12

Simulation of Geothermal Systems Using MRST

MARINE COLLIGNON, ØYSTEIN S. KLEMETSDAL, AND OLAV MØYNER

Abstract

Geothermal energy is renewable, always on, and available anywhere (at least in principle). Hot underground aquifers are therefore appealing as a source of green energy but also for large-scale energy storage, which is important to buffer the seasonal energetic imbalance associated with the use of renewable energies. The viability of a geothermal exploitation project is determined by a number of factors such as energy efficiency, storage capacity, economical aspects (e.g., drilling and operational costs), and compliance with legal regulations. Such assessments require a detailed characterization of the geology and physical properties of both the aquifer and aquiclude, groundwater chemistry, and flow properties. Proper understanding of these processes depends on accurate and flexible numerical simulation tools. In this chapter, we present `geothermal`, a module for geothermal simulations of low- to moderate-enthalpy geothermal systems. The module implements the equations for conservation of energy and conservation of mass for water and salt (NaCl), along with pressure-, temperature- and NaCl-dependent viscosity and density and other functionalities specific to geothermal problems. We demonstrate the accuracy of the module by benchmarking it with TOUGH2, a widely used groundwater flow simulator. We also show how `geothermal` can be used to simulate different geothermal applications.

12.1 Introduction

A rapid and large-scale transition from a mostly fossil fuel–based to a renewable energy supply is critical to mitigate the effects of climate change while meeting the world's increasing energy demand [6, 9, 14, 28]. Conventional sources of

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

renewable energy (e.g., wind, solar, and hydraulic) typically have a strong temporal and spatial variability, which makes them challenging to implement in a reliable, large-scale setting. Moreover, these systems often require large and invasive infrastructures. In contrast, geothermal energy constitutes a resource that shows little variability in time, is available anywhere on the planet, and requires limited infrastructure [14, 28]. Geothermal energy commonly refers to the production of new energy but also includes storage strategies in the subsurface [1, 3]. Among these, aquifer thermal energy storage (ATES) is a cost-effective and large-scale storage facility system in which the excess energy from multiple temporal resources is stored in an aquifer for later use in periods of demand [2, 10].

The viability of a geothermal energy system, either for production or for storage, depends on its economic gain (i.e., energy efficiency or storage capacity versus operational and drilling costs) and compliance with legal regulations [14, 28]. Evaluating the viability of a geothermal system requires a solid knowledge of the groundwater flow characteristics and reservoir properties [2], which can only be achieved through numerical simulations [18, 21, 28]. There are a number of software packages for simulation of geothermal systems; e.g., COMSOL, ANSYS FLUENT, UTMECH, SEAWAT (MODFLOW family), FEFLOW, and TOUGH2 [8, 11, 16–18, 25]. They all provide a high degree of complexity in terms of physical properties (e.g., equations of state (EoSs), compositional behavior, chemical reactions, etc.) but tend to lack support for complex grid geometries and realistic well modeling (including injection/production strategies and operational limits), as well as efficient linear and nonlinear solvers (e.g., HYDROTHERM [16]). On the other hand, software developed for the oil and gas industry usually offers flexible and complex gridding capabilities, well modeling, and efficient solvers but tends to limit the physical effects to those needed for simulation of hydrocarbon recovery. This is particularly true for thermal effects, because most petroleum reservoirs are almost thermally inert.

In this chapter, we present `geothermal`, a module for geothermal simulations in the MATLAB Reservoir Simulation Toolbox (MRST) based on the general, object-oriented, simulator framework. This framework offers an industry-standard finite-volume discretization with single-point upstream evaluation and two-point flux approximation and gives access to powerful solvers widely used by the reservoir simulation community. Moreover, one can easily integrate the `geothermal` module with well-established functionality from petroleum applications that are already part of MRST and apply the resulting simulators to realistic geological models and complex fluid physics. We demonstrate the applicability of `geothermal` on a number of cases, ranging from simple conceptual and validity tests to a complex high-temperature ATES system.

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

12.2 Governing Equations for Geothermal Applications

The governing equations for thermal field-scale flow in porous media describe two fundamental physical properties: conservation of mass and conservation of energy.

Mass conservation: Conservation of mass can be written as

$$\partial_t \mathcal{M}_f + \nabla \cdot \vec{\mathcal{F}}_f = \mathcal{Q}_f, \quad \text{where} \quad \mathcal{M}_f = \phi \rho_f, \quad \vec{\mathcal{F}}_f = \rho_f \vec{v}, \quad \mathcal{Q}_f = q_f \rho_f. \tag{12.1}$$

Here, $\mathcal{M}_f$, $\vec{\mathcal{F}}_f$, and $\mathcal{Q}_f$ denote fluid mass, mass flux, and sources/sinks, respectively. In particular, ϕ is the rock porosity, ρ_f is the fluid density, and q_f represents volumetric source and sink terms (e.g., wells). Darcy's law gives the velocity $\vec{v}$,

$$\vec{v} = -\frac{\mathbf{K}}{\mu_f}(\nabla p - \rho_f g \nabla z), \tag{12.2}$$

where $\mathbf{K}$ is the permeability tensor, μ_f is fluid viscosity, z is depth, and g denotes gravity acceleration.

Energy conservation: The equation for conservation of energy reads

$$\underbrace{\partial_t\left(\mathcal{M}_f u_f + \mathcal{M}_r u_r\right)}_{\text{accumulation}} + \underbrace{\nabla \cdot (\vec{\mathcal{F}}_f h_f)}_{\text{advection}} + \underbrace{\nabla \cdot (\vec{\mathcal{H}}_f + \vec{\mathcal{H}}_r)}_{\text{conduction}} = \underbrace{\mathcal{Q}_f h_f}_{\text{source}}. \tag{12.3}$$

Equations (12.1) and (12.2) are well described in other work on MRST like the textbook by Lie [19]. The equation for conservation of energy is also described briefly in [19, section 7.5] but may not be familiar to many readers and therefore merits some discussion. Equation (12.3) states that the rate of change of the thermal energy in a given volume, plus the energy transported in/out of the volume due to fluid flow, plus energy diffusion due to temperature differences should equal the energy injected into or extracted from the same volume. Because this is a bookkeeping statement about the change in thermal energy (SI unit Joule [J]) per time (SI unit second [s]), the expression is given in units of Joules per second, or watts (W).

The first term in the temporal derivative of (12.3) represents accumulation of energy in the pores of the rock, which we assume are completely occupied by the fluid. The thermal energy in a given volume of pore space is then equal to the mass $\mathcal{M}_f$ of the fluid occupying the pore space, multiplied by the energy density u_f (per unit of fluid mass), commonly referred to as internal energy. This is typically modeled as $u_f = C_f T$, where C_f is the specific heat capacity of the fluid and T denotes temperature. Simple dimensional analysis tells us that C_f must be given in

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

units of Joule per kilogram-Kelvin. Likewise, the second term models accumulation of energy in the solid parts of the rock. The solid rock mass equals

$$\mathcal{M}_r = (1 - \phi)\rho_r,$$

where ρ_r is the solid rock density. As for the fluid part, u_r denotes the energy density per unit mass of rock.

The second term in (12.3) accounts for energy transport due to advection. Here, $h_f = u_f + p/\rho_f$ denotes the enthalpy density per mass. Moreover, heat conduction is modeled by Fourier's law,

$$\vec{\mathcal{H}}_r = -\lambda_r \nabla T \quad \text{and} \quad \vec{\mathcal{H}}_f = -\lambda_f \nabla T, \tag{12.4}$$

where λ_r and λ_f denote the thermal conductivity of the fluid and rock, respectively. Again, we can use dimensional analysis to find that the thermal conductivity λ has dimension watts per meter-Kelvin. Finally, $\mathcal{Q}_f h_f$ models energy sources and sinks due to injection/extraction of fluids.

Consider a sample of solid rock ($\phi = 0$) with uniform and constant density and conductivity. If we insert Fourier's law into (12.3), we get the familiar parabolic heat equation $\partial_t T = \alpha \Delta T$, where $\alpha = \frac{\lambda_r}{\rho_r C_r}$ is the thermal diffusivity.

Notice the similarity between Fourier's law (12.4) and Darcy's law (12.2) and the analogy between thermal conductivity and permeability. Whereas temperature gradient is the only driving force for the heat flow in Fourier's law, the fluid flow in Darcy's law is driven by both the pressure gradient and gravity effects. The module currently neglects the pressure–volume work done by the heat on the fluid. This may, however, be significant in vapor-dominated systems and will be accounted for in future work.

Conservation of salts: Aquifer water is almost always rich with salts that dissolve into the water to form a brine. Conservation of a salt component c can be written as

$$\partial_t(\mathcal{M}_f X_c) + \nabla \cdot (\vec{\mathcal{F}}_f X_c) + \nabla \cdot \vec{\mathcal{D}}_c = \mathcal{Q}_f X_c, \quad \text{where} \quad \vec{\mathcal{D}}_c = -\rho_f \phi \tau D_c \nabla X_c. \tag{12.5}$$

Here, $\vec{\mathcal{D}}_c$ models diffusion of component c due to concentration differences, where X_c denotes the component mass fraction, τ is the tortuosity of the medium, and D_c is the component molecular diffusivity. For simplicity, we assume that the concentration of any salt component is always less than the saturation concentration

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

for brine so that we can safely neglect salt precipitation. The component number c runs from 1 to the number of components m. We also introduce a water component, which gives us the closure relation $\sum_c X_c = 1$.

Sodium chloride (NaCl) is the most commonly occurring dissolved salt, and its concentration may have a significant effect on the brine thermodynamic properties such as density, viscosity, enthalpy, and internal energy. In this work, we consider a two-component water–NaCl model to account for salt effects. Fluid density ρ_f and viscosity μ_f can be related to pressure, temperature, and NaCl mass fraction by a mathematical formulation, itself derived from an EoS. A large body of research has been devoted to developing reliable EOSs based on experimental data for both pure water [15, 29, 30] and NaCl–H_2O systems (e.g., [12, 13, 20, 22–24, 26]). Most of these formulations are accurate over a given pressure, temperature, or salinity range and may therefore be limited to specific applications. Here, we consider low- to moderate-enthalpy geothermal systems with a liquid brine and no phase transition. We use the formulations by Spivey et al. [27] for both density and viscosity of a brine. These formulations are valid for temperatures from 0°C to 275°C, pressures up to 200 MPa, and salinity below the solubility limit, which is usually sufficient to investigate the first 5 km of most sedimentary basins. Note that the module in principle supports any number of salt components, but this requires the user to provide a suitable EoS. In addition to density and viscosity, specific heat capacities C_f and C_r and thermal conductivities λ_f and λ_r will generally depend on pressure, temperature, and NaCl mass fraction. Our implementation supports such effects, but we omit the details here.

Discrete equations: As well described in the MRST textbook [19], the standard discretization approach in reservoir simulation for (12.1)–(12.2) is to use implicit backward Euler discretization in time. Applying the same method to (12.3) gives the following equation, written in residual form:

$$\begin{aligned}\mathcal{R}^{n+1} = \frac{1}{\Delta t^n}\left([\mathcal{M}_f u_f + \mathcal{M}_r u_r]^{n+1} - [\mathcal{M}_f u_f + \mathcal{M}_r u_r]^n\right) \\ + \nabla \cdot [\vec{\mathcal{F}}_f h_f]^{n+1} + \nabla \cdot [\vec{\mathcal{H}}_f + \vec{\mathcal{H}}_r]^{n+1} - [\mathcal{Q}_f h_f]^{n+1} = 0. \qquad (12.6)\end{aligned}$$

Here, superscript n refers to time t^n, and $\Delta t^n = t^{n+1} - t^n$ denotes the timestep. To obtain a fully discrete formulation of the equations, we use a finite-volume discretization with single-point upwind weighting, in which both the mass flux (12.2) and the heat flux (12.4) are discretized with a two-point flux approximation. The result is a robust discretization that is stable over a wide range of timesteps. Equation (12.5) for conservation of NaCl mass is discretized in an equivalent manner, with a two-point discretization of the second-order diffusion term.

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

12.3 The Geothermal Module

The `geothermal` module is built using the object-oriented, automatic differentiation (AD-OO) simulator framework in MRST, which is discussed in detail in chapters 11 and 12 of the MRST textbook [19]. In its basic form, the AD-OO framework offers a general setup for fully implicit simulators, and if we for a moment disregard wells, source terms, and boundary conditions other than no flow, all you have to do to make a new simulator is to create a simulator class that defines the necessary physical variables and implements the governing equations.

The framework has many existing simulator classes we can use as a template for implementing the flow equations. Using the discrete differentiation operators offered by the framework, we can write the fully discrete conservation equations for energy in a form very similar to (12.6):

```
eq = ((Mf.*uf + Mr.*ur) - (Mf0.*uf0 + Mr0.*ur0))/dt ...
                        + op.Div(Ff.*hf) + op.Div(Hf + Hr) - Qf.*hf
```

To compute the accumulation, flux, and source terms, we use so-called state functions, which are a recent addition to MRST that modularizes the implementation of physical models and introduces a compute-cache mechanism that avoids redundant function evaluations; see Chapter 5 for details. More about this in Subsection 12.3.2.

Evaluating the fully discrete conservation equations gives a system of nonlinear equations, which by default is solved using Newton's method in the AD-OO framework. This involves repeated computation of the Jacobian of the residual equations. In MRST, this is done by means of *automatic differentiation*, which automatically computes exact numerical values for all derivatives with respect to a prescribed set of variables when evaluating the discrete equations. This means that you entirely avoid the cumbersome and error-prone process of analytically deriving and implementing Jacobians of complex functional expressions. You can find more details on automatic differentiation in sections 4.4 and 7.2 of the MRST textbook [19]. The module is also compatible with the faster backends for automatic differentiation described in Chapter 6.

12.3.1 A Simple Worked Example

Before going into more details about the actual implementation, we present key components of the `geothermal` module by means of an example. The setup consists of a 2D vertical 100×50 m^2 domain, into which we inject warm water through

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

one side. This yields a propagating thermal front that progressively tilts due to temperature-induced density differences. You can find the example source code in the `geothermalExample2D` script of the module.

First, we load the `geothermal` module, along with the modules it depends on:

```
mrstModule add geothermal ad-core ad-props ad-blackoil compositional
```

and then use a uniform Cartesian grid to discretize the domain:

```
physdim = [100 50]; % Domain size in x, y directions
celldim = [50 50];  % Number of cells in x, y directions
G = computeGeometry(cartGrid(celldim, physdim)); % Cartesian grid
```

Fluid model: The single-phase fluid model is constructed in two steps by first using a standard routine to initialize a single-phase fluid object and then adding thermal properties (heat capacity and thermal conductivity), as well as an EoS [27] that overwrites the default (constant) pressure–volume–temperature (PVT) properties:

```
fluid = initSimpleADIFluid('phases', 'W', 'mu', 1, 'rho', 1); % Fluid structure
fluid = addThermalFluidProps(fluid    ,        ... % Original fluid structure
                             'Cp'     , 4.2e3, ... % Specific heat capacity
                             'lambdaF', 0.6  , ... % Thermal conductivity
                             'useEOS' , true );    % Use equation of state
```

Figure 12.1 shows how density and viscosity depend on pressure and temperature in this EoS model. Alternatively, you can use a simpler expression of the form

$$\rho_f = \rho_f^S \exp\left(c_p[p - p^S] + c_T[T - T^S] + c_X X_c\right), \tag{12.7}$$

where c_p, c_T, and c_X are the factors for compressibility, thermal expansion, and salinity, whereas ρ_W^S, p^S, and T^S are the density, pressure, and temperature at surface (or any other reference) conditions. Thermal expansion and salinity factor are provided to the fluid structure using the keywords `'cT'` and `'cX'` and the reference temperature by `'TRef'`. Reference density, pressure, and compressibility c_p must be provided when calling `initSimpleADIFluid` (see the function documentation). A final option is to apply your own user-defined density/viscosity functions to, e.g., interpolate tabulated data, by introducing appropriate functions as function handles in the fluid structure; e.g., `fluid.rhoW = @(p,T)rhoW(p,T)`. Note that this is also true for the specific heat capacity, `'Cp'`.

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

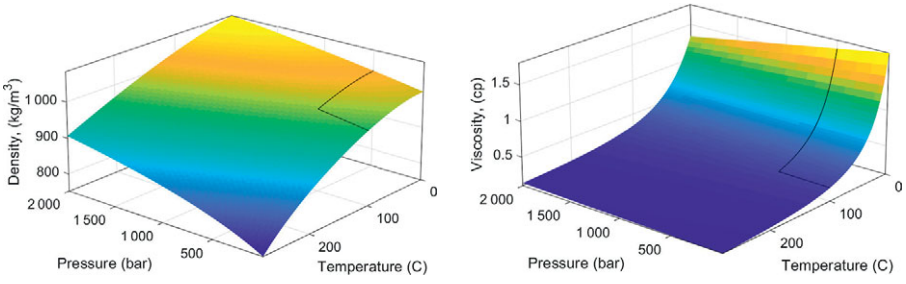

Figure 12.1 Density (left) and viscosity (right) as functions of pressure and temperature from the standard EoS [27] in the `geothermal` module. Standard operational conditions are delimited by black lines.

Rock properties: We follow the same approach as for the fluid properties by first imposing homogeneous permeability and porosity and then subsequently adding thermal properties to the `rock` object:

```
rock = makeRock(G, 1e-14, 0.1);                % Rock structure
rock = addThermalRockProps(rock     ,       ... % Original rock structure
                           'CpR'    , 1000, ... % Specific heat capacity
                           'lambdaR', 2   , ... % Thermal conductivity
                           'rhoR'   , 2700),    % Rock density
```

The rock properties can be given to the function either as a scalar variable, as an array with one entry per grid cell, or using function handles.

Conductivities: The conductivities λ_r and λ_f are assumed to depend on pressure and temperature and are given as function handles:

```
fluid.lambdaF = @(p,T) lambdaF(p,T); % Fluid thermal conductivity
rock.lambdaR  = @(p,T) lambdaR(p,T); % Rock thermal conductivity
```

At each nonlinear iteration, the class `DynamicTransmissibility` computes heat transmissibilities for the fluid and rock using a two-point expression. This class can also be used to introduce pressure- and temperature-dependent permeability but currently only supports isotropic tensors. If the user provides a constant to `'lambdaF'` or `'lambdaR'`, the conductivities are treated as constant.

Simulation model and initial state: We now have all of the parts necessary to construct a simulator. In the AD-OO framework, this is done by instantiating a *model class*. The model class in `geothermal` is called `GeothermalModel` and by default implements single-component flow modeled by (12.1) and (12.3):

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

```
gravity reset on, gravity([0 -9.81]);    % Gravity along -y
model = GeothermalModel(G, rock, fluid); % Single-phase geothermal model
```

We also need to set up the initial state, with constant pressure and temperature. This is done by first calling a standard initialization routine and then expanding the resulting state object with an extra field `T` for temperature:

```
state0   = initResSol(G, 100*barsa, 1);      % Pressure and saturation
state0.T = ones(G.cells.num,1).*(273.15+10); % Temperature
```

Drive mechanisms: To drive flow within a model, we must also prescribe forcing terms in the form of wells, boundary conditions, or source terms. In this example, we model inflow of warm water at the left boundary by imposing fixed pressure and temperature boundary conditions with higher values than those in the reservoir. Outflow at the right boundary is modeled by imposing fixed pressure and temperature boundary conditions but with values equal to the initial reservoir conditions:

```
bc  = pside([], G, 'left' , pInj, 'sat', 1); nf = numel(bc.face); % Injection
bc  = pside(bc, G, 'right', pRes, 'sat', 1);                      % Production
Tbc = repmat(Tres, numel(bc.face), 1); Tbc(1:nf) = Tinj;          % Temperature
bc  = addThermalBCProps(bc, 'T', Tbc);
```

Fixed temperature can be imposed at a boundary closed to flow by first imposing a flux boundary condition with zero value and then prescribing the temperature. For example, a heat flux can be imposed at the base of a model to represent conductive heat from a far-field magmatic source without any flow across that boundary. It is also possible to prescribe heat flux by using the keyword argument `'Hflux'` instead of `'T'` in `addThermalBCProps`. Each boundary face must have either a prescribed temperature *or* a prescribed heat flux.

Running the simulation: The only remaining part of the setup is to prescribe a series of timesteps with associated controls on drive mechanisms, which together are referred to as a `schedule` [19, section 11]. Here, we define timesteps that gradually increase up to 30 days, defining a total simulation time of one year:

```
timesteps = rampupTimesteps(1*year, 30*day, 8);       % Define time steps
schedule  = simpleSchedule(timesteps, 'bc', bc);      % Simulation schedule
```

Finally, we simulate with the standard simulator function from AD-OO:

```
[~, states] = simulateScheduleAD(state0, model, schedule);
```

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

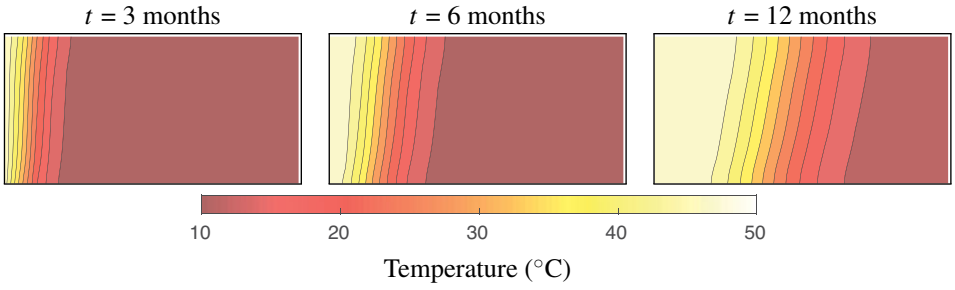

Figure 12.2 Evolution of the reservoir temperature under constant injection of warm water from the left boundary. The right boundary has a fixed temperature and pressure equivalent to the initial reservoir conditions, allowing a flow out of the reservoir. The top and bottom boundaries are thermally insulated, with no-flow conditions. Notice how the thermal front progressively tilts while propagating toward the right side of the reservoir.

You can now use the standard plotting functions in MRST to inspect the results; e.g., the graphical user interface `plotToolbar` for interactive plotting. Figure 12.2 shows the evolution of the temperature distribution in the reservoir during injection. Notice how the propagating warm water front tilts due to temperature-induced density differences.

12.3.2 Utility and State Functions

The `geothermal` module offers a set of model classes that are derived from the `ReservoirModel` class in the `ad-core` module. This generic class does not implement any flow equations per se and can therefore not be used directly for simulation but has properties to represent entities found in most reservoir models (grids, fluid model, petrophysical properties, phase indicators, well/facility models, etc.) as well as the basic numerical machinery necessary to run simulations (discretization and averaging operators, nonlinear solvers and linearization mechanisms, linear solvers, timestep controls, etc.), which are inherited from the underlying `PhysicalModel` class. You can read more about these template classes in section 12.2 of the MRST textbook [19].

Model classes: The basic `GeothermalModel` class implements core functionality for geothermal simulations, like the thermal conductivities discussed in the previous subsection, and functionality for setting up thermal effects in models of wells and surface facilities and in boundary conditions. It also declares a number of so-called state functions for evaluating necessary physical properties and discretizations of intercell fluxes. The main purpose of the class is to assemble the discrete equations, which is done in three steps: First, discrete forms of the accumulation, fluxes, and sources/sinks (i.e., terms one, two, and

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

Table 12.1 *Description of utility functions in the* `geothermal` *module.*

Function	Description
`addThermalFluidProps`	Add thermal properties to the fluid
`addThermalRockProps`	Add thermal properties to the rock
`addThermalBCProps`	Add temperature or heat flux to boundary condition struct
`addThermalWellProps`	Add temperature to wells

four) in (12.5) are obtained from the parent model through the class method `getModelEquations`. Then, molecular diffusion (i.e., term three) is computed and added to the flux, before everything is assembled. Finally, (12.3) is assembled using many of the properties already computed to discretize (12.5). In addition to this model, the `GeothermalExtendedFacilityModel` implements everything related to wells. This includes declaration of state functions for computing heat fluxes in and out of wells and setting up an additional equation to associate the temperature in the wellbore to that in the reservoir.

Utility functions: The introductory example discussed in the previous subsection already introduced you to some of the utility functions the module offers for setting thermal properties for fluids and rock and for specifying thermal boundary conditions. Similar functions also exist for specifying thermal contributions to source terms and well models. Table 12.1 summarizes the module utility functions.

State functions: Rock and fluid properties are conveniently implemented in an object-oriented framework using state functions described in detail in Chapter 5. A `StateFunction` implements a specific physical property with defined dependencies. These are organized together into a state-function group that collects interdependent state functions. The `GeothermalModel` has three such groups: `PVTPropertyFunctions`, `FlowPropertyFunctions`, and `FlowDiscretization`. These groups are populated with state functions that are common to most subsurface flow models, in addition to those specific to geothermal applications. For instance, the PVT property group holds all state functions needed to compute the enthalpy:

```
>> disp(model.PVTPropertyFunctions)
  PVTPropertyFunctions (edit|plot) state function grouping instance.

  Intrinsic state functions (Class properties for PVTPropertyFunctions,
  always present):
                          Density: ThermalDensity (edit|plot)
                 PhasePressures: PhasePressures (edit|plot)
                                  ...
```

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

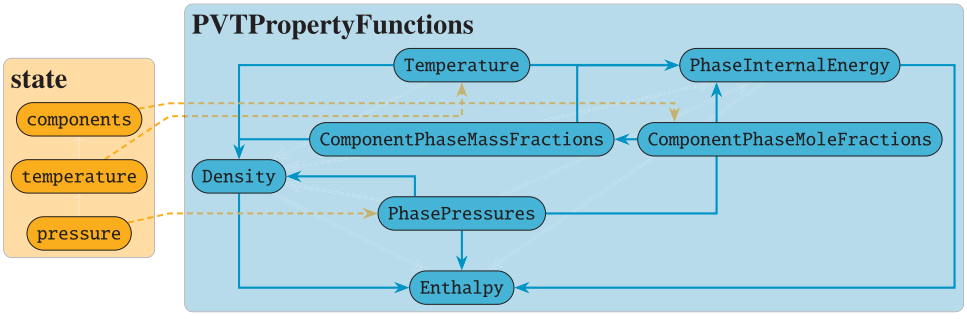

Figure 12.3 Execution graph for computing the enthalpy. Names in gray represent properties and state functions that are inherited from existing properties and state functions in MRST, whereas names in black represent properties and state functions that are specific to the module.

```
Extra state functions (Added to class instance):
   ComponentPhaseMassFractions: ComponentPhaseMassFractionsBrine (edit|plot)
   ComponentPhaseMoleFractions: ComponentPhaseMoleFractionsBrine (edit|plot)
                   Temperature: Temperature (edit|plot)
                      Enthalpy: Enthalpy (edit|plot)
             PhaseInternalEnergy: PhaseInternalEnergy (edit|plot)
                                  ...
```

Figure 12.3 illustrates these properties and the order in which they must be computed. Once a property is computed during a nonlinear solution step, it is cached to state. This way, we avoid potentially costly recomputations of these properties within the same nonlinear iteration. Notice that the state functions for pressure, internal energy, and mole and mass fractions contain `Phase`. This is a formality that will ease the extension to multiple phases. In fact, many of the state functions in `geothermal` already support multiphase fluids. Note also that temperature appears both as a property on state and as a PVT property function. This is because we intend to implement a formulation with enthalpy instead of temperature as a primary variable in the future, so that `Temperature` will be derived from the EoS. Such a formulation is needed to properly treat phase transitions. For now, however, `Temperature` simply gets the temperature from state.

12.4 Numerical Examples

In the following, we present three more examples to demonstrate how the module can be used to simulate various geothermal scenarios, including effects of brine and complex well schedules.

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

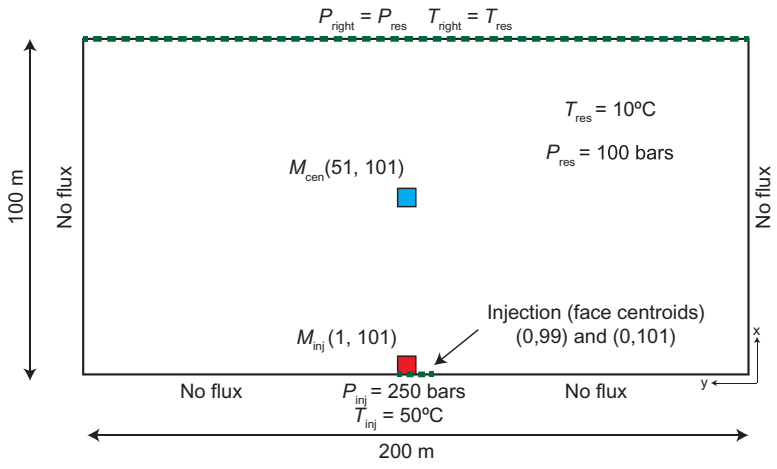

Figure 12.4 Schematic drawing of the model setup (rotated) showing grid dimensions, initial and boundary conditions, and monitoring points (red and blue) for the benchmark between `geothermal` and TOUGH2.

12.4.1 Benchmark with TOUGH2

The `geothermalExampleBenchmark` example presents a comparison with the widely used flow simulator TOUGH2 [25]. Figure 12.4 illustrates the setup, which consists of a 100×200 m^2 Cartesian grid with 50×100 cells and homogeneous rock properties. We add thermal properties to the fluid structure:

```
fluid = addThermalFluidProps(fluid           , ... % Original fluid
                             'Cp'     , 4.2e3, ... % Specific heat capacity
                             'lambdaF', 0.6  , ... % Thermal conductivity
                             'useEOS' , true );    % Use equation of state
```

Thermal properties of the rock are added as before, and we must now provide the medium's tortuosity as well:

```
rock = addThermalRockProps(rock           , ... % Original rock
                           'CpR'    , 1000, ... % Specific heat capacity
                           'lambdaR', 2   , ... % Thermal conductivity
                           'rhoR'   , 2700, ... % Rock density
                           'tau'    , 1   );    % Tortuosity
```

To define a model with brine, we must make a `CompositionalBrineFluid` object. This is inherited from `CompositionalFluid` in the `compositional` module and holds the component names, molar mass, and molecular diffusivity.

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

```
% Provide data for one-phase two-component model with H2O and NaCl
compFluid = CompositionalBrineFluid(          ...
    {'H2O'              , 'NaCl'            }, ... % Component names
    [18.015281*gram/mol, 58.442800*gram/mol], ... % Molar masses
    [0                  , 1e-6              ]);    % Molecular diffusivities
% Make model
model = GeothermalModel(G, rock, fluid, compFluid, 'extraStateOutput', true)
```

In fact, `GeothermalModel` makes a compositional fluid with a single H_2O component by default. Initial temperature, pressure, and NaCl mass fraction are set at 10°C, 100 bar, and 0.1, respectively:

```
state0   = initResSol(G, 100*barsa, 1);       % Pressure and saturation
state0.T = ones(G.cells.num,1).*283.15;       % Temperature
X = repmat([0.9, 0.1], G.cells.num, 1);       % Initial mass fractions
state0.components = model.getMoleFraction(X); % Convert to mole fractions
```

A pressure of 250 bar and a temperature of 50°C are imposed on two faces in the middle of the left boundary to simulate injection of warm pure water in the reservoir. The right boundary has the same initial conditions as the reservoir for pressure, temperature, and NaCl mass fraction to allow for outflow (Figure 12.4):

```
bc  = pside([], G, 'right', 100*barsa, 'sat', 1);        % Right (p)
bc  = addBC(bc, faces, 'pressure', 250*barsa ,'sat', 1); % Left (p)
Tbc = [repmat(283.15, 100, 1); repmat(323.15, 2, 1)];
bc  = addThermalBCProps(bc, 'T', Tbc);                   % Temperature
Xbc = [repmat(0.1, 100, 1); zeros(2, 1)];                % Mass fractions
bc.components = model.getMoleFraction([1-Xbc, Xbc]);     % Mole fractions
```

We monitor pressure, temperature, and NaCl mass fraction in two cells of the model: near the injection and in the center (red and blue, respectively, in Figure 12.4). Generally, we observe less than 1% difference in pressure, NaCl mass fraction, and temperature between both codes (Figure 12.5). However, there are two exceptions where the results show a larger discrepancy: Near the injection, the pressure shows up to 5% difference between the solvers in the beginning, and there is a slight overshoot of the pressure for TOUGH2. The latter may be due to a convergence issue in the solver; indeed, TOUGH2 requires smaller timesteps to converge in the beginning of the simulation than our implementation. This results in reduced numerical diffusion and, indirectly, a more accurate result than from the prescribed steps. The temperature in the model center shows up to 3.6% relative difference between the results after 150 days. This is possibly due to a difference in the formulation of thermal properties, such as internal energy, heat capacity, and conductivity: In TOUGH2, specific heat capacity C_α and thermal conductivity λ_α

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

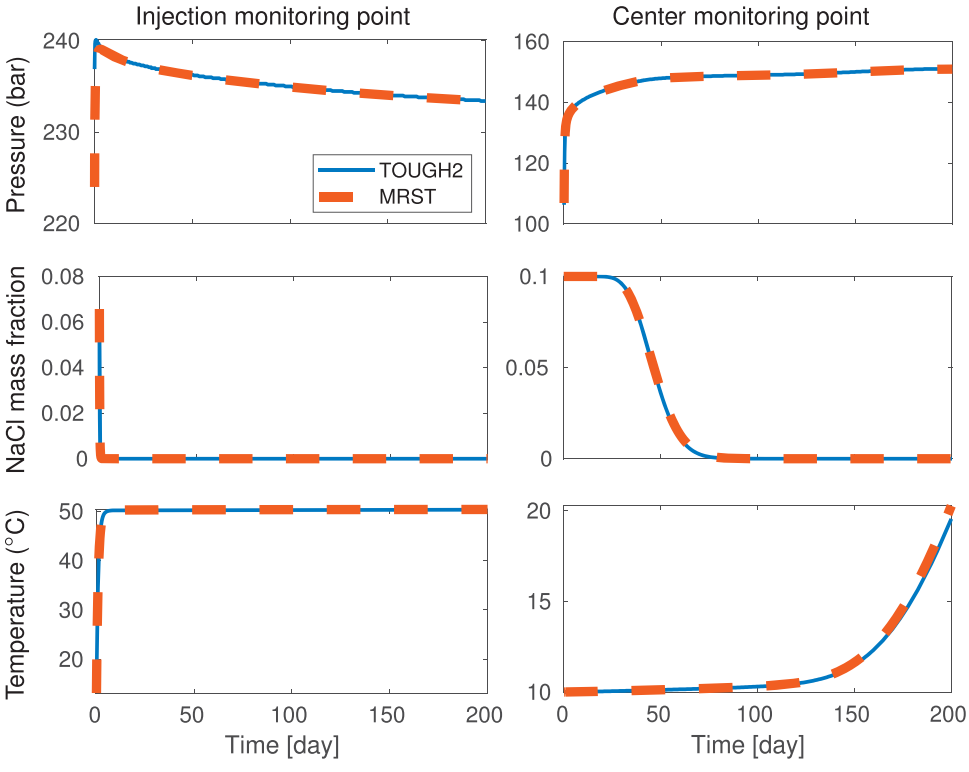

Figure 12.5 Pressure (top), NaCl mass fraction (middle), and temperature (bottom) evolution recorded at two points of the model (see Figure 12.4) for both `geothermal` and TOUGH2.

are functions of both pressure and temperature, whereas we keep these constant in our implementation.

12.4.2 Subset of SPE10 Model 2

In `geothermalExampleSPE10Subset`, we consider an inverted five-spot pattern and use the `upr` module from Chapter 1 to construct a perpendicular bisector grid with refinement around each well. We then sample permeability and porosity from Layer 10 from Model 2 of the SPE10 benchmark study [4]. Figure 12.6 shows the result. To ensure a high thermal energy throughput in the solid rock, we assign a high thermal conductivity and low specific heat capacity to the rock structure:

```
rock = addThermalRockProps(rock, 'lamdaR', 100, 'CpR', 250);
```

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

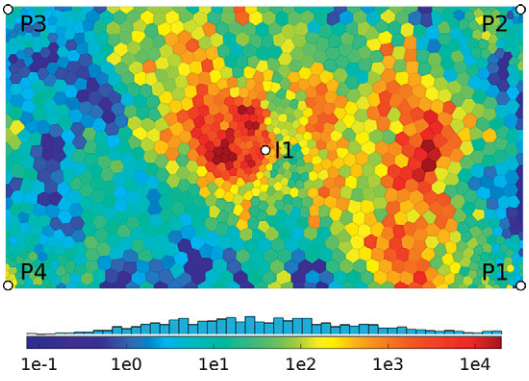

Figure 12.6 Setup for the SPE10 Model 2 example. The colors indicate permeability, and the wells (injection in the center, production in each corner) are indicated by white dots.

The center well injects 100°C water at a constant rate of 50 m^3/day, whereas the producers are operated at a fixed bottom-hole pressure (BHP) of 4 000 psi. We must prescribe the well temperatures after constructing the wells:

```
Tinj = (273.15 + 100)*Kelvin;            % 100 degrees Celsius
W    = addThermalWellProps(W, 'T', Tin); % Add temperature field T to wells
```

We prescribe no-flow boundary conditions on all sides and simulate two types of thermal boundary conditions: fully insulated and fixed temperature. By default, MRST computes mass and energy fluxes across internal interfaces only and adds in fluxes across boundary faces only where boundary conditions are given. This means that the fully insulated case can be set up without providing boundary conditions, whereas fixed-temperature conditions must be provided. We assign a fixed temperature equal to the initial reservoir temperature on all sides:

```
bc = addBC([], boundaryFaces(G), 'flux', 0); % Set up no-flow BCs
bc = addThermalBCProps(bc, 'T', Tres);       % Prescribe temperature
```

Figure 12.7 shows the reservoir temperature at the end of both simulations, along with the logarithm of the (approximate) thermal Péclet number. The thermal Péclet number measures the heat transfer efficiency and is approximated as advective to conductive heat flux:

$$\mathrm{Pe} = \frac{\|\vec{\mathcal{F}}_f h_f\|}{\|\vec{\mathcal{H}}_f + \vec{\mathcal{H}}_r\|} = \frac{\|\rho_f \vec{v}_f h_f\|}{\|-(\lambda_f + \lambda_r)\nabla T\|}.$$

As expected, the fully insulated case achieves a higher temperature near the boundary. The Péclet number also indicates that advective heat transfer dominates in

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

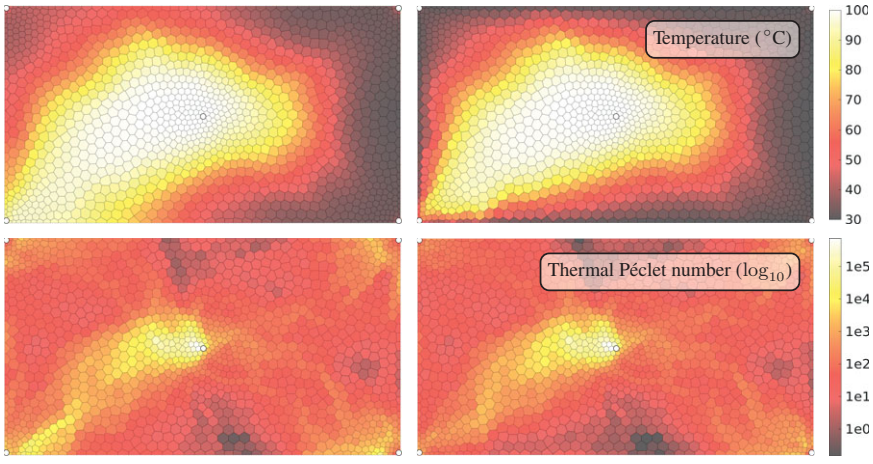

Figure 12.7 Temperature distribution at the end of the simulation using fully insulated boundary conditions (left) and fixed-temperature boundary conditions (right).

almost all parts of the domain, with more than five orders of magnitude near the center well. The Péclet number is also slightly higher with fully insulated boundary conditions.

12.4.3 Enhanced Geothermal System

Enhanced geothermal systems are built by fracturing the subsurface rock in a region with low permeability and high temperature. By injecting and extracting fluids from the resulting fracture network, the fractures serve the same purpose as the fins of a conventional heat exchanger. This setup is ideal for utilizing the abundant geothermal energy, because there is typically almost no flow outside the fractures and consequently a very small heat loss. We model a small enhanced geothermal system by a network of intersecting fractures in a confined box of $50 \times 50 \times 15$ m^3 using a predefined artificial fracture data set. The source code for this example can be found in `geothermalExampleEGS`. From a 2D grid constructed using the `upr` module, we make a 3D model by extruding it in the vertical direction:

```
G2D = pebiGrid2D(dx, xmax(1:2), ...
           'cellConstraints', data.fractures, ... % Fractures
           'CCRefinement'   , true          , ... % Refine fractures
           'CCFactor'       , 0.1           );    % Relative fracture cell size
G = makeLayeredGrid(G2D, layers);
```

Figure 12.8 shows the setup. Permeability and porosity are set to 0.1 md and 0.05 in the matrix and 10 000 md and 0.7 in the fractures.

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

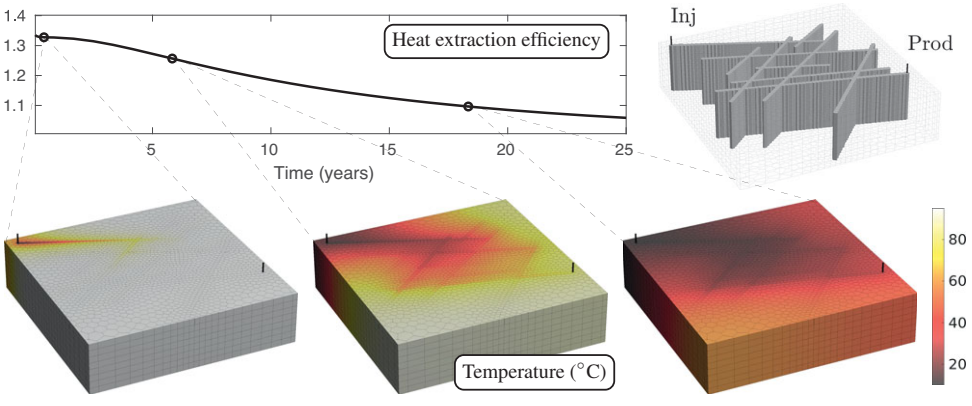

Figure 12.8 Enhanced geothermal system. The injection and production wells are placed in opposite sides of a fracture network, which is 10^4 times more permeable than the matrix rock.

```
rock = makeRock(G, 1*milli*darcy, 0.05);
rock.perm(fracture_cells) = 1e4*milli*darcy; % Fracture permeability
rock.poro(fracture_cells) = 0.8;             % Fracture porosity
```

In this example, we use a density formulation of the form (12.7):

```
fluid = initSimpleADIFluid(                    ...
                'phases', 'W'                , ... % Water only
                'mu'    , 0.5*centi*poise    , ... % Viscosity
                'rho'   , 1000*kilogram/meter^3, ... % Reference density
                'c'     , 4.4e-10/Pascal     , ... % Compressibility
                'pRef'  , 1*atm              );    % Reference pressure

% Add thermal properties
fluid = addThermalFluidProps(fluid              , ...
                'Cp'     , 4.2*joule/(gram*Kelvin), ... % Heat capacity
                'lambdaF', 0.6*Watt/(meter*Kelvin), ... % Thermal conductivity
                'cT'     , 207e-6/Kelvin         , ... % Thermal expansion
                'TRef'   , K0 + 10*Kelvin        );    % Reference temperature
```

Two wells are placed in opposite corners of the fracture network, one injects water at 10°C at a constant rate equal to half the fracture pore volume per year, whereas the other produces at a constant BHP. The initial reservoir temperature is 95°C. Figure 12.8 reports the heat extraction efficiency (produced to injected energy) during 25 years of operation, along with reservoir temperature at three selected timesteps. The reservoir gradually cools down, which is also reflected in the decreasing extraction efficiency.

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

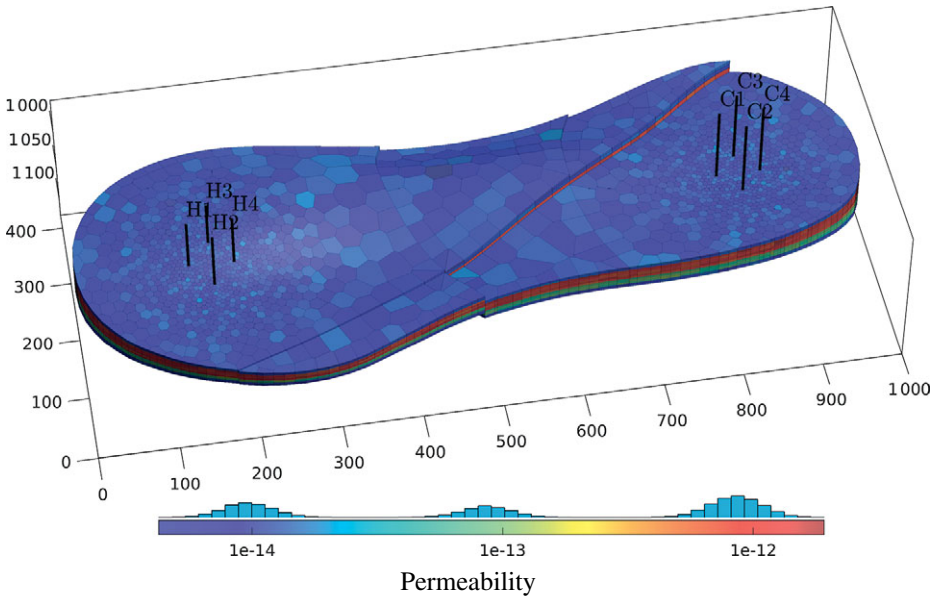

Figure 12.9 Setup for the high-temperature ATES system. Colors indicate the permeability, and the black, vertical pillars indicate the hot (H1–H4) and cold (C1–C4) well groups.

12.4.4 Thermal Aquifer Energy Storage

In the next example, we consider a setup inspired by previous work [5], in which we investigated high-temperature ATES in the Greater Geneva Basin, Switzerland. Thermal energy is produced at a waste incinerator near Geneva and distributed to the local-district heating systems for heating of buildings. However, whereas energy supply (e.g., waste) is fairly constant throughout the year, the energy demand is highly seasonal. Storing excess energy in summer to use it in winter may therefore lead to potentially large energy savings. Full source code for this example is given in the `geothermalExampleHTATES` script.

We use the unstructured Voronoi-type geomodel shown in Figure 12.9. Chapter 1 describes in detail how the `upr` module can be used to generate this grid. The model has three intersecting faults and a layered permeability structure as commonly seen in engineering geomodels. Injection and extraction of heated water are controlled by two groups of four wells each, which we refer to as hot and cold, respectively. The hot group (H1–H4) injects hot water for storage during summer and extracts the hot water for heating in winter. The cold group (C1–C4) provides pressure support by injecting or extracting cold water, with a low BHP during storage and a high BHP during extraction. In between storage and extraction, there is a period of rest during which all wells are closed.

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

For simplicity, we assume a constant energy supply and impose constant water injection of 1 000 m^3 per day at a temperature of 100°C in each of the wells in the hot group during storage and a production rate of 1 000 m^3 per day during extraction. The cold group operates at a constant BHP of 70 bar during storage and 85 bar during extraction. The storage and extraction periods last for four months each, whereas the intermediate rest period lasts for two months. In other words, one cycle (storage–rest–extraction–rest) totals one year, and we simulate four cycles. For simplicity, we neglect salinity effects and assume that the reservoir constitutes a closed, thermally insulated flow compartment. Initial hydrostatic equilibrium is approximated by simulating ten years with fixed pressure and temperature boundary conditions at the topmost part of the reservoir using eight timesteps. The EoS is valid for pressure/temperature within a given range. We provide these to the model, which ensures that pressure/temperature remain within these values during the nonlinear solution step. This is crucial to get a robust nonlinear solver, because intermediate states in a Newton loop may result in pressure and temperature values far outside the EoS validity range.

```
model.maximumPressure    = 200e6;    % Maximum pressure
model.minimumTemperature = K0;       % Minimum temperature
model.maximumTemperature = K0 + 275; % Maximum temperature
```

Because the problem is fairly large (pressure and temperature in 27 449 cells gives 54 898 degrees of freedom), we speed up the simulation using a compiled iterative Krylov solver with constrained pressure residual (CPR) preconditioning [7] together with the mex-accelerated AD backend detailed in Chapter 6.

```
model.AutoDiffBackend = DiagonalAutoDiffBackend('useMex', true);
mrstModule add linearsolvers, lsolver = AMGCL_CPRSolverAD();
```

Figure 12.10 shows the temperature distribution in the reservoir after storage and extraction in the first and last cycles. The plots only show cells where the temperature deviates from its initial value by more than 20%. We see that all important dynamics takes place in the near-well regions, which indicates that the ability to incorporate near-well refinement in the computational grid is crucial for efficient simulation of large cases. Figure 12.11 reports the temperature in all wells as a function of time, along with the energy recovery factor. The latter is defined as the ratio of extracted to stored energy, in this case reported as the total cumulative factor over time. The vertical lines indicate the storage, rest, and extraction periods, with the second cycle (from Year 1 to Year 2) emphasized in color.

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

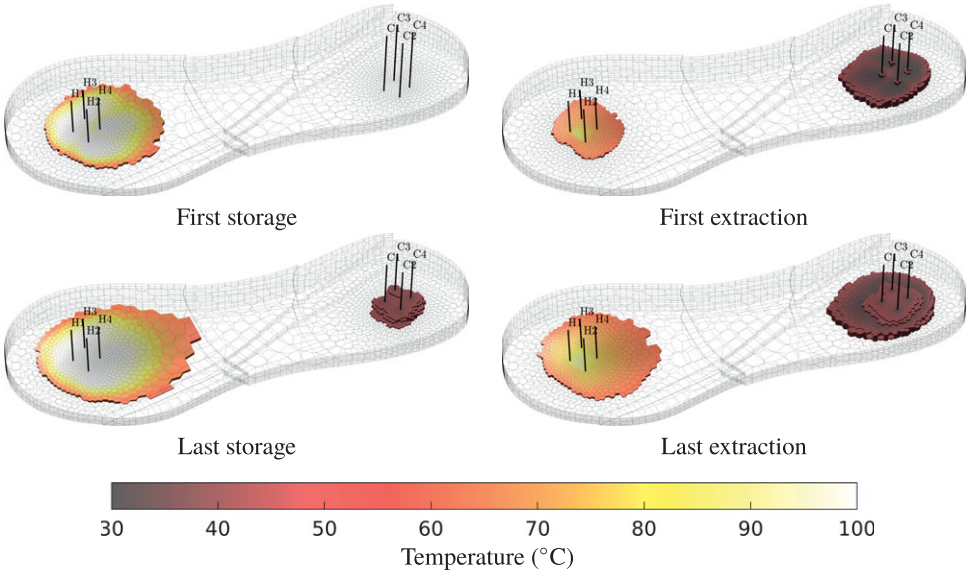

Figure 12.10 Reservoir temperature after storage and extraction in the first and last cycles, reported in cells where the temperature deviates more than 20% from the initial temperature.

This example demonstrates how `geothermal` can be used to simulate complex injection strategies with multiple wells and different constraints. The current setup is, however, very simplified compared to real high-temperature ATES applications, for which factors such as aquifer drift, thermal diffusion out of the reservoir, well operating limits, and temporal variations in energy supply play an important role and tend to result in much lower energy recovery factors. Such effects can also be simulated with `geothermal`, and the reader can refer to [5] for a more advanced example simulated with the `geothermal` module.

12.5 Concluding Remarks

The new `geothermal` module in MRST enables intuitive and rapid testing of geothermal applications involving complex injection schedules and geological grids. Currently, the module can be used to investigate low- to moderate-enthalpy geothermal systems, in which water is always present as a liquid phase. The module can also account for transport of dissolved salt and its impact on fluid flow through density and viscosity changes. The implemented EoS has a pressure, temperature, and salinity range suitable for modeling the first 5 km of most sedimentary basins. As part of MRST, the module can be integrated with well-established

https://doi.org/10.1017/9781009019781.018 Published online by Cambridge University Press

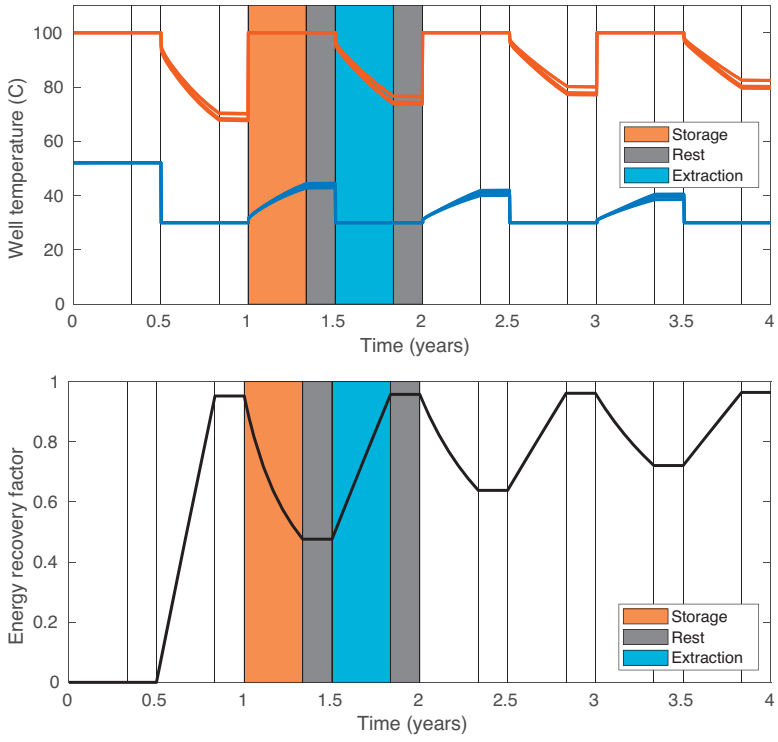

Figure 12.11 Well temperatures and energy recovery during four cycles of high-temperature ATES. Cumulative energy recovery factor measures the efficiency of the system, reported as the aggregated recovery factor for all four wells in the hot group.

functionality from petroleum applications, such as optimal well control, sensitivity analysis, adjoint-based optimization, and uncertainty quantification, and applied to realistic geological models using complex fluid physics. The module represents a first step toward a more advanced module that will consider water-phase transition and the presence of other components such as carbon dioxide or methane, which are commonly found in geothermal systems. Such an implementation will permit exploring high-enthalpy geothermal systems in volcanic settings or mud volcanoes, which possess a strong energetic potential. Other possible extensions include combining the module with existing modules in MRST to consider, e.g., thermomechanical or thermochemical processes, which may have a strong impact on the efficiency of geothermal energy systems.

Acknowledgement. The authors thank Antonio Rinaldi for his contribution to benchmarking `geothermal` with `TOUGH2` and Halvor Møll Nilsen and Odd Andersen for scientific discussions and insightful comments. Finally, we thank Knut-Andreas Lie for improving the chapter through careful review and constructive suggestions.

References

[1] R. Al-Khoury. *Computational Modeling of Shallow Geothermal Systems*, Volume 4 of *Multiphysics Modeling*. CRC Press, 2011.

[2] O. Andersson. Aquifer thermal energy storage. In H. Ö. Paksoy, ed., *Thermal Energy Storage for Sustainable Energy Consumption: Fundamentals, Case Studies and Design*, Volume 234, pp. 155–176. Springer, Dordrecht, The Netherlands, 2007. doi: 10.1007/978-1-4020-5290-3.

[3] T. A. Buscheck, J. M. Bielicki, T. A. Edmunds, Y. Hao, Y. Sun, J. B. Randolph, and M. O. Saar. Multifluid geo-energy systems: using geologic CO_2 storage for geothermal energy production and grid-scale energy storage in sedimentary basins. *Geosphere*, 12(3):678–696, 2016. doi: 10.1130/GES01207.1.

[4] M. A. Christie and M. J. Blunt. Tenth SPE comparative solution project: a comparison of upscaling techniques. *SPE Reservoir Evaluation and Engineering*, 4(4):308–316, 2001. doi: 10.2118/72469-PA.

[5] M. Collignon, Ø. S. Klemetsdal, O. Møyner, M. Alcanié, A. Pio, H. Nilsen, and M. Lupi. Evaluating thermal losses and storage capacity in high-temperature aquifer thermal energy storage (HT-ATES) systems with well operating limits: insights from a study-case in the Greater Geneva Basin, Switzerland. *Geothermics*, 85:101773, 2020. doi: 10.1016/j.geothermics.2019.101773.

[6] U. Colombo. Development and the global environment. In J. M. Hollander, ed., *The Energy-Environment Connection*, pp. 3–14. Island Press, Washington, DC, 1992.

[7] D. Demidov. AMGCL: an efficient, flexible, and extensible algebraic multigrid implementation. *Lobachevskii Journal of Mathematics*, 40(5):535–546, 2019. doi: 10.1134/S1995080219050056.

[8] H.-J. G. Diersch. *Finite Element Modeling of Flow, Mass and Heat Transport in Porous and Fractured Media*. Springer, 2014. doi: 10.1007/978-3-642-38739-5.

[9] I. Dincer. Energy and environmental impacts: present and future perspectives. *Energy Sources*, 20(4-5):427–453, 1998. doi: 10.1080/00908319808970070.

[10] I. Dincer. Renewable energy and sustainable development: a crucial review. *Renewable and Sustainable Energy Reviews*, 4(2):157–175, 2000. doi: 10.1016/S1364-0321(99)00011-8.

[11] I. Dincer and M. A. Rosen. *Thermal Energy Storage: Systems and Applications*. Wiley, 2011.

[12] T. Driesner. The system H_2O-NaCl. Part II. Correlations for molar volume, enthalpy, and isobaric heat capacity from 0 to 1 000 degrees C, 1 to 5 000 bar, and 0 to 1 X_{NaCl}. *Geochimica et Cosmochimica Acta*, 71:4902–4919, 2007. doi: 10.1016/j.gca.2007.05.026.

[13] R. Gibson and O. Loeffler. Pressure–volume–temperature relations in solutions. Part IV. The apparent volumes and thermal expansibilities of sodium chloride and sodium bromide in aqueous solutions between 25 and 95°. *Journal of the American Chemical Society*, 63(2):443–449, 1941. doi: 10.1021/ja01847a026.

[14] W. Glassley. *Geothermal Energy: Renewable Energy and the Environment*. CRC Press, 2010. doi: 10.1201/b17521.

[15] L. Haar, J. S. Gallagher, and G. S. Kell. *NBS/NRC Steam Tables Thermodynamic and Transport Properties and Computer Programs for Vapor and Liquid States of Water in SI Units*. Hemisphere Publishing Company, Washington, DC, 1984.

[16] K. L. J. Kipp, P. Hsieh, and S. Charlton. *Guide to the Revised Ground-Water Flow and Heat Transport Simulator: HYDROTHERM – Version 3*, Volume 6-A25 of *Techniques and Methods*. U.S. Geological Survey, 2008.

[17] C. D. Langevin, D. T. Thorne Jr., A. M. Dausman, M. C. Sukop, and W. Guo. *SEAWAT Version 4.0: A Computer Program for Simulation of Multi-Species Solute and Heat Transport*, Volume 6-A22 of *Techniques and Methods*. U.S. Geological Survey, 2007.

[18] K. S. Lee. A review on concepts, applications, and models of aquifer thermal energy storage systems. *Energies*, 3(6):1320–1334, 2010. doi: 10.3390/en3061320.

[19] 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.

[20] S. Lvov and R. Wood. Equation of state of aqueous NaCl solutions over a wide range of temperatures, pressures and concentrations. *Fluid Phase Equilibria*, 60:273–287, 1990. doi: 10.1016/0378-3812(90)85057-H.

[21] M. J. O'Sullivan, K. Pruess, and M. J. Lippmann. State of the art of geothermal reservoir simulation. *Geothermics*, 30(4):395–429, 2001. doi: 10.1016/S0375-6505(01)00005-0.

[22] C. Palliser and R. McKibbin. A model for deep geothermal brines, I: T-p-X state–space description. *Transport in Porous Media*, 33:65–80, 1998. doi: 10.1023/A:1006537425101.

[23] K. Pitzer, J. Peiper, and R. Busey. Thermodynamic properties of aqueous sodium chloride solutions. *Journal of Physical and Chemical Reference Data*, 13:1–102, 1984. doi: 10.1063/1.555709.

[24] K. Pitzer and M. Sterner. Equations of state for solid NaCl-KCl and saturated liquid NaCl-KCl-H_2O. *Thermochimica Acta*, 218:413–423, 1993. doi: 10.1016/0040-6031(93)80440-L.

[25] K. Pruess, C. Oldenburg, and G. Moridis. *TOUGH2 User's Guide, Version 2.0*. Lawrence Berkeley National Laboratory, 1999.

[26] P. Rogers and K. Pitzer. Volumetric properties of aqueous sodium-chloride solutions. *Journal of Physical and Chemical Reference Data*, 11:15–81, 1982. doi: 10.1016/0040-6031(93)80440-L.

[27] J. P. Spivey, W. D. McCain, and R. North. Estimating density, formation volume factor, compressibility, methane solubility, and viscosity for oilfield brines at temperatures from 0 to 275°C, pressures to 200 MPa, and salinities to 5.7 mole/kg. *Journal of Canadian Petroleum Technology*, 43(7):52–61, 2004. doi: 10.2118/04-07-05.

[28] I. Stober and K. Bucher. *Geothermal Energy: From Theoritical Models to Exploration and Development*. Springer, 2013.

[29] W. Wagner, J. Cooper, A. Dittman, J. Kijima, H.-J. Kretzschmar, A. Kruse, R. Mares, K. Oguchi, H. Sato, I. Støcker, O. Sifner, Y. Takaishi, I. Tanishita, J. Trübenbach, and T. Willkommen. The IAPWS industrial formulation 1997 for the thermodynamic properties of water and steam. *ASME Journal of Engineering for Gas Turbines and Power*, 122:150–182, 2000. doi: 10.1007/978-3-540-74234-0_3.

[30] W. Wagner and A. Pruss. The IAPWS formulation 1995 for the thermodynamic properties of ordinary water substance for general and scientific use. *Journal of Physical and Chemical Reference Data*, 31:387–535, 2002. doi: 10.1063/1.1461829.