LBL-31050
UC-800

VHBORE: A Code to Compute Borehole Fluid Conductivity Profiles With Pressure Changes in the Borehole

F.V. Hale and C.F. Tsang

Earth Sciences Division
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

June 1994

This work was carried out under U.S. Department of Energy Contract No. DE-AC03-76SF00098 for the Director, Office of Civilian Radioactive Waste Management, Office of External Relations, administered by the Nevada Operations Office in cooperation with the Swiss National Cooperative for Disposal of Radioactive Waste (Nagra).

[illegible]

Table of Contents

1. INTRODUCTION

This report describes the code VHBORE which can be used to model fluid electric conductivity profiles in a borehole intersecting fractured rock under conditions of changing pressure in the well bore. Pressure changes may be due to water level variations caused by pumping or fluid density effects as formation fluid is drawn into the borehole. Previous reports describe the method of estimating the hydrologic behavior of fractured rock using a time series of electric conductivity logs (Tsang et al., 1989), and an earlier code, BORE, to generate electric conductivity logs under constant pressure and flow rate conditions (Hale and Tsang, 1988).

The earlier model, BORE, assumed a constant flow rate, q_i, for each inflow into the well bore. In the present code the user supplies the location, constant pressure, h_i, transmissivity, T_i, and storativity, S_i, for each fracture, as well as the initial water level in the well, $h_w(0)$, In addition, the input data contains changes in the water level at later times, $\Delta h_w(t)$, typically caused by turning a pump on or off. The variable density calculation also requires input of the density of each of the inflow fluids, ρ_i, and the intial uniform density of the well bore fluid, $\rho_w(0)$. These parameters are used to compute the flow rate for each inflow point at each time step.

The numerical method of Jacob and Lohman (1952) is used to compute the flow rate into or out of the fractures based on the changes in pressure in the wellbore. A dimensionless function relates flow rate as a function of time in response to an imposed pressure change. The principle of superposition is used to determine the net flow rate from a time series of pressure changes. Additional reading on the relationship between drawdown and flow rate can be found in Earlougher (1977), particularly his Section 4.6, "Constant-Pressure Flow Testing."

The primary difference between variable and constant density applications of the code is that with variable densities the pressure profile is computed at each time step, and when density variations in the well bore result in a significant pressure change (i.e. equal to a column of water of height equal to one half of the cell height, $\rho g \Delta x/2$), this is also added to the flow rate calculations. The wellbore is assumed to be vertical for purposes of pressure calculations; however no gravity-driven flow effects are considered in this code.

The electrolyte concentration and the density are not related in the present code. Therefore the code can be considered to model the transport of two independent substances using the same flow field. Only the electrolyte is affected by diffusion in the current code, and only the density affects the pressure profile. The decision to separate the density calculation from the electrolyte concentration was made to permit modeling of conditions where the density effects are due to something other than the electrolyte solutions, such as drilling mud. Full radial mixing is assumed in the cells, and fingering effects are not considered.

2. NUMERICAL AND ANALYTICAL SOLUTIONS

Governing Equation for Borehole Flow with Sources

The differential equation for mass or solute transport in a borehole is:

$$\frac{\partial}{\partial x}\left(K\frac{\partial C}{\partial x}\right) - \frac{\partial}{\partial x}(CV) + S = \frac{\partial C}{\partial t} \qquad (1)$$

where

C is the concentration (kg/m^3)

K is the dispersion coefficient (m^2/sec)

S is the source term (kg/m^3·sec), and

V is the fluid velocity (m/sec)

This partial differential equation is solved numerically using the finite difference method (FDM). The following initial and boundary conditions are also specified:

$$C(x,0) = C_o(x) \tag{2a}$$

$$C(x > x_{max}, t) = 0 \tag{2b}$$

$$K = 0 \quad \text{for} \quad x > x_{max} \quad \text{or} \quad x < x_{min} \tag{2c}$$

The first condition allows for the specification of initial electrolyte concentrations in the borehole. The second condition implies that there is no electrolyte in the borehole fluid flowing from below the area of interest. If there is a background concentration in the fluid flowing from the borehole bottom, this value should be added to all of the resulting concentrations. The third condition indicates that dispersion does not take place across the specified boundaries of the area of interest. In general, advection will be the dominant process at the boundaries. If dispersion is dominant for a particular problem, the boundaries should be extended in order to prevent improper trapping of electrolyte.

This is the same governing equation used by the constant flow code, BORE. The differences in VHBORE are seen in the source, velocity and, if velocity dependent, dispersion terms. The source term, S_i, is no longer constant, but include the effects of two-way flow between the well bore and the formation as the pressure changes in the borehole. The velocity term, V_i, varies in response to the pressure history of the well, and thus, if the dispersion coefficient, K_i, is velocity dependent, it will also vary continuously with time.

Discretization in Space

In the borehole, uniform, one-dimensional spacing of nodes is used. It is assumed that the borehole has uniform diameter d, and that the region of interest is divided into N equal length cells of length Δx. Position values indicate depth in the borehole; thus x is zero at the surface and increases downward. The flow within the borehole is generally upward, and the cell index i increases downstream (upward, toward the surface). Thus cells 1 and N are located at the bottom and top of the region, respectively, and node x_i is upstream of and at a greater depth than node x_{i+1}. In general, node i is located at $x_{max} - (i - 1/2)\Delta x$, with boundaries of $x_{max} - (i - 1)\Delta x$ upstream (at a depth below the node), and $x_{max} - (i)\Delta x$ downstream (at a depth above the node). Note that because all cells are assumed to have the same geometry, flow rates are directly proportional to linear velocities.

Each inflow is given a specific location in the input file, and the inflows are then assigned to specific cells. If multiple inflows are assigned to the same cell, their flow rates and mass transfer rates are summed to produce a single source term for the cell. The single source for the cell is assumed to be located at the midpoint of the cell.

The BORE code modeled a fracture as an infinite source of fluid with zero concentration until a specified time, after which time the infinite source provides fluid with a constant concentration. In the codes described here, a set of 50 "buckets" of variable volume and concentration have been set between the borehole and this infinite fluid source of fixed concentration. These buckets represent no particular geometry, but rather just a volume of water with a particular concentration.

As fluid flows from the borehole to the fracture, the buckets are filled, one at each time step. In the code, this is accomplished by storing a volume and concentration in corresponding positions in the matrices VFRAC and CFZ, which are indexed by fracture and bucket. The volume and concentration of the fluid transferred to the fracture depend on the time step, flow rate, and local borehole fluid concentration. If fluid has been flowing into the fracture for some time, and all of the buckets are full (i.e., all elements of the matrices VFRAC and CFZ for the fracture are in use), the oldest ten volumes are combined (i.e., dumped into one bucket, providing nine more empty buckets) by summing the volumes and computing a new concentration based on the total mass.

As fluid flows from the fracture zone to the borehole, the buckets are emptied in the reverse order. If all of the buckets are empty, fluid is drawn from the infinite source of constant concentration. If a number of buckets are necessary to meet the volume requirements for a single time step, the concentrations are averaged.

The user specifies an initial volume and concentration in the first bucket. There is no longer an initial period of zero concentration, rather fluid transfer between the borehole and the fracture begins immediately. It is possible, however, to specify an intial volume of infiltrated water with a zero concentration.

The concentration of the fluid moving between the borehole and the fracture zone (in either direction) is determined by subroutine CTFRAC which is executed at each time step. Subroutine TSTEP computes the actual mass transfer from the borehole to the fracture zone.

Discretization in Time

Because the flow rates are not constant, a constant time step is not practical. Rather than having the user specify a time step, the subroutines FLOWS, DFLOWS, FLOWSA and TSTEP work together to compute a variable time step based on maximum velocity and mass transfer rates. The time step is initially set by FLOWS or DFLOWS based on flow rates (velocities) in the previous time step, if possible. Then FLOWSA modifies the time step based on the expected flow rates during the present time step. If due to dispersion effects this time step results in mass transfer inconsistencies, the time step is further reduced by subroutine TSTEP.

Methods of Computing the Dispersion Coefficient

Within the code, three methods are available for determining the dispersion coefficient for use at the interface between each pair of cells, $K_{i\pm 1/2}$: constant, velocity scaled, and velocity squared scaled. The first approach is used to model dispersion due to molecular diffusion; the second, velocity dependence, is an approximation for porous medium transport; and the last, velocity-squared dependence, corresponds to Taylor dispersion for flow in a pipe. With each method, the dispersion coefficients at the two adjacent cells to an interface are computed, then the harmonic mean is used at the interface. Because no dispersion occurs across the region boundaries, $K_{1/2}$ and $K_{N+1/2}$ are defined to be zero.

With the constant method, the input dispersion parameter, K_o, is used for all the cell interfaces and Equation (1) simplifies to

$$K_o \frac{\partial^2 C}{\partial x^2} - \frac{\partial}{\partial x}(CV) + S = \frac{\partial C}{\partial t} \tag{3}$$

The velocity scaled methods use a somewhat arbitrary reference dispersion coefficient K_o defined as the dispersion coefficient at a depth where the flow velocity is equal to the mean velocity or the mean velocity squared,

$$\overline{V^n} = \frac{\min(V_i^n) + \max(V_i^n)}{2} \quad , \tag{4}$$

where V_i^n is the fluid flow velocity at node i raised to the first or second power (n = 1 or 2). Then the dispersion coefficient for node i is given by

$$K_i = K_o \left(\frac{V_i^n}{\overline{V^n}} \right) \quad . \tag{5}$$

Note that since the cells have a uniform volume the velocities are proportional to the flow rates, and the actual calculations are based on q_i rather than v_i (since $q_i = v_i A$, where A is the uniform cross-sectional area).

The dispersion coefficient at the interface between two cells is the harmonic mean:

$$K_{i \pm 1/2} = 1\left(\frac{1}{K_i} + \frac{1}{k_i \pm 1} \right)^{-1} \tag{6}$$

For cells with no flow (e.g., upstream from the first feed point), the dispersion coefficient of the first cell with nonzero flow is used.

The dispersion coefficients are then adjusted for the problem geometry,

$$\gamma_{i+1/2} = \frac{AK_{i+1/2}}{\Delta x} \qquad (7)$$

where A is the uniform cross-sectional area.

If the dispersion type is flow rate dependent (ITYPDK is 2 or 3), the dispersion coefficients must be computed during each time step. This is done in subroutine FLOWSA after the current flow rates and velocities have been computed by subroutines FLOWS or DFLOWS.

Calculation of Flow Rates

At a given time the flow rate from a fracture zone to the borehole (or from the borehole to the fracture zone) is a function of the fracture zone hydraulic parameters and the pressure history in the borehole. At each time step the flow rate is computed by a superposition of the effects of individual pressure changes. This calculation is performed in subroutine FLOWS for constant density calculations and by subroutine DFLOWS for variable density calculations.

The effect of a single pressure change, j, for a single inflow, i, is computed using a dimensionless function, q_D, which is defined as follows:

$$q_{i,j}(t) = 2\pi T_i \Delta h_j q_D(t_D(t)) \qquad (8)$$

where

$$t_D(t) = \left(\frac{T_i(t-\tau_j)}{S_i r^2} \right) \qquad (9)$$

where T_i is the transmissivity of the inflow, S_i is the storativity of the inflow, r is the well bore radius, Δh_j is the drawdown for pressure change j, and τ_j is the time of pressure change j.

In these codes, the terms $2\pi T_i$ and $T_i/S_i r_2$ are calculated in subroutine RDFRAC and stored as QCOEF(I) and TCOEF(I), respectively. The dimensionless function q_D is computed using linear interpolation on a table for arguments values between 10^{-4} and 10^{12}. The table is from Jacob and Lohman (1952). For dimensionless times (t_D) greater than 10^{12}, Earlougher (1977) suggests using $q_D = 2/(\ln(t_D) + 0.80907)$. For values below 10^{-4}, a somewhat arbitrary constant value of 56.9 is used (this is the value for dimensionless time 10^{-4} given by Jacob and Lohman). Figure 1 shows the graph of the function q_D.

The calculation of flow rates is more complex for variable density cases. As the fluid of different densities enters the well bore, the pressure profile in the well bore changes. The pressure is no longer just a function of the drawdown values specified in the input file, but also of the changing density profile in the well. At any given time, the pressure (in meters of water) can be calculated as:

$$p_i(t) = \frac{1}{\rho_w} \int_{x_w}^{x_i} \rho(x,t)\, dx \tag{10}$$

where ρ_w is the density of water, g is the acceleration due to gravity, and x_w is the depth of the water surface.

The use of the Jacob-Lohman solution requires that at every time step following a pressure change the effect of the pressure change be computed. In order to provide for some reasonable number of pressure changes to be recorded during the run, the wellbore pressure at each inflow in discretized in time. If a pressure change at an inflow exceeds one half of a cell length, it is considered significant, recorded, and affects all following flow rate calculations. All of the is done in subroutine DFLOWS. The array PHIST stores the pressure history (pressure and time) for each inflow. Stated more concisely, the current criterion for recording a significant pressure change due to density is:

$$\left| p_i(t_1) - p_i(t_2) \right| > \Delta x / 2 \tag{11}$$

This condition make the assumption that the water level, x_W, does not change between times t_1 and t_2.

After the flows between the fracture zones and the borehole have been determined, the flow rates in the remaining cells are computed by subroutine FLOWSA assuming a closed lower boundary and open upper boundary in the borehole, using the discretized version of Equation (1).

Note that if the input value for the initial drawdown and inflow depths and pressure heads results in pressure differences between the input values for the fractures, it is assumed that this pressure change took place at the model starting time (TSTART).

Calculation During Each Time Step

The constant mass injection of BORE has been replaced in the variable pressure codes by possible two-way mass transfer between the borehole and the fractures. At each time step the flow rate (and direction) and fluid concentration for transfer between the borehole and the fracture zones is computed. This has been described above.

The mass transfer within the borehole during time step k is due to flow to or from the feed points (the source of electrolyte), advection, and dispersion. The finite-difference version of Equation (1) may be written in terms of mass transfer (kg/sec) as follows:

$$\begin{aligned}(\Delta C_{i,k} / \Delta t)(A\Delta x) = {} & C^f_{i,k}(q^f_i) \\ & +C_{i-1,k-1}(q_{i-1/2}) - C_{i,k-1}(q_{i+i/2}) \\ & +(C_{i-1,k-1} - C_{i,k-1})(\gamma_{i-1/2}) - (C_{i,k-1} - C_{i+1,k-1})(\gamma_{i+1/2})\end{aligned} \qquad (12)$$

where $C^f_{i,k}$ is the average concentration of electrolyte in the fluid flowing from the feed points into cell i during time step k, and $\gamma_{i\pm1/2}$ is the dispersion coefficient at the interface between two adjoining cells adjusted for the problem geometry. The first line of the right-hand side is the source term, the second line is the advection term, and the third line is the dispersion term. Upstream weighting is used in the advective terms.

This equation can be rewritten by collecting coefficients of the different cell concentrations as:

$$
\begin{aligned}
(\Delta C_{i,k} / \Delta t)(A\Delta x) = {} & C^{f}_{i,k}(q^{f}_{i}) \\
& +C_{i-1,k-1}(q_{i-1/2} + \gamma_{i-1/2}) \\
& +(C_{i,k-1}(-q_{i+i/2} - \gamma_{i-1/2} - \gamma_{i+1/2}) \\
& +C_{i+1,k-1}(\gamma_{i+1/2})
\end{aligned}
\qquad (13)
$$

At each time step k a check is made to verify that the total mass in the cell at the beginning of the time step is greater than or equal to the total mass to be transported out of the cell during the time step. If this condition is not met, an error message is printed and the time step is reduced.

Conservation of mass is verified during each time step and at the end of the problem. Mass may flow into the system from the infinite reservoir in each fracture zone, and mass may flow out of the system at the top of the borehole section. All other boundaries are closed.

Temperature Dependence of Conductivity

All calculations are made assuming a uni\%form temperature of 20°C throughout the borehole. Generally temperature increases with depth below the land surface, so temperature corrections must be applied to field conductivity data to permit direct comparison with model output.

The effect of temperature on conductivity can be estimated using the following equation from NAGRA (1987):

$$\sigma(20^\circ C) = \frac{\sigma(T_x)}{1 + S(T_x - 20^\circ C)} \qquad (14)$$

where T_x is the temperature (°C) at depth x. The value of S is estimated at 0.022.

Conductivity as a Function of Concentration

Assuming that all of the ions in the borehole fluid can be converted to NaCl equivalents, the conductivity and concentration data in Shedlovsky and Shedlovsky (1971) can be fit fairly well using a quadratic approximation:

$$\sigma = 2{,}075\,C - 45\,C^2 \qquad (15)$$

where C is the concentration in kg/{m^3 and σ is the conductivity in µS/cm at 25°C. The expression is accurate for a range of C up to 5 kg/m^3 and σ up to 10,000 µS/cm. For even lower C up to 1 kg/{m^3 } and σ up to 0.2 µS/cm, the second term may be neglected.

Although the experimental values are for 25°C, they may be used at 20°C if multiplied by 0.89 (based on Equation (14) for the temperature dependence of conductivity). Thus the above relationship would be

$$\sigma = 1{,}850\,C - 40\,C^2 \quad . \qquad (16)$$

3. DESCRIPTION OF FORTRAN CODE VHBORE

The main program is little more than a single time step loop and a series of subroutine calls. Before the time step loop begins, a single call to the subroutine INITS is made to initialize all data areas and read the problem descriptions. Within the time step loop, the flow rates are first computed using either constant or variable density approaches (using either subroutine FLOWS or subroutine DFLOWS). Once the flow rates are computed, a time step size is determined, and the mass transfer for the time step is accomplished using subroutine ONESTEP. If the time step was determined to be too large (too much mass was transferred out of a cell), the time step is reduced and the flow rates and mass transfer are preformed again. If the time step was successful, the subroutine GOODSTEP is called to check for output requests. After the ending time has been reached, a single call to the subroutine ENDPROB is made to produce the final messages regarding mass transfer.

Common Blocks

The program has four named common blocks, CORR, FRACS, SEGS and STEP which are defined in a separate source code file, VPCOMMS.F, and included as needed in the main source file. The common block source file uses a set of parameters to specify array sizes, making it easier to change array sizes as need to suit different problems and computing environments.

Common block CORR contains the coefficients for converting electrolyte concentration in kg/m^3 to conductivity in µS/cm. These terms are derived from a second degree polynomial fit to experimental data, and are read by subroutine RDCORR.

Common block FRACS contains the arrays describing the fractures, including the fracture flow rates, concentrations, positions, segment locations, transmissivities, storativities, infiltration volumes, fluid densities, etc.

Common block SEGS contains the arrays describing each segment, including the concentration and density at the beginning and end of the time step, the fracture inflow average concentration and density, external flows into and out of the segment, fracture flow into the segment, downstream flow to the next segment, total flow into the segment, the position of the segment and the dispersion coefficient between the segment and the next downstream segment.

Common block STEP contains other variables used during the time steps, including the step duration, number of segments, maximum number of segments allowed, uniform segment volume, cumulative mass and volume out of the system, cumulative mass and volume into the system, time at the beginning and end of the time step, number of fractures, conductivity output unit number, toggle index for the concentration array, etc.

Subroutines and Functions

The first level of subroutines includes INITS, FLOWS, DFLOWS, ONESTEP, GOODSTEP and ENDPROB. These subroutines are called by the main program.

INITS performs global variable initialization, opens the input and output files, reads the problem definition input file and performs the initial mass balance using subroutine SMASS.

The two main portions of the the time step loop compute the variable flow rates and perform the mass transfer. For a constant density calculation, the variable flow rates are determined by subroutine FLOWS; while for a variable density case, the subroutine DLFLOWS is used. The primary difference between these two routines is that FLOWS using only depth to determine pressure differences (assuming a constant density fluid in the well bore), whereas DFLOWS performs an actual integration of the density of all well bore fluid in order to determine the pressure at some depth. Both subroutines use the function QD to determine individual inflow rates using the Jacob-Lohman solution, and, after the individual inflow rates have been determined, both subroutines also call the subroutine FLOWSA to develop the full flow rate profile, estimate the time step size, and determine any velocity dependent aspects of the calculation.

The mass transfer for a single time step is carried out by subroutine ONESTEP. This subroutine advances the simulation clock, save current state variables in case the time step must be reversed, and calls subroutines CTFRAC and TSTEP. Subroutine CTFRAC determines the average concentration and density of the source terms for the time step based on the inflow rates, the time step, and any storage of fluid in the inflow zones. Subroutine TSTEP peforms the actual mass transfer calculation and checks for conservation of mass. If the subroutine TSTEP indicates that an attempt was made to transfer too much mass out of a cell during the time step (due to the combined effects of advection and diffusion), a flag is set and the time step is reduced.

As mentioned above, the subroutine GOODSTEP is called if the time step was successful in order to determine if any output has been requested following the current time step. Actual output of conductivity profiles is produced by subroutine CPRT.

At the end of the computation, the subroutine ENDPROB is called to produce a number of informational messages about the calculation.

4. INPUT AND OUTPUT GUIDE

Input and Output Files

The model uses one input and four output files. The input file contains the problem description consisting of borehole geometry, top and bottom borehole flows, feed point flows, timing parameters, dispersion parameters, and initial concentrations. The output files consist of (1) messages produced by the model, (2) conductivity-depth pairs for each borehole cell at the requested output times, (3) flow rate-depth pairs for each cell at the requested output times, and (4) density-depth pairs for each cell at the requested output times, The following table summarizes the input and output files and indicates their FORTRAN unit numbers.

Table 4-1. Input and Output Files

UNIT NUMBER	INPUT/OUTPUT	DESCRIPTION
5	INPUT	Problem description
6	OUTPUT	Messages
7	OUTPUT	computed conductivity data
8	OUTPUT	Flow rate profile
9	OUTPUT	Density profile

Problem Description

The problem description is entered in free format, with values being separated by spaces or commas. The number of lines in the problem description will be variable depending on the number of feed points, number of times at which conductivity output is desired, and number of initial concentrations specified. The following table provides a detailed description of each line of the input.

Table 4-2. Input Guide.

LINE	NAME	UNITS	DESCRIPTION
1	XTOP	m	Top of study area, surface is zero and positions increase downward, adjusted if necessary to fit XBOT and DELX
	XBOT	m	Bottom of study area
	DELX	m	Cell length
	DIAM	cm	Borehole diameter (uniform)
	WATLEV	m	Initial water level in well
	TOPD	none	Initial density of well bore fluid normalizedby density of water (spec. grav.). TOPD$\leq$0 for constant density calculation
2	IFLIM	none	Number of inflows
2.I	XIN(I)	m	Position of inflow
I=1,IFLIM	CIN(I,1)	kg/m^3	Constant concentration for formation water
	TFRAC(I)	m^2/sec	Inflow transmissivity
	SFRAC(I)		Inflow storativity
	HFRAC(I)	m	Inflow pressure (over ρg)
	VFRAC(I, 1)	m^3	Inflow initial volume of infiltrated deionized water
	CFZ(I,1,1)	kg/m^3	Infiltrated water concentration
	CIN(I,2)	none	Density of inflow fluid normalized by density of water (spec. grav.) (only for variable density calculation)

Table 4-2. Continued.

3	IDHLIM	none	Number of water level changes
3.I	DELHT(I)	hr	Time at which water level in borehole changes
I-1,IDHLIM	DELHH(I)	m	Change in water level (positive down)
4	TSTART	hr	Problem start time
	TEND	Hr	Problem end time
	TSMAX	min	Maximum time step
	ILPT		Number of print times
4.I I=1,ILPT	PT(I)	hr	Time to print profile
5	ITYPDK	none	Type code for dispersion coefficient 1 means DK is constant over all cells 2 means DK is the value for the mean velocity, $\bar{q}$ = (min(q_i) + max(q_i))/2, k_i = DK($q_i\sqrt{q}$) 3 means KD is the value for mean vel. squared, \O(q^2,$_i$) = (min(q\S(i,2) + max(q_2)/2, K_i = DK((q_2/q^2)
	DK	m^2/sec	Dispersion coefficient
6	ICON	none	Number of initial concentrations
6.I	X(I)	m	Position of initial concentration
I=1,ICON	CO(I)	kg/m^3	Initial concentration
	DO(I)	none	Initial density, normalized by density of water (spec. grav.) (only for variable density calculation)
7	OFSET	µS/cm	constant term for converting concentration in kg/m^3 to conductivity in µS/cm
	COEFA	(µS·m^2)/100 kg	linear ceofficient for converting concentration to conductivity
	COEFB	(µS·m^5)/1000 kg^2	quadratic coefficient for converting concentration to conductivity

The output subroutine CPRT converts concentration to conductivity just prior to output by means of the coefficients in common CORR. If the coefficients 0,1,0 are entered, then the output values are equivalent to the concentrations. The output subroutine CPRT also converts flow rates to liters per minute just prior to output.

5. EXAMPLES

Verification Example: Single inflow, single pressure change

The first example allows verification by comparing with Example 4.4 in Earlougher (1977). A single pressure change is applied to a single inflow. Earlougher's example refers to an inflow zone of length 57.9 meters; this will be represented in two different ways, both as a single inflow and as as a set of 58 inflows with a cell length of one meter. The example applies a pressure drop of 703.6 meters (1000 psi). The hydrological characteristics include a permeability of 6.5 millidarcy and a viscosity of 1.35 centipoise, giving a transmissivity of $2.697 \times 10^{-6} m^2/sec$. The porosity-compressibility product is $2.05 \times 10^{-6}\}$ psi^{-1}, giving a storativity of 168.7×10^{-6}. The single drawdown occurs at time zero hours.

The input for this example problem with the inflow represented as a single point is shown below.

LINE	DATA
1	000, 2000, 1, 60.96, 0, –1
2	1
2.1	1900. 1.00 2.697e-6 168.7e-6 1899.5 0. 0.
3	1
3.1	0 703.57
4	0,100,15,9
4.1	0.2
4.2	0.5
4.3	1
4.4	2
4.5	5
4.6	10
4.7	20
4.8	50
4.9	100
5	1,0.5e-3
6	0
7	73., 1870., –40.

The input for the case of 58 inflows is identical to that above except for lines 2 and 2.x. The transmissivity and storativity values have been scaled by 58:

LINE	DATA
2	58
2.1	1900. 1.00 46.495e-9 2.9092e-6 1899.5 0. 0.
2.2	1901. 1.00 46.495e-9 2.9092e-6 1900.5 0. 0.
2.3 to 57	(omitted)
2.58	1957. 1.00 46.495e-9 2.9092e-6 1956.5 0. 0.

The negative value at the end of the first line of the input indicates a constant density calculation. Both approaches to the inflow description result in the same number of simulation time steps (5,415), ranging from 7.8 to 73 seconds.

Figures 2 through 6 illustrate different aspects of the calculation. Figure 2 shows the flow rate as a function of time. The units are Earlougher's oil-field system of barrels per day (one liter per minute is 9.0573 barrels per day). This figure matches Figure 4.13 in Earlougher (1977). The flow rate is the same regardless of which representation of the inflow zone is used.

The flow rate and conductivity profiles for the single point representation are shown in Figures 3 and 4, and Figure 5 and 6 show the same for the 58-cell representation. The different approaches to modeling the inflow section have fairly limited effects. The flow rate shows a ramp for the line source, as opposed to a step for the point source, and there is a slower buildup of saturated concentration levels with the line source.

Variable Density Examples

In order to demonstrate the effect of well bore fluid density on the flow rate, two variations on the above single inflow, single pressure drop case are presented here. In the first, the well bore fluid is twice as dense as the inflow fluid; and, in the second case, the inflow fluid is twice as dense as the well bore fluid. The changes in the input file are shown below:

LINE	DATA
1	1000, 2000, 1, 60.96, 0, 2 [well bore fluid density high]
2	1
2.1	1900. 1.00 2.697e-6 168.7e-6 1899.5 0. 0. 1. 0.

LINE	DATA
1	1000, 2000, 1, 60.96, 0, 1
2	1
2.1	1900. 1.00 2.697e-6 168.7e-6 1899.5 0. 0. 2. 0. [denity high]

Figures 7 and 8 show the flow rate as a function time for these two cases. In Figure 7, with the well bore fluid having a higher density, the pressure difference between the inflow and the well bore at the inflow depth is initially reduced, so the flow rate is also reduced. As the less dense fluid flows into the well bore and the denser fluid is pumped out, the pressure difference at the inflow depth increases, until, after about 30 hours the denser fluid initially in the well bore has been flushed out, and the problem converges with the constant density solution.

In Figure 8, with the inflow fluid having a higher density, the flow rate is initially similar to the constant density case. But as the denser fluid flows into the well, driving up the pressure at the inflow depth, the flow rate decreases. Eventually, after a few hundred hours, the flow stops since even with the drawdown of over 700 meters, the denser fluid in the well bore has resulted in equilibrium between the well and the inflow at the inflow depth.

Figure 9 shows the driving pressure differences in meters of water as a function of time for these cases.

Multiple Inflow, Multiple Pressure Change

One more complex example is presented which involves three inflows and two drawdown changes. The problem is a variation of the constanst density verification example at the start of this section. The inflow and drawdown change input is as follows: shown below:

LINE	DATA
2	3
2.1	1500. 1.00 2.697e-6 168.7e-6 1399.5 0. 0.
2.2	1700. 1.00 2.697e-6 168.7e-6 1799.5 0. 0.
2.3	1900. 1.00 2.697e–6 168.7e–6 1899.5 0. 0.
3	2
3.1	10 703.57
3.1	50 –703.57

Notice that when the water level is at the surface, there is a circulation within the well from 1700 meters to 1500 meters due to inflow pressure head differences. The water level changes approximate turning a pump on at 10 hours and off at 50 hours.

Figure 10 shows the flow rate across the inflow at 1500 meters as a function of time. Note that initially the flow is negative, i.e. water is flowing from the wellbore into the formation. Figure 11 shows the concentration of the water flowing across the inflow.

Figures 12 and 13 show the flow rate and conductivity profiles for the entire wellbore at times before, during and after pumping.

6. ACKNOWLEDGEMENTS

Discussions and cooperation with NAGRA personnel, especially S. Vomvoris and P. Bluemling are much appreciated. We would also like to thank Christine Doughty of LBL for reviewing and commenting on the manuscript and software. This work was carried out under U.S. Department of Energy Contract No. DE-AC03-76SF00098 for the Director, Office of Civilian Radioactive Waste Management, Office of External Relations, administered by the Nevada Operations Office in cooperation with the Swiss National Cooperative for Disposal of Radioactive Waste (Nagra).

7. REFERENCES

Earlougher, R.C. (1977), "Advances in Well Test Analysis," Society of Petroleum Engineers, New York.

Hale, F.V. and C.F. Tsang (1988), "A code to compute borehole fluid conductivity profiles with multiple feed points," LBL-24928, Lawrence Berkeley Laboratory, University of California, and NDC-2, NAGRA, Baden, Switzerland.

Jacob, C.E. and S.W. Lohman (1952), "Nonsteady flow to a well of constant drawdown in an extensive aquifer," *Transactions, American Geophysical Union*, v. 33, no. 4, pp. 559-569.

NAGRA (1987), Private communication from Dr. Peter Hufschmied, NAGRA, Baden, Switzerland.

Shedlovsky, T. and L. Shedlovsky (1984), Conductometry, in *Techniques of Chemistry, Vol. 1: Physical Methods of Chemistry, Part IIA: Electrochemical Methods* A. Weissberger, ed., Wiley-Interscience, New York, 1971.

Tsang, C.F., P. Hufschmied and F.V. Hale (1989), "Determination of fracture inflow parameters with a borehole fluid conductivity logging method," *Water Resources Research*, v. 26 (1990), no. 4, pp. 561–578, and LBL-24752, Lawrence Berkeley Laboratory, University of California, and NDC-1, NAGRA, Baden Switzerland.

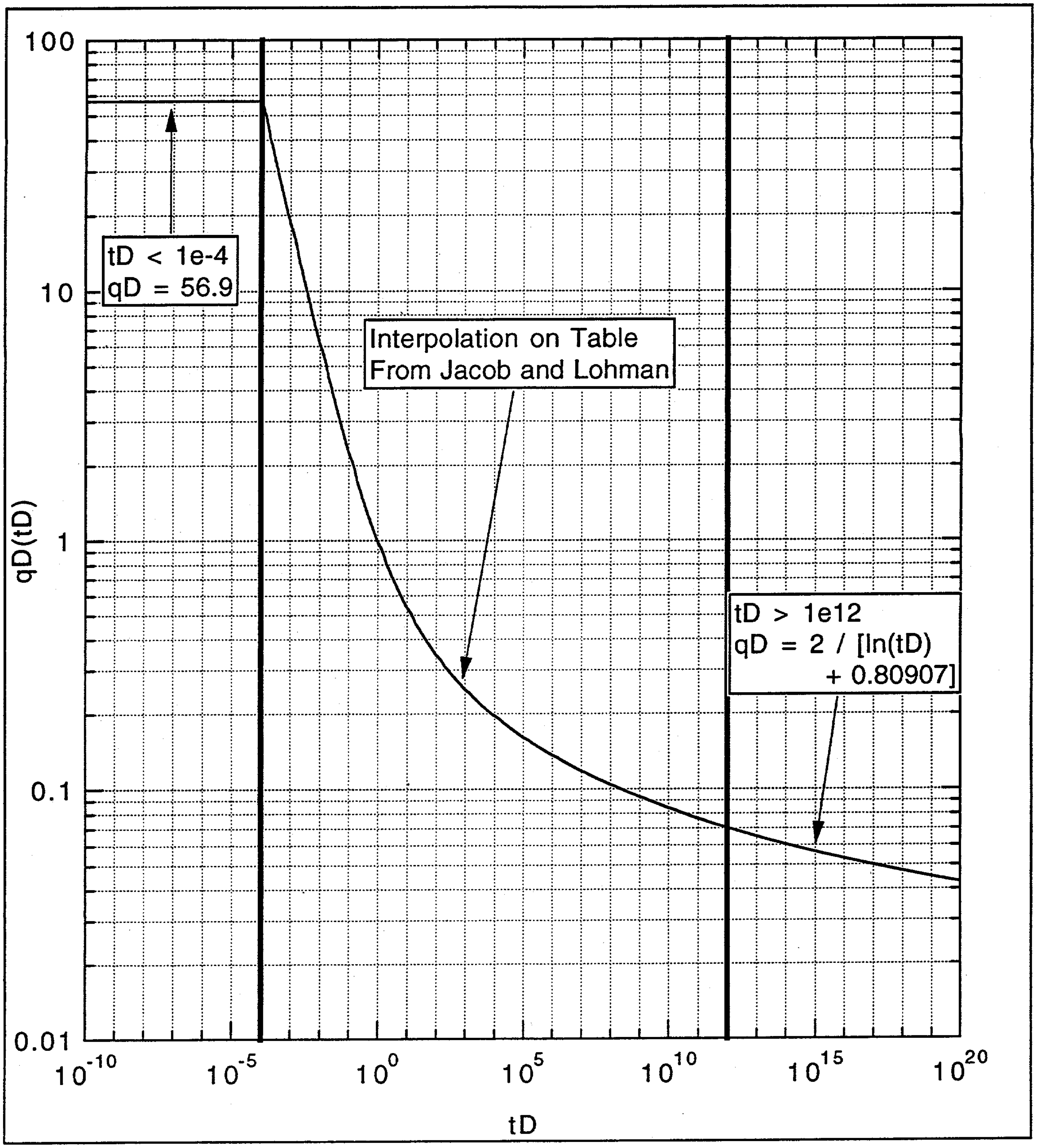

Figure 1. Dimensionless flow rate as a function of dimensionless time, showing the three solution techniques used in different subdomains.

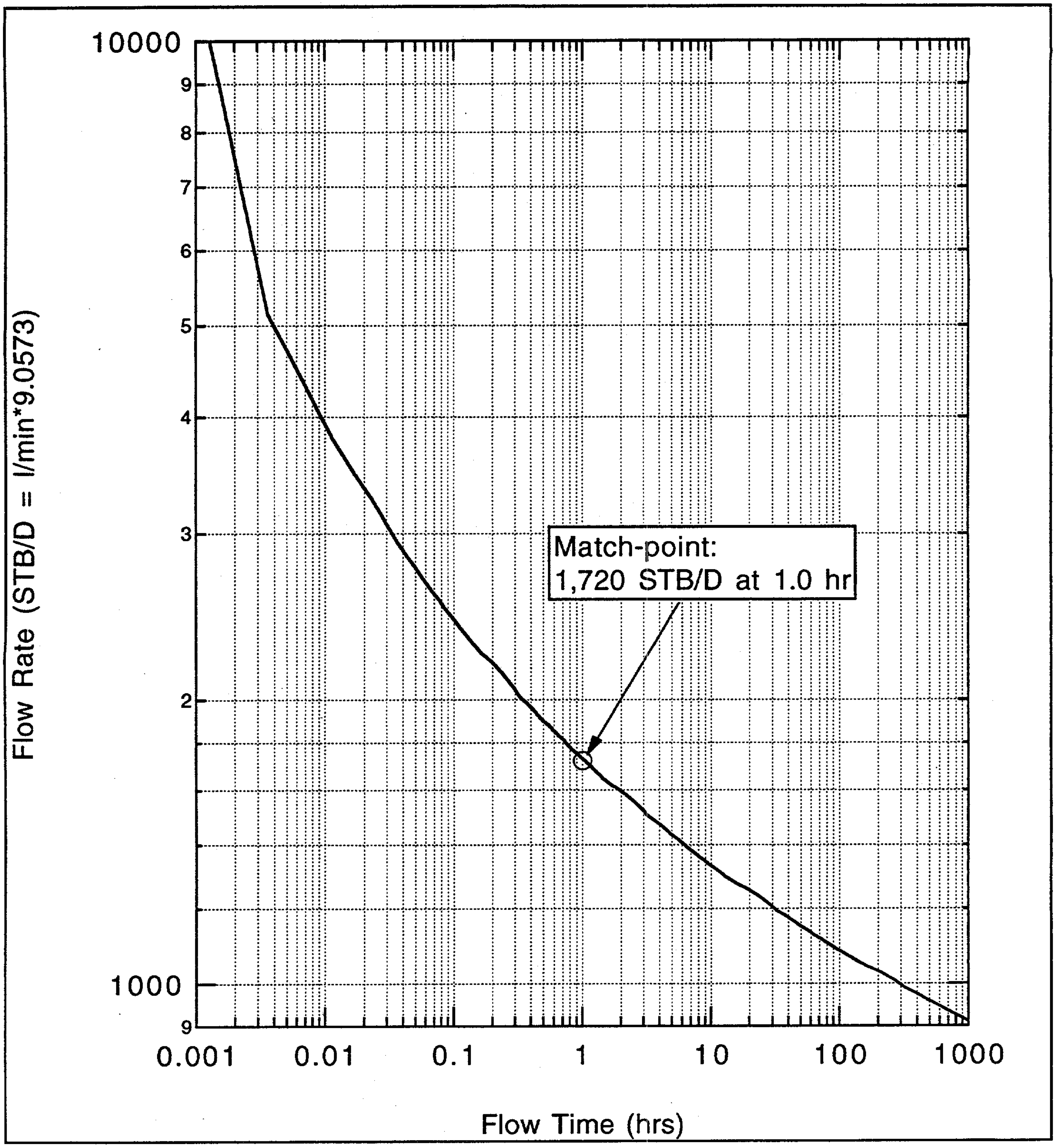

Figure 2. Flow rate as a function of time for T=2.697e-6 m*m/s, S=168.7e-6, drawdown of 703.57 meters. Matches Figure 4.13 in Earlougher (1977) for example of constant-pressure testing in an infinite-acting reservoir (Earlougher's Example 4.4).

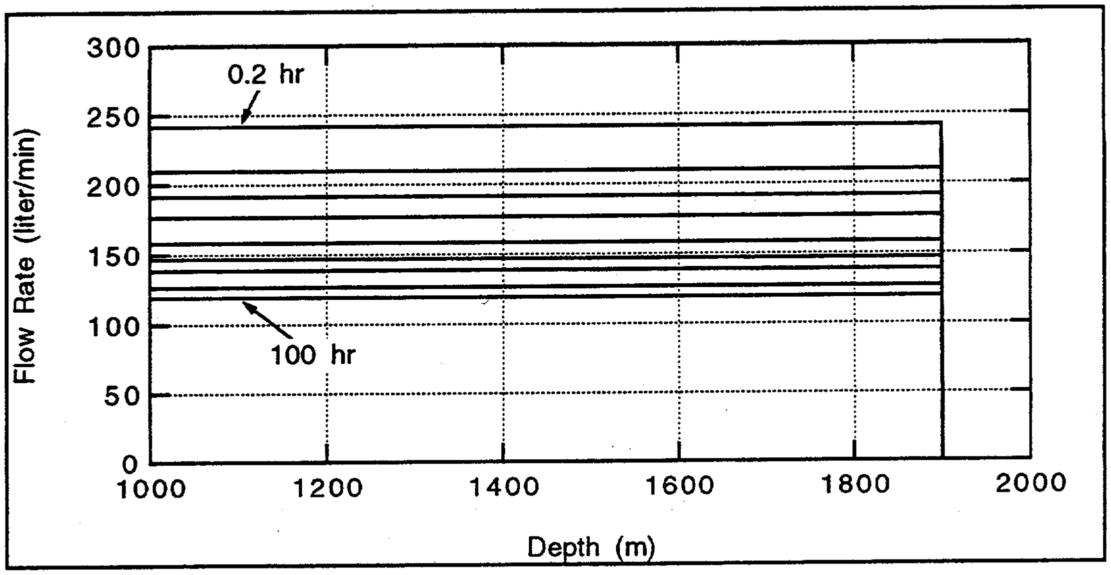

Figure 3. Flow rate profile for 1-cell inflow.

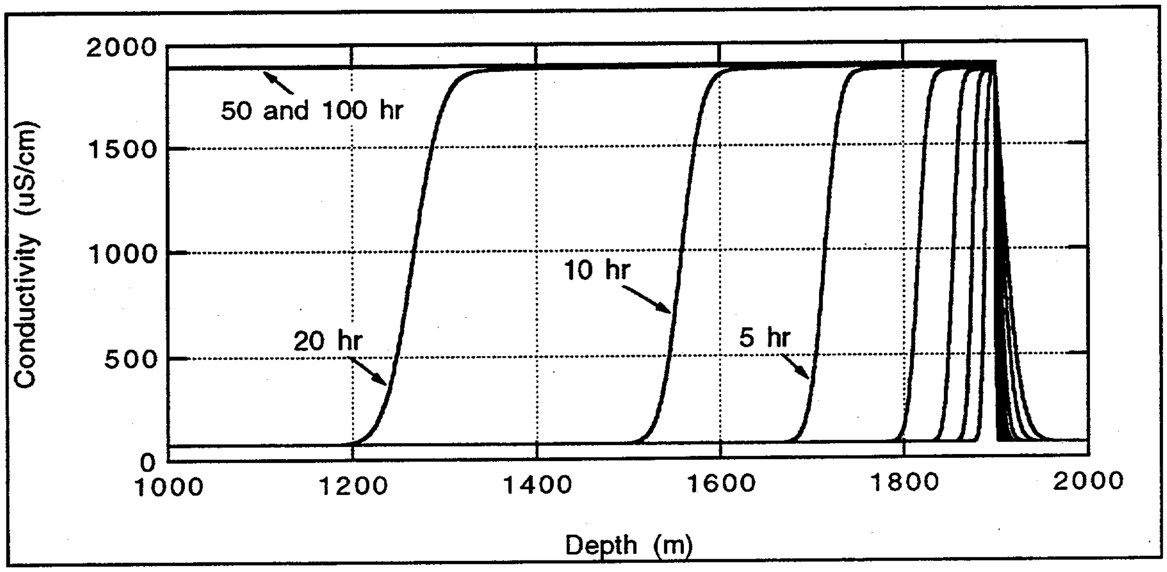

Figure 4. Conductivity profile for 1-cell inflow.

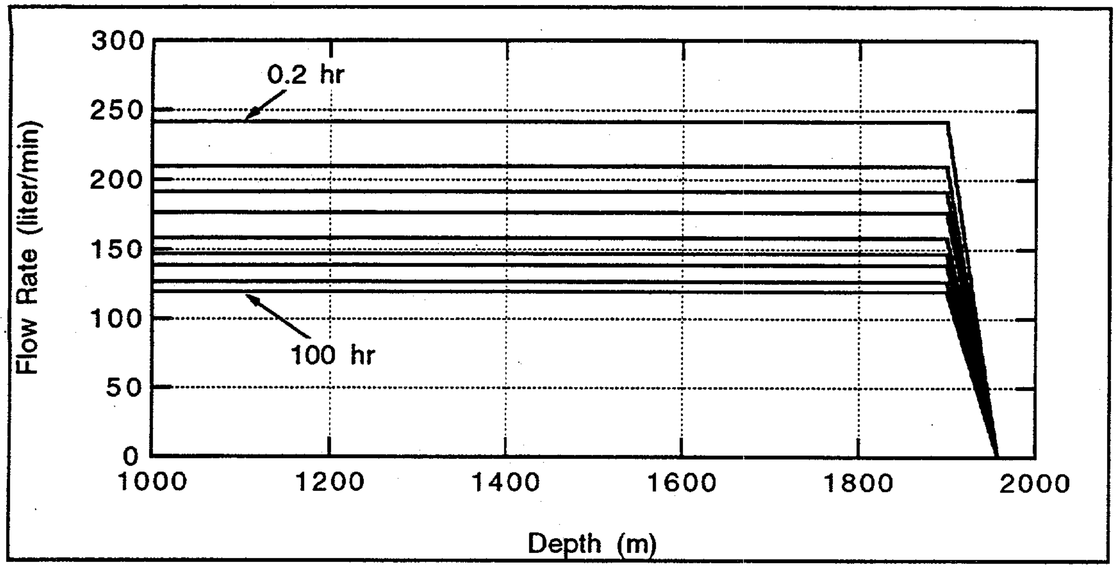

Figure 5. Flow rate profile for 58-cell inflow.

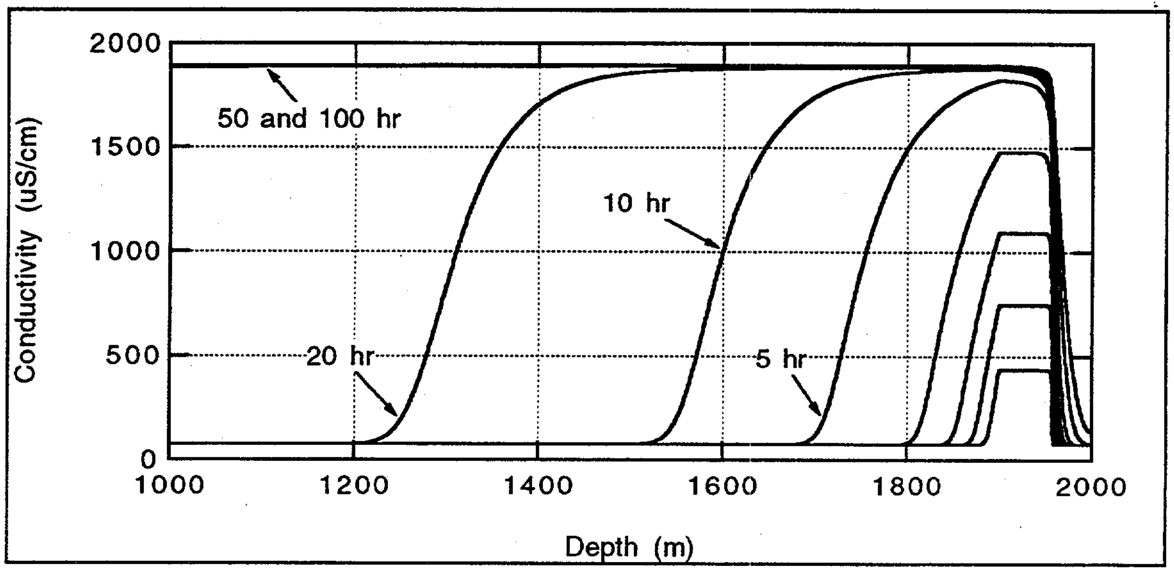

Figure 6. Conductivity profile for 58-cell inflow.

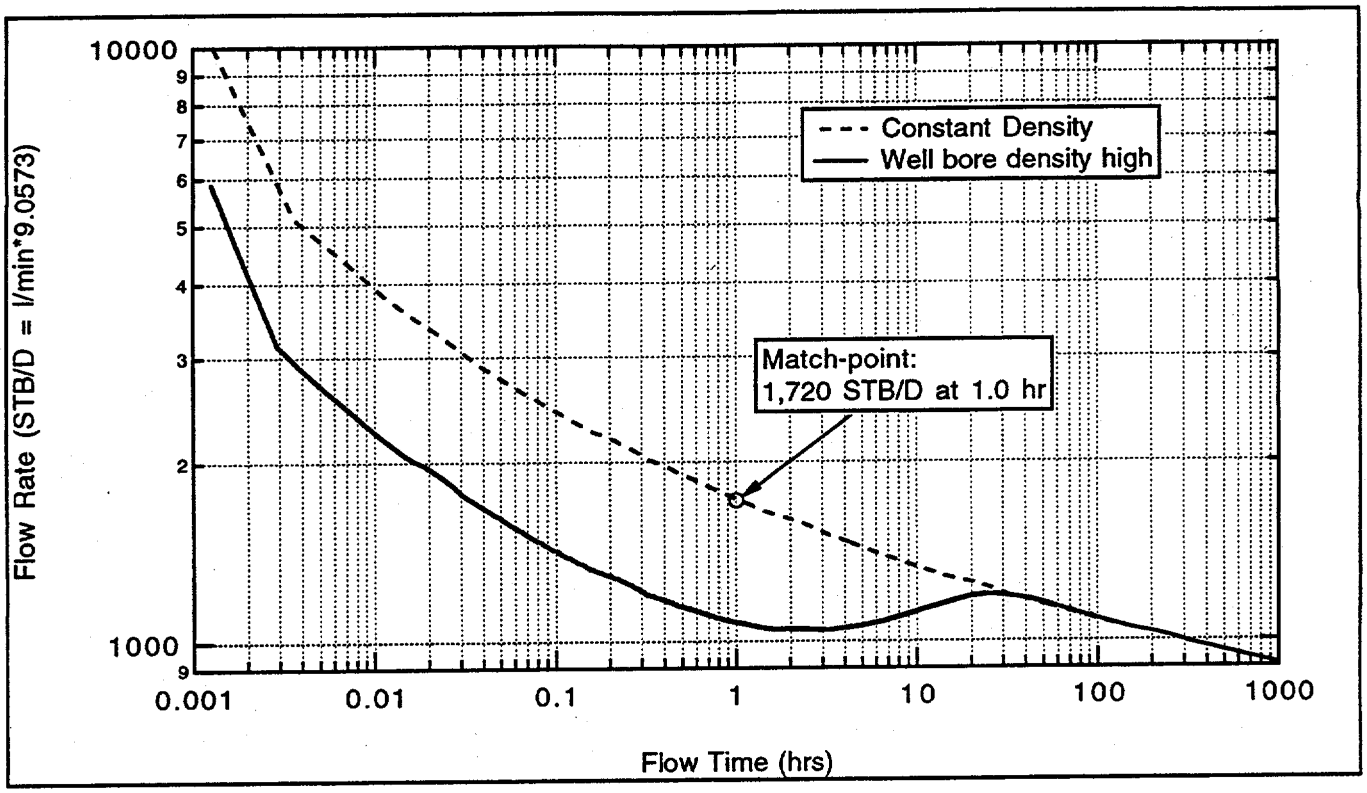

Figure 7. Flow rate as a function of time with initial well bore fluid density doubled. Initially, the flow rate is lower than the constant density case; as the higher density wellbore fluid is flushed out, the flow rate converges with the constant density curve.

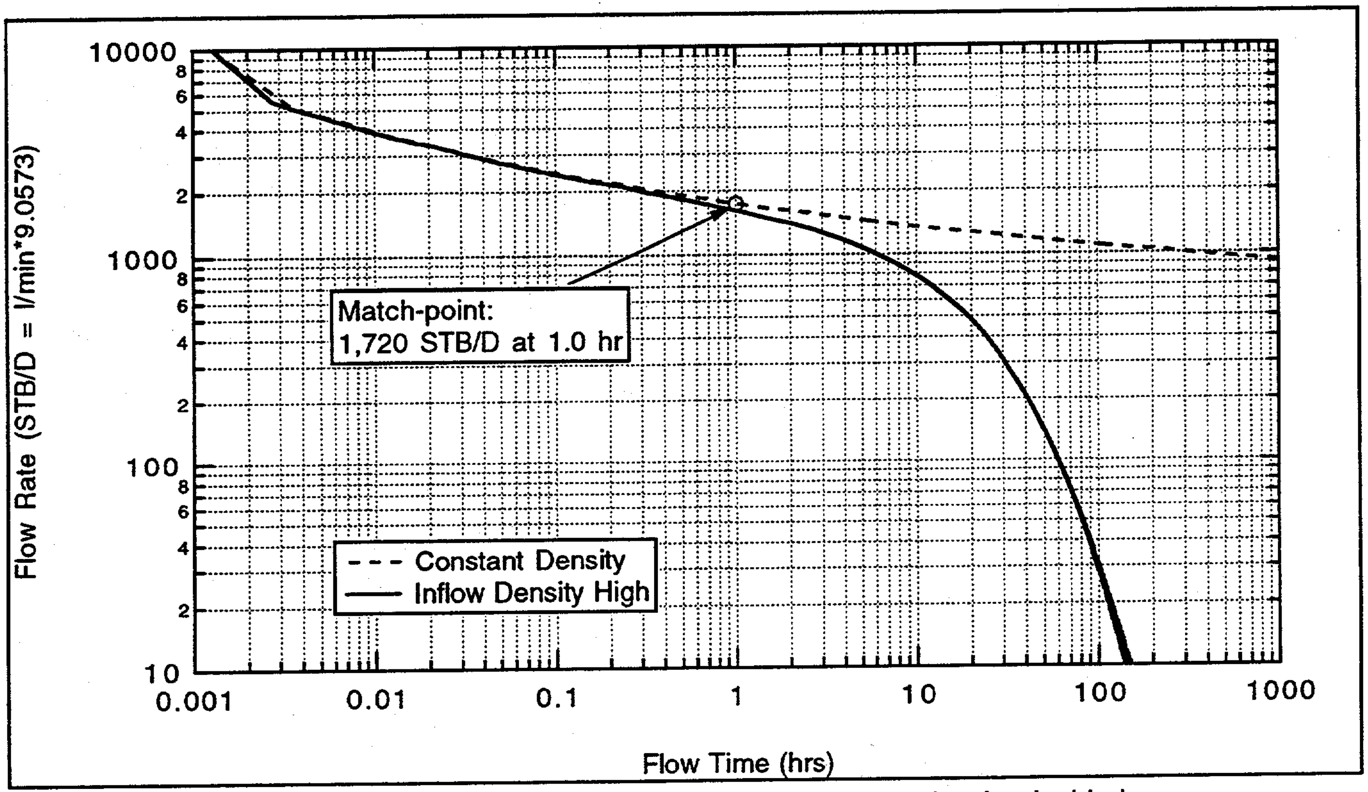

Figure 8. Flow rate as a function of time with inlow density doubled. Initially, the flow rate is the same as the constant density case, but as the higher density fluid flows into the well bore, the driving pressure difference at the inflow depth decreases until the flow stops after a few hundred hours.

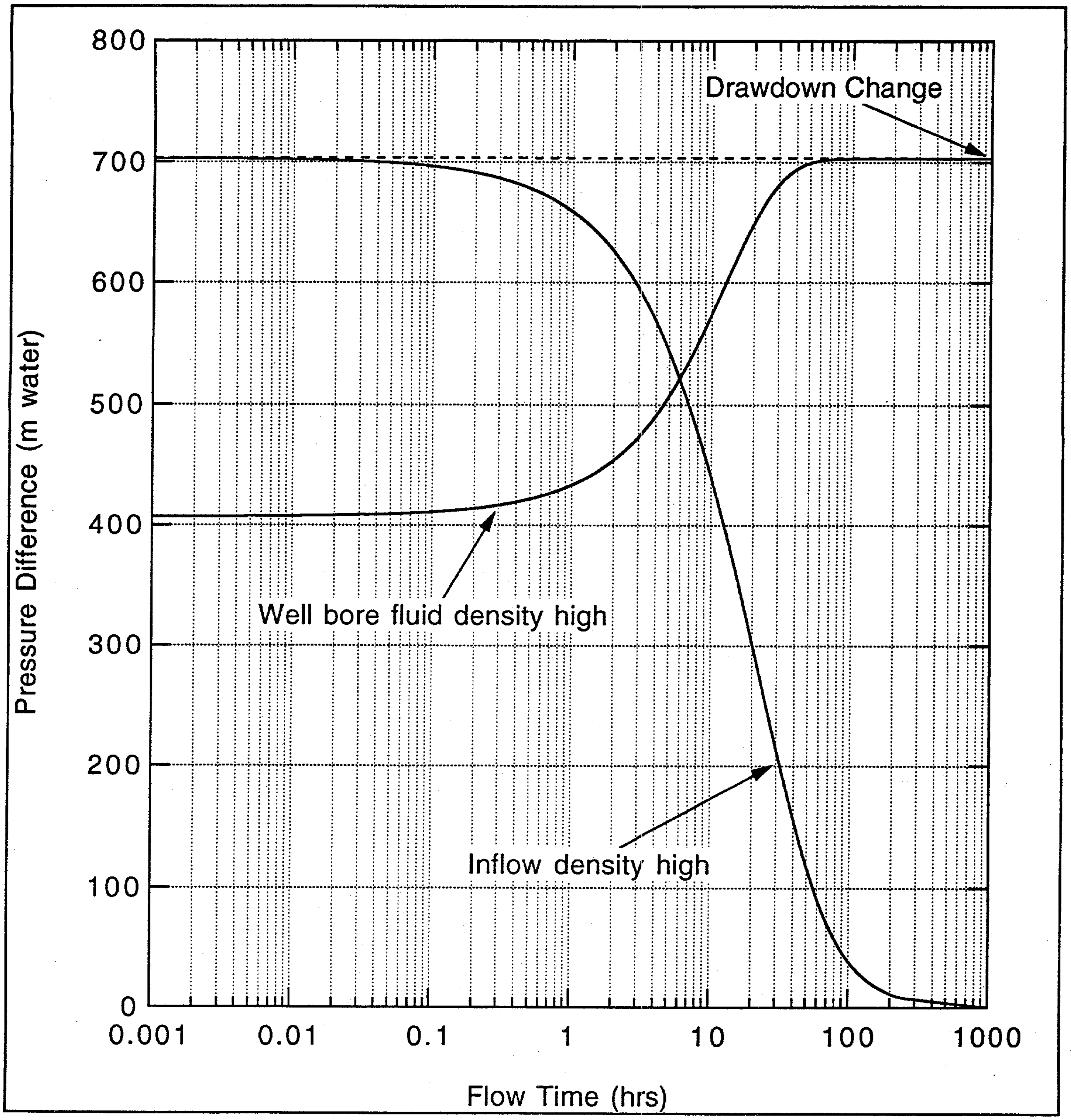

Figure 9. Driving pressure difference (in meters of water) as a function of time. For the constant density calculation, this is simply the drawdown change of 703.6 meters. With higher density initial wellbore fluid, the pressue difference is reduced initially but converges with the constant density solution. With higher inflow density, the difference is initially the same as the constant density case, but decreases to zero as higher density fluid moves into the well bore.

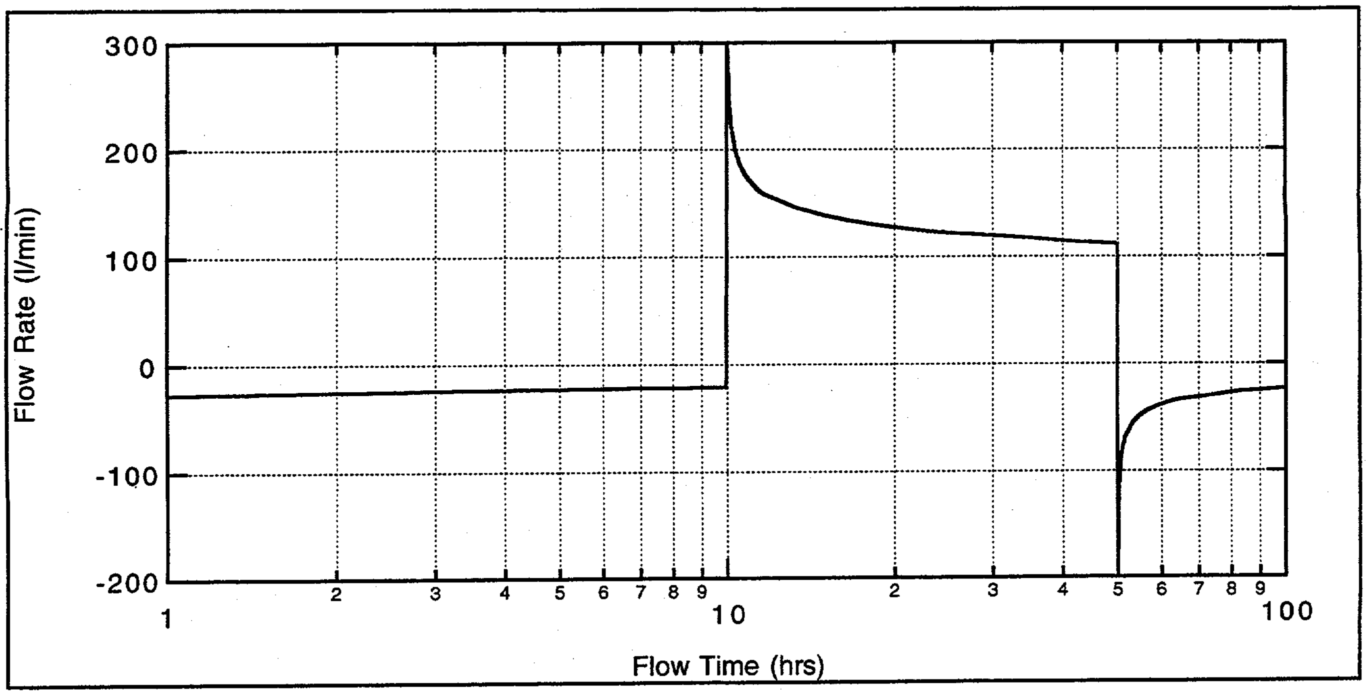

Figure 10. Flow rate as a function of time across the inflow at 1500 m. Notice that the flow is initially negative, from the borehole into the formation. The pump is turned on at 10 hours, and it is turned off at 50 hours.

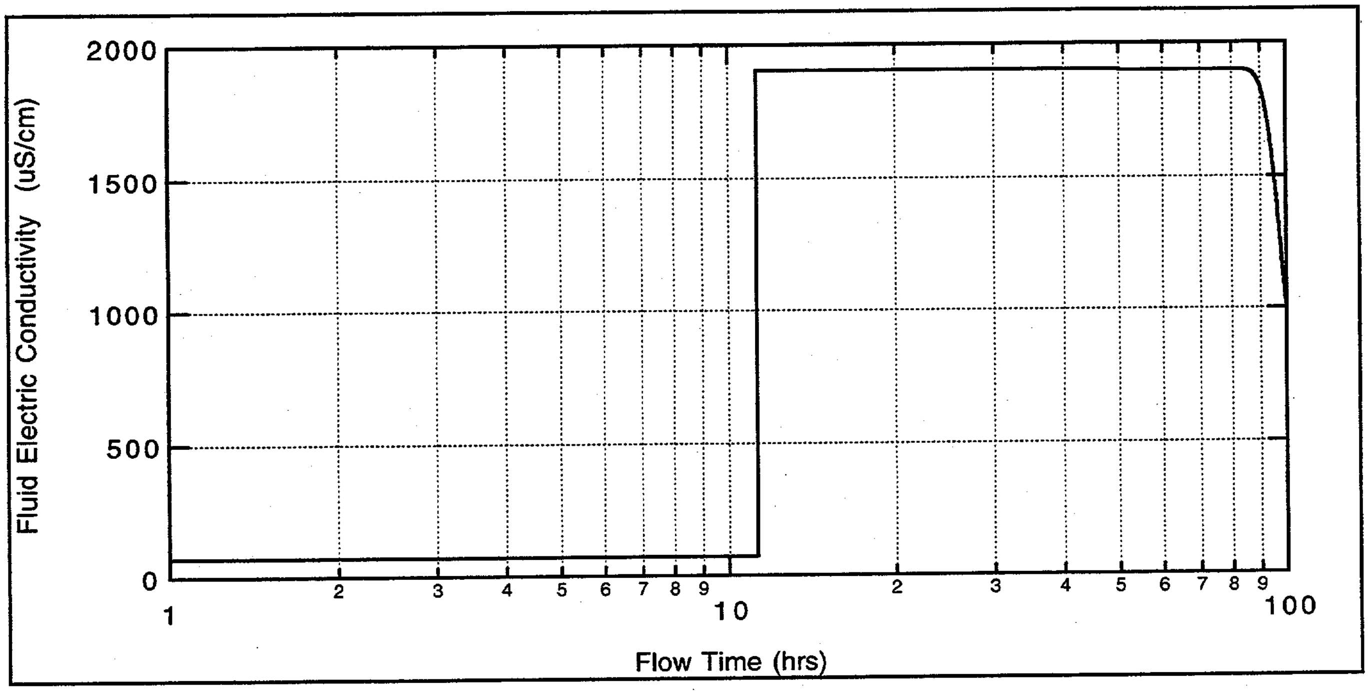

Figure 11. Fluid electric conductivity as a function of time across the inflow at 1500 m. The most striking feature is the delayed rise in conductivity after the pump starts at 10 hours. This is due to pre-production infiltration of low conductivity fluid from borehole to inflow formation. At late times after pumping, as more of the high conductivity fluid moves out of the wellbore, the concentration gradually drops as the front moves toward 1500m.

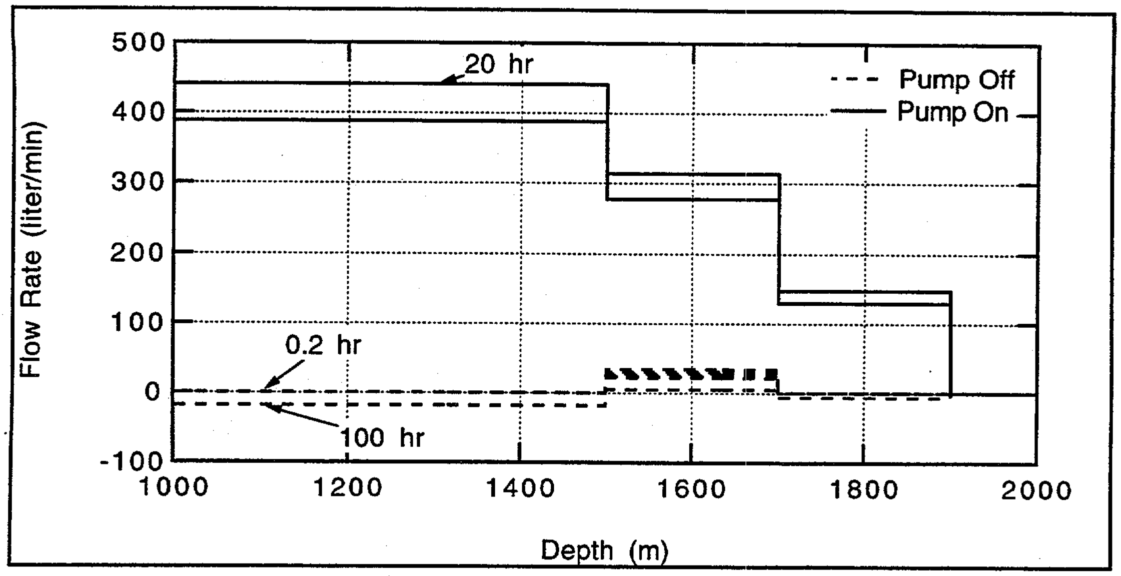

Figure 12. Flow rate profile for multiple inflow, multiple drawdown. Before pumping (e.g., 0.2 hr) there is a circulation from 1700m to 1500m. During the pumping period (e.g., 20 hr), the typical inflow pattern is seen. After pumping (e.g., 100 hr), there is a slight inflow at 1700m, a slight outflow at 1900m, and an outflow at 1500m.

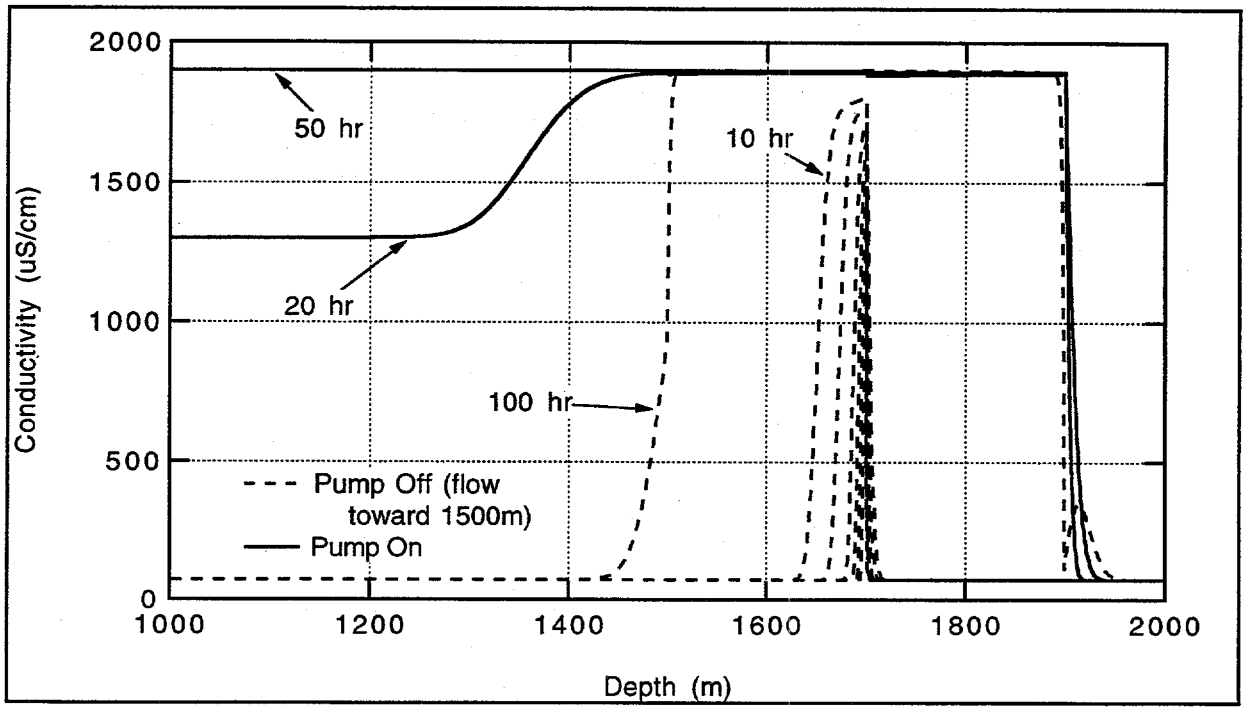

Figure 13. Conductivity profile for multiple inflow, multiple drawdown. Flow before pumping (up to 10 hr) is from 1700m to 1500m. During pumping (10 to 50 hr), inflows are seen at 1900m, 1700m and 1500m. After pumping (after 50 hr), flow is again toward 1500m, both from above and below, pulling the high conductivity fronts back toward 1500m.

```
c***  fracture/borehole transient flow model
c***  june 1994
c***  lawrence berkeley laboratory, earth sciences division
c
      implicit double precision (a-h)
      implicit double precision (o-z)
c
      include 'vpcomms.f'
c
      call inits()
c
c======================================================================
c  BEGINNING OF TIME STEP LOOP
c======================================================================
c
c***   compute flow rates and time step
c
  100 if (ivard.eq.0) then
         call flows
      else
         call dflows
      endif
      call onestep
c
c***   if the time step is to big, try again
c
      if (ifailt.eq.0) goto 200
      t = tp
      write(iout,*)'Reducing time step...'
      goto 100
c
c***   time step size is good
c
  200 call goodstep
      if (t.le.tend) goto 100
c
c======================================================================
c  END OF TIME STEP LOOP
c======================================================================
c
      call endprob
      stop
      end
c
c***   subroutine flows
c***   calculate flows through each segment
c***   this subroutine is called at the beginning of each time step
c
      subroutine flows
c
      implicit double precision (a-h)
      implicit double precision (o-z)
c
      include 'vpcomms.f'
c
c***   maximum time step is minimum time to flush one-tenth
c***   of a segment.  note that this time may still be too large
c***   if dispersion is significant
c
      if (ifailt.eq.0) then
         if (qtmax.ne.0.) then
            delt = 0.5 * vol/qtmax
            if (delt.gt.tsmax) delt = tsmax
         else
            delt = 0.01 * tsmax
         endif
      endif
c
c***   initialize segment flows
c
      do i=1,ilim
         qfrac(i) = 0.
      enddo
c
c***   compute flow into segments from fractures
c***   it is possible to have more than one fracture in a single
c***   segment
c
      do i=1,iflim
c
c***   compute flow rate as a function of pressure drop
c
         qin(i) = 0.
c
c***   calculate effect of each pressure (drawdown) change
c
         do idraw=1,idhlim
            if (t.ge.delht(idraw)) then
               qdtid = avgint(t,delht(idraw),tcoef(i),delt)
               qin(i) = qin(i) +
                        (qdtid * qcoef(i) * delhh(idraw))
            endif
         enddo
c
c***   calculate effect due to initial conditions
c
         qinp   = hfrac(i)-(xs(ifseg(i))-watlev)
         if (qinp.ne.0. .and. t.ge.tstart) then
            qdtid  = avgint(t,tstart,tcoef(i),delt)
            qin(i) = qin(i) + qdtid * qcoef(i) *  qinp
         endif
         qfrac(ifseg(i)) = qfrac(ifseg(i)) + qin(i)
      enddo
      call flowsa
      return
      end
c
c***   try out a time step
c
      subroutine onestep()
c
      implicit double precision (a-h)
      implicit double precision (o-z)
c
      include 'vpcomms.f'
c
      ifailt = 0
      tp     = t
      t      = t + delt
c
c***   save current mass and volume totals
c
      gtmins = gtmina
      gtdins = gtdina
      gtmos  = gtmout
      gtdos  = gtdout
      gtfbms = gtfbm
      gtbfms = gtbfm
      gtvins = gtvina
      gtvos  = gtvout
      gtfbvs = gtfbv
      gtbfvs = gtbfv
      topms  = topm
      topds  = topd
c
c***   compute fracture inflows and concentrations to the borehole
c
      call ctfrac
c
c***   compute mass transfer for the time step
c
      call tstep
      return
      end
c
c***   complete a time step
c
      subroutine goodstep()
c
      implicit double precision (a-h)
      implicit double precision (o-z)
c
      include 'vpcomms.f'
c
      do j=1,ilpt
         if (t.ge.pt(j)) then
            call cprt
            pt(j)=2.*tend
         endif
      enddo
c
      if (delt.le.0.) then
         write(iout,*)'ERROR--time step <= 0!'
         stop
      endif
c
      tsavg  = tsavg  + delt
      ntstep = ntstep + 1
      if (delt.lt.tslo) tslo = delt
      if (delt.gt.tshi) tshi = delt
c
c***   write extra output for report plots
c
      cf1 = cfrac(ifseg(1),1)
      write(80,'(3f12.5)')((t-delt/2.)/3600),
     .     (qin(1)*60000.),
     .     (ofset + cf1*(coefa + cf1*coefb))
      return
      end
c
c***   initialize constants, etc.
c
      subroutine inits()
c
      implicit double precision (a-h)
      implicit double precision (o-z)
c
      include 'vpcomms.f'
c
c***   input and initialization
c
      do i=1,imax
```

```
         cfrac(i,1)=0.
      enddo
      gtbfm  = 0.
      gtbfv  = 0.
      gtdina = 0.
      gtdout = 0.
      gtfbm  = 0.
      gtfbv  = 0.
      gtmina = 0.
      gtmout = 0.
      gtvina = 0.
      gtvout = 0.
      idout  = 9
      iin    = 5
      iout   = 6
      do i=1,ifmax
         iplim(i) = 0
      enddo
      iprt   = 7
      iqout  = 8
      itog   = 1
      ivard  = 0
      pcmax  = 0.
      pi     = 3.141592654
      qtmax  = 0.
c
      open (unit=iin, file='VHB.DAT',status='OLD',form='FORMATTED')
      open (unit=iout,file='VHB.MSG',status='NEW',form='FORMATTED')
      open (unit=iprt,file='VHB.OUT',status='NEW',form='FORMATTED')
      write(iout,*)'Files opened...'
c
c
c***   read borehole parameters
c***   note: set topd <= 0 for constant density
c***   xtop is top of analysis region (m)
c***        xtop may be adjusted during discretization
c***   xbot is bottom of analysis region (m)
c***     note: x increases downward, zero at surface
c***   delx is segment length (m)
c***   diam is borehole diameter (input in cm, coverted to m)
c***   area is cross-sectional area of hole (m**2)
c***   vol is segment volume (m**3)
c***   ilim is number of segment over analysis region
c
c***   input stream--one free format record
c***       xtop, xbot, delx, diam
c
      read(iin,*)xtop,xbot,delx,diam,watlev,topd
c
      if (topd.gt.0.) then
         ivard = 1
         write(iout,*)'Variable density model'
      else
         topd = 1.0
         write(iout,*)'Constant density model'
      endif
c
c***   convert diameter in cm to m
c
      diam = diam/100.
c
c***   check geometry
c
      if (xbot.lt.xtop) then
```

```
         write(iout,940)
         stop
      endif
      xdist = xbot-xtop
      xint  = xdist/delx
      if (xint.lt.1) then
         write(iout,950)
         stop
      endif
      ilim = xint
      if (ilim.gt.imax) then
         write(iout,990)imax
         stop
      endif
      xtopi  = xtop
      xtop   = xbot-(ilim*delx)
      deltop = abs(xtop-xtopi)
c
c***   if delta-x does not fit over the interval within 1% of
c***   a delta-x, then make a note
c
      if (deltop.gt.(0.01*delx)) then
         write(iout,960)ilim,delx
         write(iout,970)deltop,xtop
      else
         write(iout,960)ilim,delx
      endif
c
c***   calculate cross-sectional area and volume of a segment
c
      rad  = diam/2.
      area = pi*rad*rad
      vol  = area*delx
      write(iout,980)diam,area,vol
c
c***   compute segment midpoints
c***   note--sign reversed for graphing
c
      do i=1,ilim
         xs(i) = xbot + (0.5-i)*delx
      enddo
c
c***   read flow from fractures
c***   input stream -- free format
c***      number of fractures
c***      one record for each fracture --
c***         position (m), flow rate (m**3/sec), concentration of
c***         solute (kg/m**3), and solute flow start time (h)
c
c***   read fracture data
c
      read(iin,*)iflim
      if (iflim.gt.ifmax) then
         write(iout,910)ifmax
         stop
      endif
      j=iflim
      do i=1,j
         if (ivard.eq.0) then
            read(iin,*)xin(i),cin(i,1),tfrac(i),sfrac(i),hfrac(i),
     .        vfrac(i,1),cfz(i,1,1)
            cin(i,2)   = 1.
            cfz(i,1,2) = 1.
         else
            read(iin,*)xin(i),cin(i,1),tfrac(i),sfrac(i),hfrac(i),
     .        vfrac(i,1),cfz(i,1,1),cin(i,2),cfz(i,1,2)
         endif
         if (vfrac(i,1).gt.0.) then
            ifznum(i) = 1
         else
            ifznum(i) = 0
         endif
c
c***   locate fracture on segments
c
         iskipit = 0
         if ((xin(i).gt.xbot) .or. (xin(i).lt.xtop)) then
            write(iout,930)xin(i)
            iflim   = iflim-1
            iskipit = 1
         endif
         if (iskipit.eq.0) then
            ifseg(i) = iconvx(xtop,xbot,delx,xin(i))
            rad = diam/2
            tcoef(i) = tfrac(i) / (sfrac(i)*rad*rad)
            qcoef(i) = 2. * pi * tfrac(i)
         endif
      enddo
c
c***   read drawdowns and times
c***   input stream -- free format
c***      number of changes
c***      one record for each change--
c***         time (hrs), drawdown (m)
c
c***   read drawdown data
c
      read(iin,*)idhlim
      if (idhlim.gt.idhmax) then
         write(iout,1910)idhmax
         stop
      endif
c
      j=idhlim
c
      do i=1,j
         read(iin,*)delht(i),delhh(i)
         delht(i) = delht(i) * 3600.
      enddo
c
c***   read time parameters
c***   input stream--
c***      start time (hrs), end time (hrs), number of prints
c***      then, print times (hrs)
c***    note: all times are converted to seconds
c
      read(iin,*)tstart,tend,tsmax,ilpt
      tstart = tstart * 3600.
      tend   = tend   * 3600.
      tsmax  = tsmax  *   60.
c
      if (tend.lt.tstart) then
         write(iout,2910)
         stop
      endif
      if (ilpt.gt.impt) then
         write(iout,915)impt
      endif
```

```
c
c***   read in print-time parameters
c
      read(iin,*)(pt(i),i=1,ilpt)
      do i=1,ilpt
         pt(i)=pt(i)*3600.
      enddo
c
c***   read dispersion parameter
c***   input stream--(one record)
c***      type code, parameter
c***   at the end of this routine, dks has units m**3/sec
c
      read(iin,*)itypdk,dk
c
c***   if itypdk=1, value is constant over all segments
c***   if itypdk=2, value is for mean velocity
c***   if itypds=3, value is for mean velocity squared
c
      dksi = dk * area/delx
      do i=1,ilim
         dks(i) = dksi
      enddo
c
c***   read initial concentrations and densities
c***   input records:
c***      ic0n n=number of initial values specified
c***      x, c0  pairs of x values and initial concentrations
c
      do i=1,ilim
         cxt(i,1,1)=0.
         cxt(i,1,2)=1.
         cxt(i,2,2)=1.
      enddo
c
      read(iin,*)ic0n
      if (ic0n .gt. 0) then
         do i=1,ic0n
            if (ivard.eq.0) then
               read(iin,*)x,c0
               d0 = 1.
            else
               read(iin,*)x,c0,d0
            endif
            if ((x.gt.xbot) .or. (x.lt.xtop)) then
               write(iout,900)x
            else
               iseg=iconvx(xtop,xbot,delx,x)
               if (cxt(iseg,1,1).eq.0.) then
                  cxt(iseg,1,1)=c0
                  cxt(iseg,1,2)=d0
                  cxt(iseg,2,2)=d0
               else
                  write(iout,3910)x
               endif
            endif
         enddo
      endif
c
c***   ofset is the background conductivity in S/m
c***   coefa is the linear term for converting concentrations
c***      in kg/m**3 to conductivity in S/m (at 20 C)
c***   coefb is the second degree term for converting
c***      concentrations to conductivity (at 20 C)
```

```
c
      read(iin,*)ofset,coefa,coefb
c
c***   close input file after reading data
c
      close (unit=iin)
c
c***   compute the starting mass and volume in the system
c
      write(iout,*)'Input processed...'
      call smass
      tmf0    = smf
      tdf0    = sdf
      tm0     = smb
      td0     = sdb
      tvf0    = svf
      tv0     = svb
      watlev0 = watlev
      topv0   = (xtop-watlev)*area
      topm0   = cxt(ilim,1,1) * topv
      topd0   = topd
      tshi     = 0.
      tslo     = 1.e30
      tsavg    = 0.
      ntstep   = 0
      t        = tstart
      delt     = 0.001*tsmax
      topv     = topv0
c
      return
c
  900 format(/' *** NOTE *** initial concentration position',
     .       ' invalid--ignored'
     .       /'                 position is ',f15.5)
  910 format(' Maximum number of fractures is ',i4,' -- aborting')
  915 format(/' Maximum number of print times is ',i4,' -- aborting')
  930 format(' *** NOTE *** fracture position outside of region ',f10.4,
     .       ' -- ignored')
  940 format(' Top of region is below bottom of region--aborting')
  950 format(' delta-x is greater than region size--aborting')
  960 format(' Region has been divided into ',i4,' segments with',
     .       ' length ',f10.4,' (m)')
  970 format(' *** NOTE *** Top of region has been moved by ',
     .       f10.4,' (m) to ',f10.4,' (m)')
  980 format(' Each section has diameter        ',f12.5,' (m)'/
     .       '              cross-sectional area ',f12.5,' (m**2)'/
     .       '          and volume              ',f12.5,' (m**3)')
  990 format(' Maximum number of segments is ',i4,' -- aborting')
 1910 format(' Maximum number of drawdowns is ',i4,' -- aborting')
 2910 format(/' End time before start time -- aborting')
 3910 format(/' *** NOTE *** multiple initial concentrations',
     .       ' for one segment'
     .       /' second value ignored--position is ',f15.5)
      end
c
c***   function iconvx
c***   convert x position to segment index
c***   segment 1 begins at xbot
c***   segment ilim ends at xtop
c***   this function is used outisde of time steps
c
      function iconvx(xtop,xbot,delx,x)
c
      implicit double precision (a-h)
      implicit double precision (o-z)
c
      idxup = (xbot-x) / delx+1.
      if (x.eq.xtop) idxup = idxup-1
      iconvx = idxup
      return
      end
c
c***   compute the mass and volume in the system
c
      subroutine smass()
c
      implicit double precision (a-h)
      implicit double precision (o-z)
c
      include 'vpcomms.f'
c
      smb = 0.
      sdb = 0.
c
c***   sum mass in borehole
c
      do i=1,ilim
         smb = smb + cxt(i,itog,1)
         sdb = sdb + cxt(i,itog,2)
      enddo
      smb = smb  * vol
      sdb = sdb  * vol
c
c***   compute volume in borehole
c
      svb = ilim * vol
c
c***   sum mass and volume in fractures
c
      smf = 0.
      sdf = 0.
      svf = 0.
      do i=1,iflim
         if (ifznum(i).gt.0) then
            do j=1,ifznum(i)
               vij = vfrac(i,j)
               smf = smf + cfz(i,j,1)*vij
               sdf = sdf + cfz(i,j,2)*vij
               svf = svf + vij
            enddo
         endif
      enddo
c
      return
      end
c
c***   subroutine cprt
c***   print out concentration, flow rate and density arrays
c***   this subroutine is executed whenever concentration arrays
c***   are requested during the time steps, or at the end of the
c***   problem
c***   note:  units of output conductivity is S/m
c
      subroutine cprt
c
      implicit double precision (a-h)
      implicit double precision (o-z)
c
      include 'vpcomms.f'
c
c***   output time in hours and concentrations with ascending
c***   values (note negative increment in loop)
c
c***   output time in hours (convert from seconds)
c
      th = t / 3600.
      write(iprt ,'(f6.2)')th
      write(iqout,'(f6.2)')th
      write(idout,'(f6.2)')th
c
c***   convert concentrations in kg/m**3 to electric conductivity
c***   in uS/cm using input coefficients
c
      do i=1,ilim
         ci       = cxt(i,itog,1)
         sigma(i) = ofset + ci*(coefa + ci*coefb)
         qlm(i)   = 60000.*qnxt(i)
      enddo
c
      write(iout ,920)th
      write(iprt, '(2f15.5)')(xs(i),sigma(i),i=ilim,1,-1)
      write(idout,'(2f15.5)')(xs(i),cxt(i,itog,2),i=ilim,1,-1)
      write(iqout,'(2f15.5)')(xs(i),qlm  (i),i=ilim,1,-1)
      return
c
  920 format(/' Printing conductivity at time ',e15.5,' (hrs)')
      end
c
c***   end of problem
c
      subroutine endprob()
c
      implicit double precision (a-h)
      implicit double precision (o-z)
c
      include 'vpcomms.f'
c
c***   check for final print
c
      needp = 0
      do j=1,ilpt
         if (pt(j).lt.(2.*tend)) needp=1
      enddo
      if (needp.ne.0) call cprt
c
c***   end of problem, check mass conservation
c
      call smass
c
c***   total system mass
c
      emass  = smf + smb + topm
      edens  = sdf + sdb + (topd*topv)
      exmass = (tm0 + tmf0 + topm0 + gtmina) - gtmout
      exdens = (td0 + tdf0 + (topd0*topv0) + gtdina) - gtdout
      perr   = 0.
      derr   = 0.
      if (exmass.ne.0.) perr = ((emass-exmass) / exmass) * 100.
      if (exdens.ne.0.) derr = ((edens-exdens) / exdens) * 100.
c
c***   fracture mass
c
```

```
      exfm  = (tmf0 + gtbfm) - gtfbm
      exfd  = (tdf0 + gtbfd) - gtfbd
      pferr = 0.
      dferr = 0.
      if (exfm.ne.0.) pferr = ((smf-exfm) / exfm) * 100.
      if (exfd.ne.0.) dferr = ((sdf-exfd) / exfd) * 100.
c
c***   total system volume (convert from m**3 to l)
c
      if (topv0.gt.topv) then
         gtvout = gtvout + (topv0-topv)
      else
         if (topv0.lt.topv) gtvina = gtvina + (topv - topv0)
      endif
      svf    = svf    * 1000.
      svb    = svb    * 1000.
      topv   = topv   * 1000.
      tv0    = tv0    * 1000.
      tvf0   = tvf0   * 1000.
      topv0  = topv0  * 1000.
      gtvina = gtvina * 1000.
      gtvout = gtvout * 1000.
      gtbfv  = gtbfv  * 1000.
      gtfbv  = gtfbv  * 1000.
c
      evol  = svf + svb + topv
      exvol = (tv0 + tvf0 + topv0 + gtvina) - gtvout
      pverr = 0.
      if (exvol.ne.0.) pverr = ((evol-exvol) / exvol) * 100.
c
c***   fracture volume
c
      exfv   = (tvf0 + gtbfv) - gtfbv
      pfverr = 0.
      if (exfv.ne.0.) pfverr = ((svf-exfv) / exfv) * 100.
c
      write(iout,*)' '
      write(iout,*)'ELECTROLYTE MASS BALANCE:'
      write(iout,*)'Starting mass in system (kg): ',
     . (tm0+tmf0+topm0)
      write(iout,*)' (Borehole    ',tm0,')'
      write(iout,*)' (Fractures   ',tmf0,')'
      write(iout,*)' (Top section ',topm0,')'
      write(iout,*)'Mass added to system (kg):    ',gtmina
      write(iout,*)'Mass out of system (kg):      ',gtmout
      write(iout,*)'                              -------------------'
      write(iout,*)'Expected ending mass (kg):    ',exmass
      write(iout,*)' '
      write(iout,*)'Actual ending mass (kg):      ',emass
      write(iout,*)' (Borehole    (kg):',smb,')'
      write(iout,*)' (Fractures   (kg):',smf,')'
      write(iout,*)' (Top section (kg):',topm,')'
      write(iout,*)' '
      write(iout,*)'Percent error:                ',perr
c
      write(iout,*)' '
      write(iout,*)'FLUID MASS BALANCE (DENSITY BASED):'
      write(iout,*)'Starting mass in system (kg):  ',
     . (td0+tdf0+(topd0*topv0/1000.))
      write(iout,*)' (Borehole    ',td0,')'
      write(iout,*)' (Fractures   ',tdf0,')'
      write(iout,*)' (Top section ',(topd0*topv0/1000.),')'
      write(iout,*)'Mass added to system (kg):    ',gtdina
      write(iout,*)'Mass out of system (kg):      ',gtdout
      write(iout,*)'                              -------------------'
      write(iout,*)'Expected ending mass (kg):    ',exdens
      write(iout,*)' '
      write(iout,*)'Actual ending mass (kg):      ',edens
      write(iout,*)' (Borehole    (kg):',sdb,')'
      write(iout,*)' (Fractures   (kg):',sdf,')'
      write(iout,*)' (Top section (kg):',(topd*topv/1000.),')'
      write(iout,*)' '
      write(iout,*)'Percent error:                ',derr
      write(iout,*)' '
      write(iout,*)'VOLUME BALANCE:'
      write(iout,*)'Starting vol in system (l):   ',(tv0+tvf0+topv0)
      write(iout,*)' (Borehole   :',tv0,')'
      write(iout,*)' (Fractures  :',tvf0,')'
      write(iout,*)' (Top section:',topv0,')'
      write(iout,*)'Vol added to system (l):      ',gtvina
      write(iout,*)'Vol out of system (l):        ',gtvout
      write(iout,*)'                              -------------------'
      write(iout,*)'Expected ending vol (l):      ',exvol
      write(iout,*)' '
      write(iout,*)'Actual ending vol (l):        ',evol
      write(iout,*)' (Borehole    (l):',svb,')'
      write(iout,*)' (Fractures   (l):',svf,')'
      write(iout,*)' (Top section (l):',topv,')'
      write(iout,*)' '
      write(iout,*)'Percent error:                ',pverr
      write(iout,*)' '
      write(iout,*)' '
      write(iout,*)'FRACTURE ZONES ELECTROLYTE MASS BALANCE:'
      write(iout,*)'Starting mass in frac (kg):  ',tmf0
      write(iout,*)'Mass to frac from bore (kg): ',gtbfm
      write(iout,*)'Mass to bore from frac (kg): ',gtfbm
      write(iout,*)'                              -------------------'
      write(iout,*)'Expected ending mass (kg):   ',exfm
      write(iout,*)
      write(iout,*)'Actual ending mass (kg):     ',smf
      write(iout,*)' '
      write(iout,*)'Percent error:               ',pferr
      write(iout,*)' '
      write(iout,*)'FRACTURE ZONES FLUID MASS BALANCE:'
      write(iout,*)'Starting mass in frac (kg):  ',tdf0
      write(iout,*)'Mass to frac from bore (kg): ',gtbfd
      write(iout,*)'Mass to bore from frac (kg): ',gtfbd
      write(iout,*)'                              -------------------'
      write(iout,*)'Expected ending mass (kg):   ',exfd
      write(iout,*)
      write(iout,*)'Actual ending mass (kg):     ',sdf
      write(iout,*)' '
      write(iout,*)'Percent error:               ',dferr
      write(iout,*)' '
      write(iout,*)'FRACTURE ZONES VOLUME BALANCE:'
      write(iout,*)'Starting vol in fracs (l):   ',tvf0
      write(iout,*)'Vol to frac from bore (l):   ',gtbfv
      write(iout,*)'Vol to bore from frac (l):   ',gtfbv
      write(iout,*)'                             -------------------'
      write(iout,*)'Expected ending vol (l):     ',exfv
      write(iout,*)' '
      write(iout,*)'Actual ending vol (l):       ',svf
      write(iout,*)' '
      write(iout,*)'Percent error:               ',pfverr
      write(iout,*)' '
      write(iout,*)' '
      write(iout,*)'Number of time step used:    ',ntstep
      write(iout,*)'Minimum time step used (min):',(tslo/60.)
      write(iout,*)'Average time step used (min):',((tsavg/60.)/ntstep)
      write(iout,*)'Maximum time step used (min):',(tshi/60.)
      write(iout,*)' '
      write(iout,*)'Maximum percentage cell mass transfer: ',pcmax
      write(iout,*)' '
      write(iout,*)' '
      write(iout,*)'*** PROBLEM FINISHED ***'
      return
      end
c
c***  subroutine tstep
c***  calculate concentrations at next time step
c***  this is the main mass transport time step routine
c***  executed for each time step, and loops over each segment
c
      subroutine tstep
c
      implicit double precision (a-h)
      implicit double precision (o-z)
c
      include 'vpcomms.f'
c
c***   tm1  -- initial mass in segment
c***   tm2  -- final mass in segment
c***   dmal -- rate change in mass due to advection from left (+ = in)
c***   dmar -- rate change in mass due to advection to right (- = out)
c***   dmdl -- rate change in mass due to diffusion with left (+ = in)
c***   dmdr -- rate change in mass due to diffusion with right (- = out)
c***   dmf  -- rate change in mass due to fracture inflow (+ = in)
c***   dme  -- rate change in mass due to external outflow (- = out)
c***             (occurs at top only)
c
      pcmt = 0.
      fact = 1.e30
      dme  = 0.
      dde  = 0.
c
c***   loop over each segment
c
      itogx = 3-itog
      do i=1,ilim
         ip1  = i+1
         cxti = cxt(i,itog,1)
         dxti = cxt(i,itog,2)
c
c***   starting mass in the segment
c
         tm1  = cxti * vol
         td1  = dxti * vol
c
c***   mass transfer with fracture zone
c
         if (cfrac(i,1).ne.0.) then
            dmf = cfrac(i,1) * qfrac(i)
         else
            dmf = 0.
         endif
         if (cfrac(i,2).ne.0.) then
            ddf = cfrac(i,2) * qfrac(i)
         else
            ddf = 0.
         endif
c
c***   advection/diffusion with next lower cell
c
         if (i.gt.1) then
            im1 = i-1
            dmdl=-dmdr
            dmal=-dmar
            ddal=-ddar
         else
            dmal=0.
            ddal=0.
            dmdl=0.
         endif
c
c***   advection/diffusion with next upper cell
c
         if (i.lt.ilim) then
c
c***   note:  full upstream weighting is used.  without
c***          upstream weighting, these equations would
c***          use (cxti+cxt(ip1,itog))/2. in place of
c***          cxti or cxt(ip1,itog)--that is, the
c***          concentration at the interface would be
c***          the average instead of the upstream value
c
            if (qnxt(i).gt.0.) then
               dmar = -cxti * qnxt(i)
               ddar = -dxti * qnxt(i)
            else
               dmar = -cxt(ip1,itog,1) * qnxt(i)
               ddar = -cxt(ip1,itog,2) * qnxt(i)
            endif
            dmdr = (cxt(ip1,itog,1)-cxti) * dks(i)
            if (qfrac(i).lt.0. .and. i.gt.1) then
c
c***   if the current cell has an outflow,
c***        advection mass in flows out to outflow (as much as
c***         possible)
c
               if (qnxt(i-1).gt.0.) then
                  if (abs(qfrac(i)).ge.qnxt(i-1)) then
                     dmf = -dmal
                     ddf = -ddal
                     if (dmdl.gt.0.) dmf = dmf - dmdl
                  else
                     dmf = -(abs(qfrac(i))/qnxt(i-1))*dmal
                     ddf = -(abs(qfrac(i))/qnxt(i-1))*ddal
                     if (dmdl.gt.0.)
                        dmf = dmf-(abs(qfrac(i))/qnxt(i-1))*dmdl
                  endif
               endif
               if (qnxt(i).lt.0.)  then
                  if (abs(qfrac(i)).ge.abs(qnxt(i))) then
                     dmf = dmf - dmar
                     ddf = ddf - ddar
                     if (dmdr.gt.0.) dmf = dmf - dmdr
                  else
                     dmf = -(qfrac(i)/qnxt(i))*dmar
                     ddf = -(qfrac(i)/qnxt(i))*ddar
                     if (dmdr.gt.0.)
                        dmf = dmf-(abs(qfrac(i))/qnxt(i))*dmdr
                  endif
               endif
               qfmass(i) = -dmf*delt
               qfdens(i) = -ddf*delt
            endif
         else
            dmar=0.
            ddar=0.
            dmdr=0.
c
c***   outflow considered only at top (no mass flow back in at top)
c
            if (qnxt(i).gt.0.) then
               dme = -cxti * qnxt(i)
               dde = -dxti * qnxt(i)
            else
               if (qnxt(i).lt.0.) then
                  dde = -topd * qnxt(i)
               endif
            endif
         endif
c
c***   check for segment mass limitations
c***   if more mass is being moved out of a segment than is in
c***   the segment, write an error message and stop.  This problem
c***   can be fixed by reducing the time step or increasing the
c***   segment length
c
         summo = 0.
         if (dmf .lt.0.) summo = summo - dmf
         if (dme .lt.0.) summo = summo - dme
         if (dmal.lt.0.) summo = summo - dmal
         if (dmdl.lt.0.) summo = summo - dmdl
         if (dmar.lt.0.) summo = summo - dmar
         if (dmdr.lt.0.) summo = summo - dmdr
c
c***   attempted to transfer more than 10% mass out of the cell
c***      --time step is too big
c
         if (summo.gt.0.) then
            pcm = 100.*(summo*delt) / tm1
            if (pcm.gt.pcmt) pcmt = pcm
            if (pcm.gt.100.) then
               f = 100./pcm
               if (f.lt.fact) fact = f
               ifailt = 1
            endif
         endif
c
c***   compute new mass in segment
c
         tm2           = tm1 + (dmf+dme+dmal+dmdl+dmar+dmdr)*delt
         td2           = td1 + (ddf+dde+ddal     +ddar     )*delt
         cxt(i,itogx,1) = tm2/vol
         cxt(i,itogx,2) = td2/vol
      enddo
c
c***   take care of outflows from cell ilim (top)
c
      vout  = qnxt(ilim)*delt
      amout = -dme*delt
      adout = -dde*delt
      if (dme.lt.0.) then
         gtmout = gtmout + vout*(topm/topv)
         topm   = topm   - vout*(topm/topv)
         topm   = topm   + amout
      endif
      if (dde.lt.0.) then
         gtdout = gtdout + vout*topd
```

```
c        write(iout,*)'gtdout=',gtdout
         topd = (adout-(vout*topd)+(topd*topv))/topv
         gtvout = gtvout + vout
      else
         if (dde.gt.0.) then
            din = -vout*topd0
            gtdina = gtdina + din
            topd = (adout+din+(topd*topv))/topv
            gtvina = gtvina - vout
         endif
      endif
c
c***   end of loop over segments
c
c***   if the mass transfer is good (time step not too big),
c
      if (ifailt.eq.0) then
c
c***   add mass to fracture zone volumes if necessary
c
         do i=1,iflim
            if (qin(i).lt.0) then
               vii = abs(qin(i)*delt)
               cii = qfmass(ifseg(i))/vii
               dii = qfdens(ifseg(i))/vii
               ifznum(i) = ifznum(i)+1
               if (ifznum(i).gt.ifzmax) then
c
c***   if fracture is full, combine the older volumes in order to
c***   accomodate more inflow
c
                  do j=1,ifzmax
                     if (j.eq.1) then
                        cfz(i,j,1) = cfz(i,j,1)*vfrac(i,j)
                        cfz(i,j,2) = cfz(i,j,2)*vfrac(i,j)
                     else
                        if (j.le.10) then
                           cfz  (i,1,1) = cfz(i,1,1)
                                            + (cfz(i,j,1)*vfrac(i,j))
                           cfz  (i,1,2) = cfz(i,1,2)
                                            + (cfz(i,j,2)*vfrac(i,j))
                           vfrac(i,1) = vfrac(i,1) + vfrac(i,j)
                        else
                           cfz  (i,(j-9),1) = cfz(i,j,1)
                           cfz  (i,(j-9),2) = cfz(i,j,2)
                           vfrac(i,(j-9)) = vfrac(i,j)
                        endif
                     endif
                  enddo
                  cfz(i,1,1) = cfz(i,1,1) / vfrac(i,1)
                  cfz(i,1,2) = cfz(i,1,2) / vfrac(i,1)
                  ifznum(i) = ifzmax-8
               endif
c
c***   transfer to fracture
c
               vfrac(i,ifznum(i)) = vii
               cfz(i,ifznum(i),1)   = cii
               cfz(i,ifznum(i),2)   = dii
               gtbfm                = gtbfm + vii*cii
               gtbfd                = gtbfd + vii*dii
               gtbfv                = gtbfv + vii
            endif
         enddo
         if (pcmt.gt.pcmax) pcmax = pcmt
c
c***    write out flow at top in liters/min
c
         thrs = t / 3600.
         ql   = qnxt(ilim) * 60000.
         qol  = gtvout * 1000.
c
c***    toggle row pointer
c
         itog=itogx
c
c***    emass is expected total mass in system (total in - total out)
c***    gtm is actual total mass is system
c***    if these numbers differ by more than 0.1% of total input mass,
c***       signal a mass conservation error.  this requires that the
c***       numerical accuracy of all the discrete mass calculations be
c***       at least three digits (99.9%).
c***    this is executed once each time step
c
         call smass
         gtm = smb + smf + topm
         gtd = sdb + sdf + (topd*topv)
         if (gtmina.ne.0.) then
            emass=(tm0 + tmf0 + topm0 + gtmina)-gtmout
            errm=gtm-emass
            errp=(100.*errm)/(tm0 + tmf0 + topm0 + gtmina)
            if (abs(errp).gt.0.1) then
               tm0x = tm0 + tmf0 + topm0
               write(iout,900)errp,gtm,emass,tm0x,gtmina,gtmout,
     .                        smb,smf,topm
               stop
            endif
         endif
         if (gtdina.ne.0.) then
            edens=(td0 + tdf0 + (topd0*topv0) + gtdina)-gtdout
            errd=gtd-edens
            drrp=(100.*errd)/(td0+tdf0+(topd0*topv0)+gtdina)
            if (abs(drrp).gt.0.1) then
               td0x = td0 + tdf0 + (topd0*topv0)
               write(iout,920)drrp,gtd,edens,td0x,gtdina,gtdout,
     .                        sdb,sdf,(topd*topv)
               stop
            endif
         endif
c
      else
c
c*** time step was too big -- cancel mass transfer, reduce delta-t
c
         ot = delt
         delt = 0.5 * (delt * fact)
c
         gtmina = gtmins
         gtdina = gtdins
         gtmout = gtmos
         gtdout = gtdos
         gtbfm  = gtbfms
         gtfbm  = gtfbms
         gtvina = gtvins
         gtvout = gtvos
         gtbfv  = gtbfvs
         gtfbv  = gtfbvs
c
```

```
c***   restore fracture values if the time step is too big
c
         do i=1,iflim
            n          = fsave(i,151)
            ifznum(i) = n
            if (n.gt.0) then
               do j=1,n
                  vfrac(i,j) = fsave(i,j)
                  cfz  (i,j,1) = fsave(i,(j+n))
                  cfz  (i,j,2) = fsave(i,(j+(2*n)))
               enddo
            endif
         enddo
      endif
      return
c
  900 format(' *** NOTE *** mass conservation error ',f10.5,' (%)'/
     .       '    Observed mass in system (kg):',f20.5//
     .       '    Expected mass in system (kg):',f20.5/
     .       '    Initial  mass in system (kg):',f20.5/
     .       '    Added    mass to system (kg):',f20.5/
     .       '    Removed  mass of system (kg):',f20.5//
     .       '    Mass in borehole (kg):       ',f20.5/
     .       '    Mass in fractures (kg):      ',f20.5/
     .       '    Mass in top (kg):            ',f20.5/)
  910 format(2f15.6)
  920 format(' *** NOTE *** dens conservation error ',f10.5,' (%)'/
     .       '    Observed mass in system (kg):',f20.5//
     .       '    Expected mass in system (kg):',f20.5/
     .       '    Initial  mass in system (kg):',f20.5/
     .       '    Added    mass to system (kg):',f20.5/
     .       '    Removed  mass of system (kg):',f20.5//
     .       '    Mass in borehole (kg):       ',f20.5/
     .       '    Mass in fractures (kg):      ',f20.5/
     .       '    Mass in top (kg):            ',f20.5/)
      end
c
      subroutine flowsa
c
c***    frac holds the multipliers for schemes 2 and 3
c
      implicit double precision (a-h)
      implicit double precision (o-z)
c
      include 'vpcomms.f'
c
c***    compute flow up to next segment and total flow
c
      do i=1,ilim
         if (i.eq.1) then
            qtot(i) = qfrac(i)
            qnxt(i) = qtot(i)
         else
            isgn = qtot(i)*qfrac(i)
            if (isgn.ge.0.) then
               qtot(i) = abs(qnxt(i-1) + qfrac(i))
            else
               if (abs(qfrac(i)).le.abs(qnxt(i-1))) then
                  qtot(i) = abs(qnxt(i-1))
               else
                  qtot(i) = abs(qfrac(i))
               endif
            endif
c
```

```
c***   compute flow to next cell up the hole.  qnxt(ilim) is flow
c***     out of (into) the top of the section
c
            qnxt(i) = qnxt(i-1) + qfrac(i)
         endif
      enddo
c
c***   compute maximum time step and Reynold's number
c
      qtmax = qtot(1)
      do i=2,ilim
         if (qtot(i).gt.qtmax) qtmax=qtot(i)
      enddo
      vmax  = qtmax/area
      diam  = 2.*sqrt(area/3.14159)
      rnmax = vmax*diam / 0.3e-6
      dtm   = delt/60.
c
c***   compute the velocity dependent dispersion coefficients
c***   if dispersion type is 2 or 3
c
      if (itypdk.gt.1) then
         dksi = dk*area/delx
c
c***   find first nonzero, minimum, maximum and mean velocities
c
         qfnz = 0.
         qmin = 1.e20
         qmax = 0.
         if (itypdk.eq.2) then
            do i=1,ilim
               if ((qfnz.eq.0.).and.(qtot(i).ne.0.)) qfnz=qtot(i)
               if (qtot(i).gt.qmax) qmax=qtot(i)
               if (qtot(i).lt.qmin) qmin=qtot(i)
            enddo
         else
            do i=1,ilim
               qtoti2 = qtot(i)*qtot(i)
               if ((qfnz.eq.0.).and.(qtoti2.ne.0.)) qfnz=qtoti2
               if (qtoti2.gt.qmax) qmax=qtoti2
               if (qtoti2.lt.qmin) qmin=qtoti2
            enddo
         endif
         qavg = (qmin+qmax)/2.
         if (qavg.eq.0.) then
            write(iout,910)
            stop
         endif
c
         do i=1,ilim
            if (qtot(i).ne.0.) then
               qtoti = qtot(i)
               if (itypdk.eq.3) qtoti = qtoti*qtoti
c
c***   use first nonzero flow for zero flow area
c
            else
               qtoti = qfnz
            endif
            frac(i) = qtoti/qavg
         enddo
c
c***   compute harmonic mean of scaled dispersion parameter
c
         j=ilim-1
         do i=1,j
            qx     = 2./( (1./frac(i)) + (1./frac(i+1)) )
            dks(i) = dksi*qx
         enddo
      endif
      return
c
  910 format(/' *** NOTE *** average flow rate is zero -- aborting')
      end
c
c***   subroutine ctfrac
c***   prepare fracture concentrations for time step
c***   this subroutine is executed during each time step,
c***      and it loops over each fracture
c
      subroutine ctfrac
c
      implicit double precision (a-h)
      implicit double precision (o-z)
c
      include 'vpcomms.f'
c
c***   reset values first
c
      do i=1,iflim
         cfrac(ifseg(i),1)=0.
         cfrac(ifseg(i),2)=0.
      enddo
c
c***   save fracture values in case the time step is too big
c
      do i=1,iflim
         n = ifznum(i)
         fsave(i,151)=ifznum(i)
         if (n.gt.0) then
            do j=1,n
               fsave(i,j) = vfrac(i,j)
               fsave(i,(j+n)) = cfz(i,j,1)
               fsave(i,(j+(2*n))) = cfz(i,j,2)
            enddo
         endif
      enddo
c
c***   compute mass inflow during time step
c***   more than one fracture may be in a segment
c
      do i=1,iflim
         is=ifseg(i)
c
c***   compute concentration
c
         if (qin(i).lt.0.) then
            c = cxt(is,itog,1)
            d = cxt(is,itog,2)
         else
            if (ifznum(i).eq.0.) then
               c = cin(i,1)
               d = cin(i,2)
               gtmina = gtmina + qin(i)*c*delt
               gtdina = gtdina + qin(i)*d*delt
               gtvina = gtvina + qin(i)  *delt
            else
               v    = qin(i)*delt
            cm   = 0.
            dm   = 0.
            vz   = 0.
 500        iz   = ifznum(i)
            viiz = vfrac(i,iz)
            if (v.gt.(vz+viiz)) then
               vz = vz + viiz
               xm = cfz(i,iz,1)*viiz
               ym = cfz(i,iz,2)*viiz
               cm = cm + xm
               dm = dm + ym
               gtfbm = gtfbm + xm
               gtfbd = gtfbd + ym
               gtfbv = gtfbv + viiz
               ifznum(i) = ifznum(i) - 1
            else
               vr = v-vz
               vfrac(i,iz) = vfrac(i,iz)-vr
               cm = cm + cfz(i,iz,1)*vr
               dm = dm + cfz(i,iz,2)*vr
               gtfbm = gtfbm + cfz(i,iz,1)*vr
               gtfbd = gtfbd + cfz(i,iz,2)*vr
               gtfbv = gtfbv +            vr
               vz = v
            endif
            if ((vz.lt.v).and.(ifznum(i).gt.0)) goto 500
            if (vz.lt.v) then
               vr     = v-vz
               cm     = cm + cin(i,1)*vr
               dm     = dm + cin(i,2)*vr
               gtmina = gtmina + cin(i,1)*vr
               gtdina = gtdina + cin(i,2)*vr
               gtvina = gtvina +         vr
            endif
            if (v.ne.0.) then
               c = cm / v
               d = dm / v
            else
               c = 0.
               d = 0.
            endif
         endif
      endif
c
c***   inflow of mass flow during entire time step (kg/sec)
c
         if (qfrac(is).ne.0.) then
            cfrac(is,1) = cfrac(is,1) + c*qin(i)/qfrac(is)
            cfrac(is,2) = cfrac(is,2) + d*qin(i)/qfrac(is)
         endif
      enddo
c
      return
      end
c
c***   subroutine dflows
c***   calculate flows through each segment
c***   this subroutine is called at the beginning of each time step
c
      subroutine dflows
c
      implicit double precision (a-h)
      implicit double precision (o-z)
c
      include 'vpcomms.f'
c
c***   compute the pressures in each segment
c
      wlold  = watlev
      watlev = watlev0
      do idraw=1,idhlim
         if (t.ge.delht(idraw))
            watlev = watlev + delhh(idraw)
         enddo
         if (watlev.ne.wlold) then
            dv    = (wlold - watlev)*area
            if (dv.lt.0.) then
            dm = (dv/topv)*topm
            gtmout = gtmout - dm
            topm   = topm   + dm
            gtdout = gtdout - (dv*topd)
         else
            topdm  = topv * topd
            topd   = topdm / (topv + dv)
         endif
         topv = (xtop-watlev)*area
      endif
c
      dx2 = delx/2.
      press(ilim) = (xtop-watlev)*topd + delx*cxt(ilim,itog,2)
      do i=(ilim-1),1,-1
         press(i) = press(i+1) +
                     dx2*(cxt((i+1),itog,2)+cxt(i,itog,2))
         if (i.eq.ifseg(1))
           write(90,'(2f20.5)')(t/3600.),press(i)
      enddo
c
c
c***   maximum time step is minimum time to flush one-tenth
c***   of a segment.  note that this time may still be too large
c***   if dispersion is significant
c
      if (ifailt.eq.0) then
         if (qtmax.ne.0.) then
            delt = 0.1 * vol/qtmax
            if (delt.gt.tsmax) delt = tsmax
         else
            delt = 0.01 * tsmax
         endif
      endif
c
c***   initialize segment flows
c
      do i=1,ilim
         qfrac(i) = 0.
      enddo
c
c***   compute flow into segments from fractures
c***   it is possible to have more than one fracture in a single
c***   segment
c
      do i=1,iflim
c
c***   compute flow rate as a function of pressure drop
c
         qin(i) = 0.
c
c***   calculate effect of each pressure (drawdown) change
```

```
c
         if (iplim(i).eq.0) then
            iplim(i) = 2
            phist(i,1,1) = hfrac(i)
            phist(i,1,2) = -1.e20
            phist(i,2,1) = xs(ifseg(i))-watlev
            phist(i,2,2) = tstart
         endif
         if ( dabs( press(ifseg(i))-phist(i,iplim(i),1) )
     .       .gt. (delx/2.) ) then
            iplim(i) = iplim(i) + 1
            if (iplim(i).gt.npdps) then
               write(iout,*)'Max iplim is',npdps,'--aborting'
               do iii=1,npdps
                  write(iout,*)phist(1,iii,1)
               enddo
               stop
            endif
            phist(i,iplim(i),1) = press(ifseg(i))
            phist(i,iplim(i),2) = t
         endif
c***       note:  if pressure change occurs exactly on a
c***              time step boundary, there is no result
c
         do ip=2,iplim(i)
c
c***    calculate effect of change between times
c
            if (t.ge.phist(i,ip,2)) then
               qdtid  = avgint(t,phist(i,ip,2),tcoef(i),delt)
               qin(i) = qin(i) +
     .                  ( qdtid * qcoef(i) *
     .                  (phist(i,(ip-1),1)-phist(i,ip,1)) )
            endif
         enddo
         qfrac(ifseg(i))=qfrac(ifseg(i))+qin(i)
      enddo
      call flowsa
      return
      end
c
c***    determine effective dimensionless flow rate for time step
c
      double precision function avgint(t,tp,tcoef,delt)
      implicit double precision (a-h)
      implicit double precision (o-z)
c
      dimension ts(101),qs(101)
c
c***  first check endpoints
c
c     write(6,998)(t/3600.),(tp/3600.),delt
c 998 format(' in avging, t,tp(hr),delt(sec) = ',3f12.6)
      ts(1)    = t - tp
      qs(1)    = qd(ts(1)*tcoef)
      ts(2)    = ts(1) + delt
      qs(2)    = qd(ts(2)*tcoef)
      nvals    = 2
      slopen   = dabs((qs(2)-qs(1))/(ts(2)-ts(1)))
c     write(6,999)slopen,delt
c 999 format(' slope=',e12.5,' delt(s)=',e12.5)
c
      if (slopen.gt.1.e-2) then
         nvals = 101
         ts(nvals) = ts(2)
         qs(nvals) = qs(2)
         dt    = delt/(nvals-1.)
c write(6,*)'Integrating qd...'
c write(6,*)' qs(1)   = ',qs(1)
c write(6,*)' qs(101) = ',qs(nvals)
         do i=2,(nvals-1)
            ts(i) = (t-tp) + (i-1)*dt
            qs(i) = qd(ts(i)*tcoef)
         enddo
      endif
c
      sum = 0.
      do i=1,nvals
         if (i.eq.1 .or. i.eq.nvals) then
            sum = sum + qs(i)/2.
         else
            sum = sum + qs(i)
         endif
      enddo
c     write(6,*)' avg = ',(sum/(nvals-1.))
      avgint = sum / (nvals-1.)
c
      return
      end
c
c***   compute dimensionless flow rate as a function of dimensionless
c***   time.
c***
c***   input:  dimensionless time, td
c***   output: dimensionless flow rate, qd
c***
c***   error conditions:
c***     if td < 1.e-4, qd is set to first value in table

c***     if an interpolation error is detected,
c***                    qd is set to one endpoint value, and
c***                    an error message is printed
c***
c***   solution method:  table lookup and linear interpolation
c***
c***   range:  1.e-4 <= td <= 1.e12
c***
c***   references:  table 1, page 561, of
c***     "Nonsteady flow to a well of constant drawdown in an
c***      extensive aquifer," by C.E. Jacob and S.W. Lohman,
c***      Transactions, American Geophysical Union, v.33, no.4,
c***      August 1952, pp. 559-569.
c***
c***   solution method:   direct numerical approximation
c***
c***   range:  1.e12 < td
c***
c***   references:  equation 4.41 of
c***     "Advances in Well Test Analysis," by Robert C.
c***      Earlougher, Jr., American Institute of Mining,
c***      Metallurgical, and Petroleum Engineers, Inc., 1977,
c***      p. 40.
c***
c
      double precision function qd(td)
      implicit double precision (a-h)
      implicit double precision (o-z)
      dimension qdtd(145),tds(145)
      data qdtd /
     .   56.9, 40.4, 33.1, 28.7, 25.7, 23.5, 21.8, 20.4, 19.3,
     .  18.34,13.11,10.79, 9.41, 8.47, 7.77, 7.23, 6.79, 6.43,
     .   6.13, 4.47, 3.74, 3.30, 3.00, 2.78, 2.60, 2.46, 2.35,
     .  2.249,1.716,1.477,1.333,1.234,1.160,1.103,1.057,1.018,
     .  0.985,0.803,0.719,0.667,0.630,0.602,0.580,0.562,0.547,
     .  0.535,0.461,0.427,0.405,0.389,0.377,0.367,0.359,0.352,
     .  0.346,0.311,0.294,0.283,0.274,0.268,0.263,0.258,0.254,
     .  0.251,0.232,0.222,0.215,0.210,0.206,0.203,0.200,0.198,
     .  .1964,.1841,.1777,.1733,.1701,.1675,.1654,.1636,.1621,
     .  .1608,.1524,.1479,.1449,.1426,.1408,.1393,.1380,.1369,
     .  .1360,.1299,.1266,.1244,.1227,.1213,.1202,.1192,.1184,
     .  .1177,.1131,.1106,.1089,.1076,.1066,.1057,.1049,.1043,
     .  .1037,.1002,.0982,.0968,.0958,.0950,.0943,.0937,.0932,
     .  .0927,.0899,.0883,.0872,.0864,.0857,.0851,.0846,.0842,
     .  .0838,.0814,.0801,.0792,.0785,.0779,.0774,.0770,.0767,
     .  .0764,.0744,.0733,.0726,.0720,.0716,.0712,.0709,.0706,.0704/
      data tds /
     .  1.e-4,2.e-4,3.e-4,4.e-4,5.e-4,6.e-4,7.e-4,8.e-4,9.e-4,
     .  1.e-3,2.e-3,3.e-3,4.e-3,5.e-3,6.e-3,7.e-3,8.e-3,9.e-3,
     .  1.e-2,2.e-2,3.e-2,4.e-2,5.e-2,6.e-2,7.e-2,8.e-2,9.e-2,
     .  1.e-1,2.e-1,3.e-1,4.e-1,5.e-1,6.e-1,7.e-1,8.e-1,9.e-1,
     .   1.e0, 2.e0, 3.e0, 4.e0, 5.e0, 6.e0, 7.e0, 8.e0, 9.e0,
     .   1.e1, 2.e1, 3.e1, 4.e1, 5.e1, 6.e1, 7.e1, 8.e1, 9.e1,
     .   1.e2, 2.e2, 3.e2, 4.e2, 5.e2, 6.e2, 7.e2, 8.e2, 9.e2,
     .   1.e3, 2.e3, 3.e3, 4.e3, 5.e3, 6.e3, 7.e3, 8.e3, 9.e3,
     .   1.e4, 2.e4, 3.e4, 4.e4, 5.e4, 6.e4, 7.e4, 8.e4, 9.e4,
     .   1.e5, 2.e5, 3.e5, 4.e5, 5.e5, 6.e5, 7.e5, 8.e5, 9.e5,
     .   1.e6, 2.e6, 3.e6, 4.e6, 5.e6, 6.e6, 7.e6, 8.e6, 9.e6,
     .   1.e7, 2.e7, 3.e7, 4.e7, 5.e7, 6.e7, 7.e7, 8.e7, 9.e7,
     .   1.e8, 2.e8, 3.e8, 4.e8, 5.e8, 6.e8, 7.e8, 8.e8, 9.e8,
     .   1.e9, 2.e9, 3.e9, 4.e9, 5.e9, 6.e9, 7.e9, 8.e9, 9.e9,
     .  1.e10,2.e10,3.e10,4.e10,5.e10,6.e10,7.e10,8.e10,9.e10,
     .  1.e11,2.e11,3.e11,4.e11,5.e11,6.e11,7.e11,8.e11,9.e11,1.e12/
c
      iout = 6
c
c***   check range of td
c
      if (td.le.tds(1)) then
         qdtemp = qdtd(1)
      else
c
c***   use analytical solution for large values
c
         if (td.ge.tds(145)) then
            qdtemp = 2. / (dlog(td) + 0.80907)
         else
c
c***   compute indices into table
c
            tdl  = dlog10(td)
            itdm = tdl
            if (tdl.lt.0.d0) itdm = itdm-1
            itdf = td/10.d0**itdm
            if (itdf.eq.10.d0) then
               itdf = 1
               itdm = itdm+1
            endif
            i1 = (itdm+4)*9 + itdf
            if (tds(i1).eq.td) then
               qdtemp = qdtd(i1)
            else
c
c***   interpolate table
c
               if (td.lt.tds(i1))   i1 = i1-1
               if (td.gt.tds(i1+1)) i1 = i1+1
               if (i1.lt.1) i1 = 1
               if (i1.gt.145) i1 = 145
               if ((td.lt.tds(i1)).or.(td.gt.tds(i1+1))) then
                  write(6,*)'Error--qd interp; td=',td
                  qdtemp = qdtd(i1)
               else
                  slope  = (qdtd(i1+1)-qdtd(i1)) /
     .                     (tds (i1+1)-tds (i1))
                  delx   = td - tds(i1)
                  qdtemp = qdtd(i1) + delx*slope
               endif
            endif
         endif
      endif
c
      qd = qdtemp
      return
      end
```