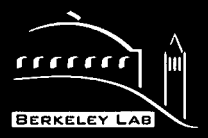
LBNL-39387
UC-000



ERNEST ORLANDO LAWRENCE
BERKELEY NATIONAL LABORATORY

FMGN, RENUMN, POLY, TRIPOLY: Suite of Programs for Calculating and Analyzing Flow and Transport in Fracture Networks Embedded in Porous Matrix Blocks

J. Birkhölzer and K. Karasaki
Earth Sciences Division

September 1996

DISCLAIMER

LBNL - 39387
UC - 000

FMGN, RENUMN, POLY, TRIPOLY:

Suite of Programs for calculating and analyzing flow and transport in fracture networks embedded in porous matrix blocks

Theory, Design, User's Manual and Sample Applications

Jens Birkhölzer and Kenzi Karasaki

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

September 1996

This work was supported by a NATO-Postdoctoral Scholarship, provided by the *German Academic Exchange Organization (DAAD)*, Bonn, Germany, and by the *Power Reactor and Nuclear Fuel Development Corporation (PNC)*, Tokyo, Japan, through the U.S. Department of Energy Contract No. DE-AC03-76SF00098.

FMGN, RENUMN, POLY, TRIPOLY:

Suite of Programs for calculating and analyzing flow and transport in fracture networks embedded in porous matrix blocks

SUMMARY

This report describes a suite of programs developed at the Ernest Orlando Lawrence Berkeley National Laboratory (Berkeley Lab) for simulating flow and solute transport in fracture networks embedded in porous matrix blocks. The codes FMGN, RENUMN and TRIPOLY are extensions of the older codes FMG, RENUM and TRINET developed at the Berkeley Lab, and references are made to previous Berkeley Lab reports which describe those codes.

FMGN generates two-dimensional fracture networks, with any desired distribution of aperture, length, and orientation. RENUMN renumbers the mesh of the fracture network to optimize the bandwidth of the linear equation system to be solved in TRIPOLY and creates the POLY and TRIPOLY input files of the fracture network. The finite element code TRIPOLY solves for flow and transport in two-dimensional fracture networks while fluid or solute exchange processes between the fractures and the porous matrix may be taken into account. The pressure or concentration profile in porous blocks is assumed to be approximately perpendicular to the fracture matrix-interface; thus a one-dimensional mass balance equation can be applied for each block. The geometry of the porous blocks is described by a so-called proximity function. POLY calculates the porous block geometry information for given fracture networks and generates the TRIPOLY input files of the porous matrix blocks.

The first section of this report describes the general background of TRIPOLY and the theory of treating the fluid and solute exchange between fractures and rock. The second section is a user's manual for the programs FMGN, RENUMN, POLY and TRIPOLY. Note that the description of FMGN and RENUMN is very short in this section. FMGN and RENUMN are relatively unchanged from the old codes FMG and RENUM, and only the differences between the old and new versions are listed in this report. For work with FMGN and RENUMN we refer to the detailed description of theory and design in BILLAUX et al. (1988) and the user's manual in BILLAUX et al. (1989), respectively. For the other codes POLY (newly developed) and TRIPOLY (features major changes compared to the old program version TRINET) a detailed user's manual is enclosed in this report. It provides the user with sufficient information to run the programs. The third section of this report comprises some sample problems as a tutorial.

Figure 1 gives a flowchart for the use of the programs in a typical application, and lists the data sets needed for such an application. Listed are only data sets which are mandatory for running the different codes. As indicated, some of these data sets have to be provided by the user. The other data sets are automatically generated by the programs. It is felt that users who have some familiarity with the old codes FMG, RENUM and TRINET can successfully run the newly developed suite of programs with the help of this report only. However, the report is not meant to be comprehensive with regard to FMGN and RENUMN, and first-time users of these codes certainly

need the description of theory and design as well as the user's manual provided in BILLAUX et al. (1988, 1989). Also it might be useful to study the TRINET user's manual (SEGAN & KARASAKI, 1993) for some additional information with regard to TRIPOLY.

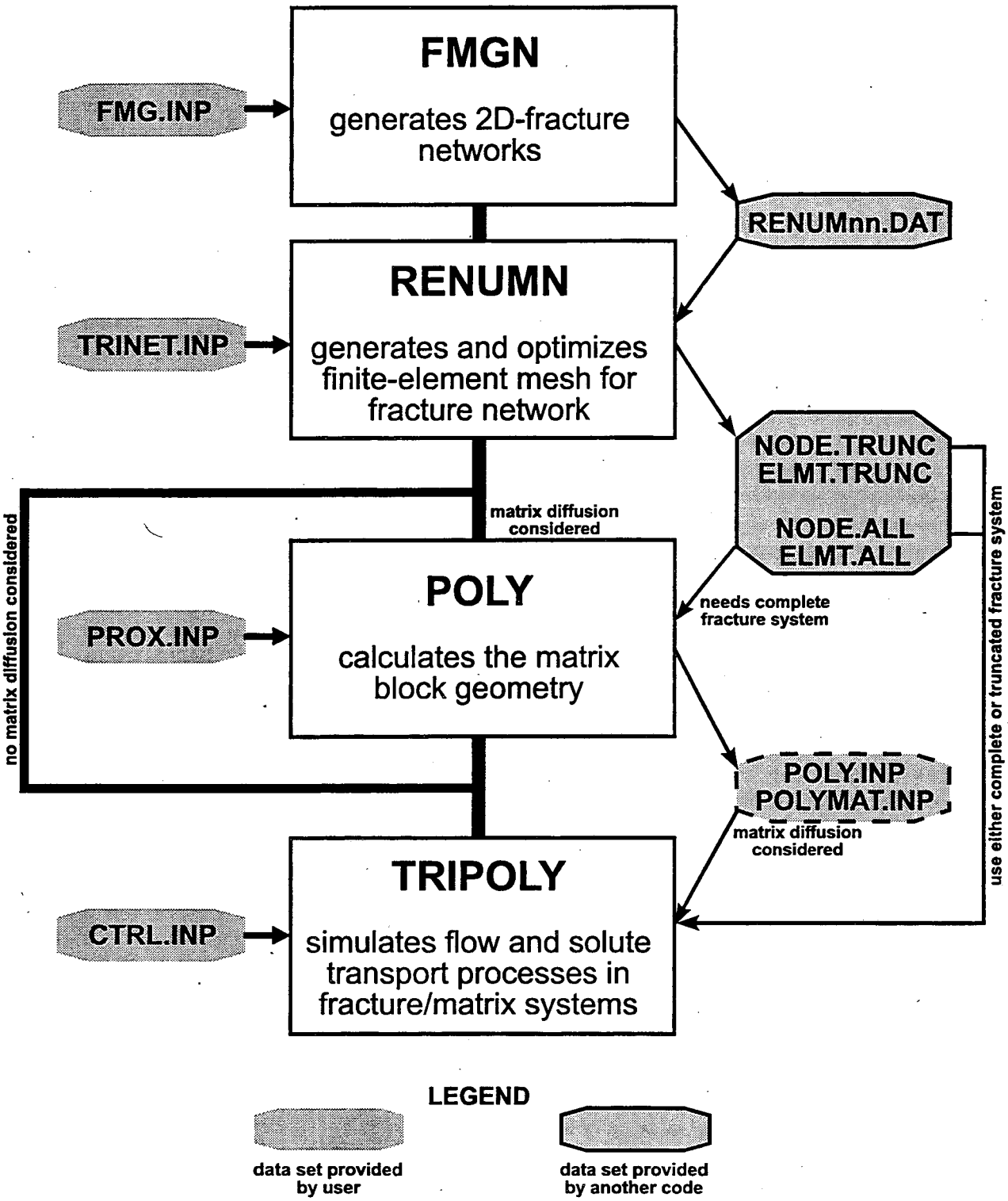


Fig. 1: Flow chart for use of programs

Table of Contents

List of Figures

List of Tables

1 TRIPOLY: Theory and Design

1.1 Introduction

Fracture network models are useful tools for understanding the hydrology of fractured rock. Such studies of fracture hydrology can be carried out by adopting a model for the network geometry, estimating the statistical distribution of the appropriate geometric parameters through field measurements, and generating realizations of statistically identical networks. Once the geometry of a particular realization is specified, flow and transport processes through the network can be studied using adequate numerical procedures. A sophisticated tool for studying such processes is the finite element code TRINET (KARASAKI, 1986; SEGAN & KARASAKI, 1993), developed at the Berkeley Lab. Because it solves the advection-dispersion equation with a mixed Lagrangian-Eulerian scheme combined with adaptive gridding techniques, it can handle very heterogeneous fracture networks with minimal numerical problems such as artificial dispersion or oscillations.

In most cases, fracture network models such as TRINET do not account for solute exchange between the fractures and the porous matrix. However, most of the capacity for storing a pollutant in a fractured formation is provided by the pore system of the rock matrix. Due to the much slower transport in the matrix, strong concentration gradients may occur from the fractures into the porous blocks. This can lead to significant diffusive solute transfer between fractures and matrix and may strongly influence the concentration field in a fractured porous formation. Many workers have shown the importance of these matrix diffusion processes, and some attempts have been made in the past to include fracture-matrix interaction in discrete fracture models.

A straightforward method for handling fracture-matrix interaction is to discretize both the fractures and the matrix blocks, and simultaneously solve for flow and transport in the domain. The fracture-matrix mesh generator FMMG, developed here at the Berkeley Lab, takes a two-dimensional fracture network and discretizes both the fractures (as line elements) and the matrix (as triangles or rectangles) in the system being studied (OKUSU et al., 1989). An adequate numerical tool, capable of handling both line and area elements (hybrid model), would then allow for sophisticated flow and transport simulations. However, due to the strong heterogeneity of the fractured porous medium, a very fine discretization is needed in the matrix blocks, especially at the fracture-matrix interface. Thus, simulation runs may become extremely costly in terms of computer time and space; even for small-scale problems practical limits may be exceeded.

Other workers consider global flow and transport processes only for the fracture network, while using simplified approaches for the fracture-matrix interaction. A very simple model is to work with a retardation factor associated with the fractures. This approach, however, is not very exact, since it is not able to describe the time-dependence of the fracture-matrix interaction. A better representation has been achieved by simulating the local transport in the matrix with a simple analytical solution for one-dimensional diffusion into a semi-infinite half-space (e.g. BIBBY, 1981). This approach is good at approximating the short-term response to perturbations when steep gradients occur at the fracture-matrix boundary, but does not accurately describe the long-term behavior, since the accumulation of solute in matrix blocks of limited size cannot be modeled.

In this report a new code TRIPOLY is introduced which combines the powerful fracture network simulator TRINET with a sophisticated numerical method to account for fracture-matrix interaction. Following the above mentioned approaches, fractures and matrix blocks are treated as two different systems, and the interaction is modeled by introducing sink/source terms in both systems. It is assumed that flow and transport in the matrix can be approximated as a one-dimensional process, perpendicular to the adjacent fracture surfaces. Under that assumption, the geometrical shape of the individual matrix blocks can be described by so-called proximity functions, which determine the fraction of matrix volume within a certain distance from the adjacent fractures. A direct solution scheme is employed to solve the coupled fracture and matrix equations.

The fracture-matrix interaction technique incorporated in TRIPOLY has been successfully used in the past for dual-continuum models (HUYAKORN et al., 1983; BIRKHÖLZER & ROUVE, 1994). It has been shown that it is capable of accurately describing flow and solute transport in arbitrarily shaped matrix blocks. The newly developed combination of the fracture network simulator TRINET and the fracture-matrix interaction module allows for detailed studies of plume spreading processes in fractured porous rock. Since no two- or three-dimensional discretization of the matrix is needed, a remarkable saving of computer time and storage is achieved compared to hybrid models for fractures and rock.

1.2 A Short Note on the Lagrangian-Eulerian Scheme in TRIPOLY

Mixed Lagrangian-Eulerian schemes have been introduced in recent years to avoid numerical problems in the solution of the advection-dispersion equation, especially in advection-dominated problems (e.g. NEUMAN, 1984). The idea is to decompose the advection-dispersion equation in two parts, one controlled purely by advection and the other by dispersion. The advected concentration profiles are calculated by Lagrangian approaches such as particle tracking methods, whereas the dispersed concentration profiles are solved by conventional numerical techniques (FDM, FEM) on Eulerian grids. Often, adaptive gridding schemes are combined with the advection part, introducing forward moving particles around sharp fronts. However, numerical dispersion may occur when the advected front is projected back to the fixed Eulerian grid. Furthermore, the accuracy of results is dependent on the number of particles in the model.

TRIPOLY, based on the Lagrangian-Eulerian finite element code TRINET, features two major improvements compared to the above mentioned methods. First, the advective tracking in the fracture network is performed for nodal concentrations and not for particles. Therefore the number of particles introduced in the model is not an issue. Second, numerical dispersion is minimized by creating new Eulerian grid points instead of interpolating the advected profile back to the fixed Eulerian grid.

In TRIPOLY the flow field in the fracture network is solved by a simple Galerkin finite element method with linear shape functions. The flow can be either steady-state or transient. The advection-dispersion equation for mass conservation is then decoupled into two parts, the advective and the dispersive part. Note that the diffusive solute exchange between fractures and matrix blocks is included in the dispersive part. Thus, the advective problem is first solved without taking the retarding effects of matrix diffusion into account. Then, a correction is made in the second stage

while solving the dispersion equation. This procedure gives rise to some numerical dispersion for large time steps.

The advection equation is solved using the method of characteristics. The concentration profile at the end of the previous time step is the initial-value distribution for the new advection problem. The profile is advected explicitly in the Lagrangian manner according to the velocity in each element. In a first step, single step backward tracking is applied to obtain an advected concentration profile for all nodes. Then, forward tracking is performed in the vicinity of sharp fronts, and new nodes are created when the tracked point does not correspond to a fixed node. This avoids the use of some interpolation scheme when mapping back from the Lagrangian grid to the Eulerian grid, and numerical dispersion is minimized. For both backward and forward tracking, a complete mixing procedure is applied at fracture intersections. If the sharp front has passed though the area, the created nodes are not needed any more and can be eliminated. Of course, the geometry of the fracture network itself has to preserved, and original nodes, which are located at fracture intersections, cannot be eliminated. Thus, at every time step the element catalog has to be revised and nodal points have to be renumbered to keep the bandwidth minimized.

The new concentration profile at the end of the advection stage becomes the initial value for the dispersion-diffusion calculation in the second stage. The equation is solved with a standard Galerkin finite element scheme and linear shape functions. Note that the simulation is performed with the new Eulerian grid, which contains both the fixed nodes and the newly generated nodes. The solute exchange between the fractures and the porous matrix is introduced in the dispersion equation as a sink/source term, which can be calculated in advance in each time step. Thus, the structure of the fracture network equation system is not changed, and a direct solution scheme can easily be employed.

1.3 Fracture-Matrix Interaction

1.3.1 Conceptual Model for Matrix Blocks

In the following section, the theory of treating the fracture-matrix interaction shall be presented only for the transport part, i.e. diffusion into the matrix. The flow problem is similar, but less complicated and at this point we assume that it has been solved. The model at the present time is limited to two-dimensional fracture-matrix systems; however, the approach can be extended easily to three-dimensional problems.

As already stated, TRIPOLY assumes that the global transport processes take place only in the fracture network; the rock medium does not contribute to those processes. However, concentration differences between the fractures and the matrix lead to a diffusive solute exchange at the fracture-matrix interface and portions of the solute may be stored in the matrix pores. As the diffusive transport in the matrix is much slower than the advective-dispersive spreading in the fractures, it can be approximated as a one-dimensional process, perpendicular to the adjacent fractures. Under these assumptions, a one-dimensional mass balance equation describes the diffusive transport in single matrix blocks (see Figure 2)

$$n^M R^M \frac{A(s)}{A_0}\frac{\partial C^M}{\partial t} - n^M D_m^M \frac{\partial}{\partial s}\left(\frac{A(s)}{A_0}\frac{\partial C^M}{\partial s}\right) = 0 \qquad (1)$$

where C^M is concentration in the matrix, n^M is porosity, R^M is the retardation coefficient, D_m^M is the effective molecular diffusion coefficient and s is a local coordinate perpendicular to the adjacent fractures. The local coordinate s is zero at the fracture-matrix interface, and has its upper limit at s=S, which is the maximum orthogonal distance of any location inside the block to the nearest fracture. A(s) is the interface area for transport in the matrix blocks at a distance s from the surface. Hence, for s=0 this area is equal to the contact area with the fracture (i.e. equal to the surface area A_0 of the matrix blocks), and for blocks of limited extent, it steadily decreases when approaching the block center (s=S).

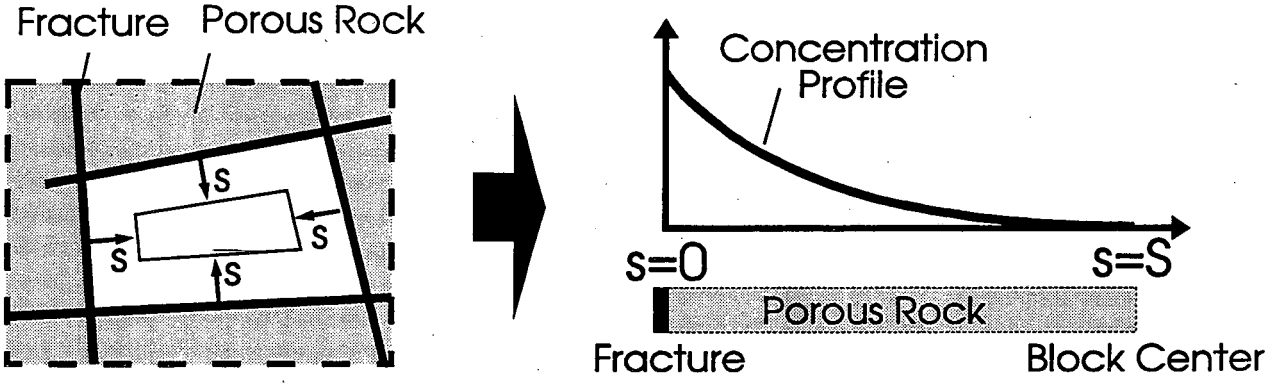


Fig. 2: One-dimensional concentration profile in matrix blocks

Two boundary conditions are needed to solve the local fracture-matrix diffusion problem. First, the concentrations in the fracture and in the matrix block are equal at the fracture-matrix interface:

$$C^M(s=0) = C^F \qquad (2)$$

Second, there is a zero-flux boundary condition at s=S, i.e. in the middle of the matrix blocks:

$$\frac{\partial C^M}{\partial s}(s=S) = 0 \qquad (3)$$

These boundary conditions imply that the diffusion equations for individual blocks are independent from each other; the local concentration profile in the matrix is only affected by the concentration in the adjacent fractures.

After solving equation (1), the diffusive solute exchange per unit fracture wall area is obtained by applying Fick's law at the interface between fractures and porous blocks

$$W^D = n^M D_m^M \left.\frac{\partial C^M}{\partial s}\right|_{s=0} \qquad (4)$$

In fact, W^D in equation (4) is the coupling term between the dispersion-diffusion equation of the fracture network and the diffusion equation of the individual matrix blocks, respectively.

1.3.2 Coupling the Fracture Network with the Matrix Blocks

The fracture network and the matrix blocks are coupled by the solute exchange term (4) which is introduced as a sink/source term in the global dispersion equation system of the fracture network. The dispersion equation for a single fracture can be written as follows

$$\frac{\partial C^F}{\partial t} - D^F \frac{\partial^2 C^F}{\partial x'^2} + \frac{\left(W^{D1} + W^{D2}\right)}{(2b)} = 0 \ , \qquad (5)$$

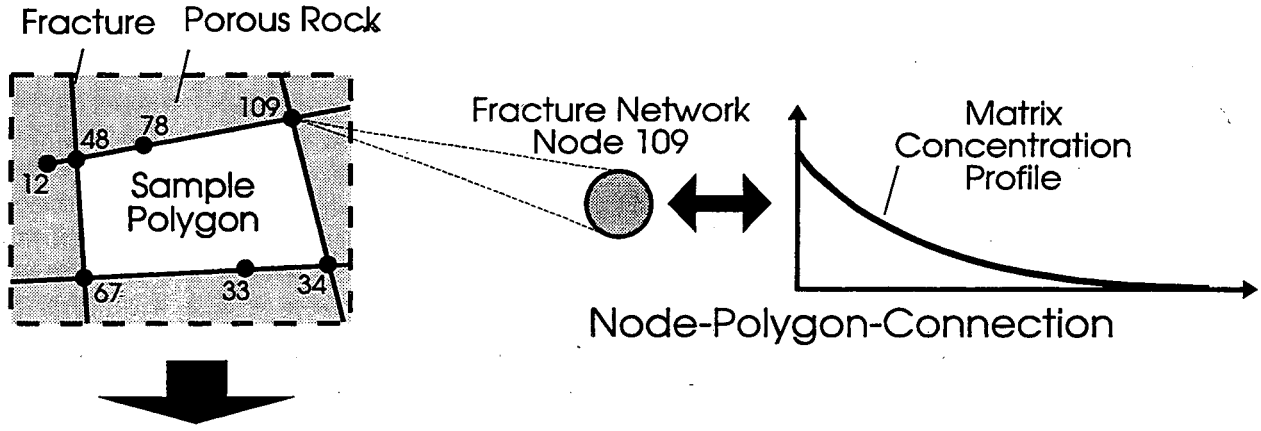
where C^F is the concentration in the fracture, D^F is the dispersion coefficient, (2b) is the fracture aperture and x' is a coordinate defined parallel to the fracture axis. W^{D1} and W^{D2} are the diffusive losses into the matrix on fracture wall one and two, respectively. Since equation (1) does not comprise global space derivatives, the solute exchange term (4) can be treated in a lumped manner in equation (5).

As stated in the previous paragraph, we describe the solute transport in each individual matrix block by a one-dimensional diffusion equation. However, the assumption of only *one* concentration profile being associated with each matrix block requires that all the fractures adjacent to that specific block have a constant concentration value (according to boundary condition (2)). Of course, this cannot be guaranteed in the numerical scheme. Furthermore, it would not be physically plausible to assume that the concentration values in each fracture surrounding a matrix block were identical. TRIPOLY solves this problem by assigning a number of one-dimensional concentration profiles to each block, depending on the number of finite element fracture nodes located at the fracture-matrix interface of the block. Physically, all these profiles should have the same concentration value in the center of the block. However, this requirement cannot always be met since the different diffusion equations associated with a matrix block are solved independently, and the resulting profile is mainly influenced by the concentration at the fracture-matrix interface. Our simulation results show, though, that the concentration differences at the block centers are very small, and that the effect of such differences is negligible with respect to the solute transfer between fractures and matrix blocks.

Figure 3 illustrates the concept of coupling the fracture network and the matrix blocks. Each matrix block in the model area is defined by its material properties (such as porosity and molecular diffusion), by geometrical parameters (interface function and block size S) and by its surface polygon which is described by the node numbers of the surrounding fractures. At the same time, each node of the fracture network is connected to a certain number of blocks (polygons), i.e. one polygon for dead-end nodes, two polygons for moving nodes in between fracture intersections, and more than two polygons for fixed nodes located on fracture intersections. Each of those node-polygon connections is related to a one-dimensional concentration distribution in the matrix. For each connection the solute exchange is calculated according to equation (4), and the resulting exchange rate is introduced into equation (5).

The adaptive gridding scheme implemented in TRIPOLY gives rise to some problems regarding the fracture-matrix interaction. New nodes, which are created during the tracking procedure, have to be connected to the adjacent polygons, and matrix concentration profiles have to be assigned to those nodes. TRIPOLY assumes that the matrix concentration profile of a new node-matrix connection can be obtained by linear interpolation from the matrix concentration profiles

associated with the next upstream node and the next downstream node. At the end of the advection stage, each node of the fracture network is connected to matrix concentration profiles, which become the initial values for the dispersion-diffusion calculation in the second stage. Extensive bookkeeping is needed to keep track of introducing and eliminating new nodes and related matrix concentration profiles.



Matrix Input Parameters in TRIPOLY:

material properties:	e.g. porosity, molecular diffusion coefficient
geometry:	surface polygon (here: 67; 33; 34; 109; 78; 48) interface function, maximum orthogonal distance S

Fig. 3: Defining matrix blocks in TRIPOLY

1.3.3 The Solution Scheme

Each individual matrix block in the domain is associated with a certain number of one-dimensional diffusion equations, describing the local transport in the matrix. As the concentration profiles in matrix blocks do not directly affect each other, the different diffusion equations are independent. However, each of these equations is coupled to the advection-dispersion equation of the fracture network via the solute exchange terms W^D. Two different numerical solution procedures can be chosen for the coupled equation system of fractures and matrix:

1. The two media are discretized and solved separately. The coupling may require iterative procedures, unless efficient direct solution schemes are available.
2. The two media are discretized simultaneously and directly solved in one equation system. However, this equation system would be much bigger than in 1 and might easily exceed the available computer memory.

TRIPOLY uses the first solution procedure and features a direct solution technique which was originally proposed for dual-porosity models (HUYAKORN et al., 1983; BIRKHÖLZER & ROUVE, 1991).

At each time step, the independent matrix diffusion equations associated with each fracture node are solved prior to the solution of the dispersion-diffusion equation of the fracture network. Thus,

the fracture concentrations of the current time step are still unknown at this stage, which means that boundary condition (2) is unknown and must be treated as variable. However, as shown later, it is possible to evaluate the mass transfer term (4) with a linear dependence on the unknown fracture concentrations. Then, the mass transfer terms of all nodes are inserted into equation (5) and a linear solver can be applied to obtain the nodal concentrations of the fracture network. Finally, the concentration profiles of the porous matrix blocks are evaluated by a backward substitution.

In the following paragraph, the solution procedure will be described briefly. The individual matrix diffusion equations are solved by a one-dimensional standard Galerkin finite element procedure. Since this method is widely used in groundwater hydraulics, it will not be explained in this report. Application of the Galerkin technique to equation (1) finally yields a tridiagonal set of equations for each matrix block which can be represented in the following manner

$$\begin{vmatrix} b_1 & c_1 & 0 & . & . & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & . & . & 0 \\ 0 & a_3 & . & . & 0 & . & . \\ . & 0 & . & . & . & 0 & . \\ . & . & 0 & . & . & c_{N-2} & 0 \\ 0 & . & . & 0 & . & b_{N-1} & c_{N-1} \\ 0 & 0 & . & . & 0 & a_N & b_N \end{vmatrix} \bullet \begin{vmatrix} C_1^M \\ C_2^M \\ . \\ C_n^M \\ . \\ C_{N-1}^M \\ C_N^M \end{vmatrix} = \begin{vmatrix} d_1 - W_J^D \\ d_2 \\ . \\ d_n \\ . \\ d_{N-1} \\ d_N \end{vmatrix} \qquad (6)$$

C_n^M is the current value of concentration at node n of the one-dimensional solution domain, N is the number of nodes in the domain and a_n, b_n, c_n and d_n are known coefficients. Node number 1 is associated with the block surface (i.e. s=0) and boundary condition (2), node N is associated with the center of the block (i.e. s=S) and boundary condition (3). Linear basis functions are used with an implicit finite difference approximation for the time integration. An exact spatial integration is performed for the element matrices, which means that A(s) is treated as a function rather than a constant value in each element. Since both the basis function and the interface function are polynomials of s, an analytical integration can easily be performed. The coefficients a_n, b_n and c_n on the left-hand side of the equation system comprise components of the first and the second term in equation (1), respectively. The coefficients d_n on the right-hand side comprise components of the first term in equation (1), multiplied with the concentrations of the old time step.

W_J^D denotes the solute exchange between the fractures and the matrix per unit interface area, associated with node J in the fracture domain. According to boundary conditions (2) and (3), an inflow/outflow of solute is only possible at node 1 of the matrix domain, i.e. at the fracture-matrix interface. Both the value of the solute exchange and the nodal concentrations are unknowns at this point.

Using the general Thomas algorithm (THOMAS, 1949), one can factor the tridiagonal matrix into a product of lower and upper bidiagonal matrices and perform a forward elimination. After setting $w_N=b_N$, the following steps are performed

$$u_n = a_n / w_n, \quad \text{for} \quad n = N,\ 2 \tag{7a}$$

$$w_n = b_n - c_n u_{n-1}, \quad \text{for} \quad n = N-1,\ 1 \tag{7b}$$

$$g_N = \frac{d_N}{w_N} \tag{7c}$$

$$g_n = \frac{d_n - c_n g_{n+1}}{w_n}, \quad \text{for} \quad n = N-1,\ 2 \tag{7d}$$

For n=1 and knowing that $g_1=C_1^M$, one finally obtains the following expression for W_J^D

$$W_J^D = d_1 - w_1 C_1^M - c_1 g_2 \tag{8}$$

Using boundary condition (2), the unknown concentration of the first matrix node C_1^M in equation (8) can be replaced by the unknown concentration C_J of fracture node J. Apart from this concentration value, all remaining coefficients in equation (8) are known quantities. Inserting (8) into equation (5) and performing this procedure for all node-polygon connections eventually gives an equation system for the fracture domain which can be directly solved for the current values of the fracture concentrations. Once the fracture concentrations C_J are obtained, the one-dimensional concentration distributions in the matrix blocks can be readily determined by performing a backward substitution as follows

$$C_1^M = C_J^F \tag{9a}$$

$$C_n^M = g_n - b_n C_{n-1}^M, \quad \text{for} \quad n = 2,N \tag{9b}$$

This completes the solution cycle for the present time step.

1.3.4 Proximity Functions

According to the one-dimensional diffusion equation (1), interface functions A(s) have to be provided, which define the interface area for diffusive transport at a distance s from the block surface. For blocks of limited extent, such an interface function would be equal to the block surface at s=0, and would steadily decrease when approaching the block center (s=S).

In TRIPOLY, interface functions for each block are defined following the concept of proximity-functions, which was originally proposed by PRUESS & KARASAKI (1982). The proximity-function Prox(s) expresses the total fraction of matrix volume V(s) within a distance s from the adjacent fractures, divided by the total block volume V

$$\text{Prox}(s) = V(s)/V \tag{10}$$

For regularly shaped blocks, proximity-functions can easily be derived from analytical expressions. For example, a square matrix block of side length a has a proximity function

$$\textbf{Prox(s)=4s/a-4s}^2\textbf{/a}^2.$$

A rectangular matrix block with side lengths **a** and **b** (while **b = 2a**) has a proximity function

$$\mathbf{Prox(s)=3s/a-2s^2/a^2.}$$

In both cases the coordinate s is defined from s=0 to s=a/2.

The interface area for diffusive flux in the matrix is simply the derivative of the proximity-functions, multiplied with the total block volume

$$A(s) = \frac{dV(s)}{ds} = V\frac{d\,Prox(s)}{ds} \qquad (11)$$

In TRIPOLY, the interface function is divided by the actual surface area A_0 of each matrix block to normalize the values. Then, according to equation (11), the normalized interface function for the square matrix block and the rectangular matrix block would be

$$\mathbf{A(s)/A_0 = (4a-8s)/(4a)} \text{ and } \mathbf{A(s)/A_0 = (6a-8s)/6a},$$

respectively. Note, that in the first case the value of the normalized interface function at the block center (s=a/2) is equal to zero, in the second case, however, it is equal to 1.0/3.0 (see Figure 4). Consequently, then, the diffusive transport behavior in the block is different for the two cases.

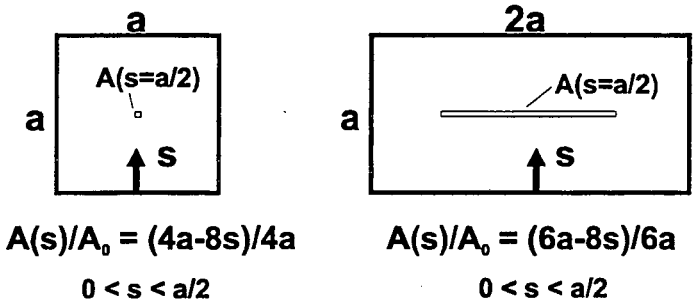


Fig. 4: Normalized interface function $A(s)/A_0$ for regular block shape

For irregular blocks, a random procedure is needed to derive proximity functions. A number of points are randomly distributed in each block, and the orthogonal distance to the nearest fracture is measured. The value of the proximity-function at the coordinate s is then given by the fraction of points within a distance s from the fracture surfaces, divided by the total number of points. Finally, the results of the random procedure are approximated by a best-fit-polynomial (see Figure 5). Such a polynomial has to be determined for each matrix block.

All the geometrical information needed for describing the porous matrix blocks is determined within the preprocessing code POLY. TRIPOLY reads the results of the preprocessing procedure (such as proximity functions, block sizes etc.) and automatically calculates the interface functions.

Note that the concept of proximity function was originally introduced to describe the characteristic geometry of a large number of matrix blocks. This is needed, for example, in the case of dual-porosity models for fractured porous formations when averaged (equivalent continuum) parameters have to be supplied for a given subdomain. In this case, the above described random procedure would simply be performed for all the blocks at the same time (rather than performing it for

each block separately), and **one** proximity function would be calculated describing the averaged geometry of all blocks in a given subdomain.

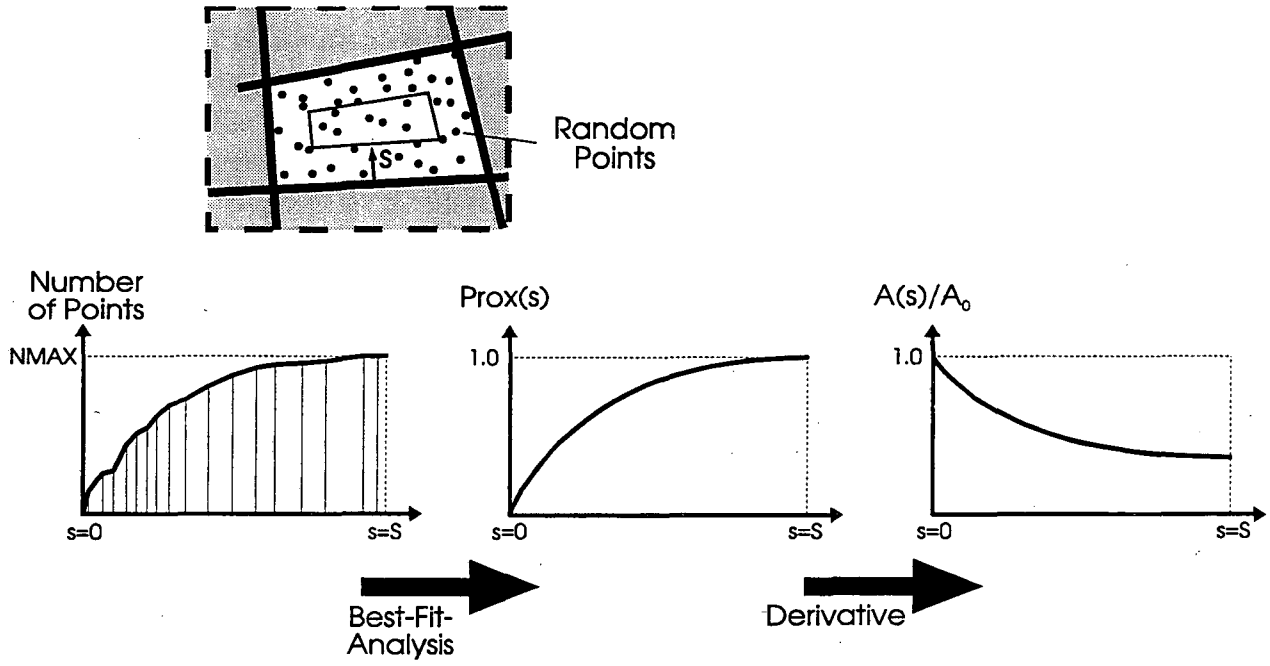


Fig. 5: Proximity function Prox(s) and normalized interface function $A(s)/A_0$ for irregular blocks

1.4 A Short Note on the Performance and Accuracy of TRIPOLY

The numerical solution of advection-dispersion processes in fracture-networks is a complex task, since natural fracture networks are very heterogeneous with regard to flow velocities. In some fractures transport may be advection-dominated (hyperbolic type), with an almost sharp front moving through the system, while transport in other fractures may be diffusion dominated (parabolic type). Most conventional numerical solution techniques are well-suited to handle either hyperbolic equations or parabolic equations, but not both simultaneously. Thus, many existing numerical models may, if not carefully used, give rise to inaccuracies in form of unwarranted oscillations or the smearing of steep concentration gradients. For example, if conventional Eulerian schemes are chosen to solve the advection-dispersion equation, the discretization in space and time must be designed using constraints defined by the dimensionless Peclet number, $Pe = L/\alpha_l$, and the dimensionless Courant number, $Cou = (v\ dt)/L$, where L is the length of a fracture element, α_l is the longitudinal dispersivity, v is the advective velocity, and dt is the time step. Often, a Peclet number smaller than 2, and a Courant number smaller than 1 are recommended to avoid numerical inaccuracies. Thus, in case that the velocity is large and the dispersivity is small, these constraints gives rise to very small element lengths and time step sizes, making the computation very inefficient. Conventional Lagrangian schemes are more efficient in that they do not require these constraints; however, they can not handle diffusion type equations.

In contrast to the above solution schemes, Lagrangian-Eulerian methods such as the one incorporated in TRIPOLY are capable of handling Peclet numbers from 0 to ∞ with negligible oscillations and numerical dispersion, while using large time steps with Courant numbers well in excess of 1. However, due to the nature of Lagrangian-Eulerian schemes, the performance and accuracy of TRIPOLY somewhat depends on the temporal and spatial discretization of the problem, and on the appropriate choice of different control variables (see Section 2.5.3). For example, the user-provided variable DCON defines a threshold concentration difference to decide whether in case of sharp gradients a new moving node has to be originated between two existing nodes. If DCON is too large, the computation may become inaccurate, because new nodes, which would actually be needed, are not introduced. If DCON is too small, the computation may become inefficient, because new nodes are introduced, which are not needed for obtaining an accurate solution. It is strongly recommended that the sensitivity of user-provided control variables is studied in each particular application, to (1) get a better feeling for the impact of these variables and (2) to optimize the performance and accuracy of the model.

Our experience shows that TRIPOLY allows for an efficient and accurate simulation of solute transport in discrete fractured-matrix systems, for a wide range of applications and a wide range of Peclet and Courant numbers. Only if the diffusive exchange between fractures and matrix is very strong compared to the advective-dispersive transport in the fractures, the user must pay some attention to the time step sizes used. This is because the interaction between fractures and matrix blocks is included in the dispersive part, i.e. the advective problem is solved without taking the retarding effects of matrix diffusion into account. Thus, if time steps are very large, a solute plume might travel a long distance in the fracture network within the advective step, and then it is be pulled back in the dispersive step by the effect of matrix diffusion. This can give rise to numerical dispersion. Therefore, the user should routinely check whether the concentration values obtained in the advective step are very different from the final concentration values after the dispersive step. If that is the case, the sensitivity of the results should be checked for different time step sizes.

2 User's Manuals

2.1 Compiling and Running the Codes

FMGN, RENUMN, POLY and TRIPOLY are written in FORTRAN 77. The codes have been compiled on a variety of machines. In the light of modern compilers and the fairly standardized input/output of FORTRAN 77 only minor changes are necessary in porting from one machine to the next. In general these changes will not be noted in this document. The release version of the code was compiled and executed on a SUN Sparc Workstation.

FMGN and RENUMN are compiled with the use of a Makefile. This is a case specific file used by the UNIX *make* program, which facilitates compiling, linking, and loading of an executable for which numerous smaller modules are involved. As is the case with FORTRAN 77, the make utility will be the same in scope from one UNIX operating system to the next, but details may change.

As all the subroutines in POLY and TRIPOLY are comprised in one file rather than in numerous modules, compiling and linking need not to be done with a *make* procedure. The user may run the standard compiling and linking procedures provided by the particular machine.

The dimension of arrays in the codes is determined in an external *common* block (COMMON.BLK) which is loaded into the different subroutines by the *include* statement. The dimension should be adjusted from case to case according to the size of the problem which is to be solved. Generally, this can be done by changing maximum variables which are given in the *parameter* statement in COMMON.BLK. The names of these variables indicate their function, e.g. MAXNO is the maximum number of nodes of the fracture mesh, MAXEL is the maximum number of elements, MXPLY is the maximum number of blocks (polygons) etc. Note that the fracture-matrix interaction modules in TRIPOLY require large arrays to store the matrix properties and variables. Thus, the maximum polygon number MXPLY should be chosen carefully to avoid an unnecessarily large assignment of computer storage.

If problems are encountered when attempting to build an executable code, either with FORTRAN 77 or the Makefile characteristics, feel free to contact the authors of this report. However, in many cases local machine documentation will very likely indicate the minor changes necessary to insure proper building of the executable. In the case that serious errors are detected, please notify the Technical Contact at once.

2.2 FMGN

The new version FMGN is similar to the fracture mesh generator FMG, which is widely used within the Earth Sciences Division of the Berkeley Lab. FMG generates fractures as line discontinuities in a two-dimensional space, with any desired distribution of aperture, length, and orientation. The locations of these fractures can be either specified or generated randomly. The intersections of these fractures with each other, and with the boundaries of a specified flow region, are determined, and a finite element network is output. The theory and design of FMG is described in BILLAUX et al. (1988). Another Berkeley Lab report (BILLAUX et al., 1989) serves as a user's manual and lists input data sets, input variables and their formats.

Only minor adjustment have been made in FMGN compared to the old code FMG. Generally, natural fracture systems comprise a network of *conducting* fracture segments, which at both end-points connect to either the conducting network or to the domain boundary, and a number of *non-conducting* fracture segments, which connect only at one end-point (see Figure 6). We refer to these non-conducting segments as "dead-ends". Since the fluid velocity in dead-ends is zero, they can only be contaminated by diffusive processes. Often their impact on solute transport in natural fracture networks is negligible, so that in most simulation runs only the conducting network will be considered. Therefore the old code FMG offers the choice of generating fracture systems with or without dead-ends, depending on the user-defined input variable IKEEP. However, the algorithm for calculating the matrix block geometry used in POLY requires a complete fracture system with all fracture segments including dead-ends, even when the solute transport simulation in TRIPOLY is done for the conducting network only. Therefore the new version FMGN always generates a fracture system with dead-ends, regardless of the input variable IKEEP. If for the simulation with TRIPOLY desired the elimination of dead-ends can be performed at a later stage using the code RENUMN. Note that both codes FMG and FMGN automatically eliminate isolated fracture segments or fracture clusters which are not connected to the conducting network at all.

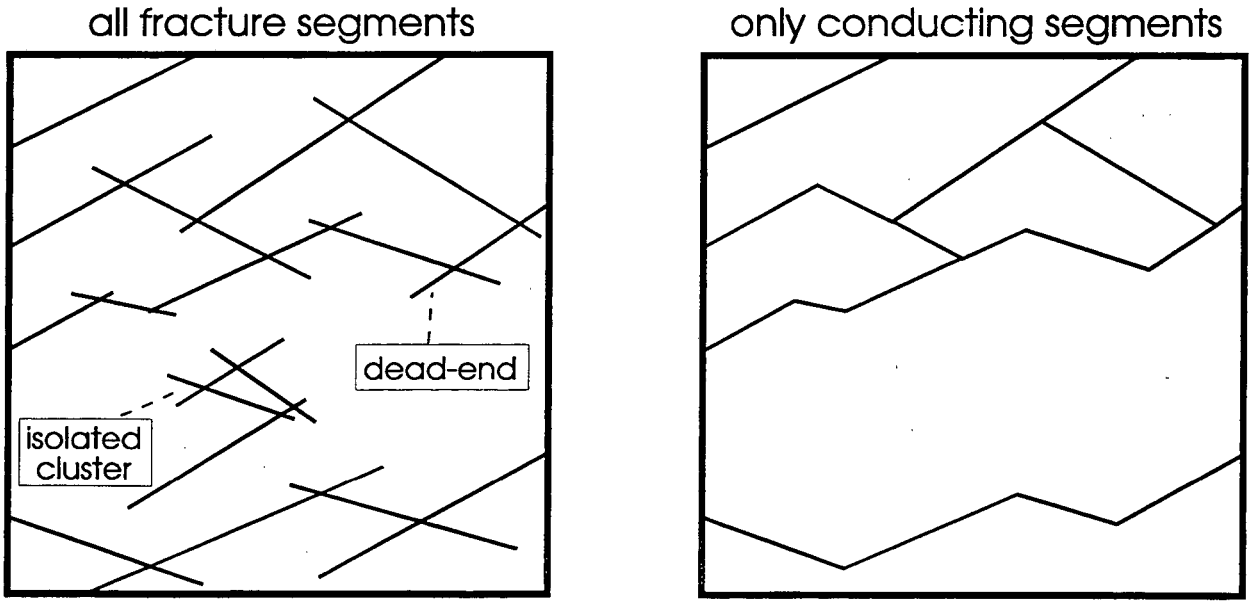


Fig. 6: Fracture system with all fracture segments and conducting segments only

While the input data sets for FMGN are exactly the same as for FMG, minor changes are made in the output data sets. Since certain simulation or visualization software is not in use anymore, some output data sets of the original FMG version are not generated by FMGN (e.g. LINESGR.DAT, LINES00.DAT and LINESnn.DAT). FMGN assumes that the input variable IPLOT is set to 0, even if the user sets it to a different value. All input data sets, variables and formats are described in BILLAUX et al. (1989). A list of the input/output data sets is given in Table 2.1. Input files in bold letters are mandatory, the other input data sets are only needed for certain options.

Table 2.1 FMGN - Input/Output Files

Name	Input/Output	Unit	Description
FMG.INP	Input	1	All input data for fracture mesh generation
SUBREG.DAT	Input	7	May contain part of FMG.INP information
FRAC.DAT	Input	8	All primary fracture system data (no random generation)
STUDY.DAT	Input	20	May contain part of FMG.INP information
RENUMGR.DAT	Output	4	Header for generation region
RENUM00.DAT	Output	4	Header for first flow region
RENUM**nn**.DAT	Output	4	Input data for RENUMN for flow region **nn**
FRAC.TXT	Output	5	Lists certain information about fractures
FMG.OUT	Output	6	Statistics on fracture sets
FRAC.DAT	Output	8	All primary fracture system data, may be used as input
CONNECTIONS	Input/Output	50	Number of runs for which a connected fracture network was generated (in some cases, only finite size clusters of fractures may exist, e.g. in case of sparse fracturing)

2.3 RENUMN

The new version RENUMN is similar to the line network optimizer RENUM, which is widely used within the Earth Sciences Division of the Berkeley Lab. RENUM reads the input generated by FMG, merges nodes very close to each other, removes dead-ends (if desired) and renumbers the nodes in order to minimize the bandwidth of the equation system. Two Berkeley Lab reports (BILLAUX et al., 1988, 1989) describe the theory and design of RENUM and serve as a user's manual.

The original version of RENUM has been changed with regard to the treatment of dead-end fractures. While RENUM either discards dead-ends or keeps them depending on the value of the user-specified input variable IKEEP, RENUMN generates both types of fracture networks, regardless of IKEEP. The node and element information is written to NODE.TRUNC, ELMT.TRUNC for the truncated network and to NODE.ALL, ELMT.ALL for the complete network. This change has been made since POLY needs a complete fracture network for calculating the porous block information. After running RENUMN the data sets NODE.TRUNC and NODE.ALL have different node numbering due to the effort of minimizing the bandwidth. In order to relate the nodal information of both networks, the data sets contain the old node number of the FMG input data set RENUMnn.DAT. Negative old node numbers in NODE.ALL indicate that this node is a dead-end node and is discarded in NODE.TRUNC.

Most input data for RENUMN are provided by FMGN, contained in the file RENUMnn.DAT. The numbering **nn** stands for different flow regions which can be defined inside of the model area within FMGN. While running RENUMN, the user is interactively asked which of the different flow regions should be chosen for the network optimizer. No. 1 would relate to RENUM01.DAT (i.e. the first flow region), No. 2 to RENUM02.DAT, etc. However, in most cases only one flow region is assigned in FMGN.

Another input data set that gives some general information on the material parameters of the fracture network (e.g. specific storage and dispersion/dispersivity; values for density, dynamic viscosity and gravity, the latter needed to calculate fracture transmissivities) must be provided by the user. This data set is called TRINET.INP and replaces the data set INTER.INP of the previous version RENUM. If TRINET.INP does not exist on the current work directory, RENUMN uses default values for the material properties.

As mentioned above, RENUMN generates four output data sets, NODE.TRUNC, ELMT.TRUNC for the truncated network and NODE.ALL and ELMT.ALL for the complete network. The values for specific storage and dispersion/dispersivity (provided in TRINET.INP) are written as spatially constant 'default' element properties into ELMT.TRUNC and ELMT.ALL. However, the user may change these data sets and provide different values for each element. The transmissivity of each element is computed according to the cubic law which considers the actual width of the fracture. Again, the user may eventually change these values before running simulations with TRIPOLY. Some data sets like LINESxx.DAT, LINEL.INP or CTRL.INP, which were written by the old program version RENUM, are no longer generated.

The initial conditions and the boundary conditions for the flow and transport problem are determined in file NODE.ALL and NODE.TRUNC (see Section 2.5.3). The user can provide information about the flow boundary condition in file FMG.INP, which is the input data set for the fracture mesh generator FMGN (see BILLAUX et al., 1989). For example, each side of a rectangular model area can be associated with either constant fluxes, constant heads or a constant linear distribution of head. The flow boundary information created by FMGN is used as input for RENUMN and written into NODE.ALL and NODE.TRUNC. The transport boundary conditions, however, usually vary from problem to problem (e.g. line source, point source, source on boundary or inside of model area); the same is true for the initial conditions. Therefore, this information cannot be specified in FMG.INP. Instead, the user must either adjust subroutine *triout* in RENUMN which writes the output data sets, or directly set the boundary conditions in NODE.ALL and NODE.TRUNC by editing them.

Table 2.2 provides a list of all input and output files for RENUMN. Bold input file names indicate that the data set is mandatory. Table 2.3 lists the input variables of TRINET.INP and their formats. These variables are described in Table 2.4. All other input data sets, variables and formats are described in BILLAUX et al. (1989).

Table 2.2 RENUMN - Input/Output Files

Name	Input/Output	Unit	Description
RENUMnn.DAT	Input	1	Input data for RENUMN written by FMGN
TRINET.INP	Input	3	General information specifying material parameters
NODE.TRUNC	Output	9	Node information for truncated network
ELMT.TRUNC	Output	10	Element information for truncated network
NODE.ALL	Output	9	Node information for network with dead-ends
ELMT.ALL	Output	10	Element information for network with dead-ends
RENUM.ERR	Output	11	Run-time error messages

Table 2.3 RENUMN - Input Variables in TRINET.INP

Variable	Format	Input Unit
TITLE2	a80	3
RHOR, RMUR, GRAVR	3(10x,f10.0)	3
SSUBS, DISPC	2(10x,f10.0)	3

Table 2.4 RENUMN - Description of Input Variables in TRINET.INP

Variable/Array	Description
DISPC	Dispersion coefficient or longitudinal dispersivity in fracture elements. Default value is 0.0 m^2/sec or 0.0 m, respectively.
GRAVR	Gravitational constant. Default value is 9.8067 m/sec^2.
RHOR	Density of fluid. Default value is 1000.0 kg/m^3.
RMUR	Dynamic viscosity of fluid. Default value is 0.001 kg/(m sec).
SSUBS	Specific storage in fracture elements. Default value is 0.0 m^{-1}.

Sample data set:

TRINET.INP

```
TEXT
RHOR     =1.0000E+03 RMUR     =1.0000E-03 GRAVR    =9.8067E+00
SSUBS    =0.1000E-04 DISPC    =5.0000E-02
```

2.4 POLY

2.4.1 Code Structure

The preprocessing code POLY calculates the geometrical properties of the matrix blocks in a given fracture network. POLY is only needed for TRIPOLY simulation runs when the fluid or solute exchange between the fracture mesh and the porous matrix shall be taken into account. The porous matrix blocks are described by polygons, defined by the node numbers of the surrounding fractures. Usually the nodes and elements of the fracture mesh are related to more than one polygon. TRIPOLY assumes that flow and transport in the matrix can be approximated as one-dimensional processes, perpendicular to the fracture surfaces. Thus the shape of the matrix blocks is given by the maximum orthogonal distance of the block center to the fractures plus the so-called proximity function which is provided as a polynomial of user-specified grade.

The algorithm in POLY starts by extending dead-end fractures until they intersect either another fracture or the flow region boundary. This process ensures that the polygons formed by fractures and fracture extensions have a simple convex shape and can easily be determined. The subroutines for extending the fracture ends and defining convex polygons are taken from the code FMMG (OKUSU et al., 1989) which is used for discretizing two-dimensional fracture-matrix systems. However, these simple polygons define only parts of the original matrix blocks; i.e. parts which are separated from each other by the extended fracture ends. Thus, in the next step, polygons sharing the same extended fracture section are connected, the extended sections are truncated and new, complicated polygons are created. Finally, all matrix blocks in the flow region are defined by the node numbers of the surrounding fractures and their coordinates (see Figure 7).

The second major part of POLY determines the geometrical shape of the matrix blocks, given by the proximity function and the maximum orthogonal distance between any location inside of a block and the nearest fracture. For regularly shaped blocks, such as triangles or rectangles, the proximity function and the maximum distance are derived from analytical expressions. For complicated blocks, however, a random procedure is needed. A given number of points is randomly distributed inside of each block, and the orthogonal distance to the nearest fracture surface is determined. The value of the proximity function at coordinate s is then given by the fraction of points within the distance s from the fracture surfaces. An approximation procedure after PRESS et al. (1986) is finally applied to determine a best-fit-polynomial of given grade; the derivative of that polynomial defines the interface function.

For matrix blocks of very complicated shape, the best-fit polynomial may not increase steadily from s=0 to s=S. In such cases, the interface function, as a derivative of the proximity function, would have negative values, and components of the element matrices in the one-dimensional Galerkin finite element approximation of equation (1) would become less than zero. To avoid such numerical problems, POLY features an automatic check for the calculated best-fit polynomials. If a problem arises for a certain polygon, a warning is written on the screen, and POLY assumes the shape of the block to a square. Then the proximity function is calculated by analytical means. The size of the square is equal to the size of the original block to guarantee mass continuity. The user should make sure that this substitution does not happen too often, since the

actual block shapes are not exactly represented. Sometimes, increasing the number of random points or changing the grade of polynomials improves the best-fit analysis.

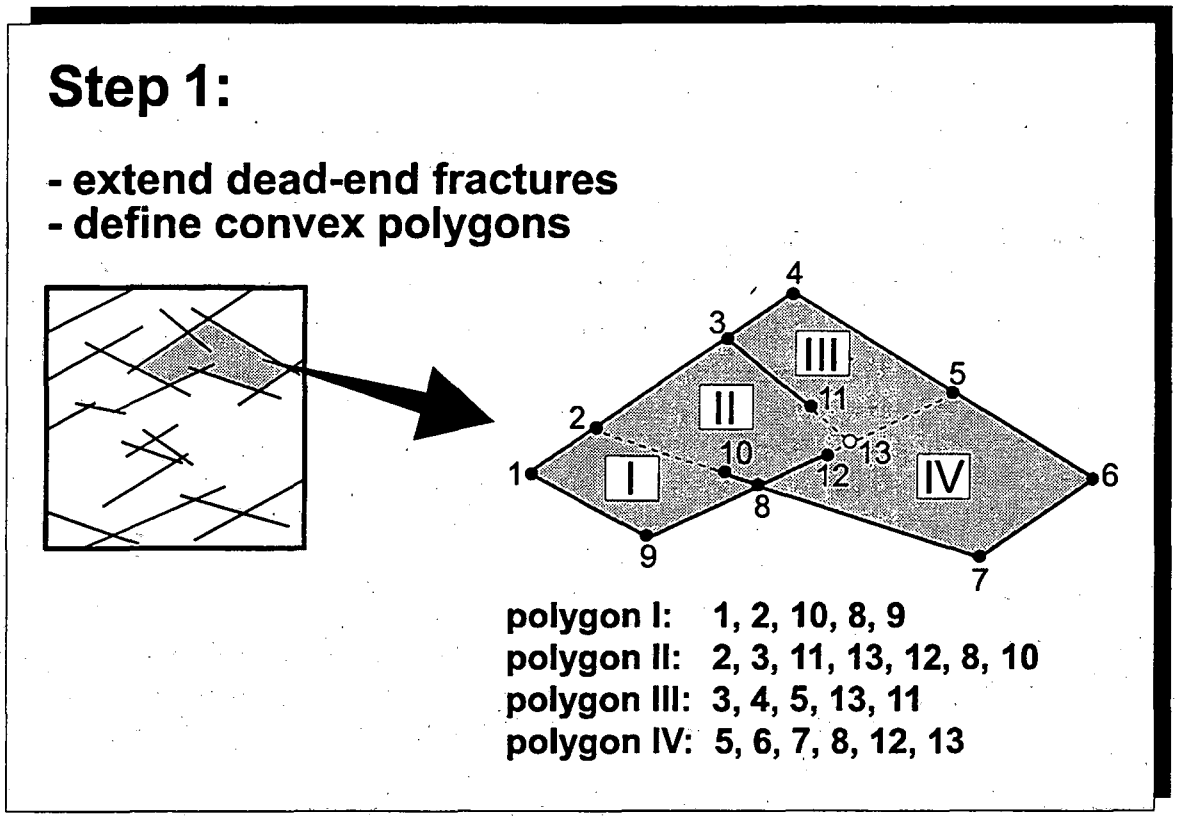


Fig. 7: Calculation of the polygons defining the block geometry

2.4.2 Input/Output Files

- **Input**

  POLY needs the input files NODE.ALL and ELMT.ALL, since the algorithm for calculating the polygons is based on the existence of dead-ends. Nevertheless, the user may choose whether the output polygons and proximity functions should include dead-end fractures, depending on what is to be simulated in TRIPOLY. In many cases, the impact of dead-end fractures on solute transport in fractured porous formations can be neglected. The structure of NODE.ALL and ELMT.ALL is the same as for the TRINET-input data sets NODE.INP and ELMT.INP which are described in Section 2.5.3.

  Besides NODE.ALL and ELMT.ALL, POLY needs the input file PROX.INP. This data set contains a number of control variables for the generation of polygons and proximity functions. Moreover, initial values for the porous block material parameters are provided. They are written as spatially constant 'default'- values into the output data set. The user may eventually change the output data set and provide different values for each matrix block when a heterogeneous system is to be modeled.

- **Output**

  POLY writes two ASCII- output files which can be used directly as input for TRIPOLY. POLY.INP contains the nodal information for each polygon, POLYMAT.INP contains material parameters and the coefficients of the proximity polynomials. Both data sets, variables and formats are described in Chapter 2.5.3.

  Two more data sets, GRAFICS.SCI and GRAFICS.SCR (unformatted), are generated which can be used for the graphical output of matrix simulation results in TRIPOLY. They are **not** needed for the simulation process itself.

Table 2.5 provides a list of all input and output files for POLY. Bold input file names indicate that the data set is mandatory. Table 2.6 lists the variables of PROX.INP, which are described in Table 2.7.

Table 2.5 POLY - Input/Output Files

Name	Input/Output	Unit	Description
PROX.INP	Input	1	General information, control variables
NODE.ALL	Input	1	Node information for network with dead-ends
ELMT.ALL	Input	2	Element information for network with dead-ends
POLY.INP	Output	3	Nodal information for polygons
POLYMAT.INP	Output	10	Material parameters for polygons
GRAFICS.SCI GRAFICS.SCR	Output *(unformatted)*	1	Geometry of matrix blocks needed for visualization

Table 2.6 POLY - Input Variables in PROX.INP

Variable	Format	Input Unit
TEXT	a80	1
TEXT	a80	1
TEXT	a80	1
INET, IRANF, NMAX, DSEED	5x,3(i5,10x),f10.0	1
NDISC, NGRAD	5x,2(i5,10x)	1
(LDISC(M),M=1,NDISC)	10i5	1
SSP, XKM	2(10x,f10.0)	1
XPOR, XDIF, XRET	3(10x,f10.0)	1

Table 2.7 POLY - Description of Input Variables in PROX.INP

Variable/Array	Description
INET	used to control the type of network considered 0 polygons for network without dead-end fractures 1 polygons for network including dead-ends
IRANF	used to control the random number generation 0 random choice of start-up value for generation (actual time used as start-up value) 1 read start-up value DSEED to duplicate previous run
NMAX	number of random points for calculating proximity function
DSEED	start-up value for random number generation
NDISC	number of elements for 1D-FE discretization of porous matrix blocks (must be smaller than 15)
NGRAD	grade of polynomial for approximation of proximity function (must be smaller than 10)
LDISC(M)	relative values for discretization of matrix blocks
SSP	specific storage of matrix blocks
XKM	hydraulic conductivity of matrix blocks
XPOR	porosity of matrix blocks
XDIF	molecular diffusion coefficient in matrix blocks
XRET	retardation factor in matrix blocks

Sample data set:

PROX.INP

```
TEST DATA SET
TEXT LINE 2
TEXT LINE 3
INET     0     IRANF    1     NMAX  2000 DSEED    1.0000D+03
NDISC    8     NGRAD    5
    1    2    5   10   20   50   70  100
SSP       0.0000E-00 XKM      0.0000E-00
XPOR      0.5000E-01 XDIF     0.4000E-09 XRET     1.0000E-00
```

2.5 TRIPOLY

TRIPOLY is based on the finite element code TRINET which features a mixed Lagrangian-Eulerian solution scheme with adaptive gridding. This procedure is particularly strong in very heterogeneous media such as fracture networks since it automatically takes care of mesh refinements at sharp fronts. TRINET has been changed in two major issues: First, modifications have been made to improve some of the old features of the code (e.g. to improve the code's speed and ability to solve large problems, to generate plot files for commercial visualization packages such as AVS and TECPLOT, to calculate tracer breakthrough curves, to allow for velocity-dependent dispersion etc.). Second, new subroutines and algorithms have been added to simulate fluid and solute exchange between the fractures and the porous matrix. A detailed Berkeley Lab report serves as user's manual and tutorial for the previous code TRINET (SEGAN & KARASAKI, 1993).

2.5.1 Changes Regarding Fracture Network Modeling

- Different criteria have been introduced to decide whether a new grid point has to be introduced while forward tracking. These criteria may be either the absolute difference between the two adjacent nodes, the relative difference or the concentration gradient. They are applied in subroutine *ftrack*. The input variable LOGDCO determines the criterion to be used in TRIPOLY.

- In the original TRINET version, a dispersion coefficient is given for each element which accounts for molecular diffusion **and** mechanical dispersion within the fractures. However, as this parameter has to be set prior to the simulation, it cannot account for changing dispersivities due to velocity changes in transient flow fields. In TRIPOLY, molecular diffusion and mechanical dispersion may be treated as separate processes, and the dispersion coefficient is calculated by

$$D^F = \alpha_l \cdot v + D_M^F \qquad (12)$$

  where α_l is the longitudinal dispersivity, D_M^F is the effective molecular diffusion coefficient in the fracture and v is the velocity along the fracture axis. The input parameter LOGDIS specifies if TRIPOLY uses a constant lumped dispersion coefficient or applies the above mentioned equation.

- TRIPOLY generates new output files for the graphical interpretation of the results. Nodal values of hydraulic head or concentration may be written into plotfiles with a file format used by the commercial visualization packages TECPLOT and AVS. Furthermore, tracer breakthrough curves for specified model boundary sides are calculated and written into data set BREAK.TEC. The user can define in subroutine *break* which boundary side should be considered for calculating the tracer breakthrough.

In the original TRINET code, new grid nodes are only introduced due to the numerical requirements of the advective transport. The concentration profile is tracked forward, and additional nodes are introduced when needed. However, in some fractures the flow velocity may be very small, or even zero, as for example in dead-end fractures or dead-end clusters. In such cases, no additional nodes would be introduced and dispersion (molecular diffusion and mechanical dispersion) basically determines the spreading of a solute. However, if the dispersive transport length in a given time step is much smaller than the distance between two adjacent nodes, strong oscillations may occur while solving the dispersion equation. To avoid these difficulties, an additional adaptive gridding procedure is applied in TRIPOLY due to the requirements of diffusive transport. Two alternative methods may be chosen:

1) The first method checks for elements with low velocities and divides those elements into smaller subelements, introducing additional nodes. The length of the subelements is chosen according to the requirements of the dispersive transport. The procedure is performed only once in the first time step, after solving the flow field, before solving the transport field. Additional nodes are treated as fixed nodes, they are not removed from the mesh. The algorithm is very stable, and minimizes numerical dispersion. Note that the effects of transient flow fields are neglected in this procedure since TRIPOLY generally uses the flow field of the first time step to design the new gridding. In some specific cases, parameters controlling the procedure of adding subelements may have to be changed by the user, e.g. when too many new nodes and elements are generated, and the computation becomes very inefficient. These parameters can be set in subroutine *addif1*.

2) The second method introduces new nodes or elements only in the vicinity of steep concentration gradients. In each time step, the diffusive transport length is calculated and compared to the element length. If it is much smaller, different criteria are applied to decide whether a new grid point has to be introduced (subroutine *addif2*). These criteria may be either absolute or relative concentration difference or the concentration gradient. When the sharp front has passed through the area, the additional nodes are removed. Usually, the number of nodes required is much smaller than with the first procedure. However, test simulations have shown that the second method is very sensitive to the control parameters assigned in CTRL.INP. If they are not chosen very carefully, strong artificial dispersion may occur. Generally, it is recommended to use the first method. Only if the computing effort exceeds practical limits, the second method should be tried. However, results have to be carefully checked.

The dispersive transport length (dl), which is needed for both methods, is estimated by the following equation

$$dl = \sqrt{2\ D^F\ dt}\ , \qquad (13)$$

where D^F is the dispersion coefficient in the fractures and dt is the time step size. In method 1), the element is divided into a number of subelements which all have a length close to the dispersive transport length. In method 2), only one grid node is introduced at the distance dl, measured from the upstream node. A user-defined variable (LOGDIF) determines if adaptive gridding for dispersion should be applied and, if so, which method is to be used.

- The performance of TRINET is sensitive to the time step size. If time steps are very small, an unnecessarily high number is needed to move a concentration front throughout the area. However, in the opposite case, another problem arises: During large time steps, particles are advected along various fracture intersections. Since nodal concentrations are advected rather than single particles, extensive branching takes place. With regard to forward tracking, a large number of new nodes are generated and the computation becomes very inefficient. With regard to backward tracking, the algorithm of tracing back the path becomes very complicated and time consuming.

  To avoid this, TRIPOLY allows for an automatic time step control. Usually a computation starts with a reasonably small time step which may be increased during the simulation by the factor PRR. However, if the number of branches compared to the number of (advected) nodes exceeds a certain value, the new time step is either kept constant or, if it happens more than five consecutive time steps, decreased by the factor PRR. As to forward tracking, the user may specify the maximum relative value of branches allowed with the variable DBRAN in file TRINET.INP. For the backward tracking procedure, a fixed value is set in subroutine *advec*; default value is 5.

2.5.2 Changes Regarding Fracture-Matrix Interaction

Several subroutines have been added to the original code TRINET to implement the fracture-matrix interaction modules. Also, a number of existing subroutines had to be significantly changed to account for the new procedures. However, the general structure of the code has not changed. The code consists of three major sections which are called by the main program *tripoly.f*. These sections are the input section, the flow section and the transport section.

- **Input Section**

  The input section has been extended to read the matrix block input files. Subroutine *iidopo* reads the nodal information for each polygon (POLY.INP), *irdopo* does the same for the unformatted data set of a previous simulation (POLY.RES). Then subroutine *rdopo* reads the POLYMAT.INP file (or HDOPO.RES, CDOPO.RES for previous simulation runs) and calculates the element matrices for each polygon as far as possible at this stage. All these subroutines are called from subroutine *rdata*. The structure of the input part of the code is (new subroutines in bold):

```
openfile
rdata          rcntrl
               rnode
               relem
               rnpn
               iidopo/irdopo
               rdopo          vprdma
                              koefdp
               renumf
linkem
renum
```

- **Flow Section**

TRIPOLY features a direct solution scheme for the coupled fracture-matrix problem. The first step is to assemble the global equation system for the fracture domain in the same way as done in the previous code TRINET. The left- and the right-hand sides of the equation system are determined according to the GALERKIN finite element scheme (subroutine *assem* and subroutine *bc*). Note, that the equation system is now ready to be solved, assuming the fracture-matrix interaction would be neglected.

Subroutine *vpaut* accounts for the fluid exchange between fractures and matrix. It solves the one-dimensional flow equation for all node-polygon connections according to equation (7). The result is a linear expression for the fluid exchange at each node-polygon interface (equation (8)). These linear expressions, related to the nodes of the fracture finite element mesh, are added to the global equation system of the fracture domain. Terms with the unknown fracture head h_J are added to the diagonal of the left-hand matrix, known quantities are added to the right-hand vector. Then, the resulting equation system can directly be solved for the current hydraulic head in the fracture domain. Finally, the matrix head distributions are determined by performing a backward substitution according to equation (9). This is done in subroutine *vpaum*.

The structure of the flow part of the code is (new subroutines in bold):

flow	store	
	assem	clear1
		koeff
		rhand
	vpaut	**jpgl01**
		jpgl02
	bc	
	solve	
	store	
	vpaum	**jpgl01**
		jpgl03

- **Transport Section**

In the same manner as for the flow problem, TRIPOLY features a direct solution scheme for the coupled fracture-matrix equation system. However, due to the Lagrangian-Eulerian scheme, the procedure becomes more complicated. Extensive bookkeeping is needed to take care of adding and eliminating new nodes and node-polygon connections, respectively.

The transport part has two major sections. One is the advection calculation and the other is the numerical solution of the dispersion part. As the fracture-matrix interaction term is introduced in the dispersion equation, the advection part is similar to the previous code TRINET, except for several bookkeeping procedures. Whenever a new fracture node is generated in subroutine *track*, a new node-polygon connection has to be established. First, the polygons associated with the new node have to be determined,

then matrix concentration profiles have to be assigned by interpolation from the adjacent polygon nodes. This is done within the subroutines *add* and *addopo*.

When only flow and transport in the fractures is considered and matrix diffusion is neglected subroutine *pluck* is called immediately after finishing the advection part, before starting with the dispersion/diffusion calculation. Subroutine *pluck* removes nodes which are no longer near the concentration front and which were not part of the original fixed mesh. Thus, the dispersion/diffusion part is performed with fewer nodes and becomes more efficient. However, our test results indicate that this procedure is **not** recommended when matrix diffusion is taken into account. In such case it is better to solve the dispersion/diffusion part first and then eliminate unnecessary nodes within subroutine *pluck*. This is less efficient, but helps to minimize numerical dispersion. Together with subroutine *pluck* (which eliminates fracture nodes) goes subroutine *dedopo* which eliminates the associated node-polygon connections and the associated matrix concentration profiles.

The solution of the dispersion part is very similar to the flow logic. The first step is to assemble the global equation system for the fracture domain in the same way as done in the previous code TRINET, according to the GALERKIN finite element scheme (subroutine *assem* and subroutine *bc*). However, as already mentioned in chapter 2.5.1, an adaptive gridding technique was introduced for the dispersive part. It is performed in subroutine *addif1* (LOGDIF = -1) or *addif2* (LOGDIF > 0), and the procedure of introducing or eliminating dispersive nodes requires essentially the same bookkeeping as the advective part.

Subroutine *vpaut* accounts for the solute exchange between fractures and matrix. After solving the one-dimensional diffusion equation for all node-polygons connections, a linear expression for the solute exchange at each node-polygon interface is obtained (equation (8)). These expressions are added to the right and left hand side of the global equation system of the fracture domain. Again, the resulting equation system for the current concentrations of the fracture domain can directly be solved. Finally, the matrix concentration distributions are determined by performing a backward substitution in subroutine *vpaum*.

The structure of the transport part of the code is (new subroutines in bold):

trans
- advec
  - btrack
    - blocate
      - piggybk
        - addnod
    - ftrack
      - flocate
        - addnod
      - btrack
    - plant
      - seed
    - pluck *(only if matrix diffusion is neglected)*
    - renum
    - **add**
      - **addopo**
  - diffu
    - **addif1**
      - **add**
        - **addopo**
    - **addif2**
      - **add**
        - **addopo**
    - assem
      - clear1
      - koeff
      - rhand
    - **vpaut**
      - **jpgl01**
      - **jpgl02**
    - bc
    - solve
    - store
    - **pluck**
    - **renum**
      - **dedopo**
    - **vpaum**
      - **jpgl01**
      - **jpgl03**

2.5.3 Input Files

TRIPOLY reads a number of input data sets. The first three data sets CTRL.INP, NODE.INP and ELMT.INP are mandatory and have to be supplied for any simulation of TRIPOLY, other data sets like POLY.INP, POLYMAT.INP or NPN.INP are only needed for special problem types or output procedures (see Table 2.8).

CTRL.INP is used to control the type of simulation. It provides a number of logical parameters and constants, listed and described in Table 2.9 and 2.10. Since many changes have been introduced in TRIPOLY compared to the previous code TRINET, the new input data set CTRL.INP for TRIPOLY is significantly different from the previous one for TRINET.

The data sets NODE.INP and ELMT.INP determine the structure of the finite element mesh for the fracture network. Both data sets are provided by the line network optimizer RENUMN. However, depending on the type of fracture mesh to be modeled (dead-ends included or dead-ends discarded), the user has to rename the extensions of the RENUMN output data sets, either xxxx.ALL or xxxx.TRUNC, into xxxx.INP. See Table 2.11 and 2.12 for details concerning variables and formats.

The data sets POLY.INP and POLYMAT.INP are **only** needed when flow or solute exchange between fractures and the porous matrix is taken into account (IDOPO = 1). POLY.INP contains the nodal information for each polygon, POLYMAT.INP contains material parameters and the coefficients of the proximity polynomials. GRAFICS.SCI and GRAFICS.SCR serve as input for the graphical interpretation of matrix simulation results, they are **not** needed for the simulation process itself. All these data sets are generated by the preprocessing code POLY; the user does not need to write them (see Table 2.13, 2.14, 2.15 and 2.16).

POLY.INP consists of two parts. The first part lists the node numbers of the different polygons. In that list adjacent nodes follow one another. The node numbers are the old numbers of FMGN before renumbering. In TRIPOLY the numbering is changed. Usually the polygons are closed, unless node numbers are negative. This might be the case when a matrix block is located at the boundary of the model area, and only a part of it is described by the polygon. The negative node number denotes the first node of the connected part of the polygon. The second part of POLY.INP contains an element list determining which polygons are related to each element. Usually two polygons are associated with each fracture segment (for each fracture wall).

The second data set POLYMAT.INP starts with the relative discretization of the porous matrix. The relative discretization is the same for all polygons. Within TRIPOLY, the NDISC relative values are multiplied with the maximum orthogonal distance S of each matrix block. The second part of POLYMAT.INP contains the material parameters for each block, such as the maximum orthogonal distance S, the storage coefficient, permeability, porosity, molecular diffusion coefficient, retardation factor and the NGRAD coefficients of the proximity function.

Another data set NPN.INP allows the user to provide a list of nodes. If NPN.INP exists, TRIPOLY generates output files containing head or concentration values versus time for the nodes listed. NPN.OUT comprises the head or concentration values of each time step for the

specified nodes itself, i.e. it represents the fracture properties. Another data set DOPO.OUT contains the head or concentration profile in all matrix blocks adjacent to the specified nodes (only if IDOPO = 1). However, only certain time steps are written into DOPO.OUT to avoid excessively large data sets (output time steps are given in CTRL.INP).

A group of input files with the extension xxx.RES is very useful for very time-consuming simulations (e.g. CTRL.RES, NODE.RES, ELMT.RES, CONC.RES, POLY.RES, CDOPO.RES etc.). These data sets contain information regarding the last time step of a previous simulation run. If they exist in the directory from which the code is run, TRIPOLY assumes that this previous simulation shall be continued, and reads these files as initial condition for subsequent simulations. Most of them are unformatted. Only CTRL.RES is formatted, so that changes can be made. Note that the xxx.RES files are only generated by simulations for which the maximum simulation time allowed (TMAX) is not reached.

Table 2.8 TRIPOLY - Input Files

Name	(un)formatted	Unit	Description
CTRL.INP	formatted	1	General information, control variables
NODE.INP	formatted	2	Node information for fracture mesh
ELMT.INP	formatted	3	Element information for fracture mesh
POLY.INP	formatted	50	Nodal information for polygons
POLYMAT.INP	formatted	50	Material parameters for polygons
GRAFICS.SCI GRAFICS.SCR	unformatted	81	Geometry of matrix blocks needed for visualization
NPN.INP	formatted	55	Node list for time series output
CTRL.RES	formatted	1	Information needed to continue a previous simulation run
xxx.RES	unformatted	...	

Table 2.9 TRIPOLY - Input Variables in CTRL.INP

Variable	Format	Input Unit
TITLE	3(a80/)	1
IMODE, IBMODE, MXSTEP, IPFREQ,	5(10x,i5)	1
IDOPO, LOGDCO, LOGDIF, LOGDIS, NSTEP0	4(10x,i5)	1
TIME0, TMAX, DTINI, PRR	4(10x,f10.0)	1
THETA, TOLE, DCINT, DCON	4(10x,f10.0)	1
DCOFF, RHOR, RMUR, GRAVR	4(10x,f10.0)	1
DEPS, DCDIF, DBRAN, DIFMOL *only for IPFREQ < 0*	4(10x,f10.0)	1
DUMMY TEXT	a80	1
(TIMPR(N), N=1,ABS(IPFREQ))	8f10.0	1

Table 2.10 TRIPOLY - Description of Input Variables in CTRL.INP

Variable/Array	Description
DBRAN	maximum number of branches compared to the number of advected nodes: if DBRAN is exceeded, the time step size is adjusted (used for the forward tracking procedure)
DCINT	used for choice of interpolation scheme: if the concentration difference between two adjacent nodes is less than DCINT, an upwind-scheme is performed
DCOFF	used for removing nodes: if the concentration difference between two adjacent nodes is less than DCOFF, and one of the nodes is a moving node, it is removed by subroutine *pluck* (**only** applied for advective moving nodes)
DCDIF	used for inserting nodes: if diffusive transport length is smaller than the element length and the concentration difference between the two element nodes is larger than DCDIF, a moving node is originated in subroutine *addif2* (only applied for diffusive moving nodes)
DCON	used for inserting nodes: if the concentration difference between two adjacent nodes is larger than DCON, a moving node is originated in subroutine *ftrack* (only applied for advective moving nodes)
DEPS	used for inserting nodes: if concentration is smaller than DEPS, no test criteria are applied
DIFMOL	molecular diffusion coefficient in fractures (only used for LOGDIS = 1)
DTINI	initial time step increment
GRAVR	gravitational constant
IDOPO	used to control type of simulation: 0 porous matrix is neglected 1 porous matrix is considered in simulation (fluid exchange for IMODE < 0, solute exchange for IMODE > 0)
IBMODE	used to control the renumbering procedure in subroutine *renum*: 0 renumbering starts from radial boundary 1 renumbering starts from side 2 of square boundary 2 renumbering starts from all sides of square boundary
IMODE	used to control type of simulation: -2 steady-state flow only (theta=1.0) -1 transient flow only 0 test data check 1 advection-dispersion in transient flow field 2 advection-dispersion in steady-state flow field (theta=1.0) 3 advection only in steady-state flow field (theta=1.0)
IPFREQ	used to determine output data sets for contouring: >0 each IPFREQ time step is written on tecplot-file <0 a number of abs(IPFREQ) given time steps is written on tecplot-file (time steps are assigned at the end of CTRL.INP)
LOGDCO	used to determine the criterion for inserting and removing advective nodes: (**only** applied for advective moving nodes) 1 absolute concentration difference (C_1-C_2) 2 relative concentration difference (C_1-C_2)/ C_1 3 concentration gradient (C_1-C_2)/ DL

➡ continued on next page

Variable/Array	Description
LOGDIF	used to determine the criterion for inserting and removing dispersive nodes: (**only** applied for dispersive moving nodes) -1 mesh refinement in zero-flow elements **(numerically stable procedure)** 0 no diffusive moving nodes added > 0 mesh refinement only adjacent to steep concentration gradients **(be aware of possible numerical dispersion !!)** 1 absolute concentration difference (C_1-C_2) 2 relative concentration difference (C_1-C_2)/ C_1 3 concentration gradient (C_1-C_2)/ DL
LOGDIS	used to determine the type of dispersion model used in TRIPOLY: 1 constant lumped dispersion coefficient for each element (coefficient is provided as element parameter in ELMT.INP) 2 dispersion coefficient is the sum of mechanical dispersion and molecular diffusion (the longitudinal dispersivity is provided as element parameter in ELMT.INP; the molecular diffusion coefficient is provided in CTRL.INP)
MXSTEP	maximum time step desired in the simulation
NSTEP0	time step number of start of simulation
PRR	factor for increasing the time step. Should be greater than or equal to 1.
RHOR	density of fluid
RMUR	dynamic viscosity of fluid
THETA	time weighting constant
TIME0	time of start of simulation
TIMPR(I)	output time steps given in real time (only needed for IPFREQ < 0)
TITLE	title of simulation run
TMAX	maximum simulation time allowed (in real time given in seconds)
TOLE	tolerance to allow for numerical round-off: e.g. if distance between two nodes is less than TOLE, they are merged to one node

Sample data set:

CTRL.INP

```
TEST DATA SET
TEXT LINE 2
TEXT LINE 3
IMODE    =    2 IBMODE  =    2 MXSTEP  =  450 IPFREQ  =   -7 NSTEP0  =    1
IDOPO    =    1 LOGDCO  =    2 LOGDIF  =    3 LOGDIS  =    2
TIME0    =0.0000E+00 TMAX    =4.0001E+06 DTINI   =5.0000E+02 PRR     =1.1000E+00
THETA    =1.0000E-00 TOLE    =1.0000E-02 DCINT   =2.0000E-01 DCON    =1.0000E-02
DCOFF    =1.0000E-03 RHOR    =1.0000E+03 RMUR    =1.0000E-03 GRAVR   =9.8067E+00
DEPS     =1.0000E-03 DCDIF   =1.0000E-02 DBRAN   =3.0000E+00 DIFMOL  =2.0000E-07
OUTPUT TIME STEPS
   2.0e+05   5.0e+05   1.0e+06   1.5e+06   2.0e+06   3.0e+06   4.0e+06
```

Table 2.11 TRIPOLY - Input Variables in NODE.INP, ELMT.INP and NPN.INP

NODE.INP:

Variable	Format	Input Unit
NN	10x,i5	2
for I=1,NN (IOLD, ISC(I), IB(I), IBC(I), (XYZ(J,I),*J=1,3*), H(I), COLD(I), BVALUE(I), CBVALUE(I))	i5,i3,2i2,7f12.0	2

ELMT.INP:

Variable	Format	Input Unit
NE	10x,i5	3
for I=1,NE (IOLD, ICAT(J,I),*J=1,2*), TRANSM(I), W(I), RL(I) SS(I), DC(I))	3i5,5f12.0	3

NPN.INP (not mandatory):

Variable	Format	Input Unit
listp	i5	5

Table 2.12 TRIPOLY - Description of Input Variables in NODE.INP, ELMT.INP and NPN.INP

Variable/Array	Description
BVALU	value of flow boundary condition assigned at the node; for IB = 1 it is the head value and for IB = -1 it is the flux value
CBVALU	value of solute transport boundary condition assigned at the node; for IBC = 1 it is the concentration value and for IBC = -1 it is the flux value
COLD	initial condition of concentration assigned at the node
DC	dispersion parameter of element. If LOGDIS = 1, DC is assumed to be the lumped dispersion coefficient of the element. For LOGDIS = 2, DC is the longitudinal dispersivity.
H	initial condition of hydraulic head assigned at the node
IB	specifies type of boundary condition for flow at the node 1 DIRICHLET condition, i.e. fixed head 0 internal node, no condition specified -1 NEUMANN condition, i.e. fixed flux
IBC	specifies type of boundary condition for solute transport at the node 1 DIRICHLET condition, i.e. fixed concentration 0 internal node, no condition specified -1 NEUMANN condition, i.e. fixed flux
ICAT(J,I)	ICAT(1,I) and ICAT(2,I) are the two end points that comprise the i-th line element

➡ continued on next page

Variable/Array	Description
ISC	specifies which boundary side the node is on. Negative ISC indicates that node is on side abs(ISC) and requests printout at every time step for the nodes in output data set FLOW.NOD. Flux is summed up among all the nodes that belong to the same boundary and printed out in file FLOW.OUT.
IOLD	old node or element number before renumbering in RENUMN. Needed to relate numbers of files xxxx.TRUNC and xxxx.ALL, respectively.
LISTP	list of nodes for output in NPN.OUT and DOPO.OUT, respectively
NE	total number of elements
NN	total number of nodes
RL	length of element. To prevent round-off errors, it is not used by code directly. It is present here only for interpretative purposes.
SS	specific storage of element
TRANSM	transmissivity of element. This value is only used when one or all of the parameters grav, rhor, rmur in CTRL.INP has a non-physical value (i.e. lower than or equal to zero). Otherwise the cubic law is applied to calculate transmissivities from the hydraulic aperture.
W	hydraulic aperture of element. If reasonable values for grav, rhor and rmur are given in CTRL.INP, the aperture values are used for calculating transmissivities.
XYZ(J,I)	XYZ(1,I), XYZ(2,I), XYZ(3,I) are x, y, z coordinates of node i.

Sample data sets:

NODE.INP

```
NN       = 3504
.....
  274  1 0 0     -4.6878     -5.0000      0.0000      0.0000      0.0000
  293  1 0 0     -4.4829     -5.0000      0.0000      0.0000      0.0000
  203  1 0 0     -4.1275     -5.0000      0.0000      0.0000      0.0000
  337  1 0 0     -3.8805     -5.0000      0.0000      0.0000      0.0000
.....
```

(boundary-values are not listed, since IB(I) and IBC(I) equal zero)

ELMT.INP

```
NE       = 5685
.....
  235    1   32  4.1840E-07  7.9999E-05  5.1580E-02  0.0000E+00  5.0000E-02
  251    2   33  4.1840E-07  7.9999E-05  5.1580E-02  0.0000E+00  5.0000E-02
  173    3   34  4.1840E-07  7.9999E-05  5.1580E-02  0.0000E+00  5.0000E-02
  290    4   35  4.1840E-07  7.9999E-05  2.2980E-01  0.0000E+00  5.0000E-02
.....
```

Table 2.13 TRIPOLY - INPUT Variables in POLY.INP

Variable Unit	Format	Output
TEXT	a80	50
NMATK	i5	50
for M=1,NMATK		
IDUM, NM	2i5	50
(IN(IKNA+N-1), N=1,NM)	10i5	50
TEXT	a80	50
NE	i5	50
for M=1,NE		
IALT, IEPOL(1,M),IEPOL(2,M)	3i5	50

Table 2.14 TRIPOLY - Description of INPUT Variables in POLY.INP

Variable/Array	Description
IALT	old element numbers for relating data sets with and without dead-ends
IEPOL(1,M)	first matrix block (=polygon) related to element M
IEPOL(2,M)	second matrix block (=polygon) related to element M
IDUM	number of polygon (dummy variable)
IKNA	index for position in field IN(....)
IN(IKNA+....)	list of nodes for polygons
NE	total number of elements
NM	number of nodes defining polygon
NMATK	total number of matrix blocks (polygons) in model area

Sample data set:

POLY.INP

```
POLYGON NODES
 2313
    1    5
 3099 3112 3113 3100 2535
    2   14
 1022 1023 1024 1025 1565  579  580  581  582 -703
  702 1048 1030 1031
    3    5
   62 1048  702  703  -63
.....
LIST OF POLYGONS AT ELEMENT (OLD NUMBERING SCHEME)
 5685
  235    1 1564
  251    1    2
.....
```

Table 2.15 TRIPOLY - INPUT Variables in POLYMAT.INP

Variable	Format	Input Unit
TEXT	a80	50
TEXT	a80	50
NDISC	i5	50
(RDISC(N),N=1,NDISC)	6e13.6	50
TEXT	a80	50
NMATK	i5	50
for M=1,NMATK		
IDUM, NPROX(M), SMAX(M)	2i5,e13.6	50
(R(IRF1+N-1), N=1,5)	5e13.6	50
(R(IRF2+N-1), N=1,NPROX(M))	6e13.6	50

Table 2.16 TRIPOLY - Description of INPUT Variables in POLYMAT.INP

Variable/Array	Description
IDUM	number of polygon (dummy variable)
IRF1, IRF2	index for position in field R(....)
NDISC	number of elements for 1D-FE discretization of porous matrix blocks (must be smaller than 15)
NPROX(M)	stores the number of coefficients of proximity function for each element
RDISC(M)	stores relative values for discretization of matrix blocks
R(IRF1+....)	stores the material parameters for each element (specific storage, hydraulic conductivity, porosity, molecular diffusion coefficient, retardation coefficient)
R(IRF2+....)	stores the coefficients of proximity function for each element (function is defined as follows: $\Pr ox(s) = a_1 s + a_2 s^2 + ... + a_{nprox} s^{nprox}$)
SMAX(M)	stores the maximum orthogonal distance of points in the matrix blocks to the adjacent fractures

Sample data set:

POLYMAT.INP

```
MATERIAL PARAMETERS FOR DISCRETE MATRIX BLOCKS
RELATIVE DISCRETIZATION
    8
 0.387597E-02 0.116279E-01 0.310078E-01 0.697674E-01 0.147287E+00 0.341085E+00
 0.612403E+00 0.100000E+01
POLYGON PARAMETERS
 2313
    1    5 0.257955E-01
 0.000000E+00 0.000000E+00 0.500000E-01 0.400000E-09 0.100000E+01
 0.134899E+01-0.111168E+01 0.230691E+01-0.253969E+01 0.996822E+00
    2    5 0.257917E-01
 0.000000E+00 0.000000E+00 0.500000E-01 0.400000E-09 0.100000E+01
 0.987857E+00 0.914764E+00-0.210270E+01 0.150887E+01-0.304360E+00
.....
```

2.5.4 Output Files

Primary output from TRIPOLY are the values of head and/or concentrations over the finite element nodes of the entire fracture network. For transient simulation runs, only specified time steps are written. The user can determine in CTRL.INP, if either each IPFREQ time step is written or if output is prepared for certain predefined simulation times. As default, the output files are written in a format used by the commercial visualization package TECPLOT; they are named DATnn.TEC, with nn being the number of the time step. Instead of the TECPLOT output format, the user can choose to write files in a format for the visualization software AVS; then the files are named RAVSnn.INP. Minor changes have to be made in subroutine *prout* to write AVS-files instead of TECPLOT files. Note that the node numbering is different from time step to time step, due to the adaptive gridding and the renumbering associated with bandwidth optimization.

If flow or solute exchange between fractures and matrix is considered in a simulation run (i.e. IDOPO=1) and the graphic input files GRAFICS.SCR and GRAFICS.SCI exist, additional information is written into DATnn.TEC. Average head or concentration values are calculated for each matrix block, and two zones are generated for TECPLOT: one for the matrix blocks, one for the fracture network.

Regarding the flow field, two data sets are written giving flow rates in volume/time. FLOW.OUT gives the total amount of fluid entering or leaving each side of the flow region. The other file FLOW.NOD does essentially the same thing, however, flow rates are printed for each node on the model sides separately rather than calculating a sum over each side.

Tracer breakthrough curves are calculated within TRIPOLY and written to the file BREAK.TEC, assuming that a transport simulation is performed. As default, the breakthrough is calculated for tracer exiting at side 4 of the model region. If other sides should be considered or only parts of sides, the user must make minor changes in subroutine *break*.

If input file NPN.INP exists, two more output files are generated: NPN.OUT and DOPO.OUT (the latter only if IDOPO=1). The NPN.OUT file contains a list of nodal head or concentration values versus time for the fracture network nodes assigned in NPN.INP. This is done for each times step calculated. DOPO.OUT comprises the head or concentration profiles in all matrix blocks connected to the specified nodes. This is only done for the time steps assigned in CTRL.INP; otherwise DOPO.OUT becomes to large for long-term simulations.

As already mentioned, a number of (mostly unformatted) files are written for the last time step calculated, when the maximum simulation time given in CTRL.INP has not been reached yet. This allows for a subsequent continuation (restarting) of simulation runs; the xxx.RES files serve as initial condition for the new simulation run.

Finally, file SOLVER.STAT records information regarding the performance of the conjugate gradient solver. For each time step, the number of iterations and the mean residual error is output.

With respect to output in general, it is felt that consistent use of the code will allow the user to become comfortable enough to manipulate the code and output subroutines to suit his own needs.

Table 2.17 provides a list of all output files generated by TRIPOLY.

Table 2.17 TRIPOLY - Output Files

Name	fracture/matrix	Unit	Description
DAT**nn**.TEC	fracture/matrix	69	List of nodal heads and concentrations for time step **nn** **TECPLOT-format** (as default)
RAVS**nn**.INP	fracture	69	List of nodal heads and concentrations for time step **nn** **AVS-format** (default: no avs-output)
BREAK.TEC	fracture	50	Tracer breakthrough for given side of model area
NPN.OUT	fracture	7	Time series of head and concentration for given nodes
DOPO.OUT	matrix	89	Head or concentration profiles for given time steps and given nodes
FLOW.OUT	fracture	9	Total flowrate over each model side
FLOW.NOD	fracture	19	Flowrate for nodes located on model sides
SOLVER.STAT		10	Convergence and accuracy of conjugate gradient solver
xxx.RES	fracture/matrix	....	Information regarding last time step of simulation run

3 Sample Problems

3.1 Introduction

In this section we present two sample problems. The first problem is very simple, it essentially serves as a tutorial to illustrate the use and performance of TRIPOLY, the preparation of input decks and the interpretation of computed output. Since the problem comprises only one fracture and the adjacent porous matrix blocks, the TRIPOLY input files can be generated by hand; no FMGN, RENUMN or POLY runs are needed. All input files are presented in detail, and the code TRIPOLY is verified in comparison to an analytical solution.

The second problem demonstrates TRIPOLY's ability to solve more complex flow and transport problems. It also shows the use of the codes FMGN, RENUMN and POLY, which provide the input decks for TRIPOLY when complicated problems are considered. A random fracture network is generated with FMGN and renumbered with RENUMN. The porous matrix information is computed by the code POLY. Then, a number of simulation runs are performed with TRIPOLY, illustrating the strong impact of matrix diffusion on tracer transport in fractured porous formations. Some input files are presented in detail in this section. However, long input and output files are not listed, rather they are presented in the form of figures and graphs.

3.2 Example 1: Longitudinal Transport in a Single Fracture with Transverse Matrix Diffusion

This problem concerns longitudinal transport along a single fracture and transverse diffusion into the adjacent matrix blocks. An analytical solution was developed by TANG et al. in 1981. Figure 8 schematically illustrates the problem. A contaminant source with C = 1.0 is located in the fracture on the left boundary of the model area at x=0. The fracture aperture is 10^{-4} m. The solute is transported in the fracture by advection and dispersion, with a flow velocity in the fracture of 0.01 m/d and a longitudinal dispersivity along the fracture axis of 0.5 m. Molecular diffusion in the fracture is chosen to 1.382 10^{-4} m^2/d. During the relatively fast transport in the fracture, part of the solute diffuses in a slow process into the adjacent porous matrix. Matrix parameters are 0.01 for the porosity and 1.382 10^{-5} m^2/d for the effective molecular diffusion coefficient. It is assumed that the system shown in Figure 8 is part of a fractured porous formation comprising parallel fractures with a separation distance of 2.4 m. Thus, the porous matrix blocks have an infinite length and their width is 2.4 m.

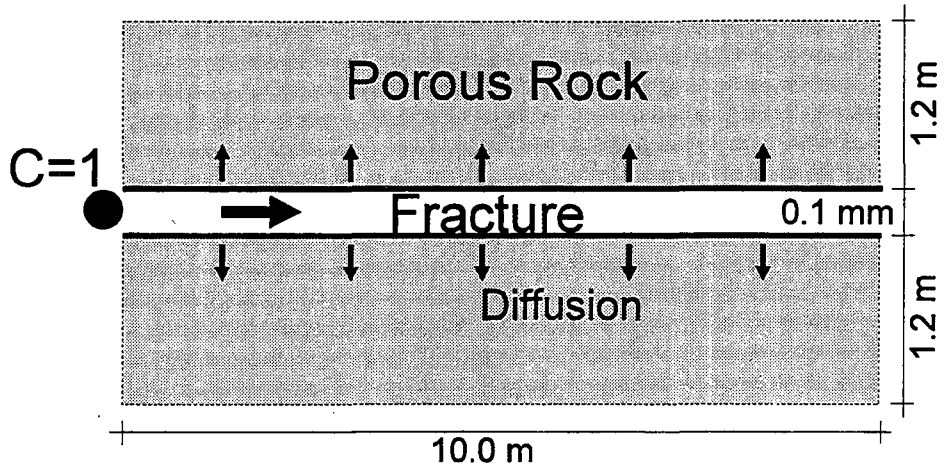


Fig. 8: Schematic of example 1

In the following section all input data sets are listed and briefly described. The first input data set is CTRL.INP, used to control the type of simulation. A maximum number of 200 time steps is calculated with an initial time step size of 20 days. The time step size is not changed within the simulation run (PRR = 1.0). The maximum simulation time is 10000 days. Output shall be written at times 97 days, 995 days and 9991 days. A steady-state flow field is considered and matrix diffusion is taken into account.

CTRL.INP

```
TEST CASE 1: SINGLE FRACTURE WITH MATRIX BLOCKS ON EACH SIDE
ANALYTICAL SOLUTION
COARSE MESH WITH ONLY TEN FRACTURE ELEMENTS
IMODE    =    2 IBMODE  =    2 MXSTEP  =  500 IPFREQ  =   -3 NSTEP0  =    1
IDOPO    =    1 LOGDCO. =    2 LOGDIF  =   -1 LOGDIS  =    2
TIME0    =0.0000E+00 TMAX    =1.0000E+04 DTINI   =2.0000E+01 PRR     =1.0000E+00
THETA    =1.0000E-00 TOLE    =1.0000E-02 DCINT   =2.0000E-01 DCON    =1.0000E-02
DCOFF    =1.0000E-03 RHOR    =0.0000E+03 RMUR    =1.0000E-03 GRAVR   =9.8067E+00
DEPS     =1.0000E-03 DCDIF   =1.0000E-01 NB/NODE =3.0000E+00 DIFMOL  =1.3820E-04
OUTPUT TIME STEPS
  97.0E+00  9.95E+02 9.991E+03
```

The data files NODE.INP and ELMT.INP determine the structure of the finite element mesh of the fracture network, prescribe boundary conditions and give material properties. The fracture is discretized with 11 nodes and 10 elements of 1.0 m length. Due to the adaptive gridding, a refined discretization adjacent to the contaminant source is **not** necessary. TRIPOLY implicitly takes care of refining the mesh wherever it is needed. Two DIRICHLET-type boundary conditions are given at node one, a prescribed hydraulic head and a prescribed concentration. Another DIRICHLET-type boundary condition is given for the hydraulic head at outflow cross section (node 11). Note that the fluid density RHOR in CTRL.INP is set to zero. This means that the transmissivity value for the fracture elements is not calculated from the aperture given in ELMT.INP. Instead, TRIPOLY directly uses the transmissivity values given in ELMT.INP.

NODE.INP

```
NN        =   11
    1   1 1 1       0.0000       0.0000       0.0000       0.0000       0.0000       0.1000       1.0000
    2   0 0 0       1.0000       0.0000       0.0000       0.0000       0.0000
    3   0 0 0       2.0000       0.0000       0.0000       0.0000       0.0000
    4   0 0 0       3.0000       0.0000       0.0000       0.0000       0.0000
    5   0 0 0       4.0000       0.0000       0.0000       0.0000       0.0000
    6   0 0 0       5.0000       0.0000       0.0000       0.0000       0.0000
    7   0 0 0       6.0000       0.0000       0.0000       0.0000       0.0000
    8   0 0 0       7.0000       0.0000       0.0000       0.0000       0.0000
    9   0 0 0       8.0000       0.0000       0.0000       0.0000       0.0000
   10   0 0 0       9.0000       0.0000       0.0000       0.0000       0.0000
   11   4 1 0      10.0000       0.0000       0.0000       0.0000       0.0000       0.0000
```

ELMT.INP

```
NE        =   10
    1     1     2  1.0000E-04  1.0000E-04  1.0000E+00  0.0000E+00  5.0000E-01
    2     2     3  1.0000E-04  1.0000E-04  1.0000E+00  0.0000E+00  5.0000E-01
    3     3     4  1.0000E-04  1.0000E-04  1.0000E+00  0.0000E+00  5.0000E-01
    4     4     5  1.0000E-04  1.0000E-04  1.0000E+00  0.0000E+00  5.0000E-01
    5     5     6  1.0000E-04  1.0000E-04  1.0000E+00  0.0000E+00  5.0000E-01
    6     6     7  1.0000E-04  1.0000E-04  1.0000E+00  0.0000E+00  5.0000E-01
    7     7     8  1.0000E-04  1.0000E-04  1.0000E+00  0.0000E+00  5.0000E-01
    8     8     9  1.0000E-04  1.0000E-04  1.0000E+00  0.0000E+00  5.0000E-01
    9     9    10  1.0000E-04  1.0000E-04  1.0000E+00  0.0000E+00  5.0000E-01
   10    10    11  1.0000E-04  1.0000E-04  1.0000E+00  0.0000E+00  5.0000E-01
```

Two more data sets are needed to describe the geometry and the properties of the matrix blocks, POLY.INP and POLYMAT.INP. The first part of POLY.INP defines the shape of the matrix blocks by providing the surface polygons. For the problem considered, the surface polygons for the upper and lower block are identical; they comprise the 11 nodes of the finite element mesh. Since the polygons are not closed, the first node of the connected part of the polygon has a negative number (here: node 1). The polygon nodes have to be numbered subsequently; however, the list of polygon nodes can start with an arbitrary node number of the polygon. The second part of POLY.INP denotes the two polygons associated with each fracture element. Here, all fracture elements are connected to the same polygons.

POLY.INP

```
POLYGON NODES (OLD NUMBERING SCHEME)
    2
    1   11
   -1    2    3    4    5    6    7    8    9   10
   11
    2   11
    5    6    7    8    9   10   11   -1    2    3
    4
ELEM
   10
    1    1    2
    2    1    2
    3    1    2
    4    1    2
    5    1    2
    6    1    2
    7    1    2
    8    1    2
    9    1    2
   10    1    2
```

POLYMAT.INP gives the relative discretization of the matrix and the porous block material parameters. The matrix blocks are discretized with 8 elements. The proximity function is simply a linear function of s, due to the infinite length of the blocks. Thus, the interface function is a constant which is reasonable since the interface for diffusion into the porous block does not change with s. The upper limit of s is S = 1.2 m which is the half width of the blocks.

POLYMAT.INP

```
MATERIAL PARAMETERS FOR DISCRETE MATRIX BLOCKS
RELATIVE DISCRETIZATION
    8
 0.387597E-02 0.116279E-01 0.310078E-01 0.697674E-01 0.147287E+00 0.341085E+00
 0.612403E+00 0.100000E+01
POLYGON PARAMETERS
    2
    1    1 1.200000E+01
 0.000000E+00 0.000000E+00 0.100000E-01 0.138200E-04 0.100000E+01
 0.833333E+00
    2    1 1.200000E+01
 0.000000E+00 0.000000E+00 0.100000E-01 0.138200E-04 0.100000E+01
 0.833333E+00
```

Figure 9 shows the results of the TRIPOLY simulation compared to the results of the analytical solution of TANG et al. (1981). The three curves exhibit concentration profiles along the fracture for three time steps 97 days, 995 days and 9991 days. The solid curve represents the analytical solution, the square symbols indicate the TRIPOLY results at the nodes of the finite element mesh. Note that the original discretization is very coarse; original nodes are only at locations 0.0 m, 0.1 m, 0.2 m etc. All the nodes in between the original ones have been added within the adaptive gridding procedure. That means that the fracture network discretization is different from time step to time step.

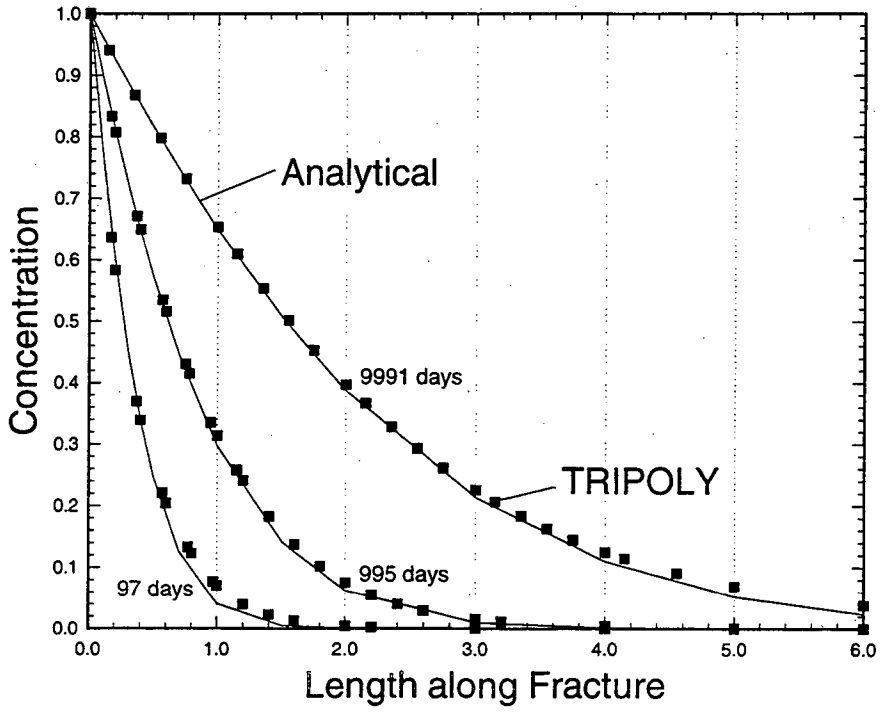


Fig. 9: Fracture concentration for the TRIPOLY results and the analytical solution (TANG et al., 1981)

Due to the relatively simple geometry of this example, it is not possible to verify the code's ability to simulate diffusive transport in matrix blocks of complex shape. However, this has been successfully demonstrated in other studies (e.g. BIRKHÖLZER, 1994). Here, our major goal is to check the performance of the Lagrangian-Eulerian scheme in combination with the fracture-matrix interaction tool. Altogether, the analytical solution and the TRIPOLY results match very well, in particular when considering that the original spatial discretization is very coarse.

3.3 Example 2: Transport in a Complex Fracture-Matrix System

The second example concerns advective-dispersive transport in a complex fracture network including diffusion into the porous matrix blocks. The whole chain of programs described in this report is used in this example. FMGN generates the fracture network, RENUMN renumbers it and eliminates dead-ends, POLY calculates all the matrix block properties and TRIPOLY finally does the flow and transport simulation. Only those input data sets shall be presented here which have to be written by the user him/herself. All of them contain control parameters or general information.

The input data set FMG.INP provides all the information needed to generate the fracture network. Please find a detailed description of FMGN and the input data set in BILLAUX et al. (1989). The fracture network of our example comprises 2 orthogonal sets of 600 fractures each in a 10 m x 10 m square flow region. The angle between the fracture axis and the x-direction is 45° for set 1 and 135° for set 2. We assume that the fractures have uniform properties within each set, i.e. uniform angles between the fractures and the x-direction, a uniform fracture aperture of 0.8 10^{-4} m, a fracture length of 1.0 m and a longitudinal dispersion of 0.5 m. Hence, the standard deviations given in FMG.INP are set to zero. Two no-flow boundaries are given at the upper and lower boundary of the flow direction. The left boundary is associated with a hydraulic head of 0.1 m, the right boundary is associated with a hydraulic head of 0.0 m.

FMG.INP

```
ICONT*****    2      IPLOT*****    2      IMESH*****    1
IKEEP*****    1      IUNITS****    1      IGENE*****    0
IRANF*****    1      RSEED*****    3.4386d+03
ANGLES, APERTURES and LENTHS ARE CONSTANT
2 FRACTURE SETS
XGENE*****   10.0    YGENE*****   10.0
NSETS*****    2   ITOLE*****    5
ICENT*****    2   IDENS **      1
RLAMBNFRAC  600      THETA **       0.0
ICHAR*orie    3
IDIST*****    5      EV ******* 45.0      SD *******   0.00
ICHAR*radi    3
IDIST*****    5      EV ******* 1.00      SD *******   0.00
ICHAR*aper    3
IDIST*****    5      EV *******0.80e-04   SD *******   0.00
ICENT*****    2   IDENS **      1
RLAMBNFRAC  600      THETA **     00.0
ICHAR*orie    3
IDIST*****    5      EV *******135.0      SD *******   0.00
ICHAR*radi    3
IDIST*****    5      EV ******* 1.00      SD *******   0.00
ICHAR*aper    3
IDIST*****    5      EV *******0.80e-04   SD *******   0.00
xmesh*****1.0e+01    ymesh*****1.0e+01    rotan*****   0.00
         0          1          0          1
    0.0000     0.1000     0.0000     0.0000
```

After running FMGN a number of output data sets have been generated (see Table 2.1). The structure of the randomly generated fracture network is already given at this point; however, dead-end nodes or dead-end clusters have not been eliminated; also the network is not optimized with regard to a minimized bandwidth. RENUMN takes care of renumbering the mesh and eliminating dead-ends. Most of the input for RENUMN is generated by FMGN, contained in the file RENUMnn.DAT. The user has to provide only one additional data set which gives some general information specifying material parameters for the fracture network. This data set is called TRINET.INP. It gives values for the fluid density, the dynamic viscosity, gravity constant as well as 'default'-values for the specific storage and the dispersion/dispersivity.

TRINET.INP

```
INPUT VARIABLES FOR TRINET
RHOR     =1.0000E+03 RMUR     =1.0000E-03 GRAVR    =9.8067E+00
SSUBS    =0.0000E+01 DISPC    =5.0000E-02
```

RENUMN generates the data sets NODE.TRUNC, ELMT.TRUNC, NODE.ALL and ELMT.ALL, comprising nodal and element information for the fracture network. Transmissivity values for each element are calculated and written into ELMT.TRUNC and ELMT.ALL, respectively. The data sets with extension xxxx.TRUNC describe the truncated fracture mesh, data sets with extension xxxx.ALL describe the fracture mesh with dead-end fractures. Dead-end clusters have been removed in both cases. All information needed to describe the geometry and material properties of the random fracture network is given at this point. Before running TRIPOLY the user has to decide whether or not to include dead-ends in a simulation run. In our case dead-ends shall not be considered. Thus we rename the data files NODE.TRUNC and ELMT.TRUNC into NODE.INP and ELMT.INP, respectively. Figure 10 shows the truncated fracture network generated by FMGN and RENUMN. It comprises 3520 fracture nodes and 5668 fracture elements. However, this number increases within the simulation due to adaptive gridding.

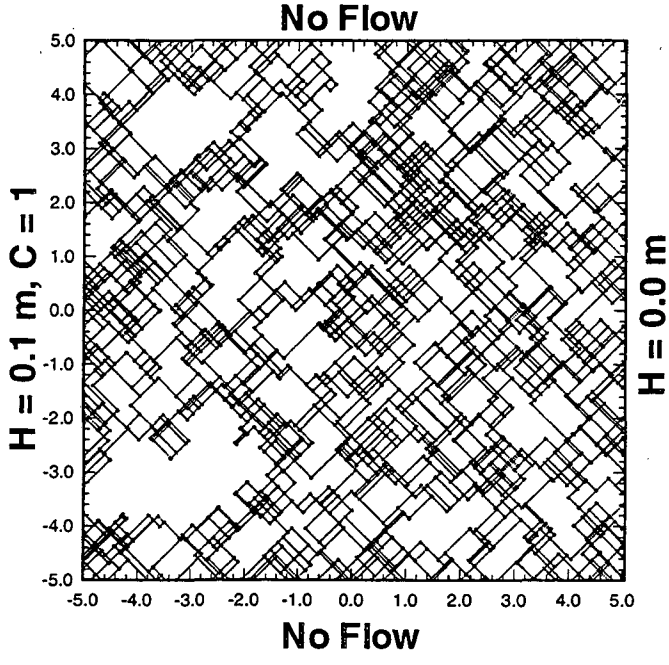


Fig. 10: Fracture network of example 2

As already mentioned, some information regarding flow boundary conditions can be provided in the file FMG.INP, and this information is automatically written into NODE.INP. However, this is just possible for relatively simple boundary conditions, i.e. each side of the model area is associated with either constant fluxes, constant heads or a constant linear distribution of head. The first and second possibility has actually been chosen in our example by prescribing no-flow boundary conditions at the top and the bottom boundary and giving a constant head on the left and the right boundary. The left boundary is associated with a hydraulic head of 0.1 m, the right boundary is associated with a hydraulic head of 0.0 m. If more complex cases of flow boundary conditions shall be given, or for complex initial conditions as well as for the transport boundary conditions, the user has to either edit data set NODE.INP or change subroutine *triout* in RENUMN. In this example the left boundary of the model area is assumed to be contaminated. A DIRICHLET-type boundary condition C=1 is imposed. Solutes released at this boundary are carried through the model area following the direction of flow and leave at the right (outflow) boundary.

Now, the preprocessing code POLY calculates the geometrical properties of the matrix blocks in the fracture network of Figure 10. This procedure is only needed if the exchange processes between fractures and matrix blocks shall be taken into account. POLY needs the input files NODE.ALL and ELMT.ALL generated by RENUMN as well as the input file PROX.INP which is provided by the user. PROX.INP contains a number of control variables for the generation of polygons and proximity functions. In our example, no dead-end fractures shall be considered, the random number generation is controlled by a certain start-up value, the discretization in the porous matrix shall be performed with 8 elements, grade of proximity function is 5, and the relative discretization is given to 1/2/5/10/20/50/70/100, with the first (small) element at the fracture-matrix interface and the last (100 times larger) element in the middle of the blocks. The last two lines provide material properties for the matrix blocks. The matrix blocks in the domain have identical hydraulic properties, with a porosity of 0.02 and a molecular diffusion coefficient of 0.2×10^{-8} m^2/s. However, the size and shape of the blocks varies significantly.

PROX.INP

```
TEST DATA SET
MATERIAL PROPERTIES FOR MATRIX BLOCKS
UNIFORM BLOCK PROPERTIES
INET     0     IRANF    1     NRAND 2000 RSEED    1.0000D+03
NDISC    8     NGRAD    5
    1    2    5   10   20   50   70  100
SSP       0.0000E-00 XKM      0.0000E-00
XPOR      0.2000E-01 XDIF     0.2000E-08 XRET     1.0000E-00
```

The fracture network shown in Figure 10 includes 2337 matrix blocks. POLY writes the data sets POLY.INP, POLYMAT.INP, GRAFICS.SCI and GRAFICS.SCR. POLY.INP contains nodal information defining the polygons, POLYMAT.INP contains the relative discretization and material properties of the matrix blocks. The data sets GRAFICS.SCI and GRAFICS.SCR are only needed when graphical output of matrix results is desired. They are not needed for the simulation itself.

At this point all input data sets are provided except for the simulation control data set CTRL.INP. In our example CTRL.INP is given as follows:

CTRL.INP

```
TEST CASE 2: COMPLEX FRACTURE NETWORK
2 FRACTURE SETS WITH UNIFORM PROPERTIES
2337 MATRIX BLOCKS WITH UNIFORM PROPERTIES
IMODE    =    2IBMODE   =    2MXSTEP   = 5000IPFREQ   =   -2NSTEP0   =    1
IDOPO    =    1LOGDCON  =    2LOGDIF   =   -1LOGDISP  =    2
TIME0 =  0.0000E+05 TMAX  =  6.0001E+07 DTINI =  5.0000E+02 PRR   =  1.1000E+00
THETA =  1.0000E+00 TOLE  =  1.0000E-02 DCINT =  2.0000E-01 DCON  =  1.0000E-02
DCOFF =  1.0000E-02 RHOR  =  1.0000E+03 RMUR  =  1.0000E-03 GRAVR =  9.8067E+00
DEPS  =  1.0000E-03 DCDIF =  1.0000E-01 DBRAN =  3.0000E+00 DIFMOL=  2.0000E-07
OUTPUT TIME STEPS
   0.5E+07   3.0E+07
```

We perform simulation runs with up to 5000 time steps and a maximum simulation time of 6.0 10^7 s (694.4 days). Figure 11 shows the concentration in the fractures after 0.5 10^7 s (57.9 days), indicated by the different shading levels of the fracture mesh nodes. Without matrix diffusion particles would cross the entire model area within less than 10^6 s (11.6 days). However, the diffusive solute exchange between the fractures and the matrix pores has a very strong impact on the transport behavior of the fracture-matrix system; the solute transport in the fractures is strongly retarded and solutes are just beginning to reach the outflow boundary.

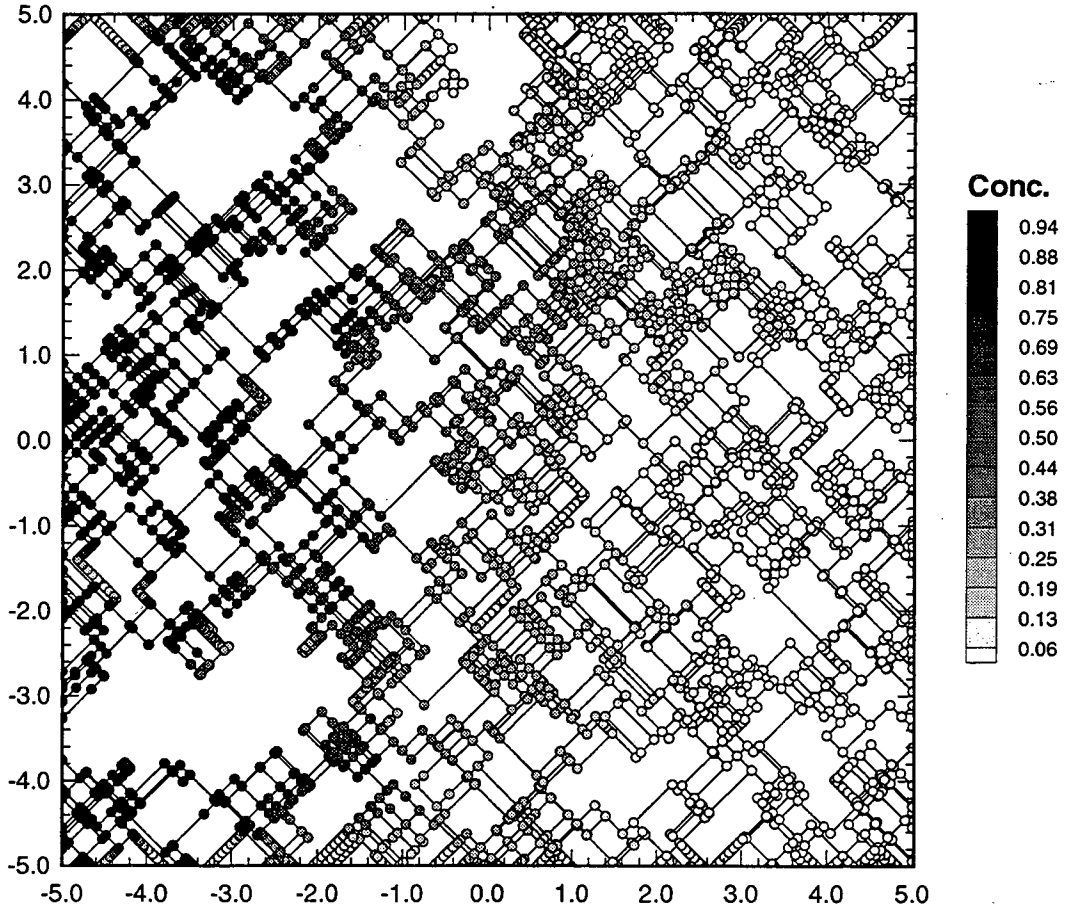


Fig. 11: Concentration in the fracture network after 0.5 10^7 s (57.9 days)

Figure 12 shows concentration values in the matrix blocks for the same time step 0.5 10^7 s (57.9 days). For the sake of visualization, average concentration values are presented for each block. Remember that TRIPOLY simulates the transport in the matrix solving one-dimensional diffusion equations for each matrix block. Therefore, in the simulation procedure it is possible to accurately account for the steep gradients between the fractures and the matrix. However, TRIPOLY internally calculates average values out of the one-dimensional concentration profiles and writes them into the output files DATnn.TEC.

A comparison of Figure 11 and Figure 12 clearly indicates that the contaminant build-up in the matrix blocks is much slower than in the fractures. Moreover, the average concentrations in the blocks vary significantly due the different block sizes. Large blocks which offer a larger pore volume for storing contaminants are fairly clean while some smaller blocks are contaminated.

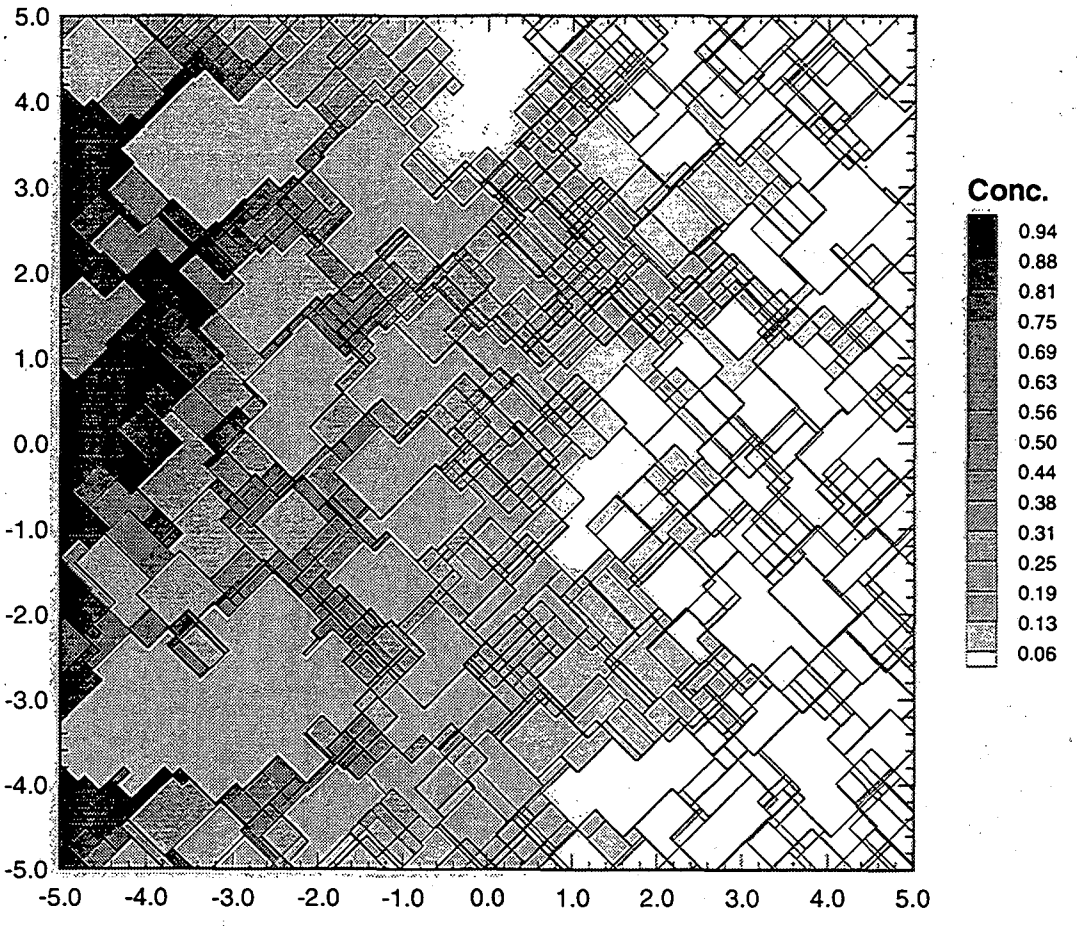


Fig. 12: Average concentration in the matrix blocks after 0.5 10^7 s (57.9 days)

Results of another time step are presented in Figures 13 and 14, respectively. After 3.0 10^7 s (347.2 days) most fractures are contaminated with concentrations close to 1.0. Lower concentrations are only obtained in no-flow fractures where molecular diffusion rather than advection is the relevant physical process. Most of these fractures are located at the upper and lower boundary.

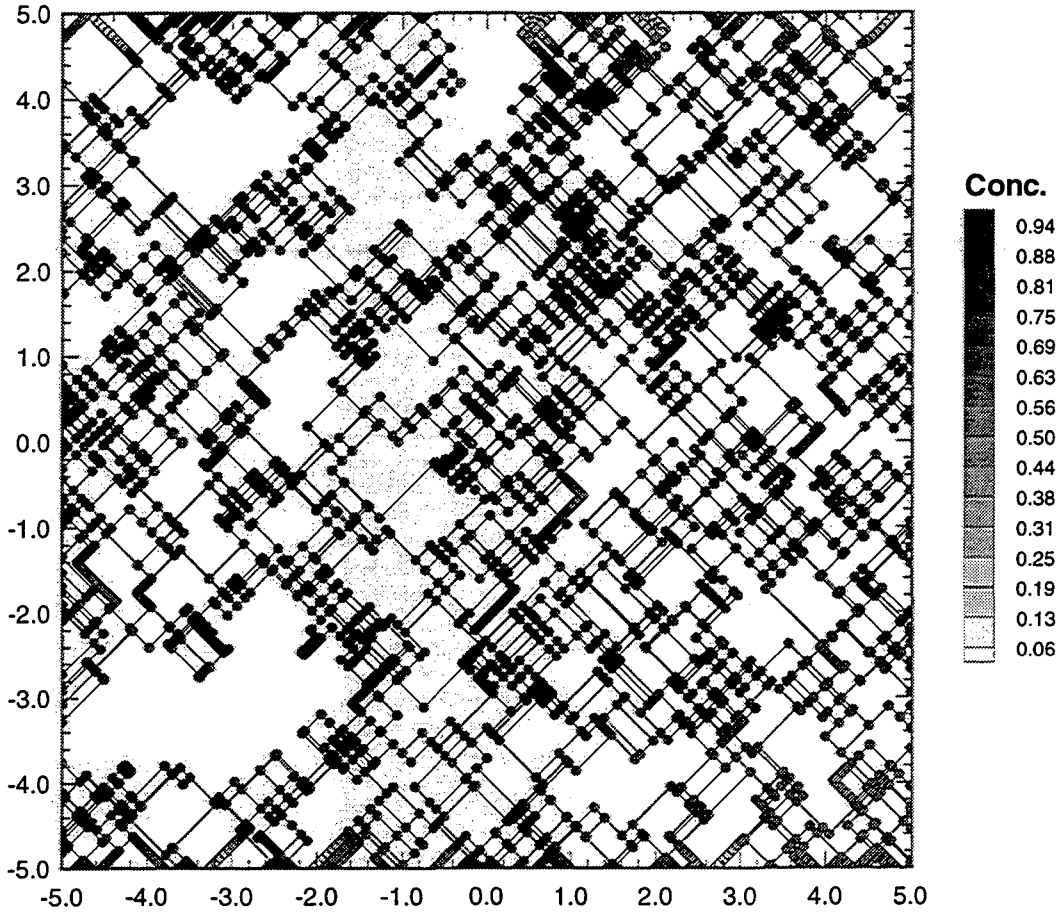


Fig. 13: Concentration in the fracture network after 3.0 10^7 s (347.2 days)

The average concentration values of the matrix blocks show a different picture. Although many of the smaller blocks are already contaminated with concentrations close to 1.0, there are a number of large blocks which are still fairly clean. This means that even after 3.0 10^7 s (347.2 days) significant concentration differences between the fractures and the matrix can be obtained for large blocks, and part of the solute diffuses from the fractures into the matrix pores. This process may continue for a much longer time because the diffusive transport in the matrix is very slow.

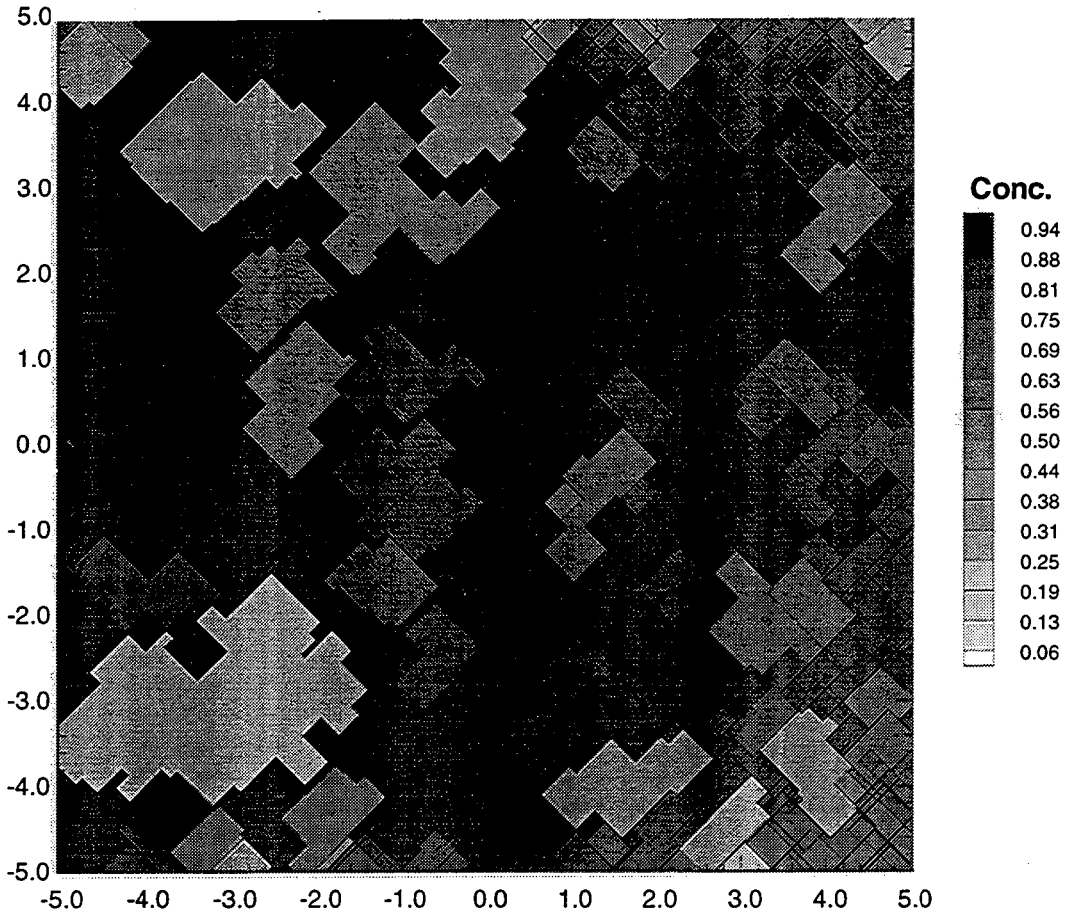


Fig. 14: Average concentration in the matrix blocks after 3.0 10^7 s (347.2 days)

Figure 15 exhibits tracer breakthrough curves measured at the right (outflow) boundary. The solid black line represents the tracer breakthrough calculated by TRIPOLY when taking matrix diffusion into account. The dashed line gives the tracer breakthrough when only the fractures are considered and matrix diffusion is neglected. The strong impact of matrix diffusion on the transport behavior in a fractured porous formation is immediately evident.

Our simulation results clearly show that the individual matrix blocks exhibit a very different response to perturbations in the fracture network, simply due to their variability in size and shape. This effect may even be enhanced by spatially varying material properties. As shown by JANSEN et al. (1996), the observed phenomenon has a strong impact on the assignment of equivalent continuum parameters for the matrix blocks, a problem which is associated with the use of dual-porosity models. Central to those models is the assumption that the heterogeneous rock formation can be separated into two homogeneous media, one continuum representing the interconnected fracture system, a second continuum representing the porous matrix blocks. Numerous studies have been performed in the past to check if a continuum representation of fracture networks is valid, to derive equivalent continuum parameters and to estimate the model error associated with this averaging process. However, little work has been done to address this task with regard to the matrix blocks of a given subdomain, which may vary considerably in size,

shape or material properties. In most cases of dual porosity modeling, the matrix continuum parameters are only roughly estimated; an error analysis is not performed.

Based on the results of the (exact) TRIPOLY results we can now address the problem of assigning equivalent continuum parameters for matrix blocks with significant heterogeneity. We repeat the TRIPOLY run with the same fracture network and the same boundary and initial conditions, but now assign averaged parameters using different averaging techniques. Breakthrough curves of both runs can be compared and analyzed to check the accuracy of the averaged representation of the matrix blocks and to estimate the error between the two solutions, respectively. The dotted line in Figure 15 is an example of such a simulation run. Here, all the differently shaped matrix blocks are described by the **same** proximity function, which was calculated by a random procedure (as described in Section 1.3.4) performed for the entire domain.

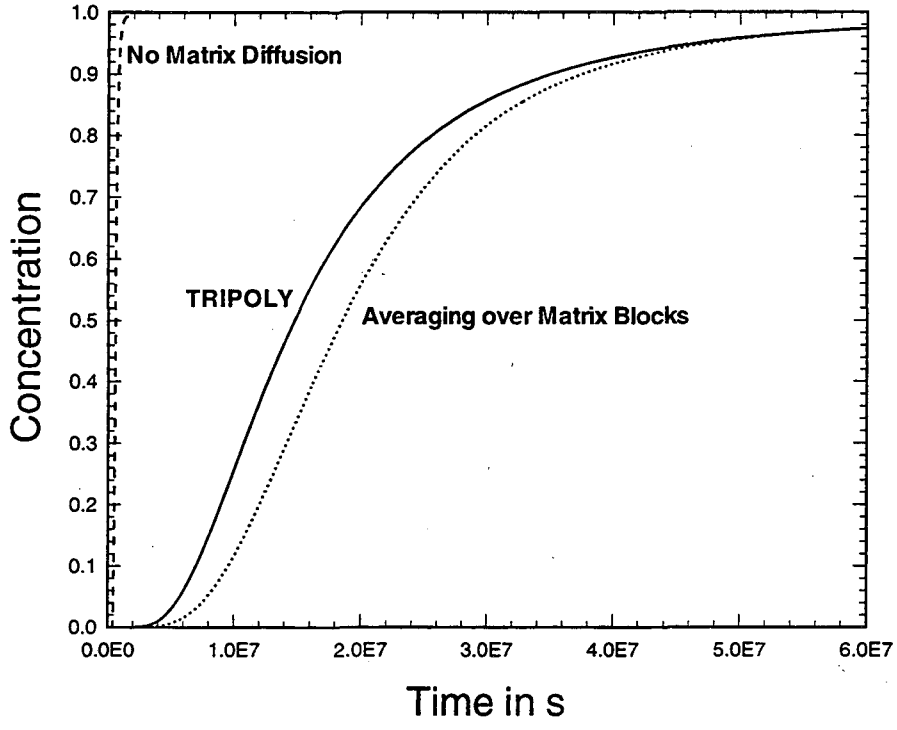


Fig. 15: Tracer breakthrough curves

The results of this run indicate that the long-term behavior of the equivalent continuum solution is quite good. However, the contaminant build-up of the breakthrough curve is slow compared to the (exact) TRIPOLY solution. Apparently, the results obtained with averaged matrix geometry overestimate the diffusive losses between fractures and matrix. A possible explanation might be the following: Densely fractured subregions within the model area are correlated to small matrix blocks. Small matrix blocks are contaminated more quickly than larger ones, and the diffusive losses between fractures and matrix decrease more quickly. This means that the transport in densely fractured zones with small matrix blocks may be faster than the average transport. Apparently, the equivalent continuum representation cannot describe this behavior. More work shall be done in the future to address this important issue, and the newly developed code TRIPOLY provides a sophisticated tool for doing so.

Acknowledgment

This work was supported by a NATO-Postdoctoral Scholarship, provided by the *German Academic Exchange Organization (DAAD),* Bonn, Germany, and by the *Power Reactor and Nuclear Fuel Development Corporation (PNC), Tokyo, Japan.* The authors are grateful to C. Oldenburg and C. Doughty of the Berkeley Lab for critical review of the manuscript and suggestions for improvements.

References

BIBBY, R. (1981), Mass transport in dual-porosity media, *Water Resour. Res.*, 17(4), 1075-1081.

BILLAUX, D., S. BODEA, J.C.S. LONG (1988), FMG, RENUM, LINEL, ELLFMG, ELLP and DIMES: Chain of programs for calculating and analyzing fluid through two-dimensional fracture networks - Theory and design, Lawrence Berkeley Laboratory report LBL-24914, Berkeley, CA.

BILLAUX, D., J. PETERSON, S. BODEA, J.C.S. LONG (1989), FMG, RENUM, LINEL, ELLFMG, ELLP and DIMES: Chain of programs for calculating and analyzing fluid through two-dimensional fracture networks - User's manual and listings, Lawrence Berkeley Laboratory report LBL-24915, Berkeley, CA.

BIRKHÖLZER, J. (1994), *Numerische Untersuchungen zur Mehrkontinuumsmodellierung von Stofftrans-portvorgängen in Kluftgrundwasserleitern*, Mitt. Inst. f. Wasserb. u. Wasserw. 93, Aachen: Academia.

BIRKHÖLZER, J., G. ROUVE (1994), Dual-continuum modeling of contaminant transport in fractured formations, Proc. 10th Int. Conf. on Numerical Methods in Water Resources at Heidelberg, Germany.

BIRKHÖLZER, J., G. ROUVE (1991), Solute transport in fractured porous rock, Proc. 7th Int. Congress Rock Mechanics at Aachen, Germany.

HUYAKORN, P.S., B.H. LESTER, J.W. MERCER (1983), An efficient finite-element technique for modeling transport in fractured porous media: 1. Single species transport, *Water Resour. Res.*, 19(3), 841-854.

KARASAKI, K. (1986), A new advection-dispersion code for calculating transport in fracture networks, Lawrence Berkeley National Laboratory annual report LBL-22090, Berkeley, CA.

JANSEN, D., J. BIRKHÖLZER, J. KÖNGETER (1996), Contaminant transport in fractured porous formations with strongly heterogeneous matrix properties, *Proc. 2nd North American Rock Mechanics Symp., NARMS '96*, Montreal.

NEUMAN, S.P. (1984), Adaptive Eulerian-Lagrangian finite element method for advection-dispersion, *Int. Journal for Num. Meth. in Eng.*, 20, 321-337.

OKUSU, N., K. KARASAKI, J.C.S. LONG, G.S. BODVARSSON (1989), FMMG: A program for discretizing two-dimensional fracture/matrix systems - Theory, design and user's manual, Lawrence Berkeley Laboratory report LBL-26782, Berkeley, CA.

PRESS, W.H., B.P. FLANNERY, S.A. TEUKOLSKY, W.T. VETTERLING (1986), *Numerical recipes: The art of scientific computing*, Cambridge University Press, Cambridge.

PRUESS, K., K. KARASAKI (1982), Proximity functions for modeling fluid and heat flow in reservoirs with stochastic fracture distribution. *Proc. 8th Workshop Geotherm. Res. Eng.*, Stanford: 219-224.

SEGAN, S., K. KARASAKI (1993), TRINET: A flow and transport code for fracture networks - User's manual and tutorial, Lawrence Berkeley Laboratory report, Berkeley, CA.

TANG, D., E. FRIND, E.A. SUDICKY (1981), Contaminant transport in fractured porous media: Analytical solution for a single fracture, *Water Resour. Res.*, 17(3), 555-564.

THOMAS, L.H. (1949): Elliptic problems in linear differential equations over a network, Watson Science Computer Lab., Rep. Columbia University, New York.

Ernest Orlando Lawrence Berkeley National Laboratory
One Cyclotron Road | Berkeley, California 94720

Prepared for the U.S. Department of Energy under Contract No. DE-AC03-76SF00098