Once the largest nodal value for the domain falls below the "steady-state" value, the current head distribution is used until a change in boundary conditions, distributed parameters, or initial conditions is imposed. Note that time stepping continues as specified, but the cost of recalculating the flow solution is eliminated.

GNTIME	Calculated time step control.
`ITMAX`	Total number of flow time steps in problem.
`ITCHNG`	Number of flow time steps between time step size resets.
`ITCHMX`	Number of flow time steps after which time steps stop changing size.
`NMSPF`	Number of mass transport steps per flow step.
`CHNG`	Multiplier for time step size (applied to `DELT` every `ITCHNG` time steps until `ITCHMX` reached).
`TLENG`	Total length of time for time step size calculation.

RDTIME	Specified time step control.
ITMAX	Total number of flow time steps in problem.
ITCHNG	Number of flow time steps between time step size resets.
ITCHMX	Number of flow time steps after which time steps stop changing size.
NMSPF	Number of mass transport steps per flow step.
DELT	Initial time step size.
CHNG	Multiplier for time step size (applied to DELT every ITCHNG time steps until ITCHMX reached).
RDSTEADY	Flow simulation steady state.
DHDTSS	Rate of change of head with respect to time which is to be considered steady state (a negative value forces continued calculation).
SETSSON	Force the current head distribution to be considered steady state.
SETSSOFF	Release the steady state override set by SETSSON.

4.13 Distributed Variables

This section describes the input of all independent variables, initial conditions for all dependent variables, all boundary conditions, model geometry, and mesh specification. Input of each of these variables is through the routine RGENIN which is the driver for flexible specification. RGENIN parses command words, verifies that a valid command word has been specified, calls the required routines in the "IOTOOLS" package to read the data, averages the data (if requested), and inserts the data into the appropriate places in the array.

Prior to input of any distributed variables, model dimensions must be specified using the *RDDIMS* command (described above). Prior to use of the averaging facilities, coordinate and incidence information must be specified.

Most of the distributed variables are defined by layer, and the layer must be specified for these variables. In general, if it makes sense that a layer must be specified, the layer flag is input; if it is not necessary to specify the layer, the layer flag is not looked for and must not be input. For example, all material parameter information must have the layer specified, but coordinates are valid for each layer and so a layer must not be specified.

4.13.1 Command Word Parsing for Distributed Variables

Each command word for the input of distributed parameters comprises three components. The first two characters must be **dr** or **de**, which signals that distributed variables are about to be read or edited. The third character may be **a**, **d**, **g**, **e**, **n**, or **s**. The remainder of the command word is a mnemonic for the parameter, such as 'condx' for x direction conductivity. For example, a distributed parameter command might be `dracondx`.

In certain cases, some options are not appropriate for a particular distributed variable. Input of an invalid option will result in the command not being recognized.

Note that since editing of an array is allowed, several commands may be issued in succession for the same array, using different data structure keys. This is useful to specify global values, then change specific values afterwards.

Edit Key

The second character is a trigger for clearing the array prior to inputting data. An **r** will clear the array first, then read the input data in. An **e** will edit the array, leaving the contents of the array intact unless overwritten by the new input.

Data Structure Key

The third character is a trigger for the way input data is supplied by the user and subsequently assigned to the distributed variable arrays.

An **a** will take as arguments the layer which is to be input and one representative group of data. All nodes or elements in the layer are then assigned this group. Note that this procedure has the effect of overwriting the present contents of the array in the specified layer.

In the case where the variable is not defined for each layer, such as for rainfall, the layer specifier is not required.

A **g** is like the **a** command, except that it acts on all layers. Accordingly, no layer specifier is ever input. In the case where the variable is not defined for each layer, the **a** and **g** options are identical.

An **s** is like a selective **a** command. An **s** will take as arguments the layer which is to be input (if required), and one representative group of data, just like the **a** option. However, in the subsequent card(s), the specified elements or nodes the representative group will apply to is input. This is specified by listing those elements or nodes. The sequence of elements or nodes is terminated by a "/". These input values are assigned by element for variables defined elementwise (i.e. model parameters), and are assigned by node for variables defined nodewise (i.e. coordinates, elevations). No averaging is performed.

An **e** assumes that the information is supplied elementwise, and an **n** assumes that the information is supplied nodewise. If the information is required in the model for each element and the **n** option is specified, the input information is read into a temporary array, averaged appropriately, and placed into the required array. The converse is also true. Note that this averaging process will not work if the element coordinates and incidences have not been previously input. Detailed information on the input format for these options is provided in the section on the distributed data facilities.

A **d** specifies that input will be in the default mode for the variable, which is the **e** mode for variables defined for each element and the **n** mode for variables defined at each node.

4.13.2 Distributed Data Facilities

In cases where there is limited information on the parameters, detailed information may not be appropriate for the model, and the **a**, **g**, and **s** options may be adequate for specifying all data for the model. However, as model development proceeds, the information supplied to the model becomes more detailed. The distributed data facilities provide a uniform set of procedures for specifying detailed information to **PTC**, where each node or element may be independently specified using the **e** or **n** options.

Since detailed information is often the result of output from other programs, flexibility in input is highly desirable. This flexibility comes at the cost of complexity of input structure; the following describes the distributed data facilities. Several examples are provided afterwards to step through implementations of the facilities.

The information which is input using the distributed data facilities may come from a file other than the **cfile**; however, the file and the input format are specified in the **cfile**.

In the simplest case, the input may simply be a formatted or list-directed stream of numbers, implicitly assigned to each element in the input array in sequence. By setting the index flag on, the input stream may explicitly assign values to elements in the input array; elements may be reassigned values. By setting the generation flag on, the input stream may fill in elements within the input array using linear interpolation. And by setting another flag, space in the input stream may be reserved for comments.

All of these flags are set prior to the command for input. The flags remain unchanged until explicitly reset.

Only the **e** and the **n** cases of the Data Structure Key are handled using the distributed data facilities; the following flags are ignored when specifying data using the **a**, **g**, and **s** options.

Index Flag

If the index flag is on, an integer is required for each data group, indicating the element or node each data group applies to. If the index flag is off, this integer is inactive and it is assumed that the data groups start at the first node or element and go to the last, including each node or element in sequence.

The index state is set using the *INDEXON* and *INDEXOFF* commands.

Data Generation Flag

If the data generation flag is on, the second integer is active and provides an increment flag `NG`. If the flag is non-zero, linear interpolation occurs between the data values in the current and subsequent data groups. The interpolated values are inserted in every `NG`th spot in the array from the current to the subsequent data group. If the data generation flag is off, the second integer is inactive and no interpolation occurs.

Note that data generation is meaningless unless the index flag is on. The generation integer is read and ignored in this case.

The generation state is set using the *DGENON* and *DGENOFF* commands.

Data Format

Data may be read in list-directed format, or may be read formatted if desired. In addition, data may be read in as a single long string irrespective of grouping (for instance, several groups occupying one card) or have a specified number of groups per card. Specifying a number of groups per card allows the remainder of the card to be used as a comment area, which may be useful. These options are specified using the *DFILE* and *DFROMC* commands. Details on these options are provided below.

WARNING: combining list-directed input and data generation requires a specified number of groups per line, as the data is read into a relatively small buffer which will overflow otherwise.

Resumption of Command Mode

The end of data input and resumption of command mode is signalled in several ways.

If the index flag is on, a 0 for the first integer signals resumption of command interpretation (note that this is the only way to end data input if the data generation flag is on). This 0 should be followed by a slash "/" if reading list-directed. When reading with a format, a blank card takes the place of the 0; a 0 may be input for the index as well.

If data generation is off, data input is ended when the expected number of data groups (such as the number of elements or the number of nodes) have been read, and command interpretation resumes.

Examples

In all cases, the input stream contains the values for the array. In addition, extra integers may be present within the input stream, depending on the state of the index and generation flags.

In each of the subsequent examples, it is assumed that the **dfile** is already set to the same unit as the **cfile**, or it is set so during the example. If the **dfile** were not the same unit, the portion of the example following the command and the layer number would be in the **dfile**. The examples would be identical, except that the numbers (and termination signal) would physically be in another file.

As a simple example, assume that there are 5 elements in the problem, with one layer, and x direction hydraulic conductivity is to be specified. Further, element 1 is to receive the value of 10.0, element 2 is to receive the value of 15.0, and so on through element 5, which is to receive the value of 30.0.

In the simplest case, assuming that the problem dimensions have been specified and list-directed input is expected, the conductivity information may be specified by the following sequence:

```
indexoff
dgenoff
drecondx
1
10.0
15.0
20.0
25.0
30.0
```

Notice that no termination signal is needed; the end of input is signalled by the 5th value having been read. In this and the following examples, the "1" following the command is the layer number flag.

The same data may be input with the following sequence:

```
indexon
dgenoff
drecondx
1
1 10.0
5 30.0
3 20.0
2 15.0
4 25.0
0 /
```

Notice that a termination signal is needed, as is always the case when the index flag is on; the end of input is signalled by the 0 flag having been read. In this case, order is not important.

The same data may be input with the following sequence:

```
indexon
dgenon
drecondx
1
1 1 10.0
3 1 20.0
5 0 30.0
0 /
```

In this case, the data generation flag, the second integer, is the increment for linear interpolation. The 1 signals that every subsequent element receives a value; the 0 signals the end to interpolation. In this example, the element 3 specification is not required; the construction is perfectly valid, however.

Notice that the same termination signal is needed. Again, order is not important.

The same data may be input with the following sequence:

```
indexon
dgenon
drecondx
1
1 2 10.0
5 0 30.0
2 2 15.0
4 0 25.0
0 /
```

This construction demonstrates that interpolation may skip elements if desired. The first pair assigns values to the odd elements; the second pair assigns values to the even elements.

The same data may be input with the following sequence:

```
dfromc
'*' 1
indexon
dgenon
drecondx
1
1 2 10.0       * input for odd elements
5 0 30.0
2 2 15.0       * input for even elements
4 0 25.0
0 /
```

The commenting is enabled by specifying that exactly 1 group per line will be input, using the *DFROMC* command.

The same data may be input with the following sequence:

```
dfromc
'(i1,i2,f5.1,i4,i2,f5.1)' 2
indexon
dgenon
```

```
drecondx
1
1 2 10.0   5 0 30.0       * input for odd elements
2 2 15.0   4 0 25.0       * input for even elements
0                   /
```

The *DFROMC* command specifies that 2 groups per line is expected, and the FORTRAN format for these groups is specified.

Notice that when formatted input is used, the terminal / is not necessary, and cannot be inside the formatted section if present.

4.13.3 Input of Distributed Values

This section describes the distributed variables, which are input using the methods outlined above, unless otherwise specified (such as for mesh generation).

Mesh

This section lists all mesh information.

Incidences are input using the distributed parameter element (**e**) protocol, unless the generation option is used. The generation option assumes that, as far as connectivities are concerned, the modelled region forms a rectangle. Node and element numbering proceeds faster in the x direction; the equation number associated with each node maintains the optimal bandwidth.

Moving node and equation ordering information are input using the distributed parameter node (**n**) protocol. The moving node information is only used when the water table option is on.

Prior to a simulation, each element is checked to see if it is rectangular, in order to simplify integrations. An element is considered rectangular if the diagonals are the same length.

The full input command is given below for incidence input. Note that the layer flag must not appear when specifying incidences.

RDINCID	Nodes forming an element, in strict counterclockwise order, starting at any node (4 integers). If the final integer is zero, the element is a triangle.
GNINCID	Generate incidences.
`NGNELX`	Number of elements in x direction.
`NGNELY`	Number of elements in y direction.
RVINCID	Reverse the current order of the nodes forming each element.
RDWTNODE	Specify nodes which are considered water table nodes when the water table option is on. A 1 indicates water table node, a 0 indicates confined. All nodes are considered water table nodes by default.

RDROWORD Specify equation number corresponding to each node (default order is by node number). This allows for bandwidth minimization using external optimizers.

Geometry

This section lists all geometrical information. Coordinates are input once, and interfacial elevations are input for each interface, where an interface is a boundary between layers. Note that there is one more interface than there are layers, as the top and bottom must also be accounted for.

The following is the last portion of the command word and the corresponding input variable. Note that the layer flag must not appear when specifying coordinates.

COORS	X and Y coordinates of each node (2 numbers).
ELEV	Elevation of an interface between layers at a node. The bottom of the bottom layer is interface number 1.

In conjunction with the mesh generation option, the nodal coordinates may also be generated. This is specified as follows:

GNRECCOR	Generate coordinates in a rectangle along axis directions.
`DGX1`	X coordinate of node number 1.
`DGY1`	Y coordinate of node number 1.
`DGX2`	X coordinate of node number `NND`.
`DGY2`	Y coordinate of node number `NND`.

Any layer can be split into 2 or more layers, for computational resolution purposes. Each layer is identical to the original, except that it is a fraction of the thickness. In addition, any specified fluxes are a fraction of the original flux. This splitting is specified as follows:

SPLAYER	Split a layer into evenly discretized sublayers.
`MLAYER`	Layer which is to be split.
`NUMNEW`	Number of sublayers desired.

The new layers displace the original layer numbering upward, and the top `NUMNEW - 1` layers are discarded.

Distributed Parameters

This section lists all model parameters which are defined for each element. Each layer must be input separately, unless the **g** mode is specified.

The following is the last portion of the command word and the corresponding input variable.

CONDX	Hydraulic conductivity in the X coordinate direction [L/T].
CONDY	Hydraulic conductivity in the Y coordinate direction [L/T].
CONDZ	Hydraulic conductivity in the Z coordinate direction [L/T].
STOR	Storage coefficient [1/L] in confined layers, specific yield [dimensionless] in unconfined layers.
POROS	Effective porosity (volume void space/volume).
ELONG	Longitudinal dispersivity [L^2/T].
ETRAN	Horizontal transverse dispersivity [L^2/T].
EVERT	Vertical transverse dispersivity [L^2/T].
ADSOR	Adsorption coefficients α, β, and γ (3 numbers). These coefficients are defined in Equation (3.19). Note that α must incorporate bulk density!
PARAM	Group input of the x, y, and z conductivities, storage coefficient, porosity, longitudinal dispersivity, and horizontal and vertical transverse dispersivities for a layer or globally (8 numbers required in above order).
XVELO	Velocity in the X coordinate direction [L/T].
YVELO	Velocity in the Y coordinate direction [L/T].
ZVELO	Velocity in the Z coordinate direction [L/T].

Note that the group input of distributed parameters only accepts the **a** and **g** options.

Initial Conditions

This section lists all initial condition information. Initial conditions must be defined for the entire domain prior to the first call to *SIM*. Subsequent calls to *SIM* will have the initial condition information overwritten by the last solution calculated during computation. Each variable is defined at the nodes and is input layerwise.

The following is the last portion of the command word and the corresponding input variable.

INITH	Initial head [L].
INITC	Initial concentration.

Applied Stresses

This section lists all boundary condition information.

All boundary conditions are applied at the nodes during simulations. All flow fluxes are treated as equivalent point sources at the node, supplied in units of length^3/time. Note that rainfall is supplied in units of length/time; internally to **PTC**, rainfall is multiplied by the area associated with each node in order to arrive at the total volumetric input rate.

All boundary conditions may be applied at any node, whether on the interior or on the boundary, and in any layer. Of course, rain flux is only applied at the top boundary. The default condition is a no-flow boundary on the entire exterior boundary, with no applied stresses on the interior of the domain.

Dirichlet and Neumann conditions are stored in the same array, as they are mutually exclusive quantities. In order to specify which is desired, a flag must be supplied prior to each individual quantity. This flag is 1 for a Dirichlet condition, and 2 for a Neumann condition. If nothing is input for a node, it is assumed that no input flux is applied at the node, if an interior node, or a zero flux condition applies, if a boundary node.

Leakage conditions require a reference head, a conductance, and a reference concentration, where conductance is hydraulic conductivity divided by length. The conductance is multiplied by the area associated with the node, internal to the program, in order to calculate fluxes. The reference concentration is input but ignored in flow simulations. Leakage is applied regardless of other boundary conditions, but in general mixing boundary conditions is not recommended.

For mass transport cases, either the concentration of the ambient water is specified (Dirichlet condition) or the concentration of the fluid flux into the system is specified (Neumann condition). A node with specified flow leaving the system has concentration leaving the system at ambient levels; a node with specified flow entering the system must have some non-negative concentration specified. Accordingly, specified mass flux information is ignored unless water is entering the system at a Neumann or leakage node.

The following is the last portion of the command word and the corresponding input variable. Techniques for resuming command interpretation (so as not to be forced to specify zero values) are discussed in the section on the distributed data facilities.

BCFLO	Specified Dirichlet and Neumann information for flow.
`IDFEQN`	Boundary condition flag (1 for Dirichlet, 2 for Neumann).
`FQZ`	Boundary condition value [L or L^3/T] for flow.
BCCON	Specified Dirichlet and Neumann information for transport.
`IDCEQN`	Boundary condition flag (1 for Dirichlet, 2 for Neumann).
`CFQZ`	Boundary condition value for transport [M/L^3].
LEAK	Specified leakage information for flow and transport.
`RHEAD`	Reference head value for flow [L].
`RCOND`	Conductance value for flow [1/T].

RCONC	Reference concentration value for transport [M/L^3].
RAIN	Specified rain (infiltration) information for flow.
RAIN	Rainfall, to be integrated over the area associated with the node [L/T]. Note that the layer flag must not appear when specifying rainfall.

4.14 Output Control

This section describes the specification of all output control requests which require periodic output during time stepping, and the specification of the files which output goes to.

Each of the periodic output requests takes two pieces of information, the initial time step for output and an increment for subsequent output. For instance, specifying KFTBEG as 10 and KFTINC as 5 will result in output after flow time step 10 and every 5th time step thereafter.

As a check, the first horizontal solution (before the vertical step) may be output by specifying KFTBEG or KMTBEG as a number less than one. For example, if every 50th solution was desired for flow, the proper specification for KFTINC would be 50 and the proper value for KFTBEG would be either 0 or 50, depending on whether the first horizontal solution was desired or not.

These options are all initialized to produce no output – any option for which output is desired must be explicitly reset. Note that *RDPRTKEY*, *RDFOPKEY*, and *RDMOPKEY* can be used interchangably.

RDPRTKEY	Specification of solution output intervals.
KFTBEG	Beginning flow time step for flow output. The first horizontal solution is output if KFTBEG is less than 1.
KFTINC	Number of flow time steps between output for flow.
KMTBEG	Beginning transport time step for mass output. The first horizontal solution is output if KMTBEG is less than 1.
KMTINC	Number of transport time steps between output for mass.
RDFOPKEY	Specification of flow solution output intervals.
KFTBEG	Beginning flow time step for output. The first horizontal solution is output if KFTBEG is less than 1.
KFTINC	Number of flow time steps between output.
KFGBEG	Beginning flow time step for graphical output.
KFGINC	Number of flow time steps between graphical output.
RDMOPKEY	Specification of mass transport solution output intervals.
KMTBEG	Beginning transport time step for output. The first horizontal solution is output if KMTBEG is less than 1.
KMTINC	Number of transport time steps between output.
KMGBEG	Beginning transport time step for graphical output.

KMGINC	Number of transport time steps between graphical output.
RDVOPKEY	Specification of velocity solution output intervals.
KVTBEG	Beginning flow time step for velocity output.
KVTINC	Number of flow time steps between output for velocity.
RDQOPKEY	Specification of output intervals for fluid flux at dirichlet nodes.
KQTBEG	Beginning flow time step for fluid flux output.
KQTINC	Number of flow time steps between output for fluid flux.
RDDOPKEY	Specification of output intervals for dispersive flux at dirichlet nodes.
KDTBEG	Beginning transport time step for dispersive flux output.
KDTINC	Number of transport time steps between output for dispersive flux.

The following commands specify the units which are to receive the output requests.

OWRIT	Specification of data output unit.
NUODAT	Unit number for formatted data output requests and error messages (negative to inhibit output).
MWRIT	Specification of mass balance output units.
NUOMBF	Unit number for flow simulations (negative to inhibit output).
NUOMBM	Unit number for transport simulations (negative to inhibit output).
RWRIT	Specification of solution result output units.
NUOHD	Unit number for flow simulations (negative to inhibit output).
NUOCC	Unit number for transport simulations (negative to inhibit output).
GWRIT	Specification of graphics output units (see graphics section).
CECHO	Specification of command echo unit.
KCECHO	Unit number for command echoes and messages (negative to inhibit output).
DECHO	Specification of distributed data echoing. This is a low-level echo, prior to scaling, data generation, etc.
KDECHO	Unit number for **dfile** input echoes (negative to inhibit output).
ZDECHF	FORTRAN format for **dfile** input echoes, enclosed in quotes (a "*" will result in list-directed output).

4.15 Mass Balance Output Interpretation

4.15.1 Fluid Mass Balance Output

The objective of the mass balance routine is to provide information about the independently computed net flux across the model domain boundary and the change of mass in the domain. While we refer to this procedure as a mass balance, all output is in terms of volume balance. This is equivalent under the assumption of constant density. The **PTC** fluid volume balance

provides information on the total volumetric flux rate attributed to each of the mechanisms by which fluid is transferred. In each case the flux into the domain and the flux out of the domain is given.

The total flux into the domain represents the total of the fluxes at nodes at which the flux is directed into the domain; similarly, the flux out is the total of all outward nodal fluxes. Thus, referring to the sample output, the second column provides the volumetric flux due to specified infiltration. Columns three and four provide the total flux into and out of the domain at specified head nodes. This is computed by back-substituting into the finite element equation for the flux term at each of these nodes. Columns five and six include fluxes in and out at specified flux points. These include specified flux boundary points and well points. Columns seven and eight describe the flux in and out of the domain resulting from leakage, computed as the difference between the solved head and the reference head, multiplied by the leakage conductance coefficient. Column nine provides the net flux across the boundary, which is the sum of all fluxes into the domain minus the sum of all fluxes out of the domain. Column ten provides the net storage flux, which is the rate of change of fluid mass within the domain. This is computed by dividing the difference in heads between time steps by the time step. Column eleven provides the net volumetric flux rate, and is simply the difference between the net boundary flux and the net storage flux. For a perfect balance this value should be zero. In practice, the net flux is often non-zero.

To provide a scaled measure of imbalance, column twelve provides a scaled net flux, which is the net flux in column eleven divided by the sum of the absolute values of all flux activity across the boundary and in the domain. Finally, column thirteen provides the accumulated volume loss, which is a running sum of the net flux, multiplied by the time step. The accumulated volume loss represents the accumulated volume of fluid violating the balance reported in column eleven. If this value is positive then the computed heads are lower than they should be to properly represent the computed volume which has crossed the boundary. If this value is negative then the computed heads are higher than they should be.

If the flow portion of **PTC** is used to model transient flow of groundwater, then the mass balance output can be used to determine the accuracy of the model at maintaining balance. In this case the user should attempt to manipulate the numerical discretization parameters to maintain a small imbalance. Often the flow component of **PTC** is used to produce a steady state solution. This is accomplished by iteration through time until the head solutions remain constant. Equivalently, steady state is achieved when the net storage flux, the net boundary flux, and the net flux are all approximately zero; that is, when there is no change in the volume of fluid in the domain. During the course of iteration to steady state, the net flux may be significantly different from zero. These transient values are not of concern, as only the final solution is of interest.

4.15.2 Contaminant Mass Balance Output

In a manner similar to that described for the fluid volume balance, the components of the contaminant mass flux across the domain and the change of mass within the domain are provided in the output. Columns two and three provide the contaminant flux in and out of the domain as a result of convection, while columns four and five provide the dispersive flux at nodes which have specified concentration (note that the default boundary condition is zero dispersive flux). Columns six and seven provide the contaminant flux at specified fluid flux points. This includes boundary fluxes, fluxes at leakage points, and fluxes at well points. Column eight provides the net flux of contaminant across the boundary.

Column nine provides the net storage flux or rate of change in contaminant mass within the domain. This is computed by determining the change in mass associated with each node and dividing this change by the time step. The difference in these fluxes is reported in column ten as the net flux; for a perfect balance this quantity should be zero. Again, as for the fluid balance, a scaled net flux is reported which consists of the absolute value of net flux divided by the total absolute value of all fluxes across the boundary and in the domain during this time step. Finally column twelve provides the accumulated mass loss. This value provides the quantity of mass which has crossed the boundary but not resulted in an increase in concentrations within the domain as a result of discretization error. If this value is positive it indicates that mass is missing from the domain (i.e. concentrations are lower than they should be); if this value is negative, too much mass is present.

4.16 Output Query Commands

This section describes the specification of all output query requests. All queries result in output to the data output file, which may be the terminal.

The following commands are each prefixed by either "EC" or "WR", short for "echo" or "write".

All output is formatted and interpreted. In general, these commands correspond to a command specifying input. Queries regarding parameters which are defined in each layer output all layers – in general, these queries should be made following the input of all layers.

TITLE	Title.
CONTRL	Problem simulation options.
DIMS	Problem dimension.
TIME	Time stepping control.
WEIGHT	Upstream weighting.
WTABLE	Water table options.
INITH	Initial (current) head at each node.
INITC	Initial (current) concentration at each node.

BC	Boundary conditions, including Dirichlet, nonzero Neumann, and specified leakage, at each node.
RAIN	Rainfall flux at each node.
COORS	Nodal coordinates.
INCID	Element incidence list.
ELEV	Nodal elevations.
QUADS	Quadrilateral element list (this is automatically generated by the *SIM* command, prior to running the simulation).
CONDX	Hydraulic conductivity in the X coordinate direction.
CONDY	Hydraulic conductivity in the Y coordinate direction.
CONDZ	Hydraulic conductivity in the Z coordinate direction.
ELONG	Longitudinal dispersivity.
ETRAN	Transverse dispersivity.
STOR	Storage coefficient/specific yield.
POROS	Effective porosity.
ADSOR	Adsorption coefficients.
DIFUSN	Molecular diffusion coefficient.
LAYER	All variables which are specified for each element, and the thickness of each layer.
VELOC	Velocity in the X, Y, and Z coordinate directions.
XVELO	Velocity in the X coordinate direction.
YVELO	Velocity in the Y coordinate direction.
ZVELO	Velocity in the Z coordinate direction.

4.17 Graphics Output Commands

This section describes the specification of all graphics output requests.

PTC is not tied to any particular graphics package, as there are a number of packages which are widely available. Output of nodal coordinates, incidences, and solutions is implemented in the format of a proprietary plotting package, as an example of output protocol, but it is expected that a small amount of recoding may be required in order to match the input requirements of individual plotting packages. This should be limited to output statements in the subroutine PTCGRF.

The following comments apply to the output protocol as implemented for the proprietary plotting package. It is suggested that a similar procedure is followed for other implementations.

Some graphics packages allow all input to reside in one file, where others require each set of information to exist in a separate file. **PTC** allows for either case, at the cost of some sophistication in the **PTC** data set programming. Plotting packages may also require certain

command sequences, in addition to information such as element incidences and information at each node, and again **PTC** allows for this possibility.

PTC will output nodal coordinates and element incidences on request, output any of the distributed parameters, and periodically output head and concentration values at each node. Any additional information which is specific to the plotting package may be echoed from the input file to the graphics file using the *GRECHON* and *GRECHOFF* commands. These commands allow verbatim translation from the input file to the graphics file by reading a card into a buffer, then outputting the buffer. No interpretation of the contents of the buffer is made, except to check if the *GRECHOFF* command has been input.

Except for the verbatim echo, all information is output using an implied DO loop for the variable, over the number of elements or number of nodes as appropriate. When outputting the solution information, an outer DO loop is made over the number of layers.

4.17.1 FORMAT Statement Syntax for Graphics Output

When outputting information to the graphics file, it is necessary to specify a FORTRAN FORMAT to be used for the coordinate, incidence, and solution information. The FORMAT is specified following the appropriate command. This FORMAT is read into a character string, and must have a single quote followed by a left parenthesis at the beginning of the FORMAT, and a right parenthesis and single quote at the end of the FORMAT.

The FORMAT may also specify heading information for the graphics package. Since the graphics package may or may not require a heading, the following provides a brief discussion of FORMAT statements. For a fuller description, see any FORTRAN manual.

Each of the graphics commands for distributed parameters have a syntax something like a possible syntax for coordinates:

```
WRITE(NUOGRF,ZECFMT) MFUNC,NND,(I1,(COOR(I2,I1),I2=1,2),I1=1,NND)
```

This statement sends output to the current graphics output unit, using `ZECFMT` for the FORMAT. The information available for output is a "function number", the number of nodes `NND`, and `NND` sets of the sequence `I1,COOR(1,I1),COOR(2,I1)`. Any or all of this information can end up in the output file, using the appropriate FORMAT.

The simplest case for this output would be to output the function number and number of nodes on one line, followed by `NND` lines, each with a node number and a pair of coordinates. For this case, the input `ZECFMT` would be something like:

```
'(2I4/(I4,2E12.4))'
```

For those not familiar with FORTRAN, the I commands are used for integers and the E commands are used for floating point numbers. The number after an I command is the width allotted for the output number. The first number of the two after the E command similarly allots a space for the floating point number, and the second number is the number

of digits following the decimal point. With the E command, the first number should be at least 8 more than the second number.

The 2 in front of the E is a repeat count, stating that two floating point numbers are expected; this is used for the first I command as well. The comma is used to separate format commands, and the slash (/) is the line feed character.

The outermost set of parentheses is always required for a FORMAT statement. The inner set illustrates a subtle but very useful point regarding FORTRAN FORMATS. In the usual case, with no inner parentheses, the entire FORMAT is reused from the start, on a new line, if there is more data to be output than allocated for in the FORMAT. However, if the FORMAT ends with commands inside parentheses, only the commands inside the parentheses are reused. For this example, `MFUNC` and `NND` is output using the first part of the FORMAT, and the node number and coordinates are output using the string inside the inner parentheses.

A more complicated case arises when the function number is not wanted. For this case, the FORMAT would be modified to read:

```
'(I4,T1,I4/(I4,2E12.4))'
```

The T command resets the output column which the next piece of information starts in. In this case, it has the effect of erasing the function number. Note that the function number must be output, THEN erased.

Similarly, if only the coordinates are wanted, the FORMAT could be modified to read:

```
'(2I4,T1,(I4,T1,2E12.4))'
```

In this case, no slash command is needed.

As a final example, consider the case where there are four nodes and the output is to look like

```
rdnod1
 4
-1 '*'
  0.000e+00  0.000e+00
  1.000e+00  0.000e+00
  2.000e+00  1.000e+01
  4.000e+01  4.000e+00
```

A FORMAT which will produce this is

```
'(6Hrdnod1/I2,T1,I2/6H-1 ''*''/(I4,T1,1PE11.3,1PE11.3))'
```

This is a fairly complicated set of instructions, to say the least, but illustrates the full power of the approach.

The H command outputs characters, with the preceding 6 in both cases stating that the next 6 characters are to be output. The first slash forces a line feed, then the function number is output and overwritten by the number of nodes. The second slash forces another line feed, and again a character string is output.

Another subtlety pops up here; pairs of quotes are needed, instead of single quotes, so that **PTC** will know that a quote is wanted, rather than the end of the FORMAT string. Each pair of quotes is condensed into one quote, so the H command has a 6 instead of an 8 preceding it.

Still another subtlety pops up inside the innermost set of parentheses. The 1P in front of the E moves the first significant digit in front of the decimal point. Without this, the first significant digit is after the decimal point, wasting a digit of output.

The two final examples should provide a template for almost any situation.

4.17.2 Graphics Output Commands

The graphics output commands are similar to the data echo commands. However, several commands are provided so that the vagaries of plotting packages may be handled gracefully. In particular, output may go to a single file or each function may go to a different file. A function refers to values for one layer; in a three layer problem, for example, a request for head output results in three functions. The function number starts at 1, is automatically incremented as required, and may be reset as desired.

If separate files are required, specifying a filename seed (8 characters or less) will generate a series of files with the names of *filename.1*, *filename.2*, and so on. Specifying a blank filename seed will result in single file output to the file corresponding to unit `NUOGRF` (this is the default). Often the file number is the same as the function number.

In general, only distributed data may be requested, and only nodal values are output. Each of the values defined as piecewise constant over elements has a nodal value calculated from an areal average over the elements attached to the node, in the same way that rainfall flux is calculated from elemental values, and this average value is output.

RDGRFKEY	Specification of graphics output intervals.
`KFGBEG`	Beginning flow time step for flow output.
`KFGINC`	Number of flow time steps between output for flow.
`KMGBEG`	Beginning transport time step for mass output.
`KMGINC`	Number of transport time steps between output for mass.
GWRIT	Specification of graphics output unit.
`NUOGRF`	Unit number for graphics output file (negative to inhibit output).
GRECHON	The first 80 characters of each subsequent card are copied to the graphics output file, until the *GRECHOFF* command is encountered.

GRECHOFF Terminate verbatim translation from the command file to the graphics output file and resume interpreted input.

GRFNAME Specification of filename seed and starting file number.

`ZFNAME` Filename seed for multiple files (blank for single files).

`MFIRST` Starting file number for multiple files.

GRFNCNUM Specification of starting function number.

`MFUNC` Starting function number.

GRSOLFMT Read the format used for function number, node number, and head, concentration and distributed parameter output to graphics output file.

`ZSLFMT` FORTRAN format for function number, node number, nodal coordinates, and head, concentration, or distributed parameter value, enclosed in quotes. The write string is `MFUNC, (I1, (COOR(I2,I1), I2=1,2), ARRND(I1), I1=1,NND)`.

GRCOORS Echo node and nodal coordinates to graphics output file.

`ZCDFMT` FORTRAN format for node and nodal coordinate echo, enclosed in quotes. The write string is `NND, (I1, (COOR(I1,I2), I1=1,2), I2=1,NND)`.

GRINCID Echo element and elemental incidences to graphics output file.

`ZIDFMT` FORTRAN format for element and elemental incidence echo, enclosed in quotes. The write string is `NEL, (I1, (INCID(I2,I1), I2=1,4), I1=1,NEL)`.

GRHEAD Echo current head solution to the graphics output file, using the current value of `ZHDFMT` for the output format.

GRCONC Echo current concentration solution to the graphics output file, using the current value of `ZCCFMT` for the output format.

The following commands are preceded by a "GR", and use the current value of `ZSLFMT` for the output format.

CONDX Hydraulic conductivity in the X coordinate direction.

CONDY Hydraulic conductivity in the Y coordinate direction.

CONDZ Hydraulic conductivity in the Z coordinate direction.

STOR Storage coefficient/specific yield.

ELONG Longitudinal dispersivity.

ETRAN Transverse dispersivity.

POROS Porosity.

XVELO Velocity in the X coordinate direction.

YVELO Velocity in the Y coordinate direction.

ZVELO Velocity in the Z coordinate direction.

Since the plotting control sequences tend to become complicated, it is suggested that all of the plot commands are isolated to a separate file, independent of the main command file. Prior to running a simulation, the plotting instructions would be executed by using the *CFILE* command. At the bottom of the separate file, a command of *CRETURN* will return to the previous command file.

4.18 Looping

In certain circumstances, it is advantageous to allow looping from within a data set. For example, the response of a system under the influence of an irrigation schedule, or yearly rainfall patterns, may naturally have some repetitious set of applied stresses. **PTC** allows looping from within the data set by copying the portions of the data set to be repeated into an auxiliary file, and using this auxiliary file as a surrogate for the command file. This procedure is transparent to the usual conventions for switching the *CFILE*. Unit 91 is reserved for this looping file.

In order to keep track of loop nesting properly, each loop must have a separate name. This loop name is an arbitrary sequence of up to six characters, signified by *xxxxx* in the description.

DOxxxxxx	Signal the top of a loop.
`NLOOP`	Number of repetitions for the loop.
ODxxxxxx	Signal the bottom of the current innermost loop. If no loop is active, this is ignored.

The number of loops for which may be concurrently active is controlled by the parameter MNDO in the INCOMM module. This is set at 10 initially. If more levels of nesting are required, redimension MNDO, recompile the IOTOOLS module, and relink **PTC**.

4.19 Sample Input Sequences

This section provides sample input sequences with explanation of the sequences. All of the sample sequences are copied directly from running data sets.

4.19.1 Simulation Initiation Sequence

This section assumes that all model parameters are already defined.

```
rdcontrl
1 0 0 1 1            *nflow nvel nconc ncore nmb
gntime
```

```
100 5 20 0 1.5 365.  *itmax itchng itchmx nmspf chng tleng
rdprtkey
0 50 50 50           *kftbeg kftinc kmtbeg kmtinc
rdwtable
1 50  0.001          *ntable niter epsiln
sim
term
```

The above sequence turns on flow simulation, with calculation incore and mass balance calculations for flow.

Then, 100 time steps are requested. The time step is calculated by multiplying it by 1.5 every 5 time steps until the 50th time step, and keeping it constant thereafter. The initial time step is calculated internally, so that following the above procedure will result in the elapsing of 365 days. Solution output is requested every 50 time steps, for both flow and mass transport simulations, starting on the 50th time step in both cases. The first horizontal solution is also requested for the flow simulation.

The water table option is turned on, with a maximum of 50 nonlinear iterations for water table convergence, where convergence is reached with the maximum change in water table elevation being less than 0.001.

Simulation is requested, followed by return to the terminal for further instructions.

4.19.2 Distributed Parameter Input Sequence

This section demonstrates various input strategies for distributed parameters.

The demonstrated sequence achieves three goals.

The problem dimensions are defined with the first command. The next sequence defines the hydraulic conductivity field for the entire region. The last sequence defines the boundary conditions for the bottom layer in the problem.

```
rddims
20 12 3               *nnd nel nly
drgparam              *condx condy condz stor por elong etran
10.0 10.0 1.0 0.001 0.3 50.0 10.0 1.0
deacondx
2 0.001              *layer 2 x conductivity
deacondy
2 0.001              *layer 2 y conductivity
descondx
2 10.0               *layer 2 x conductivity window
8/                   *1 element (8)
descondy
```

```
2 10.0              *layer 2 y conductivity window
8/                  *1 element (8)
dfromc
'*' 1
indexon
dgenon
drecondz
2                    *layer 2 z conductivity
1  1   0.0001        *initialize all elements
12 0   0.0001
8  0   1.0000        *define window element
0/
drnbcflo
1
1 1  1   88.0        *layer 1 heads (nodes 1,2,3 and 4)
4 0  1   90.0
0/
dgenoff
scale
192.5134
denbcflo
1                    *layer 1 specified fluxes
5  2 -10.0           *extraction well
8  2   7.0           *injection well
0/
scaleoff
```

The first command states that there are 20 nodes, 12 elements, and 3 layers in the problem.

The next command states that hydraulic conductivity in the X and Y directions is 10.0, hydraulic conductivity in the Z direction is 1.0, the storage coefficient is 0.001, porosity is 0.3, longitudinal dispersivity is 50.0, horizontal transverse dispersivity is 10.0, and vertical transverse dispersivity is 1.0. These values are specified globally (for all elements in all layers).

The next two commands redefine the X and Y hydraulic conductivity in the middle layer to be 0.001.

The next two commands define a window in element 8 of layer 2, with X and Y hydraulic conductivities matching the other layers.

The next four commands accomplish the same objective for the Z direction hydraulic conductivity, using the distributed data facilities. Of this sequence, the first command states that input from the **dfile** will be read list-directed, with one data group per line; note

that commenting is enabled at the end of each line. The second and third enables the index and data generation fields respectively. The last defines elements 1 through 12 as having the value 0.0001 for Z direction hydraulic conductivity, then redefines element 8 as having the value of 1.0000.

The next five commands define the Dirichlet and Neumann boundary conditions for the bottom layer. Of this sequence, the first command states that nodes 1 through 4 have values for head of 88.0, 88.67, 89.33, and 90.0 respectively, using the distributed data facilities. The second and third commands disable the data generation field and define a scale factor converting gallons per minute into cubic feet per day, which is applied hereafter. The fourth command defines a pumping well of 10 gpm at node 5 and an injection well of 7 gpm at node 8, and the last command disables automatic scaling.

4.19.3 Time Dependent Boundary Conditions

This section assumes that all time invariant parameters are defined.

```
dgenoff
scale
192.5134
denbcflo
1                    *layer 1 specified fluxes
5  2 -10.0           *extraction well
8  2   7.0           *injection well
0/
rdcontrl
1 0 0 1 1            *nflow nvel nconc ncore nmb
rdtime
100 5 50 2 0.1 1.5   *itmax itchng itchmx nmspf delt chng
rdprtkey
0 50 50 50           *kftbeg kftinc kmtbeg kmtinc
rdwtable
1 50  0.001          *ntable niter epsiln
sim
denbcflo
1                    *layer 1 specified fluxes
5  2 -20.0           *extraction well
8  2   5.0           *injection well
0/
rdtime
100 5 20 2 0.2 1.5   *itmax itchng itchmx nmspf delt chng
rdprtkey
```

```
50 50 50 50          *kftbeg kftinc kmtbeg kmtinc
sim
term
```

The above sequence specifies fluxes at a pumping well and an extraction well, turns on flow simulation, with calculation incore and mass balance calculations for flow.

Then, 100 time steps are requested, with an initial time step of 0.1. This is multiplied by 1.5 every 5 time steps until the 50th time step. Solution output is requested every 50 time steps, for both flow and mass transport simulations, starting on the 50th time step in both cases. The first horizontal solution is also requested for the flow simulation. Note that transport simulation output requests will be ignored since no transport simulation is requested.

The water table option is turned on, with a maximum of 50 nonlinear iterations for water table convergence, where convergence is reached with the maximum change in water table elevation being less than 0.001.

Simulation is requested.

The pumping well fluxes are reset, and the time step is reset. The print key is reset, turning off the first horizontal solution output, then simulation is requested. This is followed by return to the terminal for further instructions.

4.20 Verification/Debugging

Sooner or later, an error will creep into a data set. Sometimes an error will manifest itself in an obvious way, by crashing **PTC** immediately. Other times, the error is less obvious.

PTC verifies the input data as much as possible, upon the issue of a *SIM* command, in order to catch gross errors. Accordingly, it is recommended that the simulation is run for one time step the first time a data set is run. Any errors that are detected will be flagged and output to the current `NUODAT` unit (set with the *OWRIT* command). The output file may then be inspected for these gross errors.

The user is responsible for final verification. As a simple precaution, particularly in the first few runs of a newly created data set, all variables should be output before running the simulation. By comparing the output to the input, obvious errors can be detected.

Graphical verification is also strongly recommended. Using the graphics output facilities, all of the distributed parameters may be output either by element or by node (nodal output allows for contouring).

Often, simply by taking these verification steps, changes to the data set are obvious. Other times, however, the output may bear little relationship to the input. In these cases, usually the format of the data set is incorrect or an array has had input beyond the bounds of the allotted dimensioning.

The structure of the data input allows for interactive debugging, as any variable may be output as many times as desired. This can be done by putting a series of *TERM* commands in the data file, and selectively echoing variables to the terminal. Once it is apparent that a variable is not correctly input, the location of the error may quickly be pinpointed by outputting the variable at several locations. With a few iterations, the source of the error can be bracketed – often the error occurs far from the point where the variable is first input.

A number of errors are commonly made by those not familiar with the code. Some of the common errors are listed below.

Forgetting arguments. If a command expects an argument, and the argument is missing, the next command is used as an argument with unpredictable results. Use CECHO to echo interpreted commands.

Commenting arguments. If a command expects an argument, and the argument is commented out, unpredictable results occur. Use CECHO to echo interpreted commands. Note that once a valid argument is found, other arguments can be commented out below the first.

INDEXON/INDEXOFF. Forgetting these commands will result in too many, not enough, or strange data values being read, and possible array violations.

DGENON/DGENOFF. Forgetting these commands will result in too many, not enough, or strange data values being read, and possible array violations.

SCALE/SCALEOFF. Forgetting these commands will result in strange data values being read.

Forgetting command size. Each command is 8 characters long, including trailing blanks. This is only a consideration if the command has a trailing comment. In some systems, a TAB character is NOT a blank character! Use CECHO to echo interpreted commands.