10

Solvers for Incompressible Immiscible Flow

The multiphase flow equations introduced in Chapter 8 can describe very different flow behavior depending upon what are the dominant physical effects. During the formation of petroleum reservoirs, fluid movement is primarily driven by buoyancy and capillary forces, which govern how hydrocarbons migrate upward and enter new layers of consolidated sediments. The same effects are thought to dominate long-term geological carbon storage if the buoyant CO_2 phase continues to migrate upward in the formation long after injection has ceased. In recovery of conventional hydrocarbon resources, on the other hand, the predominant force is viscous advection caused by pressure differentials. Here, pressure disturbances will in most cases propagate much faster through the porous medium than the material waves that transport fluid phases and chemical components. This is one of the main reasons why solving multiphase flow equations turns out to be relatively complicated. In addition, we have all the other difficulties already encountered for single-phase flow in Chapters 5–7. The variable coefficients entering the flow equations are highly heterogeneous with orders of magnitude variation and complex spatial patterns involving a wide range of correlation lengths. The grids used to describe real geological media tend to be highly complex, having unstructured topologies, irregular cell geometries, and orders of magnitude aspect ratios. Flow in injection and production wells takes place on small scales relative to the reservoir and hence needs to be modeled using approximate analytical expressions, and so on.

This chapter will teach you how to solve multiphase flow equations in the special case of incompressible rock and immiscible and incompressible fluids. As we saw in Chapter 8, the system of PDEs can then be reformulated so that it consists of an elliptic equation for fluid pressure and one or more transport equations. These transport equations are generally parabolic, but have a strong hyperbolic character (see Section 8.3). Since the pressure and saturations equations have very different mathematical characteristics, it is natural to solve them in consecutive substeps. Examples of such methods include the classical IMPES method [276, 284] (see also [73] and references therein), the adaptive-implicit method (AIM) [280], and the sequentially implicit method [307]. Operator splitting is also used in streamline simulation [79] and in recent multiscale methods [195].

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

Incompressible models are best suited for fluid systems consisting mainly of liquid phases such as in waterflooding of oil reservoirs that either do not contain gas components or are well below the bubble point. Such systems often have weak coupling between pressure and fluid transport, which can be exploited by sequential solution procedures. For systems with a high gas content, significant compressibility effects, strong coupling between different types of flow mechanisms, or large differences in time constants, one generally has to use compressible flow models and fully implicit solvers. Nonetheless, also in this case the mixed elliptic-hyperbolic character of the model equations plays a key role in developing efficient preconditioning strategies [304, 305]. The combination of incompressible models and sequential solution procedures is very popular in academia and for research purposes, since it provides a simple means to develop more clean-cut model equations that still have many of the salient features for multiphase flow.

10.1 Fluid Objects for Multiphase Flow

In Chapter 5, we discussed the basic data objects entering a flow simulation. When going from a single-phase to a multiphase flow model, the most prominent changes take place in the fluid model. It is this model that generally will tell your solver how many phases are present and how these phases affect each other when flowing together in the same porous medium. We therefore start by briefly outlining a few fluid objects that implement the basic fluid behavior discussed in Chapter 8.

To describe an incompressible flow model, we need to know the viscosity and the constant density of each fluid phase, as well as the relative permeabilities of the fluid phases. If the fluid model includes capillary forces, we also need one or more functions that specify the capillary pressure as function of saturation. The most basic multiphase fluid object in MRST implements a simplified version of the Corey model (8.15)

```
fluid = initSimpleFluid('mu' , [   1,  10]*centi*poise     , ...
                        'rho', [1014, 859]*kilogram/meter^3, ...
                        'n'  , [   2,   2]);
```

Here, the residual saturations S_{wr} and S_{nr} are assumed to be zero and the end-points are scaled to unity, so that $k_{rw} = \left(S_w\right)^{n_w}$ and $S_{rn} = \left(1 - S_w\right)^{n_n}$. To recap from Chapter 5, the fluid object offers the following interface to evaluate the petrophysical properties of the fluid:

```
mu = fluid.properties();       % gives mu_w and mu_n
[mu,rho] = fluid.properties(); %    .... plus rho_w and rho_n
```

New to multiphase flow is the `relperm` function, which takes a single fluid saturation or an array of fluid saturations as input and outputs the corresponding values of the relative permeabilities. To plot the relative permeability curves of the `fluid` object, we can use the following code:

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

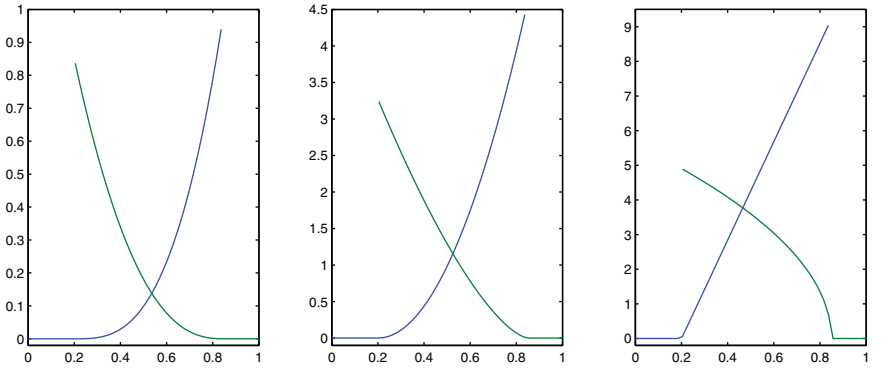


Figure 10.1 Corey relative permeabilities (left) and their first and second derivatives (middle and right) constructed by the `initCoreyFluid` function.

```
s=linspace(0,1,20)';
kr = fluid.relperm(s);
plot(s,kr(:,1),'-s',s,kr(:,2),'-o');
```

The `relperm` function can also return the first and second derivatives of the relative permeability curves when called with two or three output arguments.

The `incomp` module also implements the general Corey model with end-point scaling k_α^0 and nonzero residual saturations S_{wr} and S_{nr}.

```
fluid = initCoreyFluid('mu' , [   1,  10]*centi*poise     , ...
                       'rho', [1014, 859]*kilogram/meter^3, ...
                       'n'  , [   3, 2.5]                 , ...
                       'sr' , [ 0.2, .15]                 , ...
                       'kwm', [   1, .85]);
```

Figure 10.1 shows the relative permeabilities and their first and second derivatives for this particular model.

Whether the flow equations incorporate capillary pressure is specified by the fluid object. The `incomp` module implements two different capillary-pressure models, a simple linear relationship of the form $P_c(S) = C(1 - S)$ and the Leverett J-function scaling (8.9). Both models take the same input as `initSimpleFluid` and are generated the following functions

```
fluid = initSimpleFluidPc( .., 'pc_scale', 2*barsa);
fluid = initSimpleFluidPc( .., 'rock', rock,
         'surf_tension',10*barsa*sqrt(100*milli*darcy/0.1));
```

```
fluid =
    properties: @(varargin)properties(opt,varargin{:})
    saturation: @(x,varargin)x.s
       relperm: @(s,varargin)relperm(s,opt,varargin{:})
            pc: @(state)pc_funct(state,opt)
```

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

You may notice that the capillary function `pc` is evaluated using a *state* object and not a saturation. This may seem awkward, but provides a simpler interface to P_c functions that depend on rock properties, like Leverett-J.

Computer exercises

10.1.1 Modify the Corey model so that it also can output the residual saturations and the end-point scaling values.

10.1.2 Implement the Brooks–Corey (8.16) and the van Genuchten models (8.17) and (8.18).

10.1.3 Extend the models to also include the capillary functions (8.11) and (8.12).

10.2 Sequential Solution Procedures

To solve the two-phase, incompressible model, we rely entirely on the fractional-flow formulation developed in Section 8.3.2. As you may recall, in this formulation, the flow equations consists of an elliptic pressure equation

$$\nabla \cdot \vec{v} = q, \qquad \vec{v} = -\boldsymbol{\lambda}\big(\nabla p_n - f_w \nabla P_c - (\rho_w f_w + \rho_n f_n) g \nabla z\big) \tag{10.1}$$

and a parabolic transport equation

$$\phi \frac{\partial S_w}{\partial t} + \nabla \cdot \big[f_w \big(\vec{v} + \boldsymbol{\lambda}_n (\Delta\rho g \nabla z + \nabla P_c) \big) \big] = q_w. \tag{10.2}$$

Here, the capillary pressure $p_c = p_w - p_n$ is assumed to be a known function P_c of the wetting saturation S_w, and the transport equation becomes hyperbolic whenever P_c' is zero.

In the standard sequential solution procedure, the system (10.1)–(10.2) is evolved in time using a set of discrete time steps Δt_i. Let us assume that p, $\vec{v}$, and S_w are all known at time t and that we want to evolve the solution to time $t + \Delta t$. At the beginning of the time step, we first assume that the saturation S_w is fixed. This means that the parameters $\boldsymbol{\lambda}$, f_w, and f_n in (10.1) become functions of the spatial variable $\vec{x}$ only. We then use the resulting Poisson-type equation to update pressure p_n and Darcy velocity $\vec{v}$. Next, we hold $\vec{v}$ and p_n fixed while (10.2) is evolved a time step Δt to define an updated saturation $S_w(\vec{x}, t + \Delta t)$. This saturation is then held fixed when we update p_n and $\vec{v}$ in the next time step, and so on.

Some authors refer to this solution procedure as an *operator splitting* method, since the solution procedure effectively splits the overall solution operator of the flow model into two parts that are evolved in consecutive substeps. Likewise, some authors refer to the sequential solution procedure as IMPES, which is short-hand for *implicit pressure, explicit saturation*. Strictly speaking, using the name IMPES is only correct if the saturation evolution is approximated by a single time step of an explicit transport solver. The size of the splitting step Δt is then restricted by the CFL condition of the explicit scheme. In many

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

cases, the overall flow system does not have stability requirements that necessitate such a restriction on Δt. Indeed, by writing the flow model in the fractional-flow formulation, we have isolated the parts of the system that have stability restrictions to the hyperbolic saturation equation. The elliptic pressure equation, on the other hand, describes (smooth) solutions resulting from the instant redistribution of pressure in a system with infinite speed of propagation. For this equation, we can therefore in principle use as large time step as we want.

As long as we ensure that the evolving discontinuities and sharp transitions are propagated in a stable manner in the saturation equation, our only concern when choosing the size of the splitting step should be to control or minimize the *splitting error* introduced by accounting for pressure and transport in separate substeps. The fractional-flow formulation underlying our operator splitting was developed to minimize the coupling between saturation and pressure. For incompressible flow models, the effect that dynamic changes in the saturation field have on pressure is governed entirely by the total mobility $\boldsymbol{\lambda}(S)$, which in many cases is a function that locally has small and relatively smooth variation in time. For this reason, you can typically use splitting steps that are significantly larger than the CFL restriction from the hyperbolic part of the saturation equation and still accurately resolve the coupling between pressure and saturation. In other words, for each pressure update, the saturation can be updated by an explicit solver using multiple saturation substeps, or by an implicit solver using either a single or multiple saturation substeps. If necessary, you can also iterate on the splitting steps.

10.2.1 Pressure Solvers

The pressure equation (10.1) for incompressible, multiphase flow is time dependent. This time dependence comes as the result of three factors:

- $\mathbf{K}/\mu$ is replaced by the total mobility $\boldsymbol{\lambda}(S_w)$, which depends on time through the saturation $S_w(\vec{x}, t)$,
- the constant density ρ is replaced by a saturation-dependent quantity $\rho_w f_w(S_w) + \rho_n f_n(S_w)$, and
- the new source term $q - \nabla \boldsymbol{\lambda}_w(S_w) \nabla P_c(S_w)$ depends on saturation.

Nevertheless, once S_w is held fixed in time, all three quantities become functions of $\vec{x}$ only, and we hence end up again with an elliptic Poisson-type equation having the same spatial variation as in (4.10) on page 117. Hence, we can either use the two-point scheme introduced in Section 4.4.1 or one of the consistent discretization methods from Chapter 6, *mutatis mutandis*. The `incompTPFA` solver discussed in Chapter 5 and the `incompMimetic` and `incompMPFA` solvers from Chapter 6 are implemented so that they solve the pressure equation for a general system of m incompressible phases. Whether this system has one or more phases is determined by the fluid object and the reservoir state introduced in Section 5.1.2. We will therefore not discuss the pressure solvers in more detail.

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

10.2.2 Saturation Solvers

Apart from the time loop, which we have already encountered in Chapter 7, the only remaining part we need is a solver for the transport equation (10.2) that implements the discretizations we introduced in Section 9.4. Summarized, this can be written in the following residual form, for each cell Ω_i

$$\mathcal{F}_i(s,r) \;=\; s_i \;-\; r_i \;+\; \frac{\Delta t}{\phi_i|\Omega_i|}\left[\sum_k H_{ik}(s) - \max(q_i,0) - \min(q_i,0)\,f(S_i)\right]. \quad (10.3)$$

Here, s and r are cell-averaged quantities and subscript i refers to the cell the average is evaluated in. The sum of the interface fluxes for cell i

$$H_i(s) = \sum_k \frac{\lambda_w^u(s_i,s_k)}{\lambda_w^u(s_i,s_k)+\lambda_n^u(s_i,s_k)}\big[v_{ik} + \lambda_n^u(s_i,s_k)(g_{ik}+P_{ik})\big]. \quad (10.4)$$

is computed using the single-point, upstream mobility-weighting scheme discussed on page 287, whereas the fractional flow function f in the source term is evaluated from the cell average of S in cell Ω_i. The explicit scheme is given as $S^{n+1} = S^n - \mathcal{F}(S^n, S^n)$ and the implicit scheme follows as a coupled system of discrete nonlinear equations if we set $\mathcal{F}(S^{n+1}, S^n) = 0$. In the following, we discuss the inner workings of these solvers in more detail.

Explicit Solver

The `incomp` module offers the following explicit transport solver

```
state = explicitTransport(state, G, tf, rock, fluid, 'mech1', obj1, ..)
```

which evolves the saturation given in the `state` object a step `tf` forward in time. The function requires a complete and compatible model description consisting of a grid structure `G`, petrophysical properties `rock`, and a fluid model `fluid`. For the solver to be functional, the `state` object must contain the correct number of saturations per cell and an *incompressible* flux field that is consistent with the global drive mechanisms given by the `'mech'` argument (`'src'`, `'bc'`, and/or `'wells'`) accompanied by correctly specified objects `obj`, as discussed in Sections 5.1.3–5.1.5.

In practice, this means that the *input value* of `state` must be the *output value* of a previous call to an incompressible solver like `incompTPFA`, `incompMPFA`, or `incompMimetic`. In addition, the function takes a number of optional parameters that determine whether the time steps are prescribed by the user or to be automatically computed by the solver. The solver can also ignore the Darcy flux and work as a pure gravity segregation solver if the optional parameter `onlygrav` is set to true. Finally, the solver will issue a warning if the updated saturation value is more than `satwarn` outside the interval $[0, 1]$ of physically meaningful values (default value: `sqrt(eps)`).

The explicit solver involves many of the same operations and formulas used for the spatial discretizations as the implicit solver. To avoid duplication of code we have therefore introduced a private helper function

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

```
[F,Jac] = twophaseJacobian(G, state, rock, fluid, 'pn1', pv1, ..)
```

that implements the residual form (10.3) and its Jacobian matrix $J = d\mathcal{F}$, returned as two function handles, `F` and `Jac`. With this, the key lines of the explicit saturation solver read

```
F = twophaseJacobian(G, state, rock, fluid, 'wells', opt.wells, ..);
s  = state.s(:,1);
t  = 0;
while t < tf,
   dt   = min(tf-t, getdt(state));
   s(:) = s - F(state, state, dt);
   t    = t + dt;
   s    = correct_saturations(s, opt.satwarn);
   state.s = [s, 1-s];
 end
```

Here, the function `getdt` implements a CFL restriction on the time step by estimating the maximum derivative of each function used to assemble interface fluxes and source terms (you can find details in the code). The function `correct_saturations` ensures that the computed saturations stay inside the interval of physically valid states. If this function issues a warning, it is highly likely that your time step exceeds the stability limit, or something is wrong with your fluxes or setup of the model.

Implicit Solver

The implicit solver has the same user interface and parameter requirement as the explicit solver

```
state = implicitTransport(state, G, tf, rock, fluid, 'mech1', obj1, ..)
```

In addition, there are optional parameters controlling the Newton–Raphson method used to solve for S^{n+1}. To describe this method, we start by writing the residual equations (10.3) for all cells in vector form

$$\boldsymbol{F}(\boldsymbol{s}) = \boldsymbol{s} - \boldsymbol{S} + \frac{\Delta t}{\boldsymbol{\phi}|\boldsymbol{\Omega}|}\left[\boldsymbol{H}(\boldsymbol{s}) - \boldsymbol{Q}^{+} - \boldsymbol{Q}^{-}\boldsymbol{f}(\boldsymbol{s})\right] = \boldsymbol{0}. \tag{10.5}$$

Here, $\boldsymbol{s}$ is the unknown state at time `tf` and $\boldsymbol{S}$ is the known state at the start of the time step. As you may recall from Section 7.1, the Newton–Raphson linearization of an equation like (10.5) can be written as

$$\boldsymbol{0} = \boldsymbol{F}(\boldsymbol{s}_0 + \boldsymbol{\delta s}) \approx \boldsymbol{F}(\boldsymbol{s}_0) + \nabla \boldsymbol{F}(\boldsymbol{s}_0)\boldsymbol{\delta s},$$

which naturally suggests an iterative scheme in which the approximate solution $\boldsymbol{s}^{\ell+1}$ in the $(\ell+1)$-th iteration is obtained from

$$\boldsymbol{J}(\boldsymbol{s}^{\ell})\,\delta\boldsymbol{s}^{\ell+1} = -\boldsymbol{F}(\boldsymbol{s}^{\ell}), \qquad \boldsymbol{s}^{\ell+1} \leftarrow \boldsymbol{s}^{\ell} + \delta\boldsymbol{s}^{\ell+1}. \tag{10.6}$$

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

Here, $\delta \boldsymbol{s}^{\ell+1}$ is called the *Newton update* and $\boldsymbol{J}$ is the Jacobian matrix. The `incomp` module was implemented before automatic differentiation was introduced in MRST, and hence the Jacobian is computed analytically through the following expansion

$$\boldsymbol{J}(\boldsymbol{s}) = \frac{d\boldsymbol{F}}{d\boldsymbol{s}}(\boldsymbol{s}) = \mathbf{1} + \frac{\Delta t}{\boldsymbol{\phi}|\boldsymbol{\Omega}|}\Big[\frac{d\boldsymbol{H}}{d\boldsymbol{s}}(\boldsymbol{s}) - \boldsymbol{Q}^{-}\frac{d\boldsymbol{f}}{d\boldsymbol{s}}(\boldsymbol{s})\Big],$$
$$\frac{d\boldsymbol{H}}{d\boldsymbol{s}} = \frac{d\boldsymbol{H}}{d\boldsymbol{\lambda}_w}\frac{d\boldsymbol{\lambda}_w}{d\boldsymbol{s}} + \frac{d\boldsymbol{H}}{d\boldsymbol{\lambda}_n}\frac{d\boldsymbol{\lambda}_n}{d\boldsymbol{s}} + \boldsymbol{f}_w\boldsymbol{\lambda}_n\frac{d\boldsymbol{P}}{d\boldsymbol{s}},$$
$$\frac{d\boldsymbol{H}}{d\boldsymbol{\lambda}_w} = \frac{\boldsymbol{f}_w}{\boldsymbol{\lambda}}\big[\boldsymbol{v} + \boldsymbol{\lambda}_n(\boldsymbol{g} + \boldsymbol{P})\big], \qquad \frac{d\boldsymbol{H}}{d\boldsymbol{\lambda}_n} = -\frac{\boldsymbol{f}_w}{\boldsymbol{\lambda}}\big[\boldsymbol{v} - \boldsymbol{\lambda}_w(\boldsymbol{g} + \boldsymbol{P})\big].$$

In general, we are not guaranteed that the resulting values in the vector $\boldsymbol{s}^{\ell+1}$ lie in the interval $[0, 1]$. To ensure physically meaningful saturation values, we can introduce a *line-search* method, which uses the Newton update to define a *search direction* $\boldsymbol{p}^{\ell} = \delta \boldsymbol{s}^{\ell+1}$ and tries to find the value α that minimizes $h(\alpha) = F(\boldsymbol{s}^{\ell} + \alpha\, \boldsymbol{p}^{\ell})$. We may now either solve $h'(\alpha) = 0$ exactly, or use an *inexact line-search method* that only asks for a sufficient decrease in h. In the implicit solver discussed herein, we have chosen the latter approach and use an unsophisticated method that reduces α in a geometric sequence. The following code should give you the idea:

```
function [state, res, alph, fail] = linesearch(state, ds, target, F, ni)
   capSat = @(sat) min(max(0, sat),0);
   [alph,i,fail] = deal(0,0,true);
   sn = state;
   while fail && (i < ni),
      sn.s(:,1) = capSat(state.s(:,1) + pow2(ds, alph));
      res  = F(sn);
      alph = alph - 1;   i = i + 1;
      fail = ~(norm(res, inf) < target);
   end
   alph = pow2(alph + 1); state.s = sn.s;
```

Here, `F` is a function handle to the residual function $\boldsymbol{F}$. The number of trials `ni` in the line-search method is set through the optional parameter `'lstrails'`, whereas the target value is set as the parameter `'resred'` times the residual error upon entry. Default values for `'lstrails'` and `'resred'` are 20 and 0.99, respectively.

The implicit discretization is stable in the sense that there exists a solution $\boldsymbol{S}^{n+1}$ for an arbitrarily large time increment Δt. Unfortunately, there is no guarantee that we will be able to find this solution using the line-search method described previously. If the time step is too large, the Newton method may simply compute search directions that do not point us toward the correct solution. To compensate for this, we also need a mechanism that reduces the time step if the iteration does not converge and then uses a sequence of shorter time step to reach the prescribed time `tf`. First of all, we need to define what we mean by convergence. This is defined by the optional `'nltol'` parameter, which sets the absolute tolerance ϵ (default value 10^{-6}) on the residual $\|\mathcal{F}(S^{n+1}, S^n)\|_{\infty} \leq \epsilon$. In addition,

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

we use a parameter 'maxnewt' that gives the maximum number of iteration steps (default value 25) the method can take to reach a converged solution. The following code gives the essence of the overall algorithm of the iterative solver, as implemented in the helper function newtonRaphson2ph

```
mints   = pow2(tf, -opt.tsref);
[t, dt] = deal(0.0, tf);
while t < tf && dt >= mints,
   dt = min(dt, tf - t);
   redo_newton = true;
   while redo_newton,
      sn_0 = resSol; sn = resSol; sn.s(:) = min(1,sn.s+0.05);
      res  = F(sn, sn_0, dt);
      err  = norm(res(:), inf);
      [nwtfail, linfail, it]  = deal(err>opt.nltol,false,0);
      while nwtfail && ~linfail && it < opt.maxnewt,
         J  =  Jac(sn, sn_0, dt);
         ds = -reshape(opt.LinSolve(J, reshape(res', [], 1)), ns, [])';
         [sn, res, alph, linfail] = update(sn, sn_0, ds, dt, err);
         it      = it + 1;
         err     = norm(res(:), inf);
         nwtfail = err > opt.nltol;
      end
      if nwtfail,
         % Chop time step in two, or use previous successful dt
      else
         redo_newton = false;
         t = t + dt;
         % If five successful steps, increase dt by 50%
      end
   end
   resSol = sn;
end
```

The algorithm has two optional parameters: 'tsref' with default value 12 gives the number of times we can halve the time step, whereas 'LinSolve' is the linear solver, which defaults to mldivide. Beyond this, the best way to find more details about the solver is to read the code.

10.3 Simulation Examples

You have now been introduced to all the functionality you need to solve incompressible, multiphase flow problems. It is therefore time to start looking into the qualitative behavior of such systems and the typical multiphase phenomena you may encounter in practice. The examples presented in the following are designed to highlight individual effects, or combinations of effects, and may not always be fully realistic in terms of physical scales, magnitude of the parameters and effects involved, etc. We will also briefly look at the

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

structure of the discrete systems arising in the implicit transport solver. The last section of the chapter discusses numerical errors resulting from specific choices of discretizations and solution strategies.

We have already seen that there are essentially three effects that determine the direction of a single-phase flow field. The first is *heterogeneity* (i.e., spatial variations in permeability) that affects the local magnitude and direction of the flow. The second effect is introduced by *drive mechanisms*, such as wells and boundary conditions, that determine where fluids flow to and from. However, the further you are from the location of a well or a boundary condition, the less effect it will have on the flow direction. The last effect is gravity.

Multiphase flow is more complicated, since the fluid dynamics is now also affected by the viscosity and density ratios of the fluids present, as well as by relative permeability and capillary pressure. These effects will introduce several challenges. If the displacing fluid is more mobile than the resident fluid, it will tend to move rapidly into this fluid, giving weak shock fronts and long rarefaction waves. For a homogeneous reservoir, this will result in early breakthrough of the displacing fluid, and slow and incremental recovery of the resident fluid. In a heterogeneous reservoir, you may also observe *viscous fingering*, which essentially means that the displacing fluid moves unevenly into the resident fluid. This is a self-reinforcing effect that causes the fingers to move farther into the resident fluid.

Gravity segregation, on the other hand, will force fluids having different density to segregate, and lead to phenomena such as *gravity override*, in which a lighter fluid moves quickly on top of a denser fluid. This is a problem in many recovery methods relying on gas injection and for geological storage of CO_2. Finally, we have capillary effects, which tend to spread out the interface between the invading and displaced fluids. When combined with heterogeneity, these effects are generally difficult to predict without detailed simulations. Sometimes, they work in the same direction to aggravate sweep and displacement efficiency, but can also counteract each other to cancel undesired behavior. Gravity and capillary forces, for instance, may both reduce viscous fingering that would otherwise give undesired early breakthrough.

This section discusses most of the phenomena just outlined in more detail. In most cases, we only consider a single effect at the time. Throughout the examples, you will also learn how to set up various types of simulations using MRST. As a rule, for brevity we will not discuss complete codes, and when reading the material you should therefore take the time to also read the accompanying codes found in the `in2ph` directory of the `book` module. You will gain much more insight if you run these codes and try to modify them to study the effect of different parameters and algorithmic choices. I also strongly encourage you to do as many of the computer exercises as possible.

10.3.1 Buckley–Leverett Displacement

As a first example, let us revisit the 1D horizontal setup from Example 9.3.4 on page 284, in which we compared explicit and implicit transport solvers for the classic Buckley–Leverett

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

displacement profile arising when pure water is injected into pure oil. The following code sets up a slightly rescaled version of the problem, computes the pressure solution, and then uses the explicit transport solver to evolve the saturations forward in time

```
G     = computeGeometry(cartGrid([100,1]));
rock  = makeRock(G, 100*milli*darcy, 0.2);
fluid = initSimpleFluid('mu' , [1, 1].*centi*poise, ...
                        'rho', [1000, 1000].*kilogram/meter^3, 'n', [2,2]);
bc    = fluxside([], G, 'Left',   1, 'sat', [1 0]);
bc    = fluxside(bc, G, 'Right', -1, 'sat', [0 1]);
hT    = computeTrans(G, rock);
rSol  = initState(G, [], 0, [0 1]);
rSol  = incompTPFA(rSol, G, hT, fluid, 'bc', bc);
rSole = explicitTransport(rSol, G, 10, rock, fluid, 'bc', bc, 'verbose',true);
```

The explicit solver uses 199 time steps to reach time 10. Let us try to see if we can do this in one step with the implicit solver:

```
[rSoli, report] = ...
 implicitTransport(rSol, G, 10, rock, fluid, 'bc', bc, 'Verbose', true);
```

This corresponds to running the solver with a CFL number of approximately 200. From the output in Figure 10.2 we see that this is not a big success. With an attempted time step of $\Delta t = 10$, the solver only manages to reduce the residual by a factor 2.5 within the allowed 25 iterations. Likewise, when the time step is halved to $\Delta t = 5$, the solver still only manages to reduce the residual one order of magnitude within the 25 iterations. When the time step is halved once more, the solver converges in 20 iterations in the first step and then in 9 iterations in the next 3 substeps. This time-step chopping is not very efficient: Altogether, more than half of the iterations (50 out of 97) were wasted trying to compute time steps that would not converge. The explicit solver avoids this problem since the time step is restricted by a CFL condition, but requires significantly more time steps.

To overcome the problem with wasted iterations, we can explicitly subdivide the pressure step into multiple saturation steps:

```
rSolt = rSol;
for i=1:n
   rSolt = implicitTransport(rSolt, G, 10/n, rock, fluid, 'bc', bc);
end
```

Figure 10.3 reports the approximate solutions and the overall number of iterations used by the implicit solver with n equally spaced substeps. The solver typically needs more iterations during the first time steps when the displacement front is relatively sharp. As the simulation progresses, the shock is smeared across multiple cells, which contributes to weaken the nonlinearity of the discrete system and hence reduce the number of required iterations. This explains why the reported number of iterations is not an exact multiple of the number of time steps. We also see that to get comparable accuracy as the explicit

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

```
------------------------------------------------------------------
Time interval (s)     iter  relax   residual          rate
------------------------------------------------------------------
 [0.0e+00, 1.0e+01]:   1    1.00    7.82670e+00        NaN
 [0.0e+00, 1.0e+01]:   2    0.12    6.81721e+00       0.07
  :          :         :     :          :              :
 [0.0e+00, 1.0e+01]:  25    0.03    3.17993e+00       1.00
-------------------- Reducing step --------------------
 [0.0e+00, 5.0e+00]:   1    1.00    7.03045e+00        NaN
  :          :         :     :          :              :
 [0.0e+00, 5.0e+00]:  25    0.25    6.34074e-01       0.74
-------------------- Reducing step --------------------
 [0.0e+00, 2.5e+00]:   1    1.00    5.64024e+00        NaN
  :          :         :     :          :              :
 [0.0e+00, 2.5e+00]:  20    1.00    1.23535e-08       1.91
-------------------- Next substep ---------------------
 [2.5e+00, 5.0e+00]:   1    0.25    3.59382e-01        NaN
  :          :         :     :          :              :
 [2.5e+00, 5.0e+00]:   9    1.00    1.71654e-08       1.86
-------------------- Next substep ---------------------
 [5.0e+00, 7.5e+00]:   1    0.25    2.67914e-01        NaN
  :          :         :     :          :              :
 [5.0e+00, 7.5e+00]:   9    1.00    8.49299e-09       1.94
-------------------- Next substep ---------------------
 [7.5e+00, 1.0e+01]:   1    0.25    2.57462e-01        NaN
  :          :         :     :          :              :
 [7.5e+00, 1.0e+01]:   9    1.00    8.74041e-11       2.01

 Iterations        :  47            Wasted iterations : 50
 Sub steps         :   4            Failed steps      : 2
 Final residual    :   8.74e-11     Convergence rate  : 1.9
------------------------------------------------------------------
```

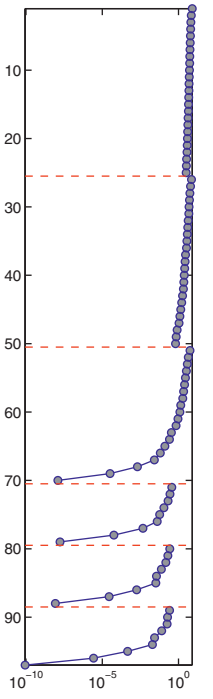


Figure 10.2 Results from running the `implicitTransport` solver on a 1D Buckley–Leverett displacement problem with CFL number 200. Left: screen output, where several lines have been deleted for brevity. Right: convergence history for the residual, with cumulative iteration number increasing from top to bottom.

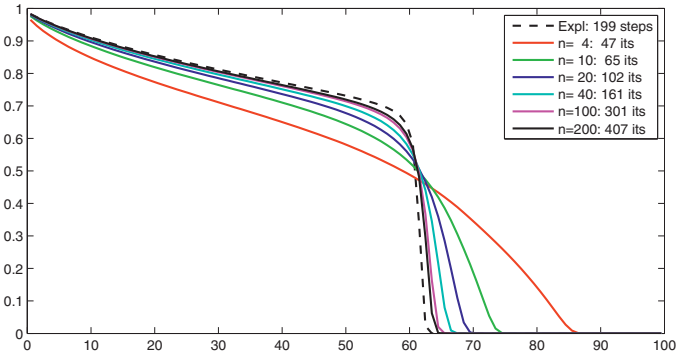


Figure 10.3 Approximate solutions computed by the explicit transport solver and the implicit transport solver with n time steps.

transport solver, the implicit solver needs to use at least 40 time steps, which amounts to more than 160 iterations. In this case, there is thus a clear advantage of using the explicit transport solver if we want to maximize accuracy versus computational cost.

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

10.3.2 Inverted Gravity Column

In the next example, we revisit the inverted gravity column from Example 8.4.3 on page 267 with a light fluid at the bottom and a heavier fluid at the top. We change the setup slightly so that the fluids are representative for supercritical CO_2 and brine found at conditions that would be plausible when storing CO_2 in a deep saline aquifer. The following is the essence of the simulator (plotting commands are not included for brevity):

```
gravity reset on
G     = computeGeometry(cartGrid([1, 1, 40], [1, 1, 10]));
rock  = makeRock(G, 0.1*darcy, 1);
fluid = initCoreyFluid('mu' , [0.30860, 0.056641]*centi*poise, ...
          'rho', [975.86,686.54]*kilogram/meter^3, ...
          'n', [2,2], 'sr', [.1,.2], 'kwm',[.2142,.85]);
hT = computeTrans(G, rock);
xr = initResSol(G, 100.0*barsa, 1.0); xr.s(end/2+1:end) = 0.0;
xr = incompTPFA(xr, G, hT, fluid);
dt = 5*day; t=0;
for i=1:150
   xr = explicitTransport(xr, G, dt, rock, fluid, 'onlygrav', true);
   t = t+dt;
   xr = incompTPFA(xr, G, hT, fluid);
end
```

In Example 8.4.3 the fluids had the same viscosity and hence moved equally fast upward and downward. Here, supercritical CO_2 is much more mobile than brine and will move faster to the top of the column than brine moves downward. Hence, whereas the CO_2 reaches the top of the column after 250 days, it takes more than 400 days before the first brine has sunk to the bottom. After approximately two years, the fluids are clearly segregated and separated by a sharp interface; see Figure 10.4.

In the simulation, we used relatively small splitting steps (150 steps of 5 days each) to march the transient solution towards steady state. Looking at Figure 10.5, which shows the vertical pressure distribution every fiftieth day (i.e., for every tenth time step), we see that the pressure behaves relatively smoothly compared with the saturation distribution. It is therefore reasonable to expect that we could get away with using a smaller number of splitting steps in this particular case. How many splitting steps do you think we need?

Before leaving the problem, let us inspect the discrete nonlinear system from the implicit transport solver in some detail. Figure 10.6 contrasts the sparsity pattern at two instances in time to that of the 1D horizontal Buckley–Leverett problem. From the discussion in Section 9.3.4, we know that the latter only has a single nonzero band below the diagonal and hence can be solved more robustly if we instead of using Newton's method, use a nonlinear substitution method with a bracketing method for each single-cell problem [220]. Since the lighter fluid moves upward and the heavier fluid moves downward during gravity segregation, there will be nonzero elements above and below the diagonal in the

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

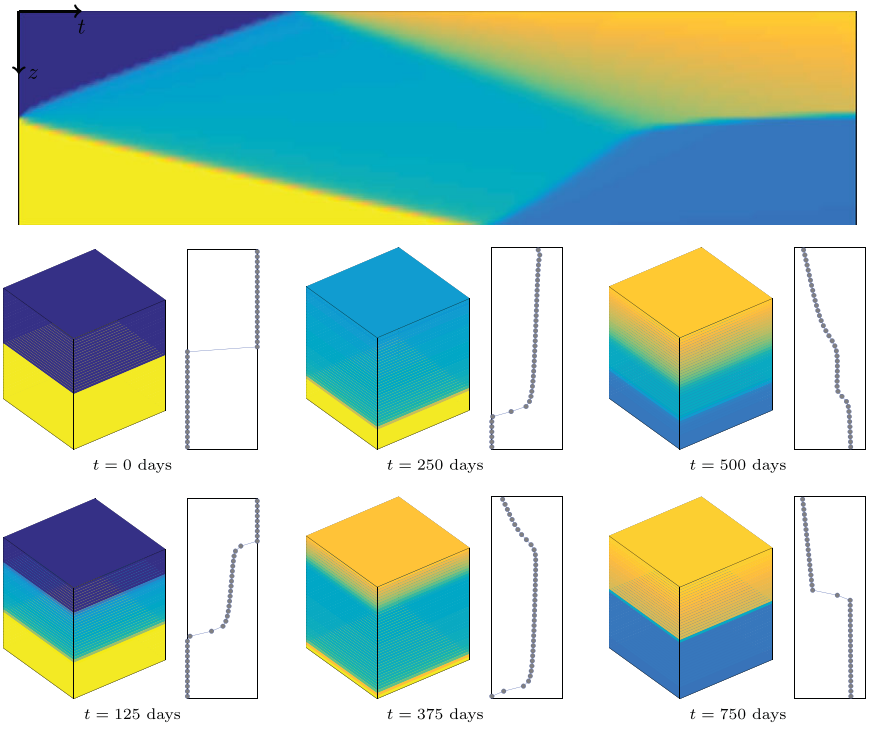


Figure 10.4 Simulation of an inverted gravity column where pure CO_2 initially fills the bottom half and brine the upper half of the volume. The upper plot shows $S(z,t)$ with yellow color signifying pure CO_2 and blue color signifying pure brine. At equilibrium, the CO_2 at the top contains some irreducible water and the brine at the bottom contains residual CO_2, which thus can be considered as safely trapped within the immobile brine.

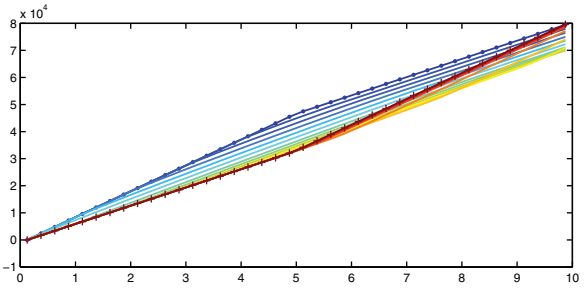


Figure 10.5 Pressure distribution at every tenth time step as function of depth from the top of the gravity column, with blue color and dotted markers indicating initial time and red color and cross markers indicating end of simulation.

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

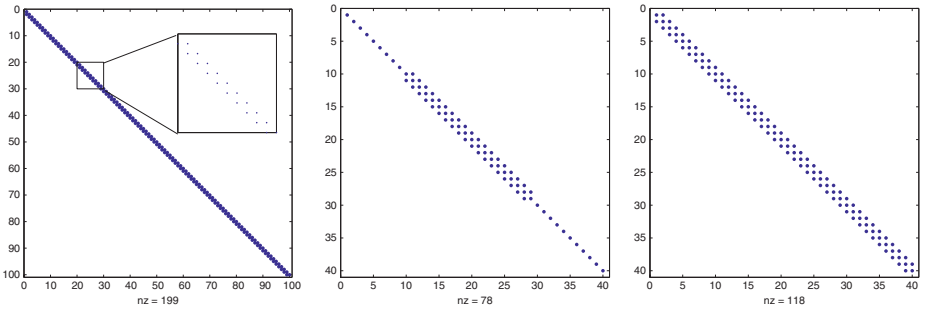


Figure 10.6 Sparsity pattern for the 1D Buckley–Leverett problem (left) and the inverted gravity column after 125 days (middle) and 390 days (right)

two-phase region and at the interface between the two phases. In the figure, we also see how the sparsity pattern changes as the two-phase region expands upward and downward from the initial interface. Since the nonlinear system is no longer triangular, a substitution method cannot be used, but each linearized system can be solved efficiently using the Thomas algorithm, which is a special $\mathcal{O}(n)$ Gaussian elimination method for tridiagonal systems. Knowing the sparsity pattern of the problem is a key to efficient solvers. The `mldivide` solver (i.e., `A\b`) in MATLAB performs an analysis of the linear system and picks efficient solvers for triangular, tridiagonal, and other special systems. To confirm that the optimal solver is indeed used, you can type `spparms('spumoni',2)` before you run the example.

10.3.3 Homogeneous Quarter Five-Spot

To gain more insight into the simulation of multiphase displacement processes, we consider the classical confined quarter five-spot test case discretized on a 128×128 grid. As you may recall from Section 5.4.1, this test case consists of one quarter of a symmetric pattern of four injectors surrounding a producer (or vice versa), repeated to infinity in each direction. We neglect capillary and gravity forces and assume a simplified Corey model with exponent 2.0 and zero residual saturation, i.e., $k_{rw} = S^2$ and $k_{ro} = (1 - S)^2$. We start by setting the viscosity to 1 cP for both fluid phases, giving a unit mobility ratio. The injector and producer operate at fixed bottom-hole pressure, giving a total pressure drop of 100 bar across the reservoir. Since we have assumed incompressible flow, equal amounts of fluid must be produced from the reservoir so that injection and production rates sum to zero. The total time is set such that 1.2 pore volumes of fluid will be injected if the initial injection is maintained throughout the whole simulation. However, the actual injection rate will depend on the total resistance to flow offered by the reservoir, and hence vary with time when the total mobility varies throughout the reservoir as a result of fluid movement. (Remember that the total mobility is less in all cells containing two fluid phases.) The simulation code

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

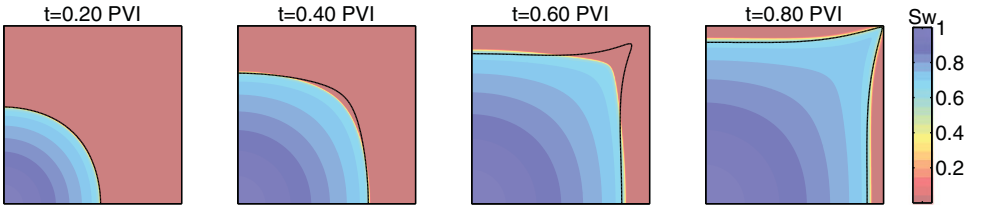


Figure 10.7 Evolution of a two-phase displacement front for a homogeneous quarter five-spot case with water injected into oil. The single line shown in each plot is a contour at value $t/(2\sqrt{2}-2)$ PVI for the time-of-flight field computed from the corresponding single-phase problem (i.e., with $\lambda \equiv 1$).

follows the same principles as outlined in the two examples just discussed; you can find details in the file `quaterFiveSpot2D.m`.

Figure 10.7 shows how the displacement profile resulting from injection of water into the oil-filled reservoir expands circularly near the injector. As the displacement front propagates into the reservoir, it gradually elongates along the diagonal and forms a finger that extends towards the producer. As a result, water breaks through in the producer long time before the displacement front has managed to sweep the stagnant regions near the northwest and southeast corners. The evolution of the saturation profile is the result of two different multiphase effects.

To better understand these effects, it is instructive to transform our 3D transport equation into streamline coordinates. Since the flow field is incompressible, we can use (4.44) to write $\vec{v} \cdot \nabla = \phi \frac{\partial}{\partial \tau}$ so that (10.2) transforms to a family of 1D transport equations, one along each streamline,

$$\frac{\partial S}{\partial t} + \frac{\partial f_w(S)}{\partial \tau} = \frac{q_w}{\phi}. \tag{10.7}$$

Hence, the first flow effect is exactly the same Buckley–Leverett displacement as we saw in Section 10.3.1, except that it now acts along streamlines rather than along the axial directions. It is therefore tempting to suggest that to get a good idea of how a multiphase displacement will evolve, we can solve a single-phase pressure equation for the initial oil-filled reservoir, compute the resulting time-of-flight field, and then map the 1D Buckley–Leverett profile (8.60) computed from (8.53) onto time-of-flight. How accurate this approximation is depends on the coupling between the saturation and pressure equations.

With linear relative permeabilities and unit mobility ratio, there is no coupling between pressure and transport, and mapping 1D solutions onto time-of-flight thus produces the correct solution. In other cases, changes in total mobility will modify the total Darcy velocity and hence the time-of-flight. To illustrate this, let us compare the propagation of the leading shock predicted by the full multiphase simulation and our simplified streamline analysis. For fluids with a viscosity ratio $\mu_w/\mu_n = M$, it follows by solving $f'(S) = f(S)/S$ that the leading shock of the Buckley–Leverett displacement profile moves at a speed $M/(2\sqrt{M+1}-2)$ relative to the Darcy velocity, shown as a single black line

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

for each snapshot in Figure 10.7. Compared with our simplified streamline analysis, the movement of the injected water is retarded by the reduced mobility in the two-phase region behind the leading displacement front.

In most simulations, the primary interest is to predict well responses. To extract these, we introduce the following function:

```
function wellSol = getWellSol(W, x, fluid)
mu = fluid.properties();
wellSol(numel(W))=struct;
for i=1:numel(W)
  out = min(x.wellSol(i).flux,0); iout = out<0;  % find producers
  in  = max(x.wellSol(i).flux,0); iin  = in>0;   % find injectors
  lamc = fluid.relperm(x.s(W(i).cells,:))./mu;   % mob in completed cell
  fc   = lamc(:,1)./sum(lamc,2);                 %
  lamw = fluid.relperm(W(i).compi)./mu;          % mob inside wellbore
  fw   = lamw(:,1)./sum(lamw,2);                 %
  wellSol(i).name = W(i).name;
  wellSol(i).bhp  = x.wellSol(i).pressure;
  wellSol(i).wcut = iout.*fc + iin.*fw;
  wellSol(i).Sw   = iout.*x.s(W(i).cells,1) + iin.*W(i).compi(1);
  wellSol(i).qWs  = sum(out.*fc) + sum(in.*fw);
  wellSol(i).qOs  = sum(out.*(1-fc)) + sum(in.*(1-fw));
end
```

Inside the simulation loop, this function is called as follows

```
[wellSols, oip] = deal(cell(N,1), zeros(N,1));
for n=1:N
   x  = incompTPFA(x, G, hT, fluid, 'wells', W);
   x  = explicitTransport(x, G, dT, rock, fluid, 'wells', W);
   wellSols{n} = getWellSol(W, x, fluid);
   oip(n)      = sum(x.s(:,2).*pv);
end
```

The loop also computes the oil in place at each time step. Storing well responses in a cell array may seem unnecessarily complicated and requires a somewhat awkward construction to plot the result

```
t = cumsum(dT);
plot(t, cellfun(@(x) x(2).qOs, wellSols));
```

The call to `cellfun` passes elements from the cell array `wellSols` to an anonymous function that extracts the desired field `qOs` containing the surface oil rate of the second well (the producer). Each element represents an individual time step. The reason for using cell arrays is to provide compatibility with the infrastructure developed for compressible models of industry-standard complexity. Here, use of a cell array provides the flexibility needed to process much more complicated well output. As a direct benefit, we can use a GUI developed for visualizing well responses:

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

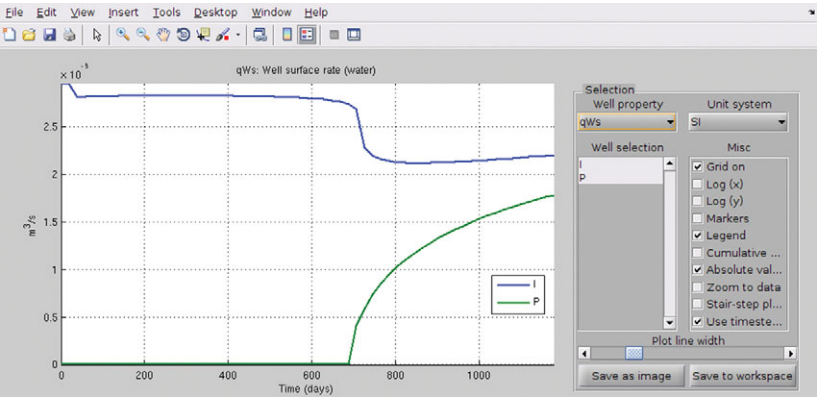


Figure 10.8 Graphical user interface `plotWellSols` from the `ad-core` module for plotting computed well responses. The GUI does not work in GNU Octave.

```
mrstModule add ad-core
plotWellSols(wellSols,cumsum(dt))
```

This brings up a plotting window as shown in Figure 10.8, where we have chosen to visualize the surface water rate for injector and producer.

To better understand the deviation between the ideal and the actual recovery, we can look at the well responses and total mass balance in more detail, as shown in Figure 10.9. We see that the oil rate drops immediately as water enters the system in the first time step and then decays slowly until water breaks through in the producer around time $t = 0.7$. Since this well now produces a mixture of water and oil, the oil rate decays rapidly as the smoothed displacement front enters the well, and then decays more slowly when the inflow of water is determined by the trailing rarefaction wave. The left plot shows the cumulative oil production computed in two different ways: (i) using the oil rate from the well solution, and (ii) measured as the difference between initial and present oil in place. Up to water breakthrough, the two estimates coincide. After water breakthrough, the production estimated from the well solution will be slightly off, since it for each time step uses a simplified approximation that multiplies the size of the time step with the total flow rate computed *at the start* of the time step and the fractional flow in the completed cell *at the end of the time step*.

Computer exercises

10.3.1 Repeat the experiment with wells controlled by rate instead of pressure. Do you observe any differences and can you explain them?

10.3.2 Repeat the experiment with different mobility ratios and Corey exponents.

10.3.3 Can you correct the computation of oil/water rates?

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

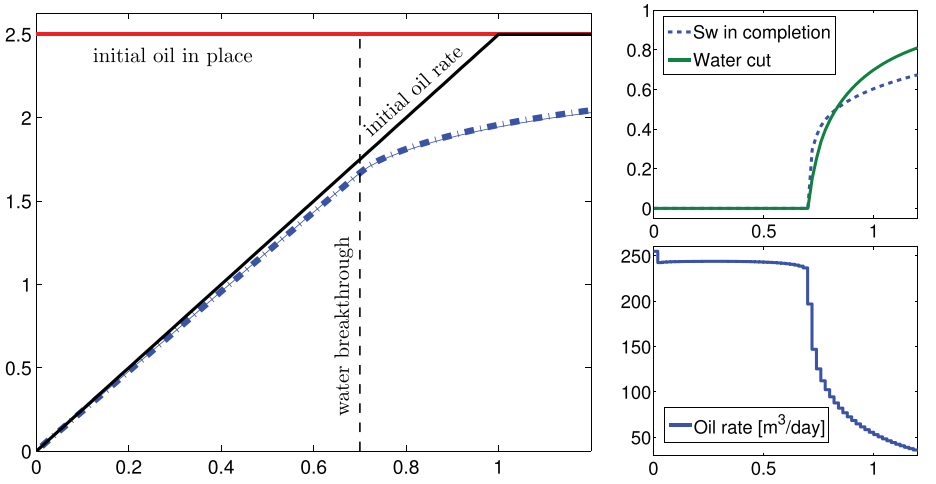


Figure 10.9 Well responses computed for the homogeneous quarter five-spot test. The left plot shows the cumulative oil production computed from the well solution shown as a thin line compared with the amount of extracted oil derived from a mass-balance computation (initial oil in place minus current oil in place) shown as a thick dashed line. The lower-right plot shows oil rate and the upper-right plot shows water saturation and water cut (fractional flow) in the well perforation.

10.3.4 Heterogeneous Quarter Five-Spot: Viscous Fingering

In the previous example we studied imbibition in a homogeneous medium, which resulted in symmetric displacement profiles. However, when a displacement front propagates through a porous medium, the combination of viscosity differences and permeability heterogeneity may introduce viscous fingering effects. In general, the term *viscous fingering* refers to the onset and evolution of instabilities at the interface between the displacing and displaced fluid phases. Fingering can arise because of viscosity differences between two phases or as a result of viscosity variations within a single phase that, for instance, contains solutes. In the laboratory, viscous fingering is usually studied in so-called Hele–Shaw cells, which consist of two flat plates separated by a tiny gap. The plates can be completely parallel, or contain small-scale variations (rugosity) to emulate a porous medium. When a viscous fluid confined in the space between the two plates is driven out by injecting a less viscous fluid (e.g., dyed water injected into glycerin), beautiful and complex fingering patterns can be observed. I recommend a search for "Hele–Shaw cell" on YouTube.

Figure 10.10 shows results of three different simulations of a quarter five-spot on a square domain represented on a uniform 60×120 grid with petrophysical properties sampled from the topmost layer of the SPE 10 data set. The wells operate at a fixed rate, corresponding to the injection of one pore volume over a period of 20 years. To reach a final time, we use 200 pressure steps and the implicit transport solver. As in the previous example, the fluid is assumed to obey a simple Corey fluid model with quadratic relative permeabilities and no residual saturations. (Complete code for the example is found in `viscousFingeringQ5.m`.)

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

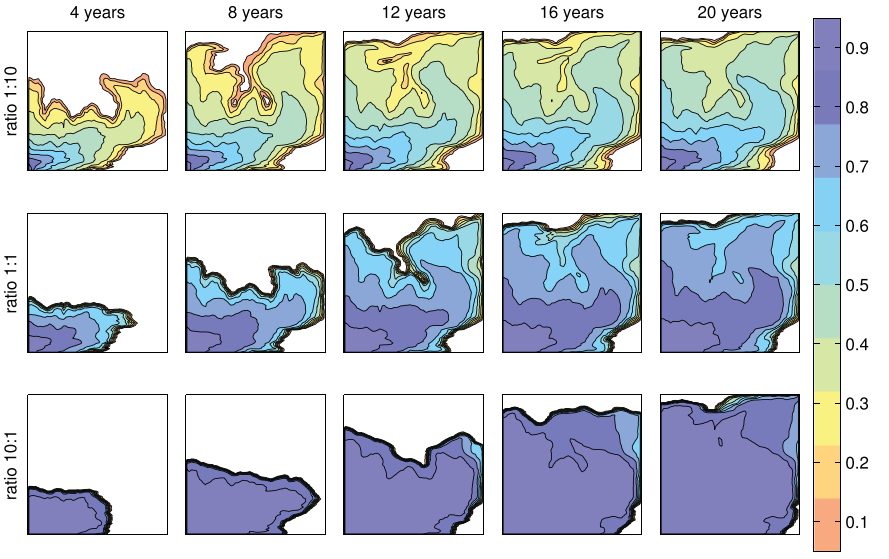


Figure 10.10 Quarter five-spot solutions on a subsample from the first layer of the SPE 10 model for three different viscosity ratios $\mu_w : \mu_n$.

If the viscosity of the injected fluid is (significantly) less than that of the resident fluid, we get an *unfavorable displacement* in which the displacing phase forms a weak shock front that will "finger" rapidly through the less mobile phase that initially fills the medium. Once a finger develops, it will create a preferential flow direction for the injected phase, which causes the finger to extend towards the producer, following the path of highest permeability. In the opposite case of a (significantly) more viscous fluid being injected into a less viscous fluid, one obtains a strong front that acts almost like a piston and creates a very *favorable and stable displacement* with a leading front that has much fewer buckles than in the unfavorable case. Not only does this front have better local displacement efficiency (i.e., it can push out more oil), but the areal sweep is also better. The unit viscosity case is somewhere in between the two, having a much better local displacement efficiency than the unfavorable case, but almost the same areal sweep at the end of the simulation.

Figure 10.11 reports well responses for the three simulations. Because water is injected at a fixed rate, the oil rate will remain constant until water breaks through in the producer. This happens after 1,825 days in the unfavorable case, after 4,050 days for equal viscosities, and after 6,300 days in the favorable case. As discussed in the previous example, the decay in oil rate depends on the strength of the displacement front and will hence be much more abrupt in the favorable mobility case, which has an almost piston-like displacement front. On the other hand, by the time the favorable case breaks through, the unfavorable case has reached a water cut of 82%. Water handling is generally expensive and in the worst case the unfavorable case might shut down before reaching the end of the 20-year production period.

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

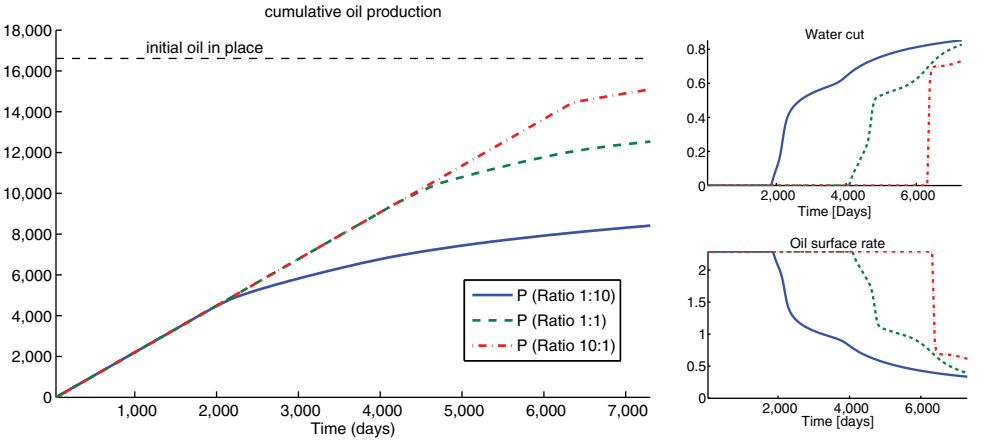


Figure 10.11 Well responses computed for the heterogeneous quarter five-spot with different mobility ratios $\mu_w : \mu_n$.

Next, we look at the sparsity of the transport equation. In the homogeneous case, all fluxes point in the positive axial directions and the Jacobian matrix is thus lower triangular. With heterogeneous permeability, or another well pattern, this unidirectional flow property is no longer present and the Jacobian matrix will have elements above and below the diagonal; see Figure 10.12. However, if we look at the transformation to streamline coordinates (10.7), it is clear that we still have unidirectional flow along streamlines. This means that the Jacobian matrix can be permuted to triangular form by performing a *topological sort* on the flux graph derived from the total Darcy velocity. In MATLAB, you can try to permute the matrix to upper-triangular form by use of the Dulmage–Mendelsohn decomposition:

```
[p,q] = dmperm(J); Js = J(p,q);
```

The result is shown in the middle plot in Figure 10.12. This permutation is similar to what is done inside MATLAB's linear solver `mldivide`. Since we can permute the Jacobian matrix to triangular form, we can also do the same for the nonlinear system, and hence apply a highly efficient nonlinear substitution method [220] as discussed for 1D Buckley–Leverett problems in Section 9.3.4. The same applies also for 3D cases as long as long as capillary forces are neglected and we have purely co-current flow. Countercurrent flow can be introduced by gravity segregation, as we saw in Section 10.3.2, or if the flow field is computed by one of the consistent discretization schemes from Chapter 6, which are generally not monotone.

It is also possible to permute the discretized system to triangular form by performing a potential ordering [175], as shown to the right in Figure 10.12:

```
[~,i] = sort(x.pressure); Jp = J(i,i);
```

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

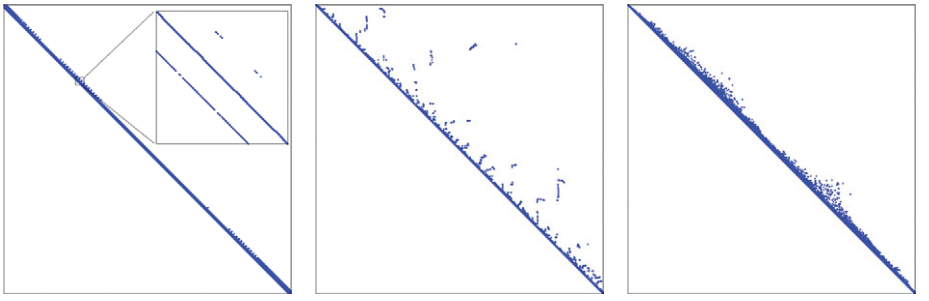


Figure 10.12 The left plot shows the sparsity structure for the Jacobian matrix for the heterogeneous quarter five-spot with viscosity ratio 1:10. The middle plot shows the sparsity structure after a topological sort, whereas the right plot shows the sparsity structure after potential ordering.

Implementing this type of nonlinear substitution methods is unfortunately not very efficient in MATLAB and should be done using a compiled language. In MRST, we therefore mainly rely on the intelligence built into `mldivide` to give us the required computational performance.

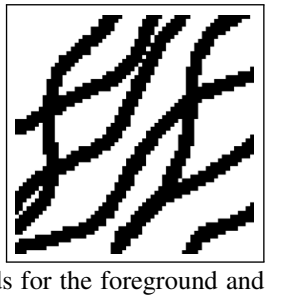
A remark at the end: in a real case, *injectivity* would obviously be a decisive factor, i.e., how high pressure is required to ensure a desired injection rate without fracturing the formation, or vice versa, which rate one would obtain for a given injection pressure below the fracturing pressure. This is not accounted for in our earlier discussion; for illustration purposes we tacitly assumed that the desired injection rate could be maintained.

Computer exercises

10.3.1 Repeat the experiments with wells controlled by pressure, fixed water viscosity, and varying oil viscosity. Can you explain the differences you observe?

10.3.2 Run a systematic study that repeats the quarter five-spot simulation from the previous exercise for each of the 85 layers of the SPE 10 model. Plot and compare the resulting production curves. (You can run the experiment for a single mobility ratio to save computational time.)

10.3.3 Run the same type of study with 100 random permeability fields, e.g., as generated by the simplified `gaussianField` routine from Section 2.5.2. Alternatively, you can use any kind of drawing program to generate a bitmap and generate a channelized permeability as follows



```
K = ones(G.cartDims)*darcy;
I = imread('test.pbm');
I = flipud(I(:,:,1))';
K(I) = milli*darcy;
```

This can easily be combined with different random fields for the foreground and background permeability.

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

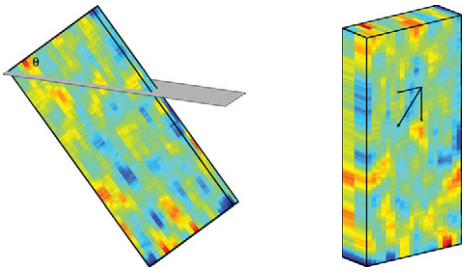


Figure 10.13 Illustration of the sloping sandbox used for the buoyancy example and how it is simulated by rotating the gravity vector. (Color: Gaussian porosity field.)

10.3.5 Buoyant Migration of CO_2 in a Sloping Sandbox

In Section 10.3.2, we considered the buoyant migration of supercritical CO_2 inside a vertical column. Here, we extend the problem to three spatial dimensions and simulate the upward movement of CO_2 inside a sloping sandbox with sealing boundaries. The rectangular sandbox has dimensions $100 \times 10 \times 200$ m^3, and we consider two different petrophysical models: homogeneous properties or Gaussian porosity with isotropic permeability given from a Carman–Kozeny transformation similar to (2.6). The sandbox is rotated around the y-axis so that the top surface makes an angle of inclination θ with the horizontal plane. Instead of rotating the grid so that it aligns with the aquifer geometry, we will rotate the coordinate system by rotating the gravity vector an angle θ around the y-axis; see Figure 10.13. The rotation is introduced as follows:

```
R = makehgtform('yrotate',-pi*theta/180);
gravity reset on
gravity( R(1:3,1:3)*gravity().' );
```

MRST defines the gravity vector as a persistent, global variable, which by default (for historical reasons) equals $\vec{0}$. The second line sets $\vec{g}$ to the standard value (pointing downward vertically) before we perform the rotation.

To initialize the problem, we assume that CO_2, which is lighter than the resident brine, fills up the model from the bottom and to a prescribed height,

```
xr = initResSol(G, 1*barsa, 1);
d  = gravity() ./ norm(gravity);
dc = G.cells.centroids * d.';
xr.s(dc>max(dc)-height) = 0;
```

For accuracy and stability, the time step is ramped up gradually as follows,

```
dT = [.5, .5, 1, 1, 1, 2, 2, 2, 5, 5, 10, 10, 15, 20, ...
      repmat(25,[1,97])].*day;
```

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

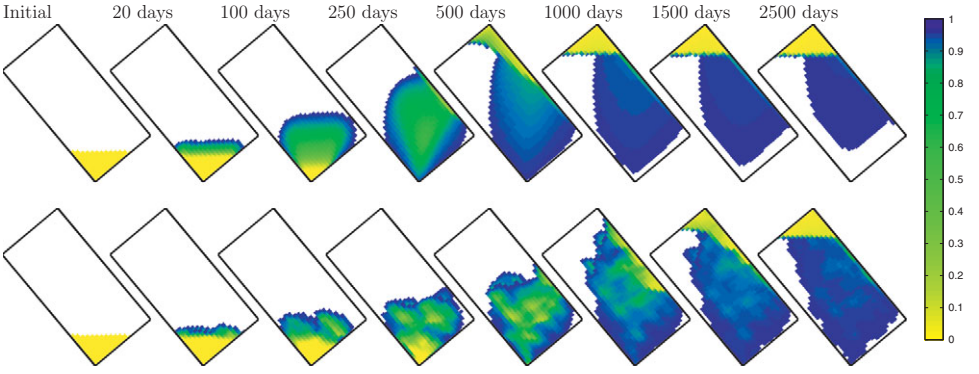


Figure 10.14 Buoyant migration of CO_2 in a sandbox with sealing boundaries. The upper plots show the homogeneous case and the lower plots the case with Gaussian heterogeneity. In the plots, only cells containing some CO_2 are colored.

to reach a final simulation time of 2,500 days. The remaining code is similar to what was discussed in earlier examples; details can be found in `buoyancyExample.m`.

For the homogeneous case, the buoyant CO_2 plume will initially form a cone shape as it migrates upward and gradually drains the resident brine. After approximately 175 days, the plume starts to accumulate as a thin layer of pure CO_2 under the sloping east face of the box. This layer will migrate quickly up towards the topmost northeast corner of the box, which is reached after approximately 400 days, This corner forms a structural trap that will gradually be filled as more CO_2 migrates upward. The trapped CO_2 forms a diffused and curved interface (see the plots at 500 and 1,000 days), but as time passes, the interface becomes sharper and flatter. During the same period, brine will imbibe into the trailing edge of the CO_2 plume and gradually form a layer of pure brine at the bottom. After approximately 1,000 days, the only CO_2 left below the interface in the northeast corner is found at small saturation values and will therefore migrate very slowly upward.

The heterogeneous case follows much of the same pattern, except that the leading drainage front will finger into high-permeability regions of the sandbox. Low-permeability cells, on the other hand, will retard the plume migration. Altogether, we see a significant delay in the buoyant migration compared with the homogeneous case. Since the permeability is isotropic, the plume will still mainly migrate upward. This should not be expected in general. Many rocks have significantly lower permeability in the vertical direction or consist of strongly layered sandstones containing mud drapes or other thin deposits that inhibit vertical movement between layers. For such cases, one can expect a much larger degree of lateral movement.

Computer exercises

10.3.1 To gain more insight into the flow physics of a buoyant phase, you should experiment more with the `buoyancyExample` script. A few examples:

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

- Set $\theta = 88$ and initial height to 10 meters for the heterogeneous case.
- Set $\theta = 60^\circ$ and impose an anisotropic permeability with ratio 0.1:1:5 to mimic a case with strong layering.
- In the experiments so far, we used an unrealistic fluid model without residual saturations. Replace the fluid model by a more general Corey model having residual saturations (typical values could be 0.1 or 0.2) and possibly also end-point scaling. How does this affect the upward plume migration? (Hint: in addition to the structural trapping at the top of the formation, you will now have residual trapping.)

10.3.2 Rerun the experiment with capillary forces included, e.g., by using `initSimpleFluidJfunc` with parameters set as in its documentation.

10.3.3 Replace the initial layer of CO_2 at the bottom of the reservoir by an injector that injects CO_2 at constant rate before it is shut in. For this, you should increase the spatial and temporal scales of the problem.

10.3.4 Gravity will introduce circular currents that destroy the unidirectional flow property we discussed in Section 10.3.4. To investigate the sparsity of the discretized transport equations, you can set a breakpoint inside the private function `newtonRaphson2ph` used by the implicit transport solver and use the `plotReorder` script from the `book` module to permute the Jacobian matrix to block-triangular form. Try to stop the simulation in multiple time steps to investigate how the sparsity structure and degree of countercurrent flow change throughout the simulation.

10.3.6 Water Coning and Gravity Override

Water coning is a production problem in which water (from a bottom drive) is sucked up in a conical shape towards a producer. This is highly undesirable since it reduces the hydrocarbon production. As an example, we consider a production setup on a sector model consisting of two different rock types separated by a fully conductive, inclined fault as shown in Figure 10.15.

A vertical injector is placed in the low-permeability stone (K= 50 md and $\phi = 0.1$) to the east of the fault, whereas a horizontal producer is perforated along the top of the more permeable rock (K= 500 md and $\phi = 0.2$) to the west of the fault. The injector operates at a fixed bottom-hole pressure of 700 bar and the producer operates at a fixed bottom-hole pressure of 100 bar. To clearly illustrate the water coning, we consider oil with somewhat contrived properties: density 100 kg/m^3 and viscosity 10 cP. The injected water has density 1,000 kg/m^3 and viscosity 1 cP. Both fluids have quadratic relative permeabilities. The large density difference was chosen to ensure a bottom water drive, while the high viscosity was chosen to enhance the coning effect. Complete source code can be found in `coningExample.m` in the `book` module.

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

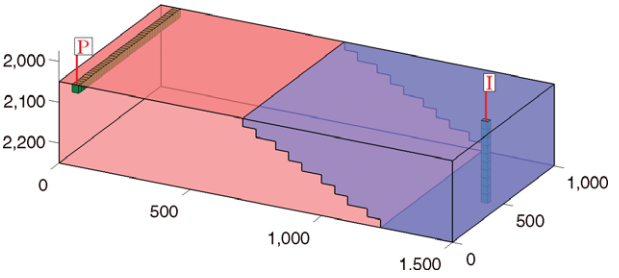


Figure 10.15 Sector model used to demonstrate water coning. Blue color indicates low permeability and red color high permeability.

The idea of using a vertical injector is that the lower completions will set up a bottom water drive in the good rock to the west of the fault, whereas the upper completions will provide volumetric sweep of the low-quality rock to the east of the fault. The water front from the lowest perforations penetrates through to the better zone west of the fault after approximately 40 days and then gradually builds up a water tongue that moves westward more rapidly along the bottom of the reservoir as seen in the two plots in the upper row of Figure 10.16. The advancing water front reaches the far west side of the reservoir after approximately 670 days and forms a cone that extends upward towards the horizontal producer. After 810 days, the water front breaks through in the midsection of the producer, and after 1,020 days, the whole well is engulfed by water. If the well had been instrumented with intelligent inflow control devices, the operator could have reduced the flow rate through the midsection of the well to try to delay water breakthrough. As the water is sucked up to the producer, it gradually forms a highly conductive pathway from the injector to the producer as seen in the snapshot from time 1,500 days shown at the bottom-left of Figure 10.16. A significant fraction of the injected water will therefore cycle through the water zone without contributing significantly to sweep any unproduced oil. Cycling water like this contributes to significantly increase the energy consumption and the operational costs of the production operation and is generally not a good production strategy.

Looking at the well responses in Figure 10.17, we first of all observe that the initial injection rate is very low because of the high viscosity of the resident oil. Thus, we need high injection pressure to push the first water into the reservoir. Once this is done, the injectivity increases steadily as more water contacts and displace a fraction of the oil. Because the reservoir rock and the two fluids are incompressible, increased injection rates give an equal increase in oil production rates until water breaks through after approximately 800 days. Since there is no heterogeneity to create pockets of bypassed oil, and residual saturations are zero, we will eventually be able to displace all oil by continuing to flush the reservoir with water. However, the oil rate drops rapidly after breakthrough, and increasing amounts of water need to be cycled through the reservoir to wash out the last parts of the remaining oil. By 4,500 days, the recovery factor is 73% and the total injected and produced water amount to approximately 2.6 and 1.8 pore volumes, respectively.

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

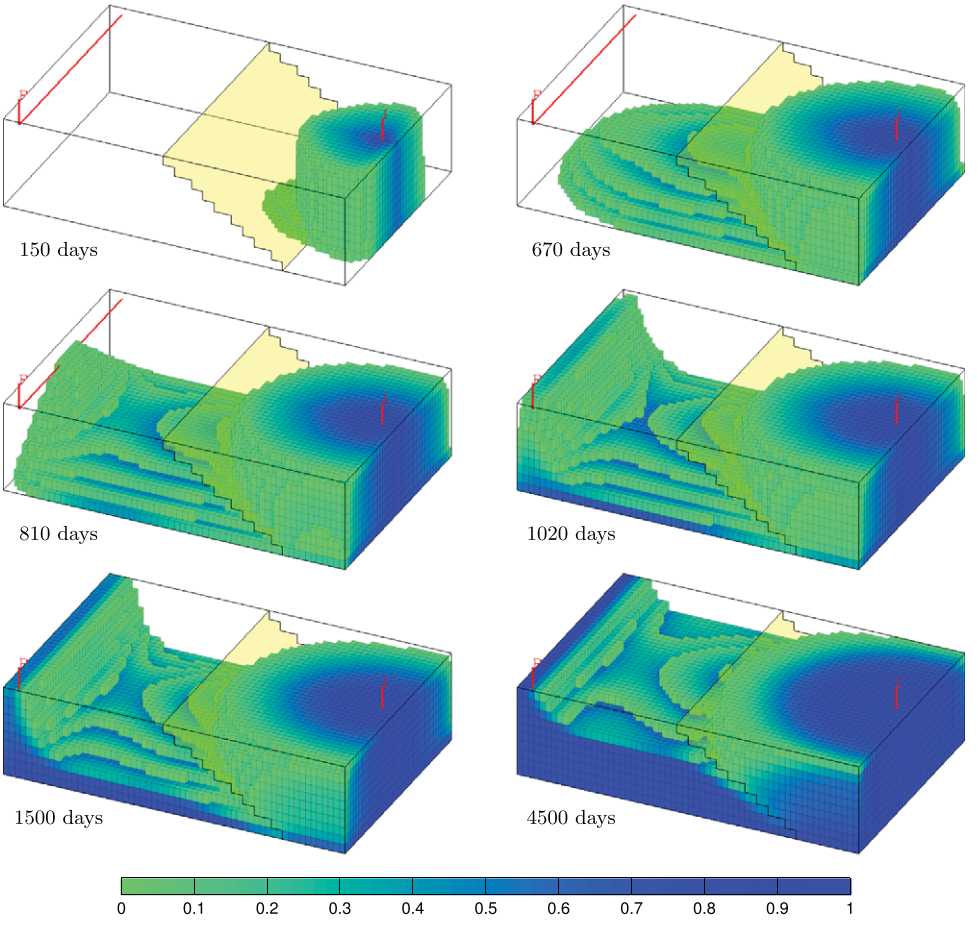


Figure 10.16 Evolution of the displacement profile for the water-coning case.

Another related problem is that of gravity override, in which a less dense and more mobile fluid flows preferentially above a denser and less mobile fluid. To illustrate this multiphase flow phenomenon, we consider a reservoir consisting of two horizontal zones placed on top of each other. Light fluid of high mobility is injected into the lower zone by a vertical well placed near the east side. Fluids are produced from a well placed near the west side, perforated in the lower zone only; see Figure 10.18. Densities of the injected and resident fluids are assumed to be 700 kg/m^3 and 1,000 kg/m^3, respectively. To accentuate the phenomenon, we assume that both fluids have quadratic relative permeabilities, zero residual saturations, and viscosities 0.1 cP and 1.0 cP, respectively. In the displacement scenario, 0.8 pore volumes of the light fluid are injected at constant rate. For simplicity, we refer to this fluid as water and the resident fluid as oil.

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

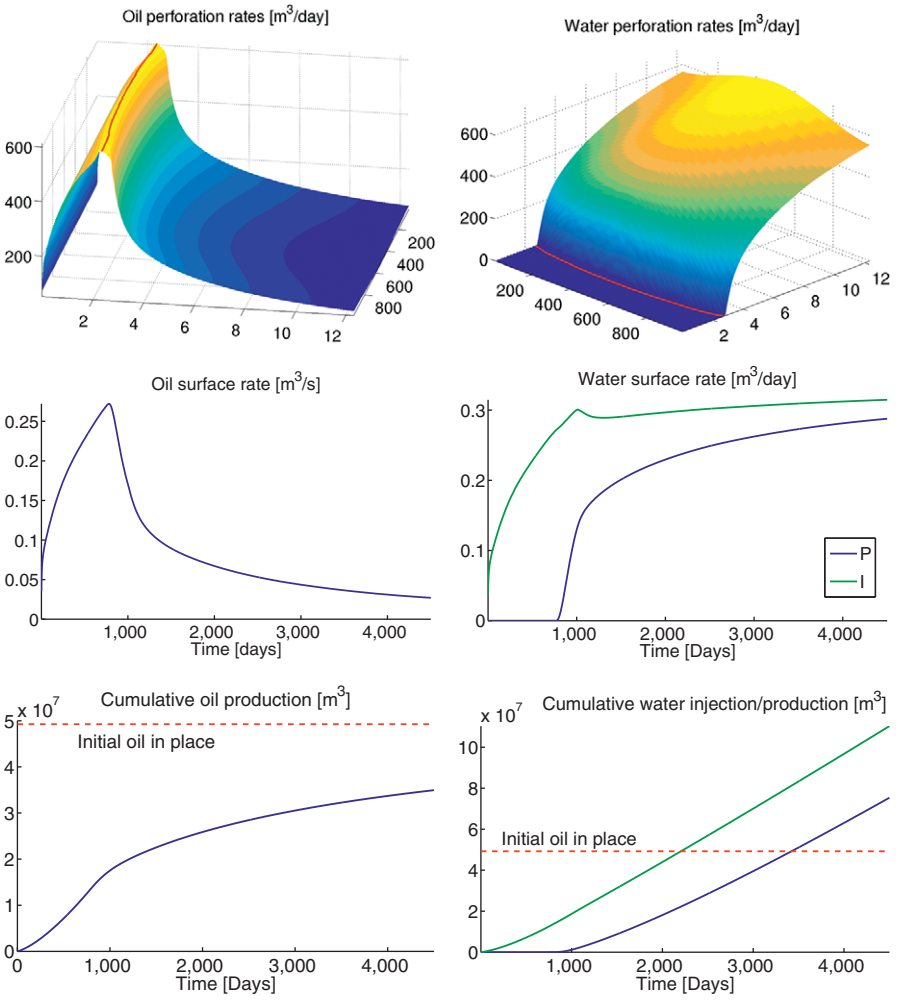


Figure 10.17 Well responses for the simulation of water coning. The top plots show rates in each perforation of the horizontal producer; the red lines indicate water breakthrough. The middle plots show total surface rates, whereas the lower plots show cumulative oil production and cumulative water injection and production.

We consider two different scenarios: one with high permeability in the upper zone and low permeability in the lower zone, and one with low permeability in the upper zone and high permeability in the lower zone. To simplify the comparison, the injection rate is the same in both cases. Figure 10.19 compares the evolving displacement profiles for the two cases. In both cases, buoyancy quickly causes the lighter injected fluid to migrate into the upper zone. With high permeability at the top, the injected fluid accumulates under the sealing top and flows fast towards the west boundary in the upper layers, as seen

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

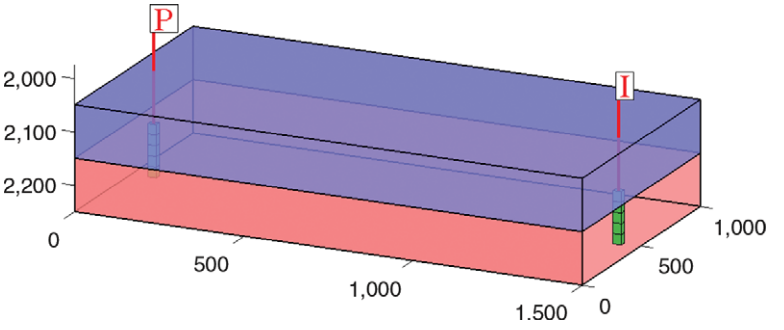


Figure 10.18 Setup for the case used to illustrate gravity override. Blue and red color indicate different permeabilities. Perforated cells are colored green.

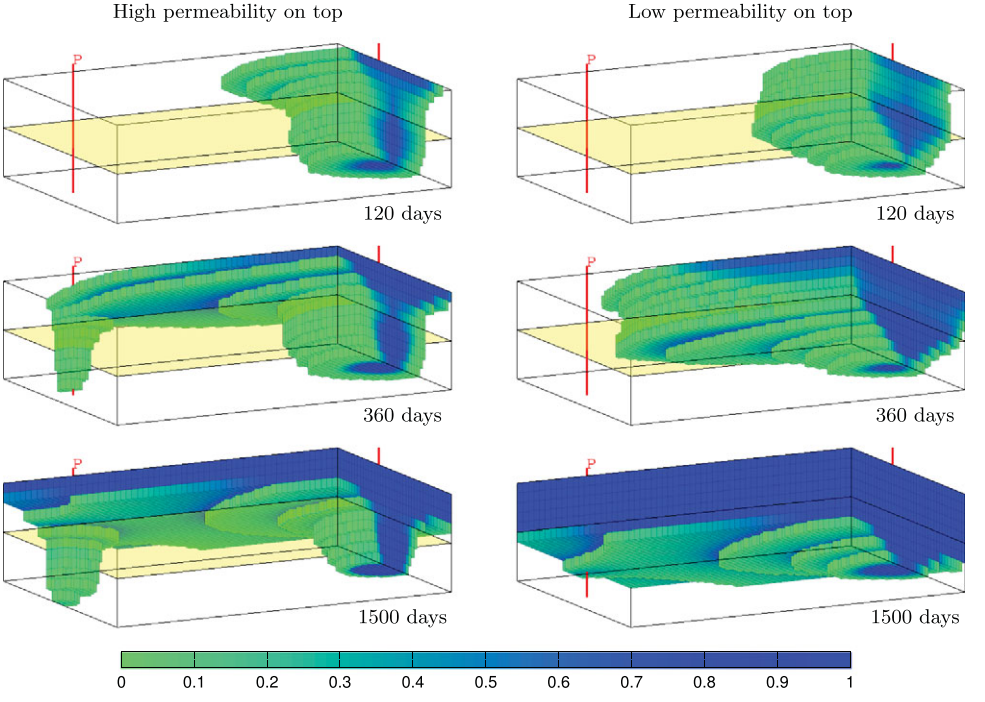


Figure 10.19 Evolution of the displacement profiles for the gravity-override setup. The reservoir is viewed from a position below and to the southeast of the reservoir. Cells containing only the resident fluid are not plotted.

in the upper-left plot of Figure 10.19. Looking at Figure 10.20, you may observe that since the production well is pressure-controlled and perforated in the low-permeability zone, the perforation rates will increase toward the top, i.e., the closer the perforation lies to the high-permeability zone above. As the leading displacement front reaches the producer

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

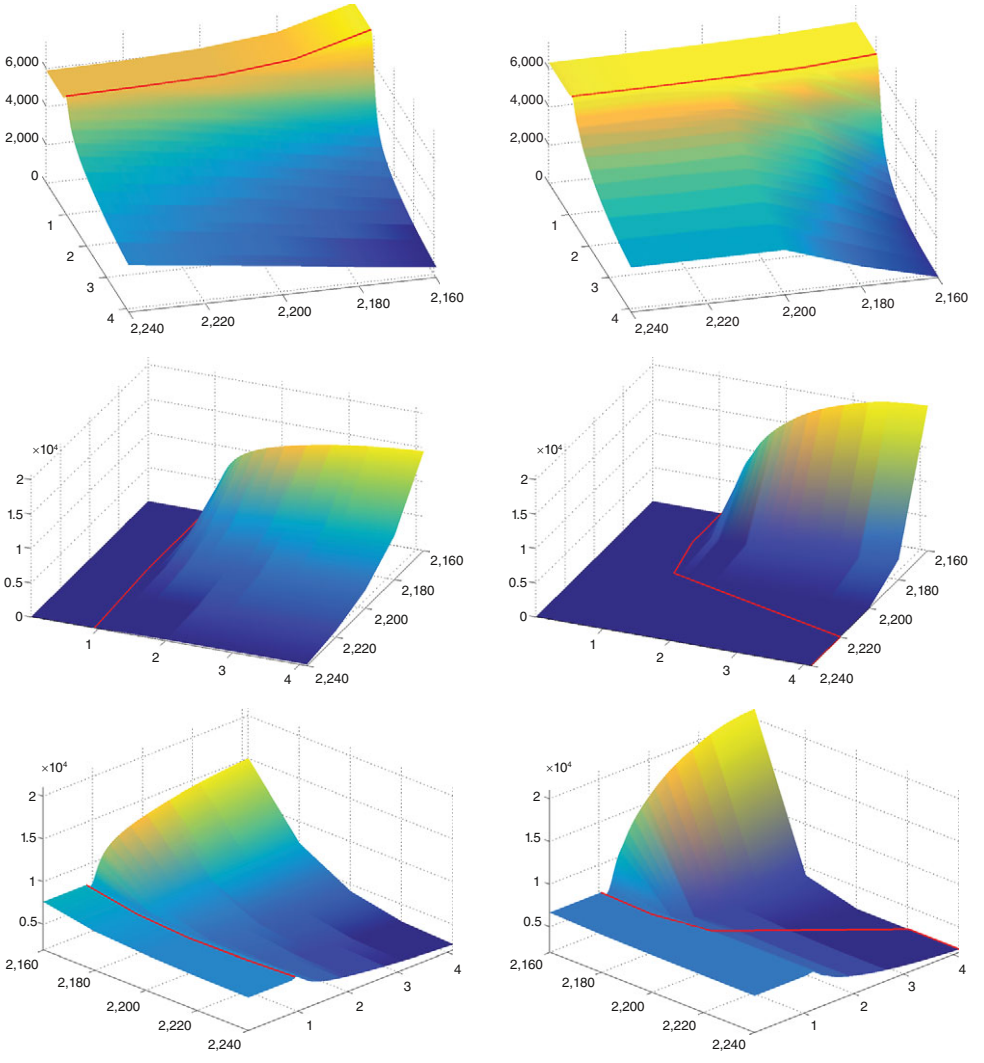


Figure 10.20 Perforation rates for the gravity-override case: oil (top), water (middle), and total rate (bottom). For the oil rate, the red lines indicate peak production, while they indicate water breakthrough in the plots of water and total rate. The left column reports the case with a high-permeability upper zone, and the right column the opposite case.

near the west boundary, it is sucked down toward the open perforations and engulfs them almost instantly, as seen after 360 days in the middle-left plot of Figure 10.19 and the red line in the lower-left plot of Figure 10.20. This causes a significant drop in oil rate and a corresponding increase in water rates towards the top of the well, which lies closer to the flooded high-permeability zone. The oil rate is also reduced in the lowest perforations, but

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

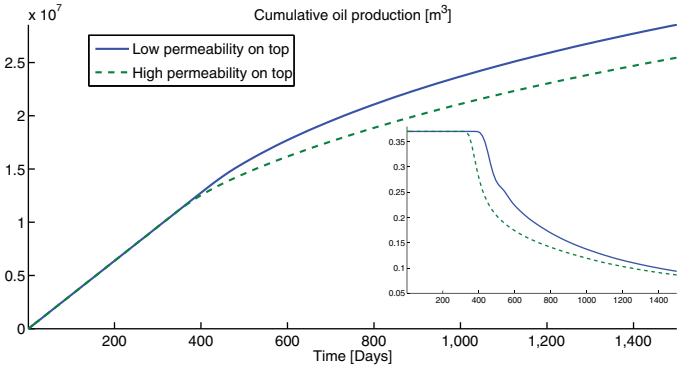


Figure 10.21 Cumulative oil production for the gravity-override case. The inset shows surface oil rates.

because these grid layers do not produce much of the injected fluid, the drop in oil rate is an effect of the internal adjustment of total perforation rates along the well, because mobility is so much higher near the top of the well. Indeed, some time after water breakthrough the oil rate is higher in the lower than in the upper perforations.

Also with a low-permeability zone on top, buoyancy will cause the injected fluid to migrate relatively quickly into the upper zone. However, because this case has better direct connection between the injector and producer through the high-permeability lower zone, the displacement front will move relatively uniformly through all layers of the top zone and the upper layer of the lower zone. The leading part of the displacement front is now both higher and wider and has swept a larger part of the upper zone by the time it breaks through in the producer. Unlike in the first case, the displacing fluid will not engulf the whole well, but only break through in the three topmost perforations. However, after breakthrough, most of the production comes from the topmost perforation, and the total rate in the three lowest perforations quickly drops below 20% of the rate in topmost perforation.

Finally, looking at cumulative oil production and surface oil rates reported in Figure 10.21, we see that low permeability on top not only delays the water breakthrough, but also enables us to maintain a higher oil rate for a longer period. This case is more economically beneficial than with high permeability on the top. To increase the recovery of the latter case, one possibility would be to look for a substance to inject with the displacing fluid to reduce its mobility in the upper zone.

10.3.7 The Effect of Capillary Forces – Capillary Fringe

As you may recall from Section 8.1.3, the pressure in a non-wetting fluid is always greater than the pressure in the wetting phase. In a reservoir simulation model, the capillary pressure – defined as $p_c = p_n - p_w$ in a two-phase system – has the macroscale effect of determining the local saturation distribution at the interface between the wetting and

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

non-wetting fluid, or in other words, there is a relation $p_c = P_c(S)$ between capillary pressure and saturation. This is seen in two ways: In a system that is initially in hydrostatic equilibrium, capillary forces enforce a smooth, vertical transition in saturation upward from a (horizontal) fluid contact. This transition is often referred to as the *capillary fringe*, and will be discussed in more detail in this section. The second effect is that capillary forces will redistribute fluids slightly near a dynamic displacement front so that this is not a pure discontinuity, as assumed in the hyperbolic models discussed in Chapter 9, but rather a smooth wave. For many field and sector models, the characteristic width of the transition zone is small compared to the typical grid size, and hence capillary forces can be safely neglected. In other cases, capillary forces have a significant dampening effect on the tendency for viscous fingering and are therefore crucial to include in the simulation model.

Returning to the formation of a capillary fringe near a fluid interface, let z_i denote the depth of the contact between the wetting and non-wetting fluid, and let $p_{n,i}$ and $p_{w,i}$ denote the phase pressures at this depth. The phase pressures and the capillary pressure are then given by

$$\begin{aligned} p_w(z) = p_{w,i} + g\rho_w(z - z_i), \qquad p_n(z) = p_{n,i} + g\rho_n(z - z_i) \\ p_c(z) = p_{c,i} + g\Delta\rho(z - z_i), \end{aligned} \tag{10.8}$$

where $\Delta\rho = \rho_n - \rho_w$ is density difference and $p_{c,i}$ is capillary pressure at z_i. This pressure is the capillary pressure necessary to initiate displacement of the wetting fluid by the non-wetting fluid and is called the *entry pressure*. It follows from (10.8), that the total height of the capillary fringe is given by $p_{c,n}/g\Delta\rho$. Figure 10.22 illustrates the concept of a capillary fringe for a case with zero residual saturations.

If we know the phase contact z_i, we can find the saturation directly by first computing the capillary force as function of depth using (10.8) and then using the capillary pressure

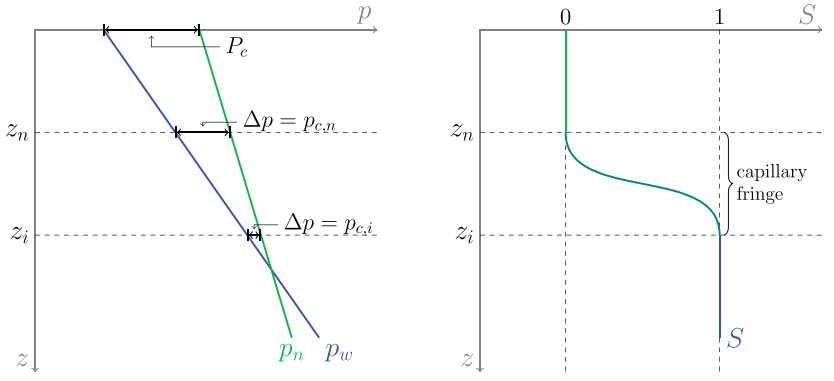


Figure 10.22 Diagrams showing phase and capillary pressures (left) and saturation (right) as function of depth. Here, z_i is the depth of the contact between the non-wetting and the wetting fluid, and z_n is the depth of pure non-wetting fluid.

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

function $P_c(S)$ to invert for saturation. Alternatively, if we only know the volume of the fluids, we can use the incompressible solvers to determine the hydrostatic fluid distribution. To illustrate, we consider a 100×100 m^2 vertical cross-section represented on a 20×40 grid. We assume a fluid system with CO_2 and brine having the same basic properties as in Section 10.3.2. With a density difference of approximately 290 kg/m^3, the capillary fringe corresponding to a capillary pressure of 1 bar has a height of 35 m. The relationship between saturation and capillary pressure depends on permeability and porosity, and to model this, we use the Leverett J-function (8.9). The `initSimpleFluidJfunc` implements a simplified Corey-type fluid in which $J(S) = 1 - S$. This gives

$$P_c(S) = \sigma\sqrt{\frac{\phi}{K}}(1 - S).$$

The surface tension σ is usually specific to the fluid. To illustrate capillary raise, we choose σ such that median rock properties give a capillary pressure of one bar,

```
fluid = initSimpleFluidJfunc('mu' , [0.30860, 0.056641]*centi*poise, ...
    'rho', [ 975.86,  686.54]*kilogram/meter^3, ...
    'n' , [      2,       2], ...
    'surf_tension',1*barsa/sqrt(mean(rock.poro)/(mean(rock.perm))),...
    'rock',rock);
```

We set initial data such that the rock is filled with half a pore volume of wetting fluid at the bottom and half a pore volume of non-wetting fluid at the top, and then simulate the system forward in time until steady-state is reached. The time-loop is set up with a gradual ramp-up of the time step to increase the stability of the solution procedure (we will come back to the choice of time step in Section 10.4.1),

```
dt = dT*[1 1 2 2 3 3 4 4 repmat(5,[1,m])]*year;
dt = [dt(1).*2.^[-5 -5:1:-1], dt(2:end)];
s  = xr.s(:,1);
for k = 1 : numel(dt),
   xr = incompTPFA(xr, G, hT, fluid);
   xr = implicitTransport(xr, G, dt(k), rock, fluid);
   t  = t+dt(k);
   if norm(xr.s(:,1)-s,inf)<1e-4, break, end;
end
```

We consider two different permeability setups. In the first case, the permeability field varies linearly from 50 md in the west to 400 md in the east. The porosity is assumed to be constant. When the system is released from the artificial initial state with a sharp interface at $z = 50$, the non-wetting fluid starts draining downward into the wetting phase, whereas the wetting phase starts imbibing upward into the non-wetting phase. After approximately 3.5 years, the system reaches the steady state shown in the middle plot in the upper row of Figure 10.23. Here, we say that steady-state is reached when the saturation difference between two consecutive time steps is less than 10^{-4} measured in the L^∞ norm. Because

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

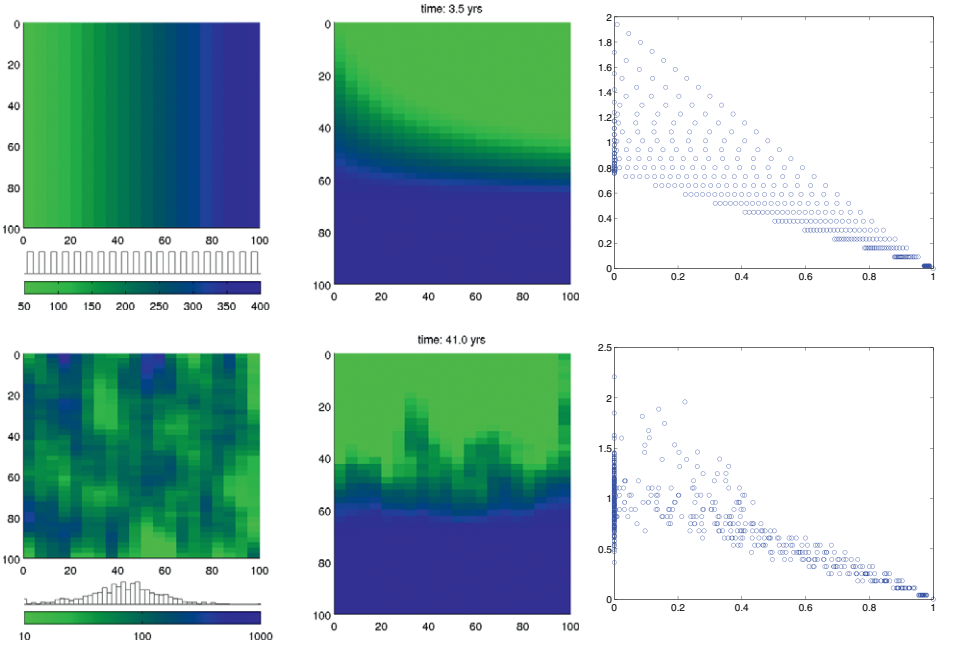


Figure 10.23 Capillary fringe for two different permeability fields: permeability increasing linearly from west to east (top), and lognormal permeability (bottom).

permeability is homogeneous in the vertical direction, the steady-state saturation decreases linearly upward from pure wetting to pure non-wetting fluid. The height of the capillary fringe is much higher in west where the permeability is low, since the capillary scales as $1/\sqrt{K}$, and lower in the east where the permeability is high. In the plot of capillary pressure versus saturation to the upper-right in Figure 10.23, you may also be able to identify the twenty different lines corresponding to the twenty columns of homogeneous permeability in the grid.

The second case has random petrophysical parameters with a permeability field that is related to a Gaussian porosity field through a Carman–Kozeny relationship; see the lower-left plot in Figure 10.23. The permeability values span three orders of magnitude, from 1 md to 1 darcy, which in turn gives a wider span in time constants than in the case with linear permeability. Because of the heterogeneities, the imbibing wetting phase will be sucked higher up and sideways into regions of lower permeability. If you run the script `capillaryColumn` yourself, you will see that the high-permeability regions surrounding the initial sharp interface reach equilibrium within a few years, whereas the rise is much slower in the low-permeability regions. The column next to the east boundary, in particular, is the last to reach steady state after approximately 41 years. At steady state, the fringe extends above the top of the reservoir section in the east-most column. Moreover, because permeability is heterogeneous, the saturation at steady state is no longer monotone in the vertical (and horizontal) direction.

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

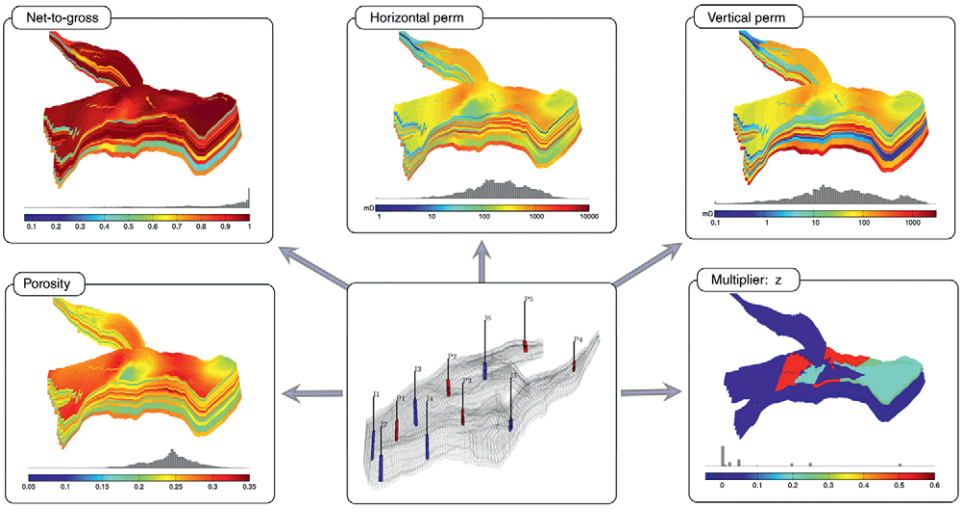


Figure 10.24 The Norne test case with grid and petrophysical data from the simulation model of the real field. The well pattern is artificial and has nothing to do with how the real field is operated.

10.3.8 Norne: Simplified Simulation of a Real-Field Model

Having worked with highly idealized models so far in this chapter, it is now time to look at a more realistic model. For this, we will use the grid geometry and petrophysical properties from a simulation model of the Norne field from the Norwegian Sea. More details about Norne and the reservoir geometry were given in Section 3.3.1. Figure 10.24 shows the petrophysical properties as well as a well-pattern that was chosen somewhat haphazardly for illustration purposes. We notice that the permeability is anisotropic and heterogeneous, with a clear layered structure. This layered structure is also reflected in the histograms, which show several modes. (Such histograms are discussed in more detail in Sections 2.5.3 and 2.5.5 for the SPE 10 and SAIGUP models.) The lateral permeability has four orders of magnitude variations, whereas the vertical permeability is up to two orders lower and has five orders of magnitude variations. In addition, the vertical communication is further reduced by a *multiplier field* (`MULTZ` keyword), which contains large regions having values close to zero in the middle layers of the reservoir. The porosities span the interval [0.094, 0.347], but since the model has a net-to-gross field to model that a portion of the cells may consist of impermeable shale, the effective porosity is much smaller in some of the cells. For a model like this, we thus cannot expect to be able to use the explicit transport solver and must instead rely on the implicit solver. (Note also that the `incomp` module contains a similar example with synthetic petrophysical properties; see `incompExampleNorne2ph`.)

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

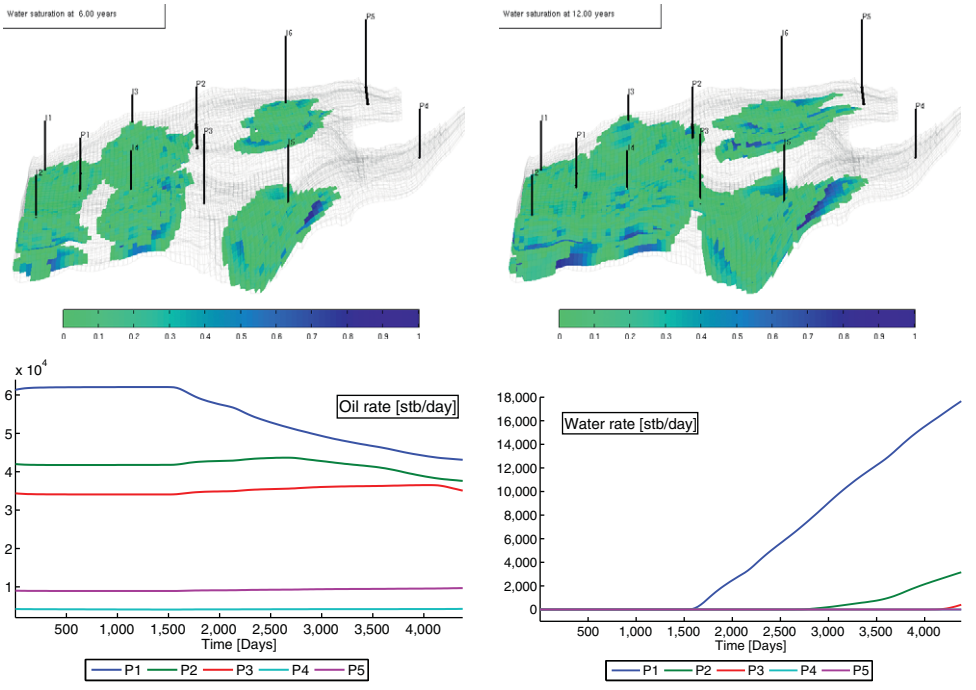


Figure 10.25 Incompressible two-phase simulation for the Norne model. The upper plots show snapshots of the solution (after 6 and 12 years, respectively) and the lower plots show oil and water surface rates for all producers.

Setting up the model proceeds as discussed previously; you can consult the `runNorne Simple` script in the `book` module for full details. The only difference is how to account for the transmissibility multipliers. This is done as follows:

```
hT = computeTrans(G, rock, 'Verbose', true);
tmult = computeTranMult(G, grdecl);
hT = hT.*tmult;
```

Here, the Eclipse input structure `grdecl` contains data for the `MULTZ` keyword. The second call takes the `MULTZ` values associated with cells and assigns a corresponding reduction value between 0 and 1 to all half-faces. We then multiply the half-transmissibilities `hT` by `tmult` to get the reduced transmissibility. It is important that these multipliers are assigned *before* computing the intercell transmissibilities. Altogether, approximately 6.5% of the half-faces have reduced transmissibility.

Figure 10.25 shows results of a simulation of a scenario in which water is injected into oil. The oil has a five times higher viscosity and hence the injected water will form an unstable displacement with a weak displacement front similar to the case discussed in Section 10.3.4. The main displacement takes place in the region that involves injectors I1

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

to I5 and producers P1–P3. The last two producers are located in regions that are poorly connected to the rest of the reservoir where the injectors are placed, and hence contribute less to the overall production. This is particularly true for producer P4, which most likely is completely misplaced. As we have seen in previous examples, once water breaks through in a well (primarily in P1 and P2), the oil rate decays significantly.

The main purpose of the example lies in the actual code and not in the results it produces. Discussing the simulation results beyond this point is therefore somewhat futile since the fluid system and the well pattern have limited relevance for the real reservoir, which contains a three-phase oil–gas–water system that is modeled by the compressible black-oil equations that will be discussed in the next chapter. The main takeaway message is that the incompressible solvers can be applied to models that have the geometrical and petrophysical complexity seen in real reservoir models.

Computer exercises

10.3.1 To get more acquainted with the multiphase incompressible solvers and see their versatility, you should return to a few examples presented earlier in the book and try to set them up as multiphase test cases:

- Consider the reservoir in Exercise 3.1.3 and place one injector to the south and two producers symmetrically along the northern perimeter. Simulate the injection of one pore volume of water into an oil.
- Consider the test case with non-rectangular reservoir geometry in Figure 5.6 on page 164 and set up a simulation that injects one pore volume from the flux boundary. How would you compute the flux out of the pressure-controlled boundary?
- Pick any of the faulted grids generated by the `simpleGrdecl` routine as shown in Figure 3.31 on page 97 and place an injector in one fault block and a producer in the other and simulate the injection of half a pore volume of water.

10.3.2 Consider a rectangular reservoir with two wells (see Figure 3.37) and compare solutions computed with three different grids: a uniform coarse grid, a uniform fine grid, and a coarse grid with radial well refinement.

10.3.3 Try to study the Norne model in more detail.

- Are the multipliers important for the simulation result?
- Is gravity important or can it be neglected?
- Can you come up with a better recovery strategy, i.e., improved placement and control strategy for wells?
- Do you get very different solutions if you use a consistent solver? (Hint: Although multipliers can be incorporated into these solvers, as described in [231], this is not part of the public implementation and for this comparison you should therefore neglect the `MULTZ` keyword.)

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

10.4 Numerical Errors

There are several errors involved in the computations in the previous section. First of all, numerical discretization errors obviously arise when approximating a continuous differential equation by a set of discrete finite-volume equations. For single-phase, incompressible flow, errors were purely spatial. These errors will decrease with decreasing size of the grid as long as the spatial discretization is consistent. However, as we saw in Chapter 6, the standard two-point scheme is not consistent unless the grid is strictly K-orthogonal, and the `incompTPFA` pressure solver can in general be expected to produce errors for anisotropic permeabilities and skewed grids. When using a sequential method to solve multiphase flow equations, there will also be *temporal errors* arising from three different factors: discretization errors arising when temporal derivatives in the transport equations are discretized by finite differences, amplifications of spatial errors with time, and errors introduced by the operator splitting underlying the sequential solution procedure. In this section, we briefly discuss the two last error types in more detail.

10.4.1 Splitting Errors

When using a sequential solution procedure, the total velocity is computed from the fluid distribution at the start of each time step. This means that the effect of mobility on the flow paths is frozen in time, and for each time step appears as if we solved a single-phase flow problem with reduced permeability in all parts of the domain that contain more than one fluid phase. Within a single splitting step the transport solver will thus only resolve the dynamic effect of mobility along each flow path, but will not account for the fact that mobility changes reduce the effective permeability along each flow path or move the flow paths themselves. This introduces a time lag in the simulation, which may lead to significant errors in the propagation of displacement fronts if the splitting steps are chosen too large.

Homogeneous Quarter Five-Spot

To illustrate this, we can revisit the homogeneous quarter five-spot from Section 10.3.3 and study the self-convergence of approximate solutions defined on a fixed grid as the number of splitting steps increases. Figure 10.26 shows approximate solutions at time $t = 0.6$ PVI (pore volumes injected) computed with 4^ℓ steps for $\ell = 0, \ldots, 3$. With a single pressure step, the displacement front coincides with the time line from the single-phase flow field, since the pressure computation only sees the initial oil saturation. The only exception is a certain smearing introduced by the spatial and temporal discretizations of the explicit scheme used to compute the transport step. The retardation effect that oil has on the invading water is better accounted for as the number of splitting steps increases, and hence the splitting solution gradually approaches the correct solution. Table 10.1 reports the self-convergence towards a reference solution computed on the same grid with 256 splitting steps. To estimate the convergence rate we assume that the error scales like $\mathcal{O}(\Delta t^r)$. If the

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

Table 10.1 *Self-convergence for the homogeneous quarter five-spot computed on a* 128×128 *grid with n splitting steps relative to a reference solution computed with 256 time steps after 0.6 PVI. The errors are reported in relative norms.*

	saturation		pressure	
n	L^1-error	rate	L^2-error	rate
1	5.574e−02	–	4.950e−03	–
2	4.368e−02	0.35	5.140e−04	3.27
4	2.778e−02	0.65	1.554e−04	1.73
8	1.445e−02	0.94	4.524e−05	1.78
16	6.389e−03	1.18	1.226e−05	1.88
32	2.394e−03	1.42	2.998e−06	2.03
64	7.869e−04	1.61	5.990e−07	2.32

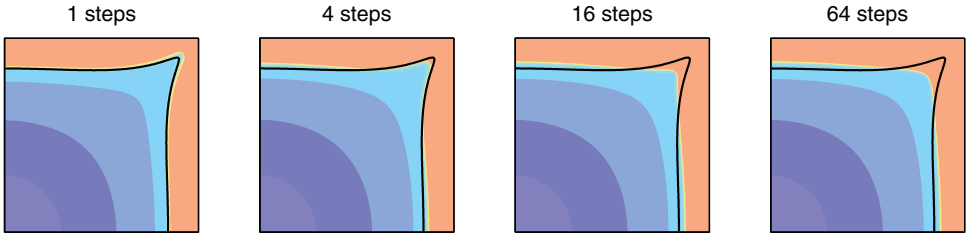


Figure 10.26 Quarter five-spot solution at time 0.6 PVI computed on a uniform 128×128 grid with the explicit transport solver and different number of splitting steps. The solid lines are time lines at $t = 0.6a$ for a single-phase displacement.

solution having error E_2 is computed using twice as many time steps as the solution having error E_1, the corresponding convergence rate is

$$r = \log(E_1/E_2)/\log(2).$$

Pressure is smooth and will therefore converge faster than saturation, which is a discontinuous quantity and hence will have much larger errors. The convergence for high n values is exaggerated since we are measuring self-convergence toward a solution computed with the same method, but with a larger number of steps. The code necessary to run this experiment is found in `splittingErrorQ5hom.m` in the `in2p` directory of the `book` module.

Heterogeneous Quarter Five-Spot

As pointed out earlier, the coupling between the pressure and transport equations depends on the variation in total mobility $\lambda(S)$ throughout the simulation. If variations are small and smooth, the two equations will remain loosely coupled, and relatively large time steps can

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

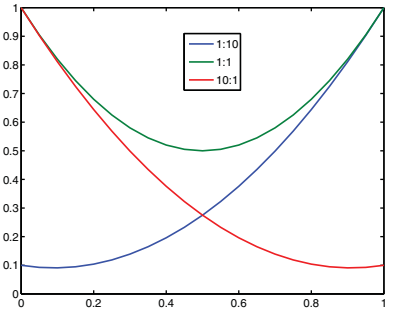


Figure 10.27 Total mobility for three fluid models with different viscosity ratios.

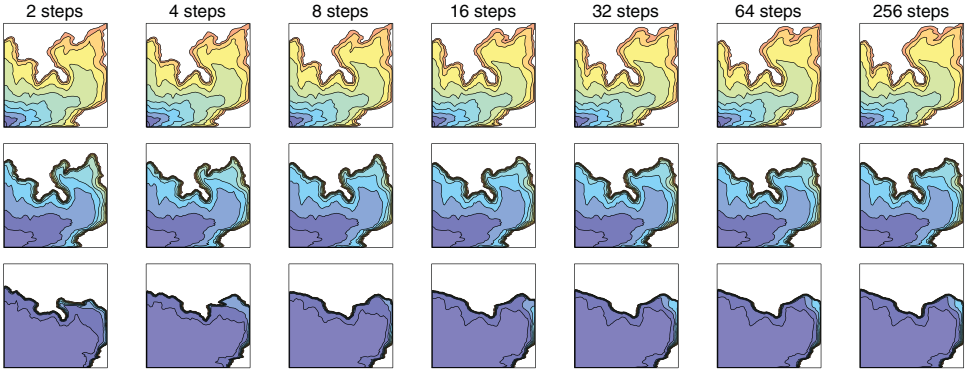


Figure 10.28 Self convergence with an increasing number of equally spaced splitting steps to reach time 0.5 PVI for a quarter five-spot setup on a subsample from the first layer of the SPE 10 model for three different viscosity ratios $M = \mu_w/\mu_n$ (top: M = 1/10, middle: M = 1, bottom: M = 10). Saturation profiles are shown at a time scaled by $v(1)/v(M)$, where $v(M)$ is the characteristic wave speed of the displacement front with viscosity ratio M.

be allowed without seriously decaying solution accuracy. On the other hand, when λ has large variations over the interval $[0, 1]$, pressure and transport are more tightly coupled, and we cannot expect to be able to use large splitting steps. To illustrate this, we revisit the setup with three different fluid models used to study viscous fingering in Section 10.3.4. From the plot of total mobilities in Figure 10.27 it is obvious that both the unfavorable (1:10) and the favorable (10:1) mobility cases have stronger coupling between saturation and pressure than the case with equal viscosities.

Figure 10.28 shows the self convergence of the saturation profiles with respect to the number of equally spaced time steps used to reach time 0.5 PVI. To isolate the effect of splitting errors and avoid introducing excessive numerical smearing in the solutions with few splitting steps, we have subdivided the transport steps into multiple steps so that the implicit solver uses the same step length and hence introduces the same magnitude of

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

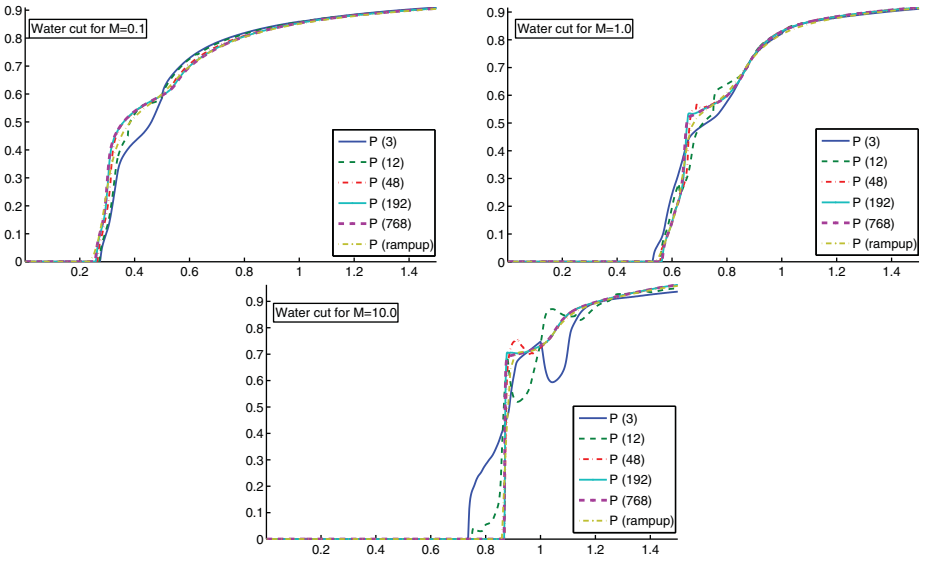


Figure 10.29 Water cut in the producer for various number of splitting steps for the heterogeneous quarter five-spot setup from Figure 10.28.

numerical smearing in all simulations. From the figure it is clear that even with very few splitting steps, the sequential solution method manages to capture the qualitatively correct behavior for all viscosity ratios. As expected, the discrepancies in solutions with few and many time steps are larger for the favorable and unfavorable cases than for unit viscosity ratio.

To investigate how the size of the splitting steps affects the *quantitative* behavior of the approximate solutions, we reran the same experiments up to 1.5 PVI. Figure 10.29 reports water cut in the producer for all three fluid models. Starting with the unfavorable case, we see that the water cut has a dent. This is a result of the secondary finger that initially extends along the western edge, making contact with the main finger and hence contributing to a more rapid incline in water production. All curves, except the one using only three time steps, follow the same basic trend. The main reason is that small variations in the saturation profile will not have a large effect on the water cut since the displacement front is so weak. With three splitting steps only, the main dent is a result of the second pressure update. For unit viscosity ratio, the production curves are still close, but here we notice a significant incline in the curve computed with 12 steps after the pressure update at 0.75 PVI. The 12-step and 48-step curves also show non-monotone behavior at 0.625 PVI and 0.69 PVI, respectively. Similar behavior can be seen for the favorable case, but since this case has an almost piston-like displacement front, the lack of monotonicity is significantly amplified. For comparison, we have also included a simulation with 96 splitting steps, in which the first two first steps have been replaced by ten smaller splitting

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

steps that gradually ramp up to the constant time step. These profiles, and similar profiles run with 48 steps, are monotone, which suggests that the inaccuracies in the evolving saturation profiles are introduced early in the simulation when the profile is rapidly expanded by high fluid velocities in the near-well region. In our experience, using such a ramp-up is generally advisable to get more well-behaved saturation profiles. You can find the code for this experiment in `splittingErrorQ5het.m` in the `in2p` directory of the `book` module.

Capillary-dominated flow

The sequential solution procedure discussed in this chapter is as a general rule reasonably well-behaved for two-phase scenarios where the fluid displacement is dominated by the hyperbolic parts of the transport equation, i.e., by viscous forces (pressure gradients) and/or gravity segregation. If the parabolic part of the solution dominates, on the other hand, a sequential solution procedure will struggle more, in particular when computing fluid equilibrium govern by a delicate balance between gravity and capillary forces. To illustrate this, we revisit the computation of capillary fringe from Section 10.3.7. If you look carefully in the accompanying code, you will see that we compute the two cases with a time step that is ten times larger for the Gaussian case than for the case with linear permeability. The time steps (and the initial ramp-up sequence) were chosen by trial and error and are close to what appears to be the stability limit. If one, for instance, increases the final time steps by 150% for the case with linear permeability, the simulation will not converge but instead ends up predicting an oscillatory interface, as illustrated in Figure 10.30. Similar problems may arise, e.g., when simulating structural trapping of CO_2 using vertical equilibrium models, in which gravity gives rise to a parabolic term that plays the same role as capillary pressure in the upscaled flow equations; see e.g., [227] for more details.

10.4.2 Grid Orientation Errors

As you may recall from the discussion on page 287, the saturation-dependent mobility at the interface between two grid blocks is usually approximated by single-point, upstream mobility weighting. Like the TPFA method, the resulting scheme only accounts for information on opposite sides of a cell interface and does not take any transverse transport effects into account. It is therefore well known that this method also suffers from grid orientation errors, especially when applied to unfavorable displacements, as we will see in the following example. In passing, we also note that to evaluate gravity and capillary-pressure terms, the transport solvers use two-point approximations similar to what is used in the TPFA method. This may introduce additional errors, but we will not discuss these in detail herein.

Homogeneous Quarter Five-Spot

When the single-point transport solver is combined with the classical TPFA pressure solver, computed displacement fronts tend to preferentially move along the axial directions of the

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

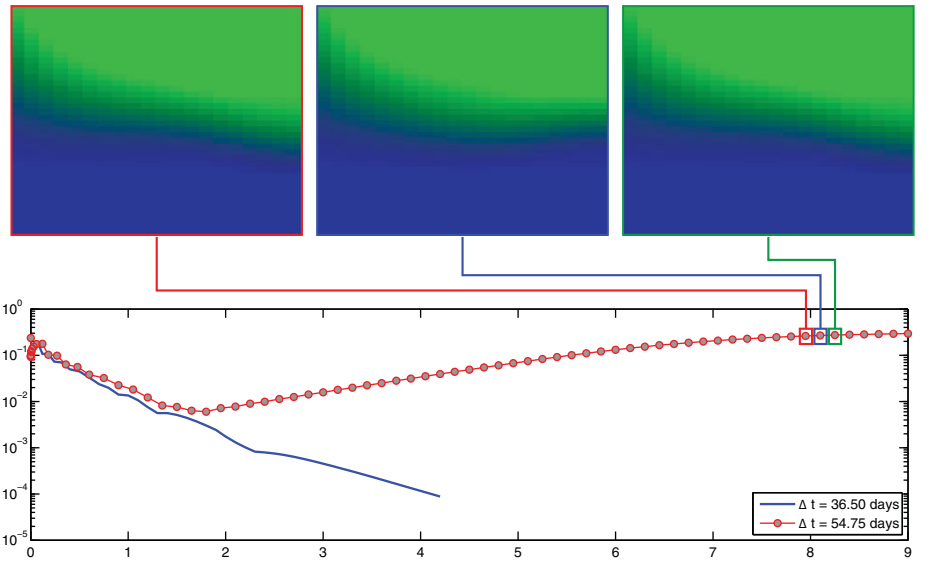


Figure 10.30 Unstable solution computed by a sequential solution with too large splitting step. The lower plot shows difference in L^{∞} norm between consecutive time steps – which is used as convergence criterion by the solver – for a convergent solution with $\Delta t = 36.5$ days and for a divergent solution with $\Delta t = 54.75$ days. The upper plot shows three consecutive solutions for the divergent simulation.

grid, i.e., in the direction of the normal vectors of the cell faces. This will lead to grid orientation effects like those discussed earlier in Chapter 6, even if the resulting grids are K-orthogonal. To illustrate this, we compare and contrast solutions of the standard quarter five-spot setup with a rotated setup in which the grid is aligned with the directions between injectors and producers, as illustrated in Figure 10.31. This test problem was first suggested by Todd et al. [290] and has later been used by many other authors to study grid orientation errors in miscible displacements [321, 255, 279], which are particularly susceptible to this type of truncation error. To avoid introducing too much diffusion when using few time steps, we use the explicit transport solver. (See `runQ5DiagParal` for complete setup of the two cases and `gridOrientationQ5` for the following experiments.)

For the standard setup, the combination of a single-point transport solver and a two-point pressure solver overestimates the movement into the stagnant regions along the x and y axes and underestimates the diagonal movement in the high-flow direction between injector and producer along the diagonal. In the rotated setup, the grid axes follow the directions between the wells and the solvers will hence tend to overestimate flow in the high-flow zone and underestimate flow toward the stagnant zones. The upper row in Figure 10.32 shows that this effect is pronounced for the unfavorable displacement ($M = 0.1$), evident with equal viscosities ($M = 1$), and hardly discernible for the favorable displacement ($M = 10$). For equal viscosities, the difference between the two grids can be almost

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

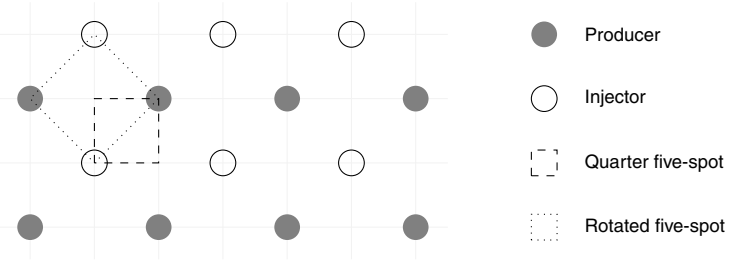


Figure 10.31 Well setup for the quarter five-spot comparison. Displacement fronts have preferential movement parallel to the axial directions and hence the rotated setup will predict earlier breakthrough than the original setup.

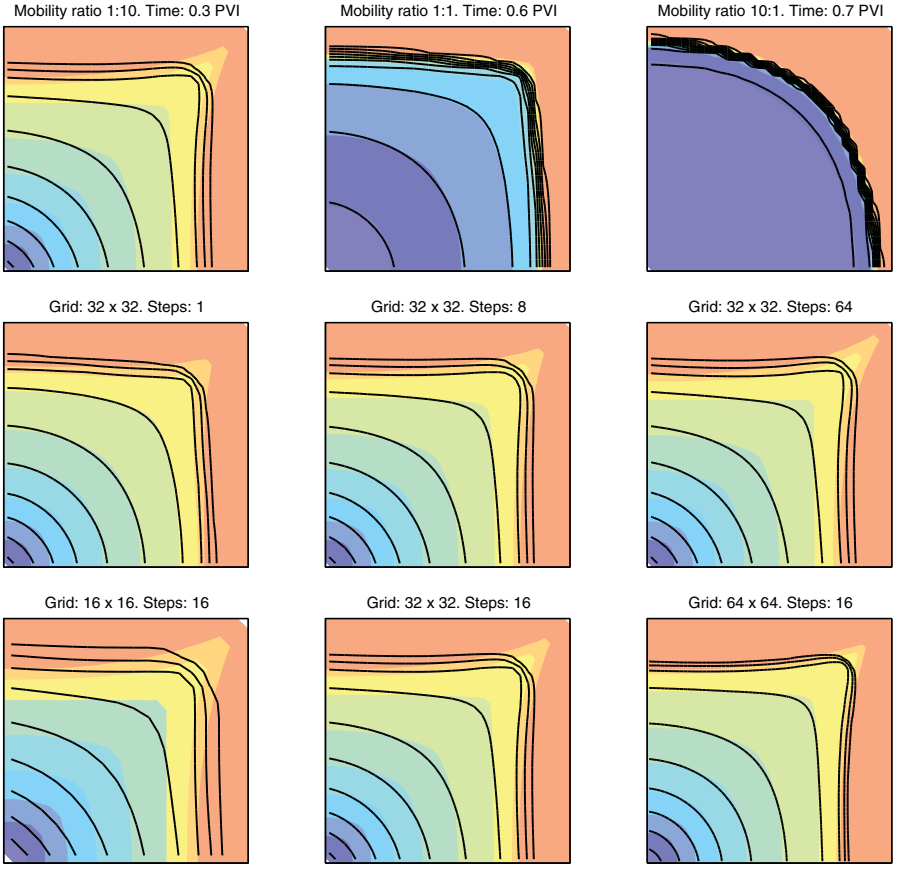


Figure 10.32 Quarter five-spot solutions computed on the rotated (colors) and original (solid lines) geometry for a 32×32 grid. The top row shows solutions computed with 16 steps for different mobility ratios. The middle plot shows convergence for mobility ratio 1:10 with respect to time step and the lower row with respect to grid resolution.

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

eliminated by increasing the number of splitting steps for a fixed Δx and/or by increasing the grid resolution provided that a certain number of time steps are used. In the unfavorable case, the plots in the middle row of Figure 10.32 show that increasing the number of time steps changes both solutions in the same direction, but does not necessarily reduce their difference. However, since both the original and the rotated grid are regular, the grid orientation effects will diminish if we increase the grid resolution.

A possible remedy to the behavior we just observed is to replace the single-point scheme by a multidimensional upwind scheme (see e.g., [155]) or a modern high-resolution scheme. Such methods are not yet part of the public MRST release, but work is currently in progress [190].

Symmetric Well Pattern on a Skew Grid

We have already seen several times that since the TPFA scheme cannot approximate transverse fluxes that are parallel to grid interfaces, the `incompTPFA` solver will introduce grid orientation errors for anisotropic permeabilities and grids that are not K-orthogonal. In Chapter 6, we showed that we can significantly reduce, but not completely eliminate, these errors if we replace TPFA by a consistent scheme. In this example, we will revisit one of the cases discussed in Chapter 6 to investigate if we see the same reduction for two-phase flow.

The computational setup consists of a horizontal 400×200 m^2 sector model. Water is injected from at the midpoint of the northern perimeter and fluids are produced from two wells located 50 m from the southeast and southwest corners, respectively. Since the well pattern is symmetric within a confined domain and the petrophysical parameters are homogeneous and isotropic, the true displacement profile will also be symmetric. The grid, however, is skewed and compressed towards the southeast corner. This will induce a preferential flow direction towards the southeast producer. Seeing that grid orientation effects are more pronounced for unstable displacements, we use the same configuration as in the previous example with a viscosity ratio $M = 10$, which gives a very mobile, weak displacement front that will tend to finger rapidly into the resident oil. Figure 10.33 shows a snapshot of the pressure and saturation profile after 250 days along with water cuts and cumulative oil production over the whole 1,200-day simulation period. The flow field computed with the `incompTPFA` pressure solver exhibits the same lack of symmetry as seen in Figure 6.5 on page 187. The result is a premature breakthrough in the southeast producer and delayed breakthrough in the southwest producer. With a mimetic finite-difference (MFD) solver, the water-cut curves are much closer and less affected by grid orientation errors. Moreover, since all wells operate under pressure control, we see that the total oil production predicted by TPFA is significantly less than for the MFD solver.

As a second example, we use the same grid to describe a vertical cross-section, in which we inject water from two horizontal injectors at the bottom of the reservoir and produce fluids from a horizontal producer at the top of the reservoir. Producers and injectors operate under the same pressure control as for the horizontal reservoir section. Figure 10.34 shows

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

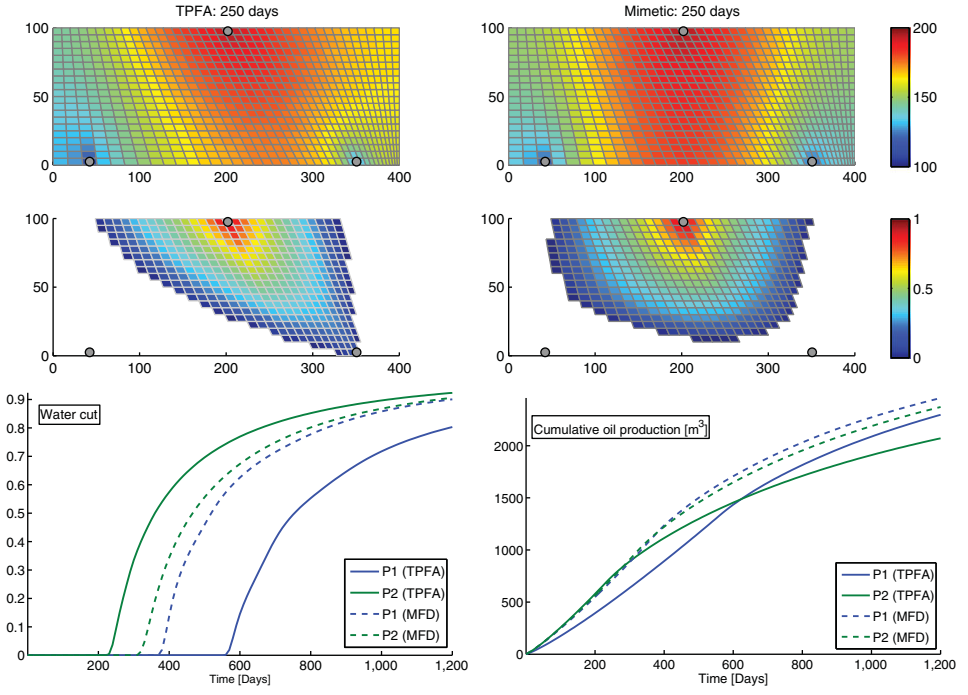


Figure 10.33 Simulation of a symmetric flow problem in a horizontal, homogeneous domain represented on a skew grid.

simulation results with the TPFA and MFD solvers. The density difference between the injected and resident fluids is 150 kg/m^3, and hence gravity tends to oppose the imbibing water front in a way that accentuates the grid orientation effects for both solvers. For comparison, Figure 10.35 reports the simulations performed on a regular Cartesian grid. Both schemes produce symmetric, but slightly different displacement profiles and the match in production profiles is largely improved compared with the skew grid.

Altogether, the two examples presented in this section hopefully show you that you not only need to take care when designing your grid, but should also be skeptic to simulations performed by a single method or a single choice of time steps. A good piece of advice is to conduct simulations with more than one scheme, different time-step selection, and, to the extent possible, different grid types to get an idea of how numerical errors influence your results. To further investigate grid orientation effects, consider any of the following computer exercises.

Computer exercises

10.4.1 Repeat the test case with the original/rotated quarter five-spot using one of the consistent solvers from Chapter 6 to compute the pressure. Do you see any differences?

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

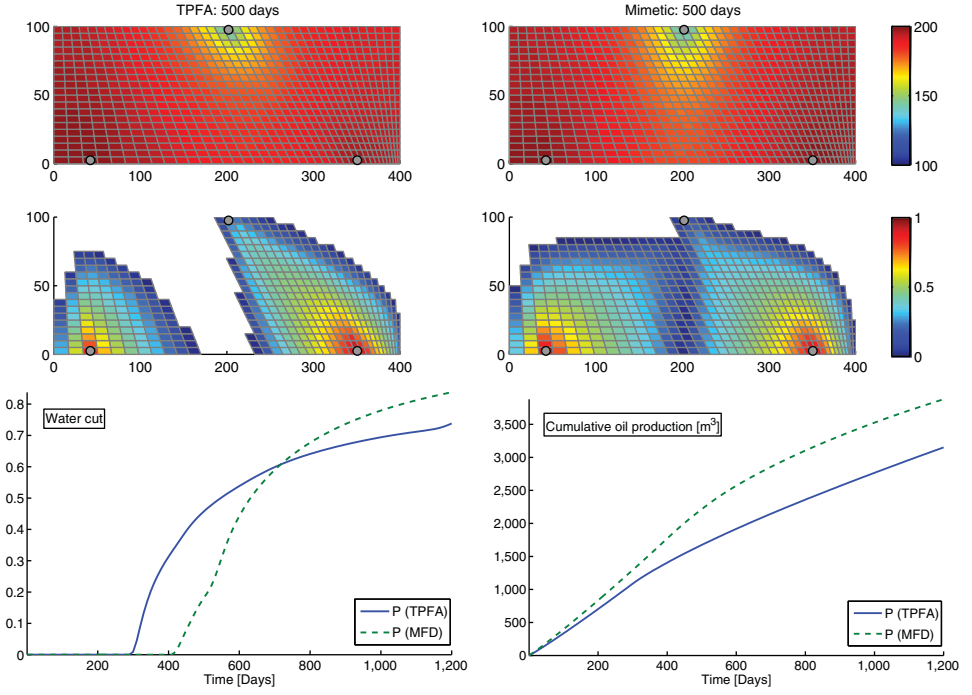


Figure 10.34 Simulation of a symmetric flow problem in a vertical, homogeneous domain represented on a skew grid with two horizontal injectors at the bottom and a horizontal producer at the top.

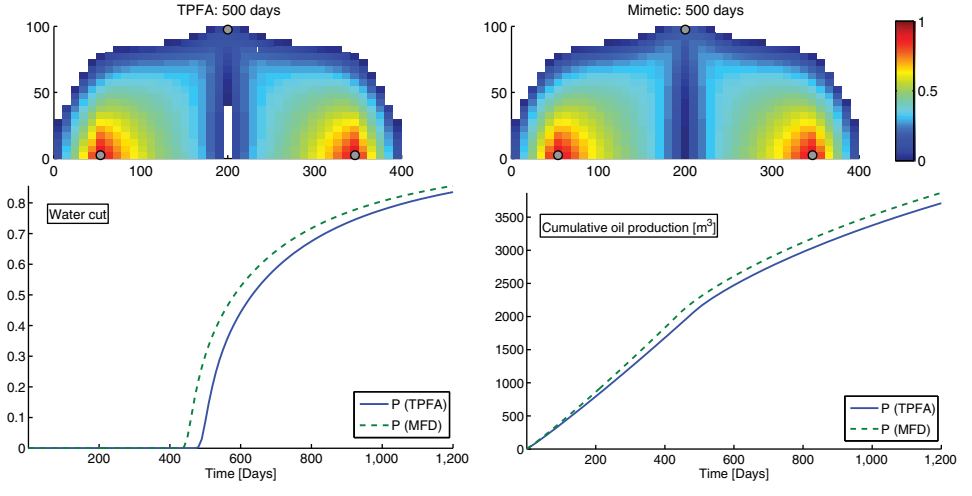


Figure 10.35 Simulation of a symmetric flow problem in a vertical, homogeneous domain computed on a regular Cartesian grid.

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press

10.4.2 Set up a flow problem to compare solutions computed on the extruded Delaunay or Voronoi grids shown in Figure 3.32 on page 97.

10.4.3 Set up a quarter five-spot test using the grids shown in Figure 3.44 on page 108.

10.4.4 The `CaseB4` test case shown in Figure 3.20 on page 80 represents the same geology with two different grid formats, using either a deviated pillar grid or a stair-stepped grid. Both grids are sampled at two different resolutions. Set up a flow problem with four wells, two injectors and two producers, one in each corner of the reservoir. Run simulations on all four grids and compare production curves. You can either use homogeneous permeability or a layered permeability with homogeneous properties within each layer.

10.4.5 Pick any of the models in the `bedModels1` or `bedModel2` data sets and run two-phase simulations injecting one pore volume from south to north and from west to east. Investigate splitting and grid orientation effects.

https://doi.org/10.1017/9781108591416.014 Published online by Cambridge University Press