14

A Brief Introduction to Poroelasticity and Simulation of Coupled Geomechanics and Flow in MRST

ODD ANDERSEN

Abstract

In this chapter, we discuss how two-way coupled fluid flow and geomechanics can be modeled in the MATLAB Reservoir Simulation Toolbox (MRST) using the `ad-mechanics` module. A brief introduction to linear poroelasticity is provided, which is a common framework for studying geomechanics in the context of reservoir management or groundwater applications. We review commonly used poroelastic coefficients and moduli and present a handy tool that removes the need to manually navigate the large number of poroelastic relationships to compute values of needed parameters. The chapter further provides three examples where well-known model cases in linear elasticity and poroelasticity are modeled in MRST and compared with results from analytical estimates. These examples include the compression of a dry sample (a linear elastic problem) as well as the compression of a wet sample (Terzaghi's problem) and Mandel's problem.

14.1 Introduction

Understanding the mechanical behavior of the subsurface is a key factor for a wide and growing range of engineering disciplines. Historical examples include building and construction, which requires a good understanding of the process of soil consolidation, and the extraction of groundwater from confined aquifers, where estimates of available resources are based on storage coefficients derived from poromechanics theory. The oil and gas industry is also concerned with the estimation of in-place resources, and growing use of hydraulic fracking for well stimulation has further increased its emphasis on understanding the underlying rock mechanics.

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

Another growing field where knowledge of the subsurface stress state is of fundamental importance is the production of energy from deep geothermal systems [8] (see also Chapter 12). Fluid flow through such systems is often through fractures whose properties, whether engineered or natural, to a large degree depend on the in situ effective stress. Although rocks are often fractured on purpose in the context of geothermal energy or hydrocarbon production, fracturing is something that must actively be avoided in other contexts. In the emerging engineering discipline of geological CO_2 storage, an important parameter is that of *maximum sustainable injection pressure* [17], which is the maximum pressure at which CO_2 can be injected into an underground reservoir without risking irreversible mechanical damage in the form of new fractures or reactivating existing faults. Other common causes of concern that arise when fluids are extracted from or injected into the subsurface through human activity include the potential for long-term land subsidence (groundwater, hydrocarbons) and the possibility of induced seismicity.

To a large extent, the underground can be thought of as a fluid-filled porous medium consisting of rock (solid) and water (liquid). Its mechanical behavior is frequently modeled and analyzed within the framework of *linear poroelasticity*. This theory models the linear elastic behavior of porous, fluid-filled systems and how the mechanics of the solid matrix and the evolving pressure of the pore fluid influence each other. As reflected by historical practice (reservoir modeling, hydrogeology), this influence can in many cases reasonably be considered to work mainly in one direction, which allows the engineer to compute one system (i.e., fluid flow) first and then estimate the impact on the other (e.g., the mechanical stresses) as a second step, only if needed. However, in modern applications it is frequently becoming the case that the two-way coupling of fluid flow and mechanical behavior cannot be neglected or be sufficiently approximated by simple multipliers.

The goal of this chapter is to show how the coupled effects of geomechanical stresses, deformations, and fluid flow can be modeled and understood within the framework of linear poroelasticity, using tools provided by the MATLAB Reservoir Simulation Toolbox (MRST). We present a brief introduction to the theory, building on the concepts originally introduced by Maurice Biot [4]. We focus on the equations, general poroelastic concepts and quantities, solution strategies, and the resolution of some well-known problems from the literature through developed code examples. On the other hand, analysis of the calculated stresses, strains, and pressures in the context of the actual engineering disciplines just mentioned is not within this chapter's scope. It should also be mentioned that linear (and nonlinear) poroelasticity is highly relevant not only for applications within geomechanics but also for a whole range of other sciences and industries (e.g., biomedical applications, manufacturing of composites, gels [5, 12]) in which the understanding of porous media flow and deformation is central.

Notation: We use lowercase symbols with arrows to denote 3D vectors ($\vec{n}$) but use plain italic letters for coordinates (x). Higher-rank tensors are usually represented with boldface and uppercase ($\mathbf{C}$), except for the stress and strain tensors, which are written using their traditional symbols $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$. On the other hand, the scalar properties *mean stress*, σ, and *volumetric strain*, ϵ, are written in regular face. For the gradient (of a scalar or vector) we use the ∇ symbol, whereas the divergence (of a vector or tensor) is written using $(\nabla\cdot)$. Application of a tensor to another tensor or vector is written with a dot between, $\boldsymbol{\sigma}\cdot\vec{n}$, whereas time derivatives are written with a dot above, $\dot{\zeta}$. We have tried to remain consistent with the established use of letters and symbols in other poroelastic literature. As such, we advise the reader to carefully distinguish between $\mathbf{K}$ (boldface), which represents the permeability tensor, and K (regular face), which represents the (drained) bulk modulus of a (poro)elastic medium.

14.2 Governing Equations

In this section, we present the equations governing mechanic deformation in a linear elastic system and then extend the formulation to the linear poroelastic case by coupling the linear elastic equations with the equation for one-phase fluid flow in a porous medium.

14.2.1 Equations of Linear Elasticity

We here consider the *elastostatic* problem of linear elasticity, for which the solid under consideration is assumed to be in static equilibrium and the problem does not depend on time. The purely elliptic governing equations are derived from the force equilibrium equations (Newton's second law with zero acceleration) applied to each point of a continuum, combined with a linear constitutive relationship relating material stresses and strains. We will here briefly derive these equations expressed in terms of the (infinitesimal) displacement field of the domain. This version of the equations is commonly called the *displacement formulation*.

Displacement and the Strain Tensor

Let Ω represent a region in 3D space occupied by some continuous solid (or part thereof). Consider a spatial transformation $\vec{\chi}$ of this solid that moves each point from its original position $x \in \Omega$ to a new position $\vec{\chi}(x) \in \vec{\chi}(\Omega)$. The associated displacement field $\vec{u}$ is defined as $\vec{u} = \vec{\chi}(x) - x$. Furthermore, assume that the deformation is such that the displacements are *infinitesimal*; i.e., that the values of $\vec{u}(x)$ are sufficiently small to be (i) negligible compared to the size of Ω and

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

(ii) small enough to ignore second-order effects of the resulting strains on material stresses (see Subsection 14.2.1). Infinitesimal displacements also mean that the change in position can be disregarded when referencing points in Ω after the deformation; in other words, we can use the original coordinates x to refer to the position of the points before and after the displacement. (This means that we make no distinction between the so-called *Lagrangian* [material] and *Eulerian* [spatial] descriptions of the solid, a distinction that becomes important when displacements are finite.) Infinitesimal displacements thus do not affect the geometric description; they only matter in the effect they have on strains and stresses [15].

We now define the infinitesimal *deformation gradient tensor*:

$$\mathbf{F} = \nabla \vec{u}. \tag{14.1}$$

As the gradient of a vector in 3D space, this is a rank 2 tensor. An infinitesimal, continuous spatial transformation of an infinitesimal volume element dV_x around a position $x \in \Omega$ can be decomposed into a *translation*, a *rotation*, and a *deformation*, described respectively by $\vec{u}$ itself and the antisymmetric and symmetric parts of the deformation gradient tensor $\mathbf{F}(x)$. The translation and rotation components ($\vec{u}$ and the antisymmetric part of $\mathbf{F}$) collectively describe a *rigid-body motion*. The deformation component (the symmetric part of $\mathbf{F}$) can be further broken down into three orthogonal, infinitesimal *strains*. The symmetric part of $\mathbf{F}$ is called the *infinitesimal strain tensor* $\boldsymbol{\epsilon}$:

$$\boldsymbol{\epsilon} = \tfrac{1}{2}(\mathbf{F} + \mathbf{F}^T) = \tfrac{1}{2}\big(\nabla \vec{u} + (\nabla \vec{u})^T\big). \tag{14.2}$$

The three (real) eigenvalues of this symmetric tensor are called *principal strains* and the associated eigenvectors the *principal directions of strain*. Strain is a dimensionless quantity. A positive value of a principal strain represents an extension ("stretch") along the associated principal direction, and a negative value represents a compression.

The Stress Tensor and Force Equilibrium

With each point $x \in \Omega$, we can associate a rank 2 tensor, called the *stress tensor*, which expresses the internal forces (stress) acting on an infinitesimal surface $dS_{x,\vec{n}}$ around x for any given surface normal $\vec{n}$. In other words, for each spatial direction $\vec{n}$, the application of the stress tensor provides the corresponding stress vector $\vec{\tau}$. The stress tensor is usually denoted $\boldsymbol{\sigma}$, and the stress vector for a given point x and direction $\vec{n}$ can be written as

$$\vec{\tau}(x, \vec{n}) = \boldsymbol{\sigma}(x) \cdot \vec{n}, \tag{14.3}$$

where we emphasize that $\boldsymbol{\sigma}$ is a function of $x \in \Omega$ so that $\vec{\tau}$ depends both on x and the chosen direction $\vec{n}$. In general, $\vec{n}$ and $\vec{\tau}$ are not colinear. The component

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

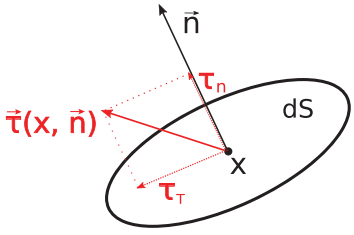

Figure 14.1 Decomposition of the stress $\vec{\tau}(x,\vec{n})$ acting on an infinitesimal surface element dS around a point x having unit normal $\vec{n}$. The stress vector $\vec{\tau}(x,\vec{n})$ can be decomposed into a normal component $\tau_n(x,\vec{n})$ (normal stress) and a tangential component $\tau_t(x,\vec{n})$ (shear stress).

of $\vec{\tau}$ parallel with $\vec{n}$ is called *normal stress* and the perpendicular component *shear stress*; see Figure 14.1.

Stress has the unit of pressure (force per area). For a solid in rotational equilibrium, Newton's second law for moments can be used to show that the tensor is necessarily symmetric. As such, it has three real eigenvalues, which we refer to as *principal stresses*, with associated eigenvectors that are called *principal directions of stress*. We here use the convention that positive eigenvalues represent *extensive* stresses (same direction as $\vec{n}$, thus "pulling" on the corresponding surface element), whereas negative eigenvalues represent *compressive* stresses (with opposite orientation of $\vec{n}$, thus "pushing" on the corresponding surface element). The reader should be aware that the opposite sign convention is also sometimes found in the literature.

Similarly, Newton's second law of motion can be used to show that in the static case (no acceleration), the following relation must hold in the interior of Ω:

$$\nabla \cdot \boldsymbol{\sigma} + \vec{b} = 0. \tag{14.4}$$

Here, $\nabla \cdot \boldsymbol{\sigma}$ denotes the divergence of the stress tensor, which here becomes a vector, whereas $\vec{b}$ represents the body forces, which usually consist of gravity in geomechanical applications. In other words, $\vec{b} = \rho\vec{g}$ where ρ is the mass density and $\vec{g}$ is the vector of gravitational acceleration.

Linking Stresses and Strains

In linear elasticity, the constitutive relation linking the stress and strain tensor is in its most general form represented by a rank 4 symmetric[1] tensor $\mathbf{C}$ such that

$$\boldsymbol{\sigma}(x) = \mathbf{C}(x) \cdot \boldsymbol{\epsilon}(x). \tag{14.5}$$

[1] More specifically, the symmetries we here refer to are $C_{ij,kl} = C_{ji,kl} = C_{ij,lk} = C_{kl,ij}$ when we consider the individual tensor components of C.

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

Table 14.1 *Common linear elastic constants.*

Name	Symbol	Unit	Description
Young's modulus	E	$[p]^*$	Resistance to uniaxial stress when lateral boundaries are unconstrained
Poisson's ratio	ν	[–]	The negative of the ratio of lateral to longitudinal strain under uniaxial loading
Bulk modulus	K	$[p]$	Resistance to uniform compression (inverse of *compressibility*)
Shear modulus[†]	G	$[p]$	Resistance to shear stress
Lamé's parameter	λ	$[p]$	No immediate physical description but results in a simple expression for the constitutive relation between stresses when used together with the shear modulus
Vertical incompressibility[‡]	K_v	$[p]$	Resistance to uniaxial stress, with constrained ("roller") lateral boundaries

*Pressure = $[M/T^2L]$.
†Also called *Lamé's second parameter*.
‡Also called p-wave modulus.

This is called the *generalized Hooke's law*. In the most general case in 3D, one needs 21 independent stiffness coefficients to specify $\mathbf{C}$ for an elastic material. However, when the material in question satisfies certain additional and commonly encountered symmetry relations (specifically: *monoclinic*, *orthotropic* materials) and we moreover assume the elastic properties to be *isotropic* (independent of direction), only two degrees of freedom remain. To link $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$ in this case, it is sufficient to specify the value of two different *elastic constants*. A number of such constants exist, connected by mathematical relationships that imply that by fixing two of them (see Table 14.1), all of the others can be derived [20].

The choice of Young's modulus E and Poisson's ratio ν enables us to describe the linear relationship between stresses and strain as

$$\boldsymbol{\sigma}(x) = \frac{E}{1+\nu}\left[\boldsymbol{\epsilon}(x) + \frac{\nu}{1-2\nu}\,\mathrm{tr}\big(\boldsymbol{\epsilon}(x)\big)\mathbf{I}\right], \tag{14.6}$$

where $\mathrm{tr}(\boldsymbol{\epsilon})$ denotes the trace of $\boldsymbol{\epsilon}$ and $\mathbf{I}$ is the identity tensor. Equivalent formulations can, of course, be obtained using different choices of constants. For the purpose of the discussion in Subsection 14.2.1, for instance, the same relationship will be presented in terms of the bulk (K) and shear (G) moduli. For the rest of this chapter, we assume that the linear elastic relationship between stresses and strains can be written as in (14.6); i.e., we restrict our discussion to isotropic materials that satisfy all of the required symmetry relationships just mentioned.

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

The Displacement Formulation

By combining the generalized Hooke's law (14.5) with the force equilibrium equation (14.4) and the definition of the infinitesimal strain tensor (14.2), we obtain the *displacement formulation* of the governing linear elasticity equation for the interior of a solid occupying the volume Ω:

$$\nabla \cdot \left(\mathbf{C}(x) \cdot \tfrac{1}{2}(\nabla \vec{u}(x) + (\nabla \vec{u}(x))^T \right) + \vec{b} = 0, \qquad \forall x \in \Omega. \tag{14.7}$$

If, moreover, the material fulfills the additional requirements mentioned in Subsection 14.2.1, and *if material properties are constant throughout* Ω, (14.7) can be further developed; for instance:

$$\frac{E}{2(1+\nu)} \left(\nabla^2 \vec{u}(x) + \frac{1}{1-2\nu} \nabla \big(\nabla \cdot \vec{u}(x) \big) \right) + \vec{b} = 0, \qquad \forall x \in \Omega. \tag{14.8}$$

Note that whereas (14.8) has here been expressed in terms of Young's modulus E and Poisson's parameter ν, other choices of elastic constants lead to different but equivalent expressions. The infinitesimal displacements $\vec{u}$ are the unknowns in (14.7) and (14.8). In 3D space, this elliptic partial differential vector equation can be written out as a set of three scalar partial differential equations for each component of $\vec{u}$ (i.e., u_x, u_y, and u_z in a Cartesian coordinate system).

To complete the specification of the linear elastic problem, (14.7) has to be supplemented with proper conditions on the boundary of Ω. This boundary, denoted $\partial\Omega$, can be divided up into two nonoverlapping parts, $\partial\Omega = \Gamma_u \cup \Gamma_\sigma$, such that

$$\vec{u} = \vec{g}_0 \ \text{ on } \Gamma_u, \tag{14.9}$$

$$\boldsymbol{\sigma} \cdot \vec{n} = \vec{t}_0 \ \text{ on } \Gamma_\sigma, \tag{14.10}$$

where $\vec{g}_0 : \Gamma_u \to \mathbb{R}^3$ gives prescribed displacements on the Dirichlet boundary Γ_u, $\vec{t}_0 : \Gamma_\sigma \to \mathbb{R}^3$ gives prescribed forces acting on the Neumann boundary Γ_σ, and $\vec{n}$ represents the surface normal of an elementary surface patch of Γ_σ. For the problem to be well-posed, it is also necessary that Γ_u has nonzero measure [6].

Dilation and Mean Stress

Equation (14.6) represents one way to express the linear relationship between the stress and strain tensors under the hypotheses of the previous section, but it is not the only one. Expressed in this form, the equation does not allow us to differentiate between the separate effects of normal and shear stresses. When discussing the force balance equations for linear poroelasticity in Subsection 14.2.3, we will see that fluid pressure affects only normal stress, not shear stress. The basic constitutive relations introduced by Biot, which we present in Subsection 14.2.2 by (14.19) and (14.20), only regard volume change and compressibility; they do not involve shear

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

effects at all. In order to understand how to move from these relations to the full tensor poroelastic equation of Subsection 14.2.3, the contents of the present section will be helpful.

We will here develop a way to express the linear relationship between stress and strain in a way that clearly identifies the separate roles of normal and shear stresses and strains. For this purpose, Young's modulus E and Poisson's ratio ν are not particularly useful. Instead, we will express relation (14.6) in terms of bulk K and shear G moduli. The relations linking K and G to E and ν are

$$K = \frac{E}{3(1-2\nu)} \qquad G = \frac{E}{2+2\nu}. \tag{14.11}$$

We start by mathematically defining normal (bulk) and shear stress. The stress tensor can be understood as a sum of a *deviatoric stress tensor* and a *mean normal stress tensor*, also known as *hydrostatic stress tensor* or *volumetric stress tensor*. The latter is given by $\sigma\mathbf{I}$, where the scalar σ represents the mean of the trace of $\boldsymbol{\sigma}$:

$$\sigma = \tfrac{1}{3}\,\mathrm{tr}(\boldsymbol{\sigma}), \tag{14.12}$$

and the deviatoric stress $\tilde{\boldsymbol{\sigma}}$ is the remaining part:

$$\tilde{\boldsymbol{\sigma}} = \boldsymbol{\sigma} - \sigma\mathbf{I}. \tag{14.13}$$

Under the assumptions of Subsection 14.2.1, the mean normal stress represents the part of the stress tensor that changes the volume of (a compressible) solid when applied, whereas deviatoric stress leads to a volume-preserving deformation.

Likewise, we can decompose the strain tensor into a *mean strain tensor*, $\epsilon\mathbf{I}$, and a *deviatoric strain tensor*, $\tilde{\boldsymbol{\epsilon}}$. The scalar $\epsilon = \mathrm{tr}(\boldsymbol{\epsilon})$ is referred to as *dilation* or *volumetric strain* and represents a (relative) volume change $\partial V/V$. On the other hand, the deviatoric strain, $\tilde{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon} - \frac{1}{3}\epsilon\mathbf{I}$, represents a volume-preserving deformation.

If we reformulate (14.6) in terms of bulk (K) and shear (G) modulus (14.11) and substitute with definitions (14.12) and (14.13), we obtain after some manipulations:

$$\boldsymbol{\sigma} = 2G\tilde{\boldsymbol{\epsilon}} + K\epsilon\mathbf{I}, \tag{14.14}$$

which we can further decompose as

$$\tilde{\boldsymbol{\sigma}} = 2G\tilde{\boldsymbol{\epsilon}}, \tag{14.15}$$

$$\sigma = K\epsilon. \tag{14.16}$$

As we see here, the bulk modulus K expresses a linear relation between mean strain (volume change) and mean normal stress, whereas shear modulus G expresses a linear relation between deviatoric strain and stress. Shear stress does not affect volume change at all.

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

Discretization of the Linear Elastic Equations in MRST

The most common way of discretizing the linear elastostatic problem is using the finite-element method (FEM), which solves the weak formulation of the governing equations. This can be obtained by posing the elastostatic problem as the minimization of an energy functional:

$$\vec{u} = \operatorname{argmin}_{\vec{v}\in\mathcal{V}^g}\left(\frac{1}{2}a(\vec{v},\vec{v}) - f(\vec{v})\right). \tag{14.17}$$

In this minimization problem, the bilinear and linear operators are defined as

$$a(\vec{u},\vec{v}) = \int_\Omega \left[\mathbf{C}(x)\cdot\boldsymbol{\epsilon}_u(x)\right]\cdot\boldsymbol{\epsilon}_v(x)\,\mathrm{d}x, \qquad f(\vec{v}) = \int_\Omega \vec{b}\cdot\vec{v}\,\mathrm{d}x + \int_{\partial\Omega_\sigma}\vec{t}_0\cdot\vec{v}\,\mathrm{d}x,$$

where $\boldsymbol{\epsilon}_u$ and $\boldsymbol{\epsilon}_v$ are the strain fields associated with displacement fields $\vec{u}$ and $\vec{v}$, and $\mathcal{V}^g$ is the set of all admissible displacement fields that equal g_0 on Γ_u while satisfying some additional integrability requirements. Using calculus of variations, one can show that the equations of the previous section can be derived directly from (14.17). FEM solves the minimization problem (14.17) on a finite-dimensional subspace $\mathcal{V}_h^g \subset \mathcal{V}^g$ consisting of (typically low-degree) piecewise polynomials defined on a discrete mesh, requiring that the unknown $\vec{u} \in \mathcal{V}_h^g$ satisfies

$$a(\vec{u},\vec{v}) = f(\vec{v}), \qquad \forall\vec{v}\in\mathcal{V}_h^0, \tag{14.18}$$

where the set $\mathcal{V}_h^0$ is defined like $\mathcal{V}_h^g$ except being zero on Γ_g.

FEM is popular for solving problems in linear elasticity because it works well on this type of elliptic problem and can be applied without trouble as long as the inner product $a(\vec{u},\vec{v})$ is easy to compute on the finite-dimensional subspaces $\mathcal{V}_h^g$ and $\mathcal{V}_h^0$. This is generally the case for simple meshes consisting of simplices or quadrilaterals but quickly becomes impractical for more complex meshes. This is a problem for industrial-standard grids used in reservoir modeling, in which cells may have arbitrary polygonal shapes that are not necessarily convex, may have any number of vertices, and frequently contain degenerate edges.

The virtual element method (VEM) began to attract attention around 2012 [2]. It arose from work on extending mimetic finite-difference methods[2] to higher-order schemes. It turned out that this approach was more conveniently understood when recast in the conceptual framework of the FEM, formulated as the solution of a variational problem that involves minimization over discretized function spaces, just like FEM. In fact, VEM can be understood as an extension of finite-element

[2] Mimetic methods are a family of finite-difference methods defined on polyhedral grids, constructed to "mimic" certain properties of the partial differential equations they were used to discretize, such as mass conservation or maximum principles; see, e.g., [6] or section 6.4 of the MRST textbook [11].

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

theory that involves function spaces described in terms of a richer set of basis functions than piecewise polynomials [3].

The word "virtual" in VEM stems from the fact that the involved basis functions are never explicitly computed and remain unknown in the interior of each cell in the simulation grid. Only the values of the functions at certain points on cell boundaries are known (for first-order schemes, these would be the nodal values). However, the discrete function space is constructed so that the bilinear form $a(\vec{u}, \vec{v})$ in (14.18) over a cell K can nevertheless be computed *exactly* from the given boundary values when either $\vec{u}$ or $\vec{v}$ restricted to K is a piecewise polynomial of degree lower than or equal to the numerical order of the scheme. On the other hand, for nonpolynomial components of $\vec{u}$ and $\vec{v}$, the approximation of the bilinear form is limited to simple estimates constructed to remain within the right order of magnitude, which ensures the stability and convergence of the method.

The big advantage that VEM holds over FEM is that it can be easily applied to complex grids with very general cell shapes. In fact, because $a(\vec{u}, \vec{v})$ can be properly approximated from point-wise boundary values with no need for actually computing the basis functions in cell interiors, the main complication of FEM just mentioned is circumvented entirely. The price to be paid is the error introduced when evaluating the bilinear form on nonpolynomial components of $\vec{u}$ and $\vec{v}$, but such components are often small compared to the polynomial parts and the scheme still converges. For meshes consisting of simplices (triangles in 2D or tetrahedra in 3D), first-order VEM and FEM are equivalent and translate into the same system of linear equations.

The `vemmech` module of MRST provides an implementation of first-order VEM for linear elasticity problems. Its implementation follows the formulation and terminology used in [7], which provides a practical description of VEM applied to linear elasticity. A demonstration of the use of the `vemmech` module is given in Subsection 14.5.1.

14.2.2 Equations of Linear Poroelasticity

Poromechanics is the study of the coupling between mechanics and fluid flow in a porous medium, whether that medium is water-filled rock or soil in geomechanics or blood-filled living tissue in the medical sciences. A *poroelastic* medium is a porous medium consisting of an elastic solid matrix whose pores are filled with a viscous fluid (Figure 14.2). This is the model most commonly used in the study of geomechanics. In this chapter, we limit ourselves to the case in which the matrix is linear elastic.

The topic of poromechanics was mostly developed within the last hundred years, motivated by the need to understand key phenomena such as soil consolidation,

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

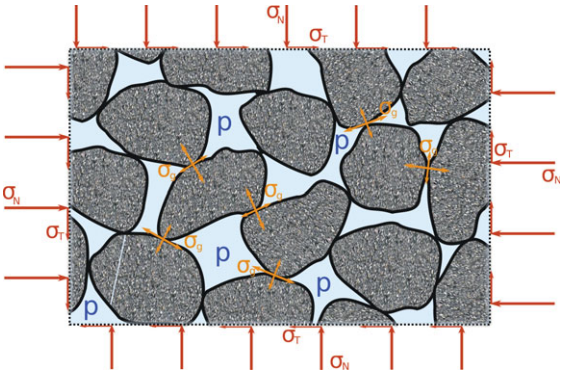

Figure 14.2 The mechanics of a porous media. The *bulk* consists of both solid grains (gray) and fluid-filled voids (blue). Forces acting on/within the system include the applied stress $\boldsymbol{\sigma}$ (red), the *pore pressure* p, and the grain-to-grain contact stresses σ_g. Note that the stress on the boundary is here explicitly represented in terms of normal stress, σ_N, and shear stress, σ_T.

storage of water in confined aquifers, and land subsidence following hydrocarbon extraction [20]. The key concept of *effective stress*, which will be discussed later in this chapter, was introduced by Karl Terzaghi [18], who conducted a series of lab experiments between 1916 and 1925 to understand the behavior of soil as a foundation material. The first full-fledged theory of poroelasticity was presented by Maurice Biot in 1941 [4]. Central to this theory were two linear constitutive relations linking isotropic mean stress σ (see Subsection 14.2.1), volumetric strain $\epsilon = \delta V/V$ (see Subsection 14.2.1), fluid pressure p, and the *increment of fluid content* ζ. The latter quantity represents the change in fluid volume for a given reference volume after a change in applied stress or in pressure occurs. As such, it is a dimensionless quantity (volume divided by volume) just like ϵ. The constitutive relations introduced by Biot are

$$\delta\epsilon = \frac{1}{K}\delta\sigma + \frac{1}{H}\delta p, \tag{14.19}$$

$$\delta\zeta = \frac{1}{H_1}\delta\sigma + \frac{1}{R}\delta p. \tag{14.20}$$

We recognize the bulk modulus K from Table 14.1 linking mean stress and volumetric strain. In addition, (14.19) and (14.20) present three additional moduli: H, H_1, and R. Assuming the existence of a potential energy density $U = \frac{1}{2}(\sigma\epsilon + p\zeta)$, it is straightforward to show that $H_1 = H$ [20], and thus the effect of a change in pore pressure on bulk volume (at constant applied stress) equals the effect of a change in applied stress on fluid content (at constant pressure). Within Biot's framework, three moduli are thus needed to characterize a linear poroelastic

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

medium. The full linear poroelastic equations that include shear also need the shear modulus, thus bringing the total number of degrees of freedom to four. Note that this is two more than what was needed to characterize an isotropic linear elastic medium in Subsection 14.2.1. This is not surprising, considering that a description of the poroelastic system also needs to account for the separate compressibility of the fluid, as well a characterization of the interaction between the fluid and the mechanics of the porous medium.

From K, H, R, and G, a large number of poroelastic moduli can be defined, with different utility depending on the context. (Although a poroelastic system can be fully characterized by different choices of four poroelastic constants, at least one of the constants must include a property related to shear deformation.) We will discuss some of them further in Section 14.3.

14.2.3 The Linear Poroelastic Equations

Starting from Biot's constitutive relationships (14.19) and (14.20), the equations of linear elasticity, and the one-phase flow equation for a porous medium, we will derive the governing equations of the linear poroelastic system. These can be seen as a set of elliptic equations expressing the force balance in the solid matrix and a parabolic equation expressing fluid flow, linked together by coupling terms. We will start by looking at the force balance of the matrix.

The Force Balance Equations for the Linear Poroelastic Matrix

Starting from relation (14.19) (and disposing of the deltas for notational convenience, while keeping in mind that we are still considering differentials), we can solve for mean stress to obtain

$$\sigma = K\epsilon - \alpha p, \tag{14.21}$$

where we have introduced the *Biot–Willis coefficient* $\alpha = \frac{K}{H}$. Comparing with the constitutive relationship for mean stress in linear elasticity, (14.16), we see that the only difference is that for poroelasticity we have the additional term $(-\alpha p)$, which expresses the influence of fluid pressure on the mean stress. On the other hand, deviatoric stress (14.15), which is volume preserving, is considered to remain unaffected by fluid pressure. The full linear poroelastic stress tensor can thus be described by simply subtracting αp from the mean stress, resulting in the following expression (see (14.14)):

$$\boldsymbol{\sigma} = 2G\tilde{\boldsymbol{\epsilon}} + (K\epsilon - \alpha p)\mathbf{I} = \mathbf{C} \cdot \boldsymbol{\epsilon} - \alpha p \mathbf{I}, \tag{14.22}$$

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

where the second equality assumes that we consider an isotropic medium (see Subsection 14.2.1). We combine (14.22) with the force equilibrium equation (14.4) to obtain

$$\nabla \cdot \left[\mathbf{C}(x) \cdot \boldsymbol{\epsilon}(x)\right] - \nabla(\alpha(x)p(x)) + \vec{b}(x) = 0, \qquad \forall x \in \Omega. \tag{14.23}$$

This expression is similar to the displacement formulation for linear elasticity (14.7), with an additional coupling term in pressure.

In the discussion of linear poroelasticity, a frequently encountered concept is that of *effective stress*. Moving around the terms of (14.22), we get $\mathbf{C}\boldsymbol{\epsilon} = \boldsymbol{\sigma}'$, where $\boldsymbol{\sigma}'$ denotes effective stress, defined as

$$\boldsymbol{\sigma}' = \boldsymbol{\sigma} + \alpha p \mathbf{I}. \tag{14.24}$$

We see that the deformation of the rock matrix results both from the applied stresses $\boldsymbol{\sigma}$ and an isotropic tensor proportional with pore pressure $\alpha p\mathbf{I}$. One way to interpret $\boldsymbol{\sigma}'$ is the stress needed in a linear elastic (not poroelastic) system to produce the same deformation as the combined effect of stress and pore pressure in the poroelastic system with the same stiffness tensor $\mathbf{C}$.

The Flow Equation

The increment of fluid content ζ plays the role of the accumulation term in the fluid continuity equation with volumetric fluid flux $\vec{q}$ and source term Q:

$$\dot{\zeta} + \nabla \cdot \vec{q} = Q. \tag{14.25}$$

By rearranging terms in (14.19) and (14.20), we can express ζ in terms of volumetric strain:

$$\zeta = \frac{K}{H}\epsilon + \left(\frac{1}{R} - \frac{K}{H^2}\right) p. \tag{14.26}$$

We recognize the Biot–Willis coefficient $\frac{K}{H} = \alpha$ as the coefficient of ϵ in the expression in (14.26). The coefficient in front of p is called the *specific storage coefficient at constraint strain*. It is a poroelastic modulus with its own symbol S_ϵ:

$$S_\epsilon = \left(\frac{\partial \zeta}{\partial p}\right)_{\epsilon=\text{const.}} = \left(\frac{1}{R} - \frac{K}{H^2}\right). \tag{14.27}$$

Substituting α and S_ϵ into (14.26) and taking the time derivative gives us the expression of the first term of (14.25) in terms of volumetric strain and pressure changes:

$$\dot{\zeta} = \alpha\dot{\epsilon} + S_\epsilon \dot{p}. \tag{14.28}$$

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

We use Darcy's law to express $\vec{q}$ in terms of pressure,

$$\vec{q} = -\frac{1}{\mu}\mathbf{K}\big(\nabla p - \rho_f \vec{g}\big), \tag{14.29}$$

where μ is the fluid viscosity, ρ_f is the fluid density, $\mathbf{K}$ is the permeability of the porous medium, and $\vec{g}$ is gravitational acceleration (z-axis oriented downwards).

Inserting the expressions for $\vec{q}$ and $\dot{\zeta}$ into (14.25) yields

$$\alpha\dot{\epsilon} + S_\epsilon \dot{p} - \frac{1}{\mu}\mathbf{K}\left(\nabla p - \rho_f \vec{g}\right) = Q. \tag{14.30}$$

This is the poroelastic fluid continuity equation, expressed in terms of pressure and volumetric strain. (Other equivalent expressions are possible, e.g., with the accumulation term given in terms of pressure and mean stress.) Equation (14.30) is a standard volumetric fluid continuity equation expressed in terms of pressure, assuming Darcy flow, and with an extra coupling term $\alpha\dot{\epsilon}$ that describes its dependence on the force balance equations for the solid matrix (14.23). Taken together, these equations constitute the full system of poroelastic equations describing flow and mechanical deformation of the system under study. In addition, we need boundary conditions. The mechanical boundary conditions for the solid matrix are those already given by (14.9) and (14.10) above. In addition, we need to specify the flow boundary conditions for the fluid phase. To do this, we once again subdivide $\partial\Omega$ into two nonoverlapping parts, $\partial\Omega = \Gamma_p \cup \Gamma_{\vec{q}}$, such that pressure is prescribed (constant or equal to some function p_0) on Γ_p and flux is prescribed (typically zero; i.e., *no-flow*) on $\Gamma_{\vec{q}}$.

The Full Linear Poroelastic Equation System

Taken together, the full poroelastic system of equations with boundary conditions thus reads:

$$\nabla \cdot \big[\mathbf{C} \cdot \boldsymbol{\epsilon}\big] - \nabla(\alpha p) - \rho_b \vec{g} = 0, \qquad \text{in } \Omega \text{ (matrix)}, \tag{14.31}$$

$$\alpha\dot{\epsilon} + S_\epsilon \dot{p} - \frac{1}{\mu}\mathbf{K}\left(\nabla p - \rho_f \vec{g}\right) = Q, \qquad \text{in } \Omega \text{ (fluid)}, \tag{14.32}$$

with mechanical boundary conditions

$$\vec{u} = \vec{g}_0, \quad \text{on } \Gamma_u, \tag{14.33}$$

$$\boldsymbol{\sigma} \cdot \vec{n} = \vec{t}_0, \quad \text{on } \Gamma_\sigma, \tag{14.34}$$

and flow boundary conditions

$$p = p_0, \quad \text{on } \Gamma_p, \tag{14.35}$$

$$\vec{q} = \vec{q}_0, \quad \text{on } \Gamma_{\vec{q}}. \tag{14.36}$$

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

Note that in (14.31) we have here replaced the general body force term $\vec{b}$ of (14.23) with gravity $-\rho_b \vec{g}$, where ρ_b is the *bulk density* of the medium, a porosity-weighted average of the density ρ_m of the solid matrix and the density ρ_f of the fluid; i.e., $\rho_b = (1-\phi)\rho_m + \phi\rho_f$.

14.3 Moduli, Moduli, Moduli ...

For the student starting out to learn poroelastic theory, the subtopic of poroelastic moduli and coefficients may seem daunting, even intimidating. A large number of such entities are encountered in the literature, many of which express similar concepts under subtly different assumptions. Though they all can be derived from the basic poroelastic constants of Subsection 14.2.2 (plus porosity), they are also linked between themselves by a much larger number of poroelastic relationships, many of which may seem to provide little intuition to the beginner – an intractable jungle of equations with unclear usefulness.

A consolation would be that the most important insight to obtain is an overview of the different main categories of parameters (compressibilities, storativities, etc.) and the main types of assumptions involved (e.g., bulk vs. grain, drained vs. undrained, uniaxial vs. triaxial). A good understanding of these concepts will make it clear when and how to use them. The student should also rest reassured that these definitions exist because they are indeed useful and needed in particular contexts. Knowledge of specific poroelastic moduli often enables quick estimates without the need of numerical simulation (how much water can this aquifer hold under hydrostatic pressure?), guides the setup and interpretation of lab experiments, provides the basis for a range of analytical solutions to particular poroelastic problems in literature, helps in choosing a mutually compatible set of parameters when setting up simulations (e.g., choosing a value of the Biot–Willis coefficient that is compatible with the other chosen parameters in a coupled reservoir simulation), and enables validation of simulation results, as we will see in Section 14.4.3.

Table 14.2 shows a number of poroelastic parameters, roughly grouped by function. The ambition of this section is not to provide a full explanation of these but rather to provide an introduction to some central concepts and categories (drained and undrained moduli, specific storage coefficients), as well as discuss a handful of particularly important parameters, such as the Biot–Willis coefficient (which is central to coupling the poroelastic equations) and Geertsma's uniaxial expansion coefficient (which, among other, helps us understand the link with pore volume multipliers in industry-standard reservoir simulation software). Some other parameters that are relevant for the examples of Section 14.5 will also be briefly discussed. On a first read, the impatient reader may choose to skip parts of the present section and refer back to it when going through the examples given in Section 14.5.

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

Table 14.2 *List of elastic and poroelastic parameters. The last column specifies whether the quantity depends on shear modulus or not.*

Symbol	Name	Shear
Poroelastic constants from Biot's basic constitutive relationships		
K	Drained bulk modulus	
H	Inverse of poroelastic expansion coefficient	
R	Inverse of unjacketed specific storage coefficient	
M	Inverse of constrained specific storage coefficient	
Compressibilities (other than K)		
K_s	Unjacketed bulk modulus	
K_p	Inverse of drained pore compressibility	
K_f	Inverse of fluid compressibility	
K_ϕ	Inverse of unjacketed pore compressibility	
K_v	Uniaxial drained bulk modulus	x
Storativities		
S	Uniaxial specific storage coefficient	x
S_σ	Unconstrained specific storage coefficient	
S_ϵ	Constrained specific storage coefficient	
S_γ	Unjacketed specific storage	
Other parameters from linear elasticity, drained or undrained		
K_u	Undrained bulk modulus	
$K_v^{(u)}$	Uniaxial undrained bulk modulus	x
E	Young's modulus (drained)	x
E_u	Young's modulus (undrained)	x
λ	Lamé's parameter (drained)	x
λ_u	Lamé's parameter (undrained)	x
ν	Poisson's ratio (drained)	x
ν_u	Poisson's ratio (undrained)	x
G	Shear modulus (drained *and* undrained)	x
Other parameters		
α	Biot–Willis coefficient	
β	Effective stress coefficient for pore volume	
γ	Loading efficiency	x
η	Poroelastic stress coefficient	x
c_m	Geertsma's parameter	x
B	Skempton's coefficient	

Lastly, the large number of relations between poroelastic parameters can make it hard to find the easiest way to compute some specific value from a given set of already defined parameters. For instance, in the practical code example presented in Subsection 14.5.2 to study Terzaghi's problem, we will need to compute the *uniaxial specific storage coefficient* S from the given values of Young's modulus, Poisson's parameter, Biot–Willis coefficient, and fluid compressibility.

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

A code excerpt shows how this can be (painstakingly) done, but another alternative is also presented, in the form of a utility script called `poroParams`, provided by MRST. This utility script, when given an arbitrary selection of input parameters, will automatically compute as many of the parameters listed in Table 14.2 as possible from the provided input. This removes the need to manually navigate the large number of poroelastic relationships in order to arrive at the value of the target parameter. In Subsection 14.3.5, we will close the discussion of poroelastic moduli by giving an explanation on how this script can be used.

14.3.1 The Biot–Willis Coefficient, α

The Biot–Willis coefficient α plays a central role in the poroelastic equation system presented in Subsection 14.2.3, as a factor in the coupling terms. The influence of pressure on the mechanics equation system (14.31) is through the term $\nabla(\alpha p)$, and the influence of the mechanical strain on the flow equation (14.32) is through the term $\alpha\dot{\epsilon}$. In Subsection 14.2.3 we gave the definition $\alpha = \frac{K}{H}$, but from this expression it is unclear what α physically represents. We will here take a closer look at this question.

The most intuitive is perhaps to consider the coupling term $\alpha\dot{\epsilon}$ from flow equation (14.32), which is the part of the accumulation term that represents an incremental change in fluid content caused by an incremental change in volumetric strain (at constant fluid pressure). In other words, if the reference volume expands (or shrinks) by $\delta\epsilon$, this results in a change in pore volume of $\alpha(\delta\epsilon)$. The remainder, $(1-\alpha)\delta\epsilon$, represents a change in the volume of the solid matrix itself. If the rock matrix consists of incompressible rock grains, any change in bulk volume is entirely due to change in pore volume, in which case $\alpha = 1$.

To formalize this, we introduce the poroelastic constant *unjacketed bulk modulus*: $K_s' = \frac{HK}{H-K}$. Its reciprocal, $(K_s')^{-1}$, is referred to as *unjacketed bulk compressibility* and represents the change in volume of a poroelastic sample saturated by, and submerged in, a fluid when the pressure of that fluid changes. If we assume that the poroelastic samples consist of a uniform solid material (e.g., solid rock grains all of the same type), then K_s' also equals the *solid grain modulus* K_s, whose reciprocal expresses the compressibility of the solid material that constitutes the matrix. This assumption is not true in general but is a frequently made approximation. Substituting for H in our previous expression of α, we derive

$$\alpha = \frac{K}{H} = 1 - \frac{K}{K_s'} \approx 1 - \frac{K}{K_s}. \tag{14.37}$$

From this relation we see that when the compressibility K_s^{-1} of the solid material becomes insignificant compared to the compressibility K^{-1} of the bulk as a whole, then $\alpha \to 1$. In the general case, α takes on a value between ϕ (porosity) and 1.

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

14.3.2 Drained and Undrained Moduli

In general, the response of a poroelastic material to a change in stress conditions is time dependent, due to the parabolic nature of the flow equation (14.32). The time it takes for the material to settle into a new equilibrium depends on the permeability $\mathbf{K}$ of the porous medium; a high value of $\mathbf{K}$ allows fluid to rapidly flow and reequilibrate with an external boundary, whereas a low value of $\mathbf{K}$ restricts fluid flow and increases the transitory time.

It is interesting to consider the two limiting cases, where fluid either (i) re-equilibrates instantly or (ii) is entirely prevented from flowing through the medium. These are respectively referred to as *drained* and *undrained* conditions and express the response we can expect from a poroelastic material when applied stresses change on a timescale either much slower or much quicker than the time it takes for the fluid to reequilibrate.

Under drained conditions, the fluid pressure remains constant throughout the change in applied stress, as it instantly equilibrates with the external boundary. As a consequence, the bulk modulus of the porous medium is just K, as can easily be seen from Biot's constitutive relation (14.19) when $\delta p = 0$. Hence, the compressibility of the fluid does not contribute to the stiffness of the material at all, which only depends on the material of the solid matrix.

On the other hand, when conditions are undrained, the fluid remains in place and thus contributes to increase the stiffness of the porous medium. The bulk modulus under undrained conditions is denoted K_u and is related to K as follows:

$$K_u = \frac{K}{1 - \alpha B}, \tag{14.38}$$

where $B = R/H$ is known as *Skempton's coefficient*. (The physical interpretation of B is that of the ratio between change in pore pressure and applied stress under undrained conditions: $B = -\partial p/\partial \sigma$ for $\delta \zeta = 0$.) Because $\alpha B > 0$, we note that $K_u > K$, which is to be expected, because the fluid now also contributes to the stiffness of the material.

Most poroelastic constants have drained and undrained variants. For instance, the drained *Poisson parameter* is expressed in terms of K and G as

$$\nu = \frac{3K - 2G}{2(3K + G)}, \tag{14.39}$$

whereas the undrained variant is

$$\nu_u = \frac{3\nu + \alpha B(1 - 2\nu)}{3 - \alpha B(1 - 2\nu)}. \tag{14.40}$$

An exception to this is the *shear modulus*, G, which remains unaffected by fluid pressure and thus has the same value whether or not conditions are drained.

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

14.3.3 Specific Storage Coefficients

Specific storage coefficients express how much the fluid content of a reference volume changes as a function of fluid pressure, which is a highly relevant property in hydrogeology. Different assumptions on boundary conditions lead to different coefficients. We will here consider the cases of *constant stress*, *constant strain*, and *uniaxial strain*.

The *specific storage coefficient at constant stress*, denoted S_σ, represents the incremental change in fluid content $\delta\zeta$ of a reference volume for an incremental change in pore pressure p, assuming constant mean stress ($\delta\sigma = 0$). From Biot's constitutive relation (14.20) we immediately obtain

$$S_\sigma = R^{-1}. \tag{14.41}$$

Conditions under which the assumption of constant mean stress holds are rarely met in geomechanical applications. The definition of S_σ will nevertheless become useful in the discussion of coupling strategies in Section 14.4.

We first encountered the *specific storage coefficient at constant strain*, denoted S_ϵ, in Subsection 14.2.3, where we derived expression (14.27). It represents the change in fluid content due to a change in fluid pressure, assuming constant bulk volume ($\delta\epsilon = 0$). If we introduce the fluid compressibility K_f^{-1} and porosity ϕ, it is also possible to derive the following expression for S_ϵ, assuming that the solid phase consists of a uniform solid material

$$S_\epsilon = \frac{1}{K}(1-\alpha)(\alpha-\phi) + \frac{\phi}{K_f}. \tag{14.42}$$

(If this approximation cannot be made, yet another poroelastic constant K_ϕ needs to be specified for completeness; see [20].) The interesting aspect of this expression is that we have clearly identified the separate influences of fluid compressibility K_f^{-1} and bulk compressibility K^{-1} on the total change in fluid content. We will see how this becomes very handy when we discuss Geertsma's coefficient in Subsection 14.3.4.

The *uniaxial specific storage coefficient* S is frequently encountered in hydrogeology, although hydrogeologists tend to work with *head* rather than pressure, which means they would use the form $S_s = (\rho_f g)S$. It represents the change in fluid content from a change in fluid pressure, under the assumption of zero lateral strain and constant vertical stress. In other words, the reference volume cannot expand or contract laterally, and all changes to volumetric strain are due to displacements in the vertical direction. The expression for S is

$$S = S_\sigma\left(1 - \frac{4\eta B}{3}\right), \tag{14.43}$$

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

where η is the *poroelastic stress parameter*

$$\eta = \frac{1-2\nu}{2(1-\nu)}\alpha. \tag{14.44}$$

Note that contrary to the other specific storage coefficients discussed here, S also depends on the shear modulus G (through ν).

The particular utility of S is that with the underlying assumptions (zero lateral strain and constant vertical stress), mean normal stress becomes a function of fluid pressure only:

$$\sigma = -\frac{4}{3}\eta p, \tag{14.45}$$

which means that the volumetric strain also becomes a function of fluid pressure alone; see (14.21). As a consequence, the flow equation (14.32) decouples from the mechanics equations (14.31) and can be solved independently. (As stated in Subsection 14.2.1, we restrict discussion to isotropic materials satisfying all of the necessary symmetry relations for (14.6) to hold.) All of the influence of mechanics on flow is expressed through the parameter S, and no (computationally costly) mechanical equation system needs to be included.

We round off the discussion of storage parameters by noting that we always have the following order on magnitudes of the storage coefficients:

$$S_\sigma \leq S \leq S_\epsilon. \tag{14.46}$$

14.3.4 Geertma's Uniaxial Expansion Coefficient, C_m

We remain under the assumptions made for S in the previous section, namely, zero lateral strain and constant vertical stress, and define a corresponding bulk modulus K_v, referred to as *vertical incompressibility* or *p-wave modulus* (Table 14.1). The following expression can be derived:

$$K_v = K + \tfrac{4}{3}G. \tag{14.47}$$

From this, we define a new constant, *Geertsma's uniaxial expansion coefficient*,

$$c_m \equiv \frac{\alpha}{K_v}, \tag{14.48}$$

which represents the ratio of change of vertical strain to pore pressure under conditions of zero lateral strain and constant vertical stress. Note that because lateral strains are assumed zero, vertical strain is equivalent to total volumetric strain in this setting. In other words, we have

$$\delta\epsilon = c_m \delta p. \tag{14.49}$$

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

Again, we here implicitly make all the assumptions of Subsection 14.2.1 required for (14.6) to hold.

If we use (14.49) to replace the coupling term of the poroelastic flow equation (14.32), we obtain

$$\left(\alpha c_m + S_\epsilon\right)\dot{p} - \frac{1}{\mu}\mathbf{K}\left(\nabla p - \rho_f \vec{g}\right) = Q. \tag{14.50}$$

Under the assumptions of zero lateral strain and constant vertical stress, we obtain a flow equation whose only unknown is pore pressure and thus can be solved independent of the force balance equations (14.31). In other words, we can model flow without considering the associated mechanics. This is, of course, standard practice in reservoir simulation, where only flow is considered and the mechanical response of the porous rock is modeled using a rock compressibility parameter.

A closer look at the accumulation term of (14.50) helps us further understand the link with reservoir simulation practice. If we insert the expression for S_ϵ given by (14.42), the accumulation term can be written as

$$\left[\alpha c_m + \frac{1}{K}(1-\alpha)(\alpha-\phi)\right]\dot{p} + \frac{\phi}{K_f}\dot{p}. \tag{14.51}$$

The first term of this sum accounts for bulk expansion and grain compressibility, whereas the second accounts for fluid compressibility. The use of a rock compressibility parameter in industry-standard reservoir simulation software therefore conceptually represents the first term, whereas a fluid compressibility parameter conceptually accounts for the second. Note that when expressed on this form, it is easy to see how fluid compressibility can be generalized to the nonlinear case where fluid density is obtained from an equation of state.

14.3.5 Automatic Computation of Poroelastic Parameters

We have already discussed the existence of a large number of poroelastic parameters that can be derived from a small set of fundamental constants, including the moduli of Biot's basic constitutive relations (K, H, R), and sometimes porosity, and/or the shear modulus G. A (not exhaustive) list of parameters is given in Table 14.2. A large number of relations link these parameter so that in practice it is sufficient to know a small subset of them (not necessarily K, H, R, and G) to determine the others. However, identifying the exact set of relations that will help compute the quantities you do not know from the quantities you know can be laborious. For this reason, the `ad-mechanics` module provides a utility function called `poroParams`. This function takes the parameters you already know, whichever they may be, and uses them to compute as many as possible of the parameters you do not know, based on a large set of poroelastic relations hard-coded within the algorithm.

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

The algorithm takes two required arguments and a number of optional ones. The first required argument is porosity (a value strictly between 0 and 1). The second required argument is a true/false switch specifying whether the algorithm is allowed to assume uniform solid material (all rock grains have the same compressibility; see Subsection 14.3.1) or not. Finally, an arbitrary number of poroelastic parameter names and their values should be provided. If these are insufficient to compute all unknown poroelastic parameter values, the undetermined ones will be left at `NaN` (not-a-number) in the resulting output. On the other hand, if the provided variable values overdetermine the solution, a warning will be displayed.

We will here provide a quick demonstration of how this utility script can be used. We start by asking the script to show us which poroelastic variables are supported and what names they take:

```
help poroParams
```

This will list the supported parameters, which happen to be the same as those listed in Table 14.2. We now try to see what values we can compute by providing the values for K, H, and R (in addition to porosity), while assuming uniform solid material:

```
poroParams(0.25, true, 'K', 1e9, 'H', 1.2e9, 'R', 1.2e9)
```

```
ans =
  struct with fields:
            K: 1.0000e+09
            H: 1.2000e+09
            R: 1.1000e+09
            M: 4.6588e+09
          K_s: 6.0000e+09
          K_p: 300000000
          K_f: 2.1290e+09
        K_phi: 6.0000e+09
          K_v: NaN
            S: NaN
      S_sigma: 9.0909e-10
    S_epsilon: 2.1465e-10
      S_gamma: 7.5758e-11
        alpha: 0.8333
         beta: 0.9500
        gamma: NaN
          eta: NaN
          c_m: NaN
            B: 0.9167
          K_u: 4.2353e+09
        K_vu: NaN
            E: NaN
          E_u: NaN
       lambda: NaN
     lambda_u: NaN
           nu: NaN
        nu_u: NaN
            G: NaN
```

We see that the algorithm was able to compute quite a few poroelastic parameters from the provided input but that many remain undetermined. We try to call the function again, this time also providing a value for shear modulus G:

```
poroParams(0.25, true, 'K', 1e9, 'H', 1.1e9, 'R', 1.2e9, 'G', 0.9e9)
```

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

```
ans =
  struct with fields:
            K: 1.0000e+09
            H: 1.2000e+09
            R: 1.1000e+09
            M: 4.6588e+09
          K_s: 6.0000e+09
          K_p: 300000000
          K_f: 2.1290e+09
        K_phi: 6.0000e+09
          K_v: 2.2000e+09
            S: 5.3030e-10
      S_sigma: 9.0909e-10
    S_epsilon: 2.1465e-10
      S_gamma: 7.5758e-11
        alpha: 0.8333
```

```
     beta: 0.9500
    gamma: 0.7143
      eta: 0.3409
      c_m: 3.7879e-10
        B: 0.9167
      K_u: 4.2353e+09
    K_vu: 5.4353e+09
        E: 2.0769e+09
      E_u: 2.5214e+09
   lambda: 4.0000e+08
 lambda_u: 3.6353e+09
       nu: 0.1538
     nu_u: 0.4008
        G: 900000000
```

The algorithm now had sufficient information to compute the full set of parameters. If we try to overspecify the system by also providing a value for M that is different from what was computed, we get a warning:

```
poroParams(0.25, true, 'K', 1e9, 'H', 1.2e9, 'R', 1.1e9, 'G', 0.9e9, 'M', 3e9)
```

```
Warning: Some residuals did not vanish. Input parameters might over-specify system.
> In poroParams (line 134)

ans =
  struct with fields:
  ...
```

On the other hand, if we remove the assumption of uniform solid material, there are still a few parameters remaining that the algorithm was not able to determine:

```
poroParams(0.25, false, 'K', 1e9, 'H', 1.2e9, 'R', 1.1e9, 'G', 0.9e9)
```

```
ans =
  struct with fields:
            K: 1.0000e+09
            H: 1.2000e+09
            R: 1.1000e+09
            M: 4.6588e+09
          K_s: 6.0000e+09
          K_p: 300000000
          K_f: NaN
        K_phi: NaN
          K_v: 2.2000e+09
            S: 5.3030e-10
      S_sigma: 9.0909e-10
    S_epsilon: 2.1465e-10
      S_gamma: 7.5758e-11
        alpha: 0.8333
```

```
     beta: NaN
    gamma: 0.7143
      eta: 0.3409
      c_m: 3.7879e-10
        B: 0.9167
      K_u: 4.2353e+09
    K_vu: 5.4353e+09
        E: 2.0769e+09
      E_u: 2.5214e+09
   lambda: 4.0000e+08
 lambda_u: 3.6353e+09
       nu: 0.1538
     nu_u: 0.4008
        G: 900000000
```

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

Finally, we note that the system can be specified by any selection of parameters, as long as they are compatible and do not overspecify the system:

```
poroParams(0.25, true, 'nu', 0.15, 'B', 1, 'alpha', 0.9, 'c_m', 1e-10)
```

```
ans =
  struct with fields:
            K: 4.0588e+09
            H: 4.5098e+09
            R: 4.5098e+09
            M: 4.5098e+10
          K_s: 4.0588e+10
          K_p: 1.1275e+09
          K_f: 4.0588e+10
        K_phi: 4.0588e+10
          K_v: 9.0000e+09
            S: 1.1217e-10
      S_sigma: 2.2174e-10
    S_epsilon: 2.2174e-11
      S_gamma: 1.2925e-26
        alpha: 0.9000
         beta: 0.9722
        gamma: 0.8915
          eta: 0.3706
          c_m: 1.0000e-10
            B: 1
          K_u: 4.0588e+10
         K_vu: 4.5529e+10
            E: 8.5235e+09
          E_u: 1.0789e+10
       lambda: 1.5882e+09
     lambda_u: 3.8118e+10
           nu: 0.1500
        nu_u: 0.4557
            G: 3.7059e+09
```

14.4 Coupling Strategies

The full linear poroelastic system presented in Subsection 14.2.3 consists of one mechanics equation (14.31), whose primary unknown is the displacement field $\vec{u}$, and one flow equation (14.32), whose primary unknown is the fluid pressure p. In the following discussion, we respectively refer to these equations as E_m and E_f. The two equations are linked with coupling terms, so that the mechanics equation depends on the gradient of the fluid pressure through the term $\alpha \nabla p$, and the flow equation depends on the time derivative of the volumetric strain, $\dot{\epsilon}$.

14.4.1 Fully Coupled and Sequentially Split Schemes

The equations E_m and E_f, with associated boundary conditions, describe an evolution in the poroelastic system $(\vec{u}, p)$ over time. If we consider a time discretization $t^n = n\Delta t$, the evolution of the system from time t^n to time t^{n+1} can be expressed in an abstract sense as

$$(\vec{u}^n, p^n) \xrightarrow{\mathcal{A}} (\vec{u}^{n+1}, p^{n+1}), \tag{14.52}$$

where $\mathcal{A}$ is the operator representing the combined system (E_m, E_f), and the time derivative of E_f has been approximated using a backward Euler time discretization. Combined with a spatial discretization of $\vec{u}$, p, and $\mathcal{A}$, the timestep can be computed by solving one combined system of equations. (In our case, for the MRST examples presented in this chapter, the spatial discretization will be *virtual elements* for the mechanics equation and *finite volumes* for the flow equation.) Note that for pure

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

linear poroelastic systems these equations would be linear but not necessarily so in the more general case; e.g., when working with an extension of the flow equation to the multiphase setting. We refer to this as solving the *fully coupled* problem.

Another approach to solving the equation system (E_m, E_f) would be to employ *operator splitting*, in which the two parts of the equation system are solved sequentially, with some approximation of the coupling term that does not depend on variables not explicitly covered by the equation. We could, for example, imagine the following scheme:

$$(\vec{u}^n, p^n) \xrightarrow{\mathcal{A}_m^{\mathrm{ds}}} (\vec{u}^{n+1}, p^n) \xrightarrow{\mathcal{A}_f^{\mathrm{ds}}} (\vec{u}^{n+1}, p^{n+1}). \tag{14.53}$$

Here, $\mathcal{A}_m^{\mathrm{ds}}$ is an operator that solves the mechanical equation E_m for $\vec{u}^{n+1}$, where the coupling term $\alpha \nabla p$ is evaluated from the already available p^n. Likewise, $\mathcal{A}_f^{\mathrm{ds}}$ is an operator that solves the flow equation E_f for p^{n+1} using the newly computed displacements u^{n+1} to compute the coupling term $\dot{\epsilon}$. The approach (14.53) is referred to as the *drained split* scheme (hence the superscript "ds"). Its name comes from the fact that keeping pressure constant at p^n when solving for $\vec{u}^{n+1}$ amounts to considering fully drained conditions (see Subsection 14.3.2).

An important motivation for employing operator-splitting schemes is the ability to employ different, existing numerical codes to solve the mechanics and the flow equations. This enables the combined use of highly sophisticated existing mechanical and flow simulators to address the coupled poroelasticity problem. A typical example would be TOUGH-FLAC [16], in which TOUGH2 is used to compute multiphase flow, whereas the geomechanical equations are solved using the commercial FLAC^{3D} simulator. Another potential advantage of operator-splitting schemes over the fully coupled scheme (14.52) could be computational efficiency, although this highly depends on the operator-splitting scheme used, the size of the discretized system, and other factors such as the nature of the linear solver and the degree of hardware parallelism employed [1, 9, 10, 14]. Another aspect relevant to computational efficiency of an operator-splitting approach is whether the scheme is applied in a *staggered* (single-pass) or *iterative* manner. In the staggered approach, the operator-splitting procedure (e.g., (14.53) for the drained split scheme) is only applied once per timestep, whereas an iterative approach repeats the procedure until convergence before proceeding to the next timestep. The former is less precise per timestep than the latter but allows the computation of a larger number of timesteps for a given computational cost.

14.4.2 The Fixed Stress Split Scheme

The main advantage of the drained split scheme (14.53) is that it is easy to explain and implement. Otherwise, it does not have much going for it. It is only stable

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

for weakly coupled problems (regardless of timestep size) and is generally not convergent for a fixed number of iterations (such as the staggered approach). In the general case, a much better approach is the *fixed stress split* method. Whereas the drained split approach solves the mechanics equation first by keeping pressure constant, the fixed stress split solves the flow equation first by keeping the rate of mean stress change constant. The accumulation term of the flow equation (14.32) reads:

$$\alpha\dot{\epsilon} + S_\epsilon \dot{p} = \frac{\alpha}{K}\dot{\sigma} + \left(\frac{\alpha^2}{K} + S_\epsilon\right)\dot{p}, \tag{14.54}$$

where the expression on the right was obtained by substituting (14.21) into the expression on the left. We discretize in time and use Δ_n to denote the forward difference $\Delta_n a = a^{n+1} - a^n$ to obtain

$$\frac{\alpha}{K}\Delta_n\sigma + \left(\frac{\alpha^2}{K} + S_\epsilon\right)\Delta_n p. \tag{14.55}$$

The fixed-stress approach considers the rate of change of stress to be constant, $\Delta_n\sigma = \Delta_{n-1}\sigma$, which enables us to write

$$\frac{\alpha}{K}\Delta_{n-1}\sigma + \left(\frac{\alpha^2}{K} + S_\epsilon\right)\Delta_n p. \tag{14.56}$$

We use relation (14.21) again to substitute back for σ

$$\left(\alpha\Delta_{n-1}\epsilon - \frac{\alpha^2}{K}\Delta_{n-1}p\right) + \left(\frac{\alpha^2}{K} + S_\epsilon\right)\Delta_n p. \tag{14.57}$$

Note that the terms inside the first parentheses are all known at timestep n, and this grouping plays the role of an additional source term in the discretized flow equation. When we replace the accumulation term of a time-discretized version of the flow equation (14.32) by the expression (14.57), we obtain

$$\left(\frac{\alpha^2}{K} + S_\epsilon\right)\Delta_n p - \Delta_n t\frac{\mathbf{K}}{\mu}\left(\nabla p^{n+1} - \rho_f \vec{g}\right) = \Delta_n t\, Q - \left(\alpha\Delta_{n-1}\epsilon - \frac{\alpha^2}{K}\Delta_{n-1}p\right). \tag{14.58}$$

This is a modified equation expressed in terms of pressure only. The fixed-stress scheme first computes p^{n+1} from (14.58) and then uses the new pressure to compute σ^{n+1} from the mechanics equation. Schematically,

$$(\vec{u}^n, p^n) \xrightarrow{\mathcal{A}_f^{\text{fss}}} (\vec{u}^*, p^n) \xrightarrow{\mathcal{A}_m^{\text{fss}}} (\vec{u}^{n+1}, p^{n+1}), \tag{14.59}$$

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

Table 14.3 *Coupled flow and mechanics models in* `ad-mechanics`.

Name	Solver	Flow model
`MechWaterModel`	fully coupled	one-phase flow
`MechOilWaterModel`	fully coupled	two-phase flow
`MechBlackOilModel`	fully coupled	three-phase black-oil flow
`MechFluidFixedStressSplitModel`	fixed stress split	one-phase, two-phase, or three-phase black-oil (chosen when setting up the model)

where $\mathcal{A}_f^{\text{fss}}$ represents the computation of p^{n+1} from (14.58), and $\mathcal{A}_m^{\text{fss}}$ represents the ensuing computation of $\vec{u}^{n+1}$ using the mechanics equations. Note that $\vec{u}^*$ is here used as a placeholder in the diagram to represent the displacements associated with fixed stress between timesteps but is never actually computed.

The fixed-stress split scheme has many favorable properties. In the fully implicit form just outlined, it is unconditionally stable regardless of timestep size or coupling strength and nonoscillatory and converges quickly in most cases when used in iterative schemes. It also converges to the correct solution when using a staggered approach. This is the splitting approach chosen for the MRST implementation of coupled flow and geomechanics discussed in Subsection 14.4.3.

The stability, convergence, and performance of sequential methods for coupled geomechanics have been studied extensively, and the interested reader may refer to [9] and [14] for in-depth detail and analysis.

14.4.3 The `ad-mechanics` *Module in MRST*

Coupled flow and geomechanics within the linear elastic framework is implemented in `MRST` in the `ad-mechanics` module. The module provides models for solving the coupled flow and mechanics problem for the cases of one-phase, two-phase, and three-phase (black-oil) flow and implements both a fully coupled solver and a solver based on operator splitting using the fixed-stress split scheme.

The mechanics equations are discretized using the first-order VEM, whereas the flow equations are discretized using standard finite volumes with two-point flux and a fully implicit first-order Euler scheme in time. Table 14.3 outlines the available models. We will see some examples of their application to (idealized) model problems in Section 14.5, but they have also been applied with success on real industry-sized reservoir grids.

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

14.5 Numerical Examples

In this section, we will demonstrate the linear elastic and poroelastic solvers of MRST on a few basic examples. We start with a simple case of compression of a solid cylinder (a pure linear elastic problem), which shows how to set up a grid's mechanical properties, define boundary conditions, and compute the corresponding displacements. We also use the result to demonstrate the physical meaning of Poisson's parameter, as well as illustrate the difference between the bulk modulus K and the vertical incompressibility K_v. We then consider the same cylinder in a poroelastic setting, which brings us to the well-known *Terzaghi problem* [18, 20], for which we can compare the numerical solution with the known analytic one. For this problem, we choose to use the fixed-stress split model `MechFluidFixedStressSplitModel` from the `ad-mechanics` module.

We then move on to model another well-known poroelastic problem, *Mandel's problem* [13, 20]. This problem demonstrates the *Mandel–Cryer effect*, a distinct poroelastic phenomenon that was first predicted theoretically by Jean Mandel and later confirmed by results of laboratory testing [19]. For this problem, we use the fully coupled `MechWaterModel` model.

14.5.1 Compression of a Dry Sample

Computing the deformation of a solid cylinder from a set of applied stresses is a linear elastic problem that we can solve directly using the MRST virtual element–based mechanics solver from the `vemmech` module. We will look at a simple uniaxial compression example and examine the impact of two different lateral boundary conditions: (i) "zero stress" and (ii) laterally constrained boundaries. The full MRST script for the problem developed in the present and following section is provided in the `ad-mechanics` module within the file `examples/terzaghi.m`.

To start off, we load the `vemmech` module and turn off gravity, because we have no interest in body forces in this example and the analytic solutions assume no gravity:

```
mrstModule add vemmech
gravity off
```

The first step is to define the grid representing the cylinder. To do this, we first define an approximately circular 2D grid using `triangleGrid` and `pebi` from

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

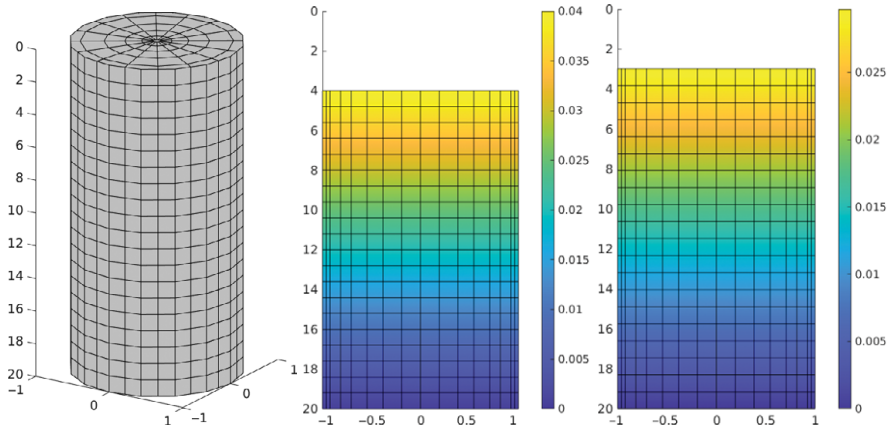

Figure 14.3 Left: undeformed cylinder; middle: cylinder deformed under zero stress lateral boundary conditions; right: cylinder deformed under "roller" boundary conditions.

MRST's core module and then extrude it vertically in 3D using `makeLayeredGrid` to produce the grid shown to the left in Figure 14.3:

```
P = [];
layers = 20;
for r = linspace(0.2, 1, 5)
    [x, y, ~] = cylinder(r^1.5, 16);
    P = [P [x(1,:); y(1,:)]];
end
P = unique([P'; 0, 0], 'rows');
aG = pebi(triangleGrid(P));
G = makeLayeredGrid(aG, layers);
G = computeGeometry(G);
G = createAugmentedGrid(G); % add geometric information for VEM mechanics
```

Existing functionality in the `vemmech` module makes it most convenient to use Young's modulus and Poisson's parameter to define elastic properties. We set these to be respectively 5 GPa and 0.3 in the following. We also set density to 2 000 kg/m^3, although this value will not influence the result unless we turn on gravity:

```
density = 2000 * kilogram / (meter^3);
Nc = G.cells.num;
E = 5 * giga * Pascal; % Young's modulus
nu = 0.3;  % Poisson's parameter
```

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

We proceed to define the boundary conditions. In general, this task can be a bit complicated because it requires us to identify the specific boundary nodes (for displacements) and faces (for forces) that constitute the different parts of the boundary. For the current problem this can be done using simple geometric considerations, but in the general case the task can quickly become more involved:

```
bottom_nodes = find(G.nodes.coords(:,3) == max(G.nodes.coords(:,3)));
top_nodes    = find(G.nodes.coords(:,3) == 0);
top_faces    = find(G.faces.centroids(:,3) == 0);
bottom_innermost = ...
    find(sqrt(sum(G.nodes.coords(bottom_nodes, 1:2).^2, 2)) < 0.1);
```

The last line in this code snippet is used to identify the single central node at the bottom of the cylinder. We will use this node to "anchor" the model, in the sense that the position of this node will be locked completely in place. In general, such anchoring is necessary to ensure unicity of the solution whenever the imposed boundary conditions are insufficient to fully prevent translations and rotations (rigid body motion) of the full model.

Boundary conditions for the linear elastic problem are specified by a structure by convention called `el_bc` in MRST. This struct has two fields, which themselves are structures: `disp_bc` and `force_bc`. As the names imply, these fields specify displacement and stress boundary conditions, respectively.

The following code defines the displacement boundary conditions, which consist of constraining the z-coordinate of bottom nodes. The nodes can still move in the x and y directions, except for the innermost node, which is locked completely in place. The targeted node indices are specified in the `nodes` field and the 3D displacements in the `uu` field. The `mask` field specifies for which of the three coordinate directions a particular constraint is active. In the code, only the innermost bottom node is constrained in the lateral directions:

```
% zero vertical displacement for bottom nodes (and zero displacement for
% innermost node to anchor the problem)
el_bc.disp_bc.nodes = bottom_nodes;
el_bc.disp_bc.uu = repmat([0, 0, 0], numel(bottom_nodes), 1);
el_bc.disp_bc.mask = repmat([false, false, true], numel(bottom_nodes), 1);
el_bc.disp_bc.mask(bottom_innermost, 1:2) = true;
```

At the top of the cylinder, we apply a downwards force $\sigma_o = 10$ MPa (megapascals). For this, we use the `force_bc` field, where we specify the faces and the 3D force vector applied to each of these faces. In this example, the force is normal to the faces and only has a z-component:

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

```
top_force = 1e7 * Pascal;   % force applied at top boundary
el_bc.force_bc.faces = top_faces;
el_bc.force_bc.force = repmat([0, 0, top_force], numel(top_faces), 1);
```

Finally, we specify the body force, which should be given as a function of the position `x`, which could be a single position or a set of `N` positions given as an `N x 3` array. Because we have turned gravity off, this function will not influence the result, but we specify it nevertheless for completeness:

```
load = @(x) repmat(density * gravity(), size(x, 1), 1);
```

We are now almost ready to solve the linear elastic system; i.e., compute the corresponding displacements. The solution is computed by the function `VEM_linElast`, which takes a grid `G`, the elasticity tensor `C` for each cell, the boundary conditions `el_bc`, and the load function `load`. The final step before calling `VEM_linElast` is thus to compute the full cell-wise elasticity tensor from our elastic parameters (Young's modulus `E` and Poisson's parameter `nu`), which is done using the function `Enu2C`. Nodal displacements are returned from `VEM_linElast` as a `(G.nodes.num x 3)` matrix of row-wise 3D displacement vectors:

```
% solve the linear elastic system
C = Enu2C(E * ones(Nc, 1), nu * ones(Nc, 1), G);
uu = VEM_linElast(G, C, el_bc, load);
```

The following code snippet plots the deformed grid (exaggerating displacements by a factor of 100) and shows it in profile view, using a color coding that indicates the Euclidean norm of the displacement, shown in the middle plot of Figure 14.3:

```
plotNodeDataDeformed(G, sqrt(sum(uu.^{2}, 2)), uu * 100);
```

From the plot, we see that the cylinder has not only shortened in length but also slightly widened. The ratio of widening to shortening per unit length of a material under uniaxial strain is exactly what Poisson's parameter expresses. To verify that our simulation honors this relation, we calculate the simulated ratio and compare it with the value used for Poisson's parameter in the simulation:

```
L = max(G.nodes.coords(:,3)) - min(G.nodes.coords(:,3)); % length
R = max(G.nodes.coords(:,1)); % radius - should equal 1
axial_strain  = uu(top_nodes(1), 3) / L;
radial_strain = max(uu(:,1)) / R;
```

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

```
measured_nu   = radial_strain / axial_strain;
relative_err  = (measured_nu - nu)/nu;
fprintf('Real nu: %1.5f.  Measured nu: %1.5f\n', nu, measured_nu);
fprintf('Relative error: %1.2e\n', relative_err);
```

```
Real nu: 0.30000.  Measured nu: 0.30007
Relative error: 2.20e-04
```

The difference between the theoretical and the simulated value is quite small (the small discrepancy is a result of our model only being approximately cylindrical).

We now repeat the experiment but with laterally constrained side boundaries. In other words, points on the side boundary are allowed to move up and down but not along the x or y axes. We identify the lateral boundary nodes by a geometric trick and redefine the displacement part of our boundary structure:

```
side_nodes = find(sqrt(sum(G.nodes.coords(:,1:2).^2, 2)) > 0.9);
side_nodes = setdiff(side_nodes, bottom_nodes);
[Nb, Ns]   = deal(numel(bottom_nodes), numel(side_nodes));

el_bc.disp_bc.nodes = [bottom_nodes; side_nodes];
el_bc.disp_bc.uu    = repmat([0, 0, 0], Nb + Ns, 1);
el_bc.disp_bc.mask  = [repmat([true, true, true], Nb, 1); ... % locked btm.
                       repmat([true, true, false], Ns, 1)]; % roller side bnd.
```

Note that we remove from the side boundaries the nodes that belong to both the bottom and side boundaries, to avoid defining these twice in `el_bc`. We rerun the solver and plot the result, which is shown to the right in Figure 14.3:

```
% solving the linear elastic system
uu = VEM_linElast(G, C, el_bc, load);

% plot result
plotNodeDataDeformed(G, sqrt(sum(uu.^2, 2)), uu * 100);
```

From the two rightmost plots in Figure 14.3, we see that the length of the cylinder has changed less in the laterally constrained case than in the case with free lateral boundaries. The deformation is one of zero lateral strain and constant lateral stress, which is described by the vertical incompressibility modulus introduced in Subsection 14.3.4 in (14.47). We end the first part of this exercise by verifying that the displacement is consistent with the one predicted by K_v:

```
K  = E / (3 * (1-2*nu)); % compute bulk modulus from E and nu
Kv = 3 * K * (1-nu)/(1+nu); % compute vertical incompressibility
axial_strain = uu(top_nodes(1), 3) / L;
predicted_strain = top_force / Kv;
```

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

```
relative_err = (axial_strain - predicted_strain)/predicted_strain;
fprintf('Predicted strain: %1.5f.  Measured strain: %1.5f\n', ...
        predicted_strain, axial_strain);
fprintf('Relative error: %1.2e\n', relative_err);
```

```
Predicted strain: 0.00149.  Measured strain: 0.00149
Relative error: -5.37e-14
```

This time, the simulation outcome matches the predicted value to a very high degree of precision.

14.5.2 Compression of a Wet Sample: The Terzaghi Problem

We now move into the poroelastic setting. We depart from the previous example[3] (the compression of a laterally constrained cylindrical sample) and make the following modifications:

- We change the cylinder from a solid elastic medium to a fluid-filled poroelastic medium.
- We initialize fluid pressure to zero inside the cylinder.
- We add no-flow boundary conditions to the bottom and lateral sides, while allowing fluid to flow across the top boundary by imposing a constant pressure of zero here.

This example, known as Terzaghi's problem, was formulated by Karl Terzaghi in his work leading up to his consolidation theory in the early part of last century [18] and was key to the development of his consolidation equation [20]. In the experiment, a constant force is applied to the top boundary, causing a compression of the medium and the internal fluid pressure to rise. As the fluid is gradually evacuated through the top boundary, fluid pressure slowly returns to its initial value. As such, the system goes through a gradual transition from fully undrained to fully drained conditions, which also implies a gradual change in the level of compression of the solid matrix.

The *loading efficiency* γ is a poroelastic parameter that describes the response in fluid pressure to the application of a vertical load on a poroelastic sample under undrained, laterally constrained conditions. Formally, it can be written as

$$\gamma = -\left(\frac{\delta p}{\delta \sigma_{zz}}\right)_{\epsilon_{xx}=\epsilon_{yy}=\zeta=0}. \tag{14.60}$$

[3] The code discussed herein is found in the same file in `ad-mechanics` as that of the previous section, namely, `examples/terzaghi.m`

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

It can be derived from other elastic parameters and expressed in several ways; for instance, in terms of the poroelastic stress parameter η from (14.44), the shear modulus G, and the uniaxial specific storage coefficient S from (14.43):

$$\gamma = \frac{\eta}{GS}. \tag{14.61}$$

In Terzaghi's problem, the initial rise in fluid pressure after applying the constant vertical force to the top boundary will therefore be $\delta p = \gamma \sigma_o$, where σ_o is the applied downward top force.

In (14.38), we introduced the drained and undrained bulk moduli, K and K_u. We also introduced the (drained) vertical incompressibility K_v in (14.47). To describe the initial response of the system studied in this example, we also need to define an *undrained* vertical incompressibility, which we denote $K_v^{(u)}$ and which can be computed from the shear modulus G and the undrained Poisson's parameter ν_u:

$$K_v^{(u)} = G\frac{2 - 2\nu_u}{1 - 2\nu_u}. \tag{14.62}$$

The knowledge of γ and $K_v^{(u)}$ enables us to predict the initial response of the (undrained) system; i.e., pore pressure and vertical strain immediately after the top force has been applied. Terzaghi showed that the evolution of the ensuing pore pressure is described by the one-dimensional homogeneous diffusion equation. When the system reaches equilibrium, the pore pressure will thus have returned to its initial condition and vertical strain can be computed using K_v, as per the previous example. We will use these relations to validate the following numerical simulation.

Because we are dealing with a poroelastic problem, we need to load `ad-mechanics`, along with a few other modules:

```
mrstModule add ad-mechanics ad-core ad-props ad-blackoil
```

We define basic poroelastic and flow properties of the system, including permeability, porosity, fluid (in)compressibility, the Biot–Willis coefficient, and the reference pressure for fluid density, which we set equal to the initial pressure (i.e., zero):

```
perm = 300 * milli * darcy;  % permeability
poro = 1/4;                  % porosity
pRef = 0;                    % zero reference pressure for fluid density
alpha = 0.9;                 % Biot-Willis coefficient
Kf = 1.96 * giga * Pascal;   % fluid incompressibility
```

The next step is to create the poroelastic model. Here, we choose to use the fixed-stress split `MechFluidFixedStressSplitModel`, but we could equally well

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

have used the fully coupled `MechWaterModel`. Setting up this model requires providing a `rock` structure that contains the values of permeability, porosity, and Biot–Willis parameter for all cells in the grid, as well as a fluid object that provides information on density, viscosity, fluid compressibility, and pore volume multiplier. Before defining our fluid, we compute the pore volume multiplier to be consistent with our choice of poroelastic parameters, as explained in Subsection 14.3.4 and (14.51).

```
rock   = struct('perm', perm * ones(Nc, 1), ...
                'poro', poro * ones(Nc, 1), ...
                'alpha', alpha * ones(Nc, 1));
pvMult = (1-alpha) * (alpha-poro) / poro / K; % pore volume multiplier,
fluid  = initSimpleADIFluid('phases', 'W', ...
                'mu'  , 1 * centi * poise, ...
                'rho' , 1000 * kilogram / meter^3, ...
                'c'   , 1 / Kf, ...
                'cR'  , pvMult, ...
                'pRef', pRef);
mech_problem = struct('E', E * ones(Nc, 1) , ...
                'nu'   , nu * ones(Nc, 1), ...
                'el_bc', el_bc, ...
                'load' , load);
model = MechFluidFixedStressSplitModel(G, rock, fluid, mech_problem);
```

Note that we reused the grid and boundary conditions defined in the last example.

Like other models using the object-oriented, automatic differentiation framework in MRST, simulation will be executed by a call to `simulateScheduleAD` [11]. To call this function, an initial state and a *schedule* must be provided. The schedule defines the timesteps and the (potentially time-dependent) driving forces that act on the system.

To define the initial state, we must define the initial nodal displacements and pore pressure values. These are all zero, so the only slightly nontrivial task is to determine the *number* of displacement values that must be provided. In general, each node of the 3D grid is associated with three degrees of freedom, except for boundary nodes, where fixed displacements may be imposed in one or more directions. The total number of degrees of freedom is therefore equal to the number of nodal displacements that are *not* imposed as Dirichlet boundary conditions, which is what we compute on the first line of the following code listing (remember also that in the previous example, we defined `Nc` to equal the number of cells in the grid):

```
num_mech_unknowns = sum(~model.mechModel.operators.isdirdofs);
initState = struct('pressure', pRef * ones(Nc, 1), ...
                   'xd', zeros(num_mech_unknowns, 1));
initState = addDerivedQuantities(model.mechModel, initState);
```

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

The call to `addDerivedQuantities` in the last line adds additional fields to `initState`, including `u` and `uu` (full lists of displacement values, also including the fixed displacement values associated with imposed boundary conditions), as well as the `vdiv`, `stress`, and `strain` fields, which provide cell-wise volumetric strains, stress tensors, and strain tensors.

To define a suitable schedule, we must decide over what time span we wish to simulate the behavior of the system; i.e., a time span within which most of the interesting behavior of the system response can be observed. From the analytical solution of the equation describing pressure evolution (i.e., the 1D homogeneous transport equation, not further discussed here), one can identify a characteristic time L^2/c, where $c = |\mathbf{K}|/(\mu S)$ is the *uniaxial hydraulic diffusivity* of the system. Note that here $\mathbf{K}$ represents the permeability of the porous medium (not the bulk modulus K), whereas μ is fluid viscosity and S the uniaxial specific storage coefficient defined in (14.43). By letting the dimensionless time $\tau = ct/L^2$ run from 0 to 1, we will capture most of the relevant response in our simulation.

To compute c, we need to compute S, which requires us to compute several other parameters in the code, as outlined in the following code excerpt:

```
Ks = K / (1-alpha);                          % grain compressibility
H  = K / alpha;
S_sigma = (1/K - 1/Ks) + poro*(1/Kf - 1/Ks); %specific storage, const. stress
R   = 1 / S_sigma;
B   = R / H;                                 % Skempton's coefficient
eta = (1 - 2*nu) / (2*(1 - nu))*alpha;       % poroelastic stress parameter
S   = S_sigma * (1 - 4*eta*B/3);             % uniaxial specific storage
c   = perm / (fluid.muW(pRef)*S);            % uniaxial hydraulic diffusivity
```

As pointed out in Section 14.3, it is generally time consuming to find a minimal sequence of formulas to evaluate a given parameter value from the ones that specify the problem. Using `poroParams`, we introduce a little computational overhead but save a lot of manual work. The following line eliminates the need of explicitly specifying the steps shown in the previous code listing:

```
params = poroParams(poro, true, 'E', E, 'nu', nu, 'alpha', alpha, 'K_f', Kf);
c   = perm / (fluid.muW(pRef)*params.S);
```

Once we have computed c, we define a time span running from 0 to L^2/c, divided into 50 equal timesteps plus an initial ministep that enables us to capture the immediate completely undrained response in our simulation. Following the standard in MRST, the simulation schedule is then defined as a structure holding the two substructures `control` and `step`. The first specifies the set of fluid driving forces that can be associated with specific time steps, whereas the latter specifies the timestep values and associates specific controls with these. In our case, we use a single

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

control for all timesteps, consisting of no-flow bottom and lateral boundaries and fixed-pressure boundary at the top. The full code listing for creating the schedule and running the simulation is the following:

```
nsteps = 50;
tsteps = [0, linspace(1e-5, 1, nsteps+1)]* L^2 / c;
schedule = struct('step', struct('val', diff(tsteps), ...
                                 'control', ones(nsteps+1, 1)), ...
                  'control', struct('W', [], 'bc', bc));
[~, states, report] = simulateScheduleAD(initState, model, schedule);
```

Upon completion of the simulation, we can graphically examine the development of the column pressure profile over time by picking and plotting selected timesteps. The result is shown in Figure 14.4, where we have scaled the x-axis by column length and the y-axis by the theoretical initial pressure response $p_{\text{init}} = \gamma\sigma_o$. The selected curves are labeled in terms of their corresponding dimensionless time τ. As is apparent from the figure, the initial pressure response is constant and equal to p_{init} across the whole column and then gradually dissipates over time. At $\tau = 1$, the pressure has not fully dissipated but reaches only approximately 10% of its initial value at the bottom of the column:

```
% compute max pressure (undrained response pressure)
% 'gamma' can be taken directly from the result of poroParams()
pmax = top_force * params.gamma;

% plot development of column pressure profile over time
z_cells  = [1:G.cells.num/layers:G.cells.num]';
z_depths = [0; G.cells.centroids(z_cells, 3)]/L;
ixs      = [0, 1, 2, 5, 20, 50] + 1;
figure; hold on
for ix = ixs
   plot(z_depths, [0; states{ix}.pressure(z_cells)/pmax]);
end
```

We also plot the vertical displacement of the top surface as a function of dimensionless time (Figure 14.5):

```
% plot vertical displacement of top surface over time
w_top = [0; cellfun(@(s) s.uu(top_nodes(1), 3), states)];
plot(tsteps * c / L^2, w_top * 100, 'linewidth', 1.5);
```

We see that there is an immediate response of approximately 1.6 cm after applying the force, after which the column keeps compressing further over time as fluid pressure drops. We compare the initial displacement observed in the simulation with the theoretical initial displacement for the undrained system based on $K_v^{(u)}$, which we obtain directly from the previous call to `poroParams`:

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

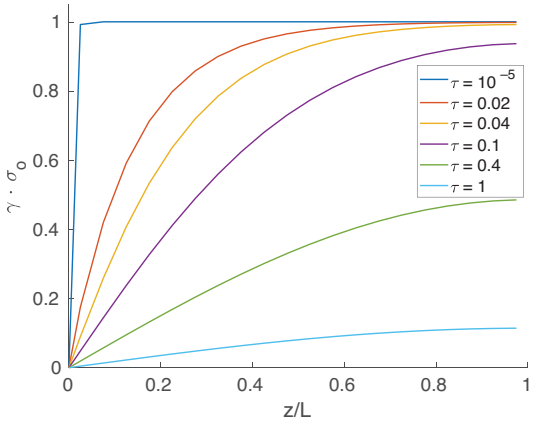

Figure 14.4 Development in the column pressure profile over time for our simulated Terzaghi's problem; $\tau = ct/L^2$ is the dimensionless time.

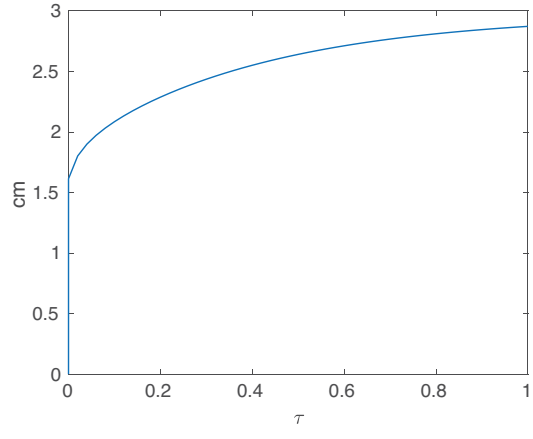

Figure 14.5 Vertical displacement of the top surface in our simulation of Terzaghi's problem as a function of dimensionless time τ.

```
w_0        = w_top(2);                    % observed initial displacement
w_0_theory = top_force * L / params.K_vu; % theoretical init. displacement

rel_err = (w_0 - w_0_theory)/w_0;
fprintf('Expected init. displacement: %1.5f m.\n', w_0_theory)
fprintf('Measured displacement: %1.5f m\n', w_0);
fprintf('Relative error: %1.2e\n', rel_err);
```

```
Expected init. displacement: 0.01614 m.
Measured displacement: 0.01614 m
Relative error: -1.93e-04
```

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

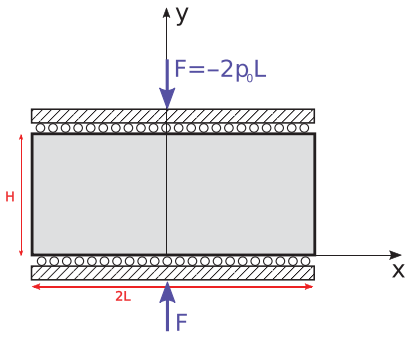

Figure 14.6 The geometry for Mandel's problem. Small circles are used to illustrate roller boundaries where movement is unrestrained in the x-direction but constrained to remain in contact with the rigid plates in the y-direction.

The agreement is reasonable. In the previous example, we also calculated the vertical strain of the drained system to be approximately $1.49 \cdot 10^{-3}$. By multiplying this by the column length $L = 20$ m, we compute the theoretical final displacement to be $2.98 \cdot 10^{-2}$ m, which seems about right from examining the asymptotic behavior of the curve in Figure 14.5.

14.5.3 Mandel's Problem

In the Terzaghi problem just explored, pore pressure is uncoupled from the (constant) vertical stress and satisfies the one-dimensional homogeneous diffusion equation. The pressure evolution can be determined completely independent of the mechanical deformation problem. *Mandel's problem*, on the other hand, is often cited as an example of a poromechanical effect exhibiting nonmonotonous pressure behavior that cannot be accounted for without considering the two-way coupling of fluid pressure and mechanical deformation.

The problem considers a rectangular slab of poroelastic material, interposed between two parallel, rigid, impermeable plates, as illustrated in Figure 14.6. The two sides of the slab in contact with the plates (top and bottom on the figure) are free to move laterally, whereas the left and right sides are fully free to move; i.e., zero-stress boundaries. The dimensions of the slab equal $2L$ in the x-direction (width) and H in the y-direction (thickness). It is infinitely long in the z-direction. Because everything remains constant along the z-direction, the problem reduces to a plane strain problem in the (x, y) plane. There are no body forces. Initially, fluid pressure and all forces are zero. At time $t = t_0$, a nonzero force F is suddenly applied normally to the rigid plates and then held constant. This leads to a gradual compression of the porous slab, as the pore fluid gradually evacuates through the side boundaries driven by the increased pore pressure. In this process, the fluid

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

pressure in the center of the slab does not uniformly decrease over time but first goes through a period where it *increases*. This nonmonotonic behavior is a uniquely poromechanical phenomenon referred to as the *Mandel–Cryer effect*, which we aim to demonstrate in the scripted example following next.

Mandel's problem is symmetric across the plane $x = 0$, which implies that the rigid plates remain parallel and aligned with the x-axis at all times. As a consequence, the pressure and stress fields are constant in the y coordinate and thus only depend on x and t. In particular, this means that fluid flow only occurs along the x direction and that $\sigma_{xx}(x, y) = 0$. However, the complete rigidity of the vertical plates in combination with the applied normal force and the roller boundary conditions lead to a particular complication when formulating the boundary conditions in a numerical code. For $t > t_0$, the normal force F is not uniformly applied along the x-direction but must be distributed to keep the interface flat. This distribution is a priori unknown and can therefore not be explicitly specified before simulation begins. However, we know that the sum of the integrated normal stress along the interface should equal $F = -2p_0L$, where p_0 is understood as the constant, "average" normal stress:

$$F = -2p_0L = \int_{-L}^{L} \sigma_{yy}(x, H, t)\, \mathrm{d}x. \tag{14.63}$$

In the example, we will show how we can setup the simulation to obey this condition, although it requires a somewhat unorthodox use of `simulateScheduleAD`. The full source code for the example can be found in the `ad-mechanics` module within the file `examples/mandel.m`.

We start by loading all of the required MRST modules and then define a standard Cartesian grid representing the poroelastic slab:

```
mrstModule add ad-mechanics ad-core ad-props ad-blackoil vemmech
gravity off

L = 10 * meter; H = 1 * meter;
G = computeGeometry( cartGrid([50, 10], [L, H]) );
```

Due to the plane symmetry, we only model the $x > 0$ part of the domain. The rigid plates do not form part of this grid, and their effect will be modeled by careful application of nonconstant boundary conditions, as explained further shortly.

We proceed by defining the `rock` and `mech_problem` objects, much as we did for Terzaghi's problem. However, we hold off the specification of mechanical boundary conditions (the `el_bc` structure) for now.

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

```
E          = 0.1*giga*Pascal;              % Young's modulus
nu         = 0.2;                          % Poisson's parameter
alpha      = 1;                            % Biot-Willis coefficient
top_press  = 1*mega*Pascal;
rock       = makeRock(G, 200*milli*darcy, 0.1);
rock.alpha = alpha * ones(G.cells.num, 1);
mech_problem.E    = E * ones(G.cells.num, 1);
mech_problem.nu   = nu * ones(G.cells.num, 1);
mech_problem.load = @(x) 0*x;  % no body force applied
```

In the same way, we set up the fluid parameters and construct a fluid object:

```
muW = 0.89 * milli * Pascal / second; % fluid viscosity
rhoW = 1000 * kilogram / meter^3;     % fluid density
cW = 1.0e-10 * Pascal^-1;             % fluid compressibility
pRef = 0;                             % reference pressure (zero)
fluid = initSimpleADIFluid('phases', 'W', 'mu', muW, 'rho', rhoW, ...
                           'c', cW, 'pRef', pRef);
```

We now proceed to define initial-boundary conditions. Most of these will remain constant throughout the simulation, with the exception of the displacement along the top ($y = H$) boundary, which will be dynamically modified during simulation. The first step is to identify all relevant boundary faces and nodes:

```
facenodes = ...   % lambda function to identify nodes for a set of faces
    @(f) unique(G.faces.nodes(mcolon(G.faces.nodePos(f), ...
                                     G.faces.nodePos(f+1)-1)));

% identify bottom faces and nodes
bfaces = find(G.faces.centroids(:,2) == min(G.faces.centroids(:,2)));
bnodes = facenodes(bfaces); % bottom nodes
nbn    = numel(bnodes); % number of bottom nodes (and top nodes)

% identify top faces and nodes
tfaces = find(G.faces.centroids(:,2) == max(G.faces.centroids(:,2)));
tnodes = facenodes(tfaces);

% identify left boundary faces and nodes
xmin_faces = find(G.faces.centroids(:,1) == 0);
lnodes     = facenodes(xmin_faces);
nln        = numel(lnodes); % number of nodes on left side

% identify right boundary faces (no need for nodes)
xmax_faces = find(G.faces.centroids(:,1) == max(G.faces.centroids(:, 1)));
```

The boundary conditions for flow are zero for the top ($y = H$) and bottom ($y = 0$) boundaries. Flow is also zero at the left ($x = 0$) boundary due to symmetry. Because no-flow is the default boundary condition for flow, we only have to spec-

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

ify the conditions at the right ($X = L$) side, where we impose a fixed pressure of zero:

```
bc = addBC([], xmax_faces, 'pressure', pRef);
```

As for the mechanics boundary conditions, we impose zero displacement along the x-axis for nodes on the left boundary and zero displacements along the y-axis for nodes on the bottom and top boundaries (the latter will be changed later):

```
% setup displacement boundary conditions (actual displacement value for top
% nodes to be determined later)
disp_bc = struct('nodes', [1:G.nodes.num]', ...
                 'uu',  zeros(size(G.nodes.coords)), ...
                 'mask', false(G.nodes.num, 2));
disp_bc.mask(tnodes, 2) = true;
disp_bc.mask(bnodes, 2) = true;
disp_bc.mask(lnodes, 1) = true;
```

Because some nodes belong to both the bottom and side boundaries, for which different restrictions are applied, we employ a slight trick to keep the code as simple as possible. We list *all* nodes (internal or not) in the `disp_bc` structure but only activate restrictions for the targeted boundary nodes and coordinate directions (the three last lines).

The next step is to define the initial state. To do this, we must identify the indices of the active degrees of freedom of the mechanical problem; i.e., node displacement that are *not* part of the imposed displacement boundary conditions. In the Terzaghi example, we did this by querying the `mechModel.operators.isdirdofs` field of the poroelastic model instance. To do the same here, we create a temporary instance of the fully coupled model, `MechWaterModel`, which will not be used for simulation in the following, because the mechanical boundary conditions still are not correct:

```
% initial state
mech_problem.el_bc = struct('disp_bc', disp_bc, 'force_bc', []);
model              = MechWaterModel(G, rock, fluid, mech_problem);
mech_unknowns      = ~model.mechModel.operators.isdirdofs;

initState.pressure = pRef * ones(G.cells.num, 1);
initState.xd       = zeros(nnz(mech_unknowns), 1);
initState          = addDerivedQuantities(model.mechModel, initState);
```

As for the Terzaghi problem, we complete the definition of the `initState` by a call to `addDerivedQuantities`.

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

We also compute a characteristic time that will be used when determining the total simulation time. This characteristic time is expressed as $T_{\text{char}} = L^2/c$ (squared length over uniaxial hydraulic diffusivity) and we compute it in a manner very similar to what we did in the Terzaghi example:

```
ppar = poroParams(poro, true, 'E', E, 'alpha', alpha, 'nu', nu, 'K_f', 1/cW);
c = perm / (muW * ppar.S);
Tchar = L^2 / c; % charateristic time
```

We now get to the central part of the scripting example: the actual simulation. For the Terzaghi problem and most other poroelastic problems encountered in practice, the boundary conditions could be set in advance, and a schedule could be defined that allowed us to simulate all timesteps of the problem by a single call to `simulateScheduleAD`. This is how simulations are generally intended to be defined and run in the object-oriented, automatic differentiation framework, but because we are here dealing with variable and a priori unknown boundary stresses, we cannot specify everything in advance. Instead, we will have to run the simulation one timestep at a time and search for the exact boundary conditions as we go along. This is done by using a simple iterative scheme that converges to the correct boundary condition. Instead of imposing a uniform *force* along the top boundary (which would be incorrect), we impose a uniform *displacement* and compute the corresponding force post hoc. As long as the resulting force (`tforce_sim * 2L`) differs significantly from the desired F (`top_press * 2L`), a corresponding adjustment is made to the imposed displacement and the timestep is run again until convergence. As we go along, the computed state for each timestep is collected in the `states` cell array. This iteration is implemented as follows:

```
for step = 1:tsteps
   while (true)
      disp_bc.uu(tnodes, 2) = dy_init;  % initial guess, vert. displ.

      % combine displacement and force boundary conditions
      mech_problem.el_bc = struct('disp_bc', disp_bc, 'force_bc', []);

      % model
      model = MechWaterModel(G, rock, fluid, mech_problem);

      % define an initial guess, to ensure the correct values for imposed
      % displacements are respected when the time step is simulated
      initGuessState = addDerivedQuantities(model.mechModel, state0);

      % simulate a single time step
      [~, state] = simulateScheduleAD(state0, model, schedule, ...
                                      'initialGuess', initGuessState);
```

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

```
        % comparing simulated mean stress with the target
        tstress_eff = state{1}.stress(tcells, 2);
        tpress      = state{1}.pressure(tcells);
        tforce_sim  = mean(-tstress_eff + alpha * tpress);
        if abs((top_press - tforce_sim)/top_press) < 1e-4
           % close enough to target, save results and proceed to next step
           break;
        else
           % adjust displacement boundary condition and try again
           dy_init = dy_init * top_press / tforce_sim;
        end
     end
     states = [states, state];
     state0 = state{1};
  end
```

A few additional comments to this listing:

- For the Terzaghi problem in the previous example, we used the fixed-stress split model `MechFluidFixedStressSplitModel`. Here, we use the fully coupled model `MechWaterModel`. The only reason we use different models for the two examples is to demonstrate the use of them both. They produce practically the same result.
- Each time boundary conditions are changed, a new instance of `MechWaterModel` has to be generated. This is because unlike the flow boundary conditions, which are specified in the `schedule` structure, mechanical boundary conditions form an integral part of the model itself. Because there is currently no established way of changing mechanical boundary conditions inside an already created model, which would entail reassembling all of the associated discrete operators, we have to create a new model each time.
- The `state` structure computed by `simulateScheduleAD` for a given timestep contains all of the updated degrees of freedom of the problem (i.e., free node displacements and pressure values). The values of the imposed displacements (Dirichlet boundary conditions) are not recomputed but are inherited from the previous timestep. This is usually what one wants, but not in our case, where the imposed displacements change from iteration to iteration. However, if we provide the function with an *initial guess*, imposed displacements will be inherited from this initial guess instead. This is why we define an `initGuessState` at each iteration in the loop, which is computed from `addDerivedQuantities` applied on the updated `model.mechModel`. It is not particularly elegant, but it gets the work done.
- An ad-hoc, one-timestep schedule is created in advance and passed to each call to `simulateScheduleAD`. It specifies a single-timestep simulation and provides the correct timestep size.

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

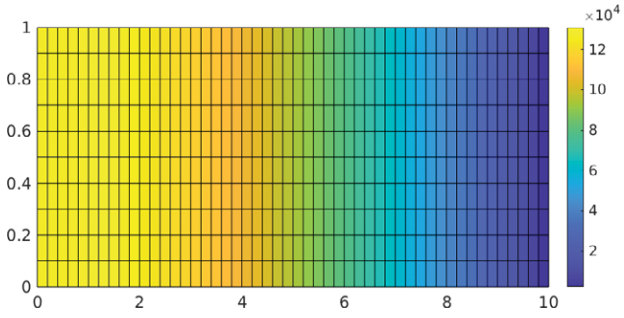

Figure 14.7 Pressure profile inside the poroelastic slab after simulating Mandel's problem for a period of one characteristic time.

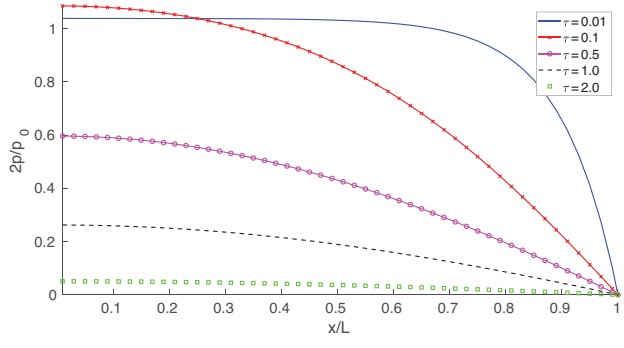

Figure 14.8 Pressure profiles along the x-axis of the slab in Mandel's problem for different multipliers of the characteristic time τ. The center of the slab is located at $x = 0$ and the boundary is located at $x = L$.

The code just outlined uses `simulateScheduleAD` in a rather unconventional way. It should be emphasized that in situations where boundary conditions are specified in advance, typically the case for most problems, you should use the approach shown in the code listing of the Terzaghi problem, where a full schedule is defined for the whole simulation and `simulateScheduleAD` is called exactly once.

With the parameters given in the code listing, the characteristic time is 4 063 seconds or approximately 1 hour and 8 minutes. We have simulated a total of two times this interval, using 200 timesteps. We can visualize the pressure inside the slab half-way through the simulation using `plotCellData`; see Figure 14.7. From the plot, we can visually verify that the pressure field does not depend on the y coordinate. To get a sense of the pressure development over time, we plot the pressure of the top cells (which is equal to the pressure along any horizontal line inside the slab) for various multipliers of the characteristic time.

https://doi.org/10.1017/9781009019781.020 Published online by Cambridge University Press

The outcome, with characteristic time τ ranging from 0.01 to 2.0, is displayed in Figure 14.8. For the very early time ($\tau = 0.01$), pore pressure is more or less constant in the central part of the slab and drops to zero toward the boundary. For $\tau = 0.1$, the pressure drop is more gradual, but the pressure at the center has actually *increased*. The reason for this temporary pressure increase can be intuitively understood as the effect of the fluid in the central part of the slab having to carry more of the applied load as the peripheral part of the slab gradually drains. In a later phase ($\tau \geq 0.5$), the pressure drops monotonously in time throughout the slab. This happens when the effect of the drainage process on pore pressure becomes sufficiently important also in the central part, shifting more of the load to the solid matrix.

14.6 Concluding Remarks

In this chapter we have provided a brief overview of linear elasticity and poroelasticity theory and introduced a number of poroelastic moduli and coefficients. We also discussed the practical solution of the equations involved in MRST. A VEM-based linear elasticity solver is given by the `vemmech` module, and the `ad-mechanics` module provides a model for time-dependent poroelasticity problems, which also generalizes to a multiphase setting. The practical examples discussed herein are well known from poroelastic theory. Historically, Terzaghi's problem was studied in the context of understanding soil consolidation, whereas the Mandel problem demonstrates a phenomenon that can only be understood in light of the two-way coupling between mechanics and fluid flow. Such couplings have often been ignored or simplified in the past, but new focus on the impact of geomechanics in several disciplines and improved high-performance computing capabilities have made the modeling of such coupled problems increasingly relevant and practical.

The focus of this chapter (and indeed the whole book) has been on subsurface applications, but poromechanical modeling is also highly relevant in many other application domains. The motivated reader might therefore be interested in employing the functionality of the modules discussed in this chapter to other applications and settings for which the hypotheses of linear poroelasticity apply. This type of "alternative" use should be straightforward to implement, because MRST to a large extent is developed with generic functionality and interfaces in mind, despite the "R" in its acronym.

Acknowledgement. The author acknowledges Halvor Møll Nilsen and Xavier Raynaud as primary architects of the MRST software modules presented in this chapter.

References

[1] M. Beck, A. P. Rinaldi, B. Flemisch, and H. Class. Accuracy of fully coupled and sequential approaches for modeling hydro- and geomechanical processes. *Computational Geosciences*, 24:1707–1723, 2020. doi: 10.1007/s10596-020-09987-w.

[2] L. Beirão da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L. D. Marini, and A. Russo. Basic principles of virtual element methods. *Mathematical Models and Methods in Applied Sciences*, 23(1):199–214, 2013. doi: 10.1142/S0218202512500492.

[3] L. Beirão da Veiga, F. Brezzi, L. Marini, and A. Russo. The hitchhiker's guide to the virtual element method. *Mathematical Models and Methods in Applied Sciences*, 24(8):1541–1573, 2014. doi: 10.1142/S021820251440003X.

[4] M. A. Biot. General theory of three-dimensional consolidation. *Journal of Applied Physics*, 12(2):155–164, 1941. doi: 10.1063/1.1712886.

[5] S. Chen, R. Huang, and K. Ravi-Chandar. Linear and nonlinear poroelastic analysis of swelling and drying behavior of gelatin-based hydrogels. *International Journal of Solids and Structures*, 195:43–56, 2020. doi: 10.1016/j.ijsolstr.2020.03.017.

[6] L. B. da Veiga, K. Lipnikov, and G. Manzini. *The Mimetic Finite Difference Method for Elliptic Problems*. Volume 11 of *MS&A – Modeling, Simulation and Applications*. Springer International Publishing, 2014. doi:10.1007/978-3-319-02663-3.

[7] A. L. Gain, C. Talischi, and G. H. Paulino. On the Virtual Element Method for three-dimensional linear elasticity problems on arbitrary polyhedral meshes. *Computer Methods in Applied Mechanics and Engineering*, 282:132–160, 2014. doi: 10.1016/j.cma.2014.05.005.

[8] A. Ghassemi. A review of some rock mechanics issues in geothermal reservoir development. *Geotechnical and Geological Engineering*, 30(3):647–664, 2012. doi: 10.1007/s10706-012-9508-3.

[9] J. Kim. Sequential methods for coupled geomechanics and multiphase flow. PhD thesis, Stanford University, 2010. URL `pangea.stanford.edu/ERE/pdf/pereports/PhD/Kim10.pdf`

[10] A. E. Kolesov, P. N. Vabishchevich, and M. V. Vasilyeva. Splitting schemes for poroelasticity and thermoelasticity problems. *Computers & Mathematics with Applications*, 67(12):2185–2198, 2014. doi: 10.1016/j.camwa.2014.02.005.

[11] 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.

[12] A. Malandrino and E. Moeendarbary. Poroelasticity of living tissues. In R. Narayan, ed., *Encyclopedia of Biomedical Engineering*, pp. 238–245. Elsevier, Oxford, UK, 2019. doi: 10.1016/B978-0-12-801238-3.99932-X.

[13] J. Mandel. Consolidation des sols (étude mathématique) [Soil consolidation (a mathematical study)]. *Geotechnique*, 3(7):287–299, 1953. doi: 10.1680/geot.1953.3.7.287.

[14] A. Mikelić and M. F. Wheeler. Convergence of iterative coupling for coupled flow and geomechanics. *Computational Geosciences*, 17(3):455–461, 2013. doi: 10.1007/s10596-012-9318-y.

[15] J. N. Reddy. *An Introduction to Continuum Mechanics*, 2nd ed. Cambridge University Press, 2013. doi: 10.1017/CBO9781139178952.

[16] J. Rutqvist. Status of the TOUGH-FLAC simulator and recent applications related to coupled fluid flow and crustal deformations. *Computers & Geosciences*, 37(6):739–750, 2011. doi: 10.1016/j.cageo.2010.08.006.

[17] J. Rutqvist. The geomechanics of CO_2 storage in deep sedimentary formations. *Geotechnical and Geological Engineering*, 30(3):525–551, 2012. doi: 10.1007/s10706-011-9491-0.

[18] K. Terzaghi. *Erdbaumechanik auf bodenphysikalischer Grundlage* [The Mechanics of Earth Construction Based on Soil Physics]. F. Deuticke, Leipzig, Germany, 1925.
[19] A. Verruijt. *Theory and Problems of Poroelasticity*. Delft University of Technology, 2016. URL geo.verruijt.net/software/PoroElasticity2016b.pdf.
[20] H. F. Wang. *Theory of Linear Poroelasticity with Applications to Geomechanics and Hydrogeology*, Volume 2 of *Princeton Series in Geophysics*. Princeton University Press, Princeton, NJ, 2000.