1992

R.A. Trompert

MOORKOP, an adaptive grid code for initial-boundary value problems in two space dimensions

Department of Numerical Mathematics Report NM-N9201 December

CWI is the research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a non-profit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands organization for scientific research (NWO).

MOORKOP, an Adaptive Grid Code for Initial-Boundary Value Problems in Two Space Dimensions

Ron Trompert
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Abstract

This report contains the manual for the adaptive grid code MOORKOP.

1991 Mathematics Subject Classification: 68-00, 65N50
1991 CR Categories: G.1.8.
Key Words & Phrases: MOORKOP, manual, adaptive grid methods, partial differential equations, fluid-flow/solute-transport in porous media with inhomogeneities.

Note: This work was carried out as a part of contract research by order of the Laboratory for Soil and Groundwater Research of RIVM - the Dutch National Institute of Public Health and Environmental Protection - in connection with project "Locatiespecifieke Modelvalidatie" 725205. Financial support for this project was provided by the Dutch Ministry of Economic Affairs.

Report NM-N9201
ISSN 0169-0388
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

MANUAL

MOORKOP,

AN ADAPTIVE GRID CODE FOR INITIAL-BOUNDARY VALUE PROBLEMS IN TWO SPACE DIMENSIONS

Ron Trompert
CWI
P.O. Box 4079, 1009 AB, Amsterdam, The Netherlands

Version 2.0 and 2.1, date 14-12-1992

1. INTRODUCTION

MOORKOP solves initial boundary value problems (IBVPs) for systems of partial differential equations (PDEs) of the following type, defined on a domain Ω with boundary $\partial\Omega$,

$$\begin{aligned} &G(x,y,t,u,u_t,u_x,u_y,u_{xx},u_{xy},u_{yy}) = 0, \quad (x,y) \in \Omega, \quad t > t_0, \\ &H(x,y,t,u,u_t,u_x,u_y) = 0, \quad (x,y) \in \partial\Omega, \quad t > t_0, \\ &u(x,y,t_0) = u_0(x,y), \quad (x,y) \in \Omega \cup \partial\Omega, \end{aligned} \tag{1.1}$$

where the exact solution u may be vector valued and Ω is a rectangle. MOORKOP is an adaptive grid code based on local uniform grid refinement. Adaptive grid methods prove to be very useful in cases where the solution to the PDEs is locally steep, for instance at a pulse or a front. For such problems, a single uniform space grid held fixed throughout the entire time evolution can be computationally very inefficient, since, to afford an accurate approximation, such a grid has to be very fine over the whole domain while a fine grid is only needed there where the solution is steep. Adaptive grid methods refine the space grid only there where it is really needed, hence, reducing the memory use and CPU time. Local uniform grid refinement is also discussed in our previous work [8-15]. The main feature of local uniform grid refinement is that integration takes place on a series of nested, local uniform finer and finer subgrids which are automatically adjusted at discrete times in order to follow the movement of rapid spatial transitions. All grids in use are uniform and cartesian. The generation of these subgrids is continued until the spatial phenomena are described with sufficient accuracy.

This manual deals with two versions of MOORKOP, MOORKOP 2.0 and MOORKOP 2.1. The first version is a general purpose adaptive grid code, designed to handle general systems like (1.1). The second one can also solve systems like (1.1) but on top of that, it has especially been designed for realistic groundwater flow problems. In natural situations in groundwater flow problems, soil parameters such as the permeability, the transversal and longitudinal dispersivity can change abruptly from one region to another. The solution profile of the pressure or the concentration of a solute etc. can exhibit a kink at such an interface. In order to get consistent numerical approximations, interface conditions are applied at grid nodes in the vicinity of these abrupt changes. These conditions connect the numerical solution on both sides of the interface and are based on continuity of fluxes across these interfaces. Numerically, the interfaces are assumed to coincide with the cell edges and the soil parameters are assumed to be piecewise constants.

For time integration we use implicit Euler for the first time step and the second order two-step implicit BDF method with variable coefficients for the following time steps where variable time stepsizes are taken. Standard second order finite differences are used for space discretization and the interpolation is linear. The discretization of the boundary conditions is of second order in MOORKOP 2.0 and of first order in MOORKOP 2.1. The discretization of the interface conditions in MOORKOP 2.1 is also of first order. The resulting systems of equations are solved by an adapted version of modified Newton's method in combination with ILU preconditioned Bi-CGSTAB [16].

An outline of the local uniform mesh refinement method is given in Section 2. In Section 3, the strategies concerning grid refinement, time stepping, and the solution procedure of the systems of nonlinear algebraic equations are discussed. As an example, we describe a brine transport model and the associated interface conditions in Section 4. A description of the program input and error exits is given in Section 5 and Section 6 deals with postprocessing. In Section 7, we use two example problems to elaborate the program input. The complete program input is given for both example problems. The first example problem is a fairly simple one, namely the heat equation. The second one involves the solution of a brine transport problem in porous media with inhomogeneities. Finally, some concluding remarks are given in Section 8.

2. Outline of the Adaptive-Grid Method

Although its elaboration readily becomes complicated, the idea behind local uniform grid refinement is simple. Starting from a coarse base grid, covering the whole domain, finer and finer uniform subgrids are created locally in a nested manner in regions of high spatial activity. Here, a set of interconnected grid cells, all having the same sizes, is called a subgrid. A set of subgrids having the same cell sizes is called a grid level or just grid. Hence, a grid level consists of a single subgrid or several disjunct non-overlapping subgrids. A new initial-boundary value problem is solved at each grid level and the integration takes place in a consecutive order, from coarse to fine. Each of these integrations spans the same time interval. Required initial values are defined by interpolation from the next coarser grid level or taken from a grid level from the previous time step when available. Internal boundaries, i.e. subgrid boundaries lying in the interior of the domain, are treated as Dirichlet boundaries and values are also interpolated from the next coarser grid level. Where the boundary of a fine subgrid coincides with the boundary of the domain, the given boundary conditions are used. The generation of grid levels is determined by the local refinement strategy and is continued until the spatial phenomena are described well enough by the finest grid. The fine grid cells are created by bisecting the sides of the cells of the next coarser grid. Note that the subgrids created this way need not be rectangles.

During each time step the following operations are performed:

1. *Solve PDEs on the coarse grid.*
2. *If the desired accuracy in space or the maximum number of grid levels is reached then go to 8.*
3. *Determine new finer grid level at forward time.*
4. *Interpolate internal boundary values at forward time.*
5. *Provide new initial values at backward time.*
6. *Solve PDEs on new grid level, using the same steplength.*
7. *go to 2.*
8. *Inject fine grid values in coinciding coarser grid points.*

Thus, for each time step the computation starts at the coarse base grid using the most accurate solution available, since fine grid solution values are always injected in coinciding coarse grid points and all grid levels are kept in storage for step continuation.

3. STRATEGIES

3.1. Refinement Strategy

The reason why the local uniform grid refinement method is an interesting method for solving PDEs with steep solutions is that it can solve these PDEs just as accurately as on a very fine grid, but with considerably less computational effort, since the involved fine subgrids cover only a part of the domain. Moreover, it creates extra refinements when necessary and removes these when they are no longer needed. This refinement process is controlled by a refinement strategy. In [9-11, 14, 15], the refinement strategy is based on a comprehensive error analysis taking into account space discretization and interpolation error estimates. The intention of this strategy is that the overall spatial accuracy is dominated by the spatial accuracy at the finest grid level. When the number of grid levels is constant for all times, this strategy should lead to a spatial accuracy which is comparable to the one achieved with a single uniform grid having cell sizes identical to those of the finest grid level in use in the adaptive grid method. The success of this refinement strategy is very much dependent on the accuracy of error estimates. It is clear that these error estimates can only be accurate when the solution is sufficiently smooth, i.e. it may be steep but it should be sufficiently differentiable in space. Since non-smoothness in the boundary conditions, or even in the solution itself, is a well known phenomenon in brine transport problems, the approach above was dropped and replaced by a more heuristic approach. In [12, 13] the refinement strategy was based on a curvature monitor. This monitor is also used here.

We are now going to introduce some notation. Let the vector U^n denote the numerical approximation to the solution u of (1.1) at time t_n on a space grid. Suppose that (1.1) consists of *npde* PDEs, so the solution vector u has length *npde*. In this case, u_i represents the i^{th} component of u and U_i^n the numerical approximation to u_i at time t_n on a space grid. Let the component of U_i^n associated with the grid node (k,l) be written as $U_i^n(k,l)$, where k and l are indices related to the space co-ordinates x and y of this node. The curvature monitor value corresponding with the i^{th} solution component in (k,l) is now defined as

$$ESTS_i(k,l) = \frac{1}{scale(i)} \{ |U_i^n(k+1,l)-2U_i^n(k,l) + U_i^n(k-1,l)| + |U_i^n(k,l+1)-2U_i^n(k,l) + U_i^n(k,l-1)| \}. \tag{3.1}$$

The user defined array *scale* has length *npde* and holds characteristic values of each solution component. At every grid node $ESTS_i(k,l)$ is computed for each solution component i. At boundary nodes, the difference formulas in (3.1) are replaced by one-sided formulas.

Suppose we have just completed a time step on grid level m; grid level 1 is the coarsest grid level or the base grid, grid level 2 is the next coarsest grid level and so on. After this time step the maximum values of $ESTS_i(k,l)$ are computed over grid level m for each component i. These maxima are denoted as $ESTSmax_i$. If for some i, $ESTSmax_i > TOLS$, then a new grid level $m+1$ is created within the current time step, provided $m+1$ does not exceed the user specified maximum number of grid levels. Here, TOLS is a user defined tolerance. Grid level $m+1$ is now determined as follows. For each i, for which $ESTSmax_i > TOLS$ holds, a node is flagged if at this node or one of its direct neighbours $ESTS_i(k,l) > ¼ \times TOLS$. The cells around the flagged nodes of grid level m will be subdivided in four identical cells. The set of these finer cells makes up grid level $m+1$, on which the current time step will now be repeated.

Finally, we have built in an extra condition to smoothen the behaviour of the code. Suppose that the maximum number of grid levels during the previous time step is *levtop* and that at grid level m<*levtop*, $ESTSmax_i \le TOLS$. Although this means that a new finer grid level $m+1$ is actually not necessary, it will still be created when $ESTSmax_i > 0.9 \times TOLS$. This way fluctuation of the maximum number of grid levels from one time point to the next is likely to be avoided. The curvature monitor is programmed in the subroutine SPCEr and the tolerance $¼ \times TOLS$ is contained in the subroutine setTOL. The flagging of the nodes is done

by the subroutine errflg. The remainder of the refinement strategy is programmed in the subroutine PDEsol.

3.2. Time Integration Aspects

We have implemented the two-step BDF method of order two which we apply in the variable stepsize mode. The time derivative in (1.1) is then approximated as

$$U_t \approx \frac{U^n - a_1 U^{n-1} - a_2 U^{n-2}}{\theta_2 \Delta t_n}, \tag{3.2}$$

where

$$a_1 = \frac{(c+1)^2}{c^2+2c}, \quad a_2 = \frac{-1}{c^2+2c}, \quad \theta_2 = \frac{c+1}{c+2}, \tag{3.3}$$
$$c = \frac{\Delta t_{n-1}}{\Delta t_n}, \quad \Delta t_n = t_n - t_{n-1}.$$

Here, U_t represents the pointwise restriction of u_t to a space grid. We note that variable time stepping is a prerequisite for brine transport problems in porous media, as they can exhibit a highly distinct behaviour in time. As starting formula we employ the one-step BDF method of order one (implicit Euler).

The time stepsize is controlled by the time error monitor value

$$ESTT = \| \frac{U_i^n - U_i^{n-1}}{scale(i)} \|_\infty, \quad i=1,.....,npde, \tag{3.4}$$

which is computed only over the interior grid nodes of each grid level for reason of robustness of the code. We will not elaborate this further here. After each time step on all grid levels, defined in Section 2, the maximum value of *ESTT* is computed over all grid levels. If this maximum exceeds a user specified tolerance *TOLT*, then the time step is rejected, otherwise accepted. For each grid level a new time stepsize is predicted such that the predicted value of *ESTT* for the new time step is equal to 0.5 × *TOLT*. The minimum of these new time stepsize estimates is taken to be the time stepsize for the next time step. However, in case of a step rejection, the new time stepsize will be taken as 0.8 × this estimated value. In all cases we require that the new time stepsize is not smaller than ⅓ × and not larger than 2 × the old time stepsize to avoid too large jumps in the stepsize selection. Finally, the new time stepsize is corrected with a small value to assure that the next output point is reached exactly. The time error monitor is evaluated in the subroutine TIMEr and the remainder of the time step strategy is programmed in the subroutine PDEsol.

3.3. Solution of the Linear and Nonlinear Systems

Because we use an implicit integration method and treat PDEs like (1.1) fully coupled, we are facing the task of solving large coupled systems of nonlinear algebraic equations. In our code we use an adapted version of modified Newton in combination with the preconditioned Bi-CGSTAB [16] for solving these equations. In the remainder of this section we will explain the implemented solution procedure.

For any system of PDEs like (1.1), the required Jacobian matrix for the Newton process is computed in a completely automatic manner. To illustrate this, consider the 1D form

$$G(u_t, u_x, u_{xx}) = 0, \tag{3.5}$$

for which the Jacobian matrix is tridiagonal. Recall that we use central 3-point finite differencing on a uniform space grid with cell size Δx. The diagonal entries, corresponding with internal grid points, are then easily seen to be defined by the functional

$$\frac{\partial G}{\partial u_t}\frac{1}{\theta_2 \Delta t} - \frac{\partial G}{\partial u_{xx}}\frac{2}{(\Delta x)^2}, \tag{3.6}$$

where Δt is the time stepsize. Similar expressions are easily found for the nondiagonal entries and for grid nodes whose finite difference expression depends on the boundary conditions. This way of constructing a Jacobian was borrowed from [6]. In our code the partial derivatives are estimated by a simple first order difference formula, so that the user does not need to specify these. The procedure we followed, described below, was obtained from [3]. For example, we use the approximation

$$\frac{\partial G}{\partial u_{xx}} \approx \frac{G(u_t, u_x, u_{xx} + \varepsilon) - G(u_t, u_x, u_{xx})}{\varepsilon}, \tag{3.7}$$
$$\varepsilon = (uround)^{1/2} \max(|u_{xx}|, \text{typ}\,(u))\, \text{sign}(u_{xx}).$$

Here *uround* is the machine roundoff error. The value "typ (u)" represents a characteristic value of u. In our code we use the user defined array *scale* for these values. In most cases, including the numerical simulation of groundwater flow and transport, (3.7) works very satisfactorily. However, in case of a very badly scaled problem it may be necessary to replace "typ (u)" by some other value. In (3.7) we use the recomputed value for ε, given by

$$\varepsilon := (u_{xx} + \varepsilon) - u_{xx}. \tag{3.8}$$

This insures that the numerical value of ε in the nominator of (3.7) is identical to the one in the denominator which enhances the accuracy of approximation (3.7).

Let the nonlinear system of equations to be solved be denoted as,

$$F(U) = 0. \tag{3.9}$$

In the modified Newton approach, the linear system,

$$\begin{aligned} J(U^0)\delta^k &= -F(U^{k-1}), \\ U^k &= U^{k-1} + \delta^k, \end{aligned} \tag{3.10}$$

is subsequently solved, starting with $k=1$, until a stopping criterion is fulfilled. Here, J is the Jacobian matrix, U^0 is the initial guess, U^k is the k^{th} iterate. The stopping criterion in our code is a relative error test, based on the correction δ^k. It reads

$$\max_i\{\max_j\{\frac{|\delta^k_{i,j}|}{w_{i,j}}\}\} < 1, \tag{3.11}$$
$$w_{i,j} = 10^{-3} \min\{TOLT^2, TOLS\} \max\{|U^k_{i,j}|, 10^{-2}\, scale(i)\}.$$

The lower index i refers to the i^{th} PDE solution component and j to the nodes of the current grid level. Since the accuracy of computed solutions increase when *TOLT* or *TOLS* decrease, it is natural to let the stopping criterion depend on these tolerances. We have used $TOLT^2$ here because the time error behaves like $O(\Delta t^2)$ while the time error monitor value *ESTT* from (3.4) only behaves like $O(\Delta t)$ as the time stepsize $\Delta t \to 0$. For this reason the stopping criterion should depend quadratically on *TOLT*. The order of convergence of the space error monitor (3.1), however, is in agreement with the order of the space discretization so there is only a linear dependence on *TOLS*. The linear system (3.10) is solved by ILU preconditioned Bi-CGSTAB. This iterative solver was obtained by modifying the public domain CGS code from the SLAP library written by Anne Greenbaum and Mark K. Seager which is available from netlib [4]. Also some alterations were made to the stopping criterion of this code to better prepare it for solving subsequent iterations in a Newton process [2]. For example, the stopping criterion in the netlib code is relative to the righthand side vector of (3.10) which vanishes in a Newton iteration. After l Bi-CGSTAB iterations we have

$$J(U^0)\delta^{k,l} = -F(U^{k-1}) + r^l, \tag{3.12}$$

where r^l is the residue of the Bi-CGSTAB process and $\delta^{k,l}$, the approximation of δ^k after l iterations. Following [2], the stopping criterion now reads

$$\max_i\{\max_j\{\frac{|(K^{-1}r^l)_{i,j}|}{w_{i,j}}\}\} < \frac{1}{20 \times \text{maximum \# Newton iterations}}, \tag{3.13}$$

where K is the ILU decomposition of the Jacobian matrix.

Suppose we solve a system of time-dependent nonlinear PDEs on a single space grid. In this case, the standard modified Newton procedure for solving a system of nonlinear algebraic equations stemming from such a system of PDEs would be, first compute a Jacobian at the beginning of a time step, then iterate and when the iteration fails to converge, start again with a smaller time stepsize. This is a good approach for the problem above because when the time stepsize decreases, the solution of the nonlinear equations will be closer to the solution at the beginning of the time step which is used as initial guess for the iteration process. This procedure, however, doesn't always work in case a system of PDEs is solved with the local uniform grid refinement method. In this adaptive grid method, the local subgrids can move or grow in space or be newly created. This means that it frequently happens that the initial values for an initial-boundary value problem, defined on a subgrid, have to be interpolated from a next coarser subgrid. When the subgrid is newly created, this interpolation takes place over the whole subgrid and when the subgrid moves or grows, interpolation only takes place over the part of the subgrid which did not exist at the backward time point. Although the refinement strategy attempts to create subgrids in such a way that interpolation only takes place in regions where the solution is smooth, it can happen that the interpolated initial values are not accurate enough. If this is the case, then it's possible that the Newton iteration does not converge because of this, since, these (partially) interpolated initial solution is used as initial guess for the iteration as well as for computing the Jacobian. When solving brine transport problems in porous media with inhomogeneities one encounters these problems. For example when a subgrid is newly created or moves in space from one time point to the next, initial values for the new fine subgrid cells are interpolated from the next coarser subgrid solution which may be kinked. This way initial data is obtained which doesn't look like the fine grid solution to the PDEs at the backward time point. To our experience, when the Newton iteration fails to converge, time stepsize reduction works very poorly, or not at all, in such a case. For this reason the modified Newton procedure has to be adapted. The adaptations we have made will now be elaborated.

The solution at the backward time point is taken as the initial guess for the finest grid level. With respect to the initial guess for the coarser grid levels, we note that, in spite of the fact that injection of fine grid values in coinciding nodes improves the accuracy of the solution at the coarser grid level (cf. Section 2, step 8), the updated coarser grid solution is usually not a very good initial guess for the solution at this grid at the

future time point. Therefore, we also keep the original, not-updated solution at the backward time point in storage which is used as initial guess for the next time step. After this, the linear system (3.10) is generated and iteratively solved. In case this iteration process terminates unsuccessfully, (3.10) is generated all over again, employing a smaller time stepsize. When (3.10) is solved at least twice (i.e. after two nonlinear iterations), we check for convergence and convergence speed as follows. Let *itmax* be the specified maximum number of nonlinear iterations, k the number of completed iterations and let $errit_k$ be defined as

$$errit_k = \max_i\{\max_j\{\frac{|\delta_{i,j}^k|}{w_{i,j}}\}\}. \tag{3.14}$$

The convergence rate is now defined as $errit_k/errit_{k-1}$. Assuming linear convergence of the iteration process, i.e. the convergence rate does not tend to zero even if $k\rightarrow\infty$, the value of $errit_{itmax}$, the value of *errit* after the maximum number of iterations have been performed, can then be approximated as

$$errit_{itmax} \lesssim errit_k\,(\frac{errit_k}{errit_{k-1}})^{itmax-k}. \tag{3.15}$$

The convergence/convergence speed criterion now reads $errit_{itmax} < 1$. When this criterion is satisfied, (3.11) is expected to be fulfilled after *itmax* iterations. Note that this criterion terminates a diverging as well as a slowly converging iteration process. When the convergence/convergence speed criterion is fulfilled, we check if the stopping criterion (3.11) is satisfied. If this is the case then we are finished, otherwise we proceed with the next iteration. In case that the convergence/convergence speed criterion is not satisfied, a new Jacobian is computed. There are two ways to compute a new Jacobian. First, the Jacobian can be computed using the previous iterate U^{k-1} as initial guess and employing the same time stepsize. Second, we can compute the new Jacobian using the original initial guess U^0 with a reduced time stepsize, just like in the standard modified Newton approach. How the Jacobian is going to be calculated depends on a number of criteria. First, the number of new Jacobians with the same time stepsize during the whole iteration process is limited to a user defined maximum. If this maximum is reached then the new Jacobian is computed with a reduced time stepsize. When a new Jacobian with the same time stepsize was already obtained during the previous iteration and the convergence/convergence speed criterion is still not satisfied, the new Jacobian is also calculated with a smaller time stepsize. Suppose that the last iteration where a new Jacobian was computed with the same time stepsize is denoted by j. We assume that when $\|F(U^{k-1})\|_\infty < \|F(U^{j-1})\|_\infty$, the iterate U^{k-1} is a "better" solution to (3.9) than U^{j-1}. A new Jacobian is only computed with the same time stepsize if this is the case, and computed with a reduced time stepsize, otherwise.

This algorithm is more complicated than standard modified Newton. Its behaviour ranges from standard modified Newton to a genuine Newton-Raphson process. The idea behind it is that when the convergence criteria are not fulfilled, the iteration is not immediately repeated with a smaller time stepsize, like in the standard modified Newton approach, but a new Jacobian is tried first, based on the last accepted iterate and with the same time stepsize. Should this fail too, then the iteration is repeated with a smaller time stepsize. The maximum number of Newton iterations and Jacobians with the same time stepsize in our code are chosen to be 10 and 5 respectively. When the time stepsize needs to be decreased, we take the new stepsize to be ¼ × the previous one. Formulas like (3.6) are contained in subroutine JAC, (3.7) is programmed in the subroutine pertrb and (3.10)-(3.15) and the maximum number of Newton iterations and Jacobians can be found in the subroutine INTGRT.

4. Model of Brine Transport in Porous Media and Interface Conditions

In this section we give an explanation of the interface conditions which should be implemented in case the porous medium is inhomogeneous. Therefore, this section applies to MOORKOP 2.1 only. First, we discuss a model of brine transport in porous media and after this we describe the interface conditions which are closely connected with this particular model. The brine transport model and the interface conditions are only used here as an example, so the user is by no means compelled to use the brine transport model and interface conditions, given in this section. The mathematical model can be altered in any way as long as it fits into format (1.1). Changes can also be made to the interface conditions as long as the discretized form of these conditions uses a five-point difference stencil and the dependent variables like the pressure, the concentration of a solute or the temperature are continuous.

4.1. Model of Brine Transport in Porous Media

Following [13] we consider a model for unsteady, isothermal, single-phase, two-component saturated flow in a porous medium in two space dimensions. This model contains two conservation laws, namely one for the mass of the total fluid, i.e. water and salt and one for the mass of salt only. The mass conservation of the total fluid supplemented with Darcy's law for the velocity field is given by

$$\frac{\partial}{\partial t}(n\rho) + \nabla.(\rho \mathbf{q}) = 0, \quad \mathbf{q} = -\frac{k}{\mu}(\nabla p - \rho \mathbf{g}), \tag{4.1}$$

where n is the porosity of the porous medium, ρ is the mass density and $\mathbf{q}$ the velocity vector of the total fluid. The permeability of the porous medium is denoted by k, μ is the dynamic viscosity, p the pressure and $\mathbf{g}$ the acceleration of gravity vector. The mass conservation law of salt and Fick's law for the dispersive mass fluxes are given by

$$\frac{\partial}{\partial t}(n\rho\omega) + \nabla.(\rho\omega\mathbf{q} + \rho\mathbf{J}) = 0, \quad \mathbf{J} = -n\mathbb{D}\nabla\omega, \tag{4.2}$$

respectively, where ω is the concentration of salt and $\mathbf{J}$ the dispersive mass flux vector. $\mathbb{D}$ is the 2×2 dispersion tensor defined by

$$n\mathbb{D} = (nd_m + \alpha_t|\mathbf{q}|)\mathbb{I} + (\alpha_l - \alpha_t)\frac{\mathbf{q}\mathbf{q}^T}{|\mathbf{q}|}, \quad |\mathbf{q}| = (\mathbf{q}^T\mathbf{q})^{1/2}, \tag{4.3}$$

where α_l denotes the longitudinal and α_t the transversal dispersivity and d_m the molecular diffusion. $\mathbb{I}$ is the 2×2 identity matrix. The soil parameters in this model are n, d_m, α_l, α_t and k. They can assume different values in different porous media. Temperature and compressibility effects are neglected in this model, as well as sources, sinks and deformation of the porous medium. To complete the model we have an equation of state for the fluid mass density ρ and a polynomial expression for the dynamic viscosity μ which depends on the concentration of salt:

$$\rho = \rho_0 \exp(\gamma\omega), \tag{4.4}$$

$$\mu = \mu_0(1 + 1.85\omega - 4.10\omega^2 + 44.50\omega^3), \tag{4.5}$$

where ρ_0 and μ_0 are the reference density and dynamic viscosity and γ is a coefficient obtained from

laboratory experiments.

In cases of a low salt concentration (4.1) and (4.2) are only weakly coupled and can be solved independently. The flow can then be regarded as independent from the density gradients caused by differences in the salt concentration since these gradients prove to be negligible. However, we consider cases of high salt concentration, in which case the flow is no longer independent from the density gradients, so these equations should be solved simultaneously. With this model we have followed [5] in the description of brine transport, except for Darcy's law and Fick's law. In this paper these laws are used in their classical formulation, valid for low concentration cases.

Using p and ω as independent variables, we have discretized the equations (4.1), (4.2) in the form

$$\begin{aligned} &-\gamma\nabla.\,\mathbf{J} - \gamma^2\mathbf{J}.\,\nabla\omega + \nabla.\,\mathbf{q} = 0, \\ &n\frac{\partial\omega}{\partial t} + \mathbf{q}.\,\nabla\omega + \gamma\mathbf{J}.\,\nabla\omega + \nabla.\,\mathbf{J} = 0, \end{aligned} \tag{4.6}$$

which is obtained after some elementary calculations. At this stage we note that this model fits into format (1.1) and can, within the limits of (1.1), be modified by the user of the code, like, for example, by adding a temperature equation or by using different formulations for Darcy's law and Fick's law or by adding compressibility effects, etcetera.

4.2. Interface Conditions

The soil parameters in porous media can show abrupt changes from one region to another. Moreover, across these interfaces, i.e. there where the sudden changes occur, p and ω are continuous but their profiles may be kinked. We will assume that, mathematically, the soil parameters are piecewise constant functions and that p and ω are both continuous functions, not differentiable in space at an interface. This means that in order to get consistent numerical approximations, we have to take care that numerical differentiation does not take place across such an interface. Therefore, the numerical solution at an interface is obtained by fulfilling interface conditions which connect the solution on both sides of the interface and involve only one-sided difference schemes. Since (4.1) and (4.2) represent two conservation laws, it is natural to impose continuity of the spatial fluxes $\rho\mathbf{q}.\,\mathbf{n}$ and $(\rho\omega\mathbf{q} + \rho\mathbf{J}).\,\mathbf{n}$ at interfaces as interface conditions, where $\mathbf{n}$ is a vector locally perpendicular to the interface. It suffices to impose continuity of $\mathbf{q}.\,\mathbf{n}$ and $\mathbf{J}.\,\mathbf{n}$, since ρ and ω are both continuous functions.

Consider the four grid cells shown in Figure. 4.1, numbered I till IV, possessing cell edges, parallel to the co-ordinate axes. First, the soil parameters are evaluated in all cell centers and are supposed to be constant over each cell. Hence, the interfaces are assumed to coincide with cell edges in the numerical approximation. When the soil parameters are constant over these four cells then none of these cells are intersected by an interface and (4.6) is discretized at grid node C using the standard second order finite differences in space. Now suppose that, for example, the soil parameters in CI are different from those in CII. Then the component of $\mathbf{q}$ and $\mathbf{J}$ in x-direction which is perpendicular to the cell edge, separating the upper left cell I from the upper right cell II, must be continuous. This cell edge is denoted as CN. Due to (4.1)-(4.3) we have,

$$\begin{aligned} q_1 &= -\frac{k}{\mu}(p_x - \rho g_1), \\ q_2 &= -\frac{k}{\mu}(p_y - \rho g_2), \\ J_1 &= -nD_{11}\omega_x - nD_{12}\omega_y, \\ nD_{11} &= nd_m + \alpha_t|\mathbf{q}| + (\alpha_l - \alpha_t)\frac{q_1^2}{|\mathbf{q}|}, \end{aligned} \tag{4.7}$$

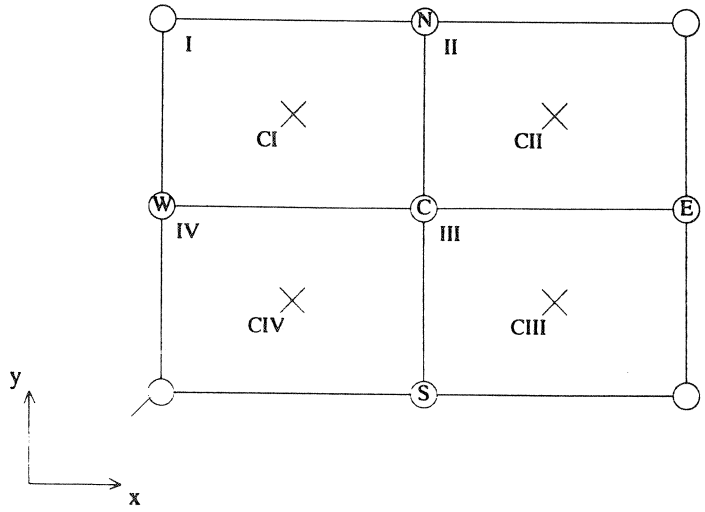

FIGURE 4.1 Four arbitrary grid cells with cell edges, parallel to the co-ordinate axes.

$$nD_{12} = (\alpha_l - \alpha_t)\frac{q_1 q_2}{|\mathbf{q}|},$$

where q_1, J_1 and g_1 are the components in x-direction of $\mathbf{q}$, $\mathbf{J}$ and $\mathbf{g}$, respectively and nD_{11} and nD_{12} are elements of the first row of the dispersion tensor $n\mathbf{D}$; q_2 and g_2 are the components in y-direction of $\mathbf{q}$ and $\mathbf{g}$. The derivatives of (4.7) are discretized using the grid nodes N, S, E, W and C, which yields a first order accurate discretization. The fluxes q_1 and J_1 on the left and righthand side of CN are denoted as $q_{1,CN,I}$, $J_{1,CN,I}$ and $q_{1,CN,II}, J_{1,CN,II}$ respectively. They are now approximated as

$$\begin{aligned}
q_{1,CN,I} &= -\frac{k_{CI}}{\mu_C}\left(\frac{p_C - p_W}{\Delta x} - \rho_C g_1\right),\\
J_{1,CN,I} &= -nD_{11,CN,I}\frac{\omega_C - \omega_W}{\Delta x} - nD_{12,CN,I}\frac{\omega_N - \omega_C}{\Delta y},\\
q_{1,CN,II} &= -\frac{k_{CII}}{\mu_C}\left(\frac{p_E - p_C}{\Delta x} - \rho_C g_1\right),\\
J_{1,CN,II} &= -nD_{11,CN,II}\frac{\omega_E - \omega_C}{\Delta x} - nD_{12,CN,II}\frac{\omega_N - \omega_C}{\Delta y},
\end{aligned} \tag{4.8}$$

where

$$\begin{aligned}
nD_{11,CN,I} &= nd_{m,CI} + \alpha_{t,CI}|\mathbf{q}_{CN,I}| + (\alpha_{l,CI} - \alpha_{t,CI})\frac{q_{1,CN,I}^2}{|\mathbf{q}_{CN,I}|},\\
nD_{12,CN,I} &= (\alpha_{l,CI} - \alpha_{t,CI})\frac{q_{1,CN,I}\, q_{2,CN,I}}{|\mathbf{q}_{CN,I}|},
\end{aligned}$$

$$nD_{11,CN,II} = nd_{m,CII} + \alpha_{t,CII}|q_{CN,II}| + (\alpha_{l,CII} - \alpha_{t,CII})\frac{q_{1,CN,II}^2}{|q_{CN,II}|}, \tag{4.9}$$

$$nD_{12,CN,II} = (\alpha_{l,CII} - \alpha_{t,CII})\frac{q_{1,CN,II}\, q_{2,CN,II}}{|q_{CN,II}|},$$

$$q_{2,CN,I} = -\frac{k_{CI}}{\mu_C}\left(\frac{p_N - p_C}{\Delta y} - \rho_C g_2\right),$$

$$q_{2,CN,II} = -\frac{k_{CII}}{\mu_C}\left(\frac{p_N - p_C}{\Delta y} - \rho_C g_2\right),$$

$$|q_{CN,I}| = (q_{1,CN,I}^2 + q_{2,CN,I}^2)^{1/2}, \qquad |q_{CN,II}| = (q_{1,CN,II}^2 + q_{2,CN,II}^2)^{1/2}.$$

Here $q_{2,CN,I}$, $q_{2,CN,II}$ represent velocities parallel to CN and $nD_{11,CN,I}$, $nD_{12,CN,I}$, $nD_{11,CN,II}$, $nD_{12,CN,II}$ the elements of the first row of the dispersion tensor, on each side of CN. Constants like k_{CII} and $\alpha_{l,CII}$ denote the permeability and longitudinal dispersivity at cell II and entries like, for example, ρ_C and μ_C are the mass density and the dynamic viscosity in C. Continuity of $\mathbf{q}.\,\mathbf{n}$ and $\mathbf{J}.\,\mathbf{n}$ across CN yields the following system of flux continuity equations for p and ω in C

$$\begin{aligned} q_{1,CN,I} - q_{1,CN,II} &= 0, \\ J_{1,CN,I} - J_{1,CN,II} &= 0. \end{aligned} \tag{4.10}$$

When not only CN is an interface but also CW, CE or CS then the flux continuity equations are generated for each interface. The equations we then solve is the sum of these flux continuity equations. Care must be taken in adding these equations because in some circumstances the unknowns p_C and ω_C can drop out of the system of equations. In the next section we describe how the user delivers these equations to MOORKOP.

5. EXECUTING THE PROGRAM

In this section we will describe how a user delivers a PDE problem to MOORKOP. In order to do this the user merely has to alter the main program, called *moorkop* and the two subroutines *PDE* and *Uinit*. When MOORKOP 2.1 is used, the user has to supply the discontinuous soil parameters in the subroutine *SOIL* also. When starting up the program, the code asks for additional information. This will also be discussed here. Finally, we note that MOORKOP is written in double precision.

5.1. Program Output

Both versions of MOORKOP create a file called *RUNINF* while running. This file contains information about the problem, the total number of accepted and rejected time steps and the generated error message in case of an error exit. If desired, this file can also contain information like error monitor values, the number of nodes at a grid level, time stepsizes etcetera for each time level. An example of such information associated with one time step taken with two grid levels is given below

```
time level nodes stepsize
 0.32000E-01 1   441 0.16000E-01
iteration correction residual # lin. iter.
 1 0.46009E+05 0.19757E-04     18
 2 0.12769E+02 0.39407E-06      3
 3 0.26926E+00 0.79180E-07      1
max_monitor_value  0.88937E+00
time level nodes stepsize
 0.32000E-01 2   778 0.16000E-01
iteration correction residual # lin. iter.
 1 0.46009E+05 0.40313E-04     11
 2 0.92169E+01 0.93103E-06      3
 3 0.22101E+00 0.53928E-07      2
max_monitor_value  0.67261E+00
level max_time-error
 1 0.32321E-05
 2 0.12066E-01
```

The first row of our example contains information about the current time level, the current grid level, the number of grid points (nodes) at that grid level and the time stepsize. In this case the current time level is .032, the current grid level is 1 which is the coarsest grid, the number of grid points on grid level 1 is 441 and the time stepsize is .016. Then we get four columns containing information about the solution process of the system of nonlinear equations. The first column contains the current Newton iteration k, the second column the value of $errit_k$, the third holds $\|F(U^k)\|_\infty$ and the fourth one shows the number of Bi-CGSTAB iterations. Then we get the row with the maximum monitor value, by which we mean the $\max_i\{ESTSmax_i\}$. After this we get a similar block of information about grid level 2 which is the next finer grid level. The information is completed with the final two columns holding the number of a grid level and its associated value of *ESTT*.

If desired, MOORKOP can create files named FILE001, FILE002, and so on which hold information about the data structure and the computed solution at all gridlevels in use, the time stepsize etcetera. These files can be used to plot or print the solution or for restarting the program. Moreover, MOORKOP2.1 can create files containing the velocity field in a porous medium. These files are called VLCT001, VLCT002, etcetera.

5.2. Program Alterations

program moorkop:

The following constants have to be supplied by the user:

npde	INTEGER: number of PDEs to be solved.
nptspl	INTEGER: maximum number of grid points allowed at each grid level.
maxlev	INTEGER: maximum number of grid levels allowed.
ntimes	INTEGER: number of time levels at which output is generated +1.
npar	INTEGER: number of user defined parameters.
nsoil	INTEGER: number of soil parameters which are discontinuous in the domain. (*only MOORKOP 2.1*)
licn	INTEGER: maximum storage needed for the sparse matrix solver Bi-CGSTAB.
outime(.)	DOUBLE PRECISION: array of length *ntimes* containing time levels. *outime*(1) is the initial time and *outime*(*ntimes*) is the final time; *outime*(2), $\cdots$, *outime*(*ntimes*) are time levels at which output is created when desired.
SPCest(.)	LOGICAL: array of length *npde* specifying the solution components for which the space error monitor should be evaluated. In general it is recommended to choose SPCest(.)=true for all components.
TMest(.)	LOGICAL: array of length *npde* specifying the solution components for which the time error monitor should be evaluated. If system (1.1) does not contain the temporal derivative of component *i* then we should choose TMest(i)=false and TMest(i)=true, otherwise.
check	LOGICAL: constant specifying whether the time error monitor should be evaluated after the first time step. In case of an inconsistency between boundary values and initial values, *check* = false, otherwise *check* = true. Putting *check* equal to "false" might help the code to overcome the inconsistency, but there is no guarantee.
prevfl	LOGICAL: constant specifying whether a finer mesh should always be created at interfaces or not. (*only MOORKOP 2.1*)
scale(.)	DOUBLE PRECISION: array of length *npde* containing typical values of the solution components. For example, the user can enter the largest value or the difference between the largest and the smallest value of a solution component.
par(.)	DOUBLE PRECISION: array of length *npar* holding user supplied parameters connected with the PDE problem.
PROBLEM	CHARACTER: character of length 80 containing information about the PDE.
Xmin,Xmax,Ymin,Ymax	DOUBLE PRECISION: constants specifying the boundaries of the space domain.

subroutine SOIL: (*only MOORKOP 2.1*)

The soil parameters which are discontinuous over the domain have to be supplied by the user in this subroutine. This will be done using the array

SP(.)	DOUBLE PRECISION: array containing discontinuous soil parameters.

Suppose that we have *nsoil* discontinuous soil parameters. Then the i^{th} parameter is defined by

$$SP(jj+(i-1)*nptspl) = \ldots, \quad i=1,\ldots,nsoil.$$

The index *jj* corresponds with a grid point and SP(*jj*+(*i*−1)**nptspl*) holds the value of a soil parameter at the center of the cell of which grid point *jj* is the lower left vertex.

Suppose we have a circular interface defined by

$$c = (x-0.5)^2 + (y-0.5)^2-0.2;$$

and let the parameters called k, α_l and α_t be given by

$$k = 10^{-10}\, m^2, \quad c>0, \quad k = 10^{-13}\, m^2, \quad c \le 0,$$
$$\alpha_l = 0.01\, m, \quad c>0, \quad \alpha_l = 0.005\, m, \quad c \le 0,$$
$$\alpha_t = 0.002\, m, \quad c>0, \quad \alpha_t = 0.001\, m, \quad c \le 0,$$

The resulting FORTRAN text in subroutine SOIL can, for example, read

```
c=(x-.5)**2+(y-.5)**2-.2
if (c.gt.0) then
  SP(jj)=1.0d-10
  SP(jj+nptspl)=.01
  SP(jj+2*nptspl)=.002
else
  SP(jj)=1.0d-13
  SP(jj+nptspl)=.005
  SP(jj+2*nptspl)=.001
end if
```

In this example, SP(*jj*), SP(*jj*+*nptspl*) and SP(*jj*+2**nptspl*) contain the values of k, α_l and α_t, respectively. It should be noted that the program detects interfaces by checking values of only SP(*jj*) at the center of the four cells surrounding a grid-point in the interior of the domain (see Figure 4.1). This means that the soil parameter stored at SP(*jj*) should always be discontinuous at all interfaces.

subroutine PDE:

With this subroutine the user delivers the system of partial differential equations (PDEs) and boundary conditions (BCs) to MOORKOP. A system of the form

$$G(x,y,t,u,u_t,u_x,u_y,u_{xx},u_{xy},u_{yy}) = 0,$$

with the BCs

$$H(x,y,t,u,u_t,u_x,u_y) = 0,$$

where G, H and u and its derivatives are vectors with length *npde*, is allowed.

The part of the subroutine *PDE* which is meant for user alteration is clearly indicated in the examples of Section 7. The problem definition by the user is carried out with the two main arrays

ipF(.)	INTEGER: array of length *nptspl* specifying the location of a grid point in the domain. This array need not be specified by the user.
F(.)	DOUBLE PRECISION: array containing the residue of the PDEs and BCs,

and the auxiliary arrays

U(.)	DOUBLE PRECISION: array containing the solution vector u,
Ut(.)	DOUBLE PRECISION: array containing u_t,
Ux(.)	DOUBLE PRECISION: array containing u_x,
Uy(.)	DOUBLE PRECISION: array containing u_y,
UxR(.)	DOUBLE PRECISION: array containing u_x, (*only MOORKOP2.1*)

UyB(.)	DOUBLE PRECISION: array containing u_y, (*only MOORKOP2.1*)
Uxx(.)	DOUBLE PRECISION: array containing u_{xx},
Uxy(.)	DOUBLE PRECISION: array containing u_{xy},
Uyy(.)	DOUBLE PRECISION: array containing u_{yy}.

The user has to supply the array F only. The vector Ut is computed using (3.2). For computing Ux and Uy we use second order central finite differences when possible and an appropriate second order (MOORKOP 2.0) or first order (MOORKOP 2.1) one-sided difference scheme otherwise. When a grid point lies at an interface (MOORKOP 2.1), first order one-sided difference schemes are used. In that case we have to keep two values in storage for both u_x and u_y, since u_x and u_y are supposed to be discontinuous at an interface. The one-sided approximation of u_x on the left is held by Ux and the one on the right by UxR and the downwards and upwards one-sided difference approximation of u_y are contained in Uy and UyB respectively. In both versions of MOORKOP, we use the second order central three-point difference scheme for u_{xx} and u_{yy} and the four-point scheme for u_{xy}. With these arrays the user can rewrite his PDE or BC in FORTRAN, using the conventions

ipF(.)=0	The grid point lies in the interior,
ipF(.)=2	The grid point lies on the lower boundary,
ipF(.)=3	The grid point lies on the upper boundary,
ipF(.)=5	The grid point lies on the left boundary,
ipF(.)=6	The grid point lies in the lower left corner,
ipF(.)=7	The grid point lies in the upper left corner,
ipF(.)=9	The grid point lies on the right boundary,
ipF(.)=10	The grid point lies in the lower right corner,
ipF(.)=11	The grid point lies in the upper right corner,
ipF(.)=12	The grid point lies at an interface, (*only MOORKOP 2.1*)

and

F(idx+i)	contains the i^{th} residual value of the PDE or BC at a certain grid point. The value *idx*, which need not be specified by the user, is the offset in F for the residual values at this grid point. The auxiliary arrays are subscripted in the same way.

Here we note that ipF(.)=12 occurs only at interior grid points.

As an example we consider Burger's equation

$$u_t = -uu_x - vu_y + \varepsilon(u_{xx} + u_{yy}),$$
$$v_t = -uv_x - vv_y + \varepsilon(v_{xx} + v_{yy}),$$
$$\text{Xmin} < x < \text{Xmax}, \quad \text{Ymin} < y < \text{Ymax}, \quad t > t_0.$$

Its FORTRAN definition simply reads

```
c-----------------------------------------------------------------------
c The specification of the PDEs at the interior of the domain
c-----------------------------------------------------------------------
          if (ipF(jj).eq.0) then
            F(idx+1)=
     +             Ut(idx+1)+U(idx+1)*Ux(idx+1)+U(idx+2)*Uy(idx+1)
     +            -par(1)*(Uxx(idx+1)+Uyy(idx+1))
            F(idx+2)=
     +             Ut(idx+2)+U(idx+1)*Ux(idx+2)+U(idx+2)*Uy(idx+2)
     +            -par(1)*(Uxx(idx+2)+Uyy(idx+2))
c
```

where ε is contained in *par*(1). The subscript *jj* refers to a particular grid point. Further we have a homogeneous Neumann boundary condition for the first and a Dirichlet boundary condition for the second component of a PDE solution on the right boundary

$$u_x = 0,$$

$$v = 1,$$

$$x = \text{Xmax}, \quad \text{Ymin} < y < \text{Ymax}, \quad t > t_0.$$

This results in the FORTRAN definition

```
          else if (ipF(jj).eq.9) then
c-----------------------------------------------------------------------
c The specification of the BCs at the right boundary of the domain
c-----------------------------------------------------------------------
              F(idx+1)=Ux(idx+1)
              F(idx+2)=U(idx+2)-1.0
c
```

See Section 6 for a complete PDE definition. We note both subscripts *jj* and *idx* are of no concern to the user and that the user has to specify BCs at all four sides and corners of the domain.

Finally, we show how interface conditions should be programmed. We use the model and the interface conditions from Section 4. First, we have to compute the fluxes at both sides of the cell faces which connect the cells, surrounding the interface node under consideration (see Figure 4.1). The vector components Ux(idx+1), UxR(idx+1), Uy(idx+1) and UyB(idx+1), that we will use, are the one-sided derivatives of the pressure p, $(p_C - p_W)/\Delta x$, $(p_E - p_C)/\Delta x$, $(p_C - p_S)/\Delta y$ and $(p_N - p_C)/\Delta y$, respectively. Further, the vector components Ux(idx+2), UxR(idx+2), Uy(idx+2) and UyB(idx+2) denote the same derivatives of the salt concentration ω. Note that the values of the i^{th} soil parameter at the center of the cell at the upper right, upper left, lower right and lower left of the grid point associated with index *jj* are stored at SP(*jj*+(*i*−1)**nptspl*), SP(*jj*−1+(*i*−1)**nptspl*), SP(iq(*jj*)+(*i*−1)**nptspl*) and SP(iq(*jj*)−1+(*i*−1)**nptspl*), respectively. Here, the array element iq(*jj*) is the index associated with the grid point below grid point *jj* and *jj*−1 is the index belonging to the grid point left of grid point *jj*. Using the notation of Section 4, the fluxes at cell I are computed by the FORTRAN text.

```
c
c Computation of the fluxes at upper left cell
c
              K=SP(jj-1)
              al=SP(jj-1+nptspl)
              at=SP(jj-1+2*nptspl)
              q1nw=-K*Ux(idx+1)/mu
              q2nw=-K*(UyB(idx+1)+rho*g)/mu
              aq=dmax1(dsqrt(q1nw*q1nw+q2nw*q2nw),1.0d-20)
              nD11nw=n*dm+at*aq+(al-at)*q1nw*q1nw/aq
              nD12nw=(al-at)*q1nw*q2nw/aq
              nD22nw=n*dm+at*aq+(al-at)*q2nw*q2nw/aq
              Jw1nw=-nD11nw*Ux(idx+2)-nD12nw*UyB(idx+2)
              Jw2nw=-nD12nw*Ux(idx+2)-nD22nw*UyB(idx+2)
```

Here K, *al* and *at* are the soil parameters at the center of the cell under consideration which is cell I in this case. They then denote k_{CI}, $\alpha_{l,CI}$ and $\alpha_{t,CI}$; *mu* is μ_C; *q1nw* and *q2nw* represent the velocities $q_{1,CN,I}$ and $q_{2,CN,I}$ at cell I which are perpendicular and tangential to cell face CN, respectively. The entry *aq* denotes $|q_{CN,I}|$ and *nD11nw*, *nD12nw* and *nD22nw* are the elements $nD_{11,CN,I}$, $nD_{12,CN,I}$ and $nD_{22,CN,I}$ of the dispersion tensor at cell I. Finally, *Jw1nw* and *Jw2nw* are the dispersive mass fluxes $J_{1,CN,I}$ and $J_{2,CN,I}$ at cell I, perpendicular and tangential to cell face CN, respectively. For the fluxes at the remaining cells we have

```
c
c Computation of the fluxes at lower left cell
c
              K=SP(iq(jj)-1)
              al=SP(iq(jj)-1+nptspl)
              at=SP(iq(jj)-1+2*nptspl)
              q1sw=-K*Ux(idx+1)/mu
              q2sw=-K*(Uy(idx+1)+rho*g)/mu
              aq=dmax1(dsqrt(q1sw*q1sw+q2sw*q2sw),1.0d-20)
              nD11sw=n*dm+at*aq+(al-at)*q1sw*q1sw/aq
              nD12sw=(al-at)*q1sw*q2sw/aq
              nD22sw=n*dm+at*aq+(al-at)*q2sw*q2sw/aq
              Jw1sw=-nD11sw*Ux(idx+2)-nD12sw*Uy(idx+2)
              Jw2sw=-nD12sw*Ux(idx+2)-nD22sw*Uy(idx+2)
```

```
c
c Computation of the fluxes at upper right cell
c
              K=SP(jj)
              al=SP(jj+nptspl)
              at=SP(jj+2*nptspl)
              q1ne=-K*UxR(idx+1)/mu
              q2ne=-K*(UyB(idx+1)+rho*g)/mu
              aq=dmax1(dsqrt(q1ne*q1ne+q2ne*q2ne),1.0d-20)
              nD11ne=n*dm+at*aq+(al-at)*q1ne*q1ne/aq
              nD12ne=(al-at)*q1ne*q2ne/aq
              nD22ne=n*dm+at*aq+(al-at)*q2ne*q2ne/aq
              Jw1ne=-nD11ne*UxR(idx+2)-nD12ne*UyB(idx+2)
              Jw2ne=-nD12ne*UxR(idx+2)-nD22ne*UyB(idx+2)
c
c Computation of the fluxes at lower right cell
c
              K=SP(iq(jj))
              al=SP(iq(jj)+nptspl)
              at=SP(iq(jj)+2*nptspl)
              q1se=-K*UxR(idx+1)/mu
              q2se=-K*(Uy(idx+1)+rho*g)/mu
              aq=dmax1(dsqrt(q1se*q1se+q2se*q2se),1.0d-20)
              nD11se=n*dm+at*aq+(al-at)*q1se*q1se/aq
              nD12se=(al-at)*q1se*q2se/aq
              nD22se=n*dm+at*aq+(al-at)*q2se*q2se/aq
              Jw1se=-nD11se*UxR(idx+2)-nD12se*Uy(idx+2)
              Jw2se=-nD12se*UxR(idx+2)-nD22se*Uy(idx+2)
```

The flux continuity equations are now programmed in the following manner

```
c
c The flux continuity equations
c
              F(idx+1)=0.0d0
              F(idx+2)=0.0d0
              if (SP(jj).ne.SP(jj-1)) then
c-----------------------------------------------------------------------
c The cell face separating the upper right from the upper left cell
c is an interface
c-----------------------------------------------------------------------
                F(idx+1)=F(idx+1)+q1ne-q1nw
                F(idx+2)=F(idx+2)+Jw1ne-Jw1nw
              end if
              if (SP(iq(jj)).ne.SP(iq(jj)-1)) then
c-----------------------------------------------------------------------
c The cell face separating the lower right from the lower left cell
c is an interface
c-----------------------------------------------------------------------
                F(idx+1)=F(idx+1)+q1se-q1sw
                F(idx+2)=F(idx+2)+Jw1se-Jw1sw
              end if
              if (SP(jj).ne.SP(iq(jj))) then
c-----------------------------------------------------------------------
c The cell face separating the upper right from the lower right cell
c is an interface
c-----------------------------------------------------------------------
                F(idx+1)=F(idx+1)+q2ne-q2se
                F(idx+2)=F(idx+2)+Jw2ne-Jw2se
              end if
              if (SP(jj-1).ne.SP(iq(jj)-1)) then
c-----------------------------------------------------------------------
c The cell face separating the upper left from the lower left cell
c is an interface
c-----------------------------------------------------------------------
                F(idx+1)=F(idx+1)+q2nw-q2sw
                F(idx+2)=F(idx+2)+Jw2nw-Jw2sw
              end if
c=======================================================================
```

We see that the resulting system of equations is the sum of flux continuity equations at each cell face which is also an interface.

subroutine Uinit:

In this subroutine the user has to supply the initial values (IVs).

Any initial value expression

$$u = f_0(x,y).$$

is allowed. For example, the following IVs

$$u = 0,$$

$$v = sin(x+y),$$

$$\text{Xmin} \le x \le \text{Xmax}, \quad \text{Ymin} \le y \le \text{Ymax}, \quad t = t_0.$$

result in the FORTRAN text

```
c
c-------------------------------------------------------------------------
c The specification of the IVs     *
c-------------------------------------------------------------------------
          U(idx+1)=0.0
          U(idx+2)=sin(x+y)
c
c=========================================================================
```

5.3. Interactive Input

In this section we discuss the program query.

TOLT =
The program asks for the value of the time error tolerance.

TOLS =
The program asks for the value of the space error tolerance.

read from file ? (y/n)
The user can restart the program from an output file. With this question the program asks the user if this is the case. If the answer is "*y*", the program asks for the name of the file by "*filename ?*" which must be 7 characters long. When the answer is "*n*", the program asks for the initial time stepsize by "*dT* =" and the number of grid cells in both x- and y-direction on the coarsest grid by "*nx* =" and "*ny* =". After this, the code will continue with the questions below.

print info every timestep ? (y/n)
The program asks the user if output must be written to RUNINF after every time step. The output which will then be generated is described in Paragraph 5.1.

files ? (y/n)

The program asks if the user wants to create output files. If the answer is "*y*", the code creates output files named FILE001, FILE002, FILE003, ... corresponding with the times outime(2), outime(3), outime(4) and so on.

velocity file ? (y/n) (only MOORKOP 2.1)
Here the program asks if the user wants to create output files containing velocities in the porous medium. These files are called VLCT001, VLCT002, ... corresponding with the times outime(2), outime(3) etcetera. The velocity field is computed by user supplied formulas in the subroutine PDE.

5.4. Error Exits

MOORKOP contains the following error exits:

ILU dec., IFAIL=?? see documentation JACP
The subroutine JACP that performs the ILU decomposition has generated an error exit. After "*IFAIL* =" an integer number is printed specifying the error message which can be found in the documentation of JACP.

Solv.lin.sys., IFAIL=?? see documentation SSLVI
The subroutine SSLVI that performs the Bi-CGSTAB iterations has generated an error exit. After "*IFAIL* =" an integer number is printed specifying the error message which can be found in the documentation of SSLVI.

the maximum number of nodes per level is exceeded
The maximum number of grid points per level *nptspl* is too small.

the time stepsize is too small
When the time stepsize becomes smaller than $4*uround*\max\{|outime(i-1)|, |outime(i)|\}$ the message above will be generated and the execution of the program aborted. Here the current time level lies between $outime(i-1)$ and $outime(i)$.

The following error messages can be generated when the program starts from an output file:

the number of PDEs is incorrect
The value of the number of PDEs *npde* is not the same as the value read from the output file.

the maximum number of nodes per level is too small
The maximum number of grid points read from the output file is greater than *nptspl*.

the maximum number of levels is too small
The maximum number of grid levels read from the output file is greater than *maxlev*.

6. POSTPROCESSING

The postprocessing is done by the programs *PSOL*, *PVEL*, *PLOTSOL* and *PLOTVEL*. The program *PSOL* reads the output files FILE??? generated by MOORKOP 2.0 or MOORKOP 2.1. When the program runs, it asks for the name of the file by "*filename*?". After the user has typed in the filename which must be 7 characters long, it creates files called FILE???.print containing output like for example the data shown below, belonging to a system of two PDEs.

level	t	y	x	U
1	0.10000E+02	0.00000E+00	0.00000E+00	0.12059E+07
				0.62798E-01
			0.50000E-01	0.12060E+07
				0.25000E+00
			0.10000E+00	0.12060E+07
				0.25000E+00
			0.15000E+00	0.12060E+07
				0.25000E+00
			0.20000E+00	0.12060E+07
				0.25000E+00
			0.25000E+00	0.12060E+07
				0.25000E+00
			0.30000E+00	0.12060E+07
				0.25000E+00
			0.35000E+00	0.12059E+07
				0.25000E+00
			0.40000E+00	0.12059E+07
				0.25000E+00
			0.45000E+00	0.12059E+07
				0.25000E+00
			0.50000E+00	0.12058E+07
				0.25000E+00
			0.55000E+00	0.12056E+07
				0.16416E-02
			0.60000E+00	0.12056E+07
				0.11127E-05
			0.65000E+00	0.12056E+07
				0.33709E-08
			0.70000E+00	0.12056E+07
				0.32286E-08
			0.75000E+00	0.12054E+07
				0.11181E-12
			0.80000E+00	0.12052E+07
				0.26945E-17
			0.85000E+00	0.12051E+07
				0.43583E-22
			0.90000E+00	0.12050E+07
				0.40936E-27
			0.95000E+00	0.12050E+07
				0.13850E-32
			0.10000E+01	0.12050E+07
				0.13850E-32
		0.50000E-01	0.00000E+00	0.12054E+07
				0.65606E-02
			0.50000E-01	0.12054E+07
				0.82069E-02
			0.10000E+00	0.12054E+07
				0.82390E-02
			0.15000E+00	0.12053E+07
				0.82131E-02
			0.20000E+00	0.12053E+07
				0.81738E-02
			0.25000E+00	0.12053E+07
				0.81158E-02
			0.30000E+00	0.12053E+07
				0.80277E-02
			0.35000E+00	0.12053E+07
				0.78855E-02
			0.40000E+00	0.12053E+07
				0.76289E-02
			0.45000E+00	0.12052E+07
				0.70609E-02
			0.50000E+00	0.12052E+07
				0.59523E-02
			0.55000E+00	0.12051E+07
				0.19311E-03
			0.60000E+00	0.12051E+07
				0.23285E-06
			0.65000E+00	0.12051E+07
				0.33709E-08
			0.70000E+00	0.12051E+07
				0.32286E-08
			0.75000E+00	0.12049E+07
				0.11181E-12

The program *PVEL* is similar to *PSOL* and it reads the velocity files VLCT??? generated by MOORKOP 2.1. When the program runs, it asks for the name of the file by "*filename*?". After the user has typed in the filename which must be 7 characters long, it creates files called VLCT???.print containing output like for example the data shown below.

level	t	y	x	q
1	0.10000E+02	0.00000E+00	0.00000E+00	-0.29893E-03
				0.29065E-12
			0.50000E-01	0.14822E-05
				0.10000E-03
			0.10000E+00	0.34168E-05
				0.10000E-03
			0.15000E+00	0.55430E-05
				0.10000E-03
			0.20000E+00	0.82578E-05
				0.10000E-03
			0.25000E+00	0.11824E-04
				0.10000E-03
			0.30000E+00	0.16636E-04
				0.10000E-03
			0.35000E+00	0.23375E-04
				0.10000E-03
			0.40000E+00	0.33486E-04
				0.10000E-03
			0.45000E+00	0.51196E-04
				0.10000E-03
			0.50000E+00	0.18994E-03
				0.10000E-03
			0.55000E+00	0.12206E-03
				-0.45519E-16
			0.60000E+00	0.59644E-04
				0.14279E-15
			0.65000E+00	0.20299E-04
				-0.12787E-15
			0.70000E+00	0.17843E-06
				0.24793E-18
			0.75000E+00	0.33166E-06
				-0.76034E-19
			0.80000E+00	0.27570E-06
				0.00000E+00
			0.85000E+00	0.20757E-06
				0.00000E+00
			0.90000E+00	0.12911E-06
				0.00000E+00
			0.95000E+00	0.43826E-07
				0.00000E+00
			0.10000E+01	0.57008E-13
				0.00000E+00
		0.50000E-01	0.00000E+00	0.00000E+00
				0.19851E-03
			0.50000E-01	0.35738E-05
				0.19711E-03
			0.10000E+00	0.62735E-05
				0.19678E-03
			0.15000E+00	0.10131E-04
				0.19618E-03
			0.20000E+00	0.15108E-04
				0.19533E-03
			0.25000E+00	0.21626E-04
				0.19411E-03
			0.30000E+00	0.30361E-04
				0.19233E-03
			0.35000E+00	0.42415E-04
				0.18950E-03
			0.40000E+00	0.59775E-04
				0.18437E-03
			0.45000E+00	0.86083E-04
				0.17246E-03
			0.50000E+00	0.11788E-03
				0.13105E-03
			0.55000E+00	0.93199E-04
				0.49232E-04
			0.60000E+00	0.53081E-04
				0.27876E-04
			0.65000E+00	0.20299E-04
				0.19942E-04
			0.70000E+00	0.35650E-06
				0.39894E-07
			0.75000E+00	0.33166E-06
				0.24843E-07

The program *PLOTSOL* reads the output files FILE??? generated by MOORKOP 2.0 or MOORKOP 2.1. When the program runs, it asks for the name of the file by "*filename*?". After the user has typed in the filename, it asks for the solution component the user wants to plot by "*which solution component* ?". After the user has typed in the solution component, it plots the solution at the rectangular coarse base grid. *PLOTSOL* makes the following data available; the number of grid cells in x- and y-direction, nx and ny; the x and y co-ordinates of the boundaries of the domain, Xmin, Xmax, Ymin and Ymax; the arrays x(.), y(.) and V(.) of length $(nx+1) \times (ny+1)$ containing the x and y co-ordinate and the desired solution component at each grid point of the rectangular coarse base grid. When the index i ranges from 0 to nx and the index j from 0 to ny then x($j*(nx+1)+i+1$), y($j*(nx+1)+i+1$) and V($j*(nx+1)+i+1$) are the x and y co-ordinate and the solution at grid point (i,j). The user has to supply the program to plot this solution component. The program *PLOTVEL* reads the output files VLCT??? generated by MOORKOP 2.1. When the program runs, it asks for the name of the file by "*filename*?". After the user has typed in the filename, it plots the velocity field at the rectangular coarse base grid. *PLOTVEL* makes the following data available; the number of grid cells in x- and y-direction, nx and ny; the x and y co-ordinates of the boundaries of the domain, Xmin, Xmax, Ymin and Ymax; the arrays x(.), y(.), qx(.) and qy(.) of length $(nx+1) \times (ny+1)$ containing the x and y co-ordinate and the velocity components in x- and y-direction at each grid point of the rectangular coarse base grid. When the index i ranges from 0 to nx and the index j from 0 to ny then x($j*(nx+1)+i+1$), y($j*(nx+1)+i+1$), qx($j*(nx+1)+i+1$) and qy($j*(nx+1)+i+1$) are the x and y co-ordinate and the velocity components in x- and y-direction at grid point (i,j). The user has to supply the program to plot the velocities.

Error exits occuring while running the postprocessing programs generate messages which are the same as the messages that can be generated when MOORKOP starts from an output file. Only *PLOTSOL* has an additional error message. When, for instance, the number of PDEs read from an output file is three and the user wants to plot the fourth solution component then the following message is generated

inappropriate solution component

7. EXAMPLES

7.1. The Heat Equation

The first example is the rotating cone problem due to [1].

7.1.1.. Problem Description

The equation is the linear parabolic equation

$$u_t = u_{xx} + u_{yy} + f(x,y,t), \quad 0<x,y<1, \quad t>0,$$

and the initial function, the Dirichlet boundary conditions and the source f are selected so that the exact solution is

$$u(x,y,t) = exp\,[-80((x - r(t))^2 + (y - s(t))^2)],$$

where $r(t) = \frac{1}{4}[2 + sin(\pi t)]$ and $s(t) = \frac{1}{4}[2 + cos(\pi t)]$. We have used this problem in [10, 12].

7.1.2.. Program Input

MOORKOP 2.0 was used to solve this problem. The sources for this problem of *program moorkop*, *subroutine PDE* and *subroutine Uinit* are given below.

```
      program  moorkop
c
c
c     MOORKOP2.0
c
c
c-----------------------------------------------------------------------
c
c moorkop solves partial differential equations (PDEs) of the type
c
c G(x,y,t,U,Ut,Ux,Uy,Uxx,Uxy,Uyy)=0
c
c with the boundary conditions (BCs)
c
c H(x,y,t,U,Ut,Ux,Uy)=0
c
c and the initial values (IVs)
c
c U=F0(x,y)
c
c on a rectangular domain.
c
c moorkop is designed to solve PDEs with solutions possessing steep
c moving transitions in space and time, using Local Unform Grid
c Refinement. This adaptive grid technique is described in the
c CWI report NM-R9224 and references therein.
c
c Local Uniform Grid Refinement is a static-regridding technique which
c creates locally nested finer-and-finer uniform subgrids until
c sufficient accuracy in space is reached.
c
c Standard second order finite differences are used
c for the space discretization of the PDEs and the BCs.
c
c The interpolation is linear.
c
c The Implicit Euler method is used for the first time step and
c BDF2 with variable coefficients is used for the next time steps.
c
c The variable time stepsize is controlled by the time error monitor
c (Un+1 - Un).
c
c The spatial mesh refinements are governed by the error monitor
c abs( Ui-1j -2 Uij + Ui+1j ) + abs( Uij-1 -2 Uij + Uij+1 )
c
c The system of nonlinear equations is solved by an adapted version of
c modified Newton in combination with preconditioned BiCGSTAB.
c
```

```
c=======================================================================
c Input by user
c=======================================================================
c
c npde                INTEGER number of PDEs to be solved.
c nptspl              INTEGER maximum number of gridpoints allowed
c                     at each gridlevel.
c maxlev              INTEGER maximum number of grid levels allowed.
c ntimes              INTEGER number of time levels at which output
c                     is generated +1.
c
      integer    npde,nptspl,maxlev,ntimes,npar
      parameter (npde=1)
      parameter (nptspl=6600)
      parameter (maxlev=3)
      parameter (ntimes=3)
      parameter (npar=50)
c
c-----------------------------------------------------------------------
c
c licn                INTEGER maximum storage needed for BiCGSTAB.
c lirn                INTEGER maximum storage needed for BiCGSTAB.
c The values for both licn and lirn should be chosen large enough.
c
      integer licn,lirn
      parameter (licn=(8+9*npde)*npde*nptspl,lirn=licn)
c
c=======================================================================
      integer nrwspl,nvlspl
      parameter (nrwspl=nptspl/5+2)
      parameter (nvlspl=nptspl*npde)
      logical SPCest,TMest,check,errFw
      integer irwst,numrws,jrn,icn,iq,ib,ipF,jrnw,icnw,irwstw
      double precision outime,VAL,SPCerr,TOLspc,SPCmx,
     +                 Fu,Fut,Fux,Fuy,Fuxx,Fuxy,Fuyy,
     +                 U,Ut,Ux,Uy,Uxx,Uxy,Uyy,
     +                 Uold,Uoldt,
     +                 F0,F1,
     +                 par,
     +                 work,scale,Xmin,Xmax,Ymin,Ymax,dx,dy
      character *80 PROBLEM,METHOD,PARAM1,PARAM2
      double precision a(licn),cor(nvlspl),wt(nvlspl),
     +                 rhs(nvlspl),w(licn)
      integer nrn(lirn),ncn(licn),iw(licn)
      dimension outime(ntimes), numrws(maxlev), jrn(nrwspl,maxlev),
     +          irwst(nrwspl,maxlev), icn(nptspl,maxlev),
     +          iq(nptspl,maxlev), ib(nptspl,maxlev),
     +          ipF(nptspl,maxlev), VAL(nvlspl,3,maxlev),
     +          jrnw(nrwspl), irwstw(nrwspl), icnw(nptspl),
     +          errFw(0:nptspl), SPCerr(nptspl,npde),
     +          TOLspc(npde),SPCmx(npde),
     +          Fu(nvlspl,npde),Fux(nvlspl,npde),Fuy(nvlspl,npde),
     +          Fut(nvlspl,npde),
     +          Fuxx(nvlspl,npde),Fuxy(nvlspl,npde),Fuyy(nvlspl,npde),
     +          F0(nvlspl), F1(nvlspl), U(nvlspl), Ut(nvlspl),
     +          Ux(nvlspl), Uxx(nvlspl),Uy(nvlspl), Uxy(nvlspl),
     +          Uyy(nvlspl), work(nvlspl,3),
     +          Uold(nvlspl,maxlev),Uoldt(nvlspl,maxlev),
     +          SPCest(npde),TMest(npde),scale(npde)
      common /param/ par(npar)
      METHOD='MOORKOP2.0, LUGR WITH BDF2'
      PARAM1='TOLT = xxxxxxxxxx  TOLS = xxxxxxxxxx'
      PARAM2='VARIABLE TIMESTEP, LINEAR INTERPOLATION'
c=======================================================================
c Input by user
c=======================================================================
c
c outime(.)           REAL array containing time levels.
c                     outime(1) is the initial time.
c                     outime(2),...,outime(ntimes) are time levels
c                     at which output is created.
c
      outime(1)=0.0d0
      outime(2)=0.5d0
      outime(3)=1.0d0
c
c-----------------------------------------------------------------------
c
c SPCest(.)           LOGICAL array specifying solution components
c                     for which the space error monitor is to be
c                     evaluated.
c
      SPCest(1)=.true.
c
c-----------------------------------------------------------------------
c
c TMest(.)            LOGICAL array specifying solution components
c                     for which the time error monitor is to be
c                     evaluated.
c
      TMest(1)=.true.
c
```

```
c-----------------------------------------------------------------------
c
c check                LOGICAL specifying whether the time error should
c                      be estimated after the first time step.
c                      In case the solution to the PDEs and boundary
c                      conditions are inconsistent with the initial
c                      values, check=.false., otherwise check=.true.
c
      check=.true.
c
c-----------------------------------------------------------------------
c
c scale(.)             REAL array containing the scales of solution
c                      components.
c
      scale(1)=1.0d0
c
c-----------------------------------------------------------------------
c
c par(.)               REAL array containing information about the PDEs.
c
      par(1)=-80.0d0
      par(2)=4.0d0*datan(1.0d0)
c
c-----------------------------------------------------------------------
c
c PROBLEM              CHARACTER containing information about the PDEs.
c
      PROBLEM='Rotating Cone'
c
c-----------------------------------------------------------------------
c
c Xmin,Xmax,Ymin,Ymax REAL constants specifying the boundaries of the
c                      domain.
c
      Xmin=0.0d0
      Xmax=1.0d0
      Ymin=0.0d0
      Ymax=1.0d0
c
c=======================================================================
      call PDEsol
     +  (outime,numrws,jrn,irwst,icn,iq,ib,ipF,VAL,jrnw,irwstw,icnw,
     +   errFw,Fu,Fut,Fux,Fuy,Fuxx,Fuxy,Fuyy,F0,F1,U,Ut,Ux,Uy,
     +   Uxx,Uxy,Uyy,work,SPCest,TMest,check,
     +   npde,nptspl,maxlev,ntimes,nrwspl,nvlspl,Xmin,Xmax,Ymin,Ymax,
     +   Uold,Uoldt,
     +   PROBLEM,METHOD,PARAM1,PARAM2,
     +   scale,SPCerr,TOLspc,SPCmx,
     +   a,rhs,w,nrn,ncn,iw,licn,lirn,cor,wt)
      end




      subroutine PDE
     +  (irwst,jrn,icn,ipF,F,U,Ut,Ux,Uy,Uxx,Uxy,Uyy,nrows,level,
     +   nrwspl,nptspl,nvlspl,npde,t,iq)
c-----------------------------------------------------------------------
c
c In this subroutine the user defines the partial differential
c equations and the boundary conditions.
c
c-----------------------------------------------------------------------
      double precision F,U,Ut,Ux,Uy,Uxx,Uxy,Uyy,t,
     +                 x,y,x0,y0,dx,dy,dxl,dyl,ratio,par,uround,
     +                 fx,fy,fxx,fyy,ex,r,s,st,ct,pi,h,ft
      integer irwst,jrn,icn,ipF,nrows,level,nrwspl,nptspl,nvlspl,npde,
     +        ii,jj,kk,idx,iq
      dimension irwst(nrwspl),jrn(nrwspl),icn(nptspl),ipF(nptspl),
     +          F(nvlspl),iq(nptspl),
     +          U(nvlspl),Ut(nvlspl),Ux(nvlspl),Uy(nvlspl),Uxx(nvlspl),
     +          Uxy(nvlspl),Uyy(nvlspl)
      common /mesh/ x0,y0,dx,dy
      common /param/ par(1)
      common /machar/ uround
      ratio=2.0d0**(1-level)
      dxl=dx*ratio
      dyl=dy*ratio
      do 30 ii=1,nrows
        y=y0+dyl*jrn(ii)
        do 20 jj=irwst(ii),irwst(ii+1)-1
          x=x0+dxl*icn(jj)
          idx=npde*(jj-1)
c=======================================================================
c The user supplied block starts here
c=======================================================================
```

```
c-----------------------------------------------------------------------
c The computation of user defined variables needed to specify the PDEs
c and BCs
c-----------------------------------------------------------------------
          pi=par(2)
          st=dsin(pi*t)
          ct=dcos(pi*t)
          r=0.25d0*(2.0d0+st)
          s=0.25d0*(2.0d0+ct)
          ex=dexp(par(1)*((x-r)**2+(y-s)**2))
          ft=-0.5d0*pi*ct*par(1)*(x-r)+0.5d0*pi*st*par(1)*(y-s)
          fx=2.0d0*par(1)*(x-r)
          fy=2.0d0*par(1)*(y-s)
          fxx=2.0d0*par(1)
          fyy=2.0d0*par(1)
          h=(ft-fxx-fyy-fx*fx-fy*fy)*ex
c-----------------------------------------------------------------------
c The specification of the PDEs at the interior of the domain
c-----------------------------------------------------------------------
          if (ipF(jj).eq.0) then
            F(idx+1)=
     +               -Ut(idx+1)+Uxx(idx+1)+Uyy(idx+1)+h
c
          else if (ipF(jj).eq.2) then
c-----------------------------------------------------------------------
c The specification of the BCs at the lower boundary of the domain
c-----------------------------------------------------------------------
              F(idx+1)=U(idx+1)-ex
c
          else if (ipF(jj).eq.3) then
c-----------------------------------------------------------------------
c The specification of the BCs at the upper boundary of the domain
c-----------------------------------------------------------------------
              F(idx+1)=U(idx+1)-ex
c
          else if (ipF(jj).eq.5) then
c-----------------------------------------------------------------------
c The specification of the BCs at the left boundary of the domain
c-----------------------------------------------------------------------
              F(idx+1)=U(idx+1)-ex
c
          else if (ipF(jj).eq.9) then
c-----------------------------------------------------------------------
c The specification of the BCs at the right boundary of the domain
c-----------------------------------------------------------------------
              F(idx+1)=U(idx+1)-ex
c
          else if (ipF(jj).eq.6) then
c-----------------------------------------------------------------------
c The specification of the BCs at the lower left corner of the domain
c-----------------------------------------------------------------------
              F(idx+1)=U(idx+1)-ex
c
          else if (ipF(jj).eq.7) then
c-----------------------------------------------------------------------
c The specification of the BCs at the upper left corner of the domain
c-----------------------------------------------------------------------
              F(idx+1)=U(idx+1)-ex
c
          else if (ipF(jj).eq.10) then
c-----------------------------------------------------------------------
c The specification of the BCs at the lower right corner of the domain
c-----------------------------------------------------------------------
              F(idx+1)=U(idx+1)-ex
c
          else if (ipF(jj).eq.11) then
c-----------------------------------------------------------------------
c The specification of the BCs at the upper right corner of the domain
c-----------------------------------------------------------------------
              F(idx+1)=U(idx+1)-ex
c
c=======================================================================
          end if
   20   continue
   30 continue
      return
      end
```

```
      subroutine Uinit
     +  (irwst,jrn,icn,U,nrows,level,nrwspl,nptspl,nvlspl,npde,t)
c-----------------------------------------------------------------------
c
c In this subroutine the user provides the initial values as a
c function of the spatial co-ordinates.
c
c-----------------------------------------------------------------------
      double precision U,t,
     +                 x,y,x0,y0,dx,dy,dxl,dyl,ratio,par,
     +                 pi,r,s,st,ct
      integer irwst,jrn,icn,nrows,level,nrwspl,nptspl,nvlspl,npde,
     +        ii,jj,idx
      dimension irwst(nrwspl),jrn(nrwspl),icn(nptspl),U(nvlspl)
      common /mesh/ x0,y0,dx,dy
      common /param/ par(1)
      ratio=2.0**(1-level)
      dxl=ratio*dx
      dyl=ratio*dy
      do 20 ii=1,nrows
        y=y0+jrn(ii)*dyl
        do 10 jj=irwst(ii),irwst(ii+1)-1
          x=x0+icn(jj)*dxl
          idx=npde*(jj-1)
c=======================================================================
c The user supplied block starts here
c=======================================================================
c-----------------------------------------------------------------------
c The computation of user defined variables needed to specify the IVs
c-----------------------------------------------------------------------
          pi=par(2)
          st=dsin(pi*t)
          ct=dcos(pi*t)
          r=0.25d0*(2.0d0+st)
          s=0.25d0*(2.0d0+ct)
c
c-----------------------------------------------------------------------
c The specification of the IVs
c-----------------------------------------------------------------------
          U(idx+1)=dexp(par(1)*((x-r)**2+(y-s)**2))
c
c=======================================================================
   10   continue
   20 continue
      return
      end
```

7.2. Brine Transport Problem in Porous Media with Inhomogeneities

This example problem deals with brine transport in porous media with inhomogeneities. We describe an example problem which was also used in [8]. After this, we give the full program input.

7.2.1. . Problem Description

This example problem deals with the displacement of fresh water by brine in a thin vertical column, filled with a porous medium and measuring one by one meter. Here we assume that $\mathbf{g} = (0, -g)^T$. The values of the parameters which are continuous are chosen as

$$n = 0.4, \quad d_m = 0\, m^2.s, \quad \rho_0 = 10^3\, kg.m^{-3}, \quad p_0 = 10^5\, N.m^{-2},$$
$$\gamma = \log(2.0), \quad g = 9.81\, m.s^{-2}, \quad \mu_0 = 10^{-3} kg.m.s^{-1}.$$

The vertical column is completely open at the top and half open at the bottom and closed at its vertical sides. This configuration is shown in Figure 7.1. Four different regions can be distinguished, indicated as I till IV. Each of these regions has its own permeability and longitudinal and transversal dispersivity.

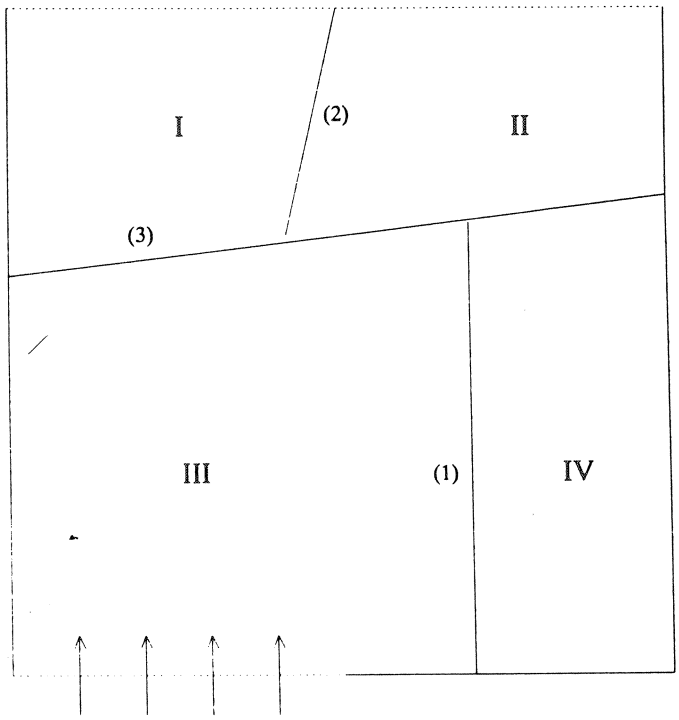

FIGURE 7.1 The vertical column of the example problem.

The initial values, boundary conditions and soil parameters are:

$$p(x,y,0) = p_0 + (1-y)\rho_0 g, \quad \omega(x,y,0) = 0, \quad 0\,m < x,y < 1\,m,$$
$$q_1 = 0\,m.s^{-1}, \quad \omega_x = 0\,m^{-1}, \quad x = 0,1\,m \text{ and } 0\,m < y < 1\,m,$$
$$q_2 = 10^{-4} \times (1 - \exp(-10t)), \quad \omega = 0.25 \times (1 - \exp(-10t)), \quad 0\,m < x \leq 0.5\,m \text{ and } y = 0\,m,$$
$$q_2 = 0\,m.s^{-1}, \quad \omega = 0, \quad 0.5\,m < x < 1\,m \text{ and } y = 0\,m,$$
$$p = p_0, \quad \omega_y = 0\,m^{-1}, \quad 0\,m < x < 1\,m \text{ and } y = 1\,m.$$

The soil parameters are given by:

$$\text{I:} \quad k = 10^{-13}\,m^2, \quad \alpha_l = 0.008\,m, \quad \alpha_t = 0.0016\,m,$$
$$\text{II:} \quad k = 10^{-15}\,m^2, \quad \alpha_l = 0.005\,m, \quad \alpha_t = 0.0010\,m,$$
$$\text{III:} \quad k = 10^{-10}\,m^2, \quad \alpha_l = 0.010\,m, \quad \alpha_t = 0.0020\,m,$$
$$\text{IV:} \quad k = 10^{-13}\,m^2, \quad \alpha_l = 0.008\,m, \quad \alpha_t = 0.0016\,m.$$

The interfaces are defined as:

1: $x=0.7\,m$,

2: $x=0.3\,m + 0.2 \times y$,

3: $y=0.6\,m + 0.1 \times x$.

7.2.2. . Program Input

MOORKOP 2.1 was used for this problem. The sources for this problem of *program moorkop*, *subroutine PDE*, *subroutine Uinit* and *subroutine SOIL* are given below.

```
      program  moorkop
c
c
c     MOORKOP2.1
c
c
c-----------------------------------------------------------------------
c
c moorkop solves partial differential equations (PDEs) of the type
c
c G(x,y,t,U,Ut,Ux,Uy,Uxx,Uxy,Uyy)=0
c
c with the boundary conditions (BCs)
c
c H(x,y,t,U,Ut,Ux,Uy)=0
c
c and the initial values (IVs)
c
c U=F0(x,y)
c
c on a rectangular domain.
c
c moorkop is designed to solve PDEs with solutions possessing steep
c moving transitions in space and time, using Local Unform Grid
c Refinement. This adaptive grid technique is described in the
c CWI report NM-R9224 and references therein.
c
c Local Uniform Grid Refinement is a static-regridding technique which
c creates locally nested finer-and-finer uniform subgrids until
c sufficient accuracy in space is reached.
c
c Standard second order finite differences are used
c for the space discretization of the PDEs and the BCs.
c
c The interpolation is linear.
c
c The Implicit Euler method is used for the first time step and
c BDF2 with variable coefficients is used for the next time steps.
c
c The variable time stepsize is controlled by the time error monitor
c (Un+1 - Un).
c
c The spatial mesh refinements are governed by the error monitor
c abs( Ui-1j -2 Uij + Ui+1j ) + abs( Uij-1 -2 Uij + Uij+1 )
c
c The system of nonlinear equations is solved by an adapted version of
c modified Newton in combination with preconditioned BiCGSTAB.
c
c=======================================================================
c Input by user
c=======================================================================
c
c npde                INTEGER number of PDEs to be solved.
c nptspl              INTEGER maximum number of gridpoints allowed
c                     at each gridlevel.
c maxlev              INTEGER maximum number of grid levels allowed.
c ntimes              INTEGER number of time levels at which output
c                     is generated +1.
c npar                INTEGER number of user defined parameters.
c nsoil               INTEGER number of soil parameters which are
c                     discontinuous in the domain.
c
      integer    npde,nptspl,maxlev,ntimes,npar,nsoil
      parameter (npde=2)
      parameter (nptspl=6600)
      parameter (maxlev=2)
      parameter (ntimes=20)
      parameter (npar=50)
      parameter (nsoil=10)
c
```

```
c-----------------------------------------------------------------------
c
c licn                INTEGER maximum storage needed for BiCGSTAB.
c lirn                INTEGER maximum storage needed for BiCGSTAB.
c The values for both licn and lirn should be chosen large enough.
c
      integer licn,lirn
      parameter (licn=(8+9*npde)*npde*nptspl,lirn=licn)
c
c=======================================================================
      integer nrwspl,nvlspl,nval,nvel
      parameter (nrwspl=nptspl/5+2)
      parameter (nvlspl=nptspl*npde)
      parameter (nval=nsoil*nptspl+1)
      parameter (nvel=nptspl*2*maxlev)
      logical SPCest,TMest,check,errFw,prevfl
      integer irwst,numrws,jrn,icn,iq,ib,ipF,jrnw,icnw,irwstw
      double precision outime,VAL,SPCerr,TOLspc,SPCmx,
     +                 Fu,Fut,Fux,Fuy,Fuxx,Fuxy,Fuyy,FuxR,FuyB,
     +                 U,Ut,Ux,Uy,Uxx,Uxy,Uyy,UxR,UyB,
     +                 Uold,Uoldt,
     +                 F0,F1,SP,q,
     +                 par,
     +                 work,scale,Xmin,Xmax,Ymin,Ymax,dx,dy
      character *80 PROBLEM,METHOD,PARAM1,PARAM2
      double precision a(licn),cor(nvlspl),wt(nvlspl),
     +                 rhs(nvlspl),w(licn)
      integer nrn(lirn),ncn(licn),iw(licn)
      dimension outime(ntimes), numrws(maxlev), jrn(nrwspl,maxlev),
     +          irwst(nrwspl,maxlev), icn(nptspl,maxlev),
     +          iq(nptspl,maxlev), ib(nptspl,maxlev),
     +          ipF(nptspl,maxlev), VAL(nvlspl,3,maxlev),
     +          jrnw(nrwspl), irwstw(nrwspl), icnw(nptspl),
     +          errFw(0:nptspl), SPCerr(nptspl,npde),
     +          TOLspc(npde),SPCmx(npde),
     +          Fu(nvlspl,npde),Fux(nvlspl,npde),Fuy(nvlspl,npde),
     +          Fut(nvlspl,npde),
     +          Fuxx(nvlspl,npde),Fuxy(nvlspl,npde),Fuyy(nvlspl,npde),
     +          F0(nvlspl), F1(nvlspl), U(nvlspl), Ut(nvlspl),
     +          Ux(nvlspl), Uxx(nvlspl),Uy(nvlspl), Uxy(nvlspl),
     +          Uyy(nvlspl), work(nvlspl,3),UxR(nvlspl),UyB(nvlspl),
     +          Uold(nvlspl,maxlev),Uoldt(nvlspl,maxlev),
     +          FuxR(nvlspl,npde),FuyB(nvlspl,npde),
     +          SPCest(npde),TMest(npde),scale(npde)
      common /param/ par(npar)
      common /perm/ SP(nval)
      common /veloc/ q(nvel)
      METHOD='MOORKOP2.1, LUGR WITH BDF2'
      PARAM1='TOLT = xxxxxxxxxx  TOLS = xxxxxxxxx'
      PARAM2='VARIABLE TIMESTEP, LINEAR INTERPOLATION'
c=======================================================================
c Input by user
c=======================================================================
c
c outime(.)           REAL array containing time levels.
c                     outime(1) is the initial time.
c                     outime(2),...,outime(ntimes) are time levels
c                     at which output is created.
c
      outime(1)=0.0d0
      outime(2)=10.0d0
      outime(3)=100.0d0
      outime(4)=500.0d0
      outime(5)=1000.0d0
      outime(6)=1500.0d0
      outime(7)=2000.0d0
      outime(8)=2500.0d0
      outime(9)=3000.0d0
      outime(10)=4000.0d0
      outime(11)=5000.0d0
      outime(12)=7500.0d0
      outime(13)=10000.0d0
      outime(14)=15000.0d0
      outime(15)=20000.0d0
      outime(16)=40000.0d0
      outime(17)=60000.0d0
      outime(18)=80000.0d0
      outime(19)=100000.0d0
      outime(20)=1000000.0d0
c
c-----------------------------------------------------------------------
c
c SPCest(.)           LOGICAL array specifying solution components
c                     for which the space error monitor is to be
c                     evaluated.
c
      SPCest(1)=.true.
      SPCest(2)=.true.
c
```

```
c-----------------------------------------------------------------------
c
c TMest(.)              LOGICAL array specifying solution components
c                       for which the time error monitor is to be
c                       evaluated.
c
      TMest(1)=.false.
      TMest(2)=.true.
c
c-----------------------------------------------------------------------
c
c check                 LOGICAL specifying whether the time error should
c                       be estimated after the first time step.
c                       In case the solution to the PDEs and boundary
c                       conditions are inconsistent with the initial
c                       values, check=.false., otherwise check=.true.
c
      check=.false.
c
c-----------------------------------------------------------------------
c
c scale(.)              REAL array containing the scales of solution
c                       components.
c
      scale(1)=1.0d+5
      scale(2)=0.25d0
c
c-----------------------------------------------------------------------
c prevfl                LOGICAL specifying whether the mesh should
c                       be always be refined at interfaces or not
      prevfl=.false.
c
c-----------------------------------------------------------------------
c
c par(.)                REAL array containing information about the PDEs.
c
c     n
      par(1)=0.4d0
c     gamma
      par(2)=dlog(2.0d0)
c     mu0
      par(4)=1.0d-3
c     rho0
      par(5)=1.0d+3
c     p0
      par(6)=1.0d+5
c     g
      par(7)=9.81d0
c     dm
      par(8)=0.0d0
c
c-----------------------------------------------------------------------
c
c PROBLEM               CHARACTER containing information about the PDEs.
c
      PROBLEM='MAJIDs SOM'
c
c-----------------------------------------------------------------------
c
c Xmin,Xmax,Ymin,Ymax REAL constants specifying the boundaries of the
c                       domain.
c
      Xmin=0.0d0
      Xmax=1.0d0
      Ymin=0.0d0
      Ymax=1.0d0
c
c=======================================================================
      call PDEsol
     +  (outime,numrws,jrn,irwst,icn,iq,ib,ipF,VAL,jrnw,irwstw,icnw,
     +   errFw,Fu,Fut,Fux,Fuy,Fuxx,Fuxy,Fuyy,FuxR,FuyB,F0,F1,U,Ut,Ux,Uy,
     +   Uxx,Uxy,Uyy,work,UxR,UyB,SPCest,TMest,check,
     +   npde,nptspl,maxlev,ntimes,nrwspl,nvlspl,Xmin,Xmax,Ymin,Ymax,
     +   Uold,Uoldt,prevfl,
     +   PROBLEM,METHOD,PARAM1,PARAM2,
     +   scale,SPCerr,TOLspc,SPCmx,
     +   a,rhs,w,nrn,ncn,iw,licn,lirn,cor,wt)
      end
```

```
      subroutine PDE
     +  (irwst,jrn,icn,ipF,F,U,Ut,Ux,Uy,Uxx,Uxy,Uyy,UxR,UyB,nrows,level,
     +   nrwspl,nptspl,nvlspl,npde,t,iq)
c-----------------------------------------------------------------------
c
c In this subroutine the user defines the partial differential
c equations and the boundary conditions.
c
c-----------------------------------------------------------------------
      double precision F,U,Ut,Ux,Uy,Uxx,Uxy,Uyy,UxR,UyB,t,
     +                 x,y,x0,y0,dx,dy,dxl,dyl,ratio,par,SP,q,
     +                 n,dm,gamma,al,at,p,pt,px,py,pxx,pxy,pyy,mu,
     +                 w,wt,wx,wy,wxx,wxy,wyy,rho,q1,q2,q1x,q1y,q2x,
     +                 q2y,aq,nD11,nD12,nD22,nD11q1,nD11q2,
     +                 nD12q1,nD12q2,nD22q1,nD22q2,nD11x,nD12x,nD12y,
     +                 nD22y,mu0,rho0,p0,g,muw,K,
     +                 rhox,rhoy,Jw1,Jw2,Jw1x,Jw2y,
     +                 Jw1ne,Jw2ne,Jw1se,Jw2se,Jw1nw,Jw2nw,Jw1sw,Jw2sw,
     +                 nD11sw,nD12sw,nD22sw,
     +                 nD11se,nD12se,nD22se,
     +                 nD11ne,nD12ne,nD22ne,
     +                 nD11nw,nD12nw,nD22nw,
     +                 q1ne,q2ne,q1se,q2se,q1nw,q2nw,q1sw,q2sw
      integer irwst,jrn,icn,ipF,nrows,level,nrwspl,nptspl,nvlspl,npde,
     +        ii,jj,kk,idx,iq
      dimension irwst(nrwspl),jrn(nrwspl),icn(nptspl),ipF(nptspl),
     +          F(nvlspl),iq(nptspl),
     +          U(nvlspl),Ut(nvlspl),Ux(nvlspl),Uy(nvlspl),Uxx(nvlspl),
     +          Uxy(nvlspl),Uyy(nvlspl),UxR(nvlspl),UyB(nvlspl)
      common /mesh/ x0,y0,dx,dy
      common /perm/ SP(1)
      common /veloc/ q(1)
      common /param/ par(1)
      ratio=2.0d0**(1-level)
      dxl=dx*ratio
      dyl=dy*ratio
      do 30 ii=1,nrows
        y=y0+dyl*jrn(ii)
        do 20 jj=irwst(ii),irwst(ii+1)-1
          x=x0+dxl*icn(jj)
          idx=npde*(jj-1)
c=======================================================================
c The user supplied block starts here
c=======================================================================
c-----------------------------------------------------------------------
c The computation of user defined variables needed to specify the PDEs
c and BCs
c-----------------------------------------------------------------------
          n=par(1)
          gamma=par(2)
          mu0=par(4)
          rho0=par(5)
          p0=par(6)
          g=par(7)
          dm=par(8)
          K=SP(jj)
          al=SP(jj+nptspl)
          at=SP(jj+2*nptspl)
          p=U(idx+1)
          px=Ux(idx+1)
          py=Uy(idx+1)
          w=U(idx+2)
          wt=Ut(idx+2)
          wx=Ux(idx+2)
          wy=Uy(idx+2)
          mu=1.0d0+1.85d0*w-4.10d0*w*w+44.5d0*w*w*w
          muw=1.85d0-8.20d0*w+133.5d0*w*w
          mu=mu*mu0
          muw=muw*mu0
          rho=rho0*dexp(gamma*w)
          rhox=rho*gamma*wx
          rhoy=rho*gamma*wy
          q1=-K*px/mu
          q2=-K*(py+rho*g)/mu
          aq=dmax1(dsqrt(q1*q1+q2*q2),1.0d-20)
          nD11=n*dm+at*aq+(al-at)*q1*q1/aq
          nD12=(al-at)*q1*q2/aq
          nD22=n*dm+at*aq+(al-at)*q2*q2/aq
          pxx=Uxx(idx+1)
          pxy=Uxy(idx+1)
          pyy=Uyy(idx+1)
          wxx=Uxx(idx+2)
          wxy=Uxy(idx+2)
          wyy=Uyy(idx+2)
          nD11q1=(at+(al-at)*(2.0d0-((q1/aq)**2)))*q1/aq
          nD11q2=(at-(al-at)*((q1/aq)**2))*q2/aq
          nD12q1=(al-at)*((q2/aq)**3)
          nD12q2=(al-at)*((q1/aq)**3)
          nD22q1=(at-(al-at)*((q2/aq)**2))*q1/aq
          nD22q2=(at+(al-at)*(2.0d0-((q2/aq)**2)))*q2/aq
          q1x=K*(-pxx/mu+px*muw*wx/(mu*mu))
          q1y=K*(-pxy/mu+px*muw*wy/(mu*mu))
          q2x=
     +         -K*(pxy+rhox*g)/mu
     +         +K*(py+rho*g)*muw*wx/(mu*mu)
          q2y=
     +         -K*(pyy+rhoy*g)/mu
     +         +K*(py+rho*g)*muw*wy/(mu*mu)
          nD11x=nD11q1*q1x+nD11q2*q2x
          nD12x=nD12q1*q1x+nD12q2*q2x
          nD12y=nD12q1*q1y+nD12q2*q2y
```

```
          nD22y=nD22q1*q1y+nD22q2*q2y
          Jw1=-nD11*wx-nD12*wy
          Jw2=-nD12*wx-nD22*wy
          Jw1x=-nD11x*wx-nD11*wxx-nD12x*wy-nD12*wxy
          Jw2y=-nD12y*wx-nD12*wxy-nD22y*wy-nD22*wyy
c-----------------------------------------------------------------------
c
c The velocities in x- and y- direction are stored in the array q(.)
c which can be written to the VLCT files.
c
c-----------------------------------------------------------------------
          if (ipF(jj).ne.12) then
c
c velocity is x- direction
c
            q(2*nptspl*(level-1)+2*(jj-1)+1)=-K*px/mu
c
c velocity is y- direction
c
            q(2*nptspl*(level-1)+2*(jj-1)+2)=-K*(py+rho*g)/mu
c
          else
c
c velocity is x- direction
c
            q(2*nptspl*(level-1)+2*(jj-1)+1)=-K*UxR(idx+1)/mu
c
c velocity is y- direction
c
            q(2*nptspl*(level-1)+2*(jj-1)+2)=-K*(UyB(idx+1)+rho*g)/mu
c
          end if
          if (ipF(jj).eq.0) then
c-----------------------------------------------------------------------
c The specification of the PDEs at the interior of the domain
c-----------------------------------------------------------------------
            F(idx+1)=
     +              -gamma*gamma*(Jw1*wx+Jw2*wy)
     +              -gamma*(Jw1x+Jw2y)
     +              +(q1x+q2y)
            F(idx+2)=
     +               n*wt+q1*wx+q2*wy
     +              +gamma*(Jw1*wx+Jw2*wy)
     +              +Jw1x+Jw2y
c**********************************************************************
c
c WARNING
c
c When the domain contains an impervious region, the flow and transport
c equations become meaningless.
c If this is the case then the user has to provide "dummy" equations
c for the impervious region. An example is given below.
c
c           if (K.eq.0.0d0) then
c             F(idx+1)=p-p0
c             F(idx+2)=w
c            end if
c
c**********************************************************************
c
          else if (ipF(jj).eq.2) then
c-----------------------------------------------------------------------
c The specification of the BCs at the lower boundary of the domain
c-----------------------------------------------------------------------
          if (x.le.0.5d0) then
              F(idx+1)=q2-1.0d-4*(1.0d0-dexp(-10*t))
              F(idx+2)=w-0.25d0*(1.0d0-dexp(-10*t))
          else
              F(idx+1)=q2
              F(idx+2)=wy
          end if
c
          else if (ipF(jj).eq.3) then
c-----------------------------------------------------------------------
c The specification of the BCs at the upper boundary of the domain
c-----------------------------------------------------------------------
              F(idx+1)=p-p0
              F(idx+2)=wy
c
          else if (ipF(jj).eq.5) then
c-----------------------------------------------------------------------
c The specification of the BCs at the left boundary of the domain
c-----------------------------------------------------------------------
              F(idx+1)=q1
              F(idx+2)=wx
c
          else if (ipF(jj).eq.9) then
c-----------------------------------------------------------------------
c The specification of the BCs at the right boundary of the domain
c-----------------------------------------------------------------------
              F(idx+1)=q1
              F(idx+2)=wx
c
          else if (ipF(jj).eq.6) then
c-----------------------------------------------------------------------
c The specification of the BCs at the lower left corner of the domain
c-----------------------------------------------------------------------
              F(idx+1)=q2
              F(idx+2)=wy
c
          else if (ipF(jj).eq.7) then
```

```
c-----------------------------------------------------------------------
c The specification of the BCs at the upper left corner of the domain
c-----------------------------------------------------------------------
              F(idx+1)=p-p0
              F(idx+2)=wy
c
         else if (ipF(jj).eq.10) then
c-----------------------------------------------------------------------
c The specification of the BCs at the lower right corner of the domain
c-----------------------------------------------------------------------
              F(idx+1)=q2
              F(idx+2)=wy
c
         else if (ipF(jj).eq.11) then
c-----------------------------------------------------------------------
c The specification of the BCs at the upper right corner of the domain
c-----------------------------------------------------------------------
              F(idx+1)=p-p0
              F(idx+2)=wy
c
         else if (ipF(jj).eq.12) then
c-----------------------------------------------------------------------
c The specification of the interface conditions
c-----------------------------------------------------------------------
c
c Computation of the fluxes at upper right cell
c
              K=SP(jj)
              al=SP(jj+nptspl)
              at=SP(jj+2*nptspl)
              q1ne=-K*UxR(idx+1)/mu
              q2ne=-K*(UyB(idx+1)+rho*g)/mu
              aq=dmax1(dsqrt(q1ne*q1ne+q2ne*q2ne),1.0d-20)
              nD11ne=n*dm+at*aq+(al-at)*q1ne*q1ne/aq
              nD12ne=(al-at)*q1ne*q2ne/aq
              nD22ne=n*dm+at*aq+(al-at)*q2ne*q2ne/aq
              Jw1ne=-nD11ne*UxR(idx+2)-nD12ne*UyB(idx+2)
              Jw2ne=-nD12ne*UxR(idx+2)-nD22ne*UyB(idx+2)
c
c Computation of the fluxes at lower right cell
c
              K=SP(iq(jj))
              al=SP(iq(jj)+nptspl)
              at=SP(iq(jj)+2*nptspl)
              q1se=-K*UxR(idx+1)/mu
              q2se=-K*(Uy(idx+1)+rho*g)/mu
              aq=dmax1(dsqrt(q1se*q1se+q2se*q2se),1.0d-20)
              nD11se=n*dm+at*aq+(al-at)*q1se*q1se/aq
              nD12se=(al-at)*q1se*q2se/aq
              nD22se=n*dm+at*aq+(al-at)*q2se*q2se/aq
              Jw1se=-nD11se*UxR(idx+2)-nD12se*Uy(idx+2)
              Jw2se=-nD12se*UxR(idx+2)-nD22se*Uy(idx+2)
c
c Computation of the fluxes at upper left cell
c
              K=SP(jj-1)
              al=SP(jj-1+nptspl)
              at=SP(jj-1+2*nptspl)
              q1nw=-K*Ux(idx+1)/mu
              q2nw=-K*(UyB(idx+1)+rho*g)/mu
              aq=dmax1(dsqrt(q1nw*q1nw+q2nw*q2nw),1.0d-20)
              nD11nw=n*dm+at*aq+(al-at)*q1nw*q1nw/aq
              nD12nw=(al-at)*q1nw*q2nw/aq
              nD22nw=n*dm+at*aq+(al-at)*q2nw*q2nw/aq
              Jw1nw=-nD11nw*Ux(idx+2)-nD12nw*UyB(idx+2)
              Jw2nw=-nD12nw*Ux(idx+2)-nD22nw*UyB(idx+2)
c
c Computation of the fluxes at lower left cell
c
              K=SP(iq(jj)-1)
              al=SP(iq(jj)-1+nptspl)
              at=SP(iq(jj)-1+2*nptspl)
              q1sw=-K*Ux(idx+1)/mu
              q2sw=-K*(Uy(idx+1)+rho*g)/mu
              aq=dmax1(dsqrt(q1sw*q1sw+q2sw*q2sw),1.0d-20)
              nD11sw=n*dm+at*aq+(al-at)*q1sw*q1sw/aq
              nD12sw=(al-at)*q1sw*q2sw/aq
              nD22sw=n*dm+at*aq+(al-at)*q2sw*q2sw/aq
              Jw1sw=-nD11sw*Ux(idx+2)-nD12sw*Uy(idx+2)
              Jw2sw=-nD12sw*Ux(idx+2)-nD22sw*Uy(idx+2)
c
c The flux continuity equations
c
              F(idx+1)=0.0d0
              F(idx+2)=0.0d0
              if (SP(jj).ne.SP(jj-1)) then
c-----------------------------------------------------------------------
c The cell face separating the upper right from the upper left cell
c is an interface
c-----------------------------------------------------------------------
                F(idx+1)=F(idx+1)+q1ne-q1nw
                F(idx+2)=F(idx+2)+Jw1ne-Jw1nw
              end if
              if (SP(iq(jj)).ne.SP(iq(jj)-1)) then
c-----------------------------------------------------------------------
c The cell face separating the lower right from the lower left cell
c is an interface
c-----------------------------------------------------------------------
                F(idx+1)=F(idx+1)+q1se-q1sw
                F(idx+2)=F(idx+2)+Jw1se-Jw1sw
              end if
```

```
            if (SP(jj).ne.SP(iq(jj))) then
c-----------------------------------------------------------------------
c The cell face separating the upper right from the lower right cell
c is an interface
c-----------------------------------------------------------------------
              F(idx+1)=F(idx+1)+q2ne-q2se
              F(idx+2)=F(idx+2)+Jw2ne-Jw2se
            end if
            if (SP(jj-1).ne.SP(iq(jj)-1)) then
c-----------------------------------------------------------------------
c The cell face separating the upper left from the lower left cell
c is an interface
c-----------------------------------------------------------------------
              F(idx+1)=F(idx+1)+q2nw-q2sw
              F(idx+2)=F(idx+2)+Jw2nw-Jw2sw
            end if
c=======================================================================
          end if
   20   continue
   30 continue
      return
      end
```

```
      subroutine Uinit
     +  (irwst,jrn,icn,U,nrows,level,nrwspl,nptspl,nvlspl,npde,t)
c-----------------------------------------------------------------------
c
c In this subroutine the user provides the initial values as a
c function of the spatial co-ordinates.
c
c-----------------------------------------------------------------------
      double precision U,t,
     +                 x,y,x0,y0,dx,dy,dxl,dyl,ratio,par
      integer irwst,jrn,icn,nrows,level,nrwspl,nptspl,nvlspl,npde,
     +        ii,jj,idx
      dimension irwst(nrwspl),jrn(nrwspl),icn(nptspl),U(nvlspl)
      common /mesh/ x0,y0,dx,dy
      common /param/ par(1)
      ratio=2.0**(1-level)
      dxl=ratio*dx
      dyl=ratio*dy
      do 20 ii=1,nrows
        y=y0+jrn(ii)*dyl
        do 10 jj=irwst(ii),irwst(ii+1)-1
          x=x0+icn(jj)*dxl
          idx=npde*(jj-1)
c=======================================================================
c The user supplied block starts here
c=======================================================================
c-----------------------------------------------------------------------
c The computation of user defined variables needed to specify the IVs
c-----------------------------------------------------------------------
c
c-----------------------------------------------------------------------
c The specification of the IVs
c-----------------------------------------------------------------------
          U(idx+1)=par(6)+(1.0d0-y)*par(7)*par(5)
          U(idx+2)=0.0d0
c
c=======================================================================
   10   continue
   20 continue
      return
      end
```

```
      subroutine SOIL
     +  (irwst,jrn,icn,nrows,level,nrwspl,nptspl)
c-----------------------------------------------------------------------
c
c In this subroutine the user can deliver the values of the
c discontinuous soilparameters to MOORKOP2.1 as function of the
c spatial co-ordinates.
c The code assumes these soilparameters to be piecewise constants and
c that the interfaces do not move in time.
c
c-----------------------------------------------------------------------
      double precision SP,
     +                 x,y,x0,y0,dx,dy,dxl,dyl,ratio,par,
     +                 x1,x2,y1
      integer irwst,jrn,icn,nrows,level,nrwspl,nptspl,
     +        ii,jj
      dimension irwst(nrwspl),jrn(nrwspl),icn(nptspl)
      common /mesh/ x0,y0,dx,dy
      common /perm/ SP(1)
      common /param/ par(1)
      ratio=2.0d0**(1-level)
      dxl=ratio*dx
      dyl=ratio*dy
      do 20 ii=1,nrows
        y=y0+(jrn(ii)+0.5d0)*dyl
        do 10 jj=irwst(ii),irwst(ii+1)-1
          x=x0+(icn(jj)+0.5d0)*dxl
c-----------------------------------------------------------------------
c The specification of the discontinuous soil parameters
c-----------------------------------------------------------------------
c The integer index jj corresponds to a grid node.
c The co-ordinates x and y are the co-ordinates of the center of
c the cell of which jj is the lower left vertex.
c Thus the array element SP(jj) holds the value of a SOIL parameter
c at (x,y).
c
c  -----------------
c  |       |       |
c  |       |       |
c  |       |   o   |
c  |       |  x,y  |
c  |       |       |
c  --------o--------
c  |     jj|       |
c  |       |       |
c  |       |       |
c  |       |       |
c  |       |       |
c  -----------------
c
c For example, when there are 4 discontinuous soilparameters then they
c should be defined as
c
c      SP(jj)          = ...
c      SP(jj+nptspl)   = ...
c      SP(jj+2*nptspl)= ...
c      SP(jj+3*nptspl)= ...
c
c
c
c=======================================================================
c The user supplied block starts here
c=======================================================================
c
c Here the user can define the interfaces as functions of the
c spatial co-ordinates.
c
c-----------------------------------------------------------------------
          y1=.6001d0+.1d0*x
          x1=0.3001d0+0.2d0*y
          x2=0.7001d0
          if (y.lt.y1) then
            if (x.lt.x2) then
              SP(jj)=1.0d-10
              SP(jj+nptspl)=.01d0
              SP(jj+2*nptspl)=.002d0
            else
              SP(jj)=1.0d-13
              SP(jj+nptspl)=.008d0
              SP(jj+2*nptspl)=.0016d0
            end if
          else
            if (x.lt.x1) then
              SP(jj)=1.0d-13
              SP(jj+nptspl)=.008d0
              SP(jj+2*nptspl)=.0016d0
            else
              SP(jj)=1.0d-15
              SP(jj+nptspl)=.005d0
              SP(jj+2*nptspl)=.001d0
            end if
          end if
c
c=======================================================================
   10   continue
   20 continue
      return
      end
```

8. Concluding Remarks

Finally, we conclude with some remarks on MOORKOP 2.1. So far, we have computed the solution to two brine transport problems in inhomogeneous porous media with this code. This is described in [8]. We consider our results, obtained for these two problems as satisfactory, but nevertheless we think that two warnings are appropriate here. First, although the adaptation of the modified Newton method has improved the robustness of the code considerably, there still is a possibility that the code breaks down, simply because the (partially) interpolated initial guess for the iteration process (cf. Section 3) is too far away from the solution of the system of nonlinear equations at hand. A remedy to this could be to create finer grids always at interfaces (i.e. prevflg=.true.), whether this is necessary, regarding the space error monitor values, or not. The second warning has to do with the dispersion tensor. In [7] we show that the mathematical formulation of the dispersion tensor can cause serious difficulties for the iterative solution of the systems of nonlinear equations, even to the extent that a code can break down. This occurs when the velocity exhibits large changes in direction during the iterative solution process. This is likely to occur when the velocity becomes relatively small or when $\nabla p - \rho \mathbf{g}$ nearly vanishes, assuming the velocity vector is given by (4.1). This happens for example near stagnation points or near the kernel of a vortex. When brine transport problems in porous media with inhomogeneities are solved, stagnation points or points where the velocities are very small are not uncommon, so there is a real danger that the problems above occur.

References

1. S. Adjerid and J.E. Flaherty (1988). A local Refinement Finite Element Method for Two Dimensional Parabolic Systems, *SIAM J. Sci. Stat Comput.*, 9, 792-811.
2. J.G. Blom, J.G. Verwer, and R.A. Trompert (1992). *A Comparison between Direct and Iterative Methods to solve the Linear Systems arising from a Time-Dependent 2D Groundwater Flow Model,* Report NM-R9205 (to appear in Int. J. Comput. Fluid Dyn.), Centre for Mathematics and Computer Science, Amsterdam.
3. J.E. Dennis and R.B. Schnabel (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall Series in Computational Mathematics.
4. J.J. Dongarra and E. Grosse. Distribution of Mathematical Software via Electronic Mail, *Communications of the ACM*, 30, 403-407 (netlib@research.att.com).
5. S.M. Hassanizadeh and T. Leijnse (1988). On the Modelling of Brine Transport in Porous Media, *Water Resources Research*, 24, 321-330.
6. W. Schönauer and E. Schnepf (1987). Software Considerations for the Black Box Solver FIDISOL for Partial Differential Equations, *ACM Trans. on Math. Softw.*, 13, 333-349.
7. R.A. Trompert (1992). *A Note on Singularities Caused by the Hydrodynamic Dispersion Tensor,* (in preparation), Centre for Mathematics and Computer Science, Amsterdam.
8. R.A. Trompert (1992). *Local Uniform Grid Refinement and Brine Transport in Porous Media with Inhomogeneities,* NM-R9224, Centre for Mathematics and Computer Science, Amsterdam.
9. R.A. Trompert (1992). *Local Uniform Grid Refinement and Systems of Coupled Partial Differential Equations with and without Time Derivatives,* (in preparation), Centre for Mathematics and Computer Science, Amsterdam.
10. R.A. Trompert and J.G. Verwer (1990). *Analysis of the Implicit Euler Local Uniform Grid Refinement Method,* Report NM-R9011 (to appear in SIAM J. Sci. Stat. Comput.), Centre for Mathematics and Computer Science, Amsterdam.
11. R.A. Trompert and J.G. Verwer (1990). *Runge-Kutta Methods and Local Uniform Grid Refinement,* Report NM-R9022 (to appear in Math. Comp.), Centre for Mathematics and Computer Science, Amsterdam.
12. R.A. Trompert and J.G. Verwer (1991). A Static-Regridding Method for Two Dimensional Parabolic Partial Differential Equations, *Appl. Numer. Math.*, 8, 65-90.
13. R.A. Trompert, J.G. Verwer, and J.G. Blom (1992). *Computing Brine Transport in Porous Media*

with an Adaptive-Grid Method, Report NM-R9201 (to appear in Int. J. Numer. Meths. in Fluids), Centre for Mathematics and Computer Science, Amsterdam.

14. J.G. VERWER and R.A. TROMPERT (1991). *Local Uniform Grid Refinement for Time-Dependent Partial Differential Equations,* Report NM-R9105, Centre for Mathematics and Computer Science, Amsterdam.
15. J.G. VERWER and R.A. TROMPERT (1992). *Analysis of Local Uniform Grid Refinement,* Report NM-R9211, Centre for Mathematics and Computer Science, Amsterdam.
16. H.A. VAN DER VORST (1992). BI-CGSTAB: A Fast and Smoothly Converging Variant of BI-CG for the Solution of Nonsymmetric Linear Systems, *SIAM J. Sci. Stat. Comput.*, 13(2), 631-644.