13

A Finite-Volume-Based Module for Unsaturated Poroelasticity

JHABRIEL VARELA, SARAH E. GASDA, EIRIK KEILEGAVLEN, AND JAN MARTIN NORDBOTTEN

Abstract

In this chapter, we present `fv-unsat`, a multipoint finite-volume-based solver for unsaturated flow in deformable and nondeformable porous media. The latter is described using the mixed form of Richards' equation, whereas the former by the equations of unsaturated linear poroelasticity. The module aims at flexibility, relying heavily on discrete operators and equations, exploiting the automatic differentiation framework provided by the MATLAB Reservoir Simulation Toolbox (MRST). Our examples cover two numerical convergence tests and two three-dimensional practical applications, including the water infiltration process in a nondeformable soil column and a realistic desiccation process of a deformable clay sample using atmospheric boundary conditions. The resulting convergence rates are in agreement with previously reported rates for single-phase models, and the practical applications capture the physical processes accurately.

13.1 Introduction

The unsaturated zone has been a constant focus of attention by the industrial and research communities due to its high relevance in areas such as environmental sciences, hydrogeology, soil mechanics, and agriculture. Relevant natural and anthropogenic processes take place in this zone; transmission of water from the atmosphere to the saturated zone via infiltration or precipitation, support of plants via root uptake, active return of water from the subsurface to the atmosphere via evapotranspiration, drying of soils during drought seasons, extraction of groundwater via wells, construction and operations of dams, etc. [46].

Although many of these processes can be studied by only taking into account the simultaneous flow of water and air, some of them, such as the desiccation of muddy soils, require the incorporation of the deformation effects due to the

https://doi.org/10.1017/9781009019781.019 Published online by Cambridge University Press

strong coupling between flow and mechanics. This gives rise to a nonlinear coupled flow/mechanical set of partial differential equations. Under the assumption of small deformations and linear constitutive relations for the mechanical behavior of the soils, this set of equations can be expressed as a natural extension of Biot's equations of poroelasticity [26], for which recently global existence of the weak solution has been proven [9].

Given the complexity of the resulting model, it is imperative to use robust discretization techniques in a flexible computational setting. The fully coupled system is not commonly treated by numerical software, and the few available codes are limited to the use of finite-element methods [26] or mixed finite-element methods [7]. In this module, we propose the use of finite-volume methods (FVM), which are inherently conservative while keeping the advantages of robust discretization schemes; i.e., flexibility in representing complex domains.

In the FVM framework, two-point flux approximation (TPFA) is the most widely used method for discretizing scalar equations. However, TPFA is only consistent for K-orthogonal grids [1] and cannot be directly applied to vector equations. The first of these issues can be addressed with the multipoint flux approximation (MPFA) method [1], and the second with the multipoint stress approximation (MPSA) method [33]. Both methods are currently well established in academia and slowly taking hold in industry.

As we mentioned before, computational flexibility is an important aspect of a module when it comes to solving a broad range of applications. With this goal in mind, we have written `fv-unsat` taking advantage of the high-level coding capacities of the MATLAB Reservoir Simulation Toolbox (MRST), such as automatic differentiation [24]. This module is based on the work of [44] and requires the module `fvbiot`, which provides the discrete MPFA and MPSA operators, along with the coupling operators for the flow/mechanical problem.[1]

The existing implementation of `fv-unsat` does not cover the full width of modeling options of the governing equations; e.g., mixed and time-dependent boundary conditions for the mechanical problem. If the interested user needs to include these setups, we recommend the Python-based framework PorePy [20], which provides a more general implementation of MPSA for poroelastic problems.

Our notation follows MRST's conventions [28]. In physical space, x represents a scalar, $\vec{x}$ a vector, and $\mathbf{x}$ a tensor. In a discrete sense, $\boldsymbol{x}$ is a vector and $\texttt{ope}(\boldsymbol{x})$ is a discrete operator acting on $\boldsymbol{x}$; i.e., the matrix–vector product between `ope` and $\boldsymbol{x}$.

The chapter is structured as follows: in Section 13.2 we provide the continuous formulations for the unsaturated flow in nondeformable (Richards' equation) and

[1] After this chapter was written, `mpsaw`, a new and improved implementation of the MPSA-W method has been released with the core MRST distribution. The module can also be downloaded separately at https://bitbucket.org/mrst/mpsaw/src/master/.

deformable (unsaturated poroelasticity) porous media; in Section 13.3 we introduce the MPFA and MPSA methods, together with the discrete operators and the discrete equations; in Section 13.4 we present two numerical convergence tests and two practical applications with in-depth explanation regarding the module; and in Section 13.5 we draw the conclusions.

13.2 Governing Equations

In this section, we provide the set of equations that governs the physical processes in the continuous domain. We do not attempt to provide detailed derivations of these equations; for that matter we refer to [13, 26, 37].

13.2.1 Richards' Equation

Richards' equation models the flow of water in partially saturated porous media, and it is based on the assumption of inviscid air. This assumption is supported by the contrast in physical properties between water and air; e.g., at atmospheric conditions air is three orders of magnitude less dense and two orders less viscous than water [37]. Because the unsaturated zone is connected to the atmosphere, it is reasonable to assume that the air remains at atmospheric pressure. This is usually referred to as the Richards assumption and it was first proposed in [40].

We start the derivation by stating the mass-balance equation for the water phase

$$\frac{\partial\left(\rho_w S_w n\right)}{\partial t}+\nabla\cdot\left(\rho_w S_w n\vec{v}_w\right)=\dot{m}_w. \tag{13.1}$$

Here, ρ_w and S_w are the density and saturation, n is the porosity of the porous medium, $\vec{v}_w$ is the water velocity, and $\dot{m}_w$ is the rate of external addition/subtraction of fluid mass per volume of representative elementary volume [5]. If water and solid phases are assumed to be incompressible, we can rewrite (13.1) as

$$n\rho_w\frac{\partial S_w}{\partial t}+\rho_w\nabla\cdot\left(S_w n\vec{v}_{ws}\right)=\dot{m}_w, \tag{13.2}$$

where $\vec{v}_{ws}:=\vec{v}_w-\vec{v}_s$ is the velocity of the water with respect to the solids [26]. We recognize the term $S_w n\vec{v}_{ws}$ as the Darcy velocity of the water phase, given by

$$\vec{q}_w=S_w n\vec{v}_{ws}=-\frac{\mathbf{k}}{\mu_w}k_{rw}\left(\nabla p_w-\rho_w\vec{g}\right), \tag{13.3}$$

where $\mathbf{k}$ is the intrinsic permeability tensor, μ_w is the water dynamic viscosity, p_w is the water pressure, and $\vec{g}$ is the gravity acceleration considered positive downwards. The relative permeability $k_{rw}\in[0,1]$ is included to account for the simultaneous flow of water and air.

In hydrology, it is common to express Darcy's law (13.3) in terms of heads,

$$\vec{q}_w = -\mathbf{K}_w^{sat} k_{rw} \nabla \left(\psi_w + \zeta\right). \tag{13.4}$$

Here, $\psi_w = (p_w - p_a)/(\rho_w g)$ is the water pressure head (relative to the atmospheric pressure p_a), $\zeta = z - z_0$ is the elevation head (e.g., the height from a reference to the measurement point), and $\mathbf{K}_w^{sat} := \rho_w g \mathbf{k}/\mu_w$ is the hydraulic conductivity at saturated conditions [17, 37].

If Richards' assumption holds true, the air pressure is constant and equal to p_a, which is assumed to be zero. This allows us to write the equations purely in terms of the water phase. Note that the capillary pressure is still present; i.e., $p_c = p_a - p_w = -p_w$. To get to the final expression, we substitute (13.4) into (13.2) and divide by ρ_w:

$$\frac{\partial \theta_w}{\partial t} - \nabla \cdot \left(\mathbf{K}_w^{sat} k_{rw} \nabla \left(\psi_w + \zeta\right)\right) = \frac{\dot{m}_w}{\rho_w}, \tag{13.5}$$

where we introduce the water content $\theta_w := n S_w$ and use the fact that the porosity is constant. Equation (13.5) is referred to as the mixed-based form of Richards' equation. The term "mixed" suggests that both the water content and the pressure head appear explicitly in the equation. Alternative formulations include the pressure head–based and the water content–based forms [37]. On a continuous level, all forms of Richards' equation are equivalent under strictly unsaturated conditions. However, on a discrete level, the ψ-based lacks conservative properties [12] and the θ-based fails to converge when $S_w \to 1$ [38]. Therefore, in this module, we employ the mixed-based formulation.

In unsaturated systems, the usual practice is to express k_{rw} and θ_w in terms of ψ_w. These relationships are called soil/water retention curves (SWRCs). One such family of curves is the van Genuchten–Mualem (vG-M) model, originally proposed in [43]. For the vG-M model, the water content is given by

$$\theta_w = \begin{cases} \dfrac{\theta_w^s - \theta_w^r}{\left[1 + \left(\alpha_v \left|\psi_w\right|\right)^{n_v}\right]^{m_v}} + \theta_w^r, & \psi_w < 0, \\ \theta_s^w, & \psi_w \geq 0, \end{cases} \tag{13.6}$$

where θ_w^s and θ_w^r are the water content at saturated and residual conditions and α_v, n_v, and m_v are fitting parameters. Note that $\psi_w < 0$ denotes unsaturated conditions. The relative permeability is given by

$$k_{rw} = \begin{cases} \dfrac{\left\{1 - \left(\alpha_v \left|\psi_w\right|\right)^{n_v - 1} \left[1 + \left(\alpha_v \left|\psi_w\right|\right)^{n_v}\right]^{-m_v}\right\}^2}{\left[1 + \left(\alpha_v \left|\psi_w\right|\right)^{n_v}\right]^{m_v/2}}, & \psi_w < 0, \\ 1, & \psi_w \geq 0. \end{cases} \tag{13.7}$$

We also introduce the specific moisture capacity $C_\psi := \mathrm{d}\theta_w/\mathrm{d}\psi_w$:

$$C_\psi = \begin{cases} -\dfrac{m_v n_v \psi_w \left(\theta_w^s - \theta_w^r\right) \left(\alpha_v \left|\psi_w\right|\right)^{n_v}}{\left|\psi_w\right|^2 \left[\left(\alpha_v \left|\psi_w\right|\right)^{n_v} + 1\right]^{m_v+1}}, & \psi_w < 0, \\ 0, & \psi_w \geq 0. \end{cases} \tag{13.8}$$

13.2.2 Unsaturated Poroelasticity

Herein, we present the equations that govern an unsaturated poroelastic medium as a natural extension of Biot's equations [26]. The momentum conservation for an unsaturated poroelastic medium reads

$$\nabla \cdot \boldsymbol{\sigma}_t + \big((1-n)\rho_s + nS_w\rho_w\big)\vec{g} = 0, \tag{13.9}$$

where $\boldsymbol{\sigma}_t$ is the total stress tensor and ρ_s the density of the solids, with the second term representing the body forces. For a poroelastic medium, the total stress has two contributions: the part that acts on the solid skeleton and the part that acts on the fluid. The relation is given by the extended principle of effective stress [16],

$$\boldsymbol{\sigma}_t = \boldsymbol{\sigma}_e - \alpha p_w S_w \mathbf{I}. \tag{13.10}$$

The term $\boldsymbol{\sigma}_e$ is the effective stress tensor, and it is responsible for causing the actual deformation of the material; thus the name "effective" [45]. The second term affects the pore pressure of the fluid, where α is the Biot coupling coefficient and $\mathbf{I}$ is the identity tensor. The negative sign follows the convention that tensile forces are positive whereas compressive forces are negative [31]. Substitution of (13.10) into (13.9) gives the final version of the unsaturated momentum balance equation,

$$\nabla \cdot \boldsymbol{\sigma}_e - \alpha\nabla\left(S_w p_w\right) + \big((1-n)\rho_s + nS_w\rho_w\big)\vec{g} = 0. \tag{13.11}$$

Assuming small deformations and a linear stress–strain relation, the effective stress $\boldsymbol{\sigma}_e$ can be related to the displacement field $\vec{u}$ employing the generalized Hooke's law

$$\boldsymbol{\sigma}_e = \mathbf{C} : \tfrac{1}{2}\big(\nabla\vec{u} + (\nabla\vec{u})^T\big), \tag{13.12}$$

where $\mathbf{C}$ is the stiffness matrix, a fourth-order tensor in its most general form. For the particular case of an isotropic medium, (13.12) can be written as

$$\boldsymbol{\sigma}_e = \mu_s\big(\nabla\vec{u} + (\nabla\vec{u})^T\big) + \lambda_s\left(\nabla\cdot\vec{u}\right)\mathbf{I}, \tag{13.13}$$

where λ_s and μ_s are the first and second Lamé parameters [30].

A statement of the mass conservation principle for both phases (water and solid skeleton) can be used to derive the unsaturated storage equation (see [44] for a detailed derivation):

$$\xi(S_w)\frac{\partial p_w}{\partial t} + \chi(S_w,p_w)\frac{\partial S_w}{\partial t} + \alpha S_w \frac{\partial}{\partial t}(\nabla\cdot\vec{u}) + \nabla\cdot\vec{q}_w = \frac{\dot{m}_w}{\rho_w}, \tag{13.14}$$

where $\xi := (\alpha - n)\,C_s S_w^2 + nC_w S_w$ and $\chi := (\alpha - n)\,C_s S_w p_w + n$ are compressibility-like terms. In (13.14), the first two terms represent accumulation terms, the third term is the change of strain at constant saturation, the fourth term is the divergence of the Darcy velocity, and the last terms are sources or sinks of water [41].

Note that the above set of equations is written in terms of (p_w, S_w) instead of (ψ_w, θ_w). Because the SWRC is expressed in terms of the latter variables, we have to adapt the original vG-M model to be consistent with the (p_w, S_w) representation. This can be easily achieved using the following relations:

$$\psi_w = \frac{p_w}{\rho_w g}, \qquad \theta_w = nS_w, \qquad C_\psi = n\rho_w g C_p,$$

where all of the terms have been previously introduced, except the specific saturation capacity $C_p := \partial S_w/\partial p_w$.

13.2.3 Boundary and Initial Conditions

To close the systems of partial differential equations, we must provide boundary and initial conditions for the flow and mechanical problems. For the flow problem, two types of conditions can be specified: pressure (or pressure head) and fluxes. For the mechanical problem, we can impose displacement and traction force vectors. Denoting Ω the domain of interest and $\partial\Omega$ its boundary, the boundary conditions are given by

$$p_w = g_{p,D} \quad \text{on} \quad \Gamma_{p,D}, \tag{13.15}$$
$$\vec{q}_w\cdot\vec{n} = g_{p,N} \quad \text{on} \quad \Gamma_{p,N}, \tag{13.16}$$
$$\vec{u} = g_{\vec{u},D} \quad \text{on} \quad \Gamma_{\vec{u},D}, \tag{13.17}$$
$$\boldsymbol{\sigma}_t\cdot\vec{n} = g_{\vec{u},N} \quad \text{on} \quad \Gamma_{\vec{u},N}, \tag{13.18}$$

where $\vec{n}$ is the normal vector pointing outwards, and the subindices D and N denote Dirichlet and Neumann boundary conditions. The boundary of the domain is given by $\partial\Omega = \Gamma_D \cup \Gamma_N$ with $\Gamma_D \cap \Gamma_N = \emptyset$.

The initial conditions are specified as

$$p_w = p_{w,0} \quad \text{for} \quad t = 0, \tag{13.19}$$
$$\vec{u} = \vec{u}_0 \quad \text{for} \quad t = 0. \tag{13.20}$$

13.3 Discretization and Implementation

This section is devoted to the discretization techniques and computational implementation. First, we briefly introduce the numerical methods; e.g., MPFA/MPSA finite-volume (FV) schemes. Then, we employ the discrete operators to derive the discrete version of the governing equations. Finally, we describe the general strategy for solving the resulting nonlinear set of equations. In particular, we discuss the workflow of the iterative solver and the timestepping algorithm.

13.3.1 MPFA and MPSA

Before writing the discrete version of the governing equations, we briefly introduce the MPFA and MPSA methods. From an implementation standpoint, the discretization routines for both methods are provided by the third-party module `fvbiot`. Nevertheless, we should remark that TPFA is the standard scheme employed in MRST for the discretization of flow equations. In addition, MRST provides an alternative MPFA implementation based on the mimetic method available through the `mpfa` module (see section 6.4 of the MRST textboox [28] for further details). Because both techniques (MPFA and MPSA) are well established in the literature we do not go in-depth. We refer to [1, 4, 23] for an introduction to MPFA and to [21, 33] for an introduction to MPSA.

MPFA

In an FVM framework applied to the flow problem, we aim to discretize the integrated version of (13.4) over a face. For a cell-centered FVM, we use the cell-centered pressures to estimate the fluxes across the faces; i.e., $Q = \int_S \vec{q} \cdot \vec{n}\, \mathrm{d}\Sigma$. Hence, for a given face, we have to define the number of points to be considered for approximating Q.

The simplest choice is to consider two points, say, 1 and 2 from top Figure 13.1. This technique is referred to as TPFA, with the flux across the shared face j given by

$$Q_j \approx \lambda_j T_j (p_1 - p_2), \tag{13.21}$$

where Q_j is the water flux, $\lambda_j = k_{rw,j}/\mu_w$ is the water mobility, and T_j is the transmissibility. For readability, we drop the subindices denoting the water phase.

The MPFA method is a generalization of the TPFA method, where instead of using two points of information, we use a larger set of potentials (see bottom of Figure 13.1). For the MPFA method, the flux can be approximated as

$$Q_j \approx \lambda_j \sum_{i \in \mathcal{I}} t_{ij} p_i, \tag{13.22}$$

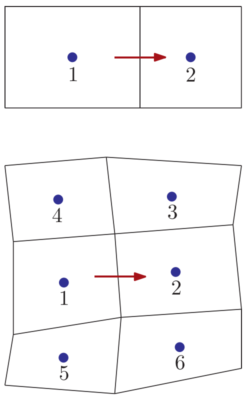


Figure 13.1 Flux approximations: TPFA (top) relies on first neighbors only, whereas MPFA (bottom) also includes second neighbors. Adapted from [1]

where t_{ij} are the transmissibility coefficients satisfying $\sum_{i\in\mathcal{I}} t_{ij} = 0$, and $\mathcal{I}$ is the set of of cells used to approximate the flux through the face j. The size of the set $\mathcal{I}$ depends on the dimensionality of the problem and the type of element employed. For quadrilaterals, the set $\mathcal{I}$ consists of six neighbors. With this increase in accuracy, MPFA results in a consistent discretization method compared to TPFA (which gives nonphysical results when applied to non-K-orthogonal grids) [1]. An interesting discussion regarding consistency of the numerical methods can be found in chapter 6 of the MRST textbook [28].

Mobilities λ_j are evaluated at the faces using either an arithmetic mean or an upstream weighting of the cell-centered values. The arithmetic mean implies $\lambda_j = (\lambda_1 + \lambda_2)/2$, whereas the upstream weighting is based on the flux direction; i.e., $\lambda_j = \lambda_1$ if $\sum_{i\in\mathcal{I}} t_{ij} p_i > 0$ ($T_j(p_1 - p_2) > 0$ for TPFA) and $\lambda_j = \lambda_2$ otherwise [1].

Provided that the pressures are known, both problems are reduced to determining T_j and t_{ij}. For TPFA, these are given by the harmonic average; however, finding t_{ij} is more complicated. Several families of MPFA methods obtain t_{ij} in different ways. The key difference among the methods lies in the way interaction regions are constructed and continuity points selected. Interaction regions are composed of the relevant neighboring cells and identified using the dual of the mesh (see Figure 13.2). We refer to [14] for an excellent discussion on the topic. In this module, we use the MPFA-O method, implemented in the `fvbiot` module.

MPSA

In recent years, the MPSA method was developed as a generalization of the MPFA method applied to vector equations, such as the Navier–Lamé equations [21, 33] or

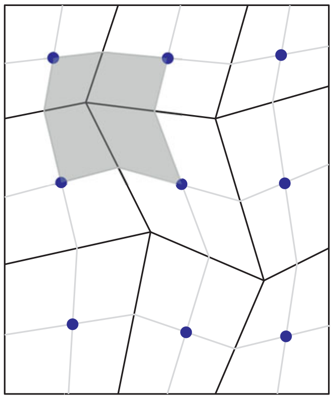


Figure 13.2 Dual mesh (light gray), conservation cells (black), and interaction region for the O-method (shaded). Adapted from [33]

the Biot equations [35]. MPSA uses the displacements $\vec{u}$ located at the cell centers as the only primary unknowns with the traction forces $\vec{T} = \int_S \boldsymbol{\sigma}_e \cdot \vec{n}\, \mathrm{d}\Sigma$ defined on the faces. On each face, the traction is linearly approximated by

$$\vec{T}_j \approx \sum_{i \in \mathcal{I}} s_{ij} \vec{u}_i, \tag{13.23}$$

where $s_{ij} = -s_{ij}$ are the stress weight tensors, and $\mathcal{I}$ is the set of neighboring cells to the face j. In essence, (13.23) can be seen as a local version of Hooke's law (13.12). Now the problem is reduced to the calculation of the stress weight tensors s_{ij} for each face of the domain. Similar to MPFA, there are several ways to estimate s_{ij} depending on the continuity points. The procedure for calculating the stress weights is beyond the scope of this chapter; we refer to [21, 33] for further details. The `fvbiot` module provides the MPSA-W version from [21], which is used herein.

13.3.2 Discretization

Herein we introduce the discrete MPFA/MPSA operators and discretize the governing equations. The way discrete operators are defined in our module is heavily inspired by MRST's rapid prototyping philosophy. In particular, they are in agreement with the basic structure of the simulators based on automatic differentiation utilized in MRST; see, for example, chapter 7 of the MRST textbook [28] for an excellent introduction. As the reader will note, this enables us to write the discrete equations in a fairly compact way, while simultaneously providing a concise way to structure the code.

Table 13.1 *Definition of the MPFA/MPSA operators.*

Description	Mapping	Operator dimension
Flux	$\texttt{F} : \mathbb{P} \to \mathbb{F}$	$N_f \times N_c$
Flux boundaries	$\texttt{boundF} : \mathbb{F} \to \mathbb{F}$	$N_f \times N_f$
Flux divergence	$\texttt{divF} : \mathbb{F} \to \mathbb{P}$	$N_c \times N_f$
Stress	$\texttt{S} : \mathbb{U} \to \mathbb{S}$	$dN_f \times dN_c$
Stress boundaries	$\texttt{boundS} : \mathbb{S} \to \mathbb{S}$	$dN_f \times dN_f$
Stress divergence	$\texttt{divS} : \mathbb{S} \to \mathbb{U}$	$dN_c \times dN_f$
Pressure gradient	$\texttt{gradP} : \mathbb{P} \to \mathbb{U}$	$dN_c \times N_c$
Displacement divergence	$\texttt{divU} : \mathbb{U} \to \mathbb{P}$	$N_c \times dN_c$
Compatibility	$\texttt{compat} : \mathbb{P} \to \mathbb{P}$	$N_c \times N_c$

Discrete MPFA/MPSA Operators

Let d denote the dimensionality of the problem – i.e., $d = 2,3$ – and let N_c and N_f represent the number of cells and faces of a nonoverlapping domain Ω. Each cell of the domain is identified as Ω_i and its enclosed surface as $\partial\Omega_i$.

We first introduce the discrete version of the variables of interest; i.e., pressure, displacement, flux, and traction:

$$\boldsymbol{p} := \{p_1, \cdots, p_{N_c}\}^T \in \mathbb{P}, \qquad \mathbb{P} = \mathbb{R}^{N_c}, \tag{13.24}$$

$$\boldsymbol{u} := \{\vec{u}_1, \cdots, \vec{u}_{N_c}\}^T \in \mathbb{U}, \qquad \mathbb{U} = \mathbb{R}^{dN_c}, \tag{13.25}$$

$$\boldsymbol{Q} := \{Q_1, \cdots, Q_{N_f}\}^T \in \mathbb{F}, \qquad \mathbb{F} = \mathbb{R}^{N_f}, \tag{13.26}$$

$$\boldsymbol{T} := \{\vec{T}_1, \cdots, \vec{T}_{N_f}\}^T \in \mathbb{S}, \qquad \mathbb{S} = \mathbb{R}^{dN_f}. \tag{13.27}$$

For vector-valued quantities, such as displacement and traction, the length of the vector depends on the dimensionality of the problem. For example, for a 2D problem using two cells, $\boldsymbol{u} = \{u_1, u_2, u_3, u_4\}^T = \{u_{1_x}, u_{1_y}, u_{2_x}, u_{2_y}\}^T$.

Following MRST's operator-based approach, in Table 13.1 we introduce the discrete MPFA and MPSA operators along with the coupling operators. The first three operators are related to the discretization of flow problems: $\texttt{F}(\cdot)$ acts on the potential and computes the fluxes (by first determining t_{ij} and then computing the gradient of the potential); $\texttt{boundF}(\cdot)$ deals with the boundary conditions; i.e., either constant pressure or constant flux. This operator will take care of the mapping from boundary values to the right discretization, keeping track of how Neumann and Dirichlet conditions should be treated differently. Finally, $\texttt{divF}(\cdot)$ computes the divergence of the flux, mapping back from faces to cell centers.

The next three operators are analogous to the first three, `S(·)` acting on the displacement, `boundS(·)` acting on the mechanic boundary conditions, and `divS(·)` computing the divergence of the (integrated) stress.

The last three operators are necessary for the coupled mechanics flow setting; `gradP(·)` computes the gradient of the pressure, `divU(·)` takes the divergence of the displacement, and `compat(·)` is a compatibility operator. This last operator (which acts on the pressure) arises naturally from the discretization process. This term has the physical interpretation of representing the volumetric expansion (or contraction) of a grid cell in response to the deviation in pressure of the cell relative to its neighbors. It is especially relevant when small timesteps are employed, providing stability to the discretized coupled system [35].

Discrete Richards' Equation

Having defined the discrete operators, we can write the discrete version of the governing equations. In an FVM framework, we typically integrate the mass conservation equation (13.5) over a cell volume,

$$\int_{\Omega_i} \frac{\partial \theta_w}{\partial t}\,\mathrm{d}V + \int_{\Omega_i} \nabla \cdot \vec{q}_w\,\mathrm{d}V = \int_{\Omega_i} \frac{\dot{m}_w}{\rho_w}\,\mathrm{d}V, \quad \forall i \in [1, N_c]. \tag{13.28}$$

Assuming that the equation is solved using an iterative strategy (see Subsection 13.3.3), after applying backward Euler, the accumulation term from (13.28) becomes

$$\frac{\partial \theta_w}{\partial t} = \frac{\theta_w^{n+1,m+1} - \theta_w^n}{\Delta t^n}, \tag{13.29}$$

where n denotes the time level and m the iteration level, and Δt is the timestep. As suggested in [12], to ensure local mass conservation, we use the modified Picard iteration to Taylor-expand $\theta_w^{n+1,m+1}$ from (13.29) as a function of ψ_w,

$$\theta_w^{n+1,m+1} = \theta_w^{n+1,m} + C_\psi^{n+1,m}\left(\psi_w^{n+1,m+1} - \psi_w^{n+1,m}\right) + H.O.T. \tag{13.30}$$

Using (13.29) and (13.30) with the higher-order terms neglected and computing the integral, the accumulation term from (13.28) is given by

$$\int_{\Omega_i} \frac{\partial \theta_w}{\partial t}\,\mathrm{d}V = \frac{V_i}{\Delta t^n}\left[\theta_{w,i}^{n+1,m} + C_{\psi,i}^{n+1,m}\left(\psi_{w,i}^{n+1,m+1} - \psi_{w,i}^{n+1,m}\right) - \theta_{w,i}^n\right], \quad \forall i \in [1, N_c],$$

where V_i is the volume of the cell i. Alternatively, we can write the previous equation in vector form as

$$\int_{\mathbf{\Omega}} \frac{\partial \theta_w}{\partial t}\,\mathrm{d}V = \frac{\boldsymbol{V}}{\Delta t^n}\left(\boldsymbol{\theta}_w^{n+1,m} + \boldsymbol{C}_\psi^{n+1,m}\left(\boldsymbol{\psi}_w^{n+1,m+1} - \boldsymbol{\psi}_w^{n+1,m}\right) - \boldsymbol{\theta}_w^n\right), \tag{13.31}$$

where with a slight abuse of notation, we denote

$$\int_{\mathbf{\Omega}} \frac{\partial \theta_w}{\partial t}\,\mathrm{d}V = \left\{\int_{\Omega_1} \frac{\partial \theta_w}{\partial t}\,\mathrm{d}V, \ldots, \int_{\Omega_{N_c}} \frac{\partial \theta_w}{\partial t}\,\mathrm{d}V\right\}^T.$$

In (13.31), $\boldsymbol{V} := \{V_1, \cdots, V_{N_c}\}^T$ is a vector representing the volumes of each cell of the domain. Note that the product between vectors should be interpreted as element-wise multiplications. An analogous procedure gives the expression for the source term,

$$\int_{\mathbf{\Omega}} \frac{\dot{m}_w}{\rho_w}\,\mathrm{d}V = \boldsymbol{V}\frac{\dot{\boldsymbol{m}}_w^n}{\rho_w}. \tag{13.32}$$

Applying the divergence theorem, the second term of (13.28) can be written as

$$\int_{\Omega_i} \nabla\cdot\vec{q}_w\,\mathrm{d}V = \int_{\partial\Omega_i} \vec{q}_w\cdot\vec{n}\,\mathrm{d}A = \sum_{j\in\mathcal{F}_i} \vec{q}_{w,j}\cdot\vec{n}_j A_j = \sum_{j\in\mathcal{F}_i} Q_j, \quad \forall i \in [1, N_c],$$

where $\mathcal{F}_i$ is the set of faces associated with the cell i. Alternatively, in vector form,

$$\int_{\mathbf{\Omega}} \nabla\cdot\vec{q}_w\,\mathrm{d}V = \texttt{divF}\left(\boldsymbol{Q}_w\right), \tag{13.33}$$

where we use the discrete divergence operator divF acting on $\boldsymbol{Q}_w$.

Combining (13.31), (13.32), and (13.33), we can write the discrete version of mass conservation as

$$\frac{\boldsymbol{V}}{\Delta t^n}\left(\boldsymbol{\theta}_w^{n+1,m} + \boldsymbol{C}_\psi^{n+1,m}\left(\boldsymbol{\psi}_w^{n+1,m+1} - \boldsymbol{\psi}_w^{n+1,m}\right) - \boldsymbol{\theta}_w^n\right) + \texttt{divF}\left(\boldsymbol{Q}_w\right) = \boldsymbol{V}\frac{\dot{\boldsymbol{m}}_w^n}{\rho_w}. \tag{13.34}$$

The discrete version of the Darcy flux through a face j is given by

$$Q_{w,j} = \frac{\rho_w g}{\mu_w}\breve{k}_{rw,j}^{n+1,m}\sum_{i\in\mathcal{I}} t_{ij}\left(\psi_{w,i}^{n+1,m+1} + \zeta_i\right), \quad \forall j \in \left[1, N_f\right],$$

where $\breve{k}_{rw,j}$ denotes the relative permeabilities evaluated at the faces; i.e., obtained by arithmetic average or upstream weighting. The previous equation written in vector form reads

$$\boldsymbol{Q}_w = \frac{\rho_w g}{\mu_w}\breve{\boldsymbol{k}}_{rw}^{n+1,m}\left(\texttt{F}(\boldsymbol{\psi}_w^{n+1,m+1} + \boldsymbol{\zeta}) + \texttt{boundF}(\boldsymbol{b}_f)\right), \tag{13.35}$$

where $\boldsymbol{b}_f \in \mathbb{F}$ is the vector of flow boundary conditions. Equations (13.34) and (13.35) represent a closed system of nonlinear algebraic equations, perfectly suited for an iterative solver. Finally, note that even though the physical model is referred to as the mixed-based version, the discretized version of the model is solved only for the pressure head $\boldsymbol{\psi}_w^{n+1,m+1}$, because we can to express $\theta_w = \theta_w(\psi_w)$ from (13.6).

Discrete Equations of Unsaturated Poroelasticity

Following the same procedure as in the Richards equation, the unsaturated storage equation (13.14) in vector form is given by

$$\begin{aligned}\boldsymbol{V}\boldsymbol{\xi}^n\left(\boldsymbol{p}_w^{n+1,m+1}-\boldsymbol{p}_w^n\right)+\boldsymbol{V}\boldsymbol{\chi}^n\left(\boldsymbol{S}_w^{n+1,m}+\boldsymbol{C}_p^{n+1,m}\left(\boldsymbol{p}_w^{n+1,m+1}-\boldsymbol{p}_w^{n+1,m}\right)-\boldsymbol{S}_w^n\right)\\ +\alpha\boldsymbol{S}_w^n\texttt{divU}\left(\boldsymbol{u}^{n+1,m+1}-\boldsymbol{u}^n\right)+\alpha^2\texttt{compat}\left(\boldsymbol{S}_w^n\boldsymbol{p}_w^{n+1,m+1}\right)\\ +\Delta t^n\texttt{divF}\left(\boldsymbol{Q}_w\right)=\boldsymbol{V}\Delta t^n\frac{\dot{\boldsymbol{m}}_w^n}{\rho_w},\end{aligned} \tag{13.36}$$

where the time derivatives are approximated using backward Euler and we applied the modified Picard iteration to Taylor-expand $S_w^{n+1,m+1}$ in terms of p_w. The `compat` operator appears naturally in the MPFA/MPSA discretization of the coupled system and provides compatibility when $\Delta t^n \ll 1$. We choose to evaluate the accumulation-like terms ξ and χ at the time level n to reduce the nonlinearities; nevertheless, we acknowledge that other choices are possible.

The Darcy flux (integrated version of (13.3)) in terms of pressure reads

$$\boldsymbol{Q}_w=\frac{1}{\mu_w}\check{\boldsymbol{k}}_{rw}^{n+1,m}\left(\texttt{F}\left(\boldsymbol{p}_w^{n+1,m+1}+\rho_w g\boldsymbol{\zeta}\right)+\texttt{boundF}\left(\boldsymbol{b}_f\right)\right). \tag{13.37}$$

The (semidiscrete) unsaturated linear momentum equation (13.11) can be integrated over each cell of the domain, giving

$$\begin{aligned}\int_{\Omega_i}\nabla\cdot\boldsymbol{\sigma}_e\,\mathrm{d}V-\int_{\Omega_i}\alpha\nabla\left(S_w^n p_w^{n+1,m+1}\right)\mathrm{d}V\\ +\int_{\Omega_i}\left[(1-n)\rho_s+nS_w^n\rho_w\right]\vec{g}\,\mathrm{d}V=0,\quad\forall i\in[1,N_c].\end{aligned} \tag{13.38}$$

Applying the divergence theorem, the first term from (13.38) can be written as

$$\int_{\Omega_i}\nabla\cdot\boldsymbol{\sigma}_e\,\mathrm{d}V=\int_{\partial\Omega_i}\boldsymbol{\sigma}_e\cdot\vec{n}\,\mathrm{d}A=\sum_{j\in\mathcal{F}_i}\boldsymbol{\sigma}_{e,j}\cdot\vec{n}_j\,A_j=\sum_{j\in\mathcal{F}_i}\vec{T}_j,\quad\forall i\in[1,N_c],$$

or in vector form as

$$\int_{\Omega} \nabla \cdot \boldsymbol{\sigma}_e \, \mathrm{d}V = \texttt{divS}(\boldsymbol{T}). \tag{13.39}$$

The second term of (13.38) is given by

$$\int_{\Omega} \alpha \nabla \left(S_w^n p_w^{n+1,m+1} \right) \, \mathrm{d}V = \alpha \texttt{gradP} \left(\boldsymbol{S}_w^n \boldsymbol{p}_w^{n+1,m+1} \right), \tag{13.40}$$

whereas the discretization of the body forces reads

$$\int_{\Omega} \left[(1-n)\rho_s + n S_w^n \rho_w \right] \vec{g} \, \mathrm{d}V = \texttt{dnc}(\boldsymbol{V}) \left((1-n)\rho_s + n \texttt{dnc}(\boldsymbol{S}_w^n)\rho_w \right) \boldsymbol{g}. \tag{13.41}$$

Here, we have used the $\texttt{dnc}(\cdot)$ operator, which converts a vector of length N_c to a vector of length dN_c by repeating each element of the N_c vector d times. For example, for a 2D problem with two cells, $\texttt{dnc}(\boldsymbol{V}) = \texttt{dnc}(\{V_1, V_2\}^T) = \{V_1, V_2, V_1, V_2\}^T$.

Combining (13.39), (13.40), and (13.41) gives the discrete version of the momentum equation in vector form,

$$\begin{aligned} &\texttt{divS}\,(\boldsymbol{T}) - \alpha \texttt{gradP} \left(\boldsymbol{S}_w^n \boldsymbol{p}_w^{n+1,m+1} \right) \\ &\quad + \texttt{dnc}\,(\boldsymbol{V}) \left((1-n)\rho_s + n \texttt{dnc}(\boldsymbol{S}_w^n)\rho_w \right) \boldsymbol{g} = \boldsymbol{0}. \end{aligned} \tag{13.42}$$

Finally, for a generic face j, the traction forces acting on that face are given by

$$\vec{T}_j = \sum_{i \in \mathcal{I}} s_{ij} \vec{u}_i^{n+1,m+1}, \quad \forall j \in \left[1, N_f\right],$$

or in vector form,

$$\boldsymbol{T} = \texttt{S}(\boldsymbol{u}^{n+1,m+1}) + \texttt{boundS}\,(\boldsymbol{b_m}), \tag{13.43}$$

where $\boldsymbol{b_m} \in \mathbb{S}$ is the vector of boundary conditions for the mechanical problem. Equations (13.36), (13.37), (13.42), and (13.43) represent the complete set of discrete equations. This set of equations can be solved using a sequential approach [6, 22] or a monolithic approach [34]. The latter is the preferred method for this module, with the vector $\{\boldsymbol{u}_w^{n+1,m+1}, \boldsymbol{p}_w^{n+1,m+1}\}^T$ as the only compound primary variable.

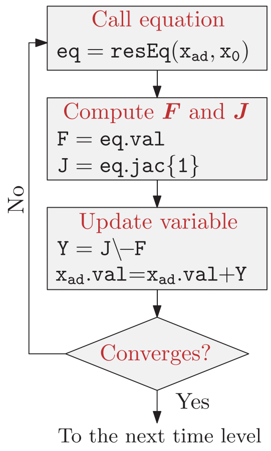


Figure 13.3 Workflow of the iterative solver applied to a generic equation.

13.3.3 Solving the Equations

To solve the system of equations we implement the modified Picard iteration method as a part of an iterative solver as presented in the MRST textbook [28]. Other types of linearization schemes have been successfully applied to Richards' equation and to a lesser extent to unsaturated poroelasticity. Usual schemes include the classical Newton method, the Picard method, the Picard–Newton method, and the L-scheme with and without Anderson acceleration (see [8, 19, 29]).

The resulting iterative scheme can be written as

$$\frac{\mathrm{d}\boldsymbol{F}}{\mathrm{d}\boldsymbol{x}}(\boldsymbol{x}^m)\,\delta\boldsymbol{x}^{m+1} = -\boldsymbol{F}(\boldsymbol{x})^m, \qquad \boldsymbol{x}^{m+1} \leftarrow \boldsymbol{x}^m + \delta\boldsymbol{x}^{m+1}, \tag{13.44}$$

where $\boldsymbol{F}$ is the residual vector, $\boldsymbol{J} := \mathrm{d}\boldsymbol{F}/\mathrm{d}\boldsymbol{x}$ is the Jacobian matrix depending on the current solution $\boldsymbol{x}^m$, and $\delta\boldsymbol{x}^{m+1}$ is the updated solution. Generally, the manual computation of $\boldsymbol{J}$ is a tedious and error-prone process. To avoid such a process, we exploit the automatic differentiation (AD) interface available in MRST, which in essence consists of breaking down the computation into nested elementary differentiation operations (see [24, 27, 28]). Figure 13.3 shows a schematic representation of the workflow of the iterative solver.

The selection of the timestep Δt plays a key role in a solver's performance. As a general rule, the smaller the timestep the greater the chances of convergence. However, decreasing the timestep too much could be unfeasible for some simulations due to the increase in computational time. A better strategy is to use an adaptive timestepping algorithm, such as the one implemented in Hydrus-1D [47].

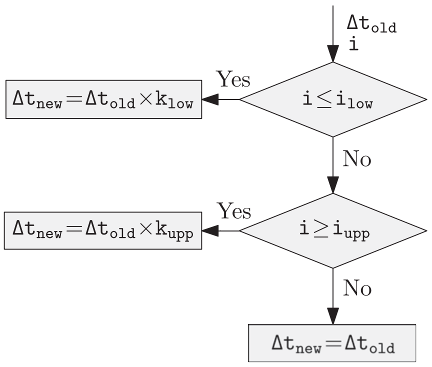

Δt_{old}
i
Yes
$\Delta t_{new} = \Delta t_{old} \times k_{low}$
$i \leq i_{low}$
No
Yes
$\Delta t_{new} = \Delta t_{old} \times k_{upp}$
$i \geq i_{upp}$
No
$\Delta t_{new} = \Delta t_{old}$

Figure 13.4 Workflow of the adaptive time stepping algorithm.

The algorithm determines the next timestep size based on the number of iterations needed to achieve convergence in the last time level (see Figure 13.4). The idea is to increase Δt in case the number of iterations `i` is less (or equal) than a lower optimal iteration range $\mathtt{i}_{\mathtt{low}}$ (i.e., 3), decrease Δt if `i` is greater (or equal) than an upper optimal iteration range $\mathtt{i}_{\mathtt{upp}}$ (i.e., 7), or keep the same value otherwise. To increase Δt, we multiply $\Delta t_{\mathtt{old}}$ by a lower multiplication factor $\mathtt{k}_{\mathtt{low}}$ (i.e., 1.3), and to decrease it, we multiply $\Delta t_{\mathtt{old}}$ by an upper multiplication factor $\mathtt{k}_{\mathtt{upp}}$ (i.e., 0.7).

13.4 Numerical Examples

In this section we present four numerical examples; the first two are numerical convergence tests and the last two are practical applications. The convergence tests include Richards' equation (`convAnalysisRE.m`) and the equations of unsaturated poroelasticity (`convAnalysisUnsatBiot.m`). The third example is a well-known problem for unsaturated flow, where we simulate the water infiltration in a nondeformable initially dry soil (see `waterInfiltrationRE.m`). The last example, `desiccationUnsatBiot.m`, consists of a desiccation process of a clayey soil under atmospheric evaporation in a Petri dish.

Even though the codes for the convergence tests are included in the module, in principle they are not meant as tutorials. To start using `fv-unsat`, we recommend `waterInfiltrationRE.m`, which offers a step-by-step explanation of the module.

13.4.1 Numerical Convergence Tests

The first two examples involve numerical convergence tests, one for Richards' equation and one for the unsaturated poroelastic equations. Before that, we define the errors used to determine the converge rates.

We are interested in measuring the errors for the pressure (or pressure head), displacement, flux, and traction forces. We use the subscript h to denote the numerical approximation and no subscript for the exact solution. We define the following relative discrete L_2-type errors as in [33]:

$$\varepsilon_p^{h,\Delta t} = \frac{\left(\sum_i^{N_c} V_i \, |p_i - p_{h,i}|^2\right)^{1/2}}{\left(\sum_i^{N_c} V_i \, |p_i|^2\right)^{1/2}}, \qquad \varepsilon_Q^{h,\Delta t} = \frac{\left(\sum_j^{N_f} A_j \, |Q_j - Q_{h,j}|^2\right)^{1/2}}{\left(\sum_j^{N_f} A_j \, |Q_j|^2\right)^{1/2}},$$

$$\varepsilon_{\vec{u}}^{h,\Delta t} = \frac{\left(\sum_i^{N_c} V_i \, |\vec{u}_i - \vec{u}_{h,i}|^2\right)^{1/2}}{\left(\sum_i^{N_c} V_i \, |\vec{u}_i|^2\right)^{1/2}}, \qquad \varepsilon_{\vec{T}}^{h,\Delta t} = \frac{\left(\sum_j^{N_f} A_j \, |\vec{T}_j - \vec{T}_{h,j}|^2\right)^{1/2}}{\left(\sum_j^{N_f} A_j \, |\vec{T}_j|^2\right)^{1/2}},$$

where V_i and A_j are the cell volumes and face areas, respectively. For a given variable, we define the reduction between two successive levels of refinement as the ratio between the errors obtained by halving the spatial resolution for a fixed time step. For example, for the pressure, we have the reduction and the convergence rate given by

$$\text{Red}_p = \varepsilon_p^{h,\Delta t} / \varepsilon_p^{h/2,\Delta t}, \qquad \text{Rate}_p = \log_2(\text{Red}_p).$$

Richards' Equation

In this example, we present a numerical convergence analysis of the two-dimensional incompressible mixed-based formulation of Richards' equation. This analysis is performed in a unit square with a final simulation time of 1 and a timestep $\Delta t = 0.1$. The computational mesh is a structured Cartesian grid. The relative permeabilities on the faces are approximated using an arithmetic mean of the cell centers, and for simplicity, gravity effects are neglected. Moreover, all of the physical parameters are assumed to be equal to one, except $\alpha_v = 0.4$, $\theta_w^s = 0.4$, $\theta_w^r = 0.1$, $n_v = 2$, and $m_v = 0.5$. We assume the existence of a time-dependent solution

$$\psi_w(x, y, t) = -t(1 - x)x \sin(\pi x)(1 - y)y \cos(\pi y) - 1,$$

satisfying $\psi_w(0, y, t) = \psi_w(1, y, t) = \psi_w(x, 0, t) = \psi_w(x, 1, t) = \psi_w(x, y, 0) = -1$. With this assumption, it is possible to obtain an exact expression for the source term $\dot{m}_w/\rho_w = f_w$ and compute the errors. We refer to [39] for more details.

Table 13.2 shows the results for five different levels of spatial refinement. Pressure head and fluxes show quadratic convergence rates. These results are consistent with reported rates for MPFA schemes on structured-uniform grids (see, e.g., [2, 3]).

Unsaturated Poroelasticity

In this analysis, we investigate the numerical convergence rates for the unsaturated poroelastic equations. The domain, final simulation time, timestep, average of k_{rw},

Table 13.2 *Convergence test for Richards' equation.*

h	$\varepsilon_{\psi}^{h,\Delta t}$	Red_{ψ}	Rate_{ψ}	$\varepsilon_{Q}^{h,\Delta t}$	Red_{Q}	Rate_{Q}
0.1	5.338×10^{-4}			2.965×10^{-2}		
0.05	1.336×10^{-4}	3.9949	1.9982	7.445×10^{-3}	3.9823	1.9936
0.025	3.343×10^{-5}	3.9970	1.9989	1.852×10^{-3}	4.0207	2.0075
0.0125	8.386×10^{-6}	3.9991	1.9997	4.608×10^{-3}	4.0182	2.0065
0.00625	2.090×10^{-6}	3.9998	1.9999	1.149×10^{-4}	4.0110	2.0040

Table 13.3 *Convergence test for unsaturated poroelasticity: pressure and displacement.*

h	$\varepsilon_{p}^{h,\Delta t}$	Red_{p}	Rate_{p}	$\varepsilon_{\vec{u}}^{h,\Delta t}$	$\text{Red}_{\vec{u}}$	$\text{Rate}_{\vec{u}}$
0.2	8.817×10^{-4}			9.156×10^{-2}		
0.1	2.255×10^{-4}	3.9106	1.9674	2.240×10^{-2}	4.0877	2.0313
0.05	5.725×10^{-5}	3.9381	1.9775	5.597×10^{-3}	4.0017	2.0006
0.025	1.437×10^{-5}	3.9836	1.9941	1.404×10^{-3}	3.9859	1.9949
0.0125	3.596×10^{-6}	3.9970	1.9989	3.524×10^{-4}	3.9852	1.9946
0.00625	8.991×10^{-7}	3.9992	1.9997	8.864×10^{-5}	3.9753	1.9911

and water retention parameters are the same as in the last example. However, we now include gravity contributions. The physical parameters different from unity are $C_s = 0.1$, $n = 0.4$, and $\alpha = 0.9$.

We are interested in convergence rates of pressures and displacements, as well as fluxes and traction forces. We assume the following time-dependent solutions for the primary variables:

$$p_w(x,y,t) = -tx(1-x)y(1-y)\sin(\pi x)\cos(\pi y) - 1,$$
$$\vec{u}(x,y,t) = tx(1-x)y(1-y)\big[\sin(\pi x),\, \cos(\pi y)\big]^T.$$

We employ Dirichlet boundary conditions for the pressure and displacement satisfying the above equations. The initial conditions are obtained by setting $t = 0$, the mesh is a structured triangular grid, and the analysis is performed for six different levels of spatial refinement. The results are shown in Tables 13.3 and 13.4.

Pressures, displacements, and fluxes show quadratic convergence rate. The convergence rate for traction is less uniform. Nevertheless, it is greater than 1.5 and lower than 2, which is in agreement with previously reported rates on structured grids for elasticity and (saturated) poroelasticity [21, 35].

Table 13.4 *Convergence test for unsaturated poroelasticity: flux and traction.*

h	$\varepsilon_Q^{h,\Delta t}$	Red_Q	Rate_Q	$\varepsilon_{\vec{T}}^{h,\Delta t}$	$\text{Red}_{\vec{T}}$	$\text{Rate}_{\vec{T}}$
0.2	1.025×10^{-2}			7.048×10^{-2}		
0.1	2.821×10^{-3}	3.6349	1.8619	2.019×10^{-2}	3.4914	1.8038
0.05	7.258×10^{-4}	3.8863	1.9584	6.152×10^{-3}	3.2813	1.7143
0.025	1.826×10^{-4}	3.9755	1.9911	1.862×10^{-3}	3.3032	1.7239
0.0125	4.568×10^{-5}	3.9968	1.9989	6.256×10^{-4}	2.9773	1.5740
0.00625	1.142×10^{-5}	4.0007	2.0020	2.031×10^{-4}	3.0802	1.6230

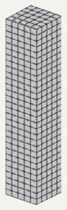
13.4.2 Water Infiltration in a Column of Dry Soil

In this example, we solve a water infiltration problem in an initially dry soil column. The water flows from top to bottom and is modeled using Richards' equation. The simplicity of the problem represents an excellent opportunity to introduce the module (see `waterInfiltrationRE.m` from the `examples` folder).

We start by constructing a Cartesian grid consisting of five cells in the x- and y-directions and 30 cells in the z-direction. The domain is $100 \times 100 \times 100$ cm^3. We refer to chapter 3 of the MRST textbook [28] for more details regarding mesh generation in MRST.

```
nx = 5;     ny = 5;     nz = 30;           % cells
Lx = 1;     Ly = 1;     Lz = 1;            % domain lenght [m]
G = cartGrid([nx, ny, nz], [Lx, Ly, Lz]); % create Cartesian grid
G = computeGeometry(G);                    % compute geometry

% Plotting grid
newplot; plotGrid(G); axis off;
pbaspect([1, 1, 5]); view([-51, 26]);
```



Next, we declare the hydraulic parameters of the soil. We use the physical parameters of a field sample from New Mexico [37]. Most of the properties can be accessed from our mini-catalog of soils (see `getHydraulicProperties.m`). The properties are stored in SI units inside the `phys` structure, which, in turn, contains the `flow` substructure. For coupled problems, the `phys` structure will also contain the `mech` substructure (see next example):

```
soil = getHydraulicProperties('newMexSample');  % get soil properties
phys = struct(); % create structure to store physical properties

% Flow parameters
phys.flow.rho       = 1 * gram / (centi * meter)^3;            % density
phys.flow.mu        = 0.01 * gram / (centi * meter * second);  % viscosity
phys.flow.g         = 980.66 * centi * meter / (second^2);     % gravity
phys.flow.gamma     = phys.flow.rho * phys.flow.g;  % specific gravity
phys.flow.K         = soil.K_s;  % saturated hydraulic conductivity
phys.flow.perm      = (phys.flow.K * phys.flow.mu / phys.flow.gamma) .* ...
                         ones(G.cells.num, 1);  % intrinsic permeability
phys.flow.alpha     = soil.alpha / meter;  % vGM parameter
phys.flow.n         = soil.n;              % vGM parameter
phys.flow.m         = 1-(1/phys.flow.n);   % vGM parameter
phys.flow.theta_s   = soil.theta_s;        % Water content at saturation conditions
phys.flow.theta_r   = soil.theta_r;        % Residual water content
```

Boundary and initial conditions are declared next. For this problem, $\psi_w = -75$ cm is set at the top and $\psi_w, = -1\,000$ cm at the bottom, and the rest are set as no flux by default. Initially, we set $\psi_w = -1\,000$ cm for all cells. Boundary conditions are declared following the MRST convention (see chapter 5 of the MRST textbook [28]). In addition, we need to create `bcVal` (a vector containing the values of the boundary conditions) for backward compatibility with the `fvbiot` module. It is important to mention that if gravity effects are considered, we must include their contributions to the Dirichlet faces in the `bcVal` vector:

```
% Extracting grid information
:

% Creating the boundary structure
psiT          = -75 * centi * meter;     % Top boundary pressure head
psiB          = -1000 * centi * meter;   % Bottom boundary pressure head
bc            = addBC([], z_min, 'pressure', psiT);
bc            = addBC(bc, z_max, 'pressure', psiB);
bcVal         = zeros(G.faces.num, 1);
bcVal(z_min)  = psiT + zetaf(z_min);     % assigning Top boundary
bcVal(z_max)  = psiB + zetaf(z_max);     % assigning Bottom boundary
```

The problem is discretized using the `mpfa` routine from the `fvbiot` module. The `mpfa` function takes as input arguments the `G` structure, the `flow` substructure, and the boundary conditions structure `bc`. The output contains the discrete operators that later will be used to construct the model:

```
%% Discretize the flow problem using MPFA
mpfa_discr = mpfa(G, phys.flow, [], 'bc', bc, 'invertBlocks', 'matlab');
```

After declaring parameters structures for time/printing (`time_param`, `print_param`) we are in position to construct the model. This is done by calling the function `modelRE` (from the `models` folder) as follows:

```
%% Call Richards' equation model
modelEqs = modelRE(G, phys, mpfa_discr, bc, bcVal, 'arithmetic', 'on');
```

Listing 13.1 *The principal parts of the workflow of* `modelRE`*.*

```
function model = modelRE(G, phys, mpfa_discr, bc, bcVal, relPermMethod, gEffects)
:

% Soil Water Retention Curves (SWRC) for the theta-psi model
[theta, krw, C_theta] = vGM_theta(phys);

% Discrete mpfa operators
F      = @(x) mpfa_discr.F * x;          % Flux
boundF = @(x) mpfa_discr.boundFlux * x;  % Boundary fluxes
divF   = @(x) mpfa_discr.div * x;        % Divergence

% Relative permeability at the faces
if strcmp(relPermMethod, 'arithmetic')
    krw_faces = @(psi_m) arithmeticAverageMPFA(G, krw, bc, psi_m);
elseif strcmp(relPermMethod, 'upstream')
    krw_faces = @(psi_m) upstreamWeightingMPFA(G, krw, bc, bcVal, ...
        mpfa_discr, phys, psi_m, 'psi', gEffects);
else
    error('Method not implemented. Use either ''arithmetic'' or ''upstream''')
end

% Darcy Flux
Q = @(psi, psi_m) (phys.flow.gamma ./ phys.flow.mu) .* krw_faces(psi_m) .* ...
    (F(psi + gravOn * zetac) + boundF(bcVal));

% Mass Conservation Equation
psiEq = @(psi, psi_n, psi_m, dt, source)  (V ./ dt) .* (theta(psi_m) ...
    + C_theta(psi_m) .* (psi - psi_m) - theta(psi_n)) ...
    + divF(Q(psi, psi_m)) - V .* source;
:
```

The function `modelRE` takes as input arguments the grid structure `G`, the physical properties structure `phys`, the discretized structure `mpfa_discr`, the boundary conditions structure and vector values `bc` and `bcVal`, and two string arguments. The first string argument specifies the way relative permeabilities at the faces should be calculated (e.g., `'arithmetic'` or `'upstream'`), and the last argument is either `'on'` or `'off'` depending whether gravity effects are included or neglected.

For completeness, we show the principal parts of the workflow of `modelRE` in Listing 13.1. First, we retrieve the SWRC quantities (see (13.6)–(13.8)) using the utility function `vGM_theta`. Because the problem is already discretized, we can create the discrete MPFA operators as introduced in Table 13.1. Next, we compute the relative permeabilities at the faces using the preferred method. Finally, we declare the discrete equations as anonymous functions; i.e., `Q` for the Darcy flux and `psiEq` for the mass conservation equation. The function `modelRE` returns the `model` structure containing the discrete equations together with the

SWRC-related quantities. We remark the straightforward equivalence between the mathematical and computational equations.

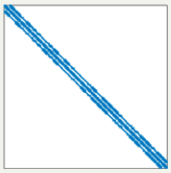
Now, we can solve the nonlinear set of equations using a nested while loop. The first corresponds to the time loop and the second to the solver `solverRE` (shown in a separate code excerpt). Once we exit the solver loop (provided successful convergence), the timestep `dt` for the next time level is calculated using the adaptive timestepping routine `timeStepping`:

```
while time_param.time < time_param.simTime
    psi_n = psi; % current time step (n-index)
    time_param.time = time_param.time + time_param.dt; % current time
    source = zeros(G.cells.num,1); % source term equal to zero
    % Newton loop
    [psi,psi_m,iter] = solverRE(psi_n,modelEqs,time_param,solver_param,source);
    % Determine next time step
    [time_param.dt,print_param.print]=timeStepping(time_param,print_param,iter);
    :
end
```

The solver `solverRE` is written in such a way that it exploits the capabilities of the AD framework:

```
function [psi, psi_m, iter] = solverRE(psi_n, modelEqs, time_param, ...
    solver_param, source)
:
psi_ad = initVariablesADI(psi_n);  % Initialiazing AD-variable

% Newton loop
while (res > solver_param.tol) && (iter <= solver_param.maxIter)
    psi_m = psi_ad.val; % current iteration level (m-index)
    eq = modelEqs.psiEq(psi_ad, psi_n, psi_m, time_param.dt, ...
        source);    % call equation from model
    R = eq.val;     % residual
    J = eq.jac{1};  % Jacobian
    Y = J\-R;       % solve linear system
    psi_ad.val  = psi_ad.val + Y; % update
    res = norm(R);  % compute tolerance
    :
end
psi = psi_ad.val; % return updated pressure head
```



In case the solver does not converge in the prescribed maximum number of iterations, an error is printed in the console. The options to enforce convergence are either to increase `maxIter` or decrease `tol`. The results can be easily accessed via the `sol` object for all printing times. In Figure 13.5, we show the pressure head and water content distributions corresponding to 21.6 hours. Alternatively, the

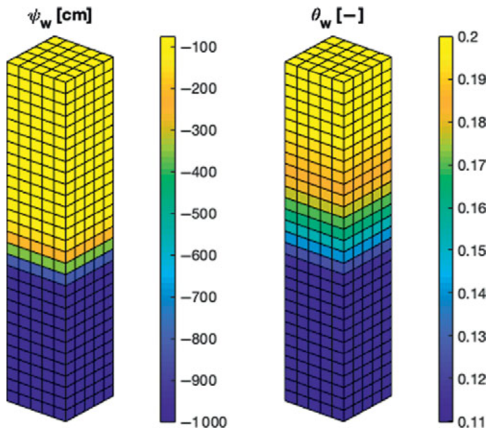


Figure 13.5 Solutions to the water infiltration problem in an initially dry soil. We show the pressure head (left) and water content (right) after 21.6 hours of water infiltration from top to bottom. Note that approximately half of the domain remains in dry conditions.

interested user can explore further plotting functionalities via the `plotToolBar` interface from the `mrst-gui` module.

13.4.3 Desiccation of a Clayey Soil in a Petri Dish

In this numerical experiment, we study the desiccation process of a clayey sample in a Petri dish using real parameters (see `desiccationUnsatBiot.m`). The desiccation is driven by an evaporation process that is modeled using atmospheric boundary conditions, allowing us to resemble with more precision a realistic evaporation scenario. Our main motivation to study soil desiccation is the formation of cracks. Even if fractures are not included in this model, it is useful to predict whether the conditions before cracking exist. The desiccation process involves a gradual reduction of saturation with a simultaneous reduction in the pressure and soil shrinkage [18].

The domain consists of a standard Petri dish (10 cm in diameter and 1.5 cm thick) containing a sample of clay. In such a setup, the soil is constrained everywhere but the top, where the evaporation takes place at stress-free conditions (see Figure 13.6). The evaporation at the top of the Petri dish can be either flux controlled or pressure controlled. In an atmospheric evaporation scenario, the soil initially dries at a maximum evaporation rate (thus a flux-controlled top boundary condition is imposed) and then smoothly decreases, approaching zero in the limit

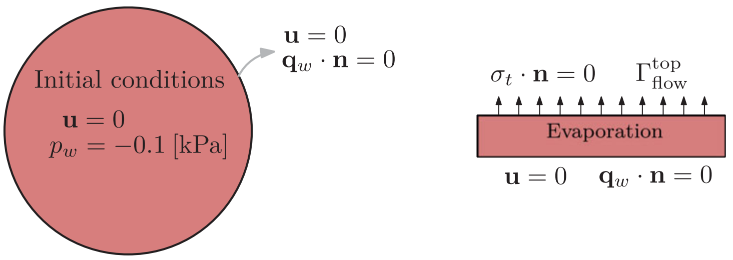

Initial conditions
$\mathbf{u} = 0$
$p_w = -0.1$ [kPa]
$\mathbf{u} = 0$
$\mathbf{q}_w \cdot \mathbf{n} = 0$
$\sigma_t \cdot \mathbf{n} = 0$
$\Gamma^{\text{top}}_{\text{flow}}$
Evaporation
$\mathbf{u} = 0$
$\mathbf{q}_w \cdot \mathbf{n} = 0$

Figure 13.6 The Petri dish domain showing the boundary and initial conditions.

when $S_w \to S_w^r$ (in this second stage a pressure-controlled boundary condition is used). The criteria can be written as

$$\Gamma^{\text{top}}_{\text{flow}} = \begin{cases} E_{\max}, & p_w^{\text{top}} \geq p_w^{\text{crit}}, \\ p_w^{\text{crit}}, & \text{otherwise}, \end{cases}$$

where $\Gamma^{\text{top}}_{\text{flow}}$ is the flow boundary condition at the top of the domain (note that the word "flow" does not refer to a "flux" boundary condition but rather the subproblem as in the flow/mechanics coupled problem), $E_{\max}$ is the maximum evaporation rate, and p_w^{crit} is the water critical pressure [15]. There are several correlations to estimate $E_{\max}$ for field-scale applications [25]. In this case, we adopt an experimental value obtained by Stirling [42] and more recently employed in numerical experiments in [10]. The critical pressure p_w^{crit} is the minimum allowed pressure at the soil surface. This value is a function of the ambient psychrometric conditions and can be estimated as

$$p_w^{\text{crit}} = \frac{\log(\phi) R T \rho_w}{M},$$

where ϕ is the relative humidity, R is the universal gas constant, T is the absolute temperature, ρ_w is the water density, and M is the molecular weight of water [47].

The soil is initially at virtually saturated conditions – i.e., $S_w = 0.9996$ – and the final simulation time is 2 hours. Now, we describe each step of the simulation process. We highly encourage the interested reader to use `desiccationUnsatBiot.m` along with this explanation.

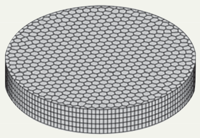
As usual, we start by generating the computational grid. First, we create a Delaunay triangulation on a circle using the (freely available) mesh generator `distemsh` [36]. To add `distmesh` to MRST, we follow the procedure described in [28]:

```
pth = fullfile(ROOTDIR,'utils','3rdparty','distmesh'); mkdir(pth)
unzip('http://persson.berkeley.edu/distmesh/distmesh.zip', pth);
mrstPath('reregister','distmesh', pth);
```

Listing 13.2 *Grid construction for the Petri dish.*

```
% Two-dimensional grid
r      = 50 * milli * meter;           % radii of the Petri-dish
fd     = @(p) sqrt(sum(p.^2, 2)) - r; % circular domain function
min_x  = -r;  max_x = r;               % min and max values in x-axis
min_y  = -r;  max_y = r;               % min and max values in y-axis
h      = (2*r)/25;                     % step size
[p, t] = distmesh2d(fd, @huniform, h, [min_x, min_y; max_x, max_y], []);
p      = p + r;                        % shifting triangulation points
G      = triangleGrid(p, t);           % creating triangular grid
G      = pebi(G);                      % creaing Voronoi diagram

% Extrude in the z-direction
Lz = 15 * milli * meter;  % thickness of the Petri-dish
nz = 5;                   % number of layers in z-axis
dz = Lz/nz;               % thickness of each layer
thick = dz .* ones(nz, 1);     % thickness vector
G = makeLayeredGrid(G, thick); % extrude grid
G = computeGeometry(G);        % compute geometry
```



We employ the function `distmesh2d` to triangulate a circle of radius `r`, with step size `h`. With the triangulation points `p` and the connectivity map `t` available, we can generate the triangular grid using `triangleGrid` and then apply a Voronoi diagram using the `pebi` routine to obtain the hexagonal grid. Finally, to generate the three-dimensional grid, we extrude the hexagonal grid in the z-direction using the function `makeLayeredGrid`[2] (see Listing 13.2).

After extracting useful topological data, we declare the physical parameters for the mechanics and the flow problem using the `phys` structure:

```
% Mechanics parameters [Kaolinite]
phys.mech.lambda = 1.229E11 .* ones(Nc, 1) * Pascal;  % first Lame parameter
phys.mech.mu     = 4.7794E10 .* ones(Nc, 1) * Pascal; % second Lame parameter
phys.mech.C_s    = 5.618E-11 / Pascal;                % solid compressibility
phys.mech.rho    = 1769 * kilo * gram / meter^3;      % solid density
phys.mech.stiff  = shear_normal_stress(Nc, Nd, ...    % stiffnes matrix
                    phys.mech.mu, phys.mech.lambda, 0 .* phys.mech.mu);
```

Here, we assume homogeneity in the physical properties. However, the code is flexible to include heterogeneous permeability and elasticity coefficients. The elastic parameters were taken from [32] for a sample of kaolinite and the hydraulic properties from [11] for clay. The mechanic discretization requires the construction of the stiffness matrix. This is done using the function `shear_normal_stress` from the `fvbiot` module.

[2] Technically speaking, these grids are referred to as 2.5–dimensional grids.

We use the soil catalog to get the hydraulic properties of the clay. The critical pressure is determined using `computeCriticalPressure`. For this example, we assume standard laboratory psychometric conditions; i.e., $T = 298.15$ K and $\phi = 0.5$:

```
% Flow parameters [Water]
soil = getHydraulicProperties('clay');
:
phys.flow.temperature = 298.15 * Kelvin;  % Ambient temperature
phys.flow.relativeHumidity = 0.5;         % Ambient relative humidity
p_crit = computeCriticalPressure(phys);
```

Now, we proceed to declare the boundary conditions. For the mechanics, we set $\vec{u} = 0$ at the sides and bottom of the domain, whereas the top is assumed to be stress-free by default (note that the keyword `'pressure'` indicates a displacement condition and `'flux'` indicates a traction condition):

```
% Creating the boundary structure for the mechanics problem
bcMech = addBC([], sides, 'pressure', 0);       % u=0 at the sides
bcMech = addBC(bcMech, z_max, 'pressure', 0);   % u=0 at the bottom
bcMechVals = zeros(Nd * Nf, 1);
```

For the flow boundary conditions, we have two scenarios: flux and pressure controlled. For the flux-controlled scenario we have only flux conditions:

```
% Creating the boundary structure for flux-controlled BC
bcFlow_f     = addBC([], z_min, 'flux', Qtop_f);
bcFlowVals_f = zeros(Nf, 1);
bcFlowVals_f(z_min) = Qtop_f;
```

whereas for the pressure-controlled, we have zero flux except at the top:

```
% Creating the boundary structure for pressure-controlled BC
bcFlow_p     = addBC([], z_min, 'pressure', p_crit);
bcFlowVals_p = zeros(Nf, 1);
bcFlowVals_p(z_min) = p_crit + phys.flow.gamma .* zetaf(z_min);
```

For the initial conditions, we assume an initially undeformed sample – that is, $\vec{u}(x,y,z,0)$=0 m – and a homogeneous pressure field of $p_w(x,y,z,0) = -0.1$ kPa:

```
u_init = zeros(Nd * Nc, 1) * meter;
p_init = -0.1 * kilo * Pascal * ones(Nc, 1);
```

Once the boundary and initial conditions have been declared, we can discretize the different problems. On one hand, we have the mechanical problem, which is discretized using the `mpsa` routine from `fvbiot`, and, on the other hand, we have the flow problem which is discretized using `mpfa`. Note that the flow problem is divided into the flux- and pressure-controlled subproblems, because different boundary conditions result in different discrete operators:

```
% Discretize mechanics problem
mpsa_discr = mpsa(G,phys.mech.stiff,[],'invertBlocks','matlab','bc',bcMech);

% Discretize flow problem for flux-controlled boundary conditions
mpfa_discr_flux = mpfa(G,phys.flow, [],'invertBlocks','matlab','bc',bcFlow_f);

% Discretize flow problem for pressure-controlled boundary conditions
mpfa_discr_pres = mpfa(G,phys.flow, [],'invertBlocks','matlab','bc',bcFlow_p);
```

After declaring the time and printing parameters, we set up the two different scenarios (flux and pressure controlled) using the function `modelUnsatBiot`:

```
%% Calling the model for the unsaturated poroelastic equations

% Setting up model for flux-controlled problem
modelEqsFlux = modelUnsatBiot(G, phys, mpfa_discr_flux, mpsa_discr, ...
    bcFlow_f, bcFlowVals_f, bcMech, bcMechVals, 'upstream', 'on');

% Setting up model for pressure-controlled problem
modelEqsPres = modelUnsatBiot(G, phys, mpfa_discr_pres, mpsa_discr, ...
    bcFlow_p, bcFlowVals_p, bcMech, bcMechVals, 'upstream', 'on');
```

Note that `modelUnsatBiot` now uses both the mechanics and flow boundary conditions as well as discretization structures. The last two string arguments are the same as in `modelRE`. To avoid being repetitive, and because `modelUnsat Biot` is essentially the same as `modelRE` (structure-wise, not complexity-wise), we prefer not to show this function and proceed with solving the coupled systems.

To solve the coupled problem we create two time loops, one for each flow scenario. The flux-controlled time loop is shown in Listing 13.3. The process is essentially the same as in `waterInfiltrationRE.m`, except for some technicalities. Note that after calling `solverUnsatBiot` we calculate the value of the top pressure of the domain using the function `computeTopPressure`. This function uses a TPFA discretization to approximate the mean value of the surface pressure. Next, we check whether the critical pressure is reached or not. If the pressure is higher, we proceed to determine the next timestep using `timeStepping`.

Listing 13.3 *Flux-controlled time loop.*

```
while (time_param.time < time_param.simTime) && (p_top > p_crit) ...
        && (pControlled == false)

    p_n = p; % current time level (n-index)
    u_n = u; % current time level (n-index)
    time_param.time = time_param.time + time_param.dt; % cumulative time

    % Source terms
    sourceFlow = zeros(Nc, 1); % no sources for the flow
    sourceMech = modelEqsFlux.body(p_n); % sourceMech = body force

    % Calling Newton solver
    [p, p_m, u, iter] = solverUnsatBiot(G, p_n, u_n, modelEqsFlux, ...
        time_param, solver_param, sourceFlow, sourceMech);

    % Approximating top pressure
    fluxTemp = modelEqsFlux.Q(p, p_m);
    p_top = computeTopPressure(G, phys, p, fluxTemp, modelEqsFlux);

    % If it is flux controlled, update time step and store solution
    if (p_top > p_crit)
        % Calling time stepping routine
        [time_param.dt, print_param.print] = timeStepping(time_param, ...
            print_param, iter);

        : % store solution if necessary
    else
        : % change to pressure controlled loop
    end
end
```

If the pressure is less than (or equal to) the critical pressure, we switch to the pressure-controlled time loop.

Because the pressure-controlled loop is essentially the same, we show the solver and the sparsity of the system in Listing 13.4. The Jacobian matrix consists of four blocks, which are characteristic of the monolithic approach:

– Upper-left: displacement contribution to the momentum equation, `eq1`.
– Upper-right: pressure contribution to the momentum equation, `eq2`.
– Lower-left: displacement contribution to the storage equation, `eq3`.
– Lower-right: pressure contribution to the storage equation, `eq4`.

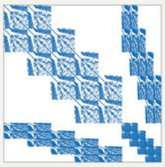
The simulation results are shown in Figures 13.7–13.10. In Figure 13.7, we show the saturation profile for the final simulation time. As expected, the lower saturation zones are located at the top layer due to the evaporation process, whereas the bottom layer remains at nearly saturated conditions. In Figure 13.8, we show the variation

Listing 13.4 *Solver for the unsaturated Biot equations.*

```
function [p, p_m, u, iter] = solverUnsatBiot(G, p_n, u_n, modelEqs, ...
    time_param, solver_param, sourceFlow, sourceMech)
:
% Initializing AD-variables
p_ad = initVariablesADI(p_n);
u_ad = initVariablesADI(u_n);

% Newton loop
while (res > solver_param.tol) && (iter <= solver_param.maxIter)
    % Calling equations
    p_m = p_ad.val; % current iteration level (m-index)
    eq1 = modelEqs.uEq1(u_ad);
    eq2 = modelEqs.uEq2(p_ad, p_n, sourceMech);
    eq3 = modelEqs.pEq1(p_n, u_ad, u_n);
    eq4 = modelEqs.pEq2(p_ad, p_n, p_m, time_param.dt, sourceFlow);

    J = [eq1.jac{1} eq2.jac{1}; eq3.jac{1}, eq4.jac{1}];
    R = [eq1.val + eq2.val; eq3.val + eq4.val];
    Y = J\-R;  % solve linear system
    u_ad.val = u_ad.val + Y(1:Nd*Nc);      % update u
    p_ad.val = p_ad.val + Y(Nd*Nc+1:end); % update p
    res = norm(R);  % compute tolerance
    :
end
p = p_ad.val; % updating pressure value
u = u_ad.val; % updating displacement value
```



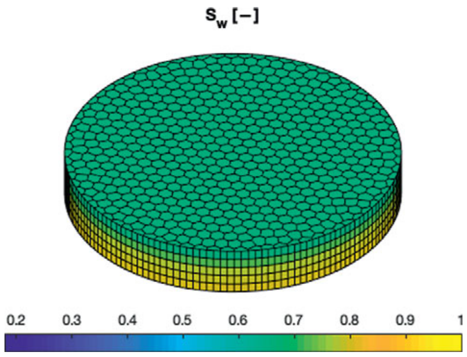


Figure 13.7 Saturation field for the final simulation time.

of the top pressure head and flux with respect to time. The change in boundary condition modes that takes place at 0.44 hours highly influences the evaporation process. After this point, the pressure declines abruptly toward the critical value, whereas the flux smoothly approaches zero as the driven force for the evaporation vanishes.

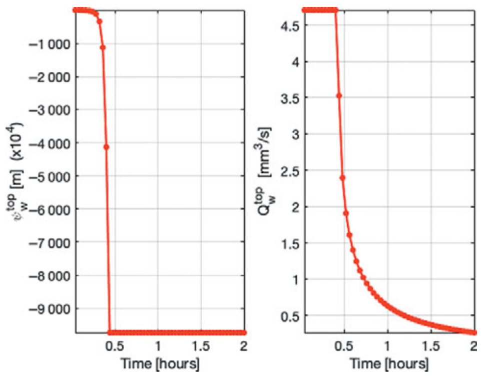


Figure 13.8 Top pressure head (left) and surface flux (right) evolution.

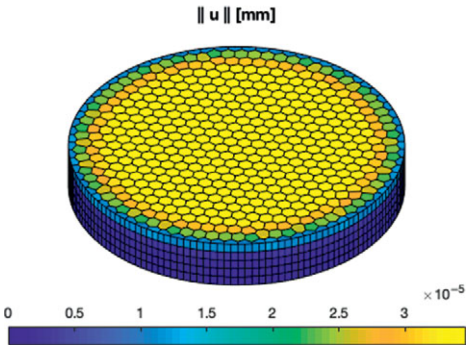


Figure 13.9 Magnitude of the displacement for the final simulation time. The deformation is maximum where the evaporation takes place.

In Figure 13.9, we show the magnitude of the displacement field for the final simulation time. Note that the displacement is maximum at the top layer, which again is in agreement with the expected results. Finally, in Figure 13.10 we show a closeup of the positive quarter domain of the top layer, where the arrows depict the direction of the displacement field, demonstrating the tensile nature of the stresses that eventually cause the rupture of the material.

13.5 Concluding Remarks

In this chapter, we presented a flexible solver based on robust multipoint finite-volume schemes (MPFA/MPSA) for simulating flow in unsaturated soils. We studied the case where deformations effects are neglected (Richards' equation) and the case where small deformations and linear elastic behavior of the soil are assumed

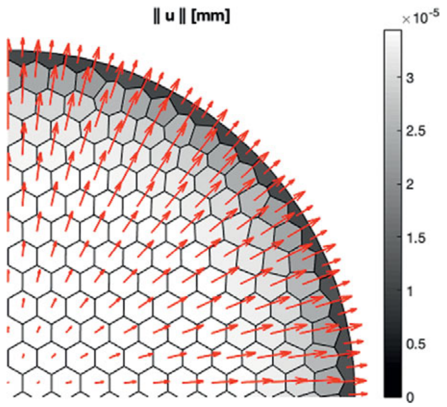


Figure 13.10 Positive quarter domain (top layer). The arrows show the characteristic tensile nature of stresses of clayey soils when subjected to desiccation.

(equations of unsaturated poroelasticity). Numerical tests showed that convergence rates previously found for saturated media are preserved when the models are extended to the (nonlinear) unsaturated case. In addition, we provided two numerical applications, a classical water infiltration case using Richards' equation and a fairly realistic desiccation process of a clayey soil driven by atmospheric evaporation. In both cases, physically coherent results are obtained. Thanks to the AD-based approach, the models presented herein can be extended to include other processes such as scalar transport, chemical reactions, or heat transfer.

Acknowledgement. The funds that made this investigation possible were provided by the Paraguayan Postgraduate Scholarship Program "Don Carlos Antonio López" (Ref 209/2016) and the Norwegian Academy of Science and Letters (VISTA project 6371).

References

[1] I. Aavatsmark. An introduction to multipoint flux approximations for quadrilateral grids. *Computational Geosciences*, 6(3-4):405–432, 2002. doi: 10.1023/A:1021291114475.

[2] I. Aavatsmark, G. T. Eigestad, and R. A. Klausen. Numerical convergence of the MPFA O-method for general quadrilateral grids in two and three dimensions. In *Compatible Spatial Discretizations*, pp. 1–21. Springer, 2006. doi: 10.1007/0-387-38034-5_1.

[3] I. Aavatsmark, G. T. Eigestad, R. A. Klausen, M. F. Wheeler, and I. Yotov. Convergence of a symmetric MPFA method on quadrilateral grids. *Computational Geosciences*, 11(4):333–345, 2007. doi: 10.1007/s10596-007-9056-8.

[4] I. Aavatsmark, G. Eigestad, B. Mallison, and J. Nordbotten. A compact multipoint flux approximation method with improved robustness. *Numerical Methods for Partial Differential Equations: An International Journal*, 24(5):1329–1360, 2008. doi: 10.1002/num.20320.

[5] J. Bear. *Dynamics of Fluids in Porous Media*. Dover Publications, 1989.

[6] J. W. Both, M. Borregales, J. M. Nordbotten, K. Kumar, and F. A. Radu. Robust fixed stress splitting for Biot's equations in heterogeneous media. *Applied Mathematics Letters*, 68:101–108, 2017. doi: 10.1016/j.aml.2016.12.019.

[7] J. W. Both, K. Kumar, J. M. Nordbotten, and F. A. Radu. Iterative methods for coupled flow and geomechanics in unsaturated porous media. In M. Vandamme, P. Dangla, J.-M. Pereira, and S. Ghabezloo, eds., *Poromechanics VI*, pp. 411–418. American Society of Civil Engineers, Reston, VA, 2017. doi: 10.1061/9780784480779.050.

[8] J. W. Both, K. Kumar, J. M. Nordbotten, and F. A. Radu. Anderson accelerated fixed-stress splitting schemes for consolidation of unsaturated porous media. *Computers & Mathematics with Applications*, 77(6):1479–1502, 2019. doi: 10.1016/j.camwa.2018.07.033.

[9] J. W. Both, I. S. Pop, and I. Yotov. Global existence of a weak solution to unsaturated poroelasticity. *arXiv preprint arXiv:1909.06679*, 2019.

[10] T. Cajuhi, L. Sanavia, and L. De Lorenzis. Phase-field modeling of fracture in variably saturated porous media. *Computational Mechanics*, 61(3):299–318, 2018. doi: 10.1007/s00466-017-1459-3.

[11] R. F. Carsel and R. S. Parrish. Developing joint probability distributions of soil water retention characteristics. *Water Resources Research*, 24(5):755–769, 1988. doi: 10.1029/wr024i005p00755.

[12] M. A. Celia, E. T. Bouloutas, and R. L. Zarba. A general mass-conservative numerical solution for the unsaturated flow equation. *Water Resources Research*, 26(7):1483–1496, 1990. doi: 10.1029/WR026i007p01483.

[13] O. Coussy. *Poromechanics*. John Wiley & Sons, 2004.

[14] J. Droniou. Finite volume schemes for diffusion equations: introduction to and review of modern methods. *Mathematical Models and Methods in Applied Sciences*, 24(8):1575–1619, 2014. doi: 10.1142/s0218202514400041.

[15] R. A. Feddes, E. Bresler, and S. P. Neuman. Field test of a modified numerical model for water uptake by root systems. *Water Resources Research*, 10(6):1199–1206, 1974. doi: 10.1029/wr010i006p01199.

[16] D. G. Fredlund. Unsaturated soil mechanics in engineering practice. *Journal of Geotechnical and Geoenvironmental Engineering*, 132(3):286–321, 2006. doi: 10.1061/(ASCE)1090-0241(2006)132:3(286).

[17] R. A. Freeze and J. A. Cherry. *Groundwater*. Prentice-Hall, 1979.

[18] L. Goehring, A. Nakahara, T. Dutta, S. Kitsunezaki, and S. Tarafdar. *Desiccation Cracks and Their Patterns: Formation and Modelling in Science and Nature*. John Wiley & Sons, 2015.

[19] D. Illiano, I. S. Pop, and F. A. Radu. Iterative schemes for surfactant transport in porous media. *Computational Geosciences*, 25:805–822, 2021. doi: 10.1007/s10596-020-09949-2.

[20] E. Keilegavlen, R. Berge, A. Fumagalli, M. Starnoni, I. Stefansson, J. Varela, and I. Berre. PorePy: an open-source software for simulation of multiphysics processes in fractured porous media. *Computational Geosciences*, 25:243–265, 2021. doi: 10.1007/s10596-020-10002-5.

[21] E. Keilegavlen and J. M. Nordbotten. Finite volume methods for elasticity with weak symmetry. *International Journal for Numerical Methods in Engineering*, 112(8):939–962, 2017. doi: 10.1002/nme.5538.

[22] J. Kim, H. Tchelepi, and R. Juanes. Stability and convergence of sequential methods for coupled flow and geomechanics: drained and undrained splits. *Computer Methods*

in Applied Mechanics and Engineering, 200(23–24):2094–2116, 2011. doi: 10.1016/j.cma.2011.02.011.

[23] R. Klausen, F. Radu, and G. Eigestad. Convergence of MPFA on triangulations and for Richards' equation. *International Journal for Numerical Methods in Fluids*, 58(12):1327–1351, 2008. doi: 10.1002/fld.1787.

[24] S. Krogstad, K.-A. Lie, O. Møyner, H. M. Nilsen, X. Raynaud, and B. Skaflestad. MRST-AD – an open-source framework for rapid prototyping and evaluation of reservoir simulation problems. In *SPE Reservoir Simulation Symposium, 23–25 February, Houston, Texas*. Society of Petroleum Engineers, 2015. doi: 10.2118/173317-MS.

[25] K. K. Kumar, K. R. Kumar, and P. Rakhecha. Comparison of Penman and Thornthwaite methods of estimating potential evapotranspiration for Indian conditions. *Theoretical and Applied Climatology*, 38(3):140–146, 1987. doi: 10.1007/bf00868097.

[26] R. B. Lewis and B. A. Schrefler. *The Finite Element Method in the Static and Dynamic Deformation and Consolidation of Porous Media*. Wiley, 1998.

[27] X. Li and D. Zhang. A backward automatic differentiation framework for reservoir simulation. *Computational Geosciences*, 18(6):1009–1022, 2014. doi: 10.1007/s10596-014-9441-z.

[28] K.-A. Lie. *An Introduction to Reservoir Simulation Using MATLAB/GNU Octave: User Guide for the MATLAB Reservoir Simulation Toolbox (MRST)*. Cambridge University Press, Cambridge, UK, 2019. doi: 10.1017/9781108591416.

[29] F. List and F. A. Radu. A study on iterative methods for solving Richards' equation. *Computational Geosciences*, 20(2):341–353, 2016. doi: 10.1007/s10596-016-9566-3.

[30] J. Lubliner and P. Papadopoulos. *Introduction to Solid Mechanics*. Springer, 2016.

[31] A. Merxhani. An introduction to linear poroelasticity. *arXiv preprint arXiv:1607.04274*, 2016.

[32] N. H. Mondol, J. Jahren, K. Bjørlykke, and I. Brevik. Elastic properties of clay minerals. *The Leading Edge*, 27(6):758–770, 2008. doi: 10.1190/1.2944161.

[33] J. M. Nordbotten. Cell-centered finite volume discretizations for deformable porous media. *International Journal for Numerical Methods in Engineering*, 100(6):399–418, 2014. doi: 10.1002/nme.4734.

[34] J. M. Nordbotten. Finite volume hydromechanical simulation in porous media. *Water Resources Research*, 50(5):4379–4394, 2014. doi: 10.1002/2013wr015179.

[35] J. M. Nordbotten. Stable cell-centered finite volume discretization for Biot equations. *SIAM Journal on Numerical Analysis*, 54(2):942–968, 2016. doi: 10.1137/15m1014280.

[36] P.-O. Persson and G. Strang. A simple mesh generator in MATLAB. *SIAM Review*, 46(2):329–345, 2004. doi: 10.1137/s0036144503429121.

[37] G. F. Pinder and M. A. Celia. *Subsurface Hydrology*. John Wiley & Sons, 2006.

[38] G. F. Pinder and W. G. Gray. *Essentials of Multiphase Flow and Transport in Porous Media*. John Wiley & Sons, 2008.

[39] F. A. Radu and W. Wang. Convergence analysis for a mixed finite element scheme for flow in strictly unsaturated porous media. *Nonlinear Analysis: Real World Applications*, 15:266–275, 2014. doi: 10.1016/j.nonrwa.2011.05.003.

[40] L. A. Richards. Capillary conduction of liquids through porous mediums. *Physics*, 1(5):318–333, 1931. doi: 10.1063/1.1745010.

[41] B. Schrefler, T. Shiomi, A. Chan, O. Zienkiewicz, and M. Pastor. *Computational Geomechanics with Special Reference to Earthquake Engineering*. Wiley, Chichester, England, 1999.

[42] R. A. Stirling. Multiphase modelling of desiccation cracking in compacted soil. PhD thesis, Newcastle University, 2014. URL hdl.handle.net/10443/2492.

[43] M. T. van Genuchten. A closed-form equation for predicting the hydraulic conductivity of unsaturated soils. *Soil Science Society of America Journal*, 44(5):892–898, 1980. doi: 10.2136/sssaj1980.03615995004400050002x.

[44] J. Varela. Implementation of an MPFA/MPSA-FV solver for the unsaturated flow in deformable porous media. Master's thesis, The University of Bergen, 2018. URL hdl.handle.net/1956/17905.

[45] A. Verruijt. *An Introduction to Soil Mechanics*, Volume 30 of *Theory and Applications of Transport in Porous Media.* Springer, 2017. doi:10.1007/978-3-319-61185-3.

[46] J. Šimůnek and S. A. Bradford. Vadose zone modeling: introduction and importance. *Vadose Zone Journal*, 7(2):581–586, 2008. doi: 10.2136/vzj2008.0012.

[47] J. Šimůnek, M. Šejna, M. T. Van Genuchten, D. Mallants, H. Saito, and M. Sakai. The HYDRUS-1D software package for simulating the one-dimensional movement of water, heat, and multiple solutes in variable-saturated media, 2013. URL www.pc-progress.com/Downloads/Pgm_hydrus1D/HYDRUS1D-4.17.pdf.