The SWMS_2D Code for Simulating Water Flow and Solute Transport in Two-Dimensional Variably Saturated Media

Version 1.21

Research Report No. 132

February 1994

U. S. SALINITY LABORATORY
AGRICULTURAL RESEARCH SERVICE
U. S. DEPARTMENT OF AGRICULTURE
RIVERSIDE, CALIFORNIA

The SWMS_2D Code for Simulating Water Flow and Solute Transport in Two-Dimensional Variably Saturated Media

Version 1.21

by

J. Šimůnek, T. Vogel and M. Th. van Genuchten

Research Report No. 132

February 1994

U. S. SALINITY LABORATORY
AGRICULTURAL RESEARCH SERVICE
U. S. DEPARTMENT OF AGRICULTURE
RIVERSIDE, CALIFORNIA

DISCLAIMER

This report documents version 1.21 of SWMS_2D, a computer program for simulating two-dimensional water flow and solute transport in variably saturated media. SWMS_2D is a public domain code, and as such may be used and copied freely. The code has been verified against a large number of test cases. However, no warranty is given that the program is completely error-free. If you do encounter problems with the code, find errors, or have suggestions for improvement, please contact one of the authors at

U. S. Salinity Laboratory
USDA, ARS
4500 Glenwood Drive
Riverside, CA 92501

Tel. 909-369-4846
Fax. 909-369-4818

ABSTRACT

Šimůnek, J., T. Vogel and M. Th. van Genuchten. 1994. The SWMS_2D Code for Simulating Water Flow and Solute Transport in Two-Dimensional Variably Saturated Media, Version 1.21. Research Report No. 132, U.S. Salinity Laboratory, USDA, ARS, Riverside, California.

This report documents version 1.21 of SWMS_2D, a computer program for simulating water and solute movement in two-dimensional variably saturated media. The program numerically solves the Richards' equation for saturated-unsaturated water flow and the convection-dispersion equation for solute transport. The flow equation incorporates a sink term to account for water uptake by plant roots. The transport equation includes provisions for linear equilibrium adsorption, zero-order production, and first-order degradation. The program may be used to analyze water and solute movement in unsaturated, partially saturated, or fully saturated porous media. SWMS_2D can handle flow regions delineated by irregular boundaries. The flow region itself may be composed of nonuniform soils having an arbitrary degree of local anisotropy. Flow and transport can occur in the vertical plane, the horizontal plane, or in a three-dimensional region exhibiting radial symmetry about the vertical axis. The water flow part of the model can deal with prescribed head and flux boundaries, as well as boundaries controlled by atmospheric conditions. New features of the present version 1.21 include the implementation of free drainage boundary conditions, and a simplified representation of nodal drains using results of electric analog experiments. This version has also more flexibility in selecting certain boundary conditions for solute transport.

The governing flow and transport equations are solved numerically using Galerkin-type linear finite element schemes. Depending upon the size of the problem, the matrix equations resulting from discretization of the governing equations are solved using either Gaussian elimination for banded matrices, or a conjugate gradient method for symmetric matrices and the ORTHOMIN method for asymmetric matrices. The program is written in ANSI standard FORTRAN 77. Computer memory is a function of the problem

definition. This report serves as both a user manual and reference document. Detailed instructions are given for data input preparation. Example input and selected output files are also provided, as is a listing of the source code.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF VARIABLES

a	dimensionless water stress response function [-]
A_e	area of a triangular element [L^2]
A_{qh}	parameter in equation (6.1) [LT^{-1}]
$[A]$	coefficient matrix in the global matrix equation for water flow, [LT^{-1}] or [L^2T^{-1}]†
b	normalized root water uptake distribution, [L^{-2}] or [L^{-3}]†
b'	arbitrary root water uptake distribution, [L^{-2}] or [L^{-3}]†
b_i, c_i	geometrical shape factors [L]
B_{qh}	parameter in equation (6.1) [L^{-1}]
$\{B\}$	vector in the global matrix equation for water flow, [L^2T^{-1}] or [L^3T^{-1}]†
c	solution concentration [ML^{-3}]
c'	finite element approximation of c [ML^{-3}]
c_i	initial solution concentration [ML^{-3}]
c_n	value of the concentration at node n [ML^{-3}]
c_s	concentration of the sink term [ML^{-3}]
c_0	prescribed concentration boundary condition [ML^{-3}]
C_d	factor used to adjust the hydraulic conductivity of elements in the vicinity of drains [-]
Cr_i^e	local Courant number [-]
d	effective drain diameter [L]
D	side length of the square in the finite element mesh surrounding a drain (elements have adjusted hydraulic conductivities) [L]
D_d	ionic or molecular diffusion coefficient in free water [L^2T^{-1}]
D_{ij}	components of the dispersion coefficient tensor [L^2T^{-1}]
D_L	longitudinal dispersivity [L]
D_T	transverse dispersivity [L]
$\{D\}$	vector in the global matrix equation for water flow, [L^2T^{-1}] or [L^3T^{-1}]†
e_n	subelements which contain node n [-]

E maximum (potential) rate of infiltration or evaporation under the prevailing atmospheric conditions [LT^{-1}]

$\{f\}$ vector in the global matrix equation for solute transport, [$MT^{-1}L^{-1}$] or [MT^{-1}]†

$[F]$ coefficient matrix in the global matrix equation for water flow, [L^2] or [L^3]†

$\{g\}$ vector in the global matrix equation for solute transport, [$MT^{-1}L^{-1}$] or [MT^{-1}]†

$[G]$ coefficient matrix in the global matrix equation for solute transport, [L^2T^{-1}] or [L^3T^{-1}]†

h pressure head [L]

h^* scaled pressure head [L]

h' finite element approximation of h [L]

h_A minimum pressure head allowed at the soil surface [L]

h_n nodal values of the pressure head [L]

h_s air-entry value in the soil water retention function [L]

h_S maximum pressure head allowed at the soil surface [L]

h_0 initial condition for the pressure head [L]

k distribution coefficient [L^3M^{-1}]

K unsaturated hydraulic conductivity [LT^{-1}]

K^* scaled unsaturated hydraulic conductivity [LT^{-1}]

$\mathbf{K}^A$ dimensionless anisotropy tensor for the unsaturated hydraulic conductivity **K** [-]

K_{drain} adjusted hydraulic conductivity in the elements surrounding a drain [LT^{-1}]

K_{ij}^A components of the dimensionless anisotropy tensor $\mathbf{K}^A$ [-]

K_k measured value of the unsaturated hydraulic conductivity corresponding to θ_k [LT^{-1}]

K_r relative hydraulic conductivity [-]

K_s saturated hydraulic conductivity [LT^{-1}]

L length of the side of an element [L]

L_i local coordinate [-]

L_n length of a boundary segment [L]

L_t width of soil surface associated with transpiration, [L] or [L^2]†

L_x width of the root zone [L]

L_z depth of the root zone [L]

m parameter in the soil water retention function [-]

M^0 cumulative amount of solute removed from the flow region by zero-order reactions, [ML^{-1}] or [M]†

M^1 cumulative amount of solute removed from the flow region by first-order reactions, [ML^{-1}] or [M]†

M_r cumulative amount of solute removed from the flow region by root water uptake, [ML^{-1}] or [M]†

M_t amount of solute in the flow region at time t, [ML^{-1}] or [M]†

M_t^e amount of solute in element e at time t, [ML^{-1}] or [M]†

M_0 amount of solute in the flow region at the beginning of the simulation, [ML^{-1}] or [M]†

M_0^e amount of solute in element e at the beginning of the simulation, [ML^{-1}] or [M]†

n exponent in the soil water retention function [-]

n_i components of the outward unit vector normal to boundary Γ_N [-]

N total number of nodes [-]

N_e number of subelements e_n which contain node n [-]

O actual rate of inflow/outflow to/from a subregion, [L^2T^{-1}] or [L^3T^{-1}]†

Pe_i^e local Peclet number [-]

q_i components of the Darcian fluid flux density [LT^{-1}]

Q_n^A convective solute flux at node n, [$MT^{-1}L^{-1}$] or [MT^{-1}]†

Q_n^D dispersive solute flux at node n, [$MT^{-1}L^{-1}$] or [MT^{-1}]†

Q_n^T total solute flux at node n, [$MT^{-1}L^{-1}$] or [MT^{-1}]†

$\{Q\}$ vector in the global matrix equation for water flow, [L^2T^{-1}] or [L^3T^{-1}]†

$[Q]$ coefficient matrix in the global matrix equation for solute transport, [L^2] or [L^3]†

R retardation factor [-]

s adsorbed solute concentration [-]

S sink term [T^{-1}]

S_e degree of saturation [-]

S_{ek} degree of saturation corresponding to θ_k [-]

S_p spatial distribution of the potential transpiration rate [T^{-1}]

$[S]$	coefficient matrix in the global matrix equation for solute transport, $[L^2T^{-1}]$ or $[L^3T^{-1}]^\dagger$
t	time [T]
T_a	actual transpiration rate per unit surface length $[LT^{-1}]$
T_p	potential transpiration rate $[LT^{-1}]$
v	pore-water velocity $[LT^{-1}]$
V	volume of water in each subregion, $[L^2]$ or $[L^3]^\dagger$
V_{new}	volume of water in each subregion at the new time level, $[L^2]$ or $[L^3]^\dagger$
V_{old}	volume of water in each subregion at the previous time level, $[L^2]$ or $[L^3]^\dagger$
V_t	volume of water in the flow domain at time t, $[L^2]$ or $[L^3]^\dagger$
V_t^e	volume of water in element e at time t, $[L^2]$ or $[L^3]^\dagger$
V_0	volume of water in the flow domain at time zero, $[L^2]$ or $[L^3]^\dagger$
V_0^e	volume of water in element e at time zero, $[L^2]$ or $[L^3]^\dagger$
x_i	spatial coordinates ($i=1,2$) [L]
Z_0	characteristic impedance of a transmission line analog to drain
Z_0'	characteristic impedance of free space (≈376.7 ohms)
α	coefficient in the soil water retention function $[L^{-1}]$
α^w	weighing factor [-]
α_K	scaling factor for the hydraulic conductivity [-]
α_h	scaling factor for the pressure head [-]
α_θ	scaling factor for the water content [-]
γ_s	zero-order rate constant for solutes adsorbed onto the solid phase $[T^{-1}]$
γ_w	zero-order rate constant for solutes in the liquid phase $[ML^{-3}T^{-1}]$
Γ_e	boundary segments connected to node n
Γ_D	part of the flow domain boundary where Dirichlet type conditions are specified
Γ_G	part of the flow domain boundary where gradient type conditions are specified
Γ_N	part of the flow domain boundary where Neumann type conditions are specified
Γ_C	part of the flow domain boundary where Cauchy type conditions are specified
δ_{ij}	Kronecker delta [-]
Δt	time increment [T]

Δt_{max} maximum permitted time increment [T]

Δt_{min} minimum permitted time increment [T]

ϵ temporal weighing factor [-]

ϵ_a^c absolute error in the solute mass balance, [ML^{-1}] or [M]†

ϵ_a^w absolute error in the water mass balance, [L^2] or [L^3]†

ϵ_r^c relative error in the solute mass balance [%]

ϵ_r^w relative error in the water mass balance [%]

ϵ_0 permittivity of free space (used in electric analog representation of drains)

θ volumetric water content [L^3L^{-3}]

θ^* scaled volumetric water content [L^3L^{-3}]

θ_a parameter in the soil water retention function [L^3L^{-3}]

θ_k volumetric water content corresponding to K_k [L^3L^{-3}]

θ_m parameter in the soil water retention function [L^3L^{-3}]

θ_r residual soil water content [L^3L^{-3}]

θ_s saturated soil water content [L^3L^{-3}]

κ parameter which depends on the type of flow being analyzed, [-] or [L]†

λ first-order rate constant [T^{-1}]

μ_s first-order rate constant for solute adsorbed onto the solid phase [T^{-1}]

μ_w first-order rate constant for solutes in the liquid phase [T^{-1}]

μ_0 permeability of free space

ρ bulk density [ML^{-3}]

ρ_d dimensionless ratio between the side of the square in the finite element mesh surrounding the drain, D, and the effective diameter of a drain, d

σ prescribed flux boundary condition at boundary Γ_N [LT^{-1}]

τ tortuosity factor [-]

ϕ_n linear basis functions [-]

ϕ_n^u upstream weighted basis functions [-]

ψ prescribed pressure head boundary condition at boundary Γ_D [L]

ω angle between principal direction of K_1^A and the x-axis of the global coordinate system [-]

Ω	flow region
Ω_e	domain occupied by element e
Ω_R	region occupied by the root zone

† for plane and axisymmetric flow, respectively

1. INTRODUCTION

The importance of the unsaturated zone as an integral part of the hydrological cycle has long been recognized. The zone plays an inextricable role in many aspects of hydrology, including infiltration, soil moisture storage, evaporation, plant water uptake, groundwater recharge, runoff and erosion. Initial studies of the unsaturated (vadose) zone focused primarily on water supply studies, inspired in part by attempts to optimally manage the root zone of agricultural soils for maximum crop production. Interest in the unsaturated zone has dramatically increased in recent years because of growing concern that the quality of the subsurface environment is being adversely affected by agricultural, industrial and municipal activities. Federal, state and local action and planning agencies, as well as the public at large, are now scrutinizing the intentional or accidental release of surface-applied and soil-incorporated chemicals into the environment. Fertilizers and pesticides applied to agricultural lands inevitably move below the soil root zone and may contaminate underlying groundwater reservoirs. Chemicals migrating from municipal and industrial disposal sites also represent environmental hazards. The same is true for radionuclides emanating from energy waste disposal facilities.

The past several decades has seen considerable progress in the conceptual understanding and mathematical description of water flow and solute transport processes in the unsaturated zone. A variety of analytical and numerical models are now available to predict water and/or solute transfer processes between the soil surface and the groundwater table. The most popular models remain the Richards' equation for variably saturated flow, and the Fickian-based convection-dispersion equation for solute transport. Deterministic solutions of these classical equations have been used, and likely will continue to be used in the near future, for predicting water and solute movement in the vadose zone, and for analyzing specific laboratory or field experiments involving unsaturated water flow and/or solute transport. These models are also helpful tools for extrapolating information from a limited number of field experiments to different soil, crop and climatic conditions, as well as to different tillage and water management schemes.

The purpose of this report is to document version 1.21 of the SWMS_2D computer program simulating water and solute movement in two-dimensional variably saturated media. The program numerically solves the Richards' equation for saturated-unsaturated water flow and the convection-dispersion equation for solute transport. The flow equation incorporates a sink term to account for water uptake by plant roots. The solute transport equation includes provisions for linear equilibrium adsorption, zero-order production, and first-order degradation. The program may be used to analyze water and solute movement in unsaturated, partially saturated, or fully saturated porous media. SWMS_2D can handle flow domains delineated by irregular boundaries. The flow region itself may be composed of nonuniform soils having an arbitrary degree of local anisotropy. Flow and transport can occur in the vertical plane, the horizontal plane, or in a three-dimensional region exhibiting radial symmetry about a vertical axis. The water flow part of the model considers prescribed head and flux boundaries, as well as boundaries controlled by atmospheric conditions. New features of the current version 1.21, as opposed to the original code [*Šimůnek et al.*, 1992], include the implementation of free drainage boundary conditions, and a simplified representation of nodal drains using results of electric analog experiments. First- or third-type boundary conditions can now be prescribed in the solute transport part of the model. Version 1.21 has also more flexibility in selecting boundary conditions for solute transport then previous version.

The governing flow and transport equations are solved numerically using Galerkin-type linear finite element schemes. Depending upon the size of the problem, the matrix equations resulting from discretization of the governing equations are solved using either Gaussian elimination for banded matrices, or the conjugate gradient method for symmetric matrices and the ORTHOMIN method for asymmetric matrices [*Mendoza et al.*, 1991]. The program is an extension of the variably saturated flow code of *Vogel* [1987], which in turn was based in part on the early numerical work of Neuman and colleagues [*Neuman*, 1972, 1973, *Neuman et al.*, 1974; *Neuman*, 1975; *Davis and Neuman*, 1983]. The SWMS_2D code is written in ANSI standard FORTRAN 77, and hence can be compiled, linked and run on any standard micro-, mini-, or mainframe system, as well as on personal computers. The

source code was developed and tested on a PC 486 using the Microsoft Fortran PowerStation.

This report serves as both a user manual and reference document. Detailed instructions are given for data input preparation. Example input and selected output files are also provided, as is a listing of the source code. Two 3½ or 5¼ inch floppy diskettes containing the source code and the input and output files of four examples discussed in this report are available upon request from the authors.

2. VARIABLY SATURATED WATER FLOW

2.1. *Governing Flow Equation*

Consider two-dimensional isothermal Darcian flow of water in a variably saturated rigid porous medium and assume that the air phase plays an insignificant role in the liquid flow process. The governing flow equation for these conditions is given by the following modified form of the Richards' equation:

$$\frac{\partial \theta}{\partial t} = \frac{\partial}{\partial x_i}[K(K_{ij}^A \frac{\partial h}{\partial x_j} + K_{iz}^A)] - S \tag{2.1}$$

where θ is the volumetric water content [L^3L^{-3}], h is the pressure head [L], S is a sink term [T^{-1}], x_i ($i=1,2$) are the spatial coordinates [L] , t is time [T], K_{ij}^A are components of a dimensionless anisotropy tensor $\mathbf{K}^A$, and K is the unsaturated hydraulic conductivity function [LT^{-1}] given by

$$K(h,x,z) = K_s(x,z)\, K_r(h,x,z) \tag{2.2}$$

where K_r is the relative hydraulic conductivity and K_s the saturated hydraulic conductivity [LT^{-1}]. The anisotropy tensor K_{ij}^A in (2.1) is used to account for an anisotropic medium. The diagonal entries of K_{ij}^A equal one and the off-diagonal entries zero for an isotropic medium. If (2.1) is applied to planar flow in a vertical cross-section, $x_1=x$ is the horizontal coordinate and $x_2=z$ is the vertical coordinate, the latter taken to be positive upward. Einstein's summation convention is used in (2.1) and throughout this report. Hence, when an index appears twice in an algebraic term, this particular term must be summed over all possible values of the index.

2.2. *Root Water Uptake*

The sink term, S, in (2.1) represents the volume of water removed per unit time from a unit volume of soil due to plant water uptake. *Feddes et al.* [1978] defined S as

$$S(h) = a(h)S_p \tag{2.3}$$

where the water stress response function $a(h)$ is a prescribed dimensionless function (Fig. 2.1) of the soil water pressure head ($0 \leq a \leq 1$), and S_p is the potential water uptake rate [T^{-1}]. Figure 2.1. gives a schematic plot of the stress response function as used by *Feddes et al.* [1978]. Notice that water uptake is assumed to be zero close to saturation (i.e., wetter than some arbitrary "anaerobiosis point", h_1). For $h < h_4$ (the wilting point pressure head), water uptake is also assumed to be zero. Water uptake is considered optimal between pressure heads h_2 and h_3, whereas for pressure head between h_3 and h_4 (or h_1 and h_2), water uptake decreases (or increases) linearly with h. S_p is equal to the water uptake rate during periods of no water stress when $a(h) = 1$.

When the potential water uptake rate is equally distributed over a two-dimensional rectangular root domain, S_p becomes

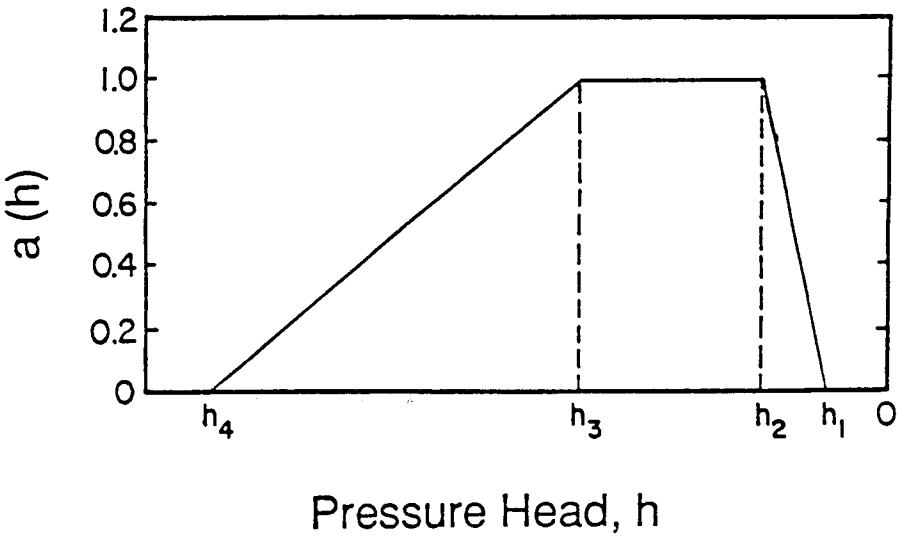

Fig. 2.1. Schematic of the plant water stress response function, $a(h)$, as used by *Feddes et al.* [1978].

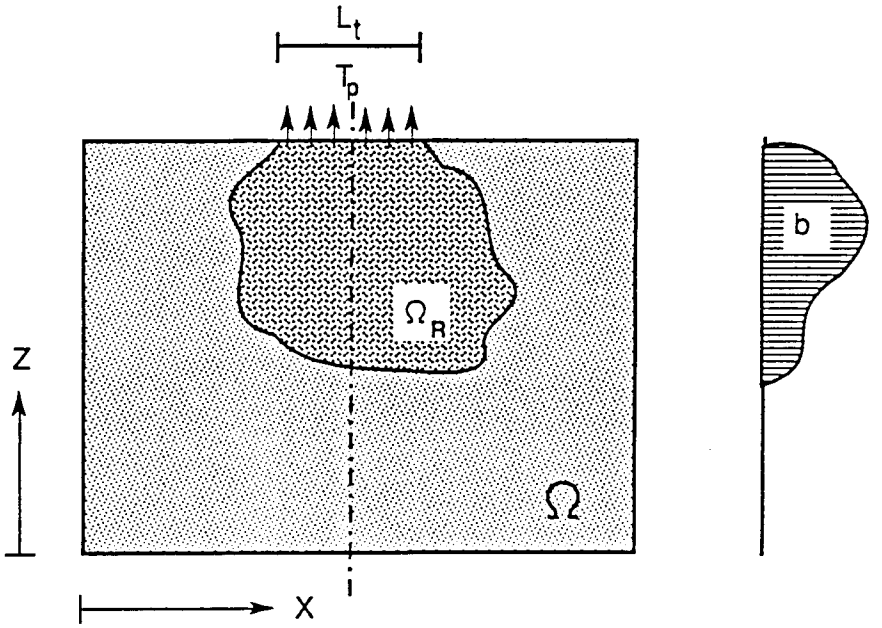

Fig. 2.2. Schematic of the potential water uptake distribution function, $b(x,z)$, in the soil root zone.

$$S_p = \frac{1}{L_x L_z} L_t T_p \tag{2.4}$$

where T_p is the potential transpiration rate [LT^{-1}], L_z is the depth [L] of the root zone, L_x is the width [L] of the root zone, and L_t is the width [L] of the soil surface associated with the transpiration process. Notice that S_p reduces to T_p/L_z when $L_t=L_x$.

Equation (2.4) may be generalized by introducing a non-uniform distribution of the potential water uptake rate over a root zone of arbitrary shape [*Vogel*, 1987]:

$$S_p = b(x,z) L_t T_p \tag{2.5}$$

where $b(x,z)$ is the normalized water uptake distribution [L^{-2}]. This function describes the spatial variation of the potential extraction term, S_p, over the root zone (Fig. 2.2), and is obtained from $b'(x,z)$ as follows

$$b(x,z) = \frac{b'(x,z)}{\int_{\Omega_R} b'(x,z)\,d\Omega} \tag{2.6}$$

where Ω_R is the region occupied by the root zone, and $b'(x,z)$ is an arbitrarily prescribed distribution function. Normalizing the uptake distribution ensures that $b(x,z)$ integrates to unity over the flow domain, i.e.,

$$\int_{\Omega_R} b(x,z)\,d\Omega = 1 \tag{2.7}$$

From (2.5) and (2.7) it follows that S_p is related to T_p by the expression

$$\frac{1}{L_t} \int_{\Omega_R} S_p\,d\Omega = T_p \tag{2.8}$$

The actual water uptake distribution is obtained by substituting (2.5) into (2.3):

$$S(h,x,z) = a(h,x,z)\,b(x,z)\,L_t\,T_p \tag{2.9}$$

whereas the actual transpiration rate, T_a, is obtained by integrating (2.9) as follows

$$T_a = \frac{1}{L_t} \int_{\Omega_R} S\,d\Omega = T_p \int_{\Omega_R} a(h,x,z)\,b(x,z)\,d\Omega \tag{2.10}$$

2.3. *The Unsaturated Soil Hydraulic Properties*

The unsaturated soil hydraulic properties in the SWMS_2D code are described by a set of closed-form equations resembling those of *van Genuchten* [1980] who used the statistical pore-size distribution model of *Mualem* [1976] to obtain a predictive equation for the unsaturated hydraulic conductivity function. The original van Genuchten equations were modified to add extra flexibility in the description of the hydraulic properties near saturation [*Vogel and Císlerová*, 1988]. The soil water retention, $\theta(h)$, and hydraulic conductivity, $K(h)$, functions in SWMS_2D are given by

$$\theta(h) = \begin{cases} \theta_a + \dfrac{\theta_m - \theta_a}{(1 + |\alpha h|^n)^m} & h < h_s \\ \theta_s & h \geq h_s \end{cases} \tag{2.11}$$

and

$$K(h) = \begin{cases} K_s K_r(h) & h \leq h_k \\ K_k + \dfrac{(h - h_k)(K_s - K_k)}{h_s - h_k} & h_k < h < h_s \\ K_s & h \geq h_s \end{cases} \tag{2.12}$$

respectively, where

$$K_r = \frac{K_k}{K_s} \left(\frac{S_e}{S_{ek}} \right)^{1/2} \left[\frac{F(\theta_r) - F(\theta)}{F(\theta_r) - F(\theta_k)} \right]^2 \tag{2.13}$$

$$F(\theta) = \left[1 - \left(\frac{\theta - \theta_a}{\theta_m - \theta_a} \right)^{1/m} \right]^m \tag{2.14}$$

$$m = 1 - 1/n \quad , \quad n > 1 \tag{2.15}$$

$$S_e = \frac{\theta - \theta_r}{\theta_s - \theta_r} \tag{2.16}$$

$$S_{ek} = \frac{\theta_k - \theta_r}{\theta_s - \theta_r} \tag{2.17}$$

in which θ_r and θ_s denote the residual and saturated water contents, respectively, and K_s is the saturated hydraulic conductivity. To increase the flexibility of the analytical expressions, and to allow for a non-zero air-entry value, h_s, the parameters θ_r and θ_s in the retention

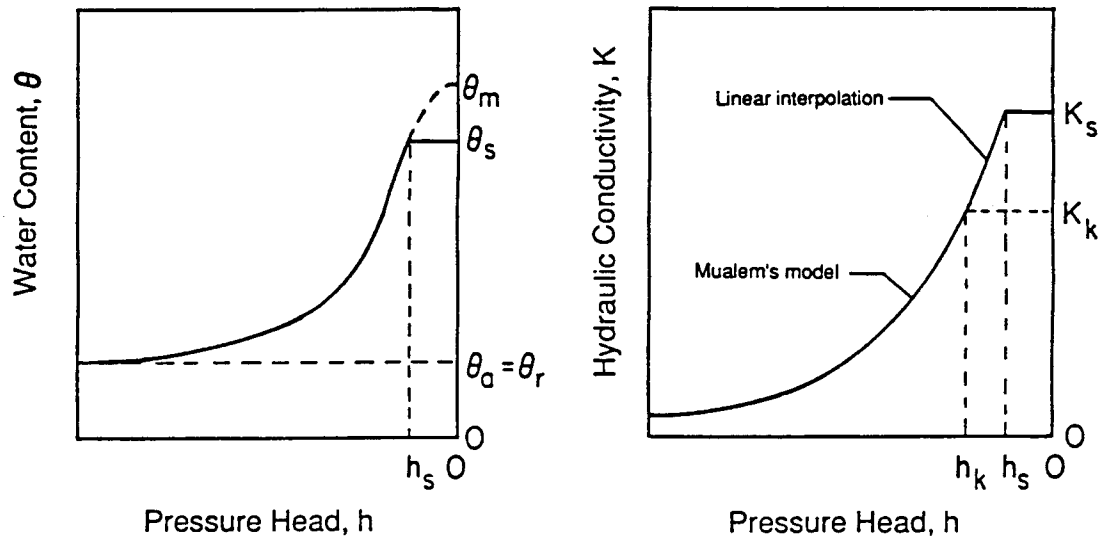

Fig. 2.3. Schematics of the soil water retention (a) and hydraulic conductivity (b) functions as given by equations (2.11) and (2.12), respectively.

function were replaced by the fictitious (extrapolated) parameters $\theta_a \leq \theta_r$ and $\theta_m \geq \theta_s$ as shown in Fig. 2.3. The approach maintains the physical meaning of θ_r and θ_s as measurable quantities. Equation (2.13) assumes that the predicted hydraulic conductivity function is matched to a measured value of the hydraulic conductivity, $K_k=K(\theta_k)$, at some water content, θ_k, less that or equal to the saturated water content, i.e., $\theta_k \leq \theta_s$ and $K_k \leq K_s$ [*Vogel and Císlerová*, 1988; *Luckner et al.*, 1989].

Inspection of (2.11) through (2.17) shows that the hydraulic characteristics contain 9 unknown parameters: θ_r, θ_s, θ_a, θ_m, α, n, K_s, K_k, and θ_k. When $\theta_a=\theta_r$, $\theta_m=\theta_k=\theta_s$ and $K_k=K_s$, the soil hydraulic functions reduce to the original expressions of *van Genuchten* [1980]:

$$\theta(h) = \begin{cases} \theta_r + \dfrac{\theta_s - \theta_r}{[1 + |\alpha h|^n]^m} & h<0 \\ \theta_s & h \geq 0 \end{cases} \qquad (2.18)$$

$$K(h) = \begin{cases} K_s K_r(h) & h<0 \\ K_s & h \geq 0 \end{cases} \tag{2.19}$$

where

$$K_r = S_e^{1/2}[1-(1-S_e^{1/m})^m]^2 \tag{2.20}$$

2.4. *Scaling of the Soil Hydraulic Functions*

SWMS_2D implements a scaling procedure designed to simplify the description of the spatial variability of the unsaturated soil hydraulic properties in the flow domain. The code assumes that the hydraulic variability in a given area can be approximated by means of a set of linear scaling transformations which relate the individual soil hydraulic characteristics $\theta(h)$ and $K(h)$ to reference characteristics $\theta^*(h^*)$ and $K^*(h^*)$. The technique is based on the similar media concept introduced by *Miller and Miller* [1956] for porous media which differ only in the scale of their internal geometry. The concept was extended by *Simmons et al.* [1979] to materials which differ in morphological properties, but which exhibit 'scale-similar' soil hydraulic functions. Three independent scaling factors are embodied in SWMS_2D. These three scaling parameters may be used to define a linear model of the actual spatial variability in the soil hydraulic properties as follows [*Vogel et al.*, 1991]:

$$\begin{aligned} K(h) &= \alpha_K K^*(h^*) \\ \theta(h) &= \theta_r + \alpha_\theta[\theta^*(h^*) - \theta_r^*] \\ h &= \alpha_h h^* \end{aligned} \tag{2.21}$$

in which, for the most general case, α_θ, α_h and α_K are mutually independent scaling factors for the water content, the pressure head and the hydraulic conductivity, respectively. Less general scaling methods arise by invoking certain relationships between α_θ, α_h and/or α_K.

For example, the original Miller-Miller scaling procedure is obtained by assuming $\alpha_\theta = 1$ (with $\theta_r^* = \theta_r$), and $\alpha_K = \alpha_h^{-2}$. A detailed discussion of the scaling relationships given by (2.21), and their application to the hydraulic description of heterogeneous soil profiles, is given by *Vogel et al.* [1991].

2.5. *Initial and Boundary Conditions*

The solution of Eq. (2.1) requires knowledge of the initial distribution of the pressure head within the flow domain, Ω:

$$h(x,z,t) = h_0(x,z) \qquad for\ t = 0 \tag{2.22}$$

where h_0 is a prescribed function of x and z.

SWMS_2D implements three types of conditions to describe system-independent interactions along the boundaries of the flow region. These conditions are specified pressure head (Dirichlet type) boundary conditions of the form

$$h(x,z,t) = \psi(x,z,t) \qquad for\ (x,z) \in \Gamma_D \tag{2.23}$$

specified flux (Neumann type) boundary conditions given by

$$-[K(K_{ij}^A \frac{\partial h}{\partial x_j} + K_{iz}^A)]n_i = \sigma_1(x,z,t) \qquad for\ (x,z) \in \Gamma_N \tag{2.24}$$

and specified gradient boundary conditions

$$(K_{ij}^A \frac{\partial h}{\partial x_j} + K_{iz}^A)n_i = \sigma_2(x,z,t) \qquad for\ (x,z) \in \Gamma_G \tag{2.25}$$

where Γ_D, Γ_N, and Γ_G indicate Dirichlet, Neumann, and gradient type boundary segments, respectively; ψ [L], σ_1 [LT^{-1}], and σ_2 [-] are prescribed functions of x, z and t; and n_i are the components of the outward unit vector normal to boundary Γ_N or Γ_G. As pointed out by *McCord* [1991], the use of the term "Neumann type boundary condition" for the flux boundary is not very appropriate since this term should hold for a gradient type condition

(see also Section 3.2 for solute transport). However, since the use of the Neumann condition is standard in the hydrologic literature [*Neuman*, 1972; *Neuman et al.*, 1974], we shall also use this term to indicate flux boundaries throughout this report. SWMS_2D implements the gradient boundary condition only in terms of a unit vertical hydraulic gradient simulating free drainage from a relatively deep soil profile. This situation is often observed in field studies of water flow and drainage in the vadose zone [*Sisson*, 1987; *McCord*, 1991]. *McCord* [1991] states that the most pertinent application of (2.25) is its use as a bottom outflow boundary condition for situations where the water table is situated far below the domain of interest.

In addition to the system-independent boundary conditions given by (2.23), (2.24), and (2.25), SWMS_2D considers three different types of system-dependent boundary conditions which cannot be defined a priori. One of these involves soil-air interfaces which are exposed to atmospheric conditions. The potential fluid flux across these interfaces is controlled exclusively by external conditions. However, the actual flux depends also on the prevailing (transient) soil moisture conditions. Soil surface boundary conditions may change from prescribed flux to prescribed head type conditions (and vice-versa). In the absence of surface ponding, the numerical solution of (2.1) is obtained by limiting the absolute value of the flux such that the following two conditions are satisfied [*Neuman et al.*, 1974]:

$$|K(K_{ij}^{A}\frac{\partial h}{\partial x_j}+K_{iz}^{A})n_i| \le E \tag{2.26}$$

and

$$h_A \le h \le h_S \tag{2.27}$$

where E is the maximum potential rate of infiltration or evaporation under the current atmospheric conditions, h is the pressure head at the soil surface, and h_A and h_S are, respectively, minimum and maximum pressure heads allowed under the prevailing soil conditions. The value for h_A is determined from the equilibrium conditions between soil water and atmospheric water vapor, whereas h_S is usually set equal to zero. SWMS_2D assumes that any excess water on the soil surface is immediately removed. When one of the

end points of (2.27) is reached, a prescribed head boundary condition will be used to calculate the actual surface flux. Methods of calculating E and h_A on the basis of atmospheric data have been discussed by *Feddes et al.* [1974].

A second type of system-dependent boundary condition considered in SWMS_2D is a seepage face through which water leaves the saturated part of the flow domain. In this case, the length of the seepage face is not known a priori. SWMS_2D assumes that the pressure head is always uniformly equal to zero along a seepage face. Additionally, the code assumes that water leaving the saturated zone across a seepage face is immediately removed by overland flow or some other removal process.

Finally, a third class of system-dependent boundary conditions in SWMS_2D concerns tile drains. Similarly as for seepage phase, SWMS_2D assumes that as long as a drain is located in the saturated zone, the pressure head along the drain will be equal to zero; the drain then acts as a pressure head sink. However, the drain will behave as a nodal sink/source with zero recharge when located in the unsaturated zone. More information can be found in Section 4.3.7.

3. SOLUTE TRANSPORT

3.1. *Governing Transport Equation*

The partial differential equation governing two-dimensional chemical transport during transient water flow in a variably saturated rigid porous medium is taken as

$$\frac{\partial \theta c}{\partial t} + \frac{\partial \rho s}{\partial t} = \frac{\partial}{\partial x_i}\left(\theta D_{ij} \frac{\partial c}{\partial x_j}\right) - \frac{\partial q_i c}{\partial x_i} + \mu_w \theta c + \mu_s \rho s + \gamma_w \theta + \gamma_s \rho - S c_s \qquad (3.1)$$

where c is the solution concentration [ML^{-3}], s is the adsorbed concentration [-], q_i is the i-th component of the volumetric flux [LT^{-1}], μ_w and μ_s are first-order rate constants for solutes in the liquid and solid phases [T^{-1}], respectively; γ_w and γ_s are zero-order rate constants for the liquid [$ML^{-3}T^{-1}$] and solid [T^{-1}] phases, respectively; ρ is the soil bulk density [ML^{-3}], S is the sink term in the water flow equation (2.1), c_s is the concentration of the sink term [ML^{-3}], and D_{ij} is the dispersion coefficient tensor [L^2T^{-1}]. The four zero- and first-order rate constants in (3.1) may be used to represent a variety of reactions or transformations including biodegradation, volatilization, precipitation and radioactive decay.

SWMS_2D assumes equilibrium interactions between the solution (c) and adsorbed (s) concentrations of the solute in the soil system. The adsorption isotherm relating s and c is described by a linear equation of the form

$$s = kc \qquad (3.2)$$

where k is an empirical constant [L^3M^{-1}].

The following continuity equation describes isothermal Darcian flow of water in a variably saturated porous medium

$$\frac{\partial \theta}{\partial t} = -\frac{\partial q_i}{\partial x_i} - S \qquad (3.3)$$

Substituting (3.2) and (3.3) into (3.1) gives

$$-\theta R\frac{\partial c}{\partial t} - q_i\frac{\partial c}{\partial x_i} + \frac{\partial}{\partial x_i}(\theta D_{ij}\frac{\partial c}{\partial x_j}) + Fc + G = 0 \tag{3.4}$$

where

$$\begin{aligned} F &= \mu_w\theta + \mu_s\rho k + S \\ G &= \gamma_w\theta + \gamma_s\rho - Sc_s \end{aligned} \tag{3.5}$$

and where the retardation factor R [-] is defined as

$$R = 1 + \frac{\rho k}{\theta} \tag{3.6}$$

In order to solve equation (3.4), it is necessary to know the water content θ and the volumetric flux q_i. Both variables are obtained from solutions of the Richards' equation.

3.2. *Initial and Boundary Conditions*

The solution of (3.4) requires knowledge of the initial concentration within the flow region, Ω, i.e.,

$$c(x,z,0) = c_i(x,z) \tag{3.7}$$

where c_i is a prescribed function of x and z.

Two types of boundary conditions (Dirichlet and Cauchy type conditions) can be specified along the boundary of Ω. First-type (or Dirichlet type) boundary conditions prescribe the concentration along a boundary segment Γ_D:

$$c(x,z,t) = c_0(x,z,t) \qquad for\ (x,z) \in \Gamma_D \tag{3.8}$$

whereas third-type (Cauchy type) boundary conditions may be used to prescribe the solute flux along a boundary segment Γ_C as follows:

$$-\theta D_{ij}\frac{\partial c}{\partial x_j}n_i + q_i n_i c = q_i n_i c_0 \qquad for\ (x,z) \in \Gamma_C \tag{3.9}$$

in which $q_i n_i$ represents the outward fluid flux, n_i is the outward unit normal vector, and c_0 is the concentration of the incoming fluid. In some cases, for example when Γ_C is an impermeable boundary ($q_i n_i = 0$) or water flow is directed out of the region, (3.9) reduces to a second-type (Neumann type) boundary condition of the form:

$$\theta D_{ij}\frac{\partial c}{\partial x_j}n_i = 0 \qquad for\ (x,z) \in \Gamma_N \tag{3.10}$$

3.3. *Dispersion Coefficient*

The components of the dispersion tensor, D_{ij}, in (3.1) are given by [*Bear*,1972]

$$\theta D_{ij} = D_T |q| \delta_{ij} + (D_L - D_T)\frac{q_j q_i}{|q|} + \theta D_d \tau \delta_{ij} \tag{3.11}$$

where D_d is the ionic or molecular diffusion coefficient in free water [L^2T^{-1}], τ is a tortuosity factor [-], $|q|$ is the absolute value of the Darcian fluid flux density [LT^{-1}], δ_{ij} is the Kronecker delta function ($\delta_{ij}=1$ if $i=j$, and $\delta_{ij}=0$ if $i \neq j$), and D_L and D_T are the longitudinal and transverse dispersivities, respectively [L]. The individual components of the dispersion tensor for two-dimensional transport are as follows:

$$\begin{aligned}
\theta D_{xx} &= D_L\frac{q_x^2}{|q|} + D_T\frac{q_z^2}{|q|} + \theta D_d \tau \\
\theta D_{zz} &= D_L\frac{q_z^2}{|q|} + D_T\frac{q_x^2}{|q|} + \theta D_d \tau \\
\theta D_{xz} &= (D_L - D_T)\frac{q_x q_z}{|q|}
\end{aligned} \tag{3.12}$$

The tortuosity factor is evaluated in SWMS_2D as a function of the water content using the relationship of *Millington and Quirk* [1961]:

$$\tau = \frac{\theta^{7/3}}{\theta_s^2} \tag{3.13}$$

4. NUMERICAL SOLUTION OF THE WATER FLOW EQUATION

The Galerkin finite element method with linear basis functions is used to obtain a solution of the flow equation (2.1) subject to the imposed initial and boundary conditions. Since the Galerkin method is relatively standard and has been covered in detail elsewhere [*Neuman*, 1975; *Zienkiewicz*, 1977; *Pinder and Gray*, 1977], only the most pertinent steps in the solution process are given here.

4.1. *Space Discretization*

The flow region is divided into a network of triangular elements. The corners of these elements are taken to be the nodal points. The dependent variable, the pressure head function $h(x,z,t)$, is approximated by a function $h'(x,z,t)$ as follows

$$h'(x,z,t) = \sum_{n=1}^{N} \phi_n(x,z)\, h_n(t) \tag{4.1}$$

where ϕ_n are piecewise linear basis functions satisfying the condition $\phi_n(x_m,z_m) = \delta_{nm}$, h_n are unknown coefficients representing the solution of (2.1) at the nodal points, and N is the total number of nodal points.

The Galerkin method postulates that the differential operator associated with the Richards' equation (2.1) is orthogonal to each of the N basis functions, i.e.,

$$\int_{\Omega} \{ \frac{\partial \theta}{\partial t} - \frac{\partial}{\partial x_i} [K(K_{ij}^A \frac{\partial h}{\partial x_j} + K_{iz}^A)] + S \} \phi_n d\Omega = 0 \tag{4.2}$$

Applying Green's first identity to (4.2), and replacing h by h', leads to

$$\sum_e \int_{\Omega_e} \left(\frac{\partial \theta}{\partial t}\phi_n + KK_{ij}^A \frac{\partial h'}{\partial x_j}\frac{\partial \phi_n}{\partial x_i}\right) d\Omega =$$

$$\sum_e \int_{\Gamma_e} K\left(K_{ij}^A \frac{\partial h'}{\partial x_j} + K_{iz}^A\right) n_i \phi_n d\Gamma + \sum_e \int_{\Omega_e} \left(-KK_{iz}^A \frac{\partial \phi_n}{\partial x_i} - S\phi_n\right) d\Omega \tag{4.3}$$

where Ω_e represents the domain occupied by element e, and Γ_e is a boundary segment of element e. Natural flux-type (Neumann) and gradient type boundary conditions can be immediately incorporated into the numerical scheme by specifying the line integral in equation (4.3).

After imposing additional simplifying assumptions to be discussed later, and performing integration over the elements, the procedure leads to a system of time-dependent ordinary differential equations with nonlinear coefficients. In matrix form, these equations are given by

$$[F]\frac{d\{\theta\}}{dt} + [A]\{h\} = \{Q\} - \{B\} - \{D\} \tag{4.4}$$

where

$$A_{nm} = \sum_e K_l K_{ij}^A \int_{\Omega_e} \phi_l \frac{\partial \phi_n}{\partial x_i}\frac{\partial \phi_m}{\partial x_j} d\Omega \tag{4.5}$$

$$= \sum_e \frac{\kappa}{4A_e}\bar{K}\ \left[K_{xx}^A b_m b_n + K_{xz}^A (c_m b_n + b_m c_n) + K_{zz}^A c_n c_m\right]$$

$$B_n = \sum_e K_l K_{iz}^A \int_{\Omega_e} \phi_l \frac{\partial \phi_n}{\partial x_i} d\Omega = \sum_e \frac{\kappa}{2}\bar{K}\ (K_{xz}^A b_n + K_{zz}^A c_n) \tag{4.6}$$

$$F_{nm} = \delta_{nm} \sum_e \int_{\Omega_e} \phi_n d\Omega = \delta_{nm} \sum_e \frac{\kappa}{3} A_e \tag{4.7}$$

$$Q_n = -\sum_e \sigma_{1_l} \int_{\Gamma_e} \phi_l \phi_n d\Gamma = -\sum_e \sigma_n \lambda_n \tag{4.8}$$

$$D_n = \sum_e S_l \int_{\Omega_e} \phi_l \phi_n d\Omega = \sum_e \frac{\kappa}{12} A_e (3\bar{S} + S_n) \tag{4.9}$$

where the overlined variables represent average values over an element e, the subscripts i and j are space direction indices (i,j = 1,2), and

$$l = 1,2,...,N \quad m = 1,2,...,N \quad n = 1,2,...,N$$

$$\begin{aligned} b_i &= z_j - z_k & c_i &= x_k - x_j \\ b_j &= z_k - z_i & c_j &= x_i - x_k \\ b_k &= z_i - z_j & c_k &= x_j - x_i \end{aligned} \tag{4.10}$$

$$A_e = \frac{c_k b_j - c_j b_k}{2} \qquad \bar{K} = \frac{K_i + K_j + K_k}{3} \qquad \bar{S} = \frac{S_i + S_j + S_k}{3}$$

Equation (4.8) is valid for a flux-type boundary condition. For a gradient-type boundary condition the variable σ_1 in (4.8) must be replaced by the product of the hydraulic conductivity K and the prescribed gradient σ_2 (=1). Equations (4.5) through (4.9) hold for flow in a two-dimensional Cartesian (x,z) domain, as well as for flow in an axisymmetric (x,z) system in which x is used as the radial coordinate. For plane flow we have

$$\kappa = 1 \qquad \lambda_n = \frac{L_n}{2} \tag{4.11}$$

while for axisymmetric flow

$$\kappa = 2\pi \frac{x_i + x_j + x_k}{3} \qquad \lambda_n = L_n \pi \frac{x'_n + 2x_n}{3} \tag{4.12}$$

The subscripts i, j and k in equations (4.10) and (4.12) represent the three corners of a triangular element e. A_e is the area of element e, $\bar{K}$ and $\bar{S}$ are the average hydraulic conductivity and root water extraction values over element e, L_n is the length of the boundary segment connected to node n, and x'_n is the x-coordinate of a boundary node adjacent to node n. The symbol σ_n in equation (4.8) stands for the flux [LT^{-1}] across the boundary in the vicinity of boundary node n (positive when directed outward of the system). The boundary flux is assumed to be uniform over each boundary segment. The entries of the vector Q_n are zero at all internal nodes which do not act as sources or sinks for water.

The numerical procedure leading to (4.4) incorporates two important assumptions in addition to those related to the Galerkin finite element approach. One assumption concerns the time derivatives of the nodal values of the water content in (4.4). These time derivatives were weighted according to

$$\frac{d\theta_n}{dt} = \frac{\sum_e \int_{\Omega_e} \frac{\partial \theta}{\partial t} \phi_n d\Omega}{\sum_e \int_{\Omega_e} \phi_n d\Omega} \tag{4.13}$$

This assumption implements mass-lumping which has been shown to improve the rate of convergence of the iterative solution process [e.g., *Neuman*, 1973].

A second assumption in the numerical scheme is related to the anisotropy tensor $\mathbf{K}^A$ which is taken to be constant over each element. By contrast, the water content θ, the hydraulic conductivity K, the soil water capacity C, and the root water extraction rate S, at a given point in time are assumed to vary linearly over each element, e. For example, the water content is expanded over each element as follows:

$$\theta(x,z) = \sum_{n=1}^{3} \theta(x_n,z_n)\,\phi_n(x,z) \qquad for\ (x,z) \in \Omega_e \tag{4.14}$$

where n stands for the corners of element e. The advantage of linear interpolation is that no numerical integration is needed to evaluate the coefficients in (4.4).

4.2. *Time Discretization*

Integration of (4.4) in time is achieved by discretizing the time domain into a sequence of finite intervals and replacing the time derivatives by finite differences. An implicit (backward) finite difference scheme is used for both saturated and unsaturated conditions:

$$[F]\frac{\{\theta\}_{j+1} - \{\theta\}_j}{\Delta t_j} + [A]_{j+1}\{h\}_{j+1} = \{Q\}_j - \{B\}_{j+1} - \{D\}_j \tag{4.15}$$

where $j+1$ denotes the current time level at which the solution is being considered, j refers to the previous time level, and $\Delta t_j = t_{j+1} - t_j$. Equation (4.15) represents the final set of algebraic equations to be solved. Since the coefficients θ, A, B, D, and Q (Q only for gradient-type boundary conditions) are functions of h, the set of equations is generally highly nonlinear. Note that vectors D and Q are evaluated at the old time level.

4.3. *Numerical Solution Strategy*

4.3.1. *Iterative Process*

Because of the nonlinear nature of (4.15), an iterative process must be used to obtain solutions of the global matrix equation at each new time step. For each iteration a system of linearized algebraic equations is first derived from (4.15) which, after incorporation of the boundary conditions, is solved using either Gaussian elimination or the conjugate gradient method (see Section 6.5). The Gaussian elimination process takes advantage of

the banded and symmetric features of the coefficient matrices in (4.15). After inversion, the coefficients in (4.15) are re-evaluated using the first solution, and the new equations are again solved. The iterative process continues until a satisfactory degree of convergence is obtained, i.e., until at all nodes in the saturated (or unsaturated) region the absolute change in pressure head (or water content) between two successive iterations becomes less than some small value determined by the imposed absolute pressure head (or water content) tolerance [*Šimůnek and Suarez*, 1993]. The first estimate (at zero iteration) of the unknown pressure heads at each time step is obtained by extrapolation from the pressure head values at the previous two time levels.

4.3.2. *Treatment of the Water Capacity Term*

The iteration process is extremely sensitive to the method used for evaluating the water content term ($\Delta\theta/\Delta t$) in equation (4.15). The present version of SWMS_2D code uses the "mass-conservative" method proposed by *Celia et al.* [1990]. Their method has been shown to provide excellent results in terms of minimizing the mass balance error. The mass-conservative method proceeds by separating the water content term into two parts:

$$[F]\frac{\{\theta\}_{j+1}-\{\theta\}_j}{\Delta t_j}=[F]\frac{\{\theta\}_{j+1}^{k+1}-\{\theta\}_{j+1}^{k}}{\Delta t_j}+[F]\frac{\{\theta\}_{j+1}^{k}-\{\theta\}_j}{\Delta t_j} \tag{4.16}$$

where $k+1$ and k denote the current and previous iteration levels, respectively; and $j+1$ and j the current and previous time levels, respectively. Notice that the second term on the right hand side of (4.16) is known prior to the current iteration. The first term on the right hand side can be expressed in terms of the pressure head, so that (4.16) becomes

$$[F]\frac{\{\theta\}_{j+1}-\{\theta\}_j}{\Delta t_j}=[F][C]_{j+1}\frac{\{h\}_{j+1}^{k+1}-\{h\}_{j+1}^{k}}{\Delta t_j}+[F]\frac{\{\theta\}_{j+1}^{k}-\{\theta\}_j}{\Delta t_j} \tag{4.17}$$

where $C_{nm}=\delta_{nm}C_n$, in which C_n represents the nodal value of the soil water capacity. The first term on the right hand side of (4.17) should vanish at the end of the iteration process

if the numerical solution converges. This particular feature guarantees relatively small mass balance errors in the solution.

4.3.3. *Time Control*

Three different time discretizations are introduced in SWMS_2D: (1) time discretizations associated with the numerical solution, (2) time discretizations associated with the implementation of boundary conditions, and (3) time discretizations which provide printed output of the simulation results (e.g., nodal values of dependent variables, water and solute mass balance components, and other information about the flow regime).

Discretizations 2 and 3 are mutually independent; they generally involve variable time steps as described in the input data file. Discretization 1 starts with a prescribed initial time increment, Δt. This time increment is automatically adjusted at each time level according to the following rules [*Mls*, 1982; *Vogel*, 1987]:

a. Discretization 1 must coincide with time values resulting from discretizations 2 and 3.
b. Time increments cannot become less than a preselected minimum time step, Δt_{min}, nor exceed a maximum time step, Δt_{max} (i.e., $\Delta t_{min} \leq \Delta t \leq \Delta t_{max}$).
c. If, during a particular time step, the number of iterations necessary to reach convergence is ≤ 3, the time increment for the next time step is increased by multiplying Δt by a predetermined constant >1 (usually between 1.1 and 1.5). If the number of iterations is ≥ 7, Δt for the next time level is multiplied by a constant <1 (usually between 0.3 and 0.9).
d. If, during a particular time step, the number of iterations at any time level becomes greater than a prescribed maximum (usually between 10 and 50), the iterative process for that time level is terminated. The time step is subsequently reset to $\Delta t/3$, and the iterative process restarted.

The selection of optimal time steps, Δt, is also influenced by the solution scheme for solute transport (see Section 5.3.6.).

4.3.4. *Treatment of Pressure Head Boundary Conditions*

Finite element equations corresponding to Dirichlet nodes where the pressure head is prescribed can, at least in principle, be eliminated from the global matrix equation. An alternative and numerically simpler approach is to replace the Dirichlet finite element equations by dummy expressions of the form [*Neuman*, 1974]

$$\delta_{nm} h_m = \psi_n \tag{4.18}$$

where δ_{nm} is the Kronecker delta and ψ_n is the prescribed value of the pressure head at node n. The values of h_n in all other equations are set equal to ψ_n and the appropriate entries containing ψ_n in the left hand side matrix are incorporated into the known vector on the right-hand side of the global matrix equation. When done properly, this rearrangement will preserve symmetry in the matrix equation. This procedure is applied only when Gaussian elimination is used to solve the matrix equation. When the conjugate gradient solver is used, then the finite element equation representing the Dirichlet node is modified as follows. The right hand side of this equation is set equal to the prescribed pressure head multiplied by a large number (10^{30}) and entry on the left hand side representing the Dirichlet node is set equal to this large number. After solving for all pressure heads, the value of the flux Q_n can be calculated explicitly and accurately from the original finite element equation associated with node n (e.g., *Lynch*, 1984).

4.3.5. *Flux and Gradient Boundary Conditions*

The values of the fluxes Q_n at nodal points along prescribed flux and gradient boundaries are computed according to equation (4.8). Internal nodes which act as Neumann type sources or sinks have values of Q_n equal to the imposed fluid injection or extraction rate.

4.3.6. *Atmospheric Boundary Conditions and Seepage Faces*

Atmospheric boundaries are simulated by applying either prescribed head or prescribed flux boundary conditions depending upon whether equation (2.26) or (2.27) is satisfied [*Neuman*, 1974]. If (2.27) is not satisfied, node n becomes a prescribed head boundary. If, at any point in time during the computations, the calculated flux exceeds the specified potential flux in (2.26), the node will be assigned a flux equal to the potential value and treated again as a prescribed flux boundary.

All nodes expected to be part of a seepage face during code execution must be identified a priori. During each iteration, the saturated part of a potential seepage face is treated as a prescribed pressure head boundary with $h=0$, while the unsaturated part is treated as a prescribed flux boundary with $Q=0$. The lengths of the two surface segments are continually adjusted [*Neuman*, 1974] during the iterative process until the calculated values of Q (equation (4.8)) along the saturated part, and the calculated values of h along the unsaturated part, are all negative, thus indicating that water is leaving the flow region through the saturated part of the surface boundary only.

4.3.7. *Tile Drains as Boundary Condition*

The representation of tile drains as boundary conditions is based on studies by *Vimoke et al.* [1963] and *Fipps et al.* [1986]. The approach uses results of electric analog experiments conducted by *Vimoke and Taylor* [1962] who reasoned that drains can be represented by nodal points in a regular finite element mesh, provided adjustments are made in the hydraulic conductivity, K, of neighboring elements. The adjustments should correspond to changes in the electric resistance of conducting paper as follows

$$K_{drain} = K C_d \tag{4.19}$$

where K_{drain} is the adjusted conductivity [LT^{-1}], and C_d is the correction factor [-]. C_d is determined from the ratio of the effective radius, d [L], of the drain to the side length, D

[L], of the square formed by finite elements surrounding the drain node [*Vimoke at al.*, 1962]:

$$C_d = \frac{Z_0'}{Z_0} \approx \frac{\sqrt{\mu_0/\epsilon_0}}{138\log_{10}\rho_d + 6.48 - 2.34A - 0.48B - 0.12C} \tag{4.20}$$

where Z_0' is the characteristic impedance of free space ($\approx$376.7 ohms), μ_0 is the permeability of free space, ϵ_0 is the permittivity of free space, and Z_0 is the characteristic impedance of a transmission line analog of the drain. The coefficients in (4.20) are given by

$$\begin{aligned} \rho_d &= \frac{D}{d} \\ A &= \frac{1 + 0.405\rho_d^{-4}}{1 - 0.405\rho_d^{-4}} \\ B &= \frac{1 + 0.163\rho_d^{-8}}{1 - 0.163\rho_d^{-8}} \\ C &= \frac{1 + 0.067\rho_d^{-12}}{1 - 0.067\rho_d^{-12}} \end{aligned} \tag{4.21}$$

where d is the effective drain diameter to be calculated from the number and size of small openings in the drain tube [*Mohammad and Skaggs*, 1984], and D is the size of the square in the finite element mesh surrounding the drain having adjusted hydraulic conductivities. The approach above assumes that the node representing a drain must be surrounded by finite elements (either triangular or quadrilateral) which form a square whose hydraulic conductivities are adjusted according to (4.19). This method of implementing drains by means of a boundary condition gives an efficient, yet relatively accurate, prediction of the hydraulic head in the immediate vicinity of the drain, as well as of the drain flow rate [*Fipps et al.*, 1986]. More recent studies have shown that the correction factor C_d could be further reduced by a factor of 2 [*Rogers and Fouss*, 1989] or 4 [*Tseng*, 1994, personal communication]. These two studies compared numerical simulations of the flow of ponded

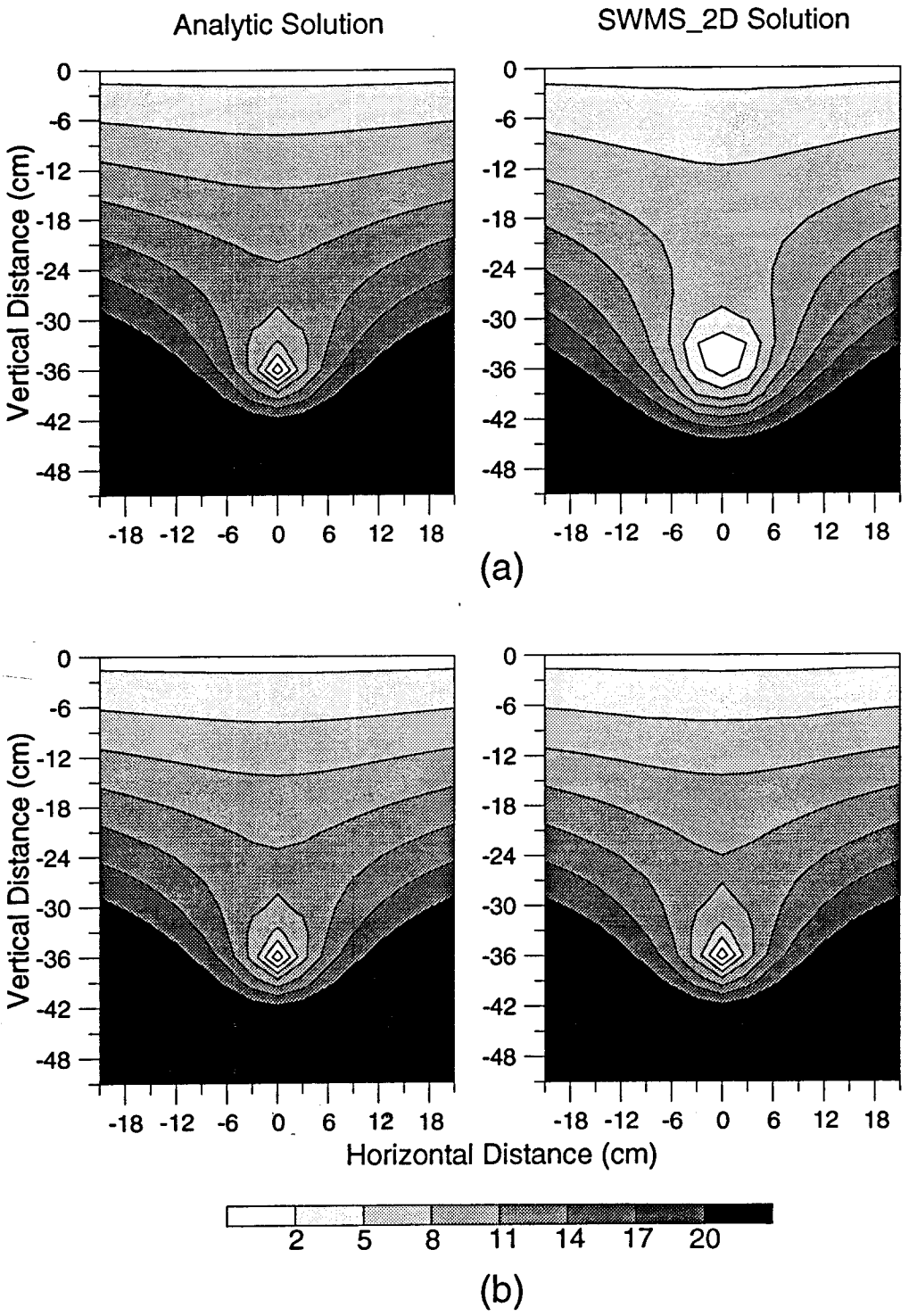

Fig. 4.1. Pressure head contours (cm) for flow of ponded water into a tile drain system as calculated analytically and using SWMS_2D with (a) correction factor C_d and (b) correction factor $C_d/4$.

water into a tile drain system with an analytical solution given by *Kirkham* [1949]. Pressure head contours calculated numerically with the original correction factor C_d (4.20), as well as with the additionally reduced correction factor $C_d/4$, are compared with the analytical results in Fig. 4.1 [*Tseng*, 1994, personal communication]. We emphasize that Figure 4.1 represents only one example; further studies of the single grid point representation of a subsurface drain may be needed, especially for transient variably-saturated flow conditions.

4.3.8. *Water Balance Computations*

The SWMS_2D code performs water balance computations at prescribed times for several preselected subregions of the flow domain. The water balance information for each subregion consists of the actual volume of water, V, in that subregion, and the rate, O, of inflow or outflow to or from the subregion. V and O are given by

$$V = \sum_e \kappa A_e \frac{\theta_i + \theta_j + \theta_k}{3} \tag{4.22}$$

and

$$O = \frac{V_{new} - V_{old}}{\Delta t} \tag{4.23}$$

respectively, where θ_i, θ_j, and θ_k are water contents evaluated at the corner nodes of element e, and where V_{new} and V_{old} are volumes of water in the subregion computed at the current and previous time levels, respectively. The summation in (4.22) is taken over all elements within the subregion.

The absolute error in the mass balance is calculated as

$$\epsilon_a^w = V_t - V_0 + \int_0^t T_a L_t dt - \int_0^t \sum_{n_r} Q_n dt \tag{4.24}$$

where V_t and V_0 are the volumes of water in the flow domain at time t and zero, respectively, as calculated with (4.22). The third term on the right-hand side represents the

cumulative root water uptake amount, while the fourth term gives the cumulative flux through nodes, n_r, located along the boundary of the flow domain or at internal source and sink nodes.

The accuracy of the numerical solution is evaluated in terms of the relative error, ϵ_r^w [%], in the water mass balance as follows:

$$\epsilon_r^w = \frac{|\epsilon_a^w|}{\max\left[\sum_e |V_t^e - V_0^e|, \int_0^t T_a L_t dt + \int_0^t \sum_{n_r} |Q_n| dt\right]} 100 \qquad (4.25)$$

where V_t^e and V_0^e are the volumes of water in element e at times t and zero, respectively. Note that SWMS_2D does not relate the absolute error to the volume of water in the flow domain, but instead to the maximum value of two quantities. The first quantity represents the sum of the absolute changes in water content over all elements, whereas the second quantity is the sum of the absolute values of all fluxes in and out of the flow domain. The above error criterion is much stricter than the usual criterion involving the total volume of water in the flow domain. This is because the cumulative boundary fluxes are often much smaller than the volume in the domain, especially at the beginning of the simulation.

4.3.9. *Computation of Nodal Fluxes*

Components of the Darcian flux are computed at each time level during the simulation only when the water flow and solute transport equations are solved simultaneously. When the flow equation is being solved alone, the flux components are calculated only at selected print times. The x- and z-components of the nodal fluxes are computed for each node n according to:

$$q_x = -\frac{K_n}{N_e}\sum_{e_n}\left[\frac{\gamma_i^x h_i + \gamma_j^x h_j + \gamma_k^x h_k}{2A_e} + K_{xz}^A\right]$$

$$q_z = -\frac{K_n}{N_e}\sum_{e_n}\left[\frac{\gamma_i^z h_i + \gamma_j^z h_j + \gamma_k^z h_k}{2A_e} + K_{zz}^A\right] \tag{4.26}$$

$$\gamma_n^x = K_{xx}^A b_n + K_{xz}^A c_n$$

$$\gamma_n^z = K_{xz}^A b_n + K_{zz}^A c_n$$

where N_e is the number of sub-elements e_n adjacent to node n. Einstein's summation convention is not used in (4.26).

4.3.10. *Water Uptake by Plant Roots*

SWMS_2D considers the root zone to consist of all nodes, n, for which the potential root water uptake distribution, b (see Section 2.2), is greater than zero. The root water extraction rate is assumed to vary linearly over each element; this leads to approximation (4.9) for the root water extraction term D_n in the global matrix equation. The values of actual root extraction rate S_n in (4.9) are evaluated with (2.9). In order to speed up the calculations, the extraction rates S_n are calculated at the old time level and are not updated during the iterative solution process at a given time step. SWMS_2D calculates the total rate of transpiration per unit soil surface length using the equation

$$T_a = \frac{1}{L_t}\sum_e \kappa A_e \bar{S} \tag{4.27}$$

in which the summation takes place over all elements within the root zone.

4.3.11. *Evaluation of the Soil Hydraulic Properties*

At the beginning of a numerical simulation, SWMS_2D generates for each soil type in the flow domain a table of water contents, hydraulic conductivities, and specific water capacities from the specified set of hydraulic parameters. The values of θ_i, K_i and C_i in the table are evaluated at prescribed pressure heads h_i within a specified interval (h_a, h_b). The entries in the table are generated such that

$$\frac{h_{i+1}}{h_i} = constant \tag{4.28}$$

which means that the spacing between two consecutive pressure head values increases in a logarithmic fashion. Values for the hydraulic properties, $\theta(h)$, $K(h)$ and $C(h)$, are computed during the iterative solution process using linear interpolation between the entries in the table. If an argument h falls outside the prescribed interval (h_a, h_b), the hydraulic characteristics are evaluated directly from the hydraulic functions, i.e., without interpolation. The above interpolation technique was found to be much faster computationally than direct evaluation of the hydraulic functions over the entire range of pressure heads, except when very simple hydraulic models were used.

4.3.12. *Implementation of Hydraulic Conductivity Anisotropy*

Since the hydraulic conductivity anisotropy tensor, $\mathbf{K}^A$, is assumed to be symmetric, it is possible to define at any point in the flow domain a local coordinate system for which the tensor $\mathbf{K}^A$ is diagonal (i.e., having zeroes everywhere except on the diagonal). The diagonal entries K_1^A and K_2^A of $\mathbf{K}^A$ are referred to as the principal components of $\mathbf{K}^A$.

The SWMS_2D code permits one to vary the orientation of the local principal directions from element to element. For this purpose, the local coordinate axes are subjected to a rotation such that they coincide with the principal directions of the tensor $\mathbf{K}^A$. The principal components K_1^A and K_2^A, together with the angle ω between the principal direction of K_1^A and the x-axis of the global coordinate system, are specified for each

element. Each locally determined tensor $\mathbf{K}^A$ is transformed to the global (x,z) coordinate system at the beginning of the simulation using the following rules:

$$\begin{aligned} K_{xx}^A &= K_1^A \cos^2\omega + K_2^A \sin^2\omega \\ K_{zz}^A &= K_1^A \sin^2\omega + K_2^A \cos^2\omega \\ K_{xz}^A &= (K_2^A - K_1^A)\sin\omega \cos\omega \end{aligned} \tag{4.29}$$

4.3.12. *Steady-State Analysis*

All transient flow problems are solved by time marching until a prescribed time is reached. The steady-state problem can be solved in the same way, i.e., by time marching until two successive solutions differ less than some prescribed pressure head tolerance. SWMS_2D implements a faster way of obtaining the steady-state solution without having to go through a large number of time steps. The steady-state solution for a set of imposed boundary conditions is obtained directly during one set of iterations at the first time step by equating the time derivative term in the Richards' equation (2.1) to zero.

5. NUMERICAL SOLUTION OF THE SOLUTE TRANSPORT EQUATION

The Galerkin finite element method is also used to solve solute transport equation (3.4) subject to appropriate initial and boundary conditions. The solution procedure below largely parallels the approach used for the flow equation.

5.1. *Space Discretization*

The dependent variable, the concentration function $c(x,z,t)$, is approximated by a finite series $c'(x,z,t)$ of the form

$$c'(x,z,t) = \sum_{n=1}^{N} \phi_n(x,z)\, c_n(t) \tag{5.1}$$

where ϕ_n are the selected linear basis functions, c_n are the unknown time dependent coefficients which represent solutions of (3.4) at the finite element nodal points and, as before, N is the total number of nodal points. Application of the standard Galerkin method leads to the following set of N equations

$$\int_{\Omega} [-\theta R \frac{\partial c}{\partial t} - q_i \frac{\partial c}{\partial x_i} + \frac{\partial}{\partial x_i}(\theta D_{ij} \frac{\partial c}{\partial x_j}) + Fc + G]\phi_n d\Omega = 0 \tag{5.2}$$

Application of Green's theorem to the second derivatives in (5.2) and substitution of c by c' results in the following system of time-dependent differential equations

$$\begin{aligned} \sum_e \int_{\Omega_e} [(-\theta R \frac{\partial c'}{\partial t} - q_i \frac{\partial c'}{\partial x_i} + Fc' + G)\phi_n - \theta D_{ij} \frac{\partial c'}{\partial x_j} \frac{\partial \phi_n}{\partial x_i}] d\Omega \\ + \sum_e \int_{\Gamma_N^e} \theta D_{ij} \frac{\partial c'}{\partial x_j} n_i \phi_n d\Gamma = 0 \end{aligned} \tag{5.3}$$

or in matrix form:

$$[Q]\frac{d\{c\}}{dt} + [S]\{c\} + \{f\} = -\{Q^D\} \tag{5.4}$$

where

$$Q_{nm} = \sum_e (-\theta R)_l \int_{\Omega_e} \phi_l \phi_n \phi_m d\Omega = -\sum_e \frac{\kappa A_e}{12}(3\overline{\theta R} + \theta_n R_n)\delta_{nm} \tag{5.5}$$

$$\begin{aligned} S_{nm} &= \sum_e [(-q_i)_l \int_{\Omega_e} \phi_l \phi_n \frac{\partial \phi_m}{\partial x_i} d\Omega - (\theta D_{ij})_l \int_{\Omega_e} \phi_l \frac{\partial \phi_n}{\partial x_i}\frac{\partial \phi_m}{\partial x_j} d\Omega + F_l \int_{\Omega_e} \phi_l \phi_n \phi_m d\Omega] \\ &= \sum_e \{-\frac{\kappa b_m}{24}(3\overline{q_x} + q_{xn}) - \frac{\kappa c_m}{24}(3\overline{q_z} + q_{zn}) + \frac{\kappa A_e}{60}(3\bar{F} + F_n + F_m)(1 + \delta_{nm}) - \\ &\quad - \frac{\kappa}{4A_e}[b_m b_n \overline{\theta D_{xx}} + (b_m c_n + c_m b_n)\overline{\theta D_{xz}} + c_m c_n \overline{\theta D_{zz}})\} \end{aligned} \tag{5.6}$$

$$f_n = \sum_e G_l \int_{\Omega_e} \phi_l \phi_n d\Omega = \sum_e \frac{\kappa A_e}{12}(3\bar{G} + G_n) \tag{5.7}$$

in which the overlined variables represent average values over a given element e. The notation in the above equations is similar as in (4.10). The boundary integral in (5.3) represents the dispersive flux, Q_n^D, across the boundary and will be discussed later in Section 5.3.4.

The derivation of equations (5.5) through (5.7) used several important assumptions in addition to those involved in the Galerkin finite element approach [*Huyakorn and Pinder*, 1983; *van Genuchten*, 1978]. First, the different coefficients under the integral signs (θR, q_i, θD_{ij}, F, G) were expanded linearly over each element, similarly as for the dependent variable, i.e., in terms of their nodal values and associated basis functions. Second, mass lumping was invoked by redefining the nodal values of the time derivative in (5.4) as weighted averages over the entire flow region:

$$\frac{dc_n}{dt} = \frac{\sum_e \int_{\Omega_e} \theta R \frac{\partial c}{\partial t} \phi_n d\Omega}{\sum_e \int_{\Omega_e} \theta R \phi_n d\Omega} \tag{5.8}$$

5.2. *Time Discretization*

The Galerkin method is used only for approximating the spatial derivatives while the time derivatives are discretized by means of finite differences. A first-order approximation of the time derivatives leads to the following set of algebraic equations:

$$[Q]_{j+\epsilon} \frac{\{c\}_{j+1} - \{c\}_j}{\Delta t} + \epsilon [S]_{j+1} \{c\}_{j+1} + (1-\epsilon)[S]_j \{c\}_j + \epsilon \{f\}_{j+1} + (1-\epsilon)\{f\}_j = 0 \tag{5.9}$$

where j and $j+1$ denote the previous and current time levels, respectively; Δt is the time increment, and ϵ is a time weighing factor. The incorporation of the dispersion flux, Q_n^D, into matrix $[Q]$ and vector $\{f\}$ is discussed in Section 5.3.4. The coefficient matrix $[Q]_{j+\epsilon}$ is evaluated using weighted averages of the current and previous nodal values of θ and R. Equation (5.9) can be rewritten in the form:

$$[G]\{c\}_{j+1} = \{g\} \tag{5.10}$$

where

$$\begin{aligned} [G] &= \frac{1}{\Delta t}[Q]_{j+\epsilon} + \epsilon [S]_{j+1} \\ \{g\} &= \frac{1}{\Delta t}[Q]_{j+\epsilon}\{c\}_j - (1-\epsilon)[S]_j\{c\}_j - \epsilon\{f\}_{j+1} - (1-\epsilon)\{f\}_j \end{aligned} \tag{5.11}$$

Higher-order approximations for the time derivative in the transport equation were derived by *van Genuchten* [1976, 1978]. The higher-order effects may be incorporated into

the transport equation by introducing time-dependent dispersion corrections as follows

$$D_{ij}^{-} = D_{ij} - \frac{q_i q_j \Delta t}{6\theta^2 R}$$
$$D_{ij}^{+} = D_{ij} + \frac{q_i q_j \Delta t}{6\theta^2 R} \tag{5.12}$$

where the superscripts + and - indicate evaluation at the old and new time levels, respectively.

5.3. *Numerical Solution Strategy*

5.3.1. *Solution Process*

The solution process at each time step proceeds as follows. First, an iterative procedure is used to obtain the solution of the Richards' equation (2.1) (see Section 4.3.1). After achieving convergence, the solution of the transport equation (5.10) is implemented. This is done by first determining the nodal values of the fluid flux from nodal values of the pressure head by applying Darcy's law. Nodal values of the water content and the fluid flux at the previous time level are already known from the solution at the previous time step. Values for the water content and the fluid flux are subsequently used as input to the transport equation, leading to the system of linear algebraic equations given by (5.10). The structure of the final set of equations depends upon the value of the temporal weighing factor, ϵ. The explicit ($\epsilon = 0$) and fully implicit ($\epsilon = 1$) schemes require that the global matrix $[G]$ and the vector $\{g\}$ be evaluated at only one time level (the previous or current time level). All other schemes require evaluation at both time levels. Also, all schemes except for the explicit formulation ($\epsilon = 0$) lead to an asymmetric banded matrix $[G]$. The associated set of algebraic equations is solved using either a standard asymmetric matrix equation solver [e.g., *Neuman*, 1972], or the ORTHOMIN method [*Mendoza et al.*, 1991], depending upon the size of final matrix. By contrast, the explicit scheme leads to a diagonal matrix $[G]$ which is much easier to solve (but generally requires smaller time steps). Since transport

is assumed to be independent of changes in the fluid density, one may proceed directly to the next time level once the transport equation is solved for the current time level.

5.3.2. *Upstream Weighted Formulation*

Upstream weighing is provided as an option in SWMS_2D to minimize some of the problems with numerical oscillations when relatively steep concentration fronts are being simulated. For this purpose the second (flux) term of equation (5.3) is not weighted by regular linear basis functions ϕ_n, but instead using the nonlinear functions ϕ_n^u [*Yeh and Tripathi*, 1990]

$$\begin{aligned}
\phi_1^u &= L_1 - 3\alpha_3^w L_2 L_1 + 3\alpha_2^w L_3 L_1 \\
\phi_2^u &= L_2 - 3\alpha_1^w L_3 L_2 + 3\alpha_3^w L_1 L_2 \\
\phi_3^u &= L_3 - 3\alpha_2^w L_1 L_3 + 3\alpha_1^w L_2 L_3
\end{aligned} \tag{5.13}$$

where α_i^w is a weighing factor associated with the length of the element opposite to node i, and L_i are the local coordinates. The weighing factors are evaluated using the equation of *Christie et al.* [1976]:

$$\alpha_i^w = \coth\left(\frac{uL}{2D}\right) - \frac{2D}{uL} \tag{5.14}$$

where u, D and L are the flow velocity, dispersion coefficient and length associated with side i. The weighing functions ϕ^u ensure that relatively more weight is placed on the flow velocities of nodes located at the upstream side of an element. Evaluating the integrals in (5.3) shows that the following additional terms must be added to the entries of global matrix S_{nm} in equation (5.6):

$$S_{1j}^{e\prime} = S_{1j}^{e} - \frac{b_j}{40}[2q_{x1}(\alpha_2^w - \alpha_3^w) + q_{x2}(\alpha_2^w - 2\alpha_3^w) + q_{x3}(2\alpha_2^w - \alpha_3^w)] - \frac{c_j}{40}[2q_{z1}(\alpha_2^w - \alpha_3^w) + q_{z2}(\alpha_2^w - 2\alpha_3^w) + q_{z3}(2\alpha_2^w - \alpha_3^w)] \tag{5.15}$$

$$S_{2j}^{e\prime} = S_{2j}^{e} - \frac{b_j}{40}[q_{x1}(2\alpha_3^w - \alpha_1^w) + 2q_{x2}(\alpha_3^w - \alpha_1^w) + q_{x3}(\alpha_3^w - 2\alpha_1^w)] - \frac{c_j}{40}[q_{z1}(2\alpha_3^w - \alpha_1^w) + 2q_{z2}(\alpha_3^w - \alpha_1^w) + q_{z3}(\alpha_3^w - 2\alpha_1^w)] \tag{5.16}$$

and

$$S_{3j}^{e\prime} = S_{3j}^{e} - \frac{b_j}{40}[q_{x1}(\alpha_1^w - 2\alpha_2^w) + q_{x2}(2\alpha_1^w - \alpha_2^w) + 2q_{x3}(\alpha_1^w - \alpha_2^w)] - \frac{c_j}{40}[q_{z1}(\alpha_1^w - 2\alpha_2^w) + q_{z2}(2\alpha_1^w - \alpha_2^w) + 2q_{z3}(\alpha_1^w - \alpha_2^w)] \tag{5.17}$$

The weighing factors are applied only to those element sides that are inclined within 20 degrees from the flow direction.

5.3.3. *Implementation of First-Type Boundary Conditions*

Individual equations in the global matrix equation which correspond to nodes at which the concentration is prescribed are replaced by new equations:

$$\delta_{nm} c_m = c_{n0} \tag{5.18}$$

where c_{n0} is the prescribed value of the concentration at node n. This is done only when Gaussian elimination is used to solve the matrix equation. A similar procedure as for water flow (described in Section 4.3.4) is applied when the ORTHOMIN method is used. Because of asymmetry of the global matrix [G], no additional manipulations are needed in the

resulting system of equations as was the case for the water flow solution.

The total material flux, Q_n^T, through a boundary at node n consists of the dispersive flux, Q_n^D, and the convective flux, Q_n^A:

$$Q_n^T = Q_n^D + Q_n^A \tag{5.19}$$

The dispersive boundary nodal flux is not known explicitly but must be calculated from equation (5.4). Hence, the dispersion flux, Q_n^D, for node n can be calculated as

$$Q_n^D = -[\epsilon S_{nm}^{j+1} + (1-\epsilon)S_{nm}^{j}]c_m^j - \epsilon f_n^{j+1} - (1-\epsilon)f_n^j - Q_{nn}^{j+\epsilon}\frac{c_n^{j+1} - c_n^j}{\Delta t} \tag{5.20}$$

The convective flux is evaluated as

$$Q_n^A = Q_n c_n \tag{5.21}$$

where the fluid flux Q_n is known from the solution of the water flow equation.

5.3.4. *Implementation of Third-Type Boundary Conditions*

Equation (3.9) is rewritten as follows

$$\theta D_{ij}\frac{\partial c'}{\partial x_j}n_i = q_i n_i (c - c_0) \tag{5.22}$$

When substituted into the last term of (5.3), the boundary integral becomes

$$\sum_e \int_{\Gamma_N^e} \theta D_{ij}\frac{\partial c'}{\partial x_j}n_i \phi_n d\Gamma = Q_n c_n - Q_n c_{n0} \tag{5.23}$$

The first term on the right-hand side of (5.23) represents the convective flux. This term is incorporated into the coefficient matrix [S] of (5.4). The last term of (5.23) represents the total material flux, which is added to the known vector {f}.

At nodes where free outflow of water and its dissolved solutes takes place, the exit concentration c_0 is equal to the local (nodal) concentration c_n. In this case the dispersive

flux becomes zero and the total material flux through the boundary is evaluated as

$$Q_n^T = Q_n c_n \tag{5.24}$$

5.3.5. *Mass Balance Calculations*

The total amount of mass in the entire flow domain, or in a preselected subregion, is given by

$$M = \sum_e \int_{\Omega_e} \theta R c \, d\Omega = \sum_e \kappa A_e \frac{\theta_i R_i c_i + \theta_j R_j c_j + \theta_k R_k c_k}{3} \tag{5.25}$$

where $\theta_{i,j,k}$, $R_{i,j,k}$ and $c_{i,j,k}$ represent, respectively, water contents, retardation factors, and concentrations evaluated at the corner nodes of element e. The summation is taken over all elements within the specified region.

The cumulative amounts M^0 and M^1 of solute removed from the flow region by zero- and first-order reactions, respectively, are calculated as follows

$$M^0 = -\int_0^t \sum_e \int_{\Omega_e} (\gamma_w \theta + \gamma_s \rho) d\Omega \, dt \tag{5.26}$$

$$M^1 = -\int_0^t \sum_e \int_{\Omega_e} (\mu_w \theta + \mu_s \rho k) c \, d\Omega \, dt \tag{5.27}$$

whereas the cumulative amount M_r of solute taken up by plant roots is given by

$$M_r = \int_0^t \sum_{e_R} \int_{\Omega_e} S c_s \, d\Omega \, dt \tag{5.28}$$

where e_R represents the elements making up the root zone.

Finally, when all boundary material fluxes, decay reactions, and root uptake mass

fluxes have been computed, the following mass balance should hold for the flow domain as a whole:

$$M_t - M_0 = + \int_0^t \sum_{n_\Gamma} Q_n^T dt - M^0 - M^1 - M_r \tag{5.29}$$

where M_t and M_0 are the amounts of solute in the flow region at times t and zero, respectively, as calculated with (5.25), and n_Γ represents nodes located along the boundary of the flow domain or at internal sinks and/or sources. The difference between the left- and right-hand sides of (5.29) represents the absolute error, ϵ_a^c, in the solute mass balance. Similarly as for water flow, the accuracy of the numerical solution for solute transport is evaluated by using the relative error, ϵ_r^c [%], in the solute mass balance as follows

$$\epsilon_r^c = \frac{100 \mid \epsilon_a^c \mid}{\max\left(\sum_e |M_t^e - M_0^e|, \quad |M^0| + |M^1| + |M_r| + \int_0^t \sum_{n_\Gamma} |Q_n^T| dt \right)} \tag{5.30}$$

where M_0^e and M_t^e are the amounts of solute in element e at times 0 and t, respectively. Note again that SWMS_2D does not relate the absolute error to the total amount of mass in the flow region. Instead, the program uses as a reference the maximum value of (1) the absolute change in element concentrations as summed over all elements, and (2) the sum of the absolute values of all cumulative solute fluxes across the flow boundaries including those resulting from sources and sinks in the flow domain.

5.3.6. *Oscillatory Behavior*

Numerical solutions of the transport equation often exhibit oscillatory behavior and/or excessive numerical dispersion near relatively sharp concentration fronts. These problems can be especially serious for convection-dominated transport characterized by small dispersivities. One way to partially circumvent numerical oscillations is to use upstream weighing as discussed in Section 5.3.2. Undesired oscillations can often be

prevented also by selecting an appropriate combination of space and time discretizations. Two dimensionless numbers may be used to characterize the space and time discretizations. One of these is the grid Peclet number, Pe_i^e, which defines the predominant type of the solute transport (notably the ratio of the convective and dispersive transport terms) in relation to coarseness of the finite element grid:

$$Pe_i^e = \frac{q_i \Delta x_i}{\theta D_{ii}} \tag{5.31}$$

where Δx_i is the characteristic length of a finite element. The Peclet number increases when the convective part of the transport equation dominates the dispersive part, i.e., when a relatively steep concentration front is present. To achieve acceptable numerical results, the spatial discretization must be kept relatively fine to maintain a low Peclet number. Numerical oscillation can be virtually eliminated when the local Peclet numbers do not exceed about 5. However, acceptably small oscillations may be obtained with local Peclet numbers as high as 10 [*Huyakorn and Pinder*, 1983].

A second dimensionless number which characterizes the relative extent of numerical oscillations is the Courant number, Cr_i^e. The Courant number is associated with the time discretization as follows

$$Cr_i^e = \frac{q_i \Delta t}{\theta R \Delta x_i} \tag{5.32}$$

Three stabilizing options are used in SWMS_2D to avoid oscillations in the numerical solution of the solute transport equation [*Šimůnek and Suarez*, 1993]. One option is upstream weighing (see Section 5.3.2), which effectively eliminates undesired oscillations at relatively high Peclet numbers. A second option for minimizing or eliminating numerical oscillations uses the following criterion developed by *Perrochet and Berod* [1993]

$$Pe \cdot Cr \leq \omega_s = 2 \tag{5.33}$$

where ω_s is the performance index [-]. This criterion indicates that convection-dominated transport problems having large Pe numbers can be safely simulated provided Cr is reduced

according to (5.33) [*Perrochet and Berod*, 1993]. When small oscillations in the solution are tolerated, ω_s can be increased to about 5 or 10.

A third stabilizing option implemented in SWMS_2D also utilizes criterion (5.33). However, instead of decreasing *Cr* to satisfy equation (5.33), this option introduces artificial dispersion to decrease the Peclet number. The amount of additional longitudinal dispersion, $\bar{D}_L$ [L], is given by [*Perrochet and Berod*, 1993]

$$\bar{D}_L = \frac{|q|\Delta t}{\theta R \omega_s} - D_L - \frac{\theta D_w \tau}{|q|} \tag{5.34}$$

The maximum permitted time step is calculated using all three options, as well as the additional requirement that the Courant number must remain less than or equal to 1. The time step calculated in this way is subsequently used as one of the time discretization rules (rule No. B) discussed in section 4.3.3.

6. PROBLEM DEFINITION

6.1. *Construction of Finite Element Mesh*

The finite element mesh is constructed by dividing the flow region into quadrilateral and/or triangular elements whose shapes are defined by the coordinates of the nodes that form the element corners. The program automatically subdivides the quadrilaterals into triangles which are then treated as subelements.

Transverse lines [*Neuman*, 1974] formed by element boundaries should transect the mesh along the general direction of its shortest dimension. These transverse lines must be continuous and non-intersecting, but need not be straight. The nodes are numbered sequentially from 1 to *NumNP* (total number of nodes) by proceeding along each transverse line in the same direction. Elements are numbered in a similar manner. The maximum number of nodes on any transverse line, *IJ*, is used to determine the effective size of the finite element matrix (i.e., its bandwidth). To minimize memory and time requirements, *IJ* should be kept as small as possible. The above rules for defining the finite element mesh apply only when Gaussian elimination is used to solve the matrix equations. Iterative methods (such as the conjugate gradient and ORTHOMIN methods) are not so restrictive since only non-zero entries in the coefficient matrix are stored in memory, and since the computational efficiency is less dependent upon the bandwidth of the matrix as compared to direct equation solvers.

The finite element dimensions must be adjusted to a particular problem. They should be made relatively small in directions where large hydraulic gradients are expected. Regions with sharp gradients are usually located in the vicinity of the internal sources or sinks, or close to the soil surface where highly variable meteorological factors can cause fast changes in pressure head. Hence, we recommend to normally use relatively small elements at and near the soil surface. The size of elements can gradually increase with depth to reflect the generally much slower changes in pressure heads at deeper depths. The element dimensions should also depend upon the soil hydraulic properties. For example, coarse-

textured soils having relatively values of high n-values and small α-values (see Eqs. (2.11) and (2.18)) generally require a finer discretization than fine-textured soils. We also recommend using elements having approximately equal sizes to decrease numerical errors. For axisymmetric three-dimensional flow systems, the vertical axis must coincide with, or be to the left of, the left boundary of the mesh. No special restrictions are necessary to facilitate the soil root zone.

6.2. *Coding of Soil Types and Subregions*

Soil Types - An integer code beginning with 1 and ending with *NMat* (the total number of soil materials) is assigned to each soil type in the flow region. The appropriate material code is subsequently assigned to each nodal point n of the finite element mesh.

Interior material interfaces do not coincide with element boundaries. When different material numbers are assigned to the corner nodes of a certain element, material properties of this element will be averaged automatically by the finite element algorithm. This procedure will somewhat smooth soil interfaces.

A set of soil hydraulic parameters and solute transport characteristics must be specified for each soil material. Also, the user must define for each element the principal components of the conductivity anisotropy tensor, as well as the angle between the local and global coordinate systems.

As explained in Section 2.3, one additional way of changing the unsaturated soil hydraulic properties in the flow domain is to introduce scaling factors associated with the water content, the pressure head and the hydraulic conductivity. The scaling factors are assigned to each nodal point n in the flow region.

Subregions - Water and solute mass balances are computed separately for each specified subregion. The subregions may or may not coincide with the material regions. Subregions are characterized by an integer code which runs from 1 to *NLay* (the total number of subregions). A subregion code is assigned to each element in the flow domain.

6.3. *Coding of Boundary Conditions*

Flow boundary conditions were programmed in a fairly similar way as done in the UNSAT1 and UNSAT2 models of *Neuman* [1972] and *Neuman et al.* [1974]. Unit vertical hydraulic gradient boundary conditions simulating free drainage are implemented in this version 1.21 of SWMS_2D, in addition to the boundary conditions used in version 1.1. A boundary code, *Kode*(n), must be assigned to each node, n. If node n is to have a prescribed pressure head during a time step (Dirichlet boundary condition), *Kode*(n) must be set positive during that time step. If the volumetric flux of water entering or leaving the system at node n is prescribed during a time step (Neumann boundary condition), *Kode*(n) must be negative or zero.

Constant Boundary Conditions - The values of constant boundary conditions for a particular node, n, are given by the initial values of the pressure head, $h(n)$, in case of Dirichlet boundary conditions, or by the initial values of the recharge/discharge flux, $Q(n)$, in case of Neumann boundary conditions. Table 6.1 summarizes the use of the variables *Kode*(n), $Q(n)$ and $h(n)$ for various types of nodes.

Table 6.1. Initial settings of *Kode*(n), $Q(n)$, and $h(n)$ for constant boundary conditions.

Node Type	*Kode*(n)	$Q(n)$	$h(n)$
Internal; not sink/source	0	0.0	Initial Value
Internal; sink/source (Dirichlet condition)	1	0.0	Prescribed
Internal; sink/source (Neumann condition)	-1	Prescribed	Initial Value
Impermeable Boundary	0	0.0	Initial Value
Specified Head Boundary	1†	0.0	Prescribed
Specified Flux Boundary	-1‡	Prescribed	Initial Value

† 5 and/or 6 may also be used
‡ -5 and/or -6 may also be used

Variable Boundary Conditions - Three types of variable boundary conditions can be imposed:

1. Atmospheric boundary conditions for which $Kode(n) = \pm 4$,
2. Variable pressure head boundary conditions for which $Kode(n) = +3$, and
3. Variable flux boundary conditions for which $Kode(n) = -3$.

These conditions can be specified along any part of the boundary. It is not possible to specify more than one time-dependent boundary condition for each type. Initial settings of the variables $Kode(n)$, $Q(n)$ and $h(n)$ for the time-dependent boundary conditions are given in Table 6.2.

Table 6.2. Initial settings of $Kode(n)$, $Q(n)$, and $h(n)$ for variable boundary conditions.

Node Type	$Kode(n)$	$Q(n)$	$h(n)$
Atmospheric Boundary	–4	0.0	Initial Value
Variable Head Boundary	+3	0.0	Initial Value
Variable Flux Boundary	–3	0.0	Initial Value

Atmospheric boundary conditions are implemented when $Kode(n) = \pm 4$, in which case time-dependent input data for the precipitation, *Prec*, and evaporation, *rSoil*, rates must be specified in the input file ATMOSPH.IN. The potential fluid flux across the soil surface is determined by *rAtm* = *rSoil-Prec*. The actual surface flux is calculated internally by the program. Two limiting values of the surface pressure head must be provided: *hCritS* which specifies the maximum allowed pressure head at the soil surface (usually 0.0), and *hCritA* which specifies the minimum allowed surface pressure head (defined from equilibrium conditions between soil water and atmospheric vapor). The program automatically switches the value of $Kode(n)$ from -4 to +4 if one of these two limiting points is reached. Table 6.3 summarizes the use of the variables *rAtm*, *hCritS* and *hCritA* during program execution. $Width(n)$ in this table denotes the length of the boundary segment associated with node n.

Table 6.3. Definition of the variables *Kode*(*n*), *Q*(*n*) and *h*(*n*) when an atmospheric boundary condition is applied.

Kode(*n*)	*Q*(*n*)	*h*(*n*)	Event
–4	*-Width(n)*rAtm*	Unknown	*rAtm=rSoil-Prec*
+4	Unknown	*hCritA*	Evaporation capacity is exceeded
+4	Unknown	*hCritS*	Infiltration capacity is exceeded

Variable head and flux boundary conditions along a certain part of the boundary are implemented when *Kode*(*n*)=+3 and -3, respectively. In that case, the input file ATMOSPH.IN must contain the prescribed time-dependent values of the pressure head, *ht*, or the flux, *rt*, imposed along the boundary. The values of *ht* or *rt* are assigned to particular nodes at specified times according to rules given in Table 6.4.

Table 6.4. Definition of the variables *Kode*(*n*), *Q*(*n*) and *h*(*n*) when variable head or flux boundary conditions are applied.

Node Type	*Kode*(*n*)	*Q*(*n*)	*h*(*n*)
Variable Head Boundary	+3	Unknown	*ht*
Variable Flux Boundary	–3	*-Width(n)*rt*	Unknown

Water Uptake by Plant Roots - The program calculates the rate at which plants extract water from the soil root zone by evaluating the term *D* (equation (4.9)) in the finite element formulation. The code requires that *Kode*(*n*) be set equal to 0 or negative for all nodes in the root zone. Values of the potential transpiration rate, *rRoot*, must be specified at preselected times in the input file ATMOSPH.IN. Actual transpiration rates are calculated internally by the program as discussed in Section 2.2. The root uptake parameters are taken

from input file SELECTOR.IN. Values of the function *Beta*(n), which describes the potential water uptake distribution over the root zone (equation (2.5)), must be specified for each node in the flow domain (see the description of input Block H in Table 8.8 of Section 8). All parts of the flow region where *Beta*(n)>0 are treated as the soil root zone.

Deep Drainage from the Soil Profile - Vertical drainage, $q(h)$, across the lower boundary of the soil profile is sometimes approximated by a flux which depends on the position of groundwater level (e.g., *Hopmans and Stricker*, 1989). If available, such a relationship can be implemented in the form of a variable flux boundary condition for which *Kode*(n)=-3. This boundary condition is implemented in SWMS_2D by setting the logical variable *qGWLF* in the input file ATMOSPH.IN equal to ".true." The discharge rate $Q(n)$ assigned to node n is determined in the program as $Q(n)=-Width(n)*q(h)$ where h is the local value of the pressure head, and $q(h)$ is given by

$$q(h) = -A_{qh}\exp(B_{qh}|h - GWL0L|) \tag{6.1}$$

where A_{qh} and B_{qh} are empirical parameters which must be specified in the input file ATMOSPH.IN, together with *GWL0L* which represents the reference position of the groundwater level (usually set equal to the z-coordinate of the soil surface).

Free Drainage - Unit vertical hydraulic gradient boundary conditions can be implemented in the form of a variable flux boundary condition for which *Kode*(n)=-3. This boundary condition is implemented in SWMS_2D by setting the logical variable *FreeD* in the input file SELECTOR.IN equal to ".true.". The program determines the discharge rate $Q(n)$ assigned to node n as $Q(n)=-Width(n)*K(h)$, where h is the local value of the pressure head, and $K(h)$ is the hydraulic conductivity corresponding to this pressure head.

Seepage Faces - The initial settings of the variables *Kode*(n), $Q(n)$ and $h(n)$ for nodes along a seepage face are summarized in Table 6.5. All potential seepage faces must be identified before starting the numerical simulation. This is done by providing a list of nodes

along each potential seepage face (see input Block E as defined in Table 8.5 of Section 8).

Table 6.5. Initial setting of *Kode*(n), $Q(n)$, and $h(n)$ for seepage faces.

Node Type	*Kode*(n)	$Q(n)$	$h(n)$
Seepage Face (initially saturated)	+2	0.0	0.0
Seepage Face (initially unsaturated)	−2	0.0	Initial Value

Drains - Table 6.6 summarizes the initial settings of the variables *Kode*(n), $Q(n)$ and $h(n)$ for nodes representing drains. All drains must be identified before starting the numerical simulation. This is done by providing a list of nodes representing drains, together with a list of elements around each drain whose hydraulic conductivities are to be adjusted according to discussion in Section 4.3.7 (see also input Block F as defined in Table 8.6 of Section 8).

Table 6.6. Initial setting of *Kode*(n), $Q(n)$, and $h(n)$ for drains.

Node Type	*Kode*(n)	$Q(n)$	$h(n)$
Drain (initially saturated)	+5	0.0	0.0
Drain (initially unsaturated)	−5	0.0	Initial Value

Solute Transport Boundary Conditions. The original version 1.1. of SWMS_2D [*Šimůnek et al.*, 1992] assumed a strict relationship between the boundary conditions for water flow and solute transport. A first-type boundary condition for water flow forced the boundary condition for solute transport also to be of the first-type. Similarly, a second-type boundary condition for water flow induced a second- or third-type boundary condition for

solute transport depending upon the direction of the water flux. These strict relationships between the boundary conditions for water flow and solute transport have been abandoned in version 1.21. Selection of the type of boundary condition for the solute transport is now much more independent of the boundary condition implemented for water flow. The type of boundary condition to be invoked for solute transport is specified by the input variable *KodCB*. A positive sign of this variable means that a first-type boundary condition will be used. When *KodCB* is negative, SWMS_2D selects a third-type boundary condition when the calculated water flux is directed into the region, or a second-type boundary condition when the water flux is zero or directed out of the region. One exception to these rules occurs for atmospheric boundary conditions when $Kode(n) = \pm 4$ and $Q(n) < 0$. SWMS_2D assumes that solutes cannot leave the flow region across atmospheric boundaries. The solute flux in this situation becomes zero, i.e., $c_0 = 0$ in equation (5.22). Cauchy and Neumann boundary conditions are automatically applied to internal sinks/sources depending upon the direction of water flow. The dependence (or independence) of the solute boundary conditions on time or the system is then still defined through the variable $Kode(n)$ as discussed above.

Although SWMS_2D can implement first-type boundary conditions, we recommend users to invoke third-type conditions where possible. This is because third-type conditions, in general, are physically more realistic and preserve solute mass in the simulated system (e.g., *van Genuchten and Parker* [1984]; *Leij et al.* [1991]).

6.4. *Program Memory Requirements*

One single parameter statement is used at the beginning of the code (see the second statement of the source code listed in Section 10.3) to define the problem dimensions. All major arrays in the program are adjusted automatically according to these dimensions. This feature makes it possible to change the dimensions of the problem to be simulated without having to recompile all program subroutines. Different problems can be investigated by changing the dimensions in the parameter statement at the beginning of the main program,

and subsequently linking all previously compiled subroutines with the main program when creating an executable file. Table 6.7 lists the array dimensions which must be defined in the parameter statement.

Table 6.7. List of array dimensions in SWMS_2D.

Dimension	Description
NumNPD	Maximum number of nodes in finite element mesh
NumElD	Maximum number of elements in finite element mesh
MBandD	Maximum dimension of the bandwidth of matrix A when Gaussian elimination is used. Maximum number of nodes adjacent to a particular node, including itself, when iterative matrix solvers are used.
NumBPD	Maximum number of boundary nodes for which $Kode(n) \neq 0$
NSeepD	Maximum number of seepage faces
NumSPD	Maximum number of nodes along a seepage face
NDrD	Maximum number of drains
NElDrD	Maximum number of elements surrounding a drain
NMatD	Maximum number of materials
NTabD	Maximum number of items in the table of hydraulic properties generated by the program for each soil material
NumKD	Maximum number of available code number values (equals 6 in present version)
NObsD	Maximum number of observation nodes for which values of the pressure head, the water content, and concentration are printed at each time level
MNorth	Maximum number of orthogonalizations performed when iterative solvers are used

6.5. *Matrix Equation Solvers*

Discretization of the governing partial differential equations for water flow (2.1) and solute transport (3.4) leads to the system of linear equations

$$[A]\{x\} = \{b\} \tag{6.2}$$

in which matrix $[A]$ is symmetric for water flow and asymmetric for solute transport.

The original version of SWMS_2D [*Šimůnek et al.*, 1992] used Gaussian elimination to solve both systems of linear algebraic equations. The invoked solvers took advantage of the banded nature of the coefficient matrices and, in the case of water flow, of the symmetric properties of the matrix. Such direct solution methods have several disadvantages as compared to iterative methods. Direct methods require a fixed number of operations (depending upon the size of the matrix) which increases approximately by the square of the number of nodes [*Mendoza et al.*, 1991]. Iterative methods, on the other hand, require a variable number of repeated steps which increase at a much smaller rate (about 1.5) with the size of a problem [*Mendoza et al.*, 1991]. A similar reduction also holds for the memory requirement since iterative methods do not require the storage of non-zero matrix elements. Memory requirements, therefore, increase at a much smaller rate with the size of the problem when iterative solvers are used [*Mendoza et al.*, 1991]. Round-off errors also represent less of a problem for iterative methods as compared to direct methods. This is because round-off errors in iterative methods are self-correcting [*Letniowski*, 1989]. Finally, for time-dependent problems, a reasonable approximation of the solution (i.e., the solution at the previous time step) exists for iterative methods, but not for direct methods [*Letniowski*, 1989]. In general, direct methods are more appropriate for relatively small problems, while iterative methods are more suitable for larger problems.

Many iterative methods have been used in the past for handling large sparse matrix equations. These methods include Jacobi, Gauss-Seidel, alternating direction implicit (ADI), block successive over-relaxation (BSSOR), successive line over-relaxation (SLOR), and strongly implicit procedures (SIP), among others [*Letniowski*, 1989]. More powerful preconditioned accelerated iterative methods, such as the preconditioned conjugate gradient method (PCG) [*Behie and Vinsome*, 1982], were introduced more recently. *Sudicky and Huyakorn* [1991] gave three advantages of the PCG procedure as compared to other iterative methods: PCG can be readily applied to finite element methods with irregular

grids, the method does not require iterative parameters, and PCG usually outperforms its iterative counterparts for situations involving relatively stiff matrix conditions.

The current version 1.21 of SWMS_2D implements both direct and iterative methods for solving the system of linear algebraic equations given by (6.2). Depending upon the size of matrix [A], we use either direct Gaussian elimination or the preconditioned conjugate gradient method [*Mendoza et al.,* 1991] for water flow and the ORTHOMIN (preconditioned conjugate gradient squared) procedure [*Mendoza et al.,* 1991] for solute transport. Gaussian elimination is used if either the bandwith of matrix [A] is smaller than 20, or the total number of nodes is smaller than 500. The iterative methods used in SWMS_2D were adopted from the ORTHOFEM software package of *Mendoza et al.* [1991].

The preconditioned conjugate gradient and ORTHOMIN methods consist of two essential parts: initial preconditioning, and iterative solution with either conjugate gradient or ORTHOMIN acceleration [*Mendoza et al.,* 1991]. Incomplete lower-upper (ILU) preconditioning is used in ORTHOFEM when matrix [A] is factorized into lower and upper triangular matrices by partial Gaussian elimination. The preconditioned matrix is subsequently repeatedly inverted using updated solution estimates to provide a new approximation of the solution. The orthogonalization-minimization acceleration technique is used to update the solution estimate. This technique insures that the search direction for each new solution is orthogonal to the previous approximate solution, and that either the norm of the residuals (for conjugate gradient acceleration [*Meijerink and van der Vorst,* 1981]) or the sum of squares of the residuals (for ORTHOMIN [*Behie and Vinsome,* 1982]) is minimized. More details about the two methods is given in the user's guide of ORTHOFEM [*Mendoza et al.,* 1991] or in *Letniowski* [1989]. *Letniowski* [1989] also gives a comprehensive review of accelerated iterative methods, as well as of different preconditioning techniques.

7. EXAMPLE PROBLEMS

Four example problems are presented in this section. Examples 1 and 2 provide comparisons of the water flow part of SWMS_2D code with results from both the UNSAT2 code of *Neuman* [1974] and the SWATRE code of *Belmans et al.* [1983]. Example 3 serves to verify the accuracy of the solute transport part of SWMS_2D by comparing numerical results against those obtained with a two-dimensional analytical solution during steady-state groundwater flow. Example 4 shows numerical results for a field infiltration experiment involving a two-layered axisymmetric three-dimensional flow domain. The input and output files of the examples are listed at the end of Sections 8 and 9, respectively.

7.1. *Example 1 - Column Infiltration Test*

This example simulates a one-dimensional laboratory infiltration experiment discussed by *Skaggs et al.* [1970]. The example was used later by *Davis and Neuman* [1983] as a test problem for the UNSAT2 code. Hence, the example provides a means of comparing results obtained with the SWMS_2D and UNSAT2 codes.

Figure 7.1 gives a graphical representation of the soil column and the finite element mesh used for the numerical simulations. The soil water retention and relative hydraulic conductivity functions of the sandy soil are presented in Figure 7.2. The soil was assumed to be homogenous and isotropic with a saturated hydraulic conductivity of 0.0433 cm/min. The initial pressure head of the soil was taken to be -150 cm. The column was subjected to ponded infiltration (a Dirichlet boundary condition) at the soil surface, resulting in one-dimensional vertical water flow. The open bottom boundary of the soil column was simulated by implementing a no-flow boundary condition during unsaturated flow ($h<0$), and a seepage face with $h=0$ when the bottom boundary becomes saturated (this last condition was not reached during the simulation). The impervious sides of the column were simulated by imposing no-flow boundary conditions.

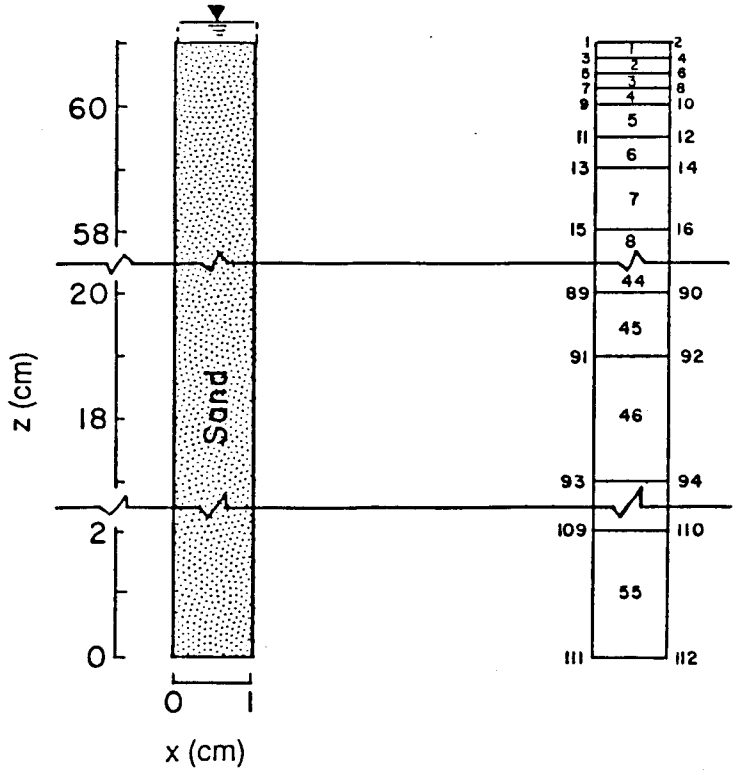

Fig. 7.1. Flow system and finite element mesh for example 1.

The simulation was carried out for 90 min, which corresponds to the total time duration of the experiment. Figure 7.3 shows the calculated instantaneous (q_0) and cumulative (I_0) infiltration rates simulated with SWMS_2D. The calculated results agree closely with those obtained by *Davis and Neuman* [1983] using their UNSAT2 code.

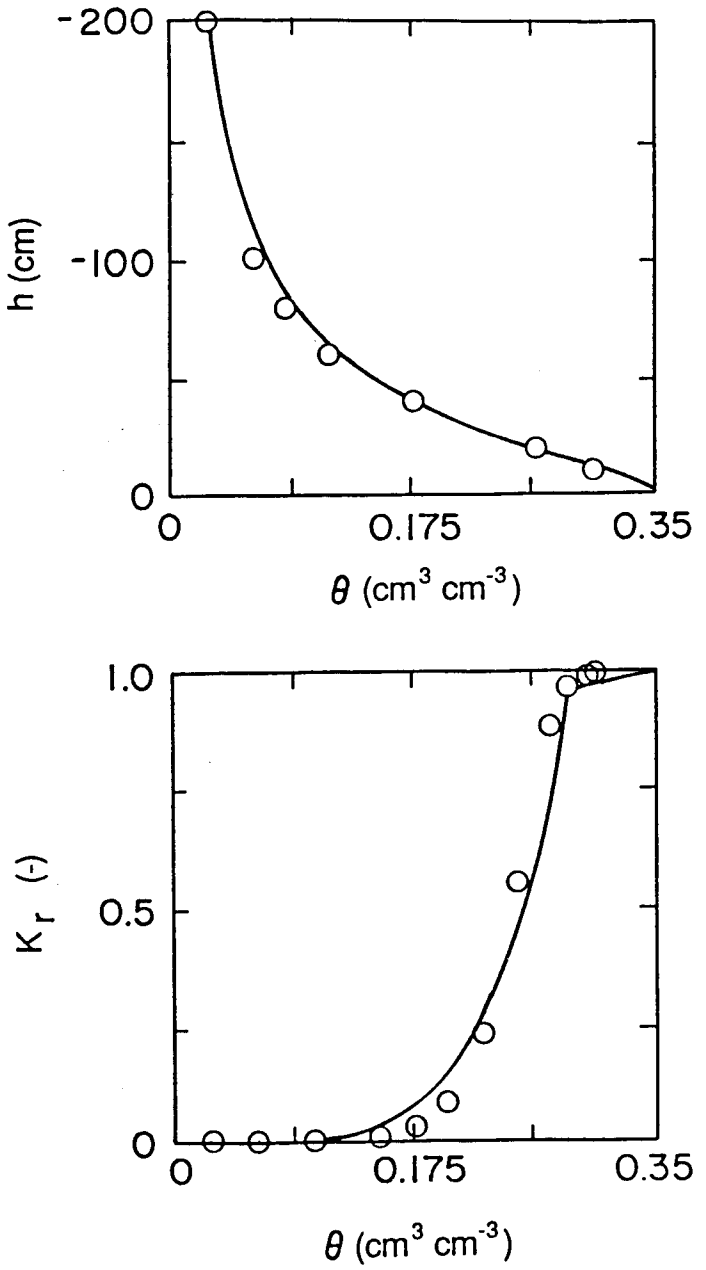

Fig. 7.2. Retention and relative hydraulic conductivity functions for example 1. The open circles are UNSAT2 input data [*Davis and Neuman*, 1983].

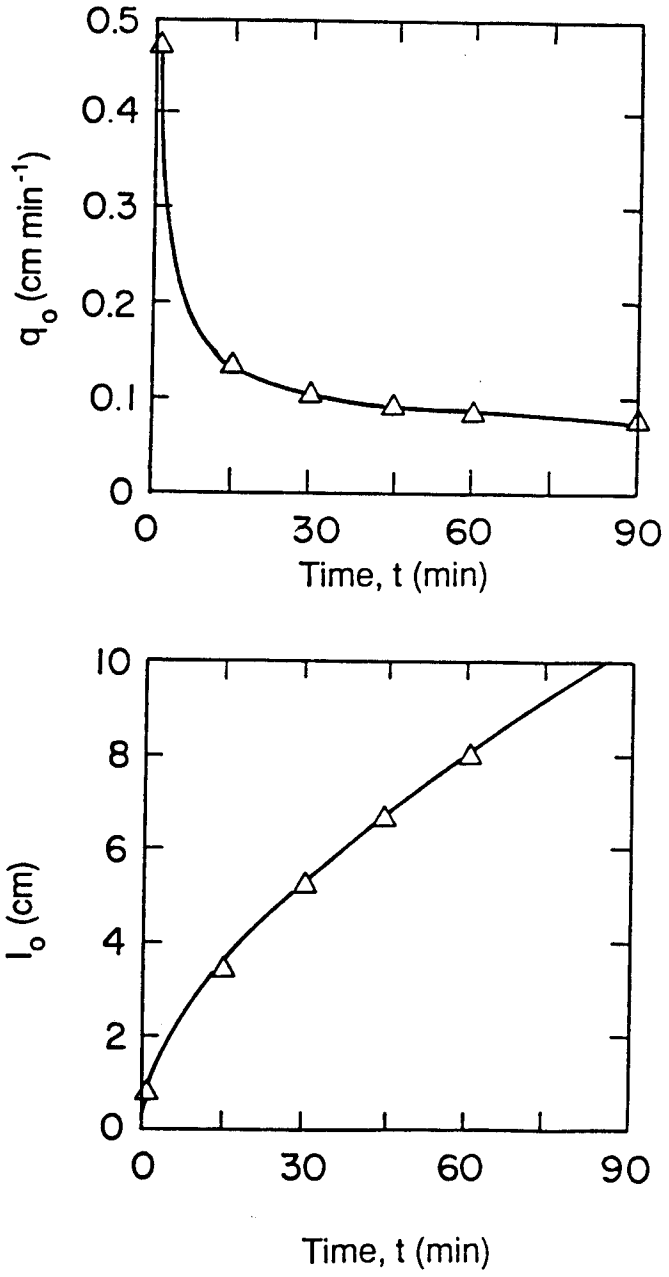

Fig. 7.3. Instantaneous, q_0, and cumulative, I_0, infiltration rates simulated with the SWMS_2D (solid lines) and UNSAT2 (triangles) codes for example 1.

7.2. *Example 2 - Water Flow in a Field Soil Profile Under Grass*

This example considers one-dimensional water flow in a field profile of the Hupselse Beek watershed in the Netherlands. Atmospheric data and observed ground water levels provided the required boundary conditions for the numerical model. Calculations were performed for the period of April 1 to September 30 of the relatively dry year 1982. Simulation results obtained with SWMS_2D will be compared with those generated with the SWATRE computer program [*Feddes et al.*, 1978, *Belmans et al.*, 1983].

The soil profile (Fig. 7.4) consisted of two layers: a 40-cm thick A-horizon, and a B/C-horizon which extended to a depth of about 300 cm. The depth of the root zone was 30 cm. The mean scaled hydraulic functions of the two soil layers in the Hupselse Beek area [*Císlerová*, 1987; *Hopmans and Stricker*, 1989] are presented in Figure 7.5.

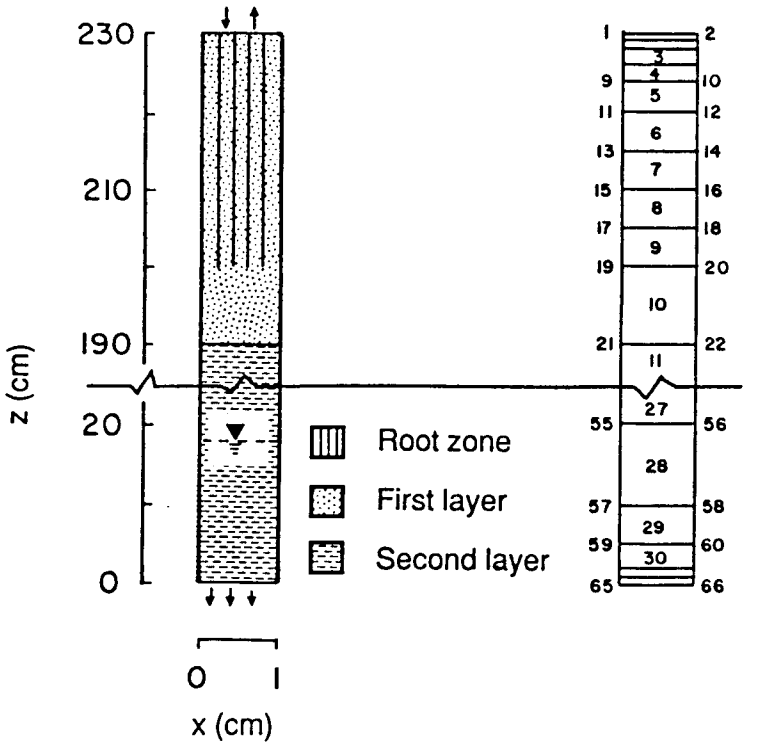

Fig. 7.4. Flow system and finite element mesh for example 2.

The soil surface boundary conditions involved actual precipitation and potential transpiration rates for a grass cover. The surface fluxes were incorporated by using average daily rates distributed uniformly over each day. The bottom boundary condition consisted of a prescribed drainage flux - groundwater level relationship, $q(h)$, as given by equation (6.1). The groundwater level was initially set at 55 cm below the soil surface. The initial moisture profile was taken to be in equilibrium with the initial ground water level.

Figure 7.6 presents input values of the precipitation and potential transpiration rates. Calculated cumulative transpiration and cumulative drainage amounts as obtained with the SWMS_2D and SWATRE codes are shown in Figure 7.7. The pressure head at the soil surface and the arithmetic mean pressure head of the root zone during the simulated season are presented in Figure 7.8. Finally, Figure 7.9 shows variations in the calculated groundwater level with time.

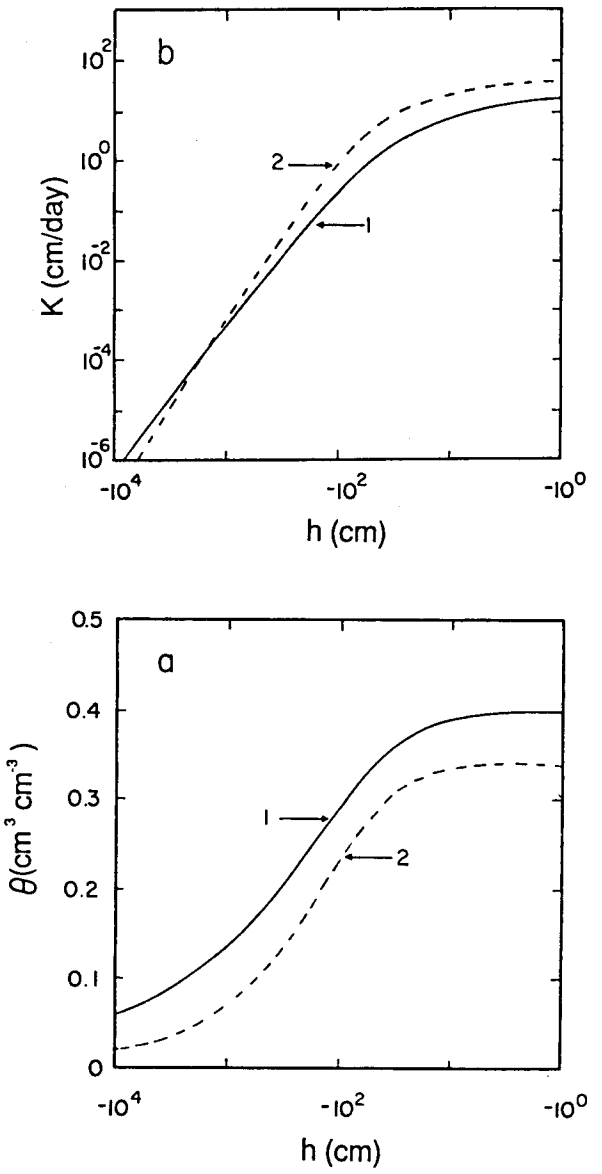

Fig. 7.5. Unsaturated hydraulic properties of the first and second soil layers for example 2.

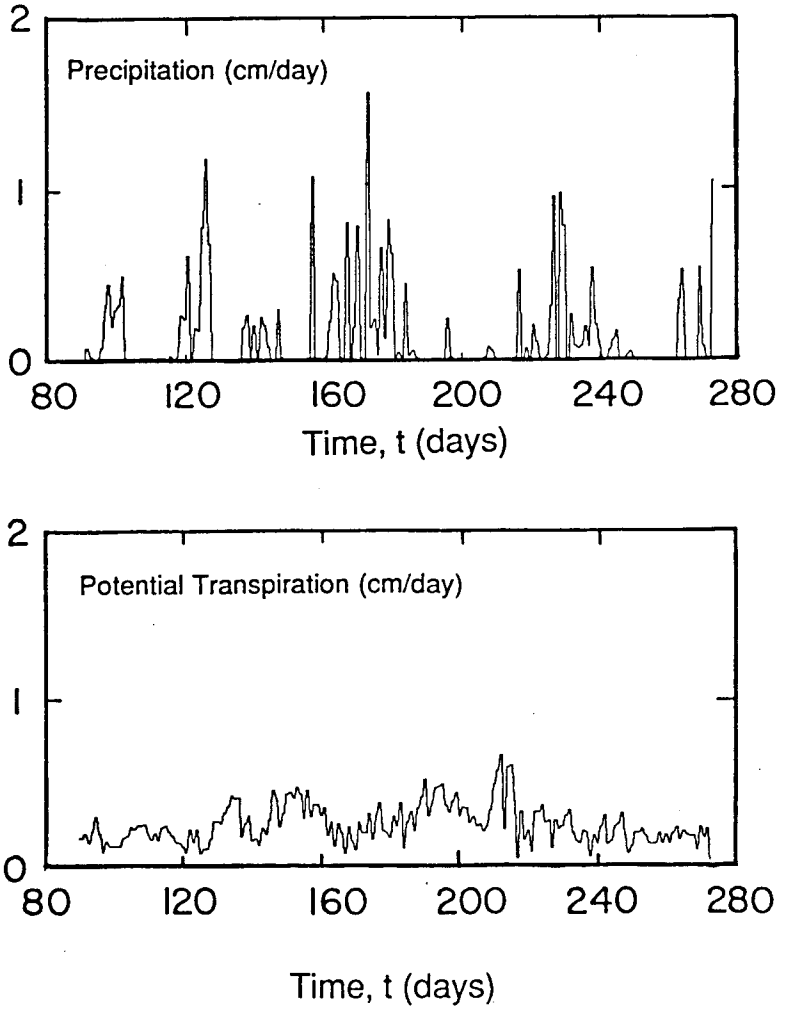

Fig. 7.6. Precipitation and potential transpiration rates for example 2.

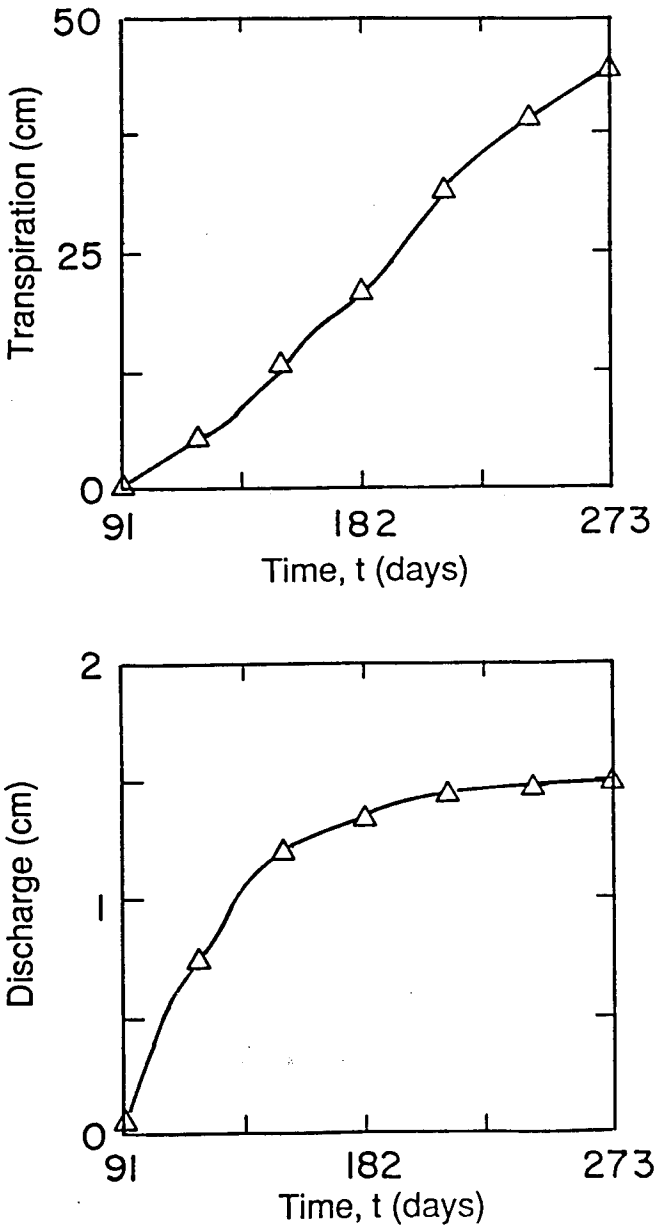

Fig. 7.7. Cumulative values for the actual transpiration and bottom discharge rates for example 2 as simulated with SWMS_2D (solid line) and SWATRE (triangles).

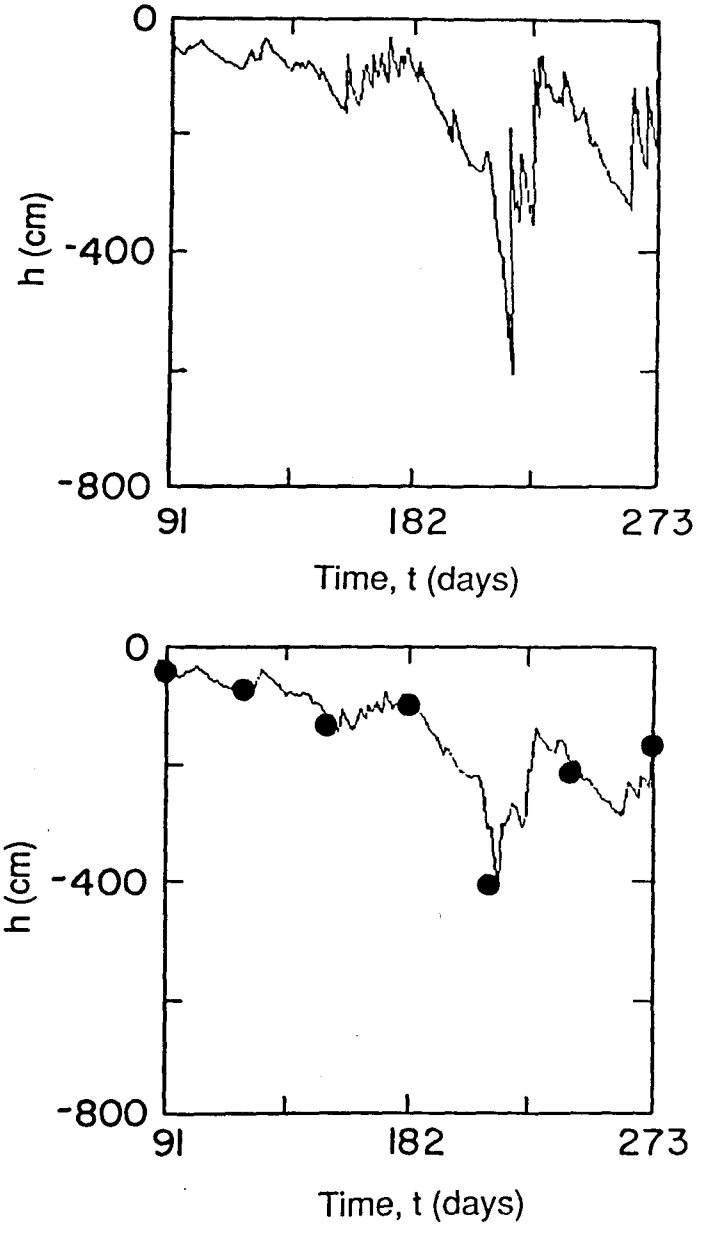

Fig. 7.8. Pressure head at the soil surface and mean pressure head of the root zone for example 2 as simulated with SWMS_2D (solid lines) and SWATRE (solid circles).

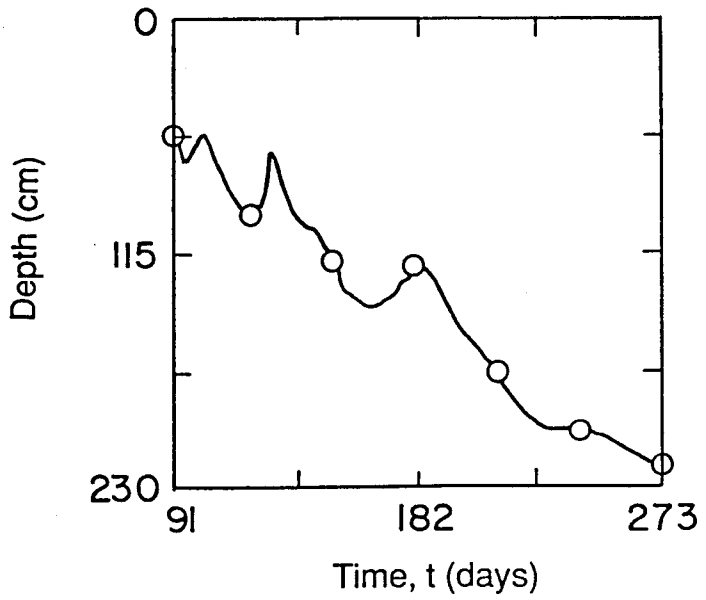

Fig. 7.9. Location of the groundwater table versus time for example 2 as simulated with the SWMS_2D (solid line) and SWATRE (open circles) computer programs.

7.3. *Example 3 - Two-Dimensional Solute Transport*

This example was used to verify the mathematical accuracy of the solute transport part of SWMS_2D. *Cleary and Ungs* [1978] published several analytical solutions for two-dimensional dispersion problems. One of these solutions holds for solute transport in a homogeneous, isotropic porous medium during steady-state unidirectional groundwater flow. The solute transport equation (3.4) for this situation reduces to

$$D_T\frac{\partial^2 c}{\partial x^2} + D_L\frac{\partial^2 c}{\partial z^2} - v\frac{\partial c}{\partial z} - \lambda R c = R\frac{\partial c}{\partial t} \tag{7.1}$$

where λ is a first-order degradation constant, D_L and D_T are the longitudinal and transverse dispersion coefficients, respectively; v is the average pore water velocity (q_z/θ) in the flow direction, and z and x are the spatial coordinates parallel and perpendicular to the direction

of flow. The initially solute-free medium is subjected to a solute source, c_0, of unit concentration. The source covers a length $2a$ along the inlet boundary at $z=0$, and is located symmetrically about the coordinate $x=0$. The transport region of interest is the half-plane ($z \geq 0$; $-\infty \leq x \leq \infty$). The boundary conditions may be written as:

$$\begin{aligned} c(x,0,t) &= c_0 \qquad -a \leq x \leq a \\ c(x,0,t) &= 0 \qquad \textit{other values of } x \\ \lim_{z \to \infty} \frac{\partial c}{\partial z} &= 0 \\ \lim_{x \to \pm\infty} \frac{\partial c}{\partial x} &= 0 \end{aligned} \tag{7.2}$$

The analytical solution of the above transport problem is [see also *Javandel et al.*, 1984]

$$c(x,z,t) = \frac{c_0 z}{4(\pi D_L)^{1/2}} \exp\left(\frac{vz}{2D_L}\right) \int_0^{t/R} \exp\left[-\left(\lambda R + \frac{v^2}{4D_L}\right)\tau - \frac{z^2}{4D_L \tau}\right] \tau^{-3/2} \cdot \left[erf\left(\frac{a-x}{2(D_T \tau)^{1/2}}\right) + erf\left(\frac{a+x}{2(D_T \tau)^{1/2}}\right)\right] d\tau \tag{7.3}$$

The input transport parameters for two simulations are listed in Table 7.1. The width of the source was assumed to be 100 m. Because of symmetry, calculations were carried out only for the quarter plane where $x \geq 0$ and $z \geq 0$.

Table 7.1. Input parameters for example 3.

Parameter	Example 3a	Example 3b
v [m/day]	0.1	1.0
D_T [m^2/day]	1.0	0.5
D_L [m^2/day]	1.0	1.0
λ [day^{-1}]	0.0	0.01
R [-]	1.0	3.0
c_0 [-]	1.0	1.0

Figure 7.10 shows the calculated concentration front (taken at a concentration of 0.1) at selected times for the first set of transport parameters in Table 7.1. Note the close agreement between the analytical and numerical results. Excellent agreement is also obtained for the calculated concentration distributions after 365 days at the end of the simulation (Fig. 7.11). Figures 7.12 and 7.13 show similar results for the second set of transport parameters listed in Table 7.1.

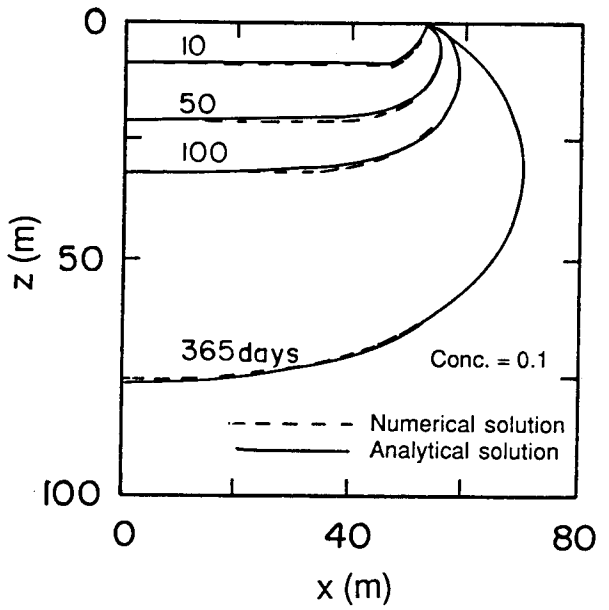

Fig. 7.10. Advancement of the concentration front (c=0.1) for example 3a as calculated with SWMS_2D (dotted lines) and the analytical solution (solid lines).

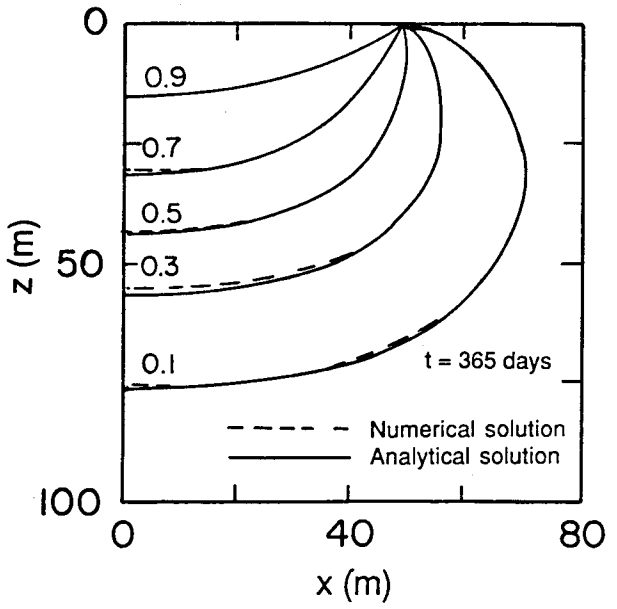

Fig. 7.11. Concentration profile at the end of the simulation (t=365 days) for example 3a as calculated with SWMS_2D (dotted lines) and the analytical solution (solid lines).

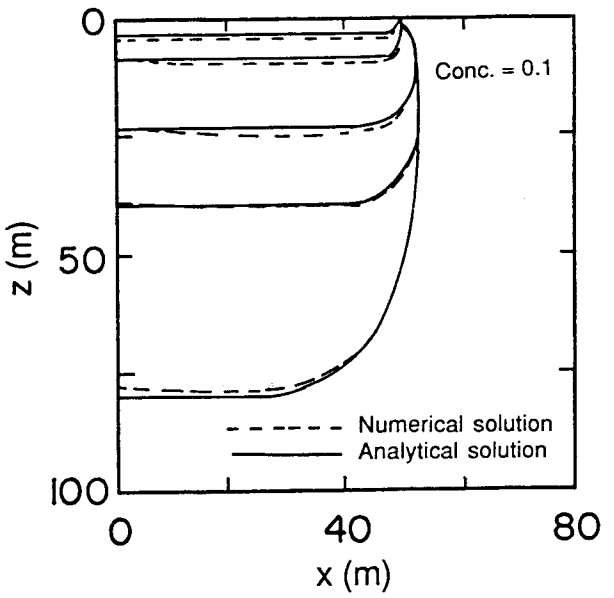

Fig. 7.12. Advancement of the concentration front (c=0.1) for example 3b as calculated with SWMS_2D (dotted lines) and the analytical solution (solid lines) .

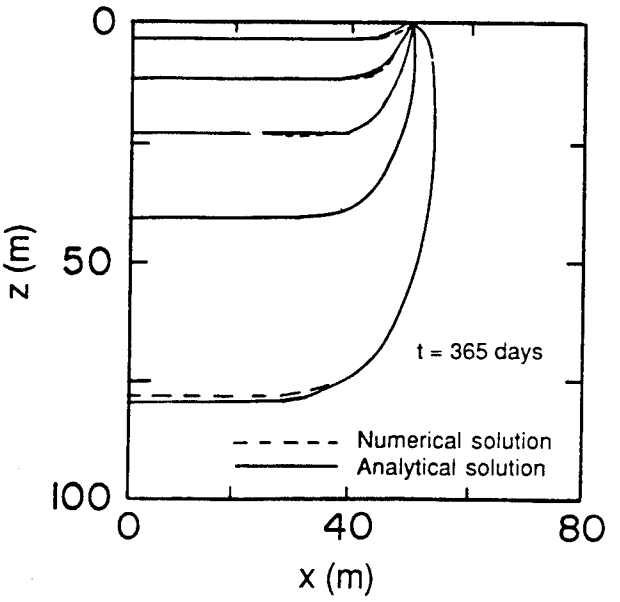

Fig. 7.13. Concentration profile at the end of the simulation (t=365 days) for example 3b as calculated with SWMS_2D (dotted lines) and the analytical solution (solid lines).

7.4. *Example 4 - Water and Solute Infiltration Test*

The SWMS_2D code was used to numerically simulate the movement of water and a dissolved solute from a single-ring infiltrometer into the upper part of the soil profile of example 2. The axisymmetric flow system and associated finite element mesh for the ponded infiltration experiment are shown in Figure 7.14. The example was used to illustrate variably-saturated water flow and solute transport in a layered and radially symmetric three-dimensional soil profile.

Calculations were carried out over a period of 5 days. The pressure head profile obtained in example problem 2 at the beginning of June 1982 was taken as the initial condition for the flow equation. The soil profile was assumed to be initially free of any solute. All sides of the flow region were considered to be impervious, except for a small portion around the origin at the surface (the ponded surface inside the ring infiltrometer)

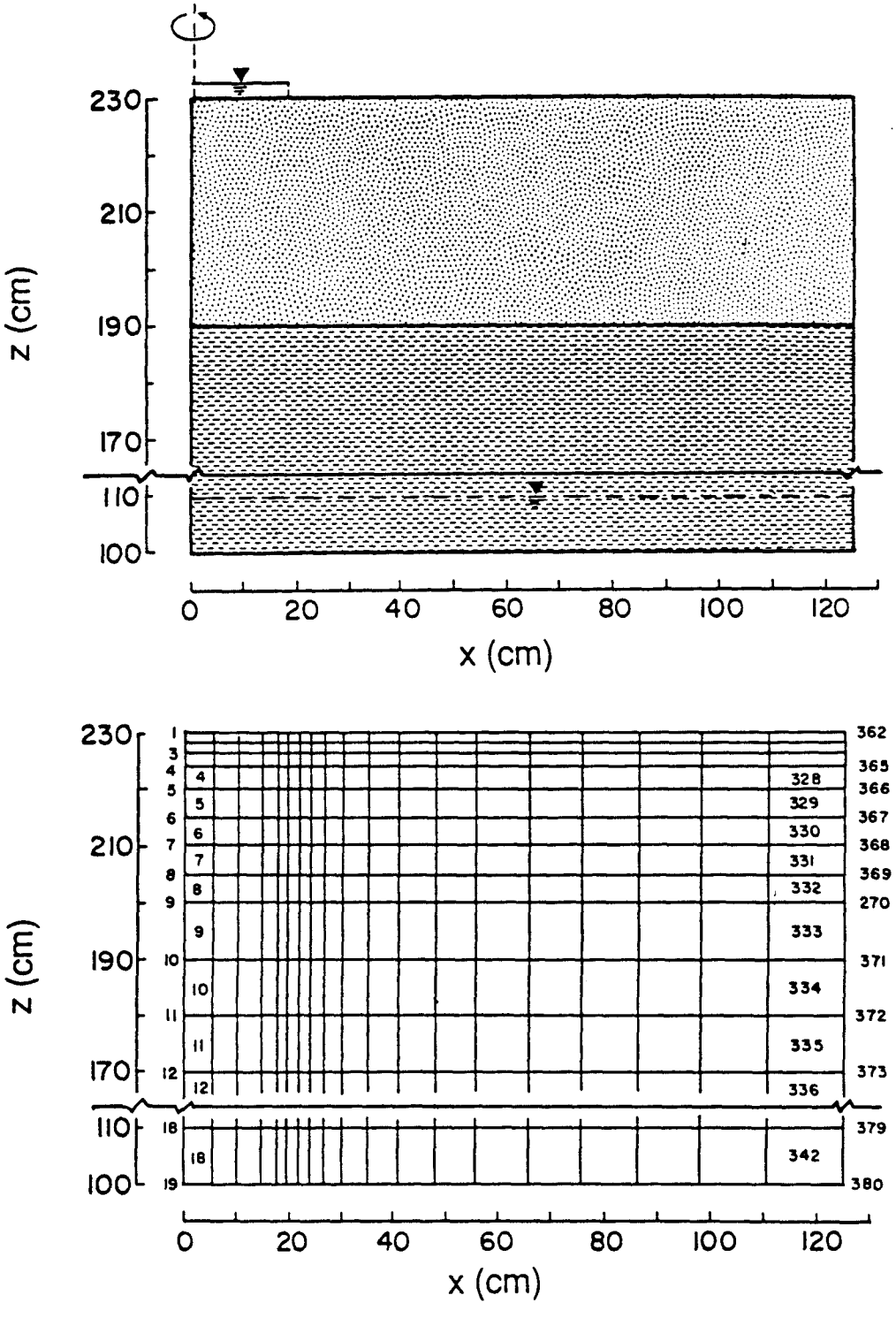

Fig. 7.14. Flow system and finite element mesh for example 4.

where constant pressure head and concentration boundary conditions were imposed. No root water extraction was considered.

Table 7.2 lists the unsaturated soil hydraulic and solute transport parameters for the two soil layers. Figures 7.15 and 7.16 presents calculated pressure head and concentration profiles, respectively, at two different times. Notice the relatively strong interaction between the infiltrating water and the saturated zone after 5 days (Fig. 7.15). The concentration front, on the other hand, did not quite reach the groundwater table during the simulation (Fig. 7.16).

Table 7.2. Input parameters for example 4.

Parameter	1st layer	2nd layer
	Hydraulic Parameters	
$\theta_s = \theta_m = \theta_k$	0.399	0.339
$\theta_r = \theta_a$	0.0001	0.0001
$K_s = K_k$ [m/day]	0.298	0.454
α [1/m]	1.74	1.39
n [-]	1.3757	1.6024
	Transport Parameters†	
ρ [kg/m^3]	1400	
D_d [m^2/day]	0.00374	
D_L [m]	0.005	
D_T [m]	0.001	
k [m^3/kg]	0.0001	
μ_w [1/day]	-0.05	
μ_s [1/day]	-0.01	
c_0 [-]	1.0	

†Assumed to be the same for both soil layers

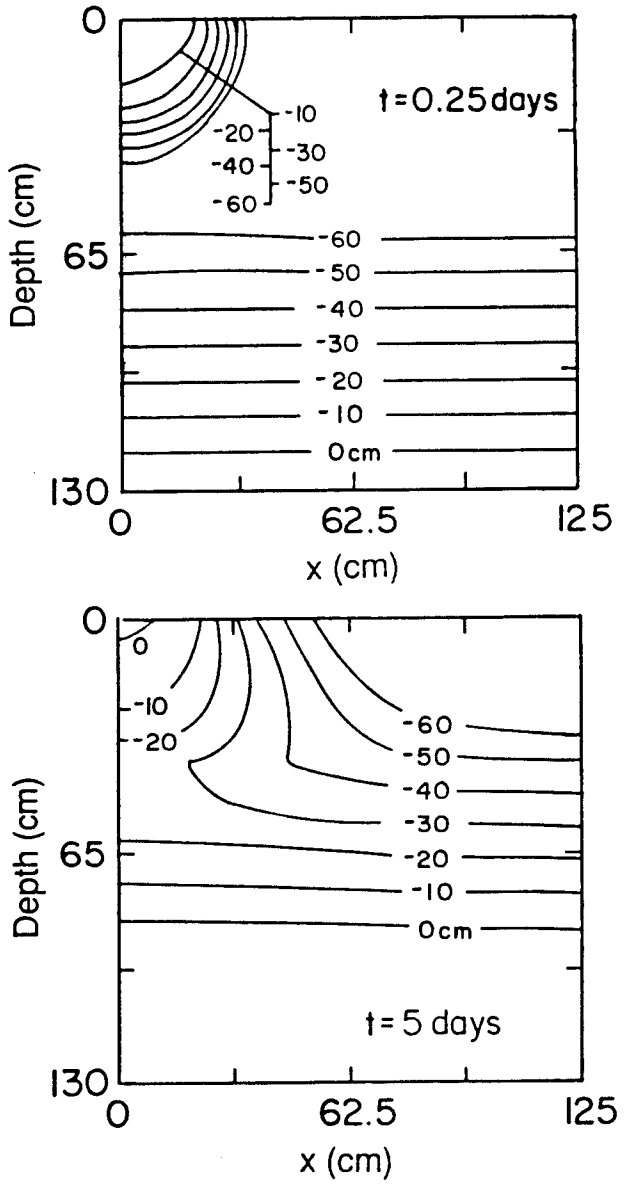

Fig. 7.15. Calculated pressure head profiles at $t = .25$ (top) and 5 days (bottom) for example 4.

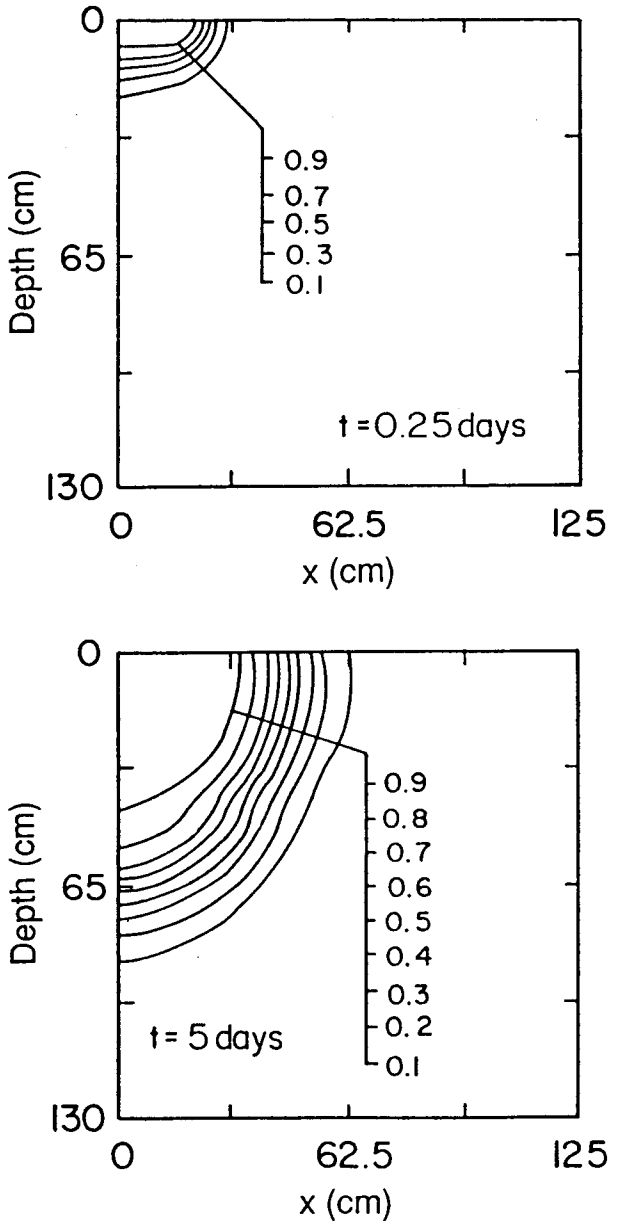

Fig. 7.16. Concentration profiles at $t = .25$ (top) and 5 days (bottom) for example 4.

8. INPUT DATA

The input data for SWMS_2D are given in three separate input files. These input files consist of one or more input blocks identified by the letters from A through K. The input files and blocks must be arranged as follows:

SELECTOR.IN
- A. Basic Information
- B. Material Information
- C. Time Information
- D. Root Water Uptake Information
- E. Seepage Information
- F. Drainage Information
- G. Solute Transport Information

GRID.IN
- H. Nodal Information
- I. Element Information
- J. Boundary Geometry Information

ATMOSPH.IN
- K. Atmospheric Information

The various input blocks are described in detail in Section 8.1, while Section 8.2 lists the actual input files for examples 1 through 4 discussed in Section 7. The output files for these examples are discussed in Section 9.

8.1. *Description of Data Input Blocks*

Tables 8.1 through 8.11 describe the data required for each input block. All data are read in using list-directed formatting (free format). Comment lines are provided at the beginning of, and within, each input block to facilitate, among other things, proper identification of the function of the block and the input variables. The comment lines are ignored during program execution; hence, they may be left blank but should not be omitted. All input files must be placed in the directory SWMS_2D.IN. The program assumes that all input data are specified in a consistent set of units for mass M, length L, and time T.

Most of the information in Tables 8.1 through 8.11 should be self-explanatory. Table 8.8 (Block H) is used to define, among other things, the nodal coordinates and initial conditions for the pressure head and the concentration. One short-cut may be used when generating the nodal coordinates. The short-cut is possible when two nodes (e.g., N_1 and N_2), not adjacent to each other, are located along a transverse line such that N_2 is greater than N_1+1. The program will automatically generate nodes between N_1 and N_2, provided all of the following conditions are met simultaneously: (1) all nodes along the transverse line between nodes N_1 and N_2 are spaced at equal intervals, (2) values of the input variables *hNew*(n), *Beta*(n), *Axz*(n), *Bxz*(n), *Dxz*(n), and *Conc*(n) vary linearly between nodes N_1 and N_2, and (3) values of *Kode*(n), $Q(n)$ and *MatNum*(n) are the same for all $n = N_1, N_1+1,..., N_2-1$ (see Table 8.8).

A similar short-cut is possible when generating the elements in Block I (Table 8.9). Consider two elements, E_1 and E_2, between two transverse lines such that E_2 is greater than E_1. The program requires input data only for element E_1 (i.e., data for elements E_1+1 through E_2 may be omitted), provided the following two conditions are met simultaneously: (1) all elements between E_1 and E_2 are quadrilaterals, including E_1 and E_2, and (2) all elements, $E_1,...., E_2$, are assigned the same values of *Angle*(e), *ConA*1(e), *ConA*2(e), and *LayNum*(e) as defined in Table 8.9.

Table 8.1. Block A - Basic information.

Record	Type	Variable	Description
1,2	-	-	Comment lines.
3	Char	*Hed*	Heading.
4	-	-	Comment line.
5	Char	*LUnit*	Length unit (e.g., 'cm').
5	Char	*TUnit*	Time unit (e.g., 'min').
5	Char	*MUnit*	Mass unit for concentration (e.g., 'g', 'mol', '-').
6	-	-	Comment line.
7	Integer	*Kat*	Type of flow system to be analyzed: 0 for a horizontal (areal) system 1 for axisymmetric flow 2 for vertical flow in a cross-section
8	-	-	Comment line.
9	Integer	*MaxIt*	Maximum number of iterations allowed during any time step (usually 20).
9	Real	*TolTh*	Absolute water content tolerance for nodes in the unsaturated part of the flow region [-] (its recommended value is 0.0001). *TolTh* represents the maximum desired absolute change in the value of the water content, θ, between two successive iterations during a particular time step.
9	Real	*TolH*	Absolute pressure head tolerance for nodes in the saturated part of the flow region [L] (its recommended value is 0.1 cm). *TolH* represents the maximum desired absolute change in the value of the pressure head, h, between two successive iterations during a particular time step.
10	-	-	Comment line.
11	Logical	*lWat*	Set this logical variable equal to **.true.** when transient water flow is considered. Set this logical variable equal to **.false.** when steady-state water flow is to be calculated.
11	Logical	*lChem*	Set this logical variable equal to **.true.** if solute transport is to be considered.
11	Logical	*CheckF*	Set this logical variable equal to **.true.** if the grid input data are to be printed for checking.
11	Logical	*ShortF*	**.true.** if information is to be printed only at preselected times, but not at each time step (T-level information, see Section 9.1), **.false.** if information is to be printed at each time step.
11	Logical	*FluxF*	**.true.** if detailed information about the element fluxes and discharge/recharge rates is to be printed.

Table 8.1. (continued)

Record	Type	Variable	Description
11	Logical	*AtmInf*	**.true.** if atmospheric boundary conditions are supplied via the input file ATMOSPH.IN, **.false.** if the file ATMOSPH.IN is not provided (i.e., in case of time independent boundary conditions).
11	Logical	*SeepF*	**.true.** if one or more seepage faces is to be considered.
11	Logical	*FreeD*	Set this logical variable equal to **.true.** if a unit vertical hydraulic gradient boundary condition (free drainage) is used at the bottom boundary. Otherwise set equal to **.false.** .
11	Logical	*DrainF*	Set this logical variable equal to **.true.** if a drain is to be simulated by means of boundary condition. Otherwise set equal to **.false.** . Section 4.3.7 explains how tile drains can be represented as boundary conditions in a regular finite element mesh.

Table 8.2. Block B - Material information.

Record	Type	Variable	Description
1,2	-	-	Comment lines.
3	Integer	*NMat*	Number of soil materials. Materials are identified by the material number, *MatNum*, specified in Block H.
3	Integer	*NLay*	Number of subregions for which separate water balances are being computed. Subregions are identified by the subregion number, *LayNum*, specified in Block I.
3	Real	*ha*	Absolute value of the upper limit [L] of the pressure head interval below which a table of hydraulic properties will be generated internally for each material (h_a must be greater than 0.0; e.g. 0.001 cm) (see Section 4.3.11).
3	Real	*hb*	Absolute value of the lower limit [L] of the pressure head interval for which a table of hydraulic properties will be generated internally for each material (e.g. 1000 m). One may assign to h_b the highest (absolute) expected pressure head to be expected during a simulation. If the absolute value of the pressure head during program execution lies outside of the interval $[h_a, h_b]$, then appropriate values for the hydraulic properties are computed directly from the hydraulic functions (i.e., without interpolation in the table).
3	Integer	*NPar*	Number of parameters specified for each material (i.e., 9 in case of the modified van Genuchten model). If the original van Genuchten model is to be used, then set $\theta_a=\theta_r$, $\theta_m=\theta_k=\theta_s$ and $K_k=K_s$ (see Section 2.3 for the description of unsaturated soil hydraulic properties).
4	-	-	Comment line.
5	Real	*Par*(1,*M*)	Parameter θ_r for material *M* [-].
5	Real	*Par*(2,*M*)	Parameter θ_s for material *M* [-].
5	Real	*Par*(3,*M*)	Parameter θ_a for material *M* [-].
5	Real	*Par*(4,*M*)	Parameter θ_m for material *M* [-].
5	Real	*Par*(5,*M*)	Parameter α for material *M* [L^{-1}].
5	Real	*Par*(6,*M*)	Parameter n for material *M* [-].
5	Real	*Par*(7,*M*)	Parameter K_s for material *M* [LT^{-1}].
5	Real	*Par*(8,*M*)	Parameter K_k for material *M* [LT^{-1}].
5	Real	*Par*(9,*M*)	Parameter θ_k for material *M* [-].
			Record 5 information is provided for each material *M* (from 1 to *NMat*).

Table 8.3. Block C - Time information.

Record	Type	Variable	Description
1,2	-	-	Comment lines.
3	Real	*dt*	Initial time increment, Δt [T]. Initial time step should be estimated in dependence on the problem solved. For problems with high pressure gradients (e.g. infiltration into an initially dry soil), Δt should be relatively small.
3	Real	*dtMin*	Minimum permitted time increment, Δt_{min} [T].
3	Real	*dtMax*	Maximum permitted time increment, Δt_{max} [T].
3	Real	*dMul*	If the number of required iterations at a particular time step is less than or equal to 3, then Δt for the next time step is multiplied by a dimensionless number $dMul \geq 1.0$ (its value is recommended not to exceed 1.3).
3	Real	*dMul2*	If the number of required iterations at a particular time step is greater than or equal to 7, then Δt for the next time step is multiplied by $dMul2 \leq 1.0$ (e.g. 0.33).
3	Integer	*MPL*	Number of specified print-times at which detailed information about the pressure head, water content, concentration, flux, and the soil water and solute balances will be printed.
4	-	-	Comment line.
5	Real	*TPrint*(1)	First specified print-time [T].
5	Real	*TPrint*(2)	Second specified print-time [T].
.	.	.	.
.	.	.	.
5	Real	*TPrint*(*MPL*)	Last specified print-time [T].

Table 8.4. Block D - Root water uptake information.†

Record	Type	Variable	Description
1,2	-	-	Comment lines.
3	Real	*P0*	Value of the pressure head, h_1 (Fig. 2.1), below which roots start to extract water from the soil.
3	Real	*P2H*	Value of the limiting pressure head, h_3, below which the roots cannot extract water at the maximum rate (assuming a potential transpiration rate of *r2H*).
3	Real	*P2L*	As above, but for a potential transpiration rate of *r2L*.
3	Real	*P3*	Value of the pressure head, h_4, below which root water uptake ceases (usually equal to the wilting point).
3	Real	*r2H*	Potential transpiration rate [LT^{-1}] (currently set at 0.5 cm/day).
3	Real	*r2L*	Potential transpiration rate [LT^{-1}] (currently set at 0.1 cm/day).
			The above input parameters permit one to make the variable h_3 a function of the potential transpiration rate, T_p (h_3 presumably decreases at higher transpiration rates). SWMS_2D currently implements the same linear interpolation scheme as used in several versions of the SWATRE code (e.g., *Wesseling and Brandyk*, 1985). The scheme is based on the following interpolation: $$h_3 = P2H + \frac{P2L - P2H}{r2H - r2L}(r2H - T_p) \quad for\ r2L < T_p < r2H$$ $$h_3 = P2L \quad for\ T_a \le r2L$$ $$h_3 = P2H \quad for\ T_a \ge r2H$$
4	-	-	Comment line.
5	Real	*POptm*(1)	Value of the pressure head, h_2, below which roots start to extract water at the maximum possible rate (material number 1).
5	Real	*POptm*(2)	As above (material number 2).
.	.	.	.
.	.	.	.
5	Real	*POptm*(*NMat*)	As above (for material number *NMat*).

† Block D is not read in if the logical variable *SinkF* (Block K) is set equal to .false. .

Table 8.5. Block E - Seepage face information.†

Record	Type	Variable	Description
1,2	-	-	Comment lines.
3	Integer	*NSeep*	Number of seepage faces expected to develop.
4	-	-	Comment line.
5	Integer	*NSP*(1)	Number of nodes on the first seepage face.
5	Integer	*NSP*(2)	Number of nodes on the second seepage face.
.	.	.	.
.	.	.	.
5	Integer	*NSP*(*NSeep*)	Number of nodes on the last seepage face.
6	-	-	Comment line.
7	Integer	*NP*(1,1)	Sequential global number of the first node on the first seepage face.
7	Integer	*NP*(1,2)	Sequential global number of the second node on the first seepage face.
.	.	.	.
.	.	.	.
7	Integer	*NP*(1,*NSP*(1))	Sequential global number of the last node on the first seepage face.
			Record 7 information is provided for each seepage face.

† Block E is not read in if the logical variable *SeepF* (Block A) is set equal to .false. .

Table 8.6. Block F - Drainage information.†

Record	Type	Variable	Description
1,2	-	-	Comment lines.
3	Integer	*NDr*	Number of drains. See Section 4.3.7 for a discussion on how tile drains can be represented as boundary conditions in a regular finite element mesh.
3	Real	*DrCorr*	Additional reduction in the correction factor C_d (See Section 4.3.7).
4	-	-	Comment line.
5	Integer	*ND*(1)	Global number of the first drain.
5	Integer	*ND*(2)	Global number of the second drain.
.	.	.	.
.	.	.	.
5	Integer	*ND*(*NDr*)	Global number of the last drain.
6	-	-	Comment line.
7	Integer	*NElD*(1)	Number of elements surrounding the first drain.
7	Integer	*NElD*(2)	Number of elements surrounding the second drain.
.	.	.	.
.	.	.	.
7	Integer	*NElD*(*NDr*)	Number of elements surrounding the last drain.
8	-	-	Comment line.
9	Real	*EfDim*(1,1)	Effective diameter of the first drain (see Section 4.3.7).
9	Real	*EfDim*(2,1)	Dimension of the square in finite element mesh representing the first drain (see Section 4.3.7).
			Record 9 information is provided for each drain.
10	-	-	Comment line.
11	Integer	*KElDr*(1,1)	Global number of the first element surrounding the first drain.
11	Integer	*KElDr*(1,2)	Global number of the second element surrounding the first drain.
.	.	.	.
.	.	.	.
11	Integer	*KElDr*(1,*NElD*(1))	Global number of the last element surrounding the first drain.
			Record 11 information is provided for each drain.

† Block F is not read in if the logical variable *DrainF* (Block A) is set equal to .false. .

Table 8.7. Block G - Solute transport information.†

Record	Type	Variable	Description
1,2	-	-	Comment lines.
3	Real	*Epsi*	Temporal weighing coefficient. =0.0 for an explicit scheme. =0.5 for a Crank-Nicholson implicit scheme. =1.0 for a fully implicit scheme.
3	Logical	*lUpW*	**.true.** if upstream weighing formulation is to be used. **.false.** if the original Galerkin formulation is to be used.
3	Logical	*lArtD*	**.true.** if artificial dispersion is to be added in order to fulfill the stability criterion *PeCr* (see Section 5.3.6). **.false.** otherwise.
3	Real	*PeCr*	Stability criterion (see Section 5.3.6). Set equal to zero when *lUpW* is equal to **.true.**.
4	-	-	Comment line.
5	Real	*ChPar*(1,*M*)	Bulk density of material *M*, ρ [ML^{-3}].
5	Real	*ChPar*(2,*M*)	Ionic or molecular diffusion coefficient in free water, D_d [L^2T^{-1}].
5	Real	*ChPar*(3,*M*)	Longitudinal dispersivity for material type *M*, D_L [L].
5	Real	*ChPar*(4,*M*)	Transverse dispersivity for material type *M*, D_T [L].
5	Real	*ChPar*(5,*M*)	Freundlich isotherm coefficient for material type *M*, *k* [$M^{-1}L^3$].
5	Real	*ChPar*(6,*M*)	First-order rate constant for dissolved phase, material type *M*, μ_w [T^{-1}].
5	Real	*ChPar*(7,*M*)	First-order rate constant for solid phase, material type *M*, μ_s [T^{-1}].
5	Real	*ChPar*(8,*M*)	Zero-order rate constant for dissolved phase, material type *M*, γ_w [$ML^{-3}T^{-1}$].
5	Real	*ChPar*(9,*M*)	Zero-order rate constant for solid phase, material type *M*, γ_s [T^{-1}]. Record 5 information is provided for each material *M* (from 1 to *NMat*).
6	-	-	Comment line.
7	Integer	*KodCB*(1)	Code specifying the type of boundary condition for solute transport applied to a particular node. Positive and negative signs indicate that first-, or second- or third- (depending upon the calculated water flux *Q*) type boundary condition are implemented, respectively. In case of time-independent boundary conditions (*Kode*(1) = ±1, ±2, or ±6 - See Block H), *KodCB*(1) also refers to the field *cBound* for the value of the solute transport boundary condition. The value of *cBound*(abs(*KodCB*(1))) specifies the boundary condition for node *KXB*(1) (the first of a set of sequentially numbered boundary nodes for which *Kode*(*N*) is not equal to zero). Permissible values are ±1, ±2,..., ±5, ±6.
7	Integer	*KodCB*(2)	Same as above for the second boundary node.
.	.	.	.
.	.	.	.
7	Integer	*KodCB*(*NumBP*)	Same as above for the last boundary node.
8	-	-	Comment line.

Table 8.7. (continued)

Record	Type	Variable	Description
9	Real	*cBound*(1)	Value of the concentration for the first time-independent BC [ML^{-3}]. Set equal to zero if no $KodCB(n) = \pm 1$ is specified.
9	Real	*cBound*(2)	Value of the concentration for the second time-independent BC [ML^{-3}]. Set equal to zero if no $KodCB(n) = \pm 2$ is specified.
.	.	.	.
.	.	.	.
9	Real	*cBound*(5)	Value of the concentration for the fifth time-independent BC [ML^{-3}]. If water uptake is specified then *cBound*(5) is automatically used for the concentration of water removed from the flow region by root water uptake [ML^{-3}]. Set equal to zero if no $KodCB(n) = \pm 5$ and no root solute uptake is specified.
9	Real	*cBound*(6)	Value of the concentration for the sixth time-independent BC [ML^{-3}]. If internal sources are specified then *cBound*(6) is automatically used for the concentration of water injected into the flow region through internal sources [ML^{-3}]. Set equal to zero if no $KodCB(n) = \pm 6$ and no internal sources are specified.
10	-	-	Comment line.
11	Real	*tPulse*	Time duration of the concentration pulse [T].

†Block G is not needed when the logical variable *lChem* in Block A is set equal to .false..

Table 8.8. Block H - Nodal information.

Record	Type	Variable	Description
1,2	-	-	Comment lines.
3	Integer	*NumNP*	Number of nodal points.
3	Integer	*NumEl*	Number of elements (quadrilaterals and/or triangles).
3	Integer	*IJ*	Maximum number of nodes on any transverse line.
3	Integer	*NumBP*	Number of boundary nodes for which *Kode(N)* is not equal to 0.
3	Integer	*NObs*	Number of observation nodes for which values of the pressure head, water content, and concentration (for *lChem* = .true.) are printed at each time level.
4	-	-	Comment line.
5	Integer	*n*	Nodal number.
5	Integer	*Kode(n)*	Code specifying the type of boundary condition applied to a particular node. Permissible values are $0, \pm 1, \pm 2, \pm 3, \pm 4, ..., \pm NumKD$ (see Section 6.3).
5	Real	*x(n)*	*x*-coordinate of node *n* [L] (always a horizontal coordinate).
5	Real	*z(n)*	*z*-coordinate of node *n* [L]. *z* is the vertical coordinate for problems involving vertical planar or axisymmetric flow. For axisymmetric flow, *z* coincides with the vertical axis of symmetry.
5	Real	*hNew(n)*	Initial value of the pressure head at node *n* [L]. If *lWat* = **.false.** in Block A, then *hNew(n)* represents the initial guess of the pressure head for steady state conditions.
5	Real	*Conc(n)*	Initial value of the concentration at node *n* [ML^{-3}] (set = 0 if *lChem* = **.false.**)
5	Real	*Q(n)*	Prescribed recharge/discharge rate at node *n*; [L^2T^{-1}] for planar flow, [L^3T^{-1}] for axisymmetric flow. *Q(n)* is negative when directed out of the system. When no value for *Q(n)* is needed, set *Q(n)* equal to zero.
5	Integer	*MatNum(n)*	Index for material whose hydraulic and transport properties are assigned to node *n*.
5	Real	*Beta(n)*	Value of the water uptake distribution, $b(x,z)$, in the soil root zone at node *n*. Set *Beta(n)* equal to zero if node *n* lies outside the root zone.
5	Real	*Axz(n)*	Nodal value of the dimensionless scaling factor α_h associated with the pressure head.
5	Real	*Bxz(n)*	Nodal value of the scaling factor α_K associated with the saturated hydraulic conductivity.
5	Real	*Dxz(n)*	Nodal value of the scaling factor α_θ associated with the water content.
			In general, record 5 information is required for each node *n*, starting with *n* = 1 and continuing sequentially until *n* = *NumNP*. Record 5 information for certain nodes may be skipped if several conditions are satisfied (see beginning of this section)

Table 8.9. Block I - Element information.

Record	Type	Variable	Description
1,2	-	-	Comment lines.
3	Integer	e	Element number.
3	Integer	$KX(e,1)$	Global nodal number of corner node i.
3	Integer	$KX(e,2)$	Global nodal number of corner node j.
3	Integer	$KX(e,3)$	Global nodal number of corner node k.
3	Integer	$KX(e,4)$	Global nodal number of corner node l. Indices i, j, k, and l, refer to the corner nodes of an element e taken in a counter-clockwise direction. For triangular elements $KX(e,4)$ must be equal to $KX(e,3)$.
3	Real	$Angle(e)$	Angle in degrees between K_1^A and the x-coordinate axis assigned to each element e.
3	Real	$ConA1(e)$	First principal component, K_1^A, of the dimensionless tensor $\mathbf{K}^A$ which describes the local anisotropy of the hydraulic conductivity assigned to element e.
3	Real	$ConA2(e)$	Second principal component, K_2^A.
3	Integer	$LayNum(e)$	Subregion number assigned to element e.
			In general, record 3 information is required for each element e, starting with $e=1$ and continuing sequentially until $e=NumEl$. Record 3 information for certain elements may be skipped if several conditions are satisfied (see beginning of this section).

Table 8.10. Block J - Boundary geometry information.

Record	Type	Variable	Description
1,2	-	-	Comment lines.
3	Integer	*KXB*(1)	Global node number of the first of a set of sequentially numbered boundary nodes for which *Kode*(*n*) is not equal to zero.
3	Integer	*KXB*(2)	As above for the second boundary node.
.	.	.	.
.	.	.	.
3	Integer	*KXB*(*NumBP*)	As above for the last boundary node.
4	-	-	Comment line.
5	Real	*Width*(1)	Width of the boundary [L] associated with boundary node *KXB*(1). *Width*(*n*) includes half the boundary length of each element connected to node *KXB*(*n*) along the boundary. The type of boundary condition assigned to *KXB*(*n*) is determined by the value of *Kode*(*n*). For axisymmetric flow, *Width*(*n*) represents the area of the boundary strip [L^2] associated with node *KXB*(*n*); this area should be calculated along a horizontal boundary as $$Width(j) = \frac{\pi}{3}\left[(x_{j-1} + 2x_j)(x_j - x_{j-1}) + (x_{j+1} + 2x_j)(x_{j+1} - x_j)\right]$$ If a unit vertical hydraulic gradient or a deep drainage boundary condition is specified at node *n*, then *Width*(*n*) represents only the horizontal component of the boundary.
5	Real	*Width*(2)	As above for node *KXB*(2).
.	.	.	.
.	.	.	.
5	Real	*Width*(*NumBP*)	As above for node *KXB*(*NumBP*).
6	-	-	Comment line.
7	Real	*rLen*	Width of soil surface associated with transpiration [L]; represents surface area [L^2] in case of axisymmetrical flow. Set *rLen* equal to zero for problems without transpiration.
8	-	-	Comment line.
9	Integer	*Node*(1)	Global node number of the first observation node for which values of the pressure head, water content, and concentration (for *lChem* = .true.) are printed at each time level.
9	Integer	*Node*(2)	Same as above for the second observation node.
.	.	.	.
.	.	.	.
9	Integer	*Node*(*NObs*)	Same as above for the last observation node.

Table 8.11. Block K - Atmospheric information.†

Record	Type	Variable	Description
1,2,3,4	-	-	Comment lines.
5	Logical	*SinkF*	Set this variable equal to **.true.** if water extraction from the root zone is imposed.
5	Logical	*qGWLF*	Set this variable equal to **.true.** if the discharge-groundwater level relationship $q(GWL)$ given by equation (6.1) is used as the bottom boundary condition; $GWL = h - GWL0L$, where h is the pressure head at the boundary.
6	-	-	Comment line.
7	Real	*GWL0L*	Reference position of groundwater table (usually the z-coordinate of the soil surface).
7	Real	*Aqh*	Value of the parameter A_{qh} in the $q(GWL)$-relationship (equation (6.1)); set to zero if *qGWLF* = **.false.**
7	Real	*Bqh*	Value of the parameter B_{qh} in the $q(GWL)$-relationship (equation (6.1)); set to zero if *qGWLF* = **.false.**
8	-	-	Comment line.
9	Real	*tInit*	Starting time [T] of the simulation.
9	Integer	*MaxAl*	Number of atmospheric data records.
10	-	-	Comment line.
11	Real	*hCritS*	Maximum allowed pressure head at the soil surface [L].
12	-	-	Comment line.
13	Real	*tAtm(i)*	Time for which the i-th data record is provided [T].
13	Real	*Prec(i)*	Precipitation [LT^{-1}] (in absolute value).
13	Real	*cPrec(i)*	Solute concentration of rainfall water [ML^{-3}] (set = 0 if *lChem* = **.false.**).
13	Real	*rSoil(i)*	Potential evaporation rate [LT^{-1}] (in absolute value).
13	Real	*rRoot(i)*	Potential transpiration rate [LT^{-1}] (in absolute value).
13	Real	*hCritA(i)*	Absolute value of the minimum allowed pressure head at the soil surface [L].
13	Real	*rGWL(i)*	Drainage flux [LT^{-1}] across the bottom boundary, or other time-dependent prescribed flux boundary condition (positive when water leaves the flow region), for nodes where $Kode(n) = -3$; set to zero when no $Kode(n) = -3$ boundary condition is specified.
13	Real	*GWL(i)*	Groundwater level [L] (usually negative), or other time-dependent prescribed head boundary condition, for nodes where $Kode(n) = +3$; set equal to zero when no $Kode(n) = +3$ is specified. The prescribed value of the pressure head is $h = GWL + GWL0L$.

Table 8.11. (continued)

Record	Type	Variable	Description
13	Real	*crt(i)*	Time-dependent concentration of the drainage flux [ML^{-3}], or some other time-dependent prescribed flux, for nodes where $Kode(n) = \pm 3$ and $KodCB(n) < 0$ (this variable does not need to be specified if *lChem* = .false.; set equal to zero when no $Kode(n) = \pm 3$ and $KodCB(n) < 0$, or when the flux *rGWL(i)* is directed out of the flow domain).
13	Real	*cht(i)*	Time-dependent concentration [ML^{-3}] for the first-type boundary condition prescribed for nodes for which $Kode(n) = \pm 3$ and $KodCB(n) > 0$ (does not need to be specified if *lChem* = .false.; set equal to zero when no $Kode(n) = \pm 3$ and $KodCB(n) > 0$).
			The total number of atmospheric data records is *MaxAl* (i=1,2, ..,*MaxAl*).

† Block K is not read in if the logical variable *AtmInf* (Block A) is set equal to .false.

Table 8.12. Input data for example 1 (input file 'SELECTOR.IN').

```
*** BLOCK A: BASIC INFORMATION ***************************************
Heading
'Example 1 - Column Test'
LUnit  TUnit  MUnit          (units are obligatory for all input data)
'cm'   'sec'  '-'
Kat (0:horizontal plane, 1:axisymmetric vertical flow, 2:vertical plane)
  2
MaxIt   TolTh    TolH (max. number of iterations and precis. tolerances)
  20    .0001    .1
lWat    lChem    CheckF  ShortF  FluxF   AtmInF  SeepF  FreeD  DrainF
 t      f        f       t       t       f       t      f      f
*** BLOCK B: MATERIAL INFORMATION ************************************
NMat    NLay    hTab1   hTabN   NPar
  1      1      .001    200.     9
thr     ths     tha     thm     Alfa    n       Ks      Kk      thk
 .02    .350    .02     .350    .0410   1.964  .000722 .000695 .2875
*** BLOCK C: TIME INFORMATION ****************************************
dt      dtMin   dtMax   DMul    DMul2   MPL
  1.    .01     60.     1.1     .33     6
TPrint(1),TPrint(2),...,TPrint(MPL)                  (print-time array)
 60 900 1800 2700 3600 5400
*** BLOCK E: SEEPAGE INFORMATION (only if SeepF =.true.) ***************
NSeep                                          (number of seepage faces)
  1
NSP(1),NSP(2),.......,NSP(NSeep)          (number of nodes in each s.f.)
  2
NP(i,1),NP(i,2),.....,NP(i,NSP(i))     (nodal number array of i-th s.f.)
  111     112
*** END OF INPUT FILE 'SELECTOR.IN' **********************************
```

Table 8.13. Input data for example 1 (input file 'GRID.IN').

```
*** BLOCK H: NODAL INFORMATION ******************************************************
     NumNP      NumEl       IJ       NumBP      NObs
      112        55          2         4         0
   n  Code     x       z         h       Conc        Q      M    B    Axz   Bxz   Dxz
   1   1     .00   61.00       .75    .10E+01   .00E+00  1  .00  1.00  1.00  1.00
   2   1    1.00   61.00       .75    .10E+01   .00E+00  1  .00  1.00  1.00  1.00
   3   0     .00   60.75   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
   4   0    1.00   60.75   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
   5   0     .00   60.50   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
   6   0    1.00   60.50   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
   7   0     .00   60.25   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
   8   0    1.00   60.25   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
   9   0     .00   60.00   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
  10   0    1.00   60.00   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
  11   0     .00   59.50   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
  12   0    1.00   59.50   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
  13   0     .00   59.00   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
  14   0    1.00   59.00   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
       .      .       .        .         .          .      .   .     .     .
       .      .       .        .         .          .      .   .     .     .
       .      .       .        .         .          .      .   .     .     .
 103   0     .00    8.00   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
 104   0    1.00    8.00   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
 105   0     .00    6.00   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
 106   0    1.00    6.00   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
 107   0     .00    4.00   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
 108   0    1.00    4.00   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
 109   0     .00    2.00   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
 110   0    1.00    2.00   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
 111  -2     .00     .00   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
 112  -2    1.00     .00   -150.00    .00E+00   .00E+00  1  .00  1.00  1.00  1.00
*** BLOCK I: ELEMENT INFORMATION ****************************************************
   e   i   j   k   l   Angle  Aniz1  Aniz2  LayNum
   1   1   3   4   2    .00   1.00   1.00   1
   2   3   5   6   4    .00   1.00   1.00   1
   3   5   7   8   6    .00   1.00   1.00   1
   4   7   9  10   8    .00   1.00   1.00   1
   5   9  11  12  10    .00   1.00   1.00   1
   6  11  13  14  12    .00   1.00   1.00   1
   7  13  15  16  14    .00   1.00   1.00   1
   8  15  17  18  16    .00   1.00   1.00   1
   9  17  19  20  18    .00   1.00   1.00   1
  10  19  21  22  20    .00   1.00   1.00   1
       .       .        .      .
       .       .        .      .
       .       .        .      .
  48  95  97  98  96    .00   1.00   1.00   1
  49  97  99 100  98    .00   1.00   1.00   1
  50  99 101 102 100    .00   1.00   1.00   1
  51 101 103 104 102    .00   1.00   1.00   1
  52 103 105 106 104    .00   1.00   1.00   1
  53 105 107 108 106    .00   1.00   1.00   1
  54 107 109 110 108    .00   1.00   1.00   1
  55 109 111 112 110    .00   1.00   1.00   1
*** BLOCK J: BOUNDARY GEOMETRY INFORMATION ******************************************
 Node number array:
        1         2        111        112
 Width array:
      .50       .50        .50        .50
 Length:
     0.00
*** END OF INPUT FILE 'GRID.IN' *****************************************************
```

Table 8.14. Input data for example 2 (input file 'SELECTOR.IN').

```
*** BLOCK A: BASIC INFORMATION ***********************************************
Heading
'Example 2 - Grass Field Problem (Hupselse Beek 1982)'
LUnit  TUnit  MUnit  (indicated units are obligatory for all input data)
'cm'   'day'  '-'
Kat (0:horizontal plane, 1:axisymmetric vertical flow, 2:vertical plane)
  2
MaxIt   TolTh   TolH  (max. number of iterations and precis. tolerances)
  20    .0001   0.1
lWat    lChem   CheckF  ShortF  FluxF   AtmInF  SeepF   FreeD   DrainF
 t       f       f       t       t       t       f       f       f
*** BLOCK B: MATERIAL INFORMATION ********************************************
NMat    NLay    hTab1   hTabN   NPar
  2      2      .001    1000.    9
thr     ths     tha     thm     Alfa    n       Ks      Kk      thk
.0001   .399    .0001   .399    .0174   1.3757  29.75   29.75   .399
.0001   .339    .0001   .339    .0139   1.6024  45.34   45.34   .339
*** BLOCK C: TIME INFORMATION ************************************************
dt      dtMin   dtMax   DMul    DMul2   MPL
 .02    1e-10   0.50    1.3     .3      6
TPrint(1),TPrint(2),...,TPrint(MPL)                  (print-time array)
 120 151 181 212 243 273
*** BLOCK D: SINK INFORMATION ************************************************
P0      P2H     P2L     P3      r2H     r2L
 -10.   -200.   -800.   -8000.  0.5     0.1
POptm(1),POptm(2),...,POptm(NMat)
 -25.   -25.
*** END OF INPUT FILE 'SELECTOR.IN' ******************************************
```

Table 8.15. Input data for example 2 (input file 'ATMOSPH.IN').

```
*** BLOCK K: ATMOSPHERIC INFORMATION  ***********************************
*** Hupselse Beek 1982                ***********************************
*****************************************************************************
SinkF   qGWLF
 t       t
GWLOL   Aqh      Bqh                              (if qGWLF=f then Aqh=Bqh=0)
 230    -.1687   -.02674
tInit   MaxAL                (MaxAL = number of atmospheric data-records)
 90.     183
hCritS                  (max. allowed pressure head at the soil surface)
 1e30
  tAtm    Prec    cPrec    rSoil    rRoot  hCritA     rt     ht    crt    cht
    91       0        0        0     0.16  1000000     0      0      0      0
    92    0.07        0        0     0.18  1000000     0      0      0      0
    93    0.02        0        0     0.13  1000000     0      0      0      0
    94       0        0        0     0.20  1000000     0      0      0      0
    95       0        0        0     0.28  1000000     0      0      0      0
    96    0.07        0        0     0.18  1000000     0      0      0      0
    97    0.29        0        0     0.08  1000000     0      0      0      0
    98    0.44        0        0     0.14  1000000     0      0      0      0
    99    0.20        0        0     0.11  1000000     0      0      0      0
   100    0.29        0        0     0.11  1000000     0      0      0      0
   101    0.32        0        0     0.11  1000000     0      0      0      0
   102    0.49        0        0     0.11  1000000     0      0      0      0
   103    0.01        0        0     0.16  1000000     0      0      0      0
   104       0        0        0     0.17  1000000     0      0      0      0
   105       0        0        0     0.22  1000000     0      0      0      0
   106       0        0        0     0.21  1000000     0      0      0      0
   107       0        0        0     0.23  1000000     0      0      0      0
   108       0        0        0     0.23  1000000     0      0      0      0
   109       0        0        0     0.24  1000000     0      0      0      0
   110       0        0        0     0.18  1000000     0      0      0      0
   111       0        0        0     0.15  1000000     0      0      0      0
   112       0        0        0     0.19  1000000     0      0      0      0
   113    0.01        0        0     0.15  1000000     0      0      0      0
   114    0.01        0        0     0.22  1000000     0      0      0      0
   115       0        0        0     0.23  1000000     0      0      0      0
   116    0.02        0        0     0.20  1000000     0      0      0      0
   117       0        0        0     0.17  1000000     0      0      0      0
   118    0.02        0        0     0.14  1000000     0      0      0      0
   119    0.26        0        0     0.13  1000000     0      0      0      0
   120    0.24        0        0     0.11  1000000     0      0      0      0
     .       .        .        .        .        .     .      .      .
     .       .        .        .        .        .     .      .      .
     .       .        .        .        .        .     .      .      .
   256       0        0        0     0.13  1000000     0      0      0      0
   257       0        0        0     0.14  1000000     0      0      0      0
   258       0        0        0     0.20  1000000     0      0      0      0
   259       0        0        0     0.14  1000000     0      0      0      0
   260       0        0        0     0.19  1000000     0      0      0      0
   261       0        0        0     0.14  1000000     0      0      0      0
   262       0        0        0     0.20  1000000     0      0      0      0
   263    0.35        0        0     0.23  1000000     0      0      0      0
   264    0.52        0        0     0.16  1000000     0      0      0      0
   265       0        0        0     0.21  1000000     0      0      0      0
   266       0        0        0     0.19  1000000     0      0      0      0
   267       0        0        0     0.18  1000000     0      0      0      0
   268       0        0        0     0.18  1000000     0      0      0      0
   269    0.53        0        0     0.09  1000000     0      0      0      0
   270    0.07        0        0     0.23  1000000     0      0      0      0
   271       0        0        0     0.17  1000000     0      0      0      0
   272       0        0        0     0.22  1000000     0      0      0      0
   273    1.04        0        0        0  1000000     0      0      0      0
*** END OF INPUT FILE 'ATMOSPH.IN' ***************************************
```

Table 8.16. Input data for example 2 (input file 'GRID.IN').

```
*** BLOCK H: NODAL INFORMATION ************************************************
     NumNP      NumEl       IJ      NumBP     NObs
        66         32        2          4        0
  n  Code     x       z          h       Conc        Q        M    B     Axz   Bxz   Dxz
  1   -4    0.00   230.00     -55.00   0.00E+00  0.00E+00   1   0.00  1.00  1.00  1.00
  2   -4    1.00   230.00     -55.00   0.00E+00  0.00E+00   1   0.00  1.00  1.00  1.00
  3    0    0.00   229.00     -54.00   0.00E+00  0.00E+00   1   0.00  1.00  1.00  1.00
  4    0    1.00   229.00     -54.00   0.00E+00  0.00E+00   1   0.00  1.00  1.00  1.00
  5    0    0.00   228.00     -53.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
  6    0    1.00   228.00     -53.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
  7    0    0.00   226.00     -51.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
  8    0    1.00   226.00     -51.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
  9    0    0.00   224.00     -49.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
 10    0    1.00   224.00     -49.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
 11    0    0.00   220.00     -45.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
 12    0    1.00   220.00     -45.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
 13    0    0.00   215.00     -40.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
 14    0    1.00   215.00     -40.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
 15    0    0.00   210.00     -35.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
 16    0    1.00   210.00     -35.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
 17    0    0.00   205.00     -30.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
 18    0    1.00   205.00     -30.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
 19    0    0.00   200.00     -25.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
 20    0    1.00   200.00     -25.00   0.00E+00  0.00E+00   1   1.00  1.00  1.00  1.00
       .      .      .          .         .         .        .     .     .     .
       .      .      .          .         .         .        .     .     .     .
       .      .      .          .         .         .        .     .     .     .
 61    0    0.00     2.00     173.00   0.00E+00  0.00E+00   2   0.00  1.00  1.00  1.00
 62    0    1.00     2.00     173.00   0.00E+00  0.00E+00   2   0.00  1.00  1.00  1.00
 63    0    0.00     1.00     174.00   0.00E+00  0.00E+00   2   0.00  1.00  1.00  1.00
 64    0    1.00     1.00     174.00   0.00E+00  0.00E+00   2   0.00  1.00  1.00  1.00
 65   -3    0.00     0.00     175.00   0.00E+00  0.00E+00   2   0.00  1.00  1.00  1.00
 66   -3    1.00     0.00     175.00   0.00E+00  0.00E+00   2   0.00  1.00  1.00  1.00
*** BLOCK I: ELEMENT INFORMATION **********************************************
  e   i   j   k   l   Angle   Aniz1   Aniz2   LayNum
  1   1   3   4   2   0.00    1.00    1.00    1
  2   3   5   6   4   0.00    1.00    1.00    1
  3   5   7   8   6   0.00    1.00    1.00    1
  4   7   9  10   8   0.00    1.00    1.00    1
  5   9  11  12  10   0.00    1.00    1.00    1
  6  11  13  14  12   0.00    1.00    1.00    1
  7  13  15  16  14   0.00    1.00    1.00    1
  8  15  17  18  16   0.00    1.00    1.00    1
  9  17  19  20  18   0.00    1.00    1.00    1
 10  19  21  22  20   0.00    1.00    1.00    2
      .   .   .   .      .       .       .
      .   .   .   .      .       .       .
      .   .   .   .      .       .       .
 25  49  51  52  50   0.00    1.00    1.00    2
 26  51  53  54  52   0.00    1.00    1.00    2
 27  53  55  56  54   0.00    1.00    1.00    2
 28  55  57  58  56   0.00    1.00    1.00    2
 29  57  59  60  58   0.00    1.00    1.00    2
 30  59  61  62  60   0.00    1.00    1.00    2
 31  61  63  64  62   0.00    1.00    1.00    2
 32  63  65  66  64   0.00    1.00    1.00    2
*** BLOCK J: BOUNDARY GEOMETRY INFORMATION ************************************
 Node number array:
      1       2       65      66
 Width array:
    0.50    0.50    0.50    0.50
 Length:
    1.00
*** END OF INPUT FILE 'GRID.IN' ***********************************************
```

Table 8.17. Input data for example 3b (input file 'SELECTOR.IN').

```
*** BLOCK A: BASIC INFORMATION ***********************************************
Heading
'Example 3b - Comparison with the 2-D analytical solution'
LUnit  TUnit  MUnit  (indicated units are obligatory for all input data)
'm'    'days' '-'
Kat (0:horizontal plane, 1:axisymmetric vertical flow, 2:vertical plane)
  2
MaxIt   TolTh    TolH (max. number of iterations and precis. tolerances)
  20    .0001    .1
lWat    lChem    CheckF   ShortF   FluxF  AtmInF  SeepF  FreeD  DrainF
f       t        f        t        f      f       t      f      f
*** BLOCK B: MATERIAL INFORMATION ********************************************
NMat    NLay    hTab1   hTabN   NPar
  1      1      .001    200.     9
thr     ths     tha     thm     Alfa      n       Ks      Kk      thk
 .02    .30    -.02     .30     .0410     1.964   0.3     0.3     .30
*** BLOCK C: TIME INFORMATION ************************************************
dt      dtMin   dtMax   DMul    DMul2   MPL
1.0    .0001  100.     1.3     .33     3
TPrint(1),TPrint(2),...,TPrint(MPL)                  (print-time array)
 50 100 365
*** BLOCK E: SEEPAGE INFORMATION (only if SeepF =.true.) ***************
NSeep                                              (number of seepage faces)
  1
NSP(1),NSP(2),.......,NSP(NSeep)          (number of nodes in each s.f.)
  15
NP(i,1),NP(i,2),.....,NP(i,NSP(i))     (nodal number array of i-th s.f.)
        301     302     303     304     305     306     307     308
        309     310     311     312     313     314     315
*** BLOCK G: SOLUTE TRANSPORT INFORMATION *************************************
Epsi     lUpW    lArtD  PeCr
 0.5     f       f      10
Bulk.d.  Difus.   Disper.    Adsorp.   SinkL1    SinkS1    SinkL0    SinkS0
1500     0.0     1.0  0.5    0.0004   -0.01     -0.01      0.0       0.0
KodCB(1),KodCB(2),.....,KodCB(NumBP)
        1       1       1       1       1       1       1       2
        2       2       2       2       2       2       2
        2       2       2       2       2       2       2       2
        2       2       2       2       2       2       2
cTop    cBot
 1.     0.      0.      0.      0.      0.
tPulse
 366
*** END OF INPUT FILE 'SELECTOR.IN' ******************************************
```

Table 8.18. Input data for example 3 (input file 'GRID.IN').

```
*** BLOCK H: NODAL INFORMATION ***********************************************
      NumNP      NumEl       IJ      NumBP      NObs
        315        280       15         30         0
  n  Code     x        z          h       Conc        Q      M    B     Axz   Bxz   Dxz
  1    1    0.00     0.00       0.00  1.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
  2    1   10.00     0.00       0.00  1.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
  3    1   20.00     0.00       0.00  1.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
  4    1   30.00     0.00       0.00  1.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
  5    1   40.00     0.00       0.00  1.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
  6    1   45.00     0.00       0.00  1.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
  7    1   49.00     0.00       0.00  1.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
  8    1   51.00     0.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
  9    1   55.00     0.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
 10    1   60.00     0.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
 11    1   67.00     0.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
 12    1   75.00     0.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
 13    1   85.00     0.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
 14    1  100.00     0.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
 15    1  120.00     0.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
 16    0    0.00    -5.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
 17    0   10.00    -5.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
 18    0   20.00    -5.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
 19    0   30.00    -5.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
 20    0   40.00    -5.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
 21    0   45.00    -5.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
 22    0   49.00    -5.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
        .        .        .       .         .         .     .     .     .     .
        .        .        .       .         .         .     .     .     .     .
        .        .        .       .         .         .     .     .     .     .
310    2   60.00 -200.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
311    2   67.00 -200.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
312    2   75.00 -200.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
313    2   85.00 -200.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
314    2  100.00 -200.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
315    2  120.00 -200.00       0.00  0.00E+00  0.00E+00  1  0.00  1.00  1.00  1.00
*** BLOCK I: ELEMENT INFORMATION *********************************************
  e   i   j   k   l   Angle  Aniz1  Aniz2  LayNum
  1   1  16  17   2   0.00   1.00   1.00   1
  2   2  17  18   3   0.00   1.00   1.00   1
  3   3  18  19   4   0.00   1.00   1.00   1
  4   4  19  20   5   0.00   1.00   1.00   1
  5   5  20  21   6   0.00   1.00   1.00   1
  6   6  21  22   7   0.00   1.00   1.00   1
  7   7  22  23   8   0.00   1.00   1.00   1
      .   .   .   .     .      .      .
      .   .   .   .     .      .      .
277 296 311 312 297   0.00   1.00   1.00   1
278 297 312 313 298   0.00   1.00   1.00   1
279 298 313 314 299   0.00   1.00   1.00   1
280 299 314 315 300   0.00   1.00   1.00   1
*** BLOCK J: BOUNDARY GEOMETRY INFORMATION ************************************
 Node number array:
        1        2        3        4        5        6        7
        8        9       10       11       12       13       14
       15      301      302      303      304      305      306
      307      308      309      310      311      312      313
      314      315
 Width array:
     5.00    10.00    10.00    10.00     7.50     4.50     3.00
     3.00     4.50     6.00     7.50     9.00    12.50    17.50
    10.00     5.00    10.00    10.00    10.00     7.50     4.50
     3.00     3.00     4.50     6.00     7.50     9.00    12.50
    17.50    10.00
 Length:
     0.00
*** END OF INPUT FILE 'GRID.IN' **********************************************
```

Table 8.19. Input data for example 4 (input file 'SELECTOR.IN').

```
*** BLOCK A: BASIC INFORMATION ***************************************
Heading
'Example 4 - Infiltration Test '
LUnit  TUnit  MUnit  (indicated units are obligatory for all input data)
'cm'   'min'  '-'
Kat (0:horizontal plane, 1:axisymmetric vertical flow, 2:vertical plane)
1
MaxIt   TolTh   TolH  (max. number of iterations and precis. tolerances)
20      .0001   .1
lWAt    lChem   CheckF  ShortF  FluxF   AtmInF  SeepF   FreeD  DrainF
t       t       f       f       t       f       f       f      f
*** BLOCK B: MATERIAL INFORMATION ************************************
NMat    NLay    hTab1   hTabN   NPar
  2      2      .001    200.    9
thr     ths     tha     thm     Alfa    n       Ks      Kk      thk
.0001   .399    .0001   .399    .0174   1.3757  .0207   .0207   .399
.0001   .339    .0001   .339    .0139   1.6024  .0315   .0315   .339
*** BLOCK C: TIME INFORMATION ****************************************
dt      dtMin   dtMax   DMul    DMul2   MPL
.1      .001    100.    1.33    .33     10
TPrint(1),TPrint(2),...,TPrint(MPL)                  (print-time array)
 60 180 360 720 1440 2160 2880 4320 5760 7200
*** BLOCK G: SOLUTE TRANSPORT INFORMATION *****************************
Epsi    lUpW    lArtD   PeCr
0.5      t      f       10.
Bulk.d. Difus.     Disper.    Adsorp.   SinkL1   SinkS1   SinkL0 SinkS0
1.4     0.026    0.50  0.10   0.100   -3.472E-5 -6.9444E-6   0.     0.
1.4     0.026    0.50  0.10   0.100   -3.472E-5 -6.9444E-6   0.     0.
KodCB(1),KodCB(2),.....,KodCB(NumBP)
1       1       1       1       1
cTop    cBot
1.      0.      0.      0.      0.      0.
tPulse
7200
*** END OF INPUT FILE 'SELECTOR.IN' **********************************
```

Table 8.20. Input data for example 4 (input file 'GRID.IN').

```
*** BLOCK H: NODAL INFORMATION ******************************************************
     NumNP      NumEl       IJ       NumBP       NObs
      380        342        19         5          0
   n  Code     x       z          h       Conc         Q       M   B     Axz   Bxz   Dxz
   1   1     0.00   230.00      0.00   1.00E+00   0.00E+00   1  0.00  1.00  1.00  1.00
   2   0     0.00   228.00   -145.50   0.00E+00   0.00E+00   1  0.00  1.00  1.00  1.00
   3   0     0.00   226.00   -143.40   0.00E+00   0.00E+00   1  0.00  1.00  1.00  1.00
   4   0     0.00   224.00   -141.00   0.00E+00   0.00E+00   1  0.00  1.00  1.00  1.00
   5   0     0.00   220.00   -135.60   0.00E+00   0.00E+00   1  0.00  1.00  1.00  1.00
   6   0     0.00   215.00   -127.70   0.00E+00   0.00E+00   1  0.00  1.00  1.00  1.00
   7   0     0.00   210.00   -119.00   0.00E+00   0.00E+00   1  0.00  1.00  1.00  1.00
   8   0     0.00   205.00   -109.90   0.00E+00   0.00E+00   1  0.00  1.00  1.00  1.00
   9   0     0.00   200.00   -100.50   0.00E+00   0.00E+00   1  0.00  1.00  1.00  1.00
  10   0     0.00   190.00    -82.80   0.00E+00   0.00E+00   1  0.00  1.00  1.00  1.00
  11   0     0.00   180.00    -71.00   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
  12   0     0.00   170.00    -60.30   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
  13   0     0.00   160.00    -49.80   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
  14   0     0.00   150.00    -39.60   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
  15   0     0.00   140.00    -29.50   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
  16   0     0.00   130.00    -19.40   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
  17   0     0.00   120.00     -9.40   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
  18   0     0.00   110.00      0.60   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
  19   0     0.00   100.00     10.20   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
  20   1     5.00   230.00      0.00   1.00E+00   0.00E+00   1  0.00  1.00  1.00  1.00
  21   0     5.00   228.00   -145.50   0.00E+00   0.00E+00   1  0.00  1.00  1.00  1.00
       .      .        .         .          .          .      .     .     .     .
       .      .        .         .          .          .      .     .     .     .
       .      .        .         .          .          .      .     .     .     .
 370   0   125.00   200.00   -100.50   0.00E+00   0.00E+00   1  0.00  1.00  1.00  1.00
 371   0   125.00   190.00    -82.80   0.00E+00   0.00E+00   1  0.00  1.00  1.00  1.00
 372   0   125.00   180.00    -71.00   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
 373   0   125.00   170.00    -60.30   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
 374   0   125.00   160.00    -49.80   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
 375   0   125.00   150.00    -39.60   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
 376   0   125.00   140.00    -29.50   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
 377   0   125.00   130.00    -19.40   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
 378   0   125.00   120.00     -9.40   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
 379   0   125.00   110.00      0.60   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
 380   0   125.00   100.00     10.20   0.00E+00   0.00E+00   2  0.00  1.00  1.00  1.00
*** BLOCK I: ELEMENT INFORMATION ***************************************************
   e   i   j   k   l   Angle  Aniz1  Aniz2  LayNum
   1   2  21  20   1   0.00   1.00   1.00   1
   2   3  22  21   2   0.00   1.00   1.00   1
   3   4  23  22   3   0.00   1.00   1.00   1
   4   5  24  23   4   0.00   1.00   1.00   1
   5   6  25  24   5   0.00   1.00   1.00   1
   6   7  26  25   6   0.00   1.00   1.00   1
   7   8  27  26   7   0.00   1.00   1.00   1
   8   9  28  27   8   0.00   1.00   1.00   1
   9  10  29  28   9   0.00   1.00   1.00   1
  10  11  30  29  10   0.00   1.00   1.00   2
       .       .       .       .       .
       .       .       .       .       .
 336 355 374 373 354   0.00   1.00   1.00   2
 337 356 375 374 355   0.00   1.00   1.00   2
 338 357 376 375 356   0.00   1.00   1.00   2
 339 358 377 376 357   0.00   1.00   1.00   2
 340 359 378 377 358   0.00   1.00   1.00   2
 341 360 379 378 359   0.00   1.00   1.00   2
 342 361 380 379 360   0.00   1.00   1.00   2
*** BLOCK J: BOUNDARY GEOMETRY INFORMATION ****************************************
 Node number array:
       1        20        39        58        77
 Width array:
    26.18    157.08    273.32    300.55    261.80
 Length:
     0.00
*** END OF INPUT FILE 'GRID.IN' ****************************************************
```

9. OUTPUT DATA

The program output consists of 16 output files which are organized into 3 groups:

T-level information
- H_MEAN.OUT
- V_MEAN.OUT
- CUM_Q.OUT
- RUN_INF.OUT
- SOLUTE.OUT
- OBSNOD.OUT

P-level information
- H.OUT
- TH.OUT
- CONC.OUT
- Q.OUT
- VX.OUT
- VZ.OUT
- BOUNDARY.OUT
- BALANCE.OUT

A-level information
- A_LEVEL.OUT

In addition, some of the input data are printed to file CHECK.OUT. All output files are directed to subdirectory SWMS_2D.OUT, which must be created by the user prior to program execution. The various output files are described in detail in Section 9.1. Section 9.2 lists selected output files for examples 1 through 4 (see Section 7). The input files for these examples were discussed in Section 8.2.

9.1. *Description of Data Output Files*

The file CHECK.OUT contains a complete description of the finite element mesh, the boundary code of each node, and the hydraulic and transport properties of each soil material. Finite element mesh data are printed only when the logical variable *CheckF* in input Block A (Table 9.1) is set equal to .true..

T-level information - This group of output files contains information which is printed at the end of each time step. Printing can be suppressed by setting the logical variable *ShortF* in input Block A equal to .true.; the information is then printed only at selected print times. Output files printed at the T-level are described in Tables 9.1 through 9.5. Output file OBSNOD.OUT contains information about the transient changes in pressure head, water content, and solute concentration at specified observation nodes.

P-level information - P-level information is printed only at prescribed print times. The following output files are printed at the P-level:

H.OUT	Nodal values of the pressure head
TH.OUT	Nodal values of the water content
CONC.OUT	Nodal values of the concentration
Q.OUT	Discharge/recharge rates assigned to boundary or internal sink/ source nodes
VX.OUT	Nodal values of the x-components of the Darcian flux vector
VZ.OUT	Nodal values of the z-components of the Darcian flux vector
BOUNDARY.OUT	This file contains information about each boundary node, n, for which $Kode(n) \neq 0$, including the discharge/recharge rate, $Q(n)$, the boundary flux, $q(n)$, the pressure head $h(n)$, the water content $\theta(n)$, and the concentration $Conc(n)$.
BALANCE.OUT	This file gives the total amount of water and solute inside each specified subregion, the inflow/outflow rates to/from that subregion, together with the mean pressure head (*hMean*) and the mean concentration (*cMean*) over each subregion (see Table 9.6). Absolute and relative errors in the water and solute mass balances are also printed to this file.

The output files H.OUT, TH.OUT, CONC.OUT, Q.OUT, VZ.OUT and VX.OUT provide printed tables of the specific variables. To better identify the output, each printed line starts with the nodal number and spatial coordinates of the first node on that line for which information is printed. Users can easily reprogram the original subroutines to restructure the output for their specific needs.

A-level information - A-level information is printed each time a time-dependent boundary condition is specified. The information is directed to output file A_LEVEL.OUT (Table 9.7).

Table 9.1. H_MEAN.OUT - mean pressure heads.

hAtm	Mean value of the pressure head calculated over a set of nodes for which $Kode(n) = \pm 4$ (i.e., along part of a boundary controlled by atmospheric conditions) [L].
hRoot	Mean value of the pressure head over a region for which $Beta(n) > 0$ (i.e., within the root zone) [L].
hKode3	Mean value of the pressure head calculated over a set of nodes for which $Kode(n) = \pm 3$ (i.e., along part of a boundary where the groundwater level, the bottom flux, or other time-dependent pressure head and/or flux is imposed) [L].
hKode1	Mean value of the pressure head calculated over a set of nodes for which $Kode(n) = \pm 1$ (i.e., along part of a boundary where time-independent pressure heads and/or fluxes are imposed) [L].
hSeep	Mean value of the pressure head calculated over a set of nodes for which $Kode(n) = \pm 2$ (i.e., along seepage faces) [L].
hKode5	Mean value of the pressure head calculated over a set of nodes for which $Kode(n) = \pm 5$ [L].
.	.
.	.
.	.
hKodeN	Mean value of the pressure head calculated over a set of nodes for which $Kode(n) = \pm NumKD$ [L].

Table 9.2. V_MEAN.OUT - mean and total water fluxes.

rAtm	Potential surface flux per unit atmospheric boundary ($Kode(n) = \pm 4$) [LT^{-1}].
rRoot	Potential transpiration rate, T_p [LT^{-1}].
vAtm	Mean value of actual surface flux per unit atmospheric boundary ($Kode(n) = \pm 4$) [LT^{-1}].
vRoot	Actual transpiration rate, T_a [LT^{-1}].
vKode3	Total value of the bottom or other flux across part of a boundary where the groundwater level, the bottom flux, or other time-dependent pressure head and/or flux is imposed ($Kode(n) = \pm 3$), [L^2T^{-1}] or [L^3T^{-1}]†.
vKode1	Total value of the boundary flux accros part of a boundary where time-independent pressure heads and/or fluxes are imposed, including internal sinks/sources ($Kode(n) = \pm 1$), [L^2T^{-1}] or [L^3T^{-1}]†.
vSeep	Total value of the boundary flux across a potential seepage face ($Kode(n) = \pm 2$), [L^2T^{-1}] or [L^3T^{-1}]†.
vKode5	Total value of the flux across a boundary containing nodes for which $Kode(n) = \pm 5$, [L^2T^{-1}] or [L^3T^{-1}]†.
.	.
.	.
.	.
vKodeN	Total value of the flux across a boundary containing nodes for which $Kode(n) = \pm NumKD$, [L^2T^{-1}] or [L^3T^{-1}]†.

† for plane and axisymmetric flow, respectively

Table 9.3. CUM_Q.OUT - total cumulative water fluxes.

CumQAP	Cumulative total potential surface flux across the atmospheric boundary (*Kode(n)* = ±4), [L^2] or [L^3]†.
CumQRP	Cumulative total potential transpiration rate, [L^2] or [L^3]†.
CumQA	Cumulative total actual surface flux across the atmospheric boundary (*Kode(n)* = ±4), [L^2] or [L^3]†.
CumQR	Cumulative total actual transpiration rate, [L^2] or [L^3]†.
CumQ3	Cumulative total value of the bottom or other boundary flux across part of a boundary where the groundwater level, the bottom flux, or other time-dependent pressure head and/or flux is imposed (*Kode(n)* = ±3), [L^2] or [L^3]†.
CumQ1	Cumulative total value of the flux across part of a boundary along which time-independent pressure heads and/or fluxes are imposed, including internal sinks/sources (*Kode(n)* = ±1), [L^2] or [L^3]†.
CumQS	Cumulative total value of the flux across a potential seepage faces (*Kode(n)* = ±2), [L^2] or [L^3]†.
CumQ5	Cumulative total value of the flux across a boundary containing nodes for which *Kode(n)* = ±5, [L^2] or [L^3]†.
.	.
.	.
.	.
CumQN	Cumulative total value of the flux across a boundary containing nodes for which *Kode(n)* = ±*NumKD*, [L^2] or [L^3]†.

† for plane and axisymmetric flow, respectively

Table 9.4. RUN_INF.OUT - time and iteration information.

TLevel	Time-level (current time-step number) [-].
Time	Time, t, at current time-level [T].
dt	Time step, Δt [T].
Iter	Number of iterations [-].
ItCum	Cumulative number of iterations [-].
Peclet	Maximum local Peclet number [-].
Courant	Maximum local Courant number [-].

Table 9.5. SOLUTE.OUT - actual and cumulative concentration fluxes.

*CumCh*0	Cumulative amount of solute removed from the flow region by zero-order reactions (positive when removed from the system), [ML^{-1}] or [M]†.
*CumCh*1	Cumulative amount of solute removed from the flow region by first-order reactions, [ML^{-1}] or [M]†.
CumChR	Cumulative amount of solute removed from the flow region by root water uptake S, [ML^{-1}] or [M]†.
*ChemS*1	Cumulative solute flux across part of a boundary along which time-independent pressure heads and/or fluxes are imposed, including internal sink/sources ($Kode(n) = \pm 1$), [ML^{-1}] or [M]†.
*ChemS*2	Cumulative solute flux across a potential seepage faces ($Kode(n) = \pm 2$), [ML^{-1}] or [M]†.
*ChemS*3	Cumulative solute flux across part of a boundary along which the groundwater level, the bottom flux, or other time-dependent pressure head and/or flux is imposed ($Kode(n) = \pm 3$), [ML^{-1}] or [M]†.
*ChemS*4	Cumulative total solute flux across the atmospheric boundary ($Kode(n) = \pm 4$), [ML^{-1}] or [M]†.
*ChemS*5	Cumulative total solute flux across an internal or external boundary containing nodes for which $Kode(n) = \pm 5$, [ML^{-1}] or [M]†.
.	.
.	.
.	.
ChemSN	Cumulative total solute flux across an internal or external boundary containing nodes for which $Kode(n) = \pm NumKD$, [ML^{-1}] or [M]†.
*qc*1	Total solute flux across part of a boundary along which time-independent pressure heads and/or fluxes are imposed ($Kode(n) = \pm 1$), [$ML^{-1}T^{-1}$] or [MT^{-1}]†.
*qc*2	Total solute flux across a potential seepage face ($Kode(n) = \pm 2$), [$ML^{-1}T^{-1}$] or [MT^{-1}]†.
*qc*3	Total solute flux calculated across a boundary containing nodes for which $Kode(n) = \pm 3$ (i.e., along part of a boundary where the groundwater level, the bottom flux, or other time-dependent pressure head and/or flux is specified), [$ML^{-1}T^{-1}$] or [MT^{-1}]†.
*qc*4	Total solute flux across the atmospheric boundary ($Kode(n) = \pm 4$), [$ML^{-1}T^{-1}$] or [MT^{-1}]†.
*qc*5	Total solute flux across an internal or external boundary containing nodes for which $Kode(n) = \pm 5$, [$ML^{-1}T^{-1}$] or [MT^{-1}]†.
.	.
.	.
.	.
qcN	Total solute flux across an internal or external boundary containing nodes for which $Kode(n) = \pm NumKD$, [$ML^{-1}T^{-1}$] or [MT^{-1}]†.

† for plane and axisymmetric flow, respectively

Table 9.6. BALANCE.OUT - mass balance variables.

Area	Area of the entire flow domain or a specified subregion, $[L^2]$ or $[L^3]^{\dagger}$.
Volume	Volume of water in the entire flow domain or a specified subregion, $[L^2]$ or $[L^3]^{\dagger}$.
InFlow	Inflow/Outflow to/from the entire flow domain or a specified subregion, $[L^2T^{-1}]$ or $[L^3T^{-1}]^{\dagger}$.
hMean	Mean pressure head in the entire flow domain or a specified subregion [L].
ConcVol	Amount of solute in the entire flow domain or a specified subregion, $[ML^{-1}]$ or $[M]^{\dagger}$.
cMean	Mean concentration in the entire flow domain or a specified subregion $[ML^{-3}]$.
WatBalT	Absolute error in the water mass balance of the entire flow domain, $[L^2]$ or $[L^3]^{\dagger}$.
WatBalR	Relative error in the water mass balance of the entire flow domain [%].
CncBalT	Absolute error in the solute mass balance of the entire flow domain, $[ML^{-1}]$ or $[M]^{\dagger}$.
CncBalR	Relative error in the solute mass balance of the entire flow domain [%].

† for plane and axisymmetric flow, respectively

Table 9.7. A_LEVEL.OUT - mean pressure heads and total cumulative fluxes.

CumQAP	Cumulative total potential flux across the atmospheric boundary (*Kode(n)* = ±4), [L^2] or [L^3]†.
CumQRP	Cumulative total potential transpiration rate, [L^2] or [L^3]†.
CumQA	Cumulative total actual flux across the atmospheric boundary (*Kode(n)* = ±4), [L^2] or [L^3]†.
CumQR	Cumulative total actual transpiration rate, [L^2] or [L^3]†.
CumQ3	Cumulative total bottom or other flux across a boundary along which the groundwater level, the bottom flux, or other time-dependent pressure head and/or flux is imposed (*Kode(n)* = ±3), [L^2] or [L^3]†.
hAtm	Mean value of the pressure head calculated over a set of nodes for which *Kode(n)* = ±4 [L].
hRoot	Mean value of the pressure head over a region for which *Beta(n)* > 0 (i.e., the root zone) [L].
hKode3	Mean value of the pressure head over a set of nodes for which *Kode(n)* = ±3 [L].

† for plane and axisymmetric flow, respectively

Table 9.8. Output data for example 1 (part of output file 'H.OUT').

Time *** 5400.0000 ***

n	x(n)	z(n)	h(n)	h(n+1) ...
1	.0	61.0	.8	.8
3	.0	60.8	.6	.6
5	.0	60.5	.4	.4
7	.0	60.3	.3	.3
9	.0	60.0	.1	.1
11	.0	59.5	-.2	-.2
13	.0	59.0	-.5	-.5
15	.0	58.0	-1.2	-1.2
17	.0	57.0	-1.8	-1.8
19	.0	56.0	-2.4	-2.4
21	.0	55.0	-3.1	-3.1
23	.0	54.0	-3.7	-3.7
25	.0	53.0	-4.4	-4.4
27	.0	52.0	-5.0	-5.0
29	.0	51.0	-5.7	-5.7
31	.0	50.0	-6.3	-6.3
33	.0	49.0	-7.0	-7.0
35	.0	48.0	-7.6	-7.6
37	.0	47.0	-8.2	-8.2
39	.0	46.0	-8.9	-8.9
41	.0	45.0	-9.5	-9.5
43	.0	44.0	-10.1	-10.1
45	.0	43.0	-10.7	-10.7
47	.0	42.0	-11.4	-11.4
49	.0	41.0	-12.0	-12.0
51	.0	40.0	-12.6	-12.6
53	.0	39.0	-13.1	-13.1
55	.0	38.0	-13.7	-13.7
57	.0	37.0	-14.3	-14.3
59	.0	36.0	-14.9	-14.9
61	.0	35.0	-15.4	-15.4
63	.0	34.0	-15.9	-15.9
65	.0	33.0	-16.5	-16.5
67	.0	32.0	-17.0	-17.0
69	.0	31.0	-17.5	-17.5
71	.0	30.0	-18.1	-18.1
73	.0	29.0	-18.8	-18.8
75	.0	28.0	-19.5	-19.5
77	.0	27.0	-20.3	-20.3
79	.0	26.0	-21.2	-21.2
81	.0	25.0	-22.4	-22.4
83	.0	24.0	-23.8	-23.8
85	.0	23.0	-25.5	-25.6
87	.0	22.0	-27.8	-27.8
89	.0	21.0	-30.7	-30.8
91	.0	20.0	-34.8	-34.9
93	.0	18.0	-48.8	-48.7
95	.0	16.0	-82.8	-81.0
97	.0	14.0	-138.8	-136.1
99	.0	12.0	-149.6	-149.5
101	.0	10.0	-150.0	-150.0
103	.0	8.0	-150.0	-150.0
105	.0	6.0	-149.9	-149.9
107	.0	4.0	-149.7	-149.7
109	.0	2.0	-149.0	-149.0
111	.0	.0	-147.4	-147.5

Table 9.9. Output data for example 1 (output file 'CUM_Q.OUT').

```
Example 1 - Column Test

Program SWMS_2D
Date:  28.10.    Time:   8:35:33
Time independent boundary conditions
Vertical plane flow, V = L*L
Units: L = cm    , T = sec   , M = -
All cumulative fluxes (CumQ) are positive out of the region
```

Time	CumQAP	CumQRP	CumQA	CumQR	CumQ3	CumQ1	CumQS	CumQ5	CumQ6
[T]	[V]	[V]	[V]	[V]	[V]	[V]	[V]	[V]	[V]
60.0000	.000E+00	.000E+00	.000E+00	.000E+00	.000E+00	-.797E+00	.000E+00	.000E+00	.000E+00
900.0000	.000E+00	.000E+00	.000E+00	.000E+00	.000E+00	-.340E+01	.000E+00	.000E+00	.000E+00
1800.0000	.000E+00	.000E+00	.000E+00	.000E+00	.000E+00	-.506E+01	.000E+00	.000E+00	.000E+00
2700.0000	.000E+00	.000E+00	.000E+00	.000E+00	.000E+00	-.644E+01	.000E+00	.000E+00	.000E+00
3600.0000	.000E+00	.000E+00	.000E+00	.000E+00	.000E+00	-.767E+01	.000E+00	.000E+00	.000E+00
5400.0000	.000E+00	.000E+00	.000E+00	.000E+00	.000E+00	-.991E+01	.000E+00	.000E+00	.000E+00

Table 9.10. Output data for example 2 (output file 'RUN_INF.OUT').

TLevel	Time	dt	Iter	ItCum
70	.120E+03	.500E+00	2	166
132	.151E+03	.500E+00	3	335
192	.181E+03	.500E+00	3	522
254	.212E+03	.500E+00	3	673
328	.243E+03	.500E+00	2	912
388	.273E+03	.500E+00	6	1067
Real time [sec]		16.000000000000000		

Table 9.11. Output data for example 2 (part of output file 'A_LEVEL.OUT').

```
Example 2 - Grass Field Problem (Hupselse Beek 1982)

Program SWMS_2D
Date:  28.10.    Time:  14: 1:23
Time dependent boundary conditions
Vertical plane flow, V = L*L
Units: L = cm    , T = day   , M = -
All cumulative fluxes (CumQ) are positive out of the region
```

Time [T]	CumQAP [V]	CumQRP [V]	CumQA [V]	CumQR [V]	CumQ3 [V]	hAtm [L]	hRoot [L]	hKode3 [L]	A-level
91.0000	.000E+00	.160E+00	.000E+00	.160E+00	.373E-01	-58.2	-37.1	171.9	1
92.0000	-.700E-01	.340E+00	-.700E-01	.340E+00	.722E-01	-59.5	-39.0	169.4	2
93.0000	-.900E-01	.470E+00	-.900E-01	.470E+00	.105E+00	-62.3	-40.9	167.4	3
94.0000	-.900E-01	.670E+00	-.900E-01	.670E+00	.136E+00	-66.3	-43.8	164.3	4
95.0000	-.900E-01	.950E+00	-.900E-01	.950E+00	.164E+00	-71.2	-47.9	160.3	5
96.0000	-.160E+00	.113E+01	-.160E+00	.113E+01	.190E+00	-71.2	-50.8	158.0	6
97.0000	-.450E+00	.121E+01	-.450E+00	.121E+01	.215E+00	-63.6	-49.7	159.0	7
98.0000	-.890E+00	.135E+01	-.890E+00	.135E+01	.240E+00	-57.8	-46.1	162.1	8
99.0000	-.109E+01	.146E+01	-.109E+01	.146E+01	.268E+00	-60.9	-44.1	164.0	9
100.0000	-.138E+01	.157E+01	-.138E+01	.157E+01	.298E+00	-57.8	-42.6	166.0	10
101.0000	-.170E+01	.168E+01	-.170E+01	.168E+01	.329E+00	-55.1	-40.2	168.6	11
102.0000	-.219E+01	.179E+01	-.219E+01	.179E+01	.362E+00	-48.8	-36.2	173.7	12
103.0000	-.220E+01	.195E+01	-.220E+01	.195E+01	.400E+00	-57.2	-35.3	172.4	13
104.0000	-.220E+01	.212E+01	-.220E+01	.212E+01	.435E+00	-60.8	-38.6	169.4	14
105.0000	-.220E+01	.234E+01	-.220E+01	.234E+01	.468E+00	-64.8	-42.1	165.8	15
106.0000	-.220E+01	.255E+01	-.220E+01	.255E+01	.497E+00	-68.4	-45.8	162.5	16
107.0000	-.220E+01	.278E+01	-.220E+01	.278E+01	.524E+00	-72.3	-49.4	159.1	17
108.0000	-.220E+01	.301E+01	-.220E+01	.301E+01	.549E+00	-76.0	-53.0	155.9	18
109.0000	-.220E+01	.325E+01	-.220E+01	.325E+01	.572E+00	-79.7	-56.5	152.7	19
110.0000	-.220E+01	.343E+01	-.220E+01	.343E+01	.593E+00	-82.1	-59.5	150.1	20
111.0000	-.220E+01	.358E+01	-.220E+01	.358E+01	.613E+00	-84.1	-61.7	148.0	21
112.0000	-.220E+01	.377E+01	-.220E+01	.377E+01	.631E+00	-87.0	-64.2	145.7	22
113.0000	-.221E+01	.392E+01	-.221E+01	.392E+01	.649E+00	-88.4	-66.3	143.7	23
114.0000	-.222E+01	.414E+01	-.222E+01	.414E+01	.665E+00	-91.7	-68.8	141.5	24
115.0000	-.222E+01	.437E+01	-.222E+01	.437E+01	.681E+00	-95.6	-71.8	139.0	25
116.0000	-.224E+01	.457E+01	-.224E+01	.457E+01	.695E+00	-97.0	-74.4	136.8	26
117.0000	-.224E+01	.474E+01	-.224E+01	.474E+01	.709E+00	-99.6	-76.4	134.9	27
118.0000	-.226E+01	.488E+01	-.226E+01	.488E+01	.722E+00	-99.7	-77.9	133.2	28
119.0000	-.252E+01	.501E+01	-.252E+01	.501E+01	.735E+00	-86.9	-76.1	132.6	29
120.0000	-.276E+01	.512E+01	-.276E+01	.512E+01	.747E+00	-84.2	-73.1	133.2	30
121.0000	-.337E+01	.520E+01	-.337E+01	.520E+01	.760E+00	-68.2	-67.1	135.9	31
122.0000	-.337E+01	.541E+01	-.337E+01	.541E+01	.774E+00	-89.2	-67.4	137.2	32
123.0000	-.356E+01	.555E+01	-.356E+01	.555E+01	.788E+00	-84.3	-70.1	137.4	33
124.0000	-.373E+01	.576E+01	-.373E+01	.576E+01	.802E+00	-85.9	-70.3	137.1	34
125.0000	-.451E+01	.583E+01	-.451E+01	.583E+01	.816E+00	-62.0	-63.5	140.5	35
126.0000	-.569E+01	.593E+01	-.569E+01	.593E+01	.833E+00	-48.2	-51.0	151.4	36
127.0000	-.637E+01	.603E+01	-.637E+01	.603E+01	.855E+00	-51.3	-43.6	162.7	37
128.0000	-.637E+01	.619E+01	-.637E+01	.619E+01	.884E+00	-65.3	-43.4	163.8	38
129.0000	-.637E+01	.645E+01	-.637E+01	.645E+01	.912E+00	-70.7	-47.5	160.5	39
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
262.0000	-.229E+02	.425E+02	-.229E+02	.425E+02	.149E+01	-361.6	-287.1	16.7	172
263.0000	-.233E+02	.427E+02	-.233E+02	.427E+02	.149E+01	-216.3	-278.6	16.0	173
264.0000	-.238E+02	.429E+02	-.238E+02	.429E+02	.149E+01	-156.9	-246.6	15.3	174
265.0000	-.238E+02	.431E+02	-.238E+02	.431E+02	.149E+01	-228.3	-229.7	14.6	175
266.0000	-.238E+02	.433E+02	-.238E+02	.433E+02	.149E+01	-257.4	-237.0	13.9	176
267.0000	-.238E+02	.435E+02	-.238E+02	.434E+02	.149E+01	-276.8	-245.1	13.2	177
268.0000	-.238E+02	.437E+02	-.238E+02	.436E+02	.149E+01	-293.1	-252.9	12.6	178
269.0000	-.243E+02	.438E+02	-.243E+02	.437E+02	.149E+01	-157.4	-236.4	12.0	179
270.0000	-.244E+02	.440E+02	-.244E+02	.439E+02	.149E+01	-206.6	-219.2	11.4	180
271.0000	-.244E+02	.442E+02	-.244E+02	.441E+02	.149E+01	-240.3	-225.3	10.8	181
272.0000	-.244E+02	.444E+02	-.244E+02	.443E+02	.149E+01	-264.1	-234.5	10.3	182
273.0000	-.254E+02	.444E+02	-.254E+02	.443E+02	.149E+01	-103.1	-203.9	9.8	183

Table 9.12. Output data for example 3b (part of output file 'SOLUTE.OUT').

All solute fluxes (SMean) and cumulative solute fluxes (ChemS) are positive out of the region

Time [T]	CumCh0 [VM/L3]	CumCh1 [VM/L3]	CumChR [VM/L3]	ChemS(i),i=1,NumKD [VM/L3]					
50.00	.000E+00	.170E+03	.000E+00	-.811E+03	.291E-14	.000E+00	.000E+00	.000E+00	.000E+00
100.00	.000E+00	.561E+03	.000E+00	-.158E+04	.245E-09	.000E+00	.000E+00	.000E+00	.000E+00
365.00	.000E+00	.410E+04	.000E+00	-.568E+04	.498E-01	.000E+00	.000E+00	.000E+00	.000E+00

Table 9.13. Output data for example 3b (output file 'BALANCE.OUT').

```
Example 3b - Comparison with the 2-D analytical
             solution

Program SWMS_2D
Date:  28.10.    Time:  14: 2:59
Time independent boundary conditions
Vertical plane flow, V = L*L
Units: L = m    , T = days , M = -

Time [T]              Total      Sub-region number

      .0000                          1
Area     [V]          .240E+05    .240E+05
Volume   [V]          .720E+04    .720E+04
InFlow   [V/T]        .000E+00    .000E+00
hMean    [L]          .000E+00          .0
ConcVol  [VM/L3]      .000E+00    .000E+00
cMean    [M/L3]       .000E+00    .000E+00

    50.0000                          1
Area     [V]          .240E+05    .240E+05
Volume   [V]          .720E+04    .720E+04
InFlow   [V/T]        .000E+00    .000E+00
hMean    [L]          .000E+00          .0
ConcVol  [VM/L3]      .633E+03    .633E+03
cMean    [M/L3]       .293E-01    .293E-01
CncBalT  [VM/L3]     -.767E+01
CncBalR  [%]              .782

   100.0000                          1
Area     [V]          .240E+05    .240E+05
ConcVol  [VM/L3]      .992E+03    .992E+03
cMean    [M/L3]       .459E-01    .459E-01
CncBalT  [VM/L3]     -.303E+02
CncBalR  [%]             1.411

   365.0000                          1
Area     [V]          .240E+05    .240E+05
ConcVol  [VM/L3]      .151E+04    .151E+04
cMean    [M/L3]       .697E-01    .697E-01
CncBalT  [VM/L3]     -.680E+02
CncBalR  [%]              .695
```

TABLE 9.14. Output data for example 3b (part of output file 'CONC.OUT'†).

Time *** 365.0000 ***

n	x(n)	z(n)	Conc(n)		Conc(n+1)		...										
1	0.0	0.0	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
16	0.0	-5.0	0.864	0.864	0.864	0.864	0.867	0.847	0.650	0.228	0.000	0.000	0.000	0.000	0.000	0.000	0.000
31	0.0	-10.0	0.747	0.747	0.747	0.748	0.750	0.704	0.492	0.267	0.024	0.000	0.000	0.000	0.000	0.000	0.000
46	0.0	-15.0	0.646	0.646	0.645	0.647	0.646	0.585	0.399	0.254	0.048	0.000	0.000	0.000	0.000	0.000	0.000
61	0.0	-20.0	0.559	0.558	0.558	0.560	0.554	0.488	0.334	0.229	0.062	0.002	0.000	0.000	0.000	0.000	0.000
76	0.0	-25.0	0.483	0.482	0.482	0.485	0.475	0.409	0.282	0.203	0.068	0.005	0.000	0.000	0.000	0.000	0.000
91	0.0	-30.0	0.417	0.417	0.417	0.419	0.406	0.345	0.241	0.178	0.069	0.009	0.000	0.000	0.000	0.000	0.000
106	0.0	-35.0	0.361	0.360	0.360	0.363	0.348	0.291	0.206	0.156	0.067	0.011	0.000	0.000	0.000	0.000	0.000
121	0.0	-40.0	0.312	0.311	0.312	0.314	0.297	0.247	0.176	0.136	0.063	0.013	0.000	0.000	0.000	0.000	0.000
136	0.0	-45.0	0.270	0.269	0.269	0.271	0.254	0.210	0.151	0.118	0.058	0.014	0.000	0.000	0.000	0.000	0.000
151	0.0	-50.0	0.233	0.232	0.233	0.234	0.217	0.179	0.130	0.103	0.053	0.015	0.000	0.000	0.000	0.000	0.000
166	0.0	-55.0	0.204	0.201	0.202	0.203	0.186	0.153	0.112	0.090	0.048	0.015	0.001	0.000	0.000	0.000	0.000
181	0.0	-60.0	0.170	0.172	0.173	0.173	0.157	0.128	0.095	0.077	0.043	0.015	0.001	0.000	0.000	0.000	0.000
196	0.0	-70.0	0.132	0.130	0.131	0.130	0.116	0.095	0.071	0.059	0.035	0.013	0.002	0.000	0.000	0.000	0.000
211	0.0	-80.0	0.095	0.097	0.096	0.095	0.084	0.069	0.052	0.043	0.027	0.011	0.002	0.000	0.000	0.000	0.000
226	0.0	-90.0	0.085	0.078	0.078	0.075	0.065	0.053	0.041	0.035	0.022	0.010	0.002	0.000	0.000	0.000	0.000
241	0.0	-100.0	0.043	0.048	0.046	0.044	0.040	0.032	0.025	0.021	0.014	0.007	0.002	0.000	0.000	0.000	0.000
256	0.0	-125.0	0.010	0.012	0.013	0.012	0.012	0.010	0.007	0.006	0.004	0.002	0.001	0.000	0.000	0.000	0.000
271	0.0	-150.0	0.002	0.002	0.003	0.003	0.003	0.002	0.002	0.001	0.001	0.001	0.000	0.000	0.000	0.000	0.000
286	0.0	-175.0	0.000	0.000	0.000	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
301	0.0	-200.0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

† This output file has different formatting than the original output file 'CONC.OUT'.

Table 9.15. Output data for example 4 (output file 'CUM_Q.OUT').

```
Program SWMS_2D
Time independent boundary conditions
Axisymmetric flow, V = L*L*L
Units: L = cm    , T = min   , M = -
All cumulative fluxes (CumQ) are positive out of the region
```

Time [T]	CumQAP [V]	CumQRP [V]	CumQA [V]	CumQR [V]	CumQ3 [V]	CumQ1 [V]	CumQS [V]	CumQ5 [V]	CumQ6.. [V]
60.0000	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	-0.297E+04	0.000E+00	0.000E+00	0.000E+00
180.0000	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	-0.679E+04	0.000E+00	0.000E+00	0.000E+00
360.0000	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	-0.121E+05	0.000E+00	0.000E+00	0.000E+00
720.0000	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	-0.222E+05	0.000E+00	0.000E+00	0.000E+00
1440.0000	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	-0.420E+05	0.000E+00	0.000E+00	0.000E+00
2160.0000	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	-0.617E+05	0.000E+00	0.000E+00	0.000E+00
2880.0000	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	-0.814E+05	0.000E+00	0.000E+00	0.000E+00
4320.0000	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	-0.121E+06	0.000E+00	0.000E+00	0.000E+00
5760.0000	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	-0.160E+06	0.000E+00	0.000E+00	0.000E+00
7200.0000	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	-0.199E+06	0.000E+00	0.000E+00	0.000E+00

Table 9.16. Output data for example 4 (part of output file 'BOUNDARY.OUT').

```
Example 4 - Infiltration Test

Program SWMS_2D
Date:  14.11.    Time:   9:20:10
Time independent boundary conditions
Axisymmetric flow, V = L*L*L
Units: L = cm   , T = min  , M = -

Time:     180.0000

   i    n     x      z     Code      Q          v          h        th       Conc
                                    [V/T]      [L/T]      [L]      [-]      [M/L3]

   1    1     .0   230.0     1   .533E+00  -.204E-01       .0     .399   .100E+01
   2   20    5.0   230.0     1   .317E+01  -.202E-01       .0     .399   .100E+01
   3   39   10.0   230.0     1   .584E+01  -.213E-01       .0     .399   .100E+01
   4   58   14.0   230.0     1   .759E+01  -.252E-01       .0     .399   .100E+01
   5   77   17.0   230.0     1   .131E+02  -.502E-01       .0     .399   .100E+01

Time:     360.0000

   i    n     x      z     Code      Q          v          h        th       Conc
                                    [V/T]      [L/T]      [L]      [-]      [M/L3]

   1    1     .0   230.0     1   .533E+00  -.203E-01       .0     .399   .100E+01
   2   20    5.0   230.0     1   .323E+01  -.206E-01       .0     .399   .100E+01
   3   39   10.0   230.0     1   .572E+01  -.209E-01       .0     .399   .100E+01
   4   58   14.0   230.0     1   .718E+01  -.239E-01       .0     .399   .100E+01
   5   77   17.0   230.0     1   .122E+02  -.467E-01       .0     .399   .100E+01

Time:     720.0000

   i    n     x      z     Code      Q          v          h        th       Conc
                                    [V/T]      [L/T]      [L]      [-]      [M/L3]

   1    1     .0   230.0     1   .535E+00  -.204E-01       .0     .399   .100E+01
   2   20    5.0   230.0     1   .316E+01  -.201E-01       .0     .399   .100E+01
   3   39   10.0   230.0     1   .557E+01  -.204E-01       .0     .399   .100E+01
   4   58   14.0   230.0     1   .692E+01  -.230E-01       .0     .399   .100E+01
   5   77   17.0   230.0     1   .115E+02  -.441E-01       .0     .399   .100E+01

Time:     1440.0000

   i    n     x      z     Code      Q          v          h        th       Conc
                                    [V/T]      [L/T]      [L]      [-]      [M/L3]

   1    1     .0   230.0     1   .535E+00  -.204E-01       .0     .399   .100E+01
   2   20    5.0   230.0     1   .317E+01  -.202E-01       .0     .399   .100E+01
   3   39   10.0   230.0     1   .556E+01  -.203E-01       .0     .399   .100E+01
   4   58   14.0   230.0     1   .685E+01  -.228E-01       .0     .399   .100E+01
   5   77   17.0   230.0     1   .113E+02  -.433E-01       .0     .399   .100E+01

Time:     7200.0000

   i    n     x      z     Code      Q          v          h        th       Conc
                                    [V/T]      [L/T]      [L]      [-]      [M/L3]

   1    1     .0   230.0     1   .531E+00  -.203E-01       .0     .399   .100E+01
   2   20    5.0   230.0     1   .316E+01  -.201E-01       .0     .399   .100E+01
   3   39   10.0   230.0     1   .554E+01  -.203E-01       .0     .399   .100E+01
   4   58   14.0   230.0     1   .677E+01  -.225E-01       .0     .399   .100E+01
   5   77   17.0   230.0     1   .111E+02  -.425E-01       .0     .399   .100E+01
```

10. PROGRAM ORGANIZATION AND LISTING

10.1. *Description of Program Units*

The program consists of a main program and 59 subprograms. The subprograms are organized by means of 8 source files which are stored and compiled separately and then linked together with the main program to form an executable program. Below are a list and brief descriptions of the source files and the associated subprograms.

SWMS_2D.FOR	(Main program unit)
INPUT2.FOR	BasInf, MatIn, GenMat, TmIn, SeepIn, NodInf, ElemIn, GeomIn, AtmIn, SinkIn, ChemIn, DrainIn
WATFLOW2.FOR	WatFlow, Reset, Dirich, Solve, Shift, SetMat, Veloc
TIME2.FOR	TmCont, SetAtm, Fgh
MATERIA2.FOR	FK, FC, FQ, FH
SINK2.FOR	SetSnk, FAlfa
OUTPUT2.FOR	TLInf, ALInf, hOut, thOut, QOut, FlxOut, SubReg, BouOut, cOut, SolInf, ObsNod
SOLUTE2.FOR	Solute, cBound, ChInit, Disper, SolveT, WeFact, PeCour
ORTHOFEM.FOR	IADMake, Insert, Find, ILU, DU, ORTHOMIN, LUSolv, MatM2, SDot, SDotK, SNRM, SAXPYK, SCopy, SCopyK

Main program unit SWMS_2D.FOR

This is the main program unit of SWMS_2D. This unit controls execution of the program and determines which optional subroutines are necessary for a particular application.

Source file INPUT.FOR

Subroutines included in this source file are designed to read data from different input blocks. The following table summarizes from which input file and input block (described in Section 8) a particular subroutine reads.

Table 10.1. Input subroutines/files.

Subroutine	Input Block	Input File
BasInf	A. Basic Information	
MatIn	B. Material Information	
TmIn	C. Time Information	
SinkIn	D. Sink Information	SELECTOR.IN
SeepIn	E. Seepage Information	
DrainIn	F. Drain Information	
ChemIn	G. Solute Transport Information	
NodInf	H. Nodal Information	
ElemIn	I. Element Information	GRID.IN
GeomIn	J. Boundary Geometry Information	
AtmIn	K. Atmospheric Information	ATMOSPH.IN

Subroutine **GenMat** generates for each soil type in the flow domain a table of water contents, hydraulic conductivities, and specific water capacities from the set of hydraulic parameters.

Source file WATFLOW2.FOR

Subroutine **WatFlow** is the main subroutine for simulating water flow; this subroutine controls the entire iterative procedure of solving the Richards equation.

Subroutine **Reset** constructs the global matrix equation for water flow, including the right-hand side vector.

Subroutine **Dirich** modifies the global matrix equation by incorporating prescribed pressure head nodes.

Subroutine **Solve** solves the banded symmetric matrix equation for water flow by Gaussian elimination.

Subroutine **Shift** changes atmospheric or seepage face boundary conditions from Dirichlet type to Neumann type conditions, or vice versa, as needed. Also updates boundary conditions for the variable boundary fluxes (free and deep drainage).

Subroutine **SetMat** determines the nodal values of the hydraulic properties $K(h)$, $C(h)$ and $\theta(h)$ by interpolation between intermediate values in the hydraulic property tables.

Subroutine **Veloc** calculates nodal water fluxes.

Source file TIME2.FOR

Subroutine **TmCont** adjusts the current value of the time increment Δt.

Subroutine **SetAtm** updates time-dependent boundary conditions.

Function **Fqh** describes the groundwater level - discharge relationship, $q(h)$, defined by equation (6.1). This function is called only from subroutine SetAtm.

Source file MATERIA2.FOR

This file includes the functions **FK, FC, FQ** and **FH** which define the unsaturated hydraulic properties $K(h)$, $C(h)$, $\theta(h)$, and $h(\theta)$, for each soil material.

Source file SINK2.FOR

This file includes subroutine **SetSnk** and function **FAlfa**. These subroutines calculate the actual root water extraction rate as a function of water stress in the soil root zone.

Source file OUTPUT2.FOR

The subroutines included in this file are designed to print data to different output files. Table 10.2 summarizes which output files are generated by a particular subroutine.

Table 10.2. Output subroutines/files.

Subroutine	Output File
TLInf	H_MEAN.OUT V_MEAN.OUT CUM_Q.OUT RUN_INF.OUT
SolInf	SOLUTE.OUT
hOut	H.OUT
thOut	TH.OUT
cOut	CONC.OUT
QOut	Q.OUT
FlxOut	VX.OUT VZ.OUT
BouOut	BOUNDARY.OUT
SubReg	BALANCE.OUT
ALInf	A_LEVEL.OUT
ObsNod	OBSNOD.OUT

Source file SOLUTE2.FOR

Subroutine **Solute** is the main subroutine for simulating solute transport; it constructs the global matrix equation for transport, including the right-hand side vector.

Subroutine **c_Bound** determines the values of the solute transport boundary codes, *cKod*(*n*), and incorporates prescribed boundary conditions in the global matrix equation for solute transport.

Subroutine **ChInit** initializes selected transport parameters at the beginning of the simulation.

Subroutine **Disper** calculates nodal values of the dispersion coefficients.

Subroutine **SolveT** solves the final asymmetric banded matrix equation for solute transport using Gaussian elimination.

Subroutine **WeFact** computes the optimum weighing factors for all sides of all elements.

Subroutine **PeCour** computes the maximum local Peclet and Courant numbers and the maximum permissible time step.

Source file ORTHOFEM.FOR

The subroutines included in this file solve large sparse systems of linear algebraic equations using the preconditioned conjugate gradient method for symmetric matrices, and the ORTHOMIN method for asymmetric matrices. The subroutines were adopted from *Mendoza et al.* [1991] (see *Mendoza et al.* [1991] for a detailed description of both methods).

Subroutine **IADMake** generates the adjacency matrix which determines nodal connections from the finite element incidence matrix.

Subroutine **Insert** adds node j to the adjacency list for node i.

Subroutine **Find** retrieves from the adjacency matrix the appropriate position of two global points in the coefficient matrix.

Subroutine **ILU** performs incomplete lower-upper decomposition of matrix $[A]$.

Function **DU** searches the ith row of the upper diagonal matrix for an adjacency of node j.

Subroutine **ORTHOMIN** governs the ORTHOMIN (conjugate gradient) acceleration.

Subroutine **LUSolv** performs lower diagonal matrix inversion by forward substitution, and upper diagonal matrix inversion by backward substitution.

Subroutine **MatM2** multiplies a matrix by a vector.

Function **SDot** calculates the dot product of two vectors.

Function **SDotK** calculates the dot product of a column in matrix by a vector.

Function **SNRM** computes the maximum norm of a vector.

Subroutine **SAXPYK** multiplies a column in a matrix by a scalar, and adds the resulting value to another vector.

Subroutine **SCopy** copies a vector into another vector.

Subroutine **SCopyK** copies a column in a matrix into a vector.

10.2. *List of Significant SWMS_2D Program Variables.*

Variables which appear in subroutines of the ORTHOFEM package are not given in following tables. Consult the user's guide of ORTHOFEM [*Mendoza et al.*, 1991] for their definition.

Table 10.3. List of significant integer variables.

ALevel	Time level at which a time-dependent boundary condition is specified.
cKod	Code which specifies the type of boundary condition used for solute transport.
II	Maximum number of nodes on any transverse line (Table 8.8).
ItCum	Cumulative number of iterations (Table 9.4).
Iter	Number of iterations (Table 9.4).
Kat	Type of flow system to be analyzed (Table 8.1).
MaxAl	Number of atmospheric data records (Table 8.11).
MaxIt	Maximum number of iterations allowed during any time step for the solution of water flow equation (Table 8.1).
MBand	Bandwidth (or half-bandwidth) of the symmetric (or asymmetric) matrix A when Gaussian elimination is used. Maximum number of nodes adjacent to another node when iterative solvers are used.
MBandD	Maximum permitted bandwidth of matrix A when Gaussian elimination is used. Maximum permitted number of nodes adjacent to another node when iterative solvers are used (Table 6.7).
MPL	Number of specified print-times at which detailed information about the pressure head, the water content, flux, concentration, and the soil water and solute balances is printed (Table 8.3).
NCorn	Number of corner nodes of a particular element.
NDr	Number of drains.
NDrD	Maximum permitted number of drains.
NLay	Number of subregions for which separate water balances are being computed (Table 8.2).
NLevel	Number of time levels at which matrix A and vector B are assembled for solute transport.
NMat	Number of soil materials (Table 8.2).
NMatD	Maximum permitted number of soil materials (Table 6.7).
NObs	Number of observation nodes for which values of the pressure head, water content, and concentration are printed at each time level
NObsD	Maximum number of observation nodes for which values of pressure head, water content, and concentration are printed at each time level

Table 10.3. (continued)

NPar	Number of unsaturated soil hydraulic parameters specified for each material (Table 8.2).
NSeep	Number of seepage faces expected to develop (Table 8.5).
NSeepD	Maximum permitted number of seepage faces (Table 6.7).
NTab	Number of entries in the internally generated tables of the hydraulic properties (see Section 4.3.11).
NTabD	Maximum permitted number of entries in the internally generated tables of the hydraulic properties (Table 6.7).
NumBP	Number of boundary nodes for which $Kode(N) \neq 0$ (Table 8.8).
NumBPD	Maximum permitted number of boundary nodes for which $Kode(n) \neq 0$ (Table 6.7).
NumEl	Number of elements (quadrilaterals and/or triangles) (Table 8.8).
NumElD	Maximum permitted number of elements in finite element mesh (Table 6.7).
NumKD	Maximum permitted number of available code number values (Table 6.7).
NumNP	Number of nodal points (Table 8.8).
NumNPD	Maximum permitted number of nodes in finite element mesh (Table 6.7).
NumSEl	Number of subelements (triangles).
NumSPD	Maximum number of nodes along a seepage face (Table 6.7).
NUS	Number of corner nodes of a particular element (=*NCorn*).
PLevel	Print time-level (current print-time number).
TLevel	Time-level (current time-step number) (Table 9.4).

Table 10.4. List of significant real variables.

AE	Area of a triangular element, [L^2] or [L^3]†.
Alf	1-*Epsi*, where *Epsi* is a temporal weighing coefficient [-].
Alfa	Parameter in the soil water retention function [L^{-1}] (see Section 2.3).
Angle	Angle in degrees between the principal direction of K_1^A and the x-axis of the global coordinate system assigned to each element (Table 8.9).
Aqh	Parameter A_{qh} in equation (6.1) [LT^{-1}] (Table 8.11).
AreaR	Area of the domain occupied by the root zone, [L^2] or [L^3]†.
ATot	Area of the entire flow domain, [L^2] or [L^3]† (*Area* in Table 9.6).
Bqh	Parameter B_{qh} in equation (6.1) [L^{-1}] (Table 8.11).
cBalR	Relative error in the solute mass balance of the entire flow domain [%] (see equation (5.30)) (*CncBalR* in Table 9.6).
cBalT	Absolute error in the solute mass balance of the entire flow domain, [ML^{-1}] or [M]† (see equation (5.29)) (*CncBalT* in Table 9.6).
cBnd	Value of the boundary condition for solute transport [ML^{-1}].
cCumA	Sum of the absolute values of all cumulative solute fluxes across the flow boundaries, including those resulting from sources and sinks in the flow domain, [ML^{-1}] or [M]† (see equation (5.30)).
cCumT	Sum of all cumulative solute fluxes across the boundaries, including those resulting from sources and sinks in the flow domain, [ML^{-1}] or [M]† (see right hand side of equation (5.29)).
cE	Average concentration of an element [ML^{-3}].
Change	Inflow/Outflow to/from the flow domain, [L^2T^{-1}] or [L^3T^{-1}]† (*InFlow* in Table 9.6).
cht	Time-dependent concentration for the first-type boundary condition assigned to nodes for which *Kode*(n) = +3 [ML^{-3}] (Table 8.11).
cNewE	Amount of solute in a particular element at the new time-level, [ML^{-1}] or [M]†.
ConA1	First principal component, K_1^A, of the dimensionless anisotropy tensor $\mathbf{K}^A$ [-] assigned to each element (Table 8.9).
ConA2	Second principal component, K_2^A [-] (Table 8.9).
ConVol	Amount of solute in the entire flow domain, [ML^{-1}] or [M]† (*ConVol* in Table 9.6).
Courant	Maximum local Courant number [-] (Table 9.4).
cPrec	Solute concentration of rainfall water [ML^{-3}] (Table 8.11).
crt	Time-dependent concentration of the drainage flux, or some other time-dependent prescribed flux for nodes were *Kode*(n) = -3 [ML^{-3}] (Table 8.11).
cSink	Concentration of the sink term [ML^{-1}].
cTot	Mean concentration in the flow domain [ML^{-3}] (*cMean* in Table 9.6).

Table 10.4. (continued)

CumCh0	Cumulative amount of solute removed from the entire flow domain by zero-order reactions, [ML^{-1}] or [M]† (Table 9.5).
CumCh1	Cumulative amount of solute removed from the entire flow domain by first-order reactions, [ML^{-1}] or [M]† (Table 9.5).
CumChR	Cumulative amount of solute removed from the entire flow domain by root water uptake, [ML^{-1}] or [M]† (Table 9.5).
CumQrR	Cumulative total potential transpiration from the entire flow domain, [L^2] or [L^3]† (*CumQRP* in Tables 9.3 and 9.7).
CumQrT	Cumulative total potential flux across the atmospheric boundary, [L^2] or [L^3]† (*CumQAP* in Tables 9.3 and 9.7).
CumQvR	Cumulative total actual transpiration from the entire flow domain, [L^2] or [L^3]† (*CumQR* in Tables 9.3 and 9.7).
cVolI	Initial amount of solute in the entire flow domain, [ML^{-1}] or [M]†.
DeltC	Sum of the absolute changes in concentrations as summed over all elements, [ML^{-1}] or [M]† (see equation (5.30)).
DeltW	Sum of the absolute changes in water content as summed over all elements, [L^2] or [L^3]† (see equation (4.25)).
dlh	Spacing (logarithmic scale) between consecutive pressure heads in the internally generated tables of the hydraulic properties [-] (see equation (4.28)).
dMul	Dimensionless number by which Δt is multiplied if the number of iterations is less than or equal to 3 [-] (Table 8.3).
dMul2	Dimensionless number by which Δt is multiplied if the number of iterations is greater than or equal to 7 [-] (Table 8.3).
dt	Time increment Δt [T] (Table 8.3).
dtMax	Maximum permitted time increment Δt_{max} [T] (Table 8.3).
dtMaxC	Maximum permitted time increment Δt_{max} for solute transport [T] (see equation (5.32)).
dtMin	Minimum permitted time increment Δt_{min} [T] (Table 8.3).
dtOld	Old time increment [T].
dtOpt	Optimal time increment [T].
EI	Potential surface flux per unit atmospheric boundary [LT^{-1}] (=*rTop*).
Epsi	Temporal weighing coefficient [-] (Table 8.7).
Epslon	Absolute change in the nodal pressure head between two successive iterations [L].
GWL	Time-dependent prescribed head boundary condition [L] for nodes indicated by *Kode*(n) = +3 (Table 8.11).
GWL0L	Parameter in equation (6.1) [L] (Table 8.11).
hCritA	Minimum allowed pressure head at the soil surface [L] (Table 8.11).

Table 10.4. (continued)

hCritS	Maximum allowed pressure head at the soil surface [L] (Table 8.11).
hE	Mean element value of the pressure head [L].
hMeanG	Mean value of the pressure head calculated over a set of nodes for which $Kode(n) = \pm 3$ [L] (*hKode3* in Tables 9.1 and 9.7).
hMeanR	Mean value of the pressure head within the root zone [L] (*hRoot* in Table 9.1 and 9.7).
hMeanT	Mean value of the pressure head calculated over a set of nodes for which $Kode(n) = \pm 4$ [L] (*hAtm* in Tables 9.1 and 9.7).
hTab1	Lower limit [L] of the pressure head interval for which tables of hydraulic properties is generated internally for each material (*ha* in Table 8.2).
hTabN	Upper limit [L] of the pressure head interval for which tables of hydraulic properties is generated internally for each material (*hb* in Table 8.2).
hTot	Mean pressure head in the entire flow domain [L] (*hMean* in Table 9.6).
Kk	Unsaturated hydraulic conductivity corresponding to θ_k [LT^{-1}] (see Section 2.3) (Table 8.2).
Ks	Saturated hydraulic conductivity [LT^{-1}] (Table 8.2).
m	Parameter in the soil water retention function [-] (see Section 2.3) (Table 8.2).
n	Parameter in the soil water retention function [-] (see Section 2.3) (Table 8.2).
Peclet	Maximum local Peclet number [-] (Table 9.4).
Prec	Precipitation [LT^{-1}] (Table 8.11).
P0	Value of the pressure head [L], h_1, below which roots start to extract water from the soil (Table 8.4).
P2H	Value of the limiting pressure head [L], h_3, below which the roots cannot extract water at the maximum rate (assuming a potential transpiration rate of *r2P*) (Table 8.4).
P2L	As above, but for a potential transpiration rate of *r2L* (Table 8.4).
P3	Value of the pressure head [L], h_4, below which root water uptake ceases (usually equal to the wilting point) (Table 8.4).
Qa	Parameter in the soil water retention function [-] (see Section 2.3) (Table 8.2).
Qk	Volumetric water content coresponding to K_k [-] (see Section 2.3) (Table 8.2).
Qm	Parameter in the soil water retention function [-] (see Section 2.3) (Table 8.2).
Qr	Residual soil water content [-].
Qs	Saturated soil water content [-].
rQWL	Time-dependent prescribed flux boundary condition [LT^{-1}] for nodes were $Kode(n) = -3$ (Table 8.11).
rLen	Width of soil surface associated with transpiration, [L] or [L^2]† (Table 8.10).
RootCh	Amount of solute removed from a particular subelement during one time step by root water uptake, [ML^{-1}] or [M]†.

Table 10.4. (continued)

rRoot	Potential transpiration rate [LT^{-1}] (Table 8.11).
rSoil	Potential evaporation rate [LT^{-1}] (Table 8.11).
rTop	Potential surface flux per unit atmospheric boundary [LT^{-1}] (*rAtm* in Table 9.2).
r2H	Potential transpiration rate [LT^{-1}] (see Table 8.4).
r2L	Potential transpiration rate [LT^{-1}] (see Table 8.4).
t	Time, *t*, at current time-level [T].
tAtm	Time for which the *i*-th data record is provided [T] (Table 8.11).
Tau	Tortuosity factor [-].
tFix	Next time resulting from time discretizations 2 and 3 [T] (see Section 4.3.3).
tInit	Starting time of the simulation [T] (Table 8.11).
tMax	Maximum duration of the simulation [T].
tOld	Previous time-level [T].
TolH	Maximum desired absolute change in the value of the pressure head, h [L], between two successive iterations during a particular time step (Table 8.1).
TolTh	Maximum desired absolute change in the value of the water content, θ [L], between two successive iterations during a particular time step (Table 8.1).
tPulse	Time duration of the concentration pulse [T] (Table 8.7).
Vabs	Absolute value of the nodal Darcy fluid flux density [LT^{-1}].
vMeanR	Actual transpiration rate [LT^{-1}] (*vRoot* in Table 9.2).
vNewE	Volume of water in a particular element at the new time-level, [L^2] or [L^3]†
vOldE	Volume of water in a particular element at the old time-level, [L^2] or [L^3]†
Volume	Volume of water in the entire flow domain, [L^2] or [L^3]† (Table 9.6).
wBalR	Relative error in the water mass balance in the entire flow domain [%] (see equation (4.25)).
wBalT	Absolute error in the water mass balance in the entire flow domain , [L^2] or [L^3]† (see equation (4.24)).
wCumA	Sum of the absolute values of all fluxes across the flow boundaries, including those resulting from sources and sinks in the region, [L^2] or [L^3]† (see equation (4.25)).
wCumT	Sum of all cumulative fluxes across the flow boundaries, including those resulting from sources and sinks in the region, [L^2] or [L^3]† (see equation (4.24)).
wVolI	Initial volume of water in the flow domain, [L^2] or [L^3]†.
xMul	Parameter which depends on the type of flow system being analyzed, [-] or [L]† (see equations (4.11) and (4.12)).

† for plane and axisymmetric flow, respectively

Table 10.5. List of significant logical variables.

AtmInf	Logical variable indicating whether or not the input file ATMOSPH.IN is provided (Table 8.1).
CheckF	Logical variable indicating whether or not the grid input data are to be printed for checking (Table 8.1).
DrainF	Logical variable indicating whether drains are, or are not, present in the transport domain (Table 8.1); if drains are present, they are represented by an electrical resistance network analog.
Explic	Logical variable indicating whether an explicit or implicit scheme was used for solving the water flow equation.
FluxF	Logical variable indicating whether or not detailed flux information is to be printed (Table 8.1).
FreeD	Logical variable indicating whether a unit hydraulic gradient (free drainage) is, or is not, invoked at the bottom of the transport domain (Table 8.1).
ItCrit	Logical variable indicating whether or not convergence was achieved.
lArtD	Logical variable indicating whether an artificial dispersion is, or is not, to be added in order to satisfy the stability criterion *PeCr* (Table 8.7).
lChem	Logical variable indicating whether or not the solute transport equation is to be solved (Table 8.1).
lConst	Logical variable indicating whether or not there is a constant number of nodes at any transverse line.
lUpW	Logical variable indicating if upstream weighing or the standard Galerkin formulation is to be used (Table 8.7).
lWat	Logical variable indicating if steady-state or transient water flow is to be considered (Table 8.1).
qGWLF	Logical variable indicating whether or not the discharge-groundwater level relationship is used as bottom boundary condition (Table 8.11).
SeepF	Logical variable indicating whether or not a seepage face is to be expected (Table 8.1).
ShortF	Logical variable indicating whether or not the printing of time-level information is to be suppressed on each time level (Table 8.1).
SinkF	Logical variable indicating whether or not plant water uptake will take place (Table 8.11).

Table 10.6. List of significant arrays.

A(MBandD,NumNPD)	Coefficient matrix.
Ac(NumNPD)	Nodal values of the product θR [-].
Area(10)	Areas of the specified subregions, [L^2] or [L^3]† (Table 9.6).
Axz(NumNPD)	Nodal values of the dimensionless scaling factor α_h associated with the pressure head [-] (Table 8.8).
B(NumNPD)	Coefficient vector.
Beta(NumNPD)	Nodal values of the normalized rootwater uptake distribution, [L^{-2}] or [L^{-3}]† (Table 8.8).
Bxz(NumNPD)	Nodal value of the scaling factor α_K associated with the saturated hydraulic conductivity [-] (Table 8.8).
Cap(NumNPD)	Nodal values of the soil water hydraulic capacity [L^{-1}].
CapTab(NTabD,NMatD)	Internal table of the soil water hydraulic capacity [L^{-1}].
cBound(6)	Values of the time independent concentration boundary condition [ML^{-3}] (Table 8.7).
ChemS(NumKD)	Cumulative boundary solute fluxes, [ML^{-1}] or [M]† (Table 9.5).
ChPar(10,*NMatD*)	Parameters which describe the transport properties of the porous media (Table 8.7).
cMean(10)	Mean concentrations of specified subregions [ML^{-3}] (Table 9.6).
Con(NumNPD)	Nodal values of the hydraulic conductivity [LT^{-1}].
ConAxx(NumElD)	Nodal values of the component K_{xx}^A of the anisotropy tensor $\mathbf{K}^A$ [-].
ConAxz(NumElD)	Nodal values of the component K_{xz}^A of the anisotropy tensor $\mathbf{K}^A$ [-].
ConAzz(NumElD)	Nodal values of the component K_{zz}^A of the anisotropy tensor $\mathbf{K}^A$ [-].
Conc(NumNPD)	Nodal values of the concentration [ML^{-3}] (Table 8.8).
ConSat(NMatD)	Saturated hydraulic conductivities of the material [LT^{-1}].
ConSub(10)	Amounts of solute in the specified subregions, [ML^{-1}] or [M]† (Table 9.6).
ConTab(NTabD,NMatD)	Internal table of the hydraulic conductivity [LT^{-1}].
CumQ(NumKD)	Cumulative boundary fluxes, [L^2] or [L^3]† (Table 9.3).
Dispxx(NumNPD)	Nodal values of the component D_{xx} of the dispersion tensor [L^2T^{-1}].
Dispxz(NumNPD)	Nodal values of the component D_{xz} of the dispersion tensor [L^2T^{-1}].
Dispzz(NumNPD)	Nodal values of the component D_{zz} of the dispersion tensor [L^2T^{-1}].
DS(NumNPD)	Vector $\{D\}$ in the global matrix equation for water flow, [L^2T^{-1}] or [L^3T^{-1}]† (see equation (4.9)); also used for the diagonal of the coefficient matrix $[Q]$ in the global matrix equation for solute transport, [L^2] or [L^3]† (see equation (5.5)).
Dxz(NumNPD)	Nodal values of the scaling factor α_θ associated with the water content (Table 8.8).
E(3,3)	Element contributions to the global matrix A for water flow [L^2] (see equation (4.5)).

Table 10.6. (continued)

EfDim(2,NDr)	Effective diameter of drains and side lengths of the finite element mesh representing the drain (Table 8.6).
F(NumNPD)	Diagonal of the coefficient matrix [*F*] in the global matrix equation for water flow, [L^2] or [L^3]† (see equation (4.7)).
Fc(NumNPD)	Nodal values of the parameter F [T^{-1}] (see equation (3.5)).
Gc(NumNPD)	Nodal values of the parameter G [$ML^{-3}T^{-1}$] (see equation (3.5)).
hMean(10)	Mean values of the pressure head in specified subregions [L] (Table 9.6).
hMean(NumKD)	Mean values of the pressure head along a certain type of boundary [L] (Table 9.6).
hNew(NumNPD)	Nodal values of the pressure head [L] at the new time-level (Table 8.8).
hOld(NumNPD)	Nodal values of the pressure head [L] at the old time-level.
hSat(NMatD)	Air-entry values for each material [L].
hTab(NTabD)	Internal table of the pressure head [L].
hTemp(NumNPD)	Nodal values of the pressure head [L] at the previous iteration.
IU(11)	Vector which contains identification numbers of output files.
KElDr(NDr, NElD)	Global numbers of elements surrounding a particular drain (Table 8.6).
KodCB(NumBPD)	Codes which identify type of boundary condition and refer to the vector *cBound* for time-independent solute transport boundary conditions (Table 8.7).
Kode(NumNPD)	Codes which specify the type of boundary condition (Table 8.8).
KX(NumElD,4)	Global nodal numbers of element corner nodes (Table 8.8).
KXB(NumBPD)	Global nodal numbers of sequentially numbered boundary nodes for which $Kode(n) \neq 0$ (Table 8.8).
LayNum(NumElD)	Subregion numbers assigned to each element (Table 8.9).
ListNE(NumNPD)	Number of subelements adjacent to a particular node.
MatNum(NumNPD)	Indices for material whose hydraulic and transport properties are assigned to a particular node (Table 8.8).
ND(NDr)	Global number of a drain (Table 8.6).
NElD(NDr)	Number of elements surrounding a drain (Table 8.6).
Node(NObsD)	Observation nodes for which values of the pressure head, water content, and concentration are printed at each time level (Table 8.10).
NP(NSeepD,NumSPD)	Sequential global numbers of nodes on the seepage face (Table 8.5).
NSP(NSeepD)	Numbers of nodes on seepage face (Table 8.5).
Par(10,NMatD)	Parameters which describe the hydraulic properties of the porous medium (Table 8.2).
POptm(NMatD)	Values of the pressure head [L], h_2, below which roots start to extract water at the maximum possible rate (Table 8.4).

Table 10.6. (continued)

Q(NumNPD)	Nodal values of the recharge/discharge rate, $[L^2T^{-1}]$ or $[L^3T^{-1}]^\dagger$ (Table 8.8).
Qc(NumNPD)	Nodal values of solute fluxes, $[ML^{-1}T^{-1}]$ or $[MT^{-1}]^\dagger$.
Sink(NumNPD)	Nodal values of the sink term $[T^{-1}]$ (see equation (2.3)).
SMean(NumKD)	Total solute fluxes, $[ML^{-1}T^{-1}]$ or $[MT^{-1}]^\dagger$ (Table 9.5).
SolIn(NumElD)	Element values of the initial amount of solute, $[ML^{-1}]$ or $[M]^\dagger$ (Table 9.6).
SubCha(10)	Inflow/Outflow to/from specified subregions, $[L^2T^{-1}]$ or $[L^3T^{-1}]^\dagger$ (Table 9.6).
SubVol(10)	Volumes of water in specified subregions, $[L^2]$ or $[L^3]^\dagger$ (Table 9.6).
SWidth(NumKD)	Length of a boundary associated with a certain type of boundary condition, [L] or $[L^2]^\dagger$.
TheTab(NTabD,NMatD)	Internal table of the soil water content [-].
ThNew(NumNPD)	Nodal values of the water content at the new time level [-].
ThOld(NumNPD)	Nodal values of the water content at the old time level [-].
thr(NMatD)	Residual water contents for specified materials [-].
thSat(NMatD)	Saturated water contents for specified materials [-].
TPrint(MPL)	Specified print-times [T] (Table 8.3).
vMean(NumKD)	Values of boundary fluxes across a certain type of boundary, $[L^2T^{-1}]$ or $[L^3T^{-1}]^\dagger$.
Vx(NumNPD)	Nodal values of the x-component of the Darcian velocity vector $[LT^{-1}]$.
Vz(NumNPD)	Nodal values of the z-component of the Darcian velocity vector $[LT^{-1}]$.
WatIn(NumElD)	Element values of the initial volume of water, $[L^2]$ or $[L^3]^\dagger$.
*WeTab(3,2*NumElD)*	Weighing factors associated with the sides of subelements [-].
Width(NumBPD)	Width of the boundary [L] associated with boundary nodes (Table 8.10).
x(NumNPD)	x-coordinates [L] of the nodal points (Table 8.8).
y(NumNPD)	y or z-coordinates [L] of the nodal points (Table 8.8).

† for plane and axisymmetric flow, respectively

```
*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
*                                                                          *
*     SWMS_2D  - Numerical model of two-dimensional flow and solute        *
*                transport in a variably saturated porous medium           *
*                Conjugate gradient solver for symmetric matrix            *
*                ORTHOMIN solver for asymmetric matrix                     *
*                version 1.21                                              *
*                                                                          *
*     Updated by J. Simunek (1994)                                         *
*     Based on model SWMS_2D (Simunek et al., 1992)                        *
*                                                                          *
*                                          Last modified: July, 1994       *
*                                                                          *
*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*

      program SWMS_2D

      parameter (NumNPD=5000,
     !           NumElD=5000,
     !           NumBPD=200,
     !           MBandD=15,
     !           NSeepD=2,
     !           NumSPD=50,
     !           NDrD=2,
     !           NElDrD=8,
     !           NMatD=5,
     !           NTabD=100,
     !           NumKD=6,
     !           NObsD=4,
     !           MNorth=4)

      double precision A,B,RTime1,RTime2
      double precision A1,B1,VRV,RES,RQI,RQ,QQ,QI,RQIDOT,ECNVRG,
     !                 RCNVRG,ACNVRG
      logical lWat,lChem,SinkF,qGWLF,AtmInF,ShortF,SeepF,CheckF,FluxF,
     !        Explic,lUpW,FreeD,DrainF,lArtD,lOrt
      integer PLevel,ALevel,TLevel

      dimension A(MBandD,NumNPD),B(NumNPD),Kode(NumNPD),Q(NumNPD),
     !  hNew(NumNPD),hTemp(NumNPD),hOld(NumNPD),ConSat(NMatD),F(NumNPD),
     !  hTab(NTabD),ConTab(NTabD,NMatD),CapTab(NTabD,NMatD),Con(NumNPD),
     !  Cap(NumNPD),x(NumNPD),y(NumNPD),MatNum(NumNPD),LayNum(NumElD),
     !  KX(NumElD,4),KXB(NumBPD),TPrint(50),Par(10,NMatD),Width(NumBPD),
     !  ConAxx(NumElD),ConAzz(NumElD),ConAxz(NumElD),SWidth(NumKD),
     !  NP(NSeepD,NumSPD),NSP(NSeepD),hSat(NMatD),WatIn(NumElD),
     !  Axz(NumNPD),Bxz(NumNPD),Dxz(NumNPD),thR(NMatD),thSat(NMatD),
     !  TheTab(NTabD,NMatD),ThNew(NumNPD),ThOld(NumNPD),ListNE(NumNPD),
     !  Sink(NumNPD),POptm(NMatD),Beta(NumNPD),DS(NumNPD),CumQ(NumKD),
     !  vMean(NumKD),hMean(NumKD),KodCB(NumBPD),Qc(NumNPD),Vx(NumNPD),
     !  Vz(NUmNPD),ChPar(10,NMatD),Dispxx(NumNPD),Dispzz(NumNPD),
     !  Dispxz(NumNPD),cBound(6),Ac(NumNPD),Fc(NumNPD),SolIn(NumElD),
     !  Conc(NumNPD),SMean(NumKD),ChemS(NumKD),WeTab(3,2*NumElD),
     !  Gc(NumNPD),ND(NDrD),NED(NDrD),EfDim(2,NDrD),KElDr(NDrD,NElDrD),
     !  ConO(NumNPD),Node(NObsD),
     !  B1(NumNPD),IAD(MBandD,NumNPD),IADN(NumNPD),IADD(NumNPD),
     !  A1(MBandD,NumNPD),RES(NumNPD),VRV(NumNPD),RQI(NumNPD,MNorth),
     !  RQ(NumNPD),QQ(NumNPD),RQIDOT(MNorth),QI(NumNPD,MNorth)

      open(30,file='SWMS_2D.IN\Selector.in',  status='old')
      open(32,file='SWMS_2D.IN\Grid.in',      status='old')
      open(50,file='SWMS_2D.OUT\Check.out',   status='new')
      open(71,file='SWMS_2D.OUT\v_Mean.out',  status='new')
      open(72,file='SWMS_2D.OUT\A_Level.out', status='new')
      open(75,file='SWMS_2D.OUT\h.out',       status='new')
      open(76,file='SWMS_2D.OUT\th.out',      status='new')
      open(77,file='SWMS_2D.OUT\h_Mean.out',  status='new')
      open(78,file='SWMS_2D.OUT\Cum_Q.out',   status='new')
      open(79,file='SWMS_2D.OUT\Boundary.out',status='new')
```

```
      open(80,file='SWMS_2D.OUT\Balance.out', status='new')
      open(81,file='SWMS_2D.OUT\vz.out',      status='new')
      open(82,file='SWMS_2D.OUT\vx.out',      status='new')
      open(92,file='SWMS_2D.OUT\ObsNod.out',  status='new')

      data SinkF  ,qGWLF  ,tInit,NTab,ItCum,Iter,TLevel,ALevel,PLevel
     !    /.false.,.false., 0.  ,100 , 0   , 0  ,  1  ,  1  ,  1 /
     !
     !     CumQ    ,Sink     ,CumQrT,CumQrR,CumQvR,ChemS   ,rRoot,rTop
     !    /NumKD*0.,NumNPD*0.,0.    ,0.    ,0.    ,NumKD*0.,  0. , 0./
     !
     !     CumCh0,CumCh1,CumChR,dtMaxC,wCumA,cCumA,Explic
     !    /  0. ,   0. ,   0. ,1.e+30,  0. ,  0. ,.false./

      data ECNVRG,ACNVRG,RCNVRG,MaxItO
     !    /1.0d-6,1.0d-6,1.0d-6, 200 /

* --- Reading of the input files and initial calculations --------------

      call BasInf (KAT,MaxIt,TolTh,TolH,lWat,lChem,AtmInF,ShortF,SeepF,
     !             CheckF,FluxF,FreeD,DrainF)
      call NodInf (NumNP,NumEl,IJ,NumBP,NumNPD,NumElD,NumBPD,NumKD,NObs,
     !             NObsD,Kode,Q,Conc,hNew,hOld,hTemp,x,y,MatNum,Beta,
     !             Axz,Bxz,Dxz,CheckF)
      call ElemIn (NumEl,NumElD,NumNP,KX,LayNum,ConAxx,ConAzz,ConAxz,
     !             CheckF,ListNE,IJ,MBand,MBandD,lChem,lOrt)
      call GeomIn (NumKD,NumNP,NumBP,NObs,SWidth,Width,Kode,KXB,rLen,
     !             Node)
      call IADMake(KX,NumNP,NumEl,NumElD,MBandD,IAD,IADN,IADD)
      close(32)
      call MatIn  (NMatD,NMat,NLay,Par,hTab(1),hTab(NTab))
      call GenMat (NTab,NTabD,NMat,thR,hSat,Par,hTab,ConTab,CapTab,
     !             ConSat,TheTab,thSat)
      call SetMat (NumNP,NTab,NTabD,NMat,hTab,ConTab,CapTab,hNew,hOld,
     !             MatNum,Par,Con,Cap,ConSat,Axz,Bxz,Dxz,hSat,hTemp,
     !             Explic,TheTab,thSat,thR,ThOld)
      if(AtmInF) then
        open(31,file='SWMS_2D.IN\Atmosph.in', status='old')
        call AtmIn (GWLOL,SinkF,qGWLF,tInit,tMax,Aqh,Bqh,hCritS,MaxAL)
        call SetAtm(tAtm,rTop,rRoot,hCritA,Width,KXB,NumBP,Kode,hNew,Q,
     !              NumNP,GWLOL,qGWLF,FreeD,cPrec,cht,crt)
      end if
      call TmIn    (tInit,tMax,tAtm,tOld,dt,dtMax,dMul,dMul2,dtMin,
     !              TPrint,t,dtOpt,dtOld,AtmInF)
      if(SinkF) then
        call SinkIn(NMat,NumEl,NumNP,NumElD,KAT,KX,x,y,PO,POptm,P2H,P2L,
     !              P3,r2H,r2L,Beta)
        call SetSnk(NumNP,NMat,MatNum,hNew,rRoot,Sink,PO,POptm,P2H,P2L,
     !              P3,r2H,r2L,Beta,rLen)
      end if
      if(SeepF)
     !  call SeepIn(NSeepD,NumSPD,NSeep,NSP,NP)
      if(DrainF)
     !  call DrainIn(NDr,NDrD,NElDrD,NumEl,ND,NED,KElDr,EfDim,ConAxx,
     !               ConAxz,ConAzz)
      if(lChem) then
        call ChemIn(NMat,NumBP,cBound,ChPar,epsi,tPulse,KodCB,NLevel,
     !              lUpW,lArtD,PeCr)
        if(lWat)
     !  call ChInit(NumNP,NumEl,NumElD,NMat,x,y,KX,MatNum,NLevel,Con,
     !              hNew,Sink,cBound(5),Vx,Vz,ConAxx,ConAzz,ConAxz,
     !              Dispxx,Dispzz,Dispxz,ChPar,ThOld,thSat,Conc,Fc,Gc,
     !              ListNE,lUpW,WeTab,dt,dtMaxC,Peclet,Courant,KAT,
     !              lArtD,PeCr,ConO)
        open(83,file='SWMS_2D.OUT\Conc.out'  ,status='new')
        open(74,file='SWMS_2D.OUT\Solute.out',status='new')
        call cOut  (NumNP,Conc,x,y,tInit,IJ)
      end if
      close(30)
      close(50)
      open(73,file='SWMS_2D.OUT\Q.out',       status='new')
      open(70,file='SWMS_2D.OUT\Run_Inf.out', status='new')
```

```
      call hOut  (hNew,x,y,NumNP,tInit,IJ)
      call thOut (ThOld,x,y,NumNP,tInit,IJ)
      call SubReg(NumEl,NumElD,NumNP,NMat,hNew,ThOld,ThOld,x,y,MatNum,
     !            LayNum,KX,KAT,tInit,dt,NLay,0,lWat,lChem,Conc,ChPar,
     !            wCumA,wCumT,cCumA,cCumT,wVolI,cVolI,WatIn,SolIn)
      if(NObs.gt.0) call ObsNod(tInit,NumNP,NObs,Node,hNew,ThOld,Conc)

      write(*,*)'beginning of numerical solution'
      call getdat(i,i,iday)
      call gettim(ihours,mins,isecs,i)
      Rtime1=iday*24.*60.*60.+ihours*60.*60.+mins*60.+isecs

* --- Beginning of time loop -------------------------------------------

11    continue

*     Calculate water flow
      if(lWat.or.TLevel.eq.1)
     !call WatFlow(NumNP,NumEl,NumElD,NTab,NTabD,MBand,MBandD,NMat,
     !             NSeep,NSeepD,NumSPD,NSP,NP,NumBP,ItCum,MaxIt,Iter,
     !             Kode,KAT,t,dt,dtMin,dtOpt,dtOld,tOld,hCritA,hCritS,
     !             TolTh,TolH,rLen,Width,rTop,vMeanR,hMeanR,AtmInf,
     !             SinkF,SeepF,qGWLF,FreeD,Par,hTab,ConTab,CapTab,
     !             TheTab,hNew,hOld,hTemp,thR,thSat,ThNew,ThOld,MatNum,
     !             Con,Cap,ConSat,Axz,Bxz,Dxz,hSat,A,B,Q,F,x,y,KX,Sink,
     !             DS,Beta,ConAxx,ConAzz,ConAxz,KXB,Explic,GWLOL,Aqh,
     !             Bqh,lWat,TLevel,lOrt,DrainF,ND,NDr,rRoot,PO,POptm,
     !             P2H,P2L,P3,r2H,r2L,ConO,
     !             A1,B1,NumNPD,IAD,IADN,IADD,VRV,RES,RQI,RQ,QQ,QI,
     !             RQIDOT,ECNVRG,RCNVRG,ACNVRG,MNorth,MaxItO)
      if(.not.lWat.and.TLevel.eq.1) then
        if(lChem) then
          call ChInit(NumNP,NumEl,NumElD,NMat,x,y,KX,MatNum,NLevel,Con,
     !                hNew,Sink,cBound(5),Vx,Vz,ConAxx,ConAzz,ConAxz,
     !                Dispxx,Dispzz,Dispxz,ChPar,ThNew,thSat,Conc,Fc,Gc,
     !                ListNE,lUpW,WeTab,dt,dtMaxC,Peclet,Courant,KAT,
     !                lArtD,PeCr,ConO)
        else
          call Veloc(KAT,NumNP,NumEl,NumElD,hNew,x,y,KX,ListNE,Con,
     !               ConAxx,ConAzz,ConAxz,Vx,Vz)
        end if
        Iter=1
      end if

*     Calculate solute transport
      if(lChem)
     !  call Solute(NumNP,NumEl,NumElD,MBand,MBandD,NMat,t,Kode,A,B,Q,
     !              hNew,hOld,F,x,y,KX,KAT,dt,DS,Sink,MatNum,Con,ConO,
     !              ConAxx,ConAzz,ConAxz,Vx,Vz,Dispxx,Dispzz,Dispxz,
     !              ChPar,ThNew,ThOld,thSat,Ac,Fc,Gc,Qc,Conc,ListNE,
     !              cBound,tPulse,NumBP,KodCB,KXB,NLevel,cPrec,crt,cht,
     !              lWat,lUpW,WeTab,epsi,CumCh0,CumCh1,CumChR,dtMaxC,
     !              Peclet,Courant,lArtD,PeCr,lOrt,
     !              A1,B1,NumNPD,IAD,IADN,IADD,VRV,RES,RQI,RQ,QQ,QI,
     !              RQIDOT,ECNVRG,RCNVRG,ACNVRG,MNorth,MaxItO)

*     T-Level information
      call TLInf   (NumNP,NumBP,Kode,Q,hNew,CumQ,Width,SWidth,KXB,t,dt,
     !              TLevel,ShortF,TPrint(PLevel),Iter,ItCum,rTop,rRoot,
     !              vMeanR,hMeanT,hMeanR,hMeanG,AtmInF,SinkF,CumQrT,
     !              CumQrR,CumQvR,NumKD,hMean,vMean,lWat,lChem,rLen,
     !              Peclet,Courant,wCumT,wCumA)
      if(lChem)
     !  call SolInf(1,NumNP,Kode,Qc,t,dt,TLevel,ShortF,TPrint(PLevel),
     !              AtmInF,NumKD,SMean,ChemS,CumCh0,CumCh1,CumChR,cCumA,
     !              cCumT,lWat)
      if(NObs.gt.0) call ObsNod(t,NumNP,NObs,Node,hNew,ThNew,Conc)

*     P-Level information
      if(abs(TPrint(PLevel)-t).lt.0.001*dt) then
        if(lWat.or.(.not.lWat.and.PLevel.eq.1)) then
          call hOut (hNew,x,y,NumNP,t,IJ)
          call thOut(ThNew,x,y,NumNP,t,IJ)
```

```
          if(FluxF) then
            if(.not.lChem)
     !        call Veloc(KAT,NumNP,NumEl,NumElD,hNew,x,y,KX,ListNE,Con,
     !                   ConAxx,ConAzz,ConAxz,Vx,Vz)
            call FlxOut(Vx,Vz,x,y,NumNP,t,IJ)
            call QOut  (Q,x,y,NumNP,t,IJ)
          end if
        end if
        call SubReg(NumEl,NumElD,NumNP,NMat,hNew,ThOld,ThNew,x,y,MatNum,
     !              LayNum,KX,KAT,t,dt,NLay,PLevel,lWat,lChem,Conc,
     !              ChPar,wCumA,wCumT,cCumA,cCumT,wVolI,cVolI,WatIn,
     !              SolIn)
        call BouOut(NumNP,NumBP,t,hNew,ThNew,Q,Width,KXB,Kode,x,y,Conc)
        if(lChem)
     !    call cOut(NumNP,Conc,x,y,t,IJ)
        PLevel=PLevel+1
      end if

*     A-level information
      if(abs(t-tAtm).le.0.001*dt.and.AtmInF) then
        if(lWat)
     !    call ALInf (t,CumQ,hMeanT,hMeanR,hMeanG,ALevel,CumQrT,CumQrR,
     !                CumQvR,NumKD)
        if(lChem)
     !    call SolInf(0,NumNP,Kode,Qc,t,dt,ALevel,ShortF,0.,AtmInF,
     !                NumKD,SMean,ChemS,CumCh0,CumCh1,CumChR,cCumA,
     !                cCumT,lWat)
        if(ALevel.lt.MaxAL) then
          call SetAtm(tAtm,rTop,rRoot,hCritA,Width,KXB,NumBP,Kode,hNew,
     !                Q,NumNP,GWLOL,qGWLF,FreeD,cPrec,cht,crt)
          ALevel=ALevel+1
        end if
      end if

*     Root extraction
      if(SinkF)
     !  call SetSnk(NumNP,NMat,MatNum,hNew,rRoot,Sink,P0,POptm,P2H,P2L,
     !              P3,r2H,r2L,Beta,rLen)

*     Time governing
      if (abs(t-tMax).le.0.001*dt) then
        call getdat(i,i,iday)
        call gettim(ihours,mins,isecs,i)
        Rtime2=iday*24.*60.*60.+ihours*60.*60.+mins*60.+isecs
        write(70,*)
        write(70,*) 'Real time [sec]',Rtime2-RTime1
        write( *,*) 'Real time [sec]',Rtime2-RTime1
        stop
      end if
      tOld=t
      dtOld=dt
      call TmCont(dt,dtMax,dtOpt,dMul,dMul2,dtMin,Iter,TPrint(PLevel),
     !            tAtm,t,tMax,dtMaxC)
      TLevel=TLevel+1
      t=t+dt
      goto 11

* --- end of time loop -------------------------------------------------

      end

*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
* Source file INPUT2.FOR !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

      subroutine BasInf(KAT,MaxIt,TolTh,TolH,lWat,lChem,AtmInF,ShortF,
     !                  SeepF,CheckF,FluxF,FreeD,DrainF)

      character*72 Hed
      character*5  LUnit,TUnit,MUnit
      logical lWat,lChem,CheckF,AtmInF,ShortF,SeepF,FluxF,FreeD,DrainF
      dimension IU(11)
      data IU /50,71,72,75,76,77,78,79,80,81,82/

      read(30,*)
      read(30,*)
      read(30,*) Hed
      read(30,*)
      read(30,*) LUnit,TUnit,MUnit
      read(30,*)
      read(30,*) KAT
      read(30,*)
      read(30,*) MaxIt,TolTh,TolH
      read(30,*)
      read(30,*) lWat,lChem,CheckF,ShortF,FluxF,AtmInF,SeepF,FreeD,
     !           DrainF
      do 11 i=1,11
        write(IU(i),*) Hed
        write(IU(i),*)
        write(IU(i),*)'Program SWMS_2D'
        call getdat(ii,imonth,iday)
        call gettim(ihours,mins,isecs,ii)
        write(IU(i),100) iday,imonth,ihours,mins,isecs
        if(AtmInF) then
          write(IU(i),*)'Time dependent boundary conditions'
        else
          write(IU(i),*)'Time independent boundary conditions'
        end if
        if(KAT.eq.0) write(IU(i),110)
        if(KAT.eq.1) write(IU(i),120)
        if(KAT.eq.2) write(IU(i),130)
        write(IU(i),*) 'Units: L = ',LUnit,', T = ',TUnit,', M = ',MUnit
11    continue
      write(*,*) '-------------------------------------------------------'
      write(*,*) '|                                                     |'
      write(*,*) '|                      SWMS_2D                        |'
      write(*,*) '|                                                     |'
      write(*,*) '|     Code for simulating water flow and solute       |'
      write(*,*) '|       transport in two-dimensional variably         |'
      write(*,*) '|             saturated porous media                  |'
      write(*,*) '|                                                     |'
      write(*,*) '|                   version 1.21                      |'
      write(*,*) '|              Last modified: July, 1994              |'
      write(*,*) '|                                                     |'
      write(*,*) '-------------------------------------------------------'
      write(*,*)
      write(*,*) Hed
      if(KAT.eq.0) write(*,110)
      if(KAT.eq.1) write(*,120)
      if(KAT.eq.2) write(*,130)
      write(50,140) MaxIt,TolTh,TolH

100   format(' Date: ',i3,'.',i2,'.',i2,'    Time: ',i3,':',i2,':',i2)
110   format(' Horizontal plane flow, V = L*L')
120   format(' Axisymmetric flow, V = L*L*L')
130   format(' Vertical plane flow, V = L*L')
140   format(/' Max. number of iterations          ',i4/
     !        ' Absolute water content tolerance [-]',f8.5/
     !        ' Absolute pressure head tolerance [L]',f8.5/)
      return
      end

*************************************************************************

      subroutine MatIn(NMatD,NMat,NLay,Par,hTab1,hTabN)
```

```
      real K
      dimension Par(10,NMatD),Qe(10)
      data Qe /1.,.99,.90,.85,.75,.65,.50,.35,.20,.10/

      write(*,*) 'reading material information'
      Imax=10
      read(30,*)
      read(30,*)
      read(30,*) NMat,NLay,hTab1,hTabN,NPar
      if(NMat.gt.NMatD) then
        write(*,*) 'Dimension in NMatD is exceeded'
        stop
      end if
      hTab1=-amin1(abs(hTab1),abs(hTabN))
      hTabN=-amax1(abs(hTab1),abs(hTabN))
      read (30,*)
      write(50,110)
      do 11 M=1,NMat
        read (30,*)     (Par(i,M),i=1,NPar)
        write(50,120) M,(Par(i,M),i=1,NPar)
11    continue
      write(50,130)
      do 13 M=1,NMat
        write(50,*)
        do 12 i=1,Imax
          h=FH(Qe(i),Par(1,M))
          K=FK(h,Par(1,M))
          C=FC(h,Par(1,M))
          Q=FQ(h,Par(1,M))
          write(50,140) M,Qe(i),Q,h,C,K
12      continue
13    continue

110   format(/' MatNum,        thR     thS     tha     thm       alpha
     !   n          Ks          Kk          thk'/)
120   format(i5,8x,4f7.3,16e12.3)
130   format(//' MatNum          Qe      Q         h          C          K')
140   format(i5,8x,2f7.3,f10.3,e10.2,e12.3)
      return
      end

*************************************************************************

      subroutine GenMat(NTab,NTabD,NMat,thR,hSat,Par,hTab,ConTab,CapTab,
     !                  ConSat,TheTab,thSat)

      dimension Par(10,NMat),ConTab(NTabD,NMat),CapTab(NTabD,NMat),
     !          TheTab(NTabD,NMat),hTab(NTab),ConSat(NMat),hSat(NMat),
     !          thR(NMat),thSat(NMat)

      write(*,*) 'generating materials'
      hTab1=hTab(1)
      hTabN=hTab(NTab)
      dlh=(alog10(-hTabN)-alog10(-hTab1))/(NTab-1)
      do 11 i=1,NTab
        alh=alog10(-hTab1)+(i-1)*dlh
        hTab(i)=-10**alh
11    continue
      do 13 M=1,NMat
        Hr       =FH(0.0,Par(1,M))
        hSat(M)  =FH(1.0,Par(1,M))
        ConSat(M)=FK(0.0,Par(1,M))
        thR(M)   =FQ(Hr ,Par(1,M))
        thSat(M) =FQ(0.0,Par(1,M))
        do 12 i=1,NTab
          ConTab(i,M)=FK(hTab(i),Par(1,M))
          CapTab(i,M)=FC(hTab(i),Par(1,M))
          TheTab(i,M)=FQ(hTab(i),Par(1,M))
12      continue
13    continue
      return
      end
```

```
************************************************************************

      subroutine TmIn(tInit,tMax,tAtm,tOld,dt,dtMax,dMul,dMul2,dtMin,
     !                TPrint,t,dtOpt,dtOld,AtmInF)

      logical AtmInF
      dimension TPrint(50)

      write(*,*) 'reading time information'
      read(30,*)
      read(30,*)
      read(30,*) dt,dtMin,dtMax,dMul,dMul2,MPL
      read(30,*)
      read(30,*) (TPrint(i),i=1,MPL)
      dtOpt=dt
      dtOld=dt
      if(.not.AtmInF) then
        tMax=TPrint(MPL)
        tAtm=tMax
      end if
      TPrint(MPL+1)=tMax
      tOld=tInit
      t=tInit+dt
      return
      end

************************************************************************

      subroutine SeepIn(NSeepD,NumSPD,NSeep,NSP,NP)

      dimension NSP(NSeepD),NP(NSeepD,NumSPD)

      write(*,*) 'reading seepage face information'
      read(30,*)
      read(30,*)
      read(30,*) NSeep
      if(NSeep.gt.NSeepD) then
        write(*,*) 'Dimension in NSeepD is exceeded'
        stop
      end if
      read(30,*)
      read(30,*) (NSP(i),i=1,NSeep)
      do 11 i=1,NSeep
        if(NSP(i).gt.NumSPD) then
          write(*,*) 'Dimension in NumSPD is exceeded'
          stop
        end if
11    continue
      read(30,*)
      do 12 i=1,NSeep
        read(30,*) (NP(i,j),j=1,NSP(i))
12    continue
      return
      end

************************************************************************

      subroutine DrainIn(NDr,NDrD,NElDrD,NumEl,ND,NED,KElDr,EfDim,
     !                   ConAxx,ConAxz,ConAzz)

      integer e
      dimension ND(NDrD),NED(NDrD),KElDr(NDrD,NElDrD),EfDim(2,NDrD),
     !          ConAxx(NumEl),ConAzz(NumEl),ConAxz(NumEl)

      write(*,*) 'reading drainage information'
      read(30,*)
      read(30,*)
      read(30,*) NDr,DrCorr
      if(NDr.gt.NDrD) then
        write(*,*) 'Dimension in NDrD is exceeded'
        stop
      end if
      read(30,*)
```

```
      read(30,*) (ND(i),i=1,NDr)
      read(30,*)
      read(30,*) (NED(i),i=1,NDr)
      do 11 i=1,NDr
        if(NED(i).gt.NElDrD) then
          write(*,*) 'Dimension in NElDrD is exceeded'
          stop
        end if
11    continue
      read(30,*)
      do 12 i=1,NDr
        read(30,*) (EfDim(i,j),j=1,2)
12    continue
      read(30,*)
      do 13 i=1,NDr
        read(30,*) (KElDr(i,j),j=1,NED(i))
13    continue
      do 15 i=1,Ndr
        rho=EfDim(i,2)/EfDim(i,1)
        A=(1.+0.405*rho**(-4))/(1.-0.405*rho**(-4))
        B=(1.+0.163*rho**(-8))/(1.-0.163*rho**(-8))
        C=(1.+0.067*rho**(-12))/(1.-0.067*rho**(-12))
        Red=376.7/(138.*alog10(rho)+6.48-2.34*A-0.48*B-0.12*C)/DrCorr
        do 14 j=1,NED(i)
          e=KElDr(i,j)
          ConAxx(e)=ConAxx(e)*Red
          ConAxz(e)=ConAxz(e)*Red
          ConAzz(e)=ConAzz(e)*Red
14      continue
15    continue
      return
      end

*************************************************************************

      subroutine NodInf(NumNP,NumEl,IJ,NumBP,NumNPD,NumElD,NumBPD,NumKD,
     !                  NObs,NObsD,Kode,Q,Conc,hNew,hOld,hTemp,x,y,
     !                  MatNum,Beta,Axz,Bxz,Dxz,CheckF)

      logical CheckF
      dimension Kode(NumNPD),Q(NumNPD),hOld(NumNPD),x(NumNPD),y(NumNPD),
     !          hNew(NumNPD),hTemp(NumNPD),MatNum(NumNPD),Beta(NumNPD),
     !          Axz(NumNPD),Bxz(NumNPD),Dxz(NumNPD),Conc(NumNPD)

      write(*,*) 'reading nodal information'
      read(32,*)
      read(32,*)
      read(32,*) NumNP,NumEl,IJ,NumBP,NObs
      if(NumNP.gt.NumNPD) then
        write(*,*) 'Dimension in NumNPD is exceeded'
        stop
      else if(NumEl.gt.NumElD) then
        write(*,*) 'Dimension in NumElD is exceeded'
        stop
      else if(NumBP.gt.NumBPD) then
        write(*,*) 'Dimension in NumBPD is exceeded'
        stop
      else if(NObs.gt.NObsD) then
        write(*,*) 'Dimension in NObsD is exceeded'
        stop
      end if
      read(32,*)
      NPR=0
      k=0

11    k=k+1
      read(32,*) n,Kode(n),x(n),y(n),hOld(n),Conc(n),Q(n),MatNum(n),
     !           Beta(n),Axz(n),Bxz(n),Dxz(n)
      if(Kode(n).gt.NumKD) then
        write(*,*) 'Dimension in NumKD is exceeded'
        stop
      end if
      if(n-k) 12,15,13
```

```
12    write(*,130) n
      stop
13    Deno=n-k+1
      DX=(x(n)-x(NPR))/Deno
      DY=(y(n)-y(NPR))/Deno
      DP=(hOld(n)-hOld(NPR))/Deno
      DConc=(Conc(n)-Conc(NPR))/Deno
      DBeta=(Beta(n)-Beta(NPR))/Deno
      DA=(Axz(n)-Axz(NPR))/Deno
      DB=(Bxz(n)-Bxz(NPR))/Deno
      DD=(Dxz(n)-Dxz(NPR))/Deno
14    x(k)=x(k-1)+DX
      y(k)=y(k-1)+DY
      hOld(k)=hOld(k-1)+DP
      Conc(k)=Conc(k-1)+DConc
      Beta(k)=Beta(k-1)+DBeta
      Axz(k)=Axz(k-1)+DA
      Bxz(k)=Bxz(k-1)+DB
      Dxz(k)=Dxz(k-1)+DD
      MatNum(k)=MatNum(k-1)
      Kode(k)=Kode(k-1)
      Q(k)=Q(k-1)
      k=k+1
      if(k.lt.N) goto 14
15    NPR=N
      if(k.lt.NumNP) goto 11

      do 16 n=1,NumNP
        hNew(n)=hOld(n)
        hTemp(n)=hOld(n)
16    continue
      if(CheckF) then
        write(50,110)
        do 17 n=1,NumNP
          write(50,120) n,Kode(n),x(n),y(n),hOld(n),Q(n),Conc(n),
     !                  MatNum(n),Beta(n),Axz(n),Bxz(n),Dxz(n)
17      continue
      end if

110   format(////' NODAL POINT INFORMATION'//'  NODE NO.',6x,'KODE',
     !       7x,'X,R',12x,'Y,Z',11x,'.PSI.',12x,'Q',11x,'Conc'/)
120   format(2i10,5e15.6,i10,4f7.3)
130   format(' ERROR IN NodInf AT N=',i5)
      return
      end

*************************************************************************

      subroutine ElemIn(NumEl,NumElD,NumNP,KX,LayNum,ConAxx,ConAzz,
     !                  ConAxz,CheckF,ListNE,IJ,MBand,MBandD,lChem,lOrt)

      logical CheckF,lChem,lConst,lOrt
      integer e
      dimension KX(NumElD,4),ConAxx(NumEl),ConAzz(NumEl),ConAxz(NumEl),
     !          LayNum(NumEl),ListNE(NumNP)

      write(*,*) 'reading element information'
      Num=0
      read(32,*)
      read(32,*)
      do 14 e=1,NumEL
        IF (Num-e) 11,14,12
11      read(32,*) Num,(KX(Num,i),i=1,4),ConAxz(Num),ConAxx(Num),
     !             ConAzz(Num),LayNum(Num)
        if(KX(Num,4).eq.0) KX(Num,4)=KX(Num,3)
        if(Num.eq.e) goto 14
12      do 13 i=1,4
          KX(e,i)=KX(e-1,i)+1
13      continue
        ConAxx(e)=ConAxx(e-1)
        ConAzz(e)=ConAzz(e-1)
        ConAxz(e)=ConAxz(e-1)
        LayNum(e)=LayNum(e-1)
```

```
14    continue
      AA=3.141592654/180.
      do 15 e=1,NumEl
        Ang=AA*ConAxz(e)
        CAxx=ConAxx(e)
        CAzz=ConAzz(e)
        ConAxx(e)=CAxx*cos(Ang)*cos(Ang) + CAzz*sin(Ang)*sin(Ang)
        ConAzz(e)=CAxx*sin(Ang)*sin(Ang) + CAzz*cos(Ang)*cos(Ang)
        ConAxz(e)=(CAxx-CAzz)*sin(Ang)*cos(Ang)
15    continue
      if(CheckF) then
        write(50,110)
        do 16 e=1,NumEl
          write(50,120) e,(KX(e,i),i=1,4),ConAxz(e),ConAxx(e),ConAzz(e),
     !                  LayNum(e)
16      continue
      end if

      do 17 i=1,NumNP
        ListNE(i)=0
17    continue
      do 19 e=1,NumEl
        NCorn=4
        if(KX(e,3).eq.KX(e,4)) NCorn=3
        do 18 n=1,NCorn-2
          i=KX(e,1)
          j=KX(e,n+1)
          k=KX(e,n+2)
          ListNE(i)=ListNE(i)+1
          ListNE(j)=ListNE(j)+1
          ListNE(k)=ListNE(k)+1
18      continue
19    continue

      lOrt=.false.
      lConst=.true.
      MBand=1
      do 21 e=1,NumEl
        NUS=4
        if(KX(e,3).eq.KX(e,4)) NUS=3
        do 20 kk=1,NUS-2
          MB=1
          i=KX(e,1)
          j=KX(e,kk+1)
          k=KX(e,kk+2)
          if(abs(i-j).gt.MB) MB=abs(i-j)
          if(abs(i-k).gt.MB) MB=abs(i-k)
          if(abs(j-k).gt.MB) MB=abs(j-k)
          if(MB.gt.MBand) MBand=MB
          if(e.eq.1.and.kk.eq.1) then
            MB1=MB
          else
            if(MB1.ne.MB) lConst=.false.
          end if
20      continue
21    continue
      MBand=MBand+1
      if(MBand.gt.MBandD.or.(lChem.and.2*MBand-1.gt.MBandD)) lOrt=.true.
      if(.not.lConst) IJ=NumNP
      if(MBand.gt.10.or.NumNP.gt.200) lOrt=.true.

110   format (////' Element Information'//' Element    C O R N E R    N
     !O D E S   ConAxz    ConAxx    ConAzz  LayNum'/)
120   format (i6,i9,3i6,e14.3,2f8.3,i5)
      return
      end

*************************************************************************

      subroutine GeomIn(NumKD,NumNP,NumBP,NObs,SWidth,Width,Kode,KXB,
     !                  rLen,Node)

      character*50 Text1
```

```
      dimension KXB(NumBP),Width(NumBP),SWidth(NumKD),Kode(NumNP),
     !          Node(NObs)

      write(*,*) 'reading geometric information'
      read(32,*)
      read(32,*)
      read(32,*) (KXB(i),i=1,NumBP)
      read(32,*)
      read(32,*) (Width(i),i=1,NumBP)
      read(32,*)
      read(32,*) rLen
      do 11 i=1,NumKD
        SWidth(i)=0.
11    continue
      do 12 i=1,NumBP
        n=KXB(i)
        j=iabs(Kode(n))
        if(j.eq.0) goto 12
        SWidth(j)=SWidth(j)+Width(i)
12    continue
      if(NObs.gt.0) then
        read(32,*)
        read(32,*) (Node(i),i=1,NObs)
        Text1='    hNew     theta     conc     '
        write(92,110) (Node(j),j=1,NObs)
        write(92,120) (Text1,i=1,NObs)
      end if
110   format (///14x,5(11x,'Node(',i3,')',11x))
120   format (/'     time     ',5(a31))
      return
      end

*************************************************************************

      subroutine AtmIn(GWLOL,SinkF,qGWLF,tInit,tMax,Aqh,Bqh,hCritS,
     !                 MaxAL)

      logical SinkF,qGWLF

      write(*,*) 'reading atmospheric information'
      read(31,*)
      read(31,*)
      read(31,*)
      read(31,*)
      read(31,*) SinkF,qGWLF
      read(31,*)
      read(31,*) GWLOL,Aqh,Bqh
      read(31,*)
      read(31,*) tInit,MaxAL
      read(31,*)
      read(31,*) hCritS
      read(31,*)
      do 11 i=1,MaxAL-1
        read(31,*)
11    continue
      read(31,*) tMax
      rewind 31
      do 12 i=1,12
        read(31,*)
12    continue
      return
      end

*************************************************************************

      subroutine SinkIn(NMat,NumEl,NumNP,NumElD,KAT,KX,x,y,PO,POptm,P2H,
     !                  P2L,P3,r2H,r2L,Beta)

      integer e
      dimension POptm(NMat),Beta(NumNP),KX(NumElD,4),x(NumNP),y(NumNP)

      write(*,*) 'reading sink information'
      read(30,*)
```

```
      read(30,*)
      read(30,*) P0,P2H,P2L,P3,r2H,r2L
      read(30,*)
      read(30,*) (POptm(i),i=1,NMat)
      P0 =-abs(P0)
      P2L=-abs(P2L)
      P2H=-abs(P2H)
      P3 =-abs(P3)
      xMul=1.
      SBeta=0.
      do 12 e=1,NumEl
        NUS=4
        IF(KX(e,3).eq.KX(e,4)) NUS=3
        do 11 k=1,NUS-2
          i=KX(e,1)
          j=KX(e,k+1)
          l=KX(e,k+2)
          CJ=x(i)-x(l)
          CK=x(j)-x(i)
          BJ=y(l)-y(i)
          BK=y(i)-y(j)
          AE=(CK*BJ-CJ*BK)/2.
          if(KAT.eq.1) xMul=2.*3.1416*(x(i)+x(j)+x(l))/3.
          BetaE=(Beta(i)+Beta(j)+Beta(l))/3.
          SBeta=SBeta+xMul*AE*BetaE
11      continue
12    continue
      do 13 i=1,NumNP
        Beta(i)=Beta(i)/SBeta
13    continue
      return
      end

*************************************************************************

      subroutine ChemIn(NMat,NumBP,cBound,ChPar,epsi,tPulse,KodCB,
     !                  NLevel,lUpW,lArtD,PeCr)

      logical lUpW,lArtD
      dimension ChPar(10,NMat),KodCB(NumBP),cBound(6)

      write(*,*) 'reading transport information'
      NLevel=1
      write(50,110)
      read(30,*)
      read(30,*)
      read(30,*) epsi,lUpW,lArtD,PeCr
      PeCr=amax1(PeCr,0.1)
      if(epsi.lt.0.999) NLevel=2
      read(30,*)
      if(lUpW) then
        write(50,120)
      else
        write(50,130)
        if(lArtD) write(50,140) PeCr
      end if
      write(50,150)
      do 11 M=1,NMat
        read(30,*) (ChPar(j,M),j=1,9)
        write(50,160) (ChPar(j,M),j=1,9)
11    continue
      read(30,*)
      read(30,*) (KodCB(i),i=1,NumBP)
      read(30,*)
      read(30,*) (cBound(i),i=1,6)
      write(50,170) (cBound(i),i=1,6)
      read(30,*)
      read(30,*) tPulse
      write(50,180) tPulse

110   format (/' Solute transport information'/1x,28('='))
120   format (/' Upstream weighting finite-element method')
130   format (/' Galerkin finite-element method')
```

```
140   format (/' Artificial dispersion is added when Peclet number is',
     !        ' higher than',f10.3)
150   format (/'  Bulk.d.   Difus.          Disper.        Adsorp.    ',
     !        'SinkL1    SinkS1    SinkL0    SinkS0')
160   format (10e10.3)
170   format (/'   Conc1     Conc2     Conc3     Conc4     cSink      ',
     !        'cWell'/8e10.3)
180   format (/' tPulse =    ',f15.3)
      return
      end

* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
* Source file WATFLOW2.FOR !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

      subroutine WatFlow(NumNP,NumEl,NumElD,NTab,NTabD,MBand,MBandD,
     !                   NMat,NSeep,NSeepD,NumSPD,NSP,NP,NumBP,ItCum,
     !                   MaxIt,Iter,Kode,KAT,t,dt,dtMin,dtOpt,dtOld,
     !                   tOld,hCritA,hCritS,TolTh,TolH,rLen,Width,rTop,
     !                   vMeanR,hMeanR,AtmInf,SinkF,SeepF,qGWLF,FreeD,
     !                   Par,hTab,ConTab,CapTab,TheTab,hNew,hOld,hTemp,
     !                   thR,thSat,ThNew,ThOld,MatNum,Con,Cap,ConSat,
     !                   Axz,Bxz,Dxz,hSat,A,B,Q,F,x,y,KX,Sink,DS,Beta,
     !                   ConAxx,ConAzz,ConAxz,KXB,Explic,GWLOL,Aqh,Bqh,
     !                   lWat,TLevel,lOrt,DrainF,ND,NDr,rRoot,PO,POptm,
     !                   P2H,P2L,P3,r2H,r2L,ConO,
     !                   A1,B1,NumNPD,IAD,IADN,IADD,VRV,RES,RQI,RQ,QQ,
     !                   QI,RQIDOT,ECNVRG,RCNVRG,ACNVRG,MNorth,MaxItO)

      logical AtmInf,SinkF,SeepF,Explic,ItCrit,lWat,qGWLF,FreeD,lOrt,
     !        DrainF
      double precision A,B,A1,B1,VRV,RES,RQI,RQ,QQ,QI,RQIDOT,ECNVRG,
     !                 RCNVRG,ACNVRG
      integer TLevel
      dimension A(MBandD,NumNP),B(NumNP),Kode(NumNP),Q(NumNP),F(NumNP),
     !          hNew(NumNP),hTemp(NumNP),hOld(NumNP),ConSat(NMat),
     !          hTab(NTab),ConTab(NTabD,NMat),CapTab(NTabD,NMat),
     !          Con(NumNP),Cap(NumNP),x(NumNP),y(NumNP),MatNum(NumNP),
     !          KX(NumElD,4),KXB(NumBP),Par(10,NMat),Width(NumBP),
     !          ConAxx(NumEl),ConAzz(NumEl),ConAxz(NumEl),hSat(NMat),
     !          NP(NSeepD,NumSPD),NSP(NSeepD),DS(NumNP),Beta(NumNP),
     !          Axz(NumNP),Bxz(NumNP),Dxz(NumNP),Sink(NumNP),thR(NMat),
     !          thSat(NMat),TheTab(NTabD,NMat),ThNew(NumNP),ND(NDr),
     !          ThOld(NumNP),POptm(NMat),ConO(NumNP),
     !          A1(MBandD,NumNP),B1(NumNP),RES(NumNP),IAD(MBandD,NumNP),
     !          IADN(NumNP),IADD(NumNP),VRV(NumNP),RQI(NumNPD,MNorth),
     !          RQ(NumNP),QQ(NumNP),RQIDOT(MNorth),QI(NumNPD,MNorth)

      if(lWat.and.TLevel.ne.1) then
        do 10 i=1,NumNP
          hOld(i) =hNew(i)
          ThOld(i)=ThNew(i)
          ConO(i)=Con(i)
          if(Kode(i).lt.1) then
            hTemp(i)=hNew(i)+(hNew(i)-hOld(i))*dt/dtOld
            hNew(i) =hTemp(i)
          else
            hTemp(i)=hNew(i)
          end if
10      continue
      end if

11    continue

      Iter=0
      Explic=.false.

* --- Beginning of iteration loop -----------------------------------

12    continue
      if(SinkF.and..not.lWat.and.Iter.ne.0)
     !  call SetSnk(NumNP,NMat,MatNum,hNew,rRoot,Sink,PO,POptm,P2H,P2L,
     !              P3,r2H,r2L,Beta,rLen)
      call SetMat(NumNP,NTab,NTabD,NMat,hTab,ConTab,CapTab,hNew,hOld,
     !            MatNum,Par,Con,Cap,ConSat,Axz,Bxz,Dxz,hSat,hTemp,
     !            Explic,TheTab,thSat,thR,ThNew)
      call Reset (A,B,Kode,Q,hNew,F,Con,Cap,x,y,KX,NumNP,NumEl,NumElD,
     !            MBand,MBandD,KAT,dt,Iter,SinkF,Sink,DS,Beta,rLen,
     !            vMeanR,hMeanR,ConAxx,ConAzz,ConAxz,ThNew,ThOld,lWat,
     !            lOrt,IAD,IADN,IADD,B1)
      call Shift (NumNP,NumBP,NSeepD,NumSPD,NSeep,NSP,NP,hNew,hOld,Q,
     !            Kode,rTop,Width,KXB,hCritA,hCritS,SeepF,AtmInF,Explic,
     !            qGWLF,FreeD,GWLOL,Aqh,Bqh,ConO,DrainF,ND,NDr,lWat)
      call Dirich(A,B,Kode,hNew,NumNP,MBand,MBandD,lOrt,IADD)
      if(lOrt) then
        call ILU (A,NumNP,MBandD,IAD,IADN,IADD,A1)
```

```
        call OrthoMin(A,B1,B,NumNP,MBandD,NumNPD,IAD,IADN,IADD,A1,VRV,
     !                RES,RQI,RQ,QQ,QI,RQIDOT,ECNVRG,RCNVRG,ACNVRG,0,
     !                MNorth,MaxItO)
      else
        call Solve (A,B,NumNP,MBand,MBandD)
      end if

      do 13 i=1,NumNP
        hTemp(i)=hNew(i)
        if(lOrt) B(i)=B1(i)
        hNew(i)=sngl(B(i))
13    continue
      Iter =Iter+1
      ItCum=ItCum+1
      if(Explic) goto 18

*     Test for convergence
      ItCrit=.true.
      do 14 i=1,NumNP
        m=MatNum(i)
        EpsTh=0.
        EpsH=0.
        if(hTemp(i).lt.hSat(m).and.hNew(i).lt.hSat(m)) then
          Th=ThNew(i)+Cap(i)*(hNew(i)-hTemp(i))/(ThSat(m)-ThR(m))/Axz(i)
          EpsTh=abs(ThNew(i)-Th)
        else
          EpsH=abs(hNew(i)-hTemp(i))
        end if
        if(EpsTh.gt.TolTh.or.EpsH.gt.TolH) then
          ItCrit=.false.
          goto 15
        end if
14    continue
15    continue

      if(.not.ItCrit) then
        if(Iter.lt.MaxIt.or.(.not.lWat.and.Iter.lt.5*MaxIt)) then
          goto 12
        else if(dt.le.dtMin) then
          Explic=.true.
          do 16 i=1,NumNP
            hNew(i) =hOld(i)
            hTemp(i)=hOld(i)
16        continue
          goto 12
        else if(.not.lWat) then
          write(*,*) ' No steady state solution found'
          stop
        else
          do 17 i=1,NumNP
            hNew(i) =hOld(i)
            hTemp(i)=hOld(i)
17        continue
          dt=amax1(dt/3.,dtMin)
          dtOpt=dt
          t=tOld+dt
          goto 11
        end if
      end if

* --- End of iteration loop ------------------------------------------

18    continue

      if(.not.lWat.and.TLevel.eq.1) then
        write(*,101) Iter
        do 19 i=1,NumNP
          hOld(i) =hNew(i)
          ThOld(i)=ThNew(i)
19      continue
      end if

101   format('Steady state was reached after',i6,' iterations.')
```

```
      return
      end

*****************************************************************************

      subroutine Reset(A,B,Kode,Q,hNew,F,Con,Cap,x,y,KX,NumNP,NumEl,
     !                 NumElD,MBand,MBandD,KAT,dt,Iter,SinkF,Sink,DS,
     !                 Beta,rLen,vMeanR,hMeanR,ConAxx,ConAzz,ConAxz,
     !                 ThNew,ThOld,lWat,lOrt,IAD,IADN,IADD,B1)

      logical SinkF,lWat,lOrt
      double precision A,B,B1
      dimension A(MBandD,NumNP),B(NumNP),Q(NumNP),hNew(NumNP),F(NumNP),
     !          Con(NumNP),Cap(NumNP),x(NumNP),y(NumNP),KX(NumElD,4),
     !          ConAxx(NumEl),ConAzz(NumEl),ConAxz(NumEl),Kode(NumNP),
     !          Sink(NumNP),DS(NumNP),Beta(NumNP),ThNew(NumNP),
     !          ThOld(NumNP),E(3,3),iLoc(3),B1(NumNP),IAD(MBandD,NumNP),
     !          IADN(NumNP),IADD(NumNP)

*     Initialisation
      xMul=1.
      if(Iter.eq.0) then
        vMeanR=0.
        hMeanR=0.
        AreaR =0.
      end if
      do 12 i=1,NumNP
        B(i)=0.
        if(lOrt) B1(i)=hNew(i)
        F(i)=0.
        if(Iter.eq.0) DS(i)=0.
        do 11 j=1,MBandD
          A(j,i)=0.
11      continue
12    continue

*     Loop on elements
      do 16 n=1,NumEl
        CondI=ConAxx(n)
        CondJ=ConAzz(n)
        CondK=ConAxz(n)
        NUS=4
        if(KX(n,3).eq.KX(n,4)) NUS=3

*       Loop on subelements
        do 15 k=1,NUS-2
          i=KX(N,1)
          j=KX(N,k+1)
          l=KX(N,k+2)
          iLoc(1)=1
          iLoc(2)=k+1
          iLoc(3)=k+2
          Ci=x(l)-x(j)
          Cj=x(i)-x(l)
          Ck=x(j)-x(i)
          Bi=y(j)-y(l)
          Bj=y(l)-y(i)
          Bk=y(i)-y(j)
          AE=(Ck*Bj-Cj*Bk)/2.
          CapE=(Cap(i)+Cap(j)+Cap(l))/3.
          ConE=(Con(i)+Con(j)+Con(l))/3.
          if(KAT.eq.1) xMul=2.*3.1416*(x(i)+x(j)+x(l))/3.
          AMul=xMul*ConE/4./AE
          BMul=xMul*ConE/2.
          FMul=xMul*AE/12.
          BetaE=(Beta(i)+Beta(j)+Beta(l))/3.
          if(SinkF.and.BetaE.gt.0..and.Iter.eq.0) then
            SinkE=(Sink(i)+Sink(j)+Sink(l))/3.
            DS(i)=DS(i)+FMul*(3.*SinkE+Sink(i))
            DS(j)=DS(j)+FMul*(3.*SinkE+Sink(j))
            DS(l)=DS(l)+FMul*(3.*SinkE+Sink(l))
            hNewE=(hNew(i)+hNew(j)+hNew(l))/3.
            vMeanR=vMeanR+xMul*AE*SinkE/rLen
```

```
            hMeanR=hMeanR+xMul*hNewE*AE
            AreaR=AreaR+xMul*AE
          end if
          F(i)=F(i)+FMul*4.
          F(j)=F(j)+FMul*4.
          F(l)=F(l)+FMul*4.
          if(KAT.ge.1) then
            B(i)=B(i)+BMul*(CondK*Bi+CondJ*Ci)
            B(j)=B(j)+BMul*(CondK*Bj+CondJ*Cj)
            B(l)=B(l)+BMul*(CondK*Bk+CondJ*Ck)
          end if
          E(1,1)=CondI*Bi*Bi+2.*CondK*Bi*Ci+CondJ*Ci*Ci
          E(1,2)=CondI*Bi*Bj+CondK*(Bi*Cj+Ci*Bj)+CondJ*Ci*Cj
          E(1,3)=CondI*Bi*Bk+CondK*(Bi*Ck+Ci*Bk)+CondJ*Ci*Ck
          E(2,1)=E(1,2)
          E(2,2)=CondI*Bj*Bj+2.*CondK*Bj*Cj+CondJ*Cj*Cj
          E(2,3)=CondI*Bj*Bk+CondK*(Bj*Ck+Bk*Cj)+CondJ*Cj*Ck
          E(3,1)=E(1,3)
          E(3,2)=E(2,3)
          E(3,3)=CondI*Bk*Bk+2.*CondK*Bk*Ck+CondJ*Ck*Ck
          do 14 i=1,3
            iG=KX(n,iLoc(i))
            do 13 j=1,3
              jG=KX(n,iLoc(j))
              if(lOrt) then
                call Find(iG,jG,kk,NumNP,MBandD,IAD,IADN)
                A(kk,iG)=A(kk,iG)+AMul*E(i,j)
              else
                iB=iG-jG+1
                if(iB.ge.1) A(iB,jG)=A(iB,jG)+AMul*E(i,j)
              end if
13          continue
14        continue
15      continue

16    continue
      if(AreaR.gt.0..and.Iter.eq.0) hMeanR=hMeanR/AreaR

*     Determine boundary fluxes
      do 19 n=1,NumNP
        if(Kode(n).lt.1) goto 19
        QN=B(n)+DS(n)+F(n)*(ThNew(n)-ThOld(n))/dt
        if(lOrt) then
          do 17 j=1,IADN(n)
            QN=QN+A(j,n)*hNew(IAD(j,n))
17        continue
        else
          QN=QN+A(1,n)*hNew(n)
          do 18 j=2,MBand
            k=n-j+1
            if(k.ge.1) then
              QN=QN+A(j,k)*hNew(k)
            end if
            k=n+j-1
            if(K.le.NumNP) then
              QN=QN+A(j,n)*hNew(k)
            end if
18        continue
        end if
        Q(n)=QN
19    continue

*     Complete construction of RHS vector and form effective matrix
      do 20 i=1,NumNP
        if(.not.lWat) F(i)=0.
        j=1
        if(lOrt) j=IADD(i)
        A(j,i)=A(j,i)+F(i)*Cap(i)/dt
        B(i)=F(i)*Cap(i)*hNew(i)/dt-F(i)*(ThNew(i)-ThOld(i))/dt+
     !       Q(i)-B(i)-DS(i)
20    continue
      return
      end
```

```
*************************************************************************

      subroutine Dirich(A,B,Kode,hNew,NumNP,MBand,MBandD,lOrt,IADD)

      logical lOrt
      double precision A,B
      dimension A(MBandD,NumNP),B(NumNP),Kode(NumNP),hNew(NumNP),
     !          IADD(NumNP)

      do 12 n=1,NumNP
        if(Kode(n).lt.1) goto 12
        if(lOrt) then
          A(IADD(n),n)=10.d30
          B(n)=10.d30*hNew(n)
        else
          do 11 m=2,MBand
            k=n-m+1
            if(k.gt.0) then
              B(k)=B(k)-A(m,k)*hNew(n)
              A(m,k)=0.
            end if
            l=n+m-1
            if(NumNP-l.ge.0) then
              B(l)=B(l)-A(m,n)*hNew(n)
            end if
            A(m,n)=0.
11        continue
          A(1,n)=1.
          B(n)=hNew(n)
        end if
12    continue
      return
      end

*************************************************************************

      subroutine Shift(NumNP,NumBP,NSeepD,NumSPD,NSeep,NSP,NP,hNew,hOld,
     !                 Q,Kode,EI,Width,KXB,hCritA,hCritS,SeepF,AtmInF,
     !                 Explic,qGWLF,FreeD,GWLOL,Aqh,Bqh,ConO,DrainF,ND,
     !                 NDr,lWat)

      logical SeepF,AtmInF,Explic,qGWLF,FreeD,DrainF,lWat
      dimension Kode(NumNP),Q(NumNP),hNew(NumNP),hOld(NumNP),
     !          Width(NumBP),KXB(NumBP),NP(NSeepD,NumSPD),NSP(NSeepD),
     !          ConO(NumNP),ND(NDr)

*     Modify conditions on seepage faces
      if(SeepF) then
        do 12 i=1,NSeep
          iCheck=0
          NS=NSP(i)
          do 11 j=1,NS
            n=NP(i,j)
            if(Kode(n).eq.-2) then
              if(hNew(n).lt.0.) then
                iCheck=1
              else
                Kode(n)=2
                hNew(n)=0.
              end if
            else
              if(iCheck.gt.0.or.Q(n).ge.0.) then
                Kode(n)=-2
                Q(n)=0.
                iCheck=1
              end if
            end if
11        continue
12      continue
      end if

*     Modify conditions in drainage node
      if(DrainF) then
```

```
        do 13 i=1,NDr
          n=ND(i)
          if(Kode(n).eq.-5) then
            if(hNew(n).ge.0.) then
              Kode(n)=5
              hNew(n)=0.
            end if
          else
            if(Q(n).ge.0.) then
              Kode(n)=-5
              Q(n)=0.
            end if
          end if
13      continue
      end if

*     Modify potential surface flux boundaries
      if(AtmInF) then
        do 14 i=1,NumBP
          n=KXB(i)
          k=Kode(n)
          if(Explic.and.iabs(k).eq.4) then
            Kode(n)=-iabs(k)
            goto 14
          end if

*         Critical surface pressure on ...
          if(K.eq.4) then
            if(abs(Q(n)).gt.abs(-EI*Width(i)).or.Q(n)*(-EI).le.0) then
              Kode(n)=-4
              Q(n)=-EI*Width(i)
            end if
            goto 14
          end if

*         Surface flux on ...
          if(K.eq.-4) then
            if(hNew(n).le.hCritA) then
              Kode(n)=4
              hNew(n)=hCritA
              goto 14
            end if
            if(hNew(n).ge.hCritS) then
              Kode(n)=4
              hNew(n)=hCritS
            end if
          end if

*         Time variable flux
          if(K.eq.-3) then
            if(lWat) then
              if(qGWLF) Q(n)=-Width(i)*Fqh(hOld(n)-GWLOL,Aqh,Bqh)
            else
              if(qGWLF) Q(n)=-Width(i)*Fqh(hNew(n)-GWLOL,Aqh,Bqh)
            end if
          end if
14      continue
      end if

*     Free Drainage
      if(FreeD) then
        do 15 i=1,NumBP
          n=KXB(i)
          k=Kode(n)
          if(k.eq.-3) Q(n)=-Width(i)*ConO(n)
15      continue
      end if
      return
      end

**************************************************************************

      subroutine SetMat(NumNP,NTab,NTabD,NMat,hTab,ConTab,CapTab,hNew,
```

```
!                 hOld,MatNum,Par,Con,Cap,ConSat,Axz,Bxz,Dxz,hSat,
!                 hTemp,Explic,TheTab,thSat,thR,theta)

      logical Explic
      dimension hTab(NTab),ConTab(NTabD,NMat),CapTab(NTabD,NMat),
!           hNew(NumNP),hOld(NumNP),MatNum(NumNP),Par(10,NMat),
!           Con(NumNP),Cap(NumNP),ConSat(NMat),hSat(NMat),
!           Axz(NumNP),Bxz(NumNP),Dxz(NumNP),hTemp(NumNP),
!           TheTab(NTabD,NMat),thSat(NMat),thR(NMat),theta(NumNP)

      alh1=alog10(-hTab(1))
      dlh =(alog10(-hTab(NTab))-alh1)/(NTab-1)
      do 11 i=1,NumNP
        M=MatNum(i)
        hi1=amin1(hSat(M),hTemp(i)/Axz(i))
        hi2=amin1(hSat(M), hNew(i)/Axz(i))
        if(Explic) hi2=hi1
        hiM=0.1*hi1+0.9*hi2
        if(hi1.ge.hSat(M).and.hi2.ge.hSat(M)) then
          Ci=ConSat(M)
        else if(hiM.ge.hTab(NTab).and.hiM.le.hTab(1)) then
          iT=int((alog10(-hiM)-alh1)/dlh)+1
          S1=(ConTab(iT+1,M)-ConTab(iT,M))/(hTab(iT+1)-hTab(iT))
          Ci=ConTab(iT,M)+S1*(hiM-hTab(iT))
        else
          Ci=FK(hiM,Par(1,M))
        end if
        Con(i)=Bxz(i)*Ci
        hi1=hOld(i)/Axz(i)
        hi2=hNew(i)/Axz(i)
        if(Explic) hi2=hi1
        if(hi2.ge.hSat(M)) then
          Ci=0
          Ti=thSat(M)
        else if(hi2.ge.hTab(NTab).and.hi2.le.hTab(1)) then
          iT=int((alog10(-hi2)-alh1)/dlh)+1
          S1=(CapTab(iT+1,M)-CapTab(iT,M))/(hTab(iT+1)-hTab(iT))
          S2=(TheTab(iT+1,M)-TheTab(iT,M))/(hTab(iT+1)-hTab(iT))
          Ci=CapTab(iT,M)+S1*(hi2-hTab(iT))
          Ti=TheTab(iT,M)+S2*(hi2-hTab(iT))
        else
          Ci=FC(hi2,Par(1,M))
          Ti=FQ(hi2,Par(1,M))
        end if
        Cap(i)=Ci*Dxz(i)/Axz(i)
        theta(i)=thR(M)+(Ti-thR(M))*Dxz(i)
11    continue
      return
      end

*************************************************************************

      subroutine Solve(A,B,NumNP,MBand,MBandD)

      double precision A,B,C
      dimension A(MBandD,NumNP),B(NumNP)

*     Reduction
      do 13 n=1,NumNP
        do 12 m=2,MBand
          if(abs(A(m,n)).lt.1.e-30) goto 12
          C=A(m,n)/A(1,n)
          i=n+m-1
          if(i.gt.NumNP) goto 12
          j=0
          do 11 k=m,MBand
            j=j+1
            A(j,i)=A(j,i)-C*A(k,n)
11        continue
          A(m,n)=C
          B(i)=B(i)-A(m,n)*B(n)
12      continue
        B(n)=B(n)/A(1,n)
```

```
13    continue

*     Back substitution
      n=NumNP
14    do 15 k=2,MBand
        l=n+k-1
        if(l.gt.NumNP) goto 16
        B(n)=B(n)-A(k,n)*B(l)
15    continue
16    n=n-1
      if(n.gt.0) goto 14
      return
      end

* ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

```
* Source file SOLUTE2.FOR ||||||||||||||||||||||||||||||||||||||||||||||||||||||||

      subroutine Solute(NumNP,NumEl,NumElD,MBand,MBandD,NMat,t,Kode,A,B,
     !                  Q,hNew,hOld,F,x,y,KX,KAT,dt,DS,Sink,MatNum,Con,
     !                  ConO,ConAxx,ConAzz,ConAxz,Vx,Vz,Dispxx,Dispzz,
     !                  Dispxz,ChPar,ThNew,ThOld,thSat,Ac,Fc,Gc,Qc,Conc,
     !                  ListNE,cBound,tPulse,NumBP,KodCB,KXB,NLevel,
     !                  cPrec,crt,cht,lWat,lUpW,WeTab,epsi,CumChO,
     !                  CumCh1,CumChR,dtMaxC,Peclet,Courant,lArtD,PeCr,
     !                  lOrt,A1,B1,NumNPD,IAD,IADN,IADD,VRV,RES,RQI,RQ,
     !                  QQ,QI,RQIDOT,ECNVRG,RCNVRG,ACNVRG,MNorth,MaxItO)

      double precision A,B,A1,B1,VRV,RES,RQI,RQ,QQ,QI,RQIDOT,ECNVRG,
     !                 RCNVRG,ACNVRG
      logical lWat,lUpW,lOrt,lArtD
      dimension A(MBandD,NumNP),B(NumNP),Q(NumNP),hNew(NumNP),F(NumNP),
     !          KX(NumElD,4),MatNum(NumNP),Sink(NumNP),DS(NumNP),
     !          x(NumNP),y(NumNP),Kode(NumNP),Vx(NumNP),Vz(NumNP),
     !          ThNew(NumNP),ThOld(NumNP),KXB(NumBP),Conc(NumNP),
     !          Con(NumNP),ConAxx(NumEl),ConAzz(NumEl),ConAxz(NumEl),
     !          Qc(NumNP),Ac(NumNP),Fc(NumNP),Gc(NumNP),KodCB(NumBP),
     !          Dispxx(NumNP),Dispzz(NumNP),Dispxz(NumNP),ListNE(NumNP),
     !          ChPar(10,NMat),cBound(6),thSat(NMat),S(3,3),Bi(3),Ci(3),
     !          List(3),Wx(3),Wz(3),WeTab(3,2*NumEl),VxE(3),VzE(3),
     !          ConO(NumNP),hOld(NumNP),
     !          A1(MBandD,NumNP),B1(NumNP),RES(NumNP),IAD(MBandD,NumNP),
     !          IADN(NumNP),IADD(NumNP),VRV(NumNP),RQI(NumNPD,MNorth),
     !          RQ(NumNP),QQ(NumNP),RQIDOT(MNorth),QI(NumNPD,MNorth)

*     Initialisation
      xMul=1.
      alf=1.-epsi
      jjj=MBand
      if(t.gt.tPulse) then
        do 11 i=1,4
          cBound(i)=0.
11      continue
      end if
      do 13 i=1,NumNP
        B(i) =0.
        if(lOrt) B1(i)=Conc(i)
        Qc(i)=0.
        if(epsi.lt.0.001) then
          if(lOrt) jjj=IADD(i)
          A(jjj,i)=0.
        else
          do 12 j=1,MBandD
            A(j,i)=0.
12        continue
        end if
13    continue

      do 21 Level=1,NLevel
        if(Level.eq.NLevel) then
          Eps=epsi
          if(lWat)
     !      call Veloc(KAT,NumNP,NumEl,NumElD,hNew,x,y,KX,ListNE,Con,
     !                 ConAxx,ConAzz,ConAxz,Vx,Vz)
          call Disper(NumNP,NMat,Dispxx,Dispzz,Dispxz,Vx,Vz,ThNew,thSat,
     !                ChPar,MatNum,lArtD,PeCr,dt)
          call PeCour(NumNP,NumEl,NumElD,NMat,x,y,Vx,Vz,KX,MatNum,
     !                Dispxx,Dispzz,ChPar,ThNew,dt,dtMaxC,Peclet,
     !                Courant,lUpW,lArtD,PeCr)
          if(lUpW.and.lWat)
     !      call WeFact(NumNP,NumEl,NumElD,x,y,KX,WeTab,Vx,Vz,Dispxx,
     !                  Dispzz,Dispxz)
        else
          Eps=1.-epsi
          call Disper(NumNP,NMat,Dispxx,Dispzz,Dispxz,Vx,Vz,ThNew,
     !                thSat,ChPar,MatNum,lArtD,PeCr,dt)
        end if
        do 14 i=1,NumNP
          M=MatNum(i)
```

```
          if(Level.ne.NLevel) then
            if(.not.lArtD.and..not.lUpW) then
              DPom=dt/6./(ThOld(i)+ChPar(1,M)*ChPar(5,M))
              Dispxx(i)=Dispxx(i)+Vx(i)*Vx(i)*DPom
              Dispzz(i)=Dispzz(i)+Vz(i)*Vz(i)*DPom
              Dispxz(i)=Dispxz(i)+Vx(i)*Vz(i)*DPom
            end if
          else
            Ac(i)=-(ThOld(i)*alf+ThNew(i)*epsi)-ChPar(1,M)*ChPar(5,M)
            if(.not.lArtD.and..not.lUpW) then
              DPom=dt/6./(ThNew(i)+ChPar(1,M)*ChPar(5,M))
              Dispxx(i)=Dispxx(i)-Vx(i)*Vx(i)*DPom
              Dispzz(i)=Dispzz(i)-Vz(i)*Vz(i)*DPom
              Dispxz(i)=Dispxz(i)-Vx(i)*Vz(i)*DPom
            end if
            cS=cBound(5)
            if(cS.gt.Conc(i)) cS=Conc(i)
            Gc(i)=ChPar(8,M)*ThNew(i)+ChPar(1,M)*ChPar(9,M)-
     !            Sink(i)*cS
            Fc(i)=ChPar(6,M)*ThNew(i)+ChPar(1,M)*ChPar(7,M)*ChPar(5,M)+
     !            Sink(i)
          end if
14      continue
        do 15 i=1,NumNP
          F(i)=0.
          if(Level.eq.NLevel) Ds(i)=0.
15      continue

*       Loop on elements
        NumSEl=0
        do 19 n=1,NumEl
          CAxx=ConAxx(n)
          CAzz=ConAzz(n)
          CAxz=ConAxz(n)
          NUS=4
          if(KX(n,3).eq.KX(n,4)) NUS=3

*         Loop on subelements
          do 18 k=1,NUS-2
            NumSEl=NumSEl+1
            i=KX(n,1)
            j=KX(n,k+1)
            l=KX(n,k+2)
            List(1)=i
            List(2)=j
            List(3)=l
            Ci(1)=x(l)-x(j)
            Ci(2)=x(i)-x(l)
            Ci(3)=x(j)-x(i)
            Bi(1)=y(j)-y(l)
            Bi(2)=y(l)-y(i)
            Bi(3)=y(i)-y(j)
            AE=(Ci(3)*Bi(2)-Ci(2)*Bi(3))/2.

*           Calculate Velocities
            AE1=1./AE/2.
            Ai=CAxx*Bi(1)+CAxz*Ci(1)
            Aj=CAxx*Bi(2)+CAxz*Ci(2)
            Ak=CAxx*Bi(3)+CAxz*Ci(3)
            if(Level.eq.NLevel) then
              Vxx=AE1*(Ai*hNew(i)+Aj*hNew(j)+Ak*hNew(l))
            else
              Vxx=AE1*(Ai*hOld(i)+Aj*hOld(j)+Ak*hOld(l))
            end if
            if(KAT.gt.0) Vxx=Vxx+CAxz
            Ai=CAxz*Bi(1)+CAzz*Ci(1)
            Aj=CAxz*Bi(2)+CAzz*Ci(2)
            Ak=CAxz*Bi(3)+CAzz*Ci(3)
            if(Level.eq.NLevel) then
              Vzz=AE1*(Ai*hNew(i)+Aj*hNew(j)+Ak*hNew(l))
            else
              Vzz=AE1*(Ai*hOld(i)+Aj*hOld(j)+Ak*hOld(l))
            end if
```

```
      if(KAT.gt.0) Vzz=Vzz+CAzz
      if(Level.ne.NLevel) then
        ConE=(ConO(i)+ConO(j)+ConO(l))/3.
        VxE(1)=-ConO(i)*Vxx
        VxE(2)=-ConO(j)*Vxx
        VxE(3)=-ConO(l)*Vxx
        VzE(1)=-ConO(i)*Vzz
        VzE(2)=-ConO(j)*Vzz
        VzE(3)=-ConO(l)*Vzz
      else
        ConE=(Con(i)+Con(j)+Con(l))/3.
        VxE(1)=-Con(i)*Vxx
        VxE(2)=-Con(j)*Vxx
        VxE(3)=-Con(l)*Vxx
        VzE(1)=-Con(i)*Vzz
        VzE(2)=-Con(j)*Vzz
        VzE(3)=-Con(l)*Vzz
      end if
      VxEE=-ConE*Vxx
      VzEE=-ConE*Vzz

      if(KAT.eq.1) xMul=2.*3.1416*(x(i)+x(j)+x(l))/3.
      if(Level.eq.1) then
        cS=cBound(5)
        if(cBound(5).gt.conc(i)) cS=cS+(conc(i)-cBound(5))/3.
        if(cBound(5).gt.conc(j)) cS=cS+(conc(j)-cBound(5))/3.
        if(cBound(5).gt.conc(l)) cS=cS+(conc(l)-cBound(5))/3.
        RootCh=xMul*AE*dt*cS*(Sink(i)+Sink(j)+Sink(l))/3.
        CumCh0=CumCh0-xMul*AE*dt*(Gc(i)+Gc(j)+Gc(l))/3.+RootCh
        CumCh1=CumCh1-xMul*AE*dt*((Fc(i)-Sink(i))*conc(i)+
     !        (Fc(j)-Sink(j))*conc(j)+(Fc(l)-Sink(l))*conc(l))/3.
        CumChR=CumChR+RootCh
      end if
      FMul=xMul*AE/4.
      GcE=(Gc(i)+Gc(j)+Gc(l))/3.
      Ec1=(Dispxx(i)+Dispxx(j)+Dispxx(l))/3.
      Ec2=(Dispxz(i)+Dispxz(j)+Dispxz(l))/3.
      Ec3=(Dispzz(i)+Dispzz(j)+Dispzz(l))/3.
      if(Level.eq.NLevel) AcE=(Ac(i)+Ac(j)+Ac(l))/3.
      FcE=(Fc(i)+Fc(j)+Fc(l))/3.
      SMul1=-1./AE/4.*xMul
      SMul2=AE/20.*xMul
      if(lUpW) then
        NS=NumSEl
        W1=WeTab(1,NS)
        W2=WeTab(2,NS)
        W3=WeTab(3,NS)
        Wx(1)=2*VxE(1)*(W2-W3)+VxE(2)*(W2-2.*W3)+VxE(3)*(2.*W2-W3)
        Wx(2)=VxE(1)*(2.*W3-W1)+2*VxE(2)*(W3-W1)+VxE(3)*(W3-2.*W1)
        Wx(3)=VxE(1)*(W1-2.*W2)+VxE(2)*(2.*W1-W2)+2*VxE(3)*(W1-W2)
        Wz(1)=2*VzE(1)*(W2-W3)+VzE(2)*(W2-2.*W3)+VzE(3)*(2.*W2-W3)
        Wz(2)=VzE(1)*(2.*W3-W1)+2*VzE(2)*(W3-W1)+VzE(3)*(W3-2.*W1)
        Wz(3)=VzE(1)*(W1-2.*W2)+VzE(2)*(2.*W1-W2)+2*VzE(3)*(W1-W2)
      end if
      do 17 j1=1,3
        i1=List(j1)
        F(i1)=F(i1)+FMul*(GcE+Gc(i1)/3.)
        if(Level.eq.NLevel) DS(i1)=DS(i1)+FMul*(AcE+Ac(i1)/3.)
        iBound=0
        if(Kode(i).ne.0) then
          do 24 id=1,NumBP
            if(KXB(id).eq.i1.and.KodCB(id).gt.0) iBound=1
24        continue
        end if
        if(iBound.eq.1) Qc(i1)=Qc(i1)-Eps*FMul*(GcE+Gc(i1)/3.)
        do 16 j2=1,3
          i2=List(j2)
          S(j1,j2)=SMul1*(Ec1*Bi(j1)*Bi(j2)+Ec3*Ci(j1)*Ci(j2)+
     !                    Ec2*(Bi(j1)*Ci(j2)+Ci(j1)*Bi(j2)))
          S(j1,j2)=S(j1,j2)-(Bi(j2)/8.*(VxEE+VxE(j1)/3.)+
     !                       Ci(j2)/8.*(VzEE+VzE(j1)/3.))*xMul
          if(lUpW) S(j1,j2)=S(j1,j2)-xMul*
     !                       (Bi(j2)/40.*Wx(j1)+Ci(j2)/40.*Wz(j1))
```

```
                ic=1
                if(i1.eq.i2) ic=2
                S(j1,j2)=S(j1,j2)+SMul2*ic*(FcE+(Fc(i1)+Fc(i2))/3.)
                if(Level.ne.NLevel) then
                  B(i1)=B(i1)-alf*S(j1,j2)*Conc(i2)
                else
                  if(lOrt) then
                    call Find(i1,i2,kk,NumNP,MBandD,IAD,IADN)
                    iB=kk
                  else
                    iB=MBand+i2-i1
                  end if
                  A(iB,i1)=A(iB,i1)+epsi*S(j1,j2)
                end if
                if(iBound.eq.1) Qc(i1)=Qc(i1)-Eps*S(j1,j2)*Conc(i2)
16            continue
17          continue
18        continue
19      continue

        do 20 i=1,NumNP
          M=MatNum(i)
          if(Level.ne.NLevel) then
            B(i)=B(i)-alf*F(i)
          else
            if(lOrt) jjj=IADD(i)
            A(jjj,i)=A(jjj,i)+DS(i)/dt
            B(i)=B(i)+DS(i)/dt*Conc(i)-epsi*F(i)
          end if
20      continue
21    continue

*     Boundary condition
      call c_Bound(NumNP,MBand,MBandD,NumBP,A,B,Q,Qc,Conc,Kode,KXB,
     !             KodCB,cBound,cPrec,crt,cht,epsi,dt,DS,lOrt,IADD)

*     Solve the global matrix equation for transport
      if(epsi.lt.0.001) then
        do 22 i=1,NumNP
          if(lOrt) jjj=IADD(i)
          B(i)=B(i)/A(jjj,i)
22      continue
      else
        if(lOrt) then
          call ILU     (A,NumNP,MBandD,IAD,IADN,IADD,A1)
          call OrthoMin(A,B1,B,NumNP,MBandD,NumNPD,IAD,IADN,IADD,A1,VRV,
     !                  RES,RQI,RQ,QQ,QI,RQIDOT,ECNVRG,RCNVRG,ACNVRG,4,
     !                  MNorth,MaxItO)
        else
          call SolveT(A,B,MBand,MBandD,NumNP)
        end if
      end if
      do 23 i=1,NumNP
        if(lOrt) B(i)=B1(i)
        Conc(i)=sngl(B(i))
23    continue
      return
      end

*****************************************************************************

      subroutine c_Bound(NumNP,MBand,MBandD,NumBP,A,B,Q,Qc,Conc,Kode,
     !                   KXB,KodCB,cBound,cPrec,crt,cht,epsi,dt,DS,lOrt,
     !                   IADD)

      double precision A,B
      integer cKod
      logical lOrt
      dimension A(MBandD,NumNP),B(NumNP),Q(NumNP),Conc(NumNP),
     !          Qc(NumNP),Kode(NumNP),KXB(NumBP),KodCB(NumBP),cBound(6),
     !          DS(NumNP),IADD(NumNP)

      alf=1.-epsi
```

```
      jjj=MBand
      do 14 i=1,NumNP
        if(Kode(i).ne.0) then
          do 11 j=1,NumBP
            if(KXB(j).eq.i) then
              if(KodCB(j).gt.0) then
                cKod=1
                if(abs(Kode(i)).le.2.or.abs(Kode(i)).ge.5)
     !                                        cBnd=cBound(KodCB(j))
                if(abs(Kode(i)).eq.3) cBnd=cht
                if(abs(Kode(i)).eq.4) cBnd=cPrec
              else
                if(Q(i).gt.0.) then
                  cKod=3
                  if(abs(Kode(i)).eq.1.or.abs(Kode(i)).ge.5)
     !                                        cBnd=cBound(-KodCB(j))
                  if(abs(Kode(i)).eq.3) cBnd=crt
                  if(abs(Kode(i)).eq.4) cBnd=cPrec
                else
                  cKod=2
                  if(Kode(i).eq.-4) then
                    cKod=3
                    cBnd=0.
                  end if
                end if
              end if
              if(abs(Kode(i)).eq.2) cKod=2
              goto 12
            end if
11        continue

*     point source or sink
          if(Q(i).lt.0.) then
            cKod=2
          else
            cBnd=cBound(6)
            cKod=3
          end if

12        continue

*       Dirichlet boundary condition
          if(cKod.eq.1) then
            Qc(i)=Qc(i)+Q(i)*(epsi*cBnd+alf*Conc(i))-
     !                          DS(i)*(cBnd-Conc(i))/dt
            if(lOrt) then
              A(IADD(i),i)=1.d30
              B(i)=1.d30*cBnd
            else
              do 13 j=1,2*MBand-1
                A(j,i)=0.
13            continue
              A(MBand,i)=1.
              B(i)=cBnd
            end if
          end if

*       Neumann boundary condition
          if(cKod.eq.2) then
            Qc(i)=Q(i)*Conc(i)
          end if

*       Cauchy boundary condition
          if(cKod.eq.3) then
            B(i)=B(i)-Q(i)*(cBnd-alf*Conc(i))
            if(lOrt) jjj=IADD(i)
            A(jjj,i)=A(jjj,i)-epsi*Q(i)
            Qc(i)=Q(i)*cBnd
          end if

        end if
14    continue
      return
```

```
      end
*************************************************************************

*     Initial values for solute transport calculation

      subroutine ChInit(NumNP,NumEl,NumElD,NMat,x,y,KX,MatNum,NLevel,
     !                  Con,hNew,Sink,cSink,Vx,Vz,ConAxx,ConAzz,ConAxz,
     !                  Dispxx,Dispzz,Dispxz,ChPar,theta,thSat,Conc,Fc,
     !                  Gc,ListNE,lUpW,WeTab,dt,dtMaxC,Peclet,Courant,
     !                  KAT,lArtD,PeCr,ConO)

      logical lUpW,lArtD
      dimension hNew(NumNP),x(NumNP),y(NumNP),KX(NumElD,4),theta(NumNP),
     !          Sink(NumNP),ChPar(10,NMat),Vx(NumNP),Vz(NumNP),
     !          MatNum(NumNP),Con(NumNP),ConAxx(NumEl),ConAzz(NumEl),
     !          ConAxz(NumEl),Dispxx(NumNP),Dispzz(NumNP),Dispxz(NumNP),
     !          ListNE(NumNP),Fc(NumNP),Gc(NumNP),WeTab(3,2*NumEl),
     !          thSat(NMat),Conc(NumNP),ConO(NumNP)

      do 11 i=1,NumNP
        M=MatNum(i)
        if(NLevel.eq.2) then
          cS=cSink
          if(cS.gt.Conc(i)) cS=Conc(i)
          Gc(i)=ChPar(8,M)*theta(i)+ChPar(1,M)*ChPar(9,M)-Sink(i)*cS
          Fc(i)=ChPar(6,M)*theta(i)+ChPar(1,M)*ChPar(7,M)*ChPar(5,M)+
     !          Sink(i)
        end if
        ConO(i)=Con(i)
11    continue
      call Veloc(KAT,NumNP,NumEl,NumElD,hNew,x,y,KX,ListNE,Con,ConAxx,
     !           ConAzz,ConAxz,Vx,Vz)
      call Disper(NumNP,NMat,Dispxx,Dispzz,Dispxz,Vx,Vz,theta,thSat,
     !            ChPar,MatNum,lArtD,PeCr,dt)
      call PeCour(NumNP,NumEl,NumElD,NMat,x,y,Vx,Vz,KX,MatNum,Dispxx,
     !            Dispzz,ChPar,theta,dt,dtMaxC,Peclet,Courant,lUpW,
     !            lArtD,PeCr)
      if(lUpW)
     !  call WeFact(NumNP,NumEl,NumElD,x,y,KX,WeTab,Vx,Vz,Dispxx,
     !              Dispzz,Dispxz)
      return
      end

*************************************************************************

*     Calculate  velocities

      subroutine Veloc(KAT,NumNP,NumEl,NumElD,hNew,x,y,KX,ListNE,Con,
     !                 ConAxx,ConAzz,ConAxz,Vx,Vz)

      integer e
      dimension hNew(NumNP),x(NumNP),y(NumNP),ListNE(NumNP),Con(NumNP),
     !          KX(NumElD,4),Vx(NumNP),Vz(NumNP),ConAxx(NumEl),
     !          ConAzz(NumEl),ConAxz(NumEl),List(3)

      do 11 i=1,NumNP
        Vx(i)=0.
        Vz(i)=0.
11    continue
      do 14 e=1,NumEl
        CAxx=ConAxx(e)
        CAzz=ConAzz(e)
        CAxz=ConAxz(e)
        NCorn=4
        if(KX(e,3).eq.KX(e,4)) NCorn=3
        do 13 n=1,NCorn-2
          i=KX(e,1)
          j=KX(e,n+1)
          k=KX(e,n+2)
          List(1)=i
          List(2)=j
          List(3)=k
```

```
          vi=y(j)-y(k)
          vj=y(k)-y(i)
          vk=y(i)-y(j)
          wi=x(k)-x(j)
          wj=x(i)-x(k)
          wk=x(j)-x(i)
          Area=.5*(wk*vj-wj*vk)
          A=1./Area/2.
          Ai=CAxx*vi+CAxz*wi
          Aj=CAxx*vj+CAxz*wj
          Ak=CAxx*vk+CAxz*wk
          Vxx=A*(Ai*hNew(i)+Aj*hNew(j)+Ak*hNew(k))
          if(KAT.gt.0) Vxx=Vxx+CAxz
          Ai=CAxz*vi+CAzz*wi
          Aj=CAxz*vj+CAzz*wj
          Ak=CAxz*vk+CAzz*wk
          Vzz=A*(Ai*hNew(i)+Aj*hNew(j)+Ak*hNew(k))
          if(KAT.gt.0) Vzz=Vzz+CAzz
          do 12 m=1,3
            l=List(m)
            Vx(l)=Vx(l)-Con(l)*Vxx
            Vz(l)=Vz(l)-Con(l)*Vzz
12        continue
13      continue
14    continue
      do 15 i=1,NumNP
        Vx(i)=Vx(i)/ListNE(i)
        Vz(i)=Vz(i)/ListNE(i)
15    continue
      return
      end

*************************************************************************

*     Calculate the dispersion coefficient

      subroutine Disper(NumNP,NMat,Dispxx,Dispzz,Dispxz,Vx,Vz,theta,
     !                  thSat,ChPar,MatNum,lArtD,PeCr,dt)

      logical lArtD
      dimension Vx(NumNP),Vz(NumNP),theta(NumNP),ChPar(10,NMat),
     !          Dispxx(NumNP),Dispzz(NumNP),Dispxz(NumNP),MatNum(NumNP),
     !          thSat(NMat)

      do 11 i=1,NumNP
        M=MatNum(i)
        Tau=theta(i)**(7./3.)/thSat(M)**2
        Vabs=sqrt(Vx(i)*Vx(i)+Vz(i)*Vz(i))
        Dif=theta(i)*ChPar(2,M)*Tau
        DispL=ChPar(3,M)
        DispT=ChPar(4,M)
        if(lArtD) DispL=amax1(DispL,Vabs*dt
     !                  /(theta(i)+ChPar(1,M)*ChPar(5,M))/PeCr-Dif/Vabs)
        Dispxx(i)=Dif
        Dispzz(i)=Dif
        Dispxz(i)=0.
        if(Vabs.gt.0.) then
          Dispxx(i)=DispL*Vx(i)*Vx(i)/Vabs+DispT*Vz(i)*Vz(i)/Vabs+Dif
          Dispzz(i)=DispL*Vz(i)*Vz(i)/Vabs+DispT*Vx(i)*Vx(i)/Vabs+Dif
          Dispxz(i)=(DispL-DispT)*Vx(i)*Vz(i)/Vabs
        end if
11    continue
      return
      end

*************************************************************************

*     Calculate upstream weighing factors

      subroutine WeFact(NumNP,NumEl,NumElD,x,y,KX,WeTab,Vx,Vz,Dispxx,
     !                  Dispzz,Dispxz)
```

```
      integer e
      dimension x(NumNP),y(NumNP),KX(NumElD,4),Vx(NumNP),Vz(NumNP),
     !          Dispxx(NumNP),Dispzz(NumNP),Dispxz(NumNP),
     !          WeTab(3,2*NumEl),Beta(3),List(3)

      TanH(z)=(exp(z)-exp(-z))/(exp(z)+exp(-z))

      NumSEl=0
      do 13 e=1,NumEl
        NCorn=4
        if(KX(e,3).eq.KX(e,4)) NCorn=3
        do 12 n=1,NCorn-2
          NumSEl=NumSEl+1
          M1=KX(e,1)
          M2=KX(e,n+1)
          M3=KX(e,n+2)
          A=y(M2)-y(M1)
          B=x(M2)-x(M1)
          Beta(1)=atan2(A,B)
          A=y(M3)-y(M2)
          B=x(M3)-x(M2)
          Beta(2)=atan2(A,B)
          A=y(M1)-y(M3)
          B=x(M1)-x(M3)
          Beta(3)=atan2(A,B)
          List(1)=M1
          List(2)=M2
          List(3)=M3
          do 11 j=1,3
            k=j-1
            if(k.eq.0) k=3
            WeTab(k,NumSEl)=0.
            M1=List(j)
            jp1=j+1
            if(j.eq.3) jp1=1
            M2=List(jp1)
            Vxx=(Vx(M1)+Vx(M2))/2.
            Vzz=(Vz(M1)+Vz(M2))/2.
            if(abs(Vxx).lt.1.e-30.and.abs(Vzz).lt.1.e-30) goto 11
            BetaV=atan2(Vzz,Vxx)
            Delta=abs(BetaV-Beta(j))
            if(Delta.gt.0.314.and.abs(Delta-3.1416).gt.0.314) goto 11
            ALeng=sqrt((x(M2)-x(M1))**2+(y(M2)-y(M1))**2)
            CBeta=cos(Beta(j))
            SBeta=sin(Beta(j))
            Val=Vxx*CBeta+Vzz*SBeta
            VV=sqrt(Vxx*Vxx+Vzz*Vzz)
            DLL=(Dispxx(M1)+Dispxx(M2))/2.
            DLT=(Dispxz(M1)+Dispxz(M2))/2.
            DTT=(Dispzz(M1)+Dispzz(M2))/2.
            DAL=abs(DLL*CBeta*CBeta+2.0*CBeta*SBeta*DLT+DTT*SBeta*SBeta)
            Vel=VAL*ALeng
            Disp=2.0*DAL
            aa=11.
            if(Disp.gt.0.) aa=abs(Vel/Disp)
            if(Disp.lt.1.e-30.or.abs(Vel).lt.0.001*VV.or.
     !                           abs(aa).gt.10.) then
              if(abs(Vel).lt.0.001*VV) WeTab(k,NumSEl)=0.0
              if(Vel.gt.0.001*VV) WeTab(k,NumSEl)=1.0
              if(Vel.lt.-0.001*VV) WeTab(k,NumSEl)=-1.0
            else
              WeTab(k,NumSEl)=1.0/TanH(Vel/Disp)-Disp/Vel
            end if
11        continue
12      continue
13    continue
      return
      end

*****************************************************************************

*     Calculate the maximum local Peclet and Courant numbers
```

```
      subroutine PeCour(NumNP,NumEl,NumElD,NMat,x,y,Vx,Vz,KX,MatNum,
     !                  Dispxx,Dispzz,ChPar,theta,dt,dtMaxC,Peclet,
     !                  Courant,lUpW,lArtD,PeCr)

      logical lUpW,lArtD
      dimension KX(NumElD,4),x(NumNP),y(NumNP),Vx(NumNP),Vz(NumNP),
     !          MatNum(NumNP),Dispxx(NumNP),Dispzz(NumNP),theta(NumNP),
     !          ChPar(10,NMat),Bi(3),Ci(3)

      Peclet=0.
      Courant=0.
      dtMaxC=1.e+30
      do 12 n=1,NumEl
        NUS=4
        if(KX(n,3).eq.KX(n,4)) NUS=3
        do 11 k=1,NUS-2
          PecX=99999.
          PecY=99999.
          dt1=1.e+30
          dt2=1.e+30
          i=KX(n,1)
          j=KX(n,k+1)
          l=KX(n,k+2)
          Ci(1)=x(l)-x(j)
          Ci(2)=x(i)-x(l)
          Ci(3)=x(j)-x(i)
          Bi(1)=y(j)-y(l)
          Bi(2)=y(l)-y(i)
          Bi(3)=y(i)-y(j)
          delX=amax1(abs(Ci(1)),abs(Ci(2)),abs(Ci(3)))
          delY=amax1(abs(Bi(1)),abs(Bi(2)),abs(Bi(3)))
          DxE=(Dispxx(i)+Dispxx(j)+Dispxx(l))/3.
          DzE=(Dispzz(i)+Dispzz(j)+Dispzz(l))/3.
          VxE=abs(Vx(i)+Vx(j)+Vx(l))/3.
          VzE=abs(Vz(i)+Vz(j)+Vz(l))/3.
          if(DxE.gt.0.) PecX=VxE*delX/DxE
          if(DzE.gt.0.) PecY=VzE*delY/DzE
          Peclet=amax1(Peclet,PecX,PecY)
          Peclet=amin1(Peclet,99999.)

          VxMax=amax1(abs(Vx(i))/theta(i),abs(Vx(j))/theta(j),abs(Vx(l))
     !                /theta(l))
          VzMax=amax1(abs(Vz(i))/theta(i),abs(Vz(j))/theta(j),abs(Vz(l))
     !                /theta(l))
          R1=1.+ChPar(1,MatNum(i))*ChPar(5,MatNum(i))/theta(i)
          R2=1.+ChPar(1,MatNum(j))*ChPar(5,MatNum(j))/theta(j)
          R3=1.+ChPar(1,MatNum(l))*ChPar(5,MatNum(l))/theta(l)
          RMin=amin1(R1,R2,R3)
          CourX=VxMax*dt/delX/RMin
          CourY=VzMax*dt/delY/RMin
          Courant=amax1(Courant,CourX,CourY)

          CourMax=1.0
          if(.not.lUpW.and..not.lArtD)
     !      CourMax=amin1(1.,PeCr/amax1(0.5,PecX),PeCr/amax1(0.5,PecY))
          if(VxMax.gt.0.) dt1=CourMax*delX*RMin/VxMax
          if(VzMax.gt.0.) dt2=CourMax*delY*RMin/VzMax
          dtMaxC=amin1(dtMaxC,dt1,dt2)

11      continue
12    continue
      return
      end

*****************************************************************************

*     Solve the global matrix equation for transport

      subroutine SolveT(A,B,MBand,MBandD,NumNP)

      double precision A,B,P,C,Sum
      dimension A(MBandD,NumNP),B(NumNP)
```

```
      N1=NumNP-1
      do 12 k=1,N1
        P=1./A(MBand,k)
        kk=k+1
        kc=MBand
        do 11 i=kk,NumNP
          kc=kc-1
          if(kc.le.0) goto 12
          C=-P*A(kc,i)
          A(kc,i)=C
          ii=kc+1
          L=kc+MBand-1
          do 11 j=ii,L
            jj=j+MBand-kc
            A(j,i)=A(j,i)+C*A(jj,k)
11      continue
12    continue
      do 14 i=2,NumNP
        jj=MBand+1-i
        ii=1
        if(jj.le.0) then
          jj=1
          ii=i-MBand+1
        end if
        Sum=0.
        do 13 j=jj,MBand-1
          Sum=Sum+A(j,i)*B(ii)
          ii=ii+1
13      continue
        B(i)=B(i)+Sum
14    continue
      B(NumNP)=B(NumNP)/A(MBand,NumNP)
      do 16 k=1,N1
        i=NumNP-k
        jj=i
        m=min0(2*MBand-1,MBand+k)
        Sum=0.
        do 15 j=MBand+1,m
          jj=jj+1
          Sum=Sum+A(j,i)*B(jj)
15      continue
        B(i)=(B(i)-Sum)/A(MBand,i)
16    continue
      return
      end

* ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

```
* Source file MATERIAL.FOR ||||||||||||||||||||||||||||||||||||||||||||||||||||||

      real function FK(h,Par)

      implicit double precision(A-H,O-Z)
      double precision n,m,Ks,Kr,Kk
      real h,Par(10)
      integer PPar

      BPar=.5
      PPar=2.
      Qr=Par(1)
      Qs=Par(2)
      Qa=Par(3)
      Qm=Par(4)
      Alfa=Par(5)
      n=Par(6)
      Ks=Par(7)
      Kk=Par(8)
      Qk=Par(9)
      m=1.-1./n
      HMin=-1.d300**(1./n)/max(Alfa,1.d0)
      HH=max(dble(h),HMin)
      Qees=dmin1((Qs-Qa)/(Qm-Qa),.999999999999999d0)
      Qeek=dmin1((Qk-Qa)/(Qm-Qa),Qees)
      Hs=-1./Alfa*(Qees**(-1./m)-1.)**(1./n)
      Hk=-1./Alfa*(Qeek**(-1./m)-1.)**(1./n)
      if(h.lt.Hk) then
        Qee=(1.+(-Alfa*HH)**n)**(-m)
        Qe =(Qm-Qa)/(Qs-Qa)*Qee
        Qek=(Qm-Qa)/(Qs-Qa)*Qeek
        FFQ =1.-(1.-Qee**(1./m))**m
        FFQk=1.-(1.-Qeek**(1./m))**m
        if(FFQ.le.0.d0) FFQ=m*Qee**(1./m)
        Kr=(Qe/Qek)**Bpar*(FFQ/FFQk)**PPar*Kk/Ks
        FK=max(Ks*Kr,1.d-37)
        return
      end if
      if(h.ge.Hk.and.h.lt.Hs) then
        Kr=(1.-Kk/Ks)/(Hs-Hk)*(h-Hs)+1.
        FK=Ks*Kr
      end if
      if(h.ge.Hs) FK=Ks
      return
      end

**************************************************************************

      real function FC(h,Par)

      implicit double precision(A-H,O-Z)
      double precision n,m
      real h,Par(9)

      Qr=Par(1)
      Qs=Par(2)
      Qa=Par(3)
      Qm=Par(4)
      Alfa=Par(5)
      n=Par(6)
      m=1.-1./n
      HMin=-1.d300**(1./n)/max(Alfa,1.d0)
      HH=max(dble(h),HMin)
      Qees=dmin1((Qs-Qa)/(Qm-Qa),.999999999999999d0)
      Hs=-1./Alfa*(Qees**(-1./m)-1.)**(1./n)
      if(h.lt.Hs) then
        C1=(1.+(-Alfa*HH)**n)**(-m-1.)
        C2=(Qm-Qa)*m*n*(Alfa**n)*(-HH)**(n-1.)*C1
        FC=max(C2,1.d-37)
        return
      else
        FC=0.0
      end if
```

```
      return
      end
*****************************************************************************

      real function FQ(h,Par)

      implicit double precision(A-H,O-Z)
      double precision n,m
      real h,Par(9)

      Qr=Par(1)
      Qs=Par(2)
      Qa=Par(3)
      Qm=Par(4)
      Alfa=Par(5)
      n=Par(6)
      m=1.-1./n
      HMin=-1.d300**(1./n)/max(Alfa,1.d0)
      HH=max(dble(h),HMin)
      Qees=dmin1((Qs-Qa)/(Qm-Qa),.999999999999999d0)
      Hs=-1./Alfa*(Qees**(-1./m)-1.)**(1./n)
      if(h.lt.Hs) then
        Qee=(1.+(-Alfa*HH)**n)**(-m)
        FQ=max(Qa+(Qm-Qa)*Qee,1.d-37)
        return
      else
        FQ=Qs
      end if
      return
      end
*****************************************************************************

      real function FH(Qe,Par)

      implicit double precision(A-H,O-Z)
      double precision n,m
      real Qe,Par(9)

      Qr=Par(1)
      Qs=Par(2)
      Qa=Par(3)
      Qm=Par(4)
      Alfa=Par(5)
      n=Par(6)
      m=1.-1./n
      HMin=-1.d300**(1./n)/max(Alfa,1.d0)
      QeeM=(1.+(-Alfa*HMin)**n)**(-m)
      Qee=dmin1(dmax1(Qe*(Qs-Qa)/(Qm-Qa),QeeM),.999999999999999d0)
      FH=max(-1./Alfa*(Qee**(-1./m)-1.)**(1./n),-1.d37)
      return
      end

* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
* Source file SINK2.FOR !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

      subroutine SetSnk(NumNP,NMat,MatNum,hNew,TPot,Sink,P0,POptm,P2H,
     !           P2L,P3,r2H,r2L,Beta,Length)

      real Length
      dimension MatNum(NumNP),hNew(NumNP),POptm(NMat),Beta(NumNP),
     !     Sink(NumNP)

      do 11 i=1,NumNP
        if(Beta(i).gt.0.) then
          M=MatNum(i)
          Alfa=FAlfa(TPot,hNew(i),P0,POptm(M),P2H,P2L,P3,r2H,r2L)
          Sink(i)=Alfa*Beta(i)*Length*TPot
        end if
11    continue
      return
      end

*************************************************************************

      real function FAlfa(TPot,h,P0,P1,P2H,P2L,P3,r2H,r2L)

      if(TPot.lt.r2L) P2=P2L
      if(TPot.gt.r2H) P2=P2H
      if((TPot.ge.r2L).and.(TPot.le.r2H))
     !  P2=P2H+(r2H-TPot)/(r2H-r2L)*(P2L-P2H)
      FAlfa=0.0
      if((h.gt.P3).and.(h.lt.P2)) FAlfa=(h-P3)/(P2-P3)
      if((h.ge.P2).and.(h.le.P1)) FAlfa=1.0
      if((h.gt.P1).and.(h.lt.P0)) FAlfa=(h-P0)/(P1-P0)
      return
      end

* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
* Source file TIME2.FOR !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

      subroutine TmCont(dt,dtMaxW,dtOpt,dMul,dMul2,dtMin,Iter,tPrint,
     !                  tAtm,t,tMax,dtMaxC)

      dtMax=amin1(dtMaxW,dtMaxC)
      tFix=amin1(tPrint,tAtm,tMax)
      if(Iter.le.3.and.(tFix-t).ge.dMul*dtOpt)
     !  dtOpt=amin1(dtMax,dMul*dtOpt)
      if(Iter.ge.7)
     !  dtOpt=amax1(dtMin,dMul2*dtOpt)
      dt=amin1(dtOpt,tFix-t)
      dt=amin1((tFix-t)/anint((tFix-t)/dt),dtMax)
      if(tFix-t.ne.dt.and.dt.gt.(tFix-t)/2.) dt=(tFix-t)/2.
      return
      end

*************************************************************************

      subroutine SetAtm(tAtm,rTop,rRoot,hCritA,Width,KXB,NumBP,Kode,
     !                  hNew,Q,NumNP,GWLOL,qGWLF,FreeD,cPrec,cht,crt)

      logical qGWLF,FreeD
      dimension Width(NumBP),KXB(NumBP),Kode(NumNP),hNew(NumNP),Q(NumNP)

      read (31,*) tAtm,Prec,cPrec,rSoil,rRoot,hCritA,rGWL,GWL,crt,cht
      Prec=abs(Prec)
      rSoil=abs(rSoil)
      rRoot=abs(rRoot)
      hCritA=-abs(hCritA)
      hGWL=GWL+GWLOL
      rTop=rSoil-Prec
      do 11 i=1,NumBP
        n=KXB(i)
        K=Kode(n)
        if(K.eq.4.or.K.eq.-4) then
          Kode(n)=-4
          Q(n)=-Width(i)*rTop
          if((Prec-rSoil).gt.0.) then
            cPrec=Prec/(Prec-rSoil)*cPrec
          else
            cPrec=0.
          end if
          goto 11
        end if
        if(K.eq. 3) hNew(n)=hGWL
        if(K.eq.-3.and..not.qGWLF.and..not.FreeD) Q(n)=-Width(i)*rGWL
11    continue
      return
      end

*************************************************************************

      real function Fqh(GWL,Aqh,Bqh)
      Fqh=-Aqh*exp(Bqh*abs(GWL))
      return
      end

* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
* Source file OUTPUT2.FOR ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

      subroutine TLInf(NumNP,NumBP,Kode,Q,hNew,CumQ,Width,SWidth,KXB,t,
     !                 dt,TLevel,ShortF,TPrint,Iter,ItCum,rTop,rRoot,
     !                 vMeanR,hMeanT,hMeanR,hMeanG,AtmInF,SinkF,CumQrT,
     !                 CumQrR,CumQvR,NumKD,hMean,vMean,lWat,lChem,rLen,
     !                 Peclet,Courant,wCumT,wCumA)

      integer TLevel
      logical ShortF,SinkF,AtmInF,lWat,lChem
      dimension Q(NumNP),Kode(NumNP),KXB(NumBP),SWidth(NumKD),
     !          hNew(NumNP),Width(NumBP),CumQ(NumKD),hMean(NumKD),
     !          vMean(NumKD)

      if(TLevel.eq.1) then
        if(.not.lChem) then
          write(70,110)
        else
          if(lWat) then
            write(70,120)
          else
            write(70,130)
          end if
        end if
        write(71,140)
        write(77,150)
        if(lWat) write(78,160)
      end if
      if(lWat.or.TLevel.eq.1) then
        do 11 i=1,NumKD
          vMean(i)=0.
          hMean(i)=0.
11      continue
        do 12 i=1,NumBP
          n=KXB(i)
          j=iabs(Kode(n))
          if(j.eq.0) goto 12
          hMean(j)=hMean(j)+hNew(n)*Width(i)/SWidth(j)
          if(j.eq.4) vMean(j)=vMean(j)-Q(n)/SWidth(j)
12      continue
        hMeanT=hMean(4)
        hMeanG=hMean(3)
        do 13 i=1,NumNP
          j=iabs(Kode(i))
          wCumA=wCumA+abs(Q(i))*dt
          if(j.ne.0.and.j.ne.4) then
            vMean(j)=vMean(j)-Q(i)
          end if
13      continue
        if(.not.lWat.and.TLevel.eq.1) then
          write(71,170) t,rTop,rRoot,vMean(4),vMeanR,vMean(3),vMean(1),
     !                  vMean(2),(vMean(i),i=5,NumKD)
          write(77,180) t,hMean(4),hMeanR,hMean(3),hMean(1),hMean(2),
     !                  (hMean(i),i=5,NumKD)
        end if
      end if
      if(lWat) then
        wCumA=wCumA+abs(vMeanR*dt*rLen)
        wCumT=CumQvR
        do 14 j=1,NumKD
          if(j.eq.4) then
            CumQ(j)=CumQ(j)+vMean(j)*dt*SWidth(4)
          else
            CumQ(j)=CumQ(j)+vMean(j)*dt
          end if
          wCumT=wCumT+CumQ(j)
14      continue
        CumQrT=CumQrT+rTop  *dt*SWidth(4)
        CumQrR=CumQrR+rRoot *dt*rLen
        CumQvR=CumQvR+vMeanR*dt*rLen
      end if
      if(.not.ShortF.or.abs(TPrint-t).lt.0.001*dt) then
        if(lWat) then
```

```
        write(71,170) t,rTop,rRoot,vMean(4),vMeanR,vMean(3),vMean(1),
     !                vMean(2),(vMean(i),i=5,NumKD)
        write(77,180) t,hMean(4),hMeanR,hMean(3),hMean(1),hMean(2),
     !                (hMean(i),i=5,NumKD)
        write(78,190) t,CumQrT,CumQrR,CumQ(4),CumQvR,CumQ(3),
     !                CumQ(1),CumQ(2),(CumQ(i),i=5,NumKD)
      end if
      if(lChem) then
        if(lWat) then
          write(70,200) TLevel,t,dt,Iter,ItCum,Peclet,Courant
        else
          write(70,210) TLevel,t,dt,Peclet,Courant
        end if
      else
        write(70,200) TLevel,t,dt,Iter,ItCum
      end if
    end if
    if(lWat) then
      if(SinkF) then
        write(*,220) t,Iter,ItCum,CumQ(4),CumQvR,CumQ(3),hMean(4),
     !               hMeanR,hMean(3)
      else
        if(AtmInF) then
          write(*,220) t,Iter,ItCum,CumQ(4),CumQ(1),CumQ(3),hMean(4),
     !                 hMean(1),hMean(3)
        else
          write(*,220) t,Iter,ItCum,vMean(1),CumQ(1),CumQ(2),hMean(1),
     !                 hMean(2)
        end if
      end if
    end if

110   format(//' TLevel   Time         dt      Iter  ItCum')
120   format(//' TLevel   Time         dt      Iter  ItCum  Peclet   ',
     !'Courant')
130   format(//' TLevel   Time         dt        Peclet   Courant')
140   format(' All fluxes (v) are positive out of the region'//
     !'     Time       rAtm       rRoot      vAtm       vRoot      ',
     !'vKode3     vKode1     vSeep      vKode5     vKode6 ...'/
     !'      [T]       [L/T]      [L/T]      [L/T]      [L/T]      ',
     !' [V/T]      [V/T]      [V/T]      [V/T]      [V/T]'/)
150   format(//
     !'     Time         hAtm       hRoot     hKode3     hKode1 ',
     !'     hSeep     hKode5     hKode6 ... '/
     !'      [T]          [L]        [L]        [L]        [L]  ',
     !'      [L]        [L]        [L]'/)
160   format(
     !' All cumulative fluxes (CumQ) are positive out of the region'//
     !'     Time     CumQAP     CumQRP     CumQA      CumQR     CumQ3',
     !'       CumQ1      CumQS      CumQ5      CumQ6 ....'/
     !'      [T]        [V]        [V]        [V]        [V]       [V] ',
     !'        [V]        [V]        [V]        [V]'/)
170   format(f12.4,9e11.3)
180   format(f12.4,9f11.1)
190   format(f12.4,9e11.3)
200   format(i5,2e12.3,i5,i6,2f10.3)
210   format(i5,2e12.3,2f10.3)
220   format(f14.4,i3,i6,1x,3e11.3,3f7.0)
      return
      end

*************************************************************************

      subroutine ALInf(t,CumQ,hMeanT,hMeanR,hMeanG,ALevel,CumQrT,CumQrR,
     !                 CumQvR,NumKD)

      integer ALevel
      dimension CumQ(NumKD)

      if(ALevel.eq.1) write(72,110)
      write(72,120) t,CumQrT,CumQrR,CumQ(4),CumQvR,CumQ(3),hMeanT,
     !              hMeanR,hMeanG,ALevel
```

```
110   format(
     !' All cumulative fluxes (CumQ) are positive out of the region'//
     !'      Time      CumQAP      CumQRP      CumQA       CumQR ',
     !'     CumQ3        hAtm        hRoot      hKode3     A-level'/
     !'      [T]         [V]         [V]        [V]         [V]  ',
     !'      [V]          [L]         [L]         [L] '/)
120   format(f12.4,5e11.3,3f11.1,i8)
      return
      end

*************************************************************************

      subroutine SubReg(NumEl,NumElD,NumNP,NMat,hNew,ThO,ThN,x,y,MatNum,
     !                  LayNum,KX,KAT,t,dt,NLay,PLevel,lWat,lChem,Conc,
     !                  ChPar,wCumA,wCumT,cCumA,cCumT,wVolI,cVolI,WatIn,
     !                  SolIn)

      logical lWat,lChem
      integer PLevel,e
      dimension hNew(NumNP),x(NumNP),y(NumNP),MatNum(NumNP),Conc(NumNP),
     !          KX(NumElD,4),ChPar(10,NMat),LayNum(NumEl),ThO(NumNP),
     !          ThN(NumNP),WatIn(NumEl),SolIn(NumEl),Area(10),hMean(10),
     !          SubVol(10),SubCha(10),cMean(10),ConSub(10)

      xMul=1.
      ATot=0.
      if(lWat.or.PLevel.le.1) then
        Volume=0.
        Change=0.
        hTot=0.
        DeltW=0.
      end if
      if(lChem) then
        cTot=0.
        ConVol=0.
        DeltC=0.
      end if
      do 11 i=1,NLay
        Area(i)=0.
        if(lWat.or.PLevel.le.1) then
          SubVol(i)=0.
          SubCha(i)=0.
          hMean(i)=0.
        end if
        if(lChem) then
          ConSub(i)=0.
          cMean(i)=0.
        end if
11    continue
      do 13 e=1,NumEl
        Lay=LayNum(e)
        wEl=0.
        cEl=0.
        NUS=4
        if(KX(e,3).eq.KX(e,4)) NUS=3
        do 12 k=1,NUS-2
          i=KX(e,1)
          j=KX(e,k+1)
          l=KX(e,k+2)
          Mi=MatNum(i)
          Mj=MatNum(j)
          Mk=MatNum(l)
          Cj=x(i)-x(l)
          Ck=x(j)-x(i)
          Bj=y(l)-y(i)
          Bk=y(i)-y(j)
          if(KAT.eq.1) xMul=2.*3.1416*(x(i)+x(j)+x(l))/3.
          AE=xMul*(Ck*Bj-Cj*Bk)/2.
          Area(Lay)=Area(Lay)+AE
          if(lWat.or.PLevel.le.1) then
            hE=(hNew(i)+hNew(j)+hNew(l))/3.
            VNewE=AE*(thN(i)+thN(j)+thN(l))/3.
            VOldE=AE*(thO(i)+thO(j)+thO(l))/3.
```

```
          Volume=Volume+VNewE
          wEl=wEl+VNewE
          Change=Change+(VNewE-VOldE)/dt
          SubVol(Lay)=SubVol(Lay)+VNewE
          SubCha(Lay)=SubCha(Lay)+(VNewE-VOldE)/dt
          hTot=hTot+hE*AE
          hMean(Lay)=hMean(Lay)+hE*AE
        end if
        if(lChem) then
          cE=(Conc(i)+Conc(j)+Conc(l))/3.
          cNewE=AE*((thN(i)+ChPar(1,Mi)*ChPar(5,Mi))*Conc(i)+
     !              (thN(j)+ChPar(1,Mj)*ChPar(5,Mj))*Conc(j)+
     !              (thN(l)+ChPar(1,Mk)*ChPar(5,Mk))*Conc(l))/3.
          ConVol=ConVol+cNewE
          cEl=cEl+cNewE
          ConSub(Lay)=ConSub(Lay)+cNewE
          cTot=cTot+cE*AE
          cMean(Lay)=cMean(Lay)+cE*AE
        end if
        if(k.eq.NUS-2) then
          if(PLevel.eq.0) then
            if(lWat) WatIn(e)=wEl
            if(lChem) SolIn(e)=cEl
          else
            if(lWat) DeltW=DeltW+abs(WatIn(e)-wEl)
            if(lChem) DeltC=DeltC+abs(SolIn(e)-cEl)
          end if
        end if
12    continue
13    continue
      do 14 Lay=1,NLay
        if(lWat.or.PLevel.le.1) hMean(Lay)=hMean(Lay)/Area(Lay)
        if(lChem) cMean(Lay)=cMean(Lay)/Area(Lay)
        ATot=ATot+Area(Lay)
14    continue
      if(lWat.or.PLevel.le.1) hTot=hTot/ATot
      if(lChem) cTot=cTot/ATot
      if(PLevel.eq.0) write(80,110)
      write(80,120) t,     (        i,i=1,NLay)
      write(80,130) ATot,  (  Area(i),i=1,NLay)
      if(lWat.or.PLevel.le.1) then
        write(80,140) Volume,(SubVol(i),i=1,NLay)
        write(80,150) Change,(SubCha(i),i=1,NLay)
        write(80,160) hTot,  ( hMean(i),i=1,NLay)
      end if
      if(lChem) then
        write(80,170) ConVol,(ConSub(i),i=1,NLay)
        write(80,180) cTot,  ( cMean(i),i=1,NLay)
      end if

*     Mass balance calculation
      if(PLevel.eq.0) then
        wVolI=Volume
        if(lChem) cVolI=ConVol
      else
        if(lWat) then
          wBalT=Volume-wVolI+wCumT
          write(80,190) wBalT
          ww=amax1(DeltW,wCumA)
          if(ww.ge.1.e-25) then
            wBalR=abs(wBalT)/ww*100.
            write(80,200) wBalR
          end if
        end if
        if(lChem) then
          cBalT=ConVol-cVolI+cCumT
          write(80,210) cBalT
          cc=amax1(DeltC,cCumA)
          if(cc.ge.1.e-25) then
            cBalR=abs(cBalT)/cc*100.
            write(80,220) cBalR
          end if
        end if
```

```
      end if

110   format(/' Time [T]           Total     Sub-region number ...')
120   format(/f12.4,16x,10(i7,4x))
130   format(' Area    [V]       ',e11.3,10e11.3)
140   format(' Volume  [V]       ',e11.3,10e11.3)
150   format(' InFlow  [V/T]     ',e11.3,10e11.3)
160   format(' hMean   [L]       ',e11.3,10f11.1)
170   format(' ConcVol [VM/L3]   ',e11.3,10e11.3)
180   format(' cMean   [M/L3]    ',e11.3,10e11.3)
190   format(' WatBalT [V]       ',e11.3)
200   format(' WatBalR [%]       ',f11.3)
210   format(' CncBalT [VM/L3]   ',e11.3)
220   format(' CncBalR [%]       ',f11.3)
      return
      end

*************************************************************************

      subroutine BouOut(NumNP,NumBP,t,hNew,theta,Q,Width,KXB,Kode,x,y,
     !                  Conc)

      dimension hNew(NumNP),Q(NumNP),Width(NumBP),theta(NumNP),
     !          KXB(NumBP),Kode(NumNP),x(NumNP),y(NumNP),Conc(NumNP)

      write(79,110) t
      ii=0
      do 12 i=1,NumNP
        if(Kode(i).ne.0) then
          do 11 j=1,NumBP
            n=KXB(j)
            if(n.eq.i) then
              ii=ii+1
              v=-Q(i)/Width(j)
              write(79,120) ii,i,x(i),y(i),Kode(i),Q(i),v,hNew(i),
     !                      theta(i),Conc(i)
              goto 12
            end if
11        continue
          ii=ii+1
          write(79,130) ii,i,x(i),y(i),Kode(i),Q(i),hNew(i),theta(i),
     !                  Conc(i)
        end if
12    continue

110   format(//' Time:',f12.4//
     !'    i    n    x      z    Code     Q          v       ',
     !'    h       th      Conc'/
     !'                                    [V/T]      [L/T]     ',
     !'   [L]     [-]     [M/L3]'/)
120   format(2i5,2f7.1,i5,2e11.3,f11.1,f7.3,e10.3)
130   format(2i5,2f7.1,i5,e11.3,11x,f11.1,f7.3,e10.3)
      return
      end

*************************************************************************

      subroutine SolInf(K,NumNP,Kode,Qc,t,dt,TLevel,ShortF,TPrint,
     !                  AtmInF,NumKD,SMean,ChemS,CumCh0,CumCh1,CumChR,
     !                  cCumA,cCumT,lWat)

      integer TLevel
      logical ShortF,AtmInF,lWat
      dimension Qc(NumNP),Kode(NumNP),ChemS(NumKD),SMean(NumKD)

      if(K.eq.1) then
        do 11 i=1,NumKD
          SMean(i)=0.
11      continue
        do 12 i=1,NumNP
          j=iabs(Kode(i))
          if(j.ne.0) then
            SMean(j)=Smean(j)-Qc(i)
```

```
          end if
12      continue
        cCumA=abs(CumCh0)+abs(CumCh1)+abs(CumChR)
        cCumT=CumCh0+CumCh1+CumChR
        do 13 j=1,NumKD
          ChemS(j)=ChemS(j)+SMean(j)*dt
          cCumT=cCumT+ChemS(j)
          cCumA=cCumA+abs(ChemS(j))
13      continue
        if(TLevel.eq.1) write(74,110)
        if(.not.AtmInf.and.(.not.ShortF.or.abs(TPrint-t).lt.0.001*dt))
     !    write(74,120) t,CumCh0,CumCh1,CumChR,(ChemS(j),j=1,NumKD),
     !                  (SMean(i),i=1,NumKD)
        else
          write(74,120) t,CumCh0,CumCh1,CumChR,(ChemS(j),j=1,NumKD),
     !                (SMean(i),i=1,NumKD)
        end if
        if(.not.lWat) write(*,130) t,TLevel,CumCh0,CumCh1,ChemS(1)

110     format(' All solute fluxes (SMean) and cumulative solute fluxes',
     !' (ChemS) are positive out of the region'//
     !'     Time     CumCh0     CumCh1     CumChR   ',20('-'),'  ChemS(i
     !),i=1,NumKD  ',22('-'),'  ',21('-'),'  SMean(j),j=1,NumKD ',22('-'
     !)/'      [T]     [VM/L3]    [VM/L3]    [VM/L3]',31(' '),'[VM/L3]',5
     !9(' '),'[VM/T/L3]'/)
120     format(f10.2,15e11.3)
130     format(f14.4,i6,1x,3e11.3,2x,2e11.3)
        return
        end

*************************************************************************

        subroutine ObsNod(t,NumNP,NObs,Node,hNew,ThNew,Conc)

        dimension Node(NObs),hNew(NumNP),ThNew(NumNP),Conc(NumNP)

        write(92,110) t,(hNew(Node(i)),ThNew(Node(i)),Conc(Node(i)),
     !              i=1,NObs)

110     format(f11.3,5(f11.3,f9.4,e11.3))
        return
        end

*************************************************************************

        subroutine hOut(hNew,x,y,NumNP,t,IJ)

        dimension hNew(NumNP),x(NumNP),y(NumNP)

        write(75,110) t
        L1=(IJ-1)/10+1
        do 12 n=1,NumNP,IJ
          do 11 L=1,L1
            m=n+(L-1)*10
            k=m+9
            if(L.eq.L1) k=n+IJ-1
            write(75,120) m,x(m),y(m),(hNew(j),j=m,k)
11        continue
12      continue

110     format(//' Time  ***',f12.4,' ***'//
     !         '    n    x(n)   z(n)       h(n)      h(n+1) ...'/)
120     format(i5,2f8.1,10f10.1)
        return
        end

*************************************************************************

        subroutine QOut(Q,x,y,NumNP,t,IJ)

        dimension Q(NumNP),x(NumNP),y(NumNP)

        write(73,110) t
```

```
      L1=(IJ-1)/10+1
      do 12 n=1,NumNP,IJ
        do 11 L=1,L1
          m=n+(L-1)*10
          k=m+9
          if(L.eq.L1) k=n+IJ-1
          write(73,120) m,x(m),y(m),(Q(j),j=m,k)
11      continue
12    continue

110   format(//' Time  ***',f12.4,' ***'//
     !       '    n    x(n)   z(n)       Q(n)      Q(n+1) ...'/)
120   format(i5,2f8.1,10e11.3)
      return
      end

*************************************************************************

      subroutine thOut(theta,x,y,NumNP,t,IJ)

      dimension theta(NumNP),x(NumNP),y(NumNP)

      write(76,110) t
      L1=(IJ-1)/16+1
      do 12 n=1,NumNP,IJ
        do 11 L=1,L1
          m=n+(L-1)*16
          k=m+15
          if(L.eq.L1) k=n+IJ-1
          write(76,120) m,x(m),y(m),(theta(j),j=m,k)
11      continue
12    continue

110   format(//' Time  ***',f12.4,' ***'//
     !       '    n    x(n)   z(n)      th(n)     th(n+1) ...'/)
120   format(i5,2f8.1,16f6.3)
      return
      end

*************************************************************************

      subroutine FlxOut(Vx,Vz,x,y,NumNP,t,IJ)

      dimension x(NumNP),y(NumNP),Vx(NumNP),Vz(NumNP)

      write(81,110) t
      write(82,120) t
      L1=(IJ-1)/10+1
      do 12 n=1,NumNP,IJ
        do 11 L=1,L1
          m=n+(L-1)*10
          k=m+9
          if(L.eq.L1) k=n+IJ-1
          write(81,130) m,x(m),y(m),(Vz(j),j=m,k)
          write(82,130) m,x(m),y(m),(Vx(j),j=m,k)
11      continue
12    continue

110   format(//' Time  ***',f12.4,' ***'//
     !       '    n    x(n)  z(n)     vz(n)     vz(n+1) ...'/)
120   format(//' Time  ***',f12.4,' ***'//
     !       '    n    x(n)  z(n)     vx(n)     vx(n+1) ...'/)
130   format(i5,2f8.1,10e10.2)
      return
      end

*************************************************************************

      subroutine cOut(NumNP,Conc,x,y,t,IJ)

      dimension Conc(NumNP),x(NumNP),y(NumNP)

      write(83,110) t
```

```
      L1=(IJ-1)/10+1
      do 12 n=1,NumNP,IJ
        do 11 L=1,L1
          m=n+(L-1)*10
          k=m+9
          if(L.eq.L1) k=n+IJ-1
          write(83,120) m,x(m),y(m),(Conc(j),j=m,k)
11      continue
12    continue

110   format(//' Time  ***',f12.4,' ***'//
     !       '    n    x(n)   z(n)      Conc(n)   Conc(n+1)  ...'/)
120   format(i5,2f8.1,10e11.3)
      return
      end

* ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

```
* Source file ORTHOFEM.FOR !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
*
*                              ORTHOFEM
*
*                             VERSION 1.02
*
*                     FORTRAN SUBROUTINES FOR
*                  ORTHOMIN OR CONJUGATE GRADIENT
*               MATRIX SOLUTION ON FINITE-ELEMENT GRIDS
*
*     ************************************************************
*
*                           CARL A. MENDOZA
*
*                     WITH CONTRIBUTIONS FROM:
*                          RENE THERRIEN
*
*                     BASED ON AN ORIGINAL CODE BY:
*                        FRANK W. LETNIOWSKI
*
*              WATERLOO CENTRE FOR GROUNDWATER RESEARCH
*                        UNIVERSITY OF WATERLOO
*                          WATERLOO, ONTARIO
*                           CANADA, N2L 3G1
*
*                     LATEST UPDATE: JANUARY 1991
*
*
*     ************************************************************
*
*                  COPYRIGHT (c) 1989, 1990, 1991
*                           E.A. SUDICKY
*                           C.A. MENDOZA
*              WATERLOO CENTRE FOR GROUNDWATER RESEARCH
*
*          DUPLICATION OF THIS PROGRAM, OR ANY PART THEREOF,
*          WITHOUT THE EXPRESS PERMISSION OF THE COPYRIGHT
*          HOLDERS IS STRICTLY FORBIDDEN
*
*     ************************************************************
*
*               Modified for SWMS_2D code by Jirka Simunek
*                           april 1993
*
*     ************************************************************
*
*                              DISCLAIMER
*
*     ALTHOUGH GREAT CARE HAS BEEN TAKEN IN PREPARING THIS CODE
*     AND THE ACCOMPANYING DOCUMENTATION, THE AUTHORS CANNOT BE
*     HELD RESPONSIBLE FOR ANY ERRORS OR OMISSIONS.  THE USER IS
*     EXPECTED TO BE FAMILIAR WITH THE FINITE-ELEMENT METHOD,
*     PRECONDITIONED ITERATIVE TECHNIQUES AND FORTRAN PROGRAMING.
*
*     ************************************************************
*
*            A USER'S GUIDE IS AVAILABLE -> CONSULT IT!
*
*     THESE SUBROUTINES SOLVE A BANDED (OR SPARSE) MATRIX USING:
*       - PRECONDITIONED ORTHOMIN FOR ASYMMETRIC MATRICES, OR
*       - PRECONDITIONED CONJUGATE GRADIENT FOR SYMMETRIC MATRICES
*         (FULL MATRIX STORAGE REQUIRED)
*
*     PRECONDITIONING IS BY INCOMPLETE LOWER-UPPER DECOMPOSITION
*       - ONLY ONE FACTORIZATION (GAUSSIAN ELIMINATION) IS PERFORMED
*       - EQUIVALENT TO DKR FACTORIZATION
*
*     THE SUBROUTINES ARE DESIGNED FOR FINITE-ELEMENT GRIDS
*       - ARBITRARY ELEMENT SHAPES AND NUMBERING MAY BE USED
*         - NUMBERING MAY, HOWEVER, AFFECT EFFICIENCY
*           - TRY TO MINIMIZE THE BANDWIDTH AS MUCH AS POSSIBLE
*         - ALL ELEMENTS MUST HAVE THE SAME NUMBER OF LOCAL NODES
*
```

```
*
*     THE FOLLOWING ROUTINES ARE CALLED FROM THE SOURCE PROGRAM:
*       IADMAKE (IN,NN,NE,NLN,MNLN,MAXNB,IAD,IADN,IADD)
*         -> ASSEMBLE ADJACENCY MATRIX
*       FIND (I,J,K,NN,MAXNB,IAD,IADN)
*         -> LOCATE MATRIX POSITION FOR A NODAL PAIR (ASSEMBLY)
*       ILU (R,NN,MAXNB,IAD,IADN,IADD,B)
*         -> DECOMPOSE GLOBAL MATRIX
*       ORTHOMIN (R,C,GT,NNR,MAXNB,MAXNN,IAD,IADN,IADD,B,VRV,
*                 RES,RQI,RQ,Q,QI,RQIDOT,ECNVRG,RCNVRG,ACNVRG,
*                 NORTH,MNORTH,MAXIT)
*         -> SOLVE DECOMPOSED MATRIX
*
*     THESE ROUTINES CALL OTHER ROUTINES (LOCATED DIRECTLY BELOW THE
*     APPROPRIATE PRIMARY ROUTINE IN THE CODE)
*
*     THE FOLLOWING ARRAYS MUST BE DEFINED IN THE SOURCE PROGRAM
*     (THESE ARRAYS ARE PASSED TO THE SOLVER SUBROUTINES):
*
*     IN(MNLN,MAXNE) - INCIDENCE MATRIX (ELEMENTAL NODE DEFINITION)
*
*     GT(MAXNN)      - RIGHT-HAND-SIDE VECTOR
*     C(MAXNN)       - SOLUTION VECTOR
*     R(MAXNB,MAXNN) - GLOBAL MATRIX TO BE SOLVED
*
*     ARRAY DIMENSIONING PARAMETERS
*
*     MAXNN  - MAXIMUM NUMBER OF NODES
*     MAXNE  - MAXIMUM NUMBER OF ELEMENTS
*     MNLN   - MAXIMUM NUMBER OF LOCAL NODES (IN AN ELEMENT)
*     MAXNB  - MAXIMUM NUMBER OF NODES ADJACENT TO A PARTICULAR NODE
*              (INCLUDING ITSELF).
*            - IE. THE MAXIMUM NUMBER OF INDEPENDENT NODES THAT A
*              PARTICULAR NODE SHARES AN ELEMENT WITH.
*            - THIS WILL BE IDENTICALLY EQUIVALENT TO THE MAXIMUM
*              NUMBER OF NONZERO ENTRIES IN A ROW OF THE FULL MATRIX.
*     MNORTH - MAXIMUM NUMBER OF ORTHOGONALIZATIONS PERFORMED
*              (AT LEAST MNORTH = 1 REQUIRED FOR CONJUGATE GRADIENT)
*
*
*     ORTHOMIN ARRAY SPACE/VARIABLES
*
*     NORTH  - NUMBER OF ORTHOGONALIZATIONS TO PERFORM
*            - SET NORTH=0 FOR SYMMETRIC MATRICES (CONJUGATE GRADIENT)
*     ECNVRG - RESIDUAL CONVERGENCE TOLERANCE
*     ACNVRG - ABSOLUTE CONVERGENCE TOLERANCE
*     RCNVRG - RELATIVE CONVERGENCE TOLERANCE
*     MAXIT  - MAXIMUM NUMBER OF ITERATIONS TO PERFORM
*     ITERP  - NUMBER OF ITERATIONS PERFORMED
*
*     B(MAXNB,MAXNN) - ILU DECOMPOSED MATRIX
*     Q(MAXNN)   - SEARCH DIRECTION Q
*     RQ(MAXNN)  - PRODUCT OF R AND Q
*     VRV(MAXNN) - EITHER V OR PRODUCT OF R AND V
*     RES(MAXNN) - RESIDUAL
*
*     QI(MAXNN,MNORTH)  - STORAGE OF Q'S
*     RQI(MAXNN,MNORTH) - STORAGE OF PRODUCTS OF R AND Q
*     RQIDOT(MNORTH)    - STORAGE OF DOT PRODUCTS OF RQ AND RQ
*
*     RESV   - PREVIOUS VALUE OF RES V DOT PRODUCT (CONJUGATE GRADIENT)
*
*     IAD(MAXNB,MAXNN) - ADJACENCY MATRIX (NODAL CONNECTIONS)
*
*     IADN(MAXNN) - NUMBER OF ADJACENT NODES IN IAD (SELF-INCLUSIVE)
*
*     IADD(MAXNN) - POSITION OF DIAGONAL IN ADJACENCY MATRIX
*
*
*     OTHER PARAMETERS PASSED FROM SOURCE PROGRAM
*
*     NN  - NUMBER OF NODES
*     NE  - NUMBER OF ELEMENTS
```

```
*     NLN - NUMBER OF LOCAL NODES IN AN ELEMENT
*
*
*     APPROXIMATE REAL STORAGE SPACE FOR ORTHOMIN AND MATRIX EQUATION
*
*       ((6 + 2*MAXNB + 2*MNORTH)*MAXNN)*(8 BYTES)
*
************************************************************************

      subroutine IADMake(KX,NumNP,NumEl,NumElD,MaxNB,IAD,IADN,IADD)

*     Generate the adjacency matrix for nodes from the element
*     indidence matrix

*     Requires subroutine Insert

      dimension KX(NumElD,4),IAD(MaxNB,NumNP),IADN(NumNP),IADD(NumNP)
      integer e

*     Determine independent adjacency within each element
*     version for SWMS_2D

      do 12 i=1,NumNP
        IADN(i)=0
        IADD(i)=0
        do 11 j=1,MaxNB
          IAD(j,i)=0
11      continue
12    continue

      do 14 e=1,NumEl
        NCorn=4
        if(KX(e,3).eq.KX(e,4)) NCorn=3
        do 13 n=1,NCorn-2
          i=KX(e,1)
          j=KX(e,n+1)
          k=KX(e,n+2)

          call Insert(i,j,kk,NumNP,MaxNB,IAD,IADN)
          call Insert(j,i,kk,NumNP,MaxNB,IAD,IADN)
          call Insert(i,k,kk,NumNP,MaxNB,IAD,IADN)
          call Insert(k,i,kk,NumNP,MaxNB,IAD,IADN)
          call Insert(j,k,kk,NumNP,MaxNB,IAD,IADN)
          call Insert(k,j,kk,NumNP,MaxNB,IAD,IADN)
13      continue
14    continue

*     Determine self-adjacency terms

      do 15 i=1,NumNP
        call Insert(i,i,kk,NumNP,MaxNB,IAD,IADN)
*     Store self-adjacency position
        IADD(i)=kk
15    continue
      return
      end

************************************************************************

      SUBROUTINE INSERT (I,J,K,NN,MAXNB,IAD,IADN)

*     ADD J TO THE ADJACENCY LIST FOR I

*     RETURNS THE POSITION K WHERE IT HAS BEEN ADDED, OR WHERE IT
*     WAS ALREADY IN THE LIST.

      DIMENSION IAD(MAXNB,NN),IADN(NN)

      LOGICAL FOUND
      FOUND = .FALSE.

*     DETERMINE NUMBER OF NODES ALREADY IN ADJACENCY LIST
```

```
      N = IADN(I)
      K = N + 1

*     DETERMINE WHETHER ALREADY IN LIST

      DO 10 L=1,N
        INODE = IAD(L,I)
        IF (INODE.GE.J) THEN
          K = L
          IF (INODE.EQ.J) FOUND = .TRUE.
          GO TO 15
        ENDIF
   10 CONTINUE

   15 CONTINUE

*     PLACE IN LIST (NUMERICAL ORDER)

      IF (FOUND) THEN
       CONTINUE
      ELSE
        IF ((N+1).GT.MAXNB) THEN
          WRITE (* ,601) I,MAXNB
          WRITE (50,601) I,MAXNB
  601     FORMAT (//5X,'ERROR IN IADMAKE: NODE ',I5,' HAS > '
     *           ,I5,' ADJACENCIES')
          STOP
        ENDIF

        IADN(I) = N + 1
        DO 20 L=(N+1),(K+1),(-1)
          IAD(L,I) = IAD(L-1,I)
   20   CONTINUE
       IAD(K,I) = J
      ENDIF

      RETURN
      END

*******************************************************************************

      SUBROUTINE FIND (I,J,K,NN,MAXNB,IAD,IADN)

*     FOR NODE I, DETERMINE THE 'BAND' (K) RELATED TO ITS ADJACENCY TO
*     NODE J.

*     IF NODE NOT ADJACENT, RETURN 0 AS THE 'BAND'

      DIMENSION IAD(MAXNB,NN),IADN(NN)

      K = 0
      N = IADN(I)

      DO 10 L=1,N
        INODE = IAD(L,I)

*     EXIT THE LOOP IF AT OR PAST THE REQUIRED POSITION

        IF (INODE.GE.J) THEN
          IF (INODE.EQ.J) K = L
          GO TO 20
        ENDIF
   10 CONTINUE

   20 CONTINUE

      RETURN
      END

*******************************************************************************

      SUBROUTINE ILU (R,NN,MAXNB,IAD,IADN,IADD,B)
```

```
*     INCOMPLETE LOWER-UPPER DECOMPOSITION OF MATRIX R INTO B
*     ONE STEP OF GAUSSIAN ELIMINATION PERFORMED
*     DIAGONAL DOMINANCE IS ASSUMED - NO PIVOTING PERFORMED
*     REQUIRES FUNCTION DU

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION R(MAXNB,NN),IAD(MAXNB,NN),IADN(NN),IADD(NN),B(MAXNB,NN)

*     INITIALIZE B

      DO 10 I=1,NN
        DO 10 J=1,MAXNB
          B(J,I) = 0.0D0
   10 CONTINUE

*     LOOP OVER NODES

      DO 20 I=1,NN

*     DETERMINE NUMBER OF BANDS/POSITION OF DIAGONAL IN THIS ROW

        N = IADN(I)
        K = IADD(I)

*     LOWER TRIANGULAR MATRIX

        DO 30 J=1,(K-1)
          SUM = R(J,I)
          ICUR = IAD(J,I)
          DO 40 L=1,(J-1)
            INODE = IAD(L,I)
            SUM = SUM - B(L,I)*DU(INODE,ICUR,NN,MAXNB,IAD,IADN,IADD,B)
   40     CONTINUE
          B(J,I) = SUM
   30   CONTINUE

*     DIAGONAL

        SUM = R(K,I)
        DO 50 L=1,(K-1)
          INODE = IAD(L,I)
          SUM = SUM - B(L,I)*DU(INODE,I,NN,MAXNB,IAD,IADN,IADD,B)
   50   CONTINUE
        D = 1.0D0/SUM
        B(K,I) = D

*     UPPER TRIANGULAR MATRIX
*       - ACTUALLY D*U TO OBTAIN UNIT DIAGONAL

        DO 60 J=(K+1),N
          SUM = R(J,I)
          ICUR = IAD(J,I)
          DO 70 L=1,(K-1)
            INODE = IAD(L,I)
            SUM = SUM - B(L,I)*DU(INODE,ICUR,NN,MAXNB,IAD,IADN,IADD,B)
   70     CONTINUE
          B(J,I) = D*SUM
   60   CONTINUE

   20 CONTINUE

      RETURN
      END

*************************************************************************

      FUNCTION DU (I,INODE,NN,MAXNB,IAD,IADN,IADD,B)

*     SEARCHES THE I'TH ROW OF THE UPPER DIAGONAL MATRIX
*     FOR AN ADJACENCY TO THE NODE 'INODE'

*     RETURNS CORRESPONDING VALUE OF B (OR ZERO)
```

```
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION IAD(MAXNB,NN),IADN(NN),IADD(NN),B(MAXNB,NN)

      TEMP = 0.0D0
      N = IADN(I)
      K = IADD(I)

      IF (I.EQ.INODE) THEN
        TEMP = 1.0D0
        GO TO 20
      ENDIF

      DO 10 J=(K+1),N
        IF (INODE.EQ.IAD(J,I)) THEN
          TEMP = B(J,I)
          GO TO 20
        ENDIF
   10 CONTINUE

   20 CONTINUE
      DU = TEMP

      RETURN
      END

*****************************************************************************

      SUBROUTINE ORTHOMIN (R,C,GT,NN,MAXNB,MAXNN,IAD,IADN,IADD,B,VRV,
     !                     RES,RQI,RQ,Q,QI,RQIDOT,ECNVRG,RCNVRG,ACNVRG,
     !                     NORTH,MNORTH,MAXIT)

*     ORTHOMIN OR CONJUGATE GRADIENT ACCELERATION/SOLUTION

*     CONJUGATE GRADIENT (SYMMETRIC MATRIX) IF NORTH=0
*     (HOWEVER, NOTE THAT MNORTH MUST BE AT LEAST 1)

*     REQUIRES FUNCTIONS SDOT,SDOTK,SNRM
*     REQUIRES SUBROUTINES LUSOLV,MATM2,SAXPYK,SCOPY,SCOPYK

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION R(MAXNB,NN),C(NN),GT(NN),IAD(MAXNB,NN),IADN(NN),
     !          IADD(NN),B(MAXNB,NN),VRV(NN),RES(NN),RQI(MAXNN,MNORTH),
     !          RQ(NN),Q(NN),RQIDOT(MNORTH),QI(MAXNN,MNORTH)

*     INITIALIZE RESIDUAL VECTOR

      CALL MATM2 (RES,R,C,NN,IAD,IADN,MAXNB)

*     Solution for homogeneous system of equations - Modified by Simunek
      iHomog=0
      DO 10 I=1,NN
        RES(I) = GT(I) - RES(I)
        if(abs(GT(i)).gt.1.d-300) iHomog=1
   10 CONTINUE
      if(iHomog.eq.0) then
        do 11 i=1,NN
                              C(i)=GT(i)
11      continue
        return
      end if

*     LOOP OVER A MAXIMUM OF MAXIT ITERATIONS

      NORCUR = 0

      DO 100 ITER=1,MAXIT

*     INVERT LOWER/UPPER MATRICES

        CALL SCOPY (NN,RES,VRV)

        CALL LUSOLV (NN,MAXNB,IAD,IADN,IADD,B,VRV)
```

```
*     COPY V INTO Q

        CALL SCOPY (NN,VRV,Q)

*     CALCULATE PRODUCT OF R AND V

        CALL MATM2 (VRV,R,Q,NN,IAD,IADN,MAXNB)

*     COPY RV INTO RQ

        CALL SCOPY (NN,VRV,RQ)

*     RES V DOT PRODUCT (CONJUGATE GRADIENT)

        IF (NORTH.EQ.0) THEN

          DOT = SDOT(NN,RES,Q)
          IF (NORCUR.EQ.0) RESV = DOT

        ENDIF

*     LOOP OVER PREVIOUS ORTHOGONALIZATIONS

        K = 1

   20   IF (K.GT.NORCUR) GO TO 30

*     DETERMINE WEIGHTING FACTOR (CONJUGATE GRADIENT)

          IF (NORTH.EQ.0) THEN

            ALPHA = DOT/RESV
            RESV = DOT

*     DETERMINE WEIGHTING FACTOR (ORTHOMIN)

          ELSE

            DOT = SDOTK(NN,K,RQI,VRV,MAXNN,MNORTH)
            ALPHA = -DOT/RQIDOT(K)

          ENDIF

*     SUM TO OBTAIN NEW Q AND RQ

          CALL SAXPYK (NN,ALPHA,K,QI,Q,MAXNN,MNORTH)
          CALL SAXPYK (NN,ALPHA,K,RQI,RQ,MAXNN,MNORTH)

          K = K + 1
          GO TO 20
   30   CONTINUE

*     CALCULATE WEIGHTING FACTOR (CONJUGATE GRADIENT)

        IF (NORTH.EQ.0) THEN

          DOT = SDOT(NN,Q,RQ)
          OMEGA = RESV/DOT

*     CALCULATE WEIGHTING FACTOR (ORTHOMIN)

        ELSE

          DOT = SDOT(NN,RES,RQ)
          RQNORM = SDOT(NN,RQ,RQ)
          OMEGA = DOT/RQNORM

        ENDIF

*     SAVE VALUES FOR FUTURE ORTHOGONALIZATIONS

        NORCUR = NORCUR + 1
        IF (NORCUR.GT.NORTH) NORCUR = 1
```

```
        CALL SCOPYK (NN,NORCUR,Q,QI,MAXNN,MNORTH)
        CALL SCOPYK (NN,NORCUR,RQ,RQI,MAXNN,MNORTH)
        RQIDOT(NORCUR) = RQNORM

*     UPDATE SOLUTION/RESIDUAL VECTORS

        CALL SAXPYK (NN,OMEGA,NORCUR,QI,C,MAXNN,MNORTH)
        CALL SAXPYK (NN,-OMEGA,NORCUR,RQI,RES,MAXNN,MNORTH)

*     DETERMINE CONVERGENCE PARAMETERS

        RESMAX = SNRM(NN,RES)
        DXNORM = DABS(OMEGA)*SNRM(NN,Q)
        CNORM = SNRM(NN,C)
        XRATIO = DXNORM/CNORM

*     ITERATION (DEBUG) OUTPUT

*     STOP ITERATING IF CONVERGED

        IF (RESMAX.LT.ECNVRG) GO TO 200
        IF (XRATIO.LT.RCNVRG) GO TO 200
        IF (DXNORM.LT.ACNVRG) GO TO 200

  100 CONTINUE

*     TERMINATE IF TOO MANY ITERATIONS

      WRITE (* ,602) MAXIT,RESMAX,XRATIO,DXNORM
      WRITE (70,602) MAXIT,RESMAX,XRATIO,DXNORM

 602  FORMAT (///5X,'ORTHOMIN TERMINATES -- TOO MANY ITERATIONS',
     *          /8X,'MAXIT  = ',I5,
     *          /8X,'RESMAX = ',E12.4,
     *          /8X,'XRATIO = ',E12.4,
     *          /8X,'DXNORM = ',E12.4)

      STOP

  200 CONTINUE

*     RETURN NUMBER OF ITERATIONS REQUIRED

      ITERP = ITER

      RETURN
      END

*************************************************************************

      SUBROUTINE LUSOLV (NN,MAXNB,IAD,IADN,IADD,B,VRV)

*     LOWER DIAGONAL MATRIX INVERSION BY FORWARD SUBSTITUTION
*     UPPER DIAGONAL MATRIX INVERSION BY BACKWARD SUBSTITUTION
*     LOWER/UPPER MATRICES ARE IN B
*     RIGHT-HAND-SIDE VECTOR IS IN VRV AT START
*     'SOLUTION' IS RETURNED IN VRV UPON EXIT

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION IAD(MAXNB,NN),IADN(NN),IADD(NN),B(MAXNB,NN),VRV(NN)

*     LOWER INVERSION

      DO 20 I=1,NN
        SUM = VRV(I)
        K = IADD(I)
        DO 30 J=1,(K-1)
          INODE = IAD(J,I)
          SUM = SUM - B(J,I)*VRV(INODE)
   30   CONTINUE

        VRV(I) = B(K,I)*SUM
```

```
   20 CONTINUE

*     UPPER INVERSION

      DO 40 I=NN,1,-1
        SUM = VRV(I)
        N = IADN(I)
        K = IADD(I)
        DO 50 J=(K+1),N
          INODE = IAD(J,I)
          SUM = SUM - B(J,I)*VRV(INODE)
   50   CONTINUE

        VRV(I) = SUM

   40 CONTINUE

      RETURN
      END

*******************************************************************************

      SUBROUTINE MATM2 (S1,R,P,NN,IAD,IADN,MAXNB)

*     MULTIPLY MATRIX R BY VECTOR P TO OBTAIN S1

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION S1(NN),P(NN),R(MAXNB,NN),IAD(MAXNB,NN),IADN(NN)

      DO 30 I=1,NN
        SUM = 0.0D0
        N = IADN(I)
        DO 40 J=1,N
          INODE = IAD(J,I)
          SUM = SUM + R(J,I)*P(INODE)
   40   CONTINUE

        S1(I) = SUM
   30 CONTINUE

      RETURN
      END

*******************************************************************************

      FUNCTION SDOT (NN,R,B)

*     OBTAIN DOT PRODUCT OF R AND B

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION R(NN),B(NN)

      SDOT = 0.0D0
      DO 100 L=1,NN
        SDOT = SDOT + R(L)*B(L)
100   CONTINUE

      RETURN
      END

*******************************************************************************

      FUNCTION SDOTK (NN,K,R,B,MAXNN,MNORTH)

*     OBTAIN DOT PRODUCT OF R AND B

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION R(MAXNN,MNORTH),B(NN)

      SDOTK = 0.0D0
      DO 100 L=1,NN
        SDOTK = SDOTK + R(L,K)*B(L)
100   CONTINUE
```

```
      RETURN
      END
*******************************************************************************

      FUNCTION SNRM (NN,R)

*     COMPUTE MAXIMUM NORM OF R

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION R(NN)

      SNRM = 0.0D0
      DO 100 L=1,NN
        TEMP = DABS(R(L))
        IF (TEMP.GT.SNRM) SNRM = TEMP
100   CONTINUE

      RETURN
      END
*******************************************************************************

      SUBROUTINE SAXPYK (NN,SA,K,FX,FY,MAXNN,MNORTH)

*     MULTIPLY VECTOR FX BY SCALAR SA AND ADD TO VECTOR FY

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION FX(MAXNN,MNORTH),FY(NN)

      IF (NN.GT.0) THEN
        DO 100 I=1,NN
           FY(I) = SA*FX(I,K) + FY(I)
100     CONTINUE
      ENDIF

      RETURN
      END
*******************************************************************************

      SUBROUTINE SCOPY (NN,FX,FY)

*     COPY VECTOR FX INTO VECTOR FY

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION FX(NN),FY(NN)

      IF (NN.GT.0) THEN
        DO 100 I=1,NN
           FY(I) = FX(I)
100     CONTINUE
      ENDIF

      RETURN
      END
*******************************************************************************

      SUBROUTINE SCOPYK (N,K,FX,FY,MAXNN,MNORTH)

*     COPY VECTOR FX INTO VECTOR FY

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION FX(N),FY(MAXNN,MNORTH)

      IF (N.GT.0) THEN
        DO 100 I=1,N
           FY(I,K) = FX(I)
100     CONTINUE
      ENDIF

      RETURN
```

```
      END
* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

11. REFERENCES

Bear, J. 1972. *Dynamics of Fluid in Porous Media.* Elsevier, New York.

Behie, A., and P. K. W. Vinsome. 1982. Block iterative methods for fully implicit reservoir simulation, *Soc. Pet. Eng. J., 22,* 658-668.

Belmans, C., J. G. Wesseling, and R. A. Feddes. 1983. Simulation model of the water balance of a cropped soil: SWATRE, *J. Hydrol., 63,* 271- 286.

Celia, M. A., E. T. Bouloutas, and R. L. Zarba. 1990. A general mass-conservative numerical solution for the unsaturated flow equation, *Water Resour. Res., 26,* 1483-1496.

Christie, L. D., D. F. Griffiths, A. R. Mitchell, and O. C. Zienkiewicz. 1976. Finite element methods for second order differential equations with significant first derivatives, *Int. J. Num. Methods in Engineering, 10,* 1389-1396.

Císlerová, M. 1987. Comparison of simulated water balance for ordinary and scaled soil hydraulic characteristics, *Publ. No. 82,* Dept. of Hydraulics and Catchment hydrology, Agricultural Univ., Wageningen, The Netherlands.

Cleary, R. W., and M. J. Ungs. 1978. Groundwater pollution and hydrology, Mathematical models and computer programs, *Research Report No. 78-WR-15,* Water Resour. Program, Princeton Univ. Princeton, New Jersey.

Davis, L. A., and S. P. Neuman. 1983. Documentation and user's guide: UNSAT2 - Variably saturated flow model, *Final Report, WWL/TM-1791-1,* Water, Waste & Land, Inc., Ft. Collins, Colorado.

Feddes, R. A., E. Bresler, and S. P. Neuman. 1974. Field test of a modified numerical model for water uptake by root systems, *Water Resour. Res., 10*(6), 1199-1206.

Feddes, R. A., P. J. Kowalik, and H. Zaradny. 1978. *Simulation of Field Water Use and Crop Yield,* Simulation Monographs, 188 p., Pudoc, Wageningen, The Netherlands.

Fipps, G., R. W. Skaggs, and J. L. Nieber. 1986. Drain as a boundary condition in finite element, *Water Resour. Res., 22*(11), 1613-1621.

Hopmans, J. W., and J. N. M. Stricker. 1989. Stochastic analysis of soil water regime in a watershed, *J. Hydrol., 105,* 57-84.

Huyakorn, P. S., and G. F. Pinder. 1983. *Computational Methods in Subsurface Flow,* Academic press, London, United Kingdom.

Javandel, I., Ch. Doughty, Chin-Fu Tsang. 1984. *Groundwater Transport: Handbook of Mathematical Models*, Water Resour. Monograph No. 10, 228 p., Am. Geophys. Union, Washington, D.C.

Kirkham, D. 1949. Flow of ponded water into drain tubes in soil overlying an impervious layer, *Transactions, Amer. Geoph. Union, 30*(3), 369-385.

Leij, F. J., T. H. Skaggs, and M. Th. van Genuchten. 1991. Analytical solutions for solute transport in three-dimensional semi-infinite porous media, *Water Resour. Res., 27*(10), 2719-2733.

Letniowski, F. W. 1989. An overview of preconditioned iterative methods for sparse matrix equations. *Research Report CS-89-26,* Faculty of Mathematics, Univ. of Waterloo, Waterloo, Ontario, Canada.

Luckner, L., M. Th. van Genuchten, and D. R. Nielsen. 1989. A consistent set of parametric models for the two-phase flow of immiscible fluids in the subsurface. *Water Resour. Res., 25*(10), 2187-2193.

Lynch, D. 1984. Mass conservation in finite element groundwater models. *Adv. Water Resour., 7*, 67-75.

McCord, J. T. 1991. Application of second-type boundaries in unsaturated flow modeling. *Water Resour. Res., 27*(12), 3257-3260.

Meijerink, J. A. , and H. A. van der Vorst. 1977. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation, 31*(137), 148-162.

Mendoza, C. A., R. Therrien, and E. A. Sudicky. 1991. *ORTHOFEM User's Guide, Version 1.02.* Waterloo Centre for Groundwater Research, Univ. of Waterloo, Waterloo, Ontario, Canada.

Miller, E. E., and R. D. Miller. 1956. Physical theory for capillary flow phenomena, *J. Appl. Phys., 27*, 324-332.

Millington, R. J., and J. M. Quirk. 1961. Permeability of porous solids, *Trans. Faraday Soc.*, 57, 1200-1207.

Mls, J. 1982. Formulation and solution of fundamental problems of vertical infiltration, *Vodohosp. Čas., 30*, 304-313 (in Czech).

Mohammad, F. S., and R. W. Skaggs. 1983. Drain tube opening effects on drain inflow, *J. Irrig. Drain. Div., Am. Soc. Civ. Eng., 109*(4), 393-404.

Mualem, Y. 1976. A new model for predicting the hydraulic conductivity of unsaturated porous media, *Water Resour. Res., 12*(3), 513-522.

Neuman, S. P. 1972. Finite element computer programs for flow in saturated-unsaturated porous media, *Second Annual Report, Part 3, Project No. A10-SWC-77*, 87 p. Hydraulic Engineering Lab., Technion, Haifa, Israel.

Neuman, S. P. 1973. Saturated-unsaturated seepage by finite elements, *J. Hydraul. Div.*, ASCE, 99 (HY12), 2233-2250.

Neuman, S. P. 1975. Galerkin approach to saturated-unsaturated flow in porous media, Chapter 10 in *Finite Elements in Fluids, Vol. I, Viscous Flow and Hydrodynamics*, edited by R. H. Gallagher, J. T. Oden, C. Taylor, and O.C. Zienkiewicz, pp. 201-217, John Wiley and Sons, London.

Neuman, S. P., R. A. Feddes, and E. Bresler. 1974. Finite element simulation of flow in saturated-unsaturated soils considering water uptake by plants, *Third Annual Report, Project No. A10-SWC-77*, Hydraulic Engineering Lab., Technion, Haifa, Israel.

Perrochet, P., and D. Berod. 1993. Stability of the standard Crank-Nicolson-Galerkin scheme applied to the diffusion-convection equation: some new insights, *Water Resour. Res., 29*(9), 3291-3297.

Pinder, G. F., and W. G. Gray. 1977. *Finite Element Simulation in Surface and Subsurface Hydrology*, Academic Press, New York, N.Y.

Rogers, J. S., J. L. Fouss. 1989. Hydraulic conductivity determination from vertical and horizontal drains in layered soil profiles, *Transaction of the ASAE, 32*(2), 589-595.

Simmons, C. S., D. R. Nielsen, and J. W. Biggar. 1980. Scaling of field-measured soil water properties, *Hilgardia, 47*, 101-122.

Šimůnek, J., T. Vogel, and M. Th. van Genuchten. 1992. The SWMS_2D code for simulating water flow and solute transport in two-dimensional variably saturated media, Version 1.1. *Research Report No. 126*, 169 p., U.S. Salinity Laboratory, USDA, ARS, Riverside, California.

Šimůnek, J., and D. L. Suarez. 1993. UNSATCHEM-2D code for simulating two-dimensional variably saturated water flow, heat transport, carbon dioxide production and transport, and multicomponent solute transport with major ion equilibrium and kinetic chemistry, Version 1.1. *Research Report No. 128*, 218 p., U.S. Salinity Laboratory, USDA, ARS, Riverside, California.

Sisson, J. B. 1987. Drainage from layered field soils: Fixed gradient models, *Water Resour. Res., 23*(11), 2071-2075.

Skaggs, R. W., E. J. Monke, and L. F. Huggins. 1970. An approximate method for determining the hydraulic conductivity function of an unsaturated soil, *Techn. Report No. 11*, Water Resour. Res. Center, Purdue University, Lafayette, Indiana.

Sudicky, E. A., and P. S. Huyakorn. 1991. Contaminant migration in imperfectly known heterogeneous groundwater systems, *Reviews of Geophysics*, Supplement, U.S. National Report. to Int. Union of Geodesy and Geophysics 1987-1990, pp. 240-253, Am. Geophys. Union, Washington, DC.

van Genuchten, M. Th. 1976. On the accuracy and efficiency of several numerical schemes for solving the convective-dispersive equation, in *Finite elements in Water Resources*, edited by W. G. Gray et al., pp. 1.71-1.90, Pentech Press, London.

van Genuchten, M. Th. 1978. Mass transport in saturated-unsaturated media: one-dimensional solutions, *Research Report No. 78-WR-11*, 118 p., Water Resources Program, Princeton Univ., Princeton, New Jersey.

van Genuchten, M. Th. 1980. A closed-form equation for predicting the hydraulic conductivity of unsaturated soils, *Soil Sci. Soc. Am. J., 44*, 892-898.

van Genuchten, M. Th., and J. Parker. 1984. Boundary conditions for displacement experiment through short laboratory soil columns, *Soil Sci. Soc. Am. J., 48*, 703-708.

Vimoke, B. S., and G. S. Taylor. 1962. Simulating water flow in soil with an electric resistance network, *Report No. 41-65*, 51 p., Soil and Water Conserv. Res. Div., U. S. Agric. Res. Serv., Columbus, Ohio.

Vimoke, B. S., T. D. Yura, T. J. Thiel, and G. S. Taylor. 1963. Improvements in construction and use of resistance networks for studying drainage problems, *Soil Sci. Soc. Am. J., 26*(2), 203-207.

Vogel, T. 1987. SWMII - Numerical model of two-dimensional flow in a variably saturated porous medium, *Research Report No. 87*, Dept. of Hydraulics and Catchment Hydrology, Agricultural Univ., Wageningen, The Netherlands.

Vogel, T., and M. Císlerová. 1988. On the reliability of unsaturated hydraulic conductivity calculated from the moisture retention curve, *Transport in Porous Media, 3*, 1-15.

Vogel, T., M. Císlerová, and J. W. Hopmans. 1991. Porous media with linearly variable hydraulic properties, *Water Resour. Res., 27*(10), 2735-2741.

Wesseling, J. G., and T. Brandyk. 1985. Introduction of the occurrence of high groundwater levels and surface water storage in computer program SWATRE, *Nota 1636*, Institute for Land and Water Management Research (ICW), Wageningen, The Netherlands.

Yeh, G. T., and V. S. Tripathi. 1990. HYDROGEOCHEM: A coupled model of HYDROlogic transport and GEOCHEMical equilibria in reactive multicomponent systems, *Environm. Sci. Div., Publ. No. 3170.*, Oak Ridge National Lab., Oak Ridge, Tennessee.

Zienkiewicz, O. C. 1977. *The Finite Element Method*, 3rd ed., McGraw-Hill, London, United Kingdom.