8

Compositional Simulation with the AD-OO Framework

OLAV MØYNER

Abstract

The compositional module in the MATLAB Reservoir Simulation Toolbox (MRST) implements two different formulations of a three-phase compositional system that consists of a pair of multicomponent phases and an optional immisicible phase. In petroleum engineering, the aqueous phase is taken to be immiscible and the hydrocarbon liquid and vapor phases are governed by an equation of state (EoS). The overall composition formulation uses pressure and overall mole fractions as primary variables, whereas the natural variable formulation relies on solving for phase mole fractions and phase saturations simultaneously. Thermodynamic behavior is modeled using K-values or a (standard) cubic EoS. In the chapter, you will learn about the model equations, choice of primary variables, and numerical strategies for solving the thermodynamic problem, alone or coupled to the flow equations. We discuss details of the implementation, which builds upon the object-oriented, automatic differentiation (AD-OO) framework and utilizes state functions and generic model classes for increased modularity. We also present a few relatively simple simulation examples to illustrate typical behavior and teach you how to set up simulation cases yourself.

8.1 Introduction

The predominant approach to simulate recovery of hydrocarbons from the subsurface has been to use a black-oil-type model in which chemical hydrocarbon species are lumped into into two pseudocomponents, gas or oil, depending on whether the species are found in gaseous or liquid form at standard surface conditions. At reservoir conditions, the two pseudocomponents generally form a gaseous/vapor phase and an oleic liquid phase. To describe the pressure–volume–temperature (PVT) behavior of the two fluid phases and the distribution of the oil and gas

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

components across these two phases one uses correlations and/or (tabulated) data, which are interpolated as functions of pressure. In the MATLAB Reservoir Simulation Toolbox (MRST), such models have been implemented in the `ad-blackoil` module, which constitutes an important part of the object-oriented, automatic differentiation (AD-OO) simulator framework, as described in detail in the first MRST textbook [21].

For many recovery processes, and miscibility flooding in particular, black-oil-type models are not sufficient to describe important flow physics. Instead, one needs to represent how individual hydrocarbon species (or a larger set of lumped species, and possibly other nonhydrocarbon reservoir gases like carbon dioxide, nitrogen, and solvents) flow as two or more hydrocarbon phases and exchange components across these phases. In compositional simulation, one therefore simultaneously studies the coupled process of multiphase flow, multicomponent transport, and thermodynamics at the high-pressure, high-temperature conditions typically found in the hydrocarbon-bearing layers of the subsurface. Compositional flow simulation is also needed in other subsurface applications like CO_2 storage, geothermal energy, hydrosystem engineering, subsurface contamination and remediation, etc., particularly when accurate description of dissolution and mixing processes is important to understand and predict the flow system.

Compositional simulation generally uses an equation of state to compute equilibrium compositions and densities of the individual chemical components that make up the fluid system. Once these are known, other necessary fluid properties can be computed using known correlations and/or interpolated as a function of pressure from tabulated black-oil fluid models. As such, compositional simulation is the most mechanistically correct description of a multiphase, multicomponent system. Simpler models like black-oil and immiscible flow equations can be shown to be special cases that can be represented within the general compositional framework. One particular challenge with black-oil-type models is that one can easily end up with an inconsistent and incorrect description of the phase behavior if PVT properties are extrapolated outside their intended domain of validity (see, e.g., subsection 11.8.3 in the MRST textbook [21]). Provided that the equation of state is properly tuned to experimental data, a compositional model can predict the correct phase behavior, for both miscible and immiscible conditions. By utilizing the optional prediction of densities and phase behavior from an equation of state (EoS), it is generally easier to formulate a consistent and accurate description of a complex multiphase flow problem using a compositional formulation than with a black-oil equation. Compositional simulations are thus often used to run fine-scale reference simulations to generate the tabulated data necessary for the simplified PVT description of a black-oil model. In many cases, compositional simulators also include the effects of interfacial tension, molecular diffusion, convective dispersion, and nonisothermal conditions.

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

The improved physical realism of a compositional simulation has a price. As you will see in some of the examples discussed later in this chapter, compositional systems tend to contain a complex combination of waves (shocks, rarefactions, and contact discontinuities) that must be resolved to predict displacement and fluid injection/recovery accurately. Among these, it is particularly challenging to resolve linear or weakly nonlinear waves, which typically require (very) high grid resolution (or higher-order discretizations) to avoid excessive smearing, as also discussed in Chapter 7. The second disadvantage of compositional simulators is that they are computationally expensive. Even without extra grid resolution, a compositional simulation will generally contain many more unknowns per cell than a black-oil simulator and thus incur significantly higher computational cost. Coupling of flow and thermodynamics can also introduce severe nonlinearities that result in slow iterative convergence. It is therefore important to develop numerical formulations that reduce the number of nonlinear iterations and their cost as much as possible. A key to this end is the choice of primary unknowns, for which there are many possibilities.

The compositional module in MRST implements a standard three-phase compositional model, consisting, by default, of two multicomponent phases (vapor and liquid) and one immiscible phase. In this chapter, we will introduce you to the governing flow and thermodynamics equations for the two-phase multicomponent part and discuss the numerical procedures used to solve the thermodynamic equations both as a standalone problem and as part of a coupled flow–thermodynamics system. We also present a set of illustrative numerical examples and describe how the equations and numerical algorithms are implemented in the AD-OO framework of MRST using the generic model classes and the state functions introduced in Chapter 5. To benefit from the discussions of implementation details, you should be familiar with the AD-OO framework and conventions used therein and preferably have read Chapters 8, 11, and 12 from the MRST textbook as well as chapter 5 herein.

8.2 Governing Equations

A compositional model for multiphase flow enables you to track numerous species that make up different fluid phases. In the following description, we limit our discussion to the case of two-phase liquid–vapor flow, for which a system of N individual species is described. We note that the MRST implementation also supports the optional inclusion of an additional immiscible phase, but because this additional pseudo-component does not interact with the EoS, we omit it from the discussion herein.

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

Each species can exist in both the liquid and vapor phases. We denote the *molar fraction* of component i in the liquid and vapor phases as x_i and y_i, respectively. The overall mole fraction of a component is denoted z_i. We can then relate the three via the liquid- and vapor-phase mole fractions L and V,

$$x_i L + y_i V = z_i, \qquad L + V = 1. \tag{8.1}$$

8.2.1 Basic Flow Equations

Our starting point is the general form of the discrete conservation equation defined in each cell of a computational grid for a system made up of N components,

$$\frac{\mathbf{M}_i^{n+1} - \mathbf{M}_i^n}{\Delta t^n} + \texttt{div}(\mathbf{V}_i) - \mathbf{Q}_i = 0, \quad i \in \{1, \ldots, N\}. \tag{8.2}$$

Here, the vectors $\mathbf{M}_i$ and $\mathbf{Q}_i$ have one entry per cell, whereas $\mathbf{V}_i$ has one entry per interface in the grid. This equation was also discussed in Chapter 5, and you may recall that `div` denotes a discrete version of the divergence operator that essentially summarizes flux contributions over all faces delimiting each cell; this operator is defined more precisely in subsection 4.4.2 of the MRST textbook [21].

To define the total cell mass $\mathbf{M}_i$ and total mass flux $\mathbf{V}_i$ for each component, we take the sum over the two phases and introduce the *mass fractions* $\boldsymbol{X}_i$ and $\boldsymbol{Y}_i$,

$$\mathbf{M}_i = \boldsymbol{\Phi}\big(\boldsymbol{\rho}_\ell \boldsymbol{S}_\ell \boldsymbol{X}_i + \boldsymbol{\rho}_v \boldsymbol{S}_v \boldsymbol{Y}_i\big), \quad \mathbf{V}_i = -\mathbf{T}^f\big(\boldsymbol{\lambda}_{i,\ell}^f \boldsymbol{\Theta}_\ell + \boldsymbol{\lambda}_{i,v}^f \boldsymbol{\Theta}_v\big). \tag{8.3}$$

Here, $\boldsymbol{\rho}_\alpha$, $\boldsymbol{S}_\alpha$, and $\boldsymbol{\Theta}_\alpha$ denote the density, saturation, and phase potential of phase α; $\boldsymbol{\lambda}_{i,\alpha}^f$ is the component mobility, evaluated at the cell interfaces; $\boldsymbol{\Phi}$ is the cell-wise pore volume; and $\mathbf{T}^f$ is the vector (or matrix) of intercell transmissibilities.

We can relate the mass fractions to the mole fractions via the molar masses m_i as $X_i = m_i x_i / \big(\sum_i^N m_i x_i\big)$. To solve the system (8.2)–(8.3), we have to know the phase saturations and the N mole fractions for both phases. In addition, the flow potential and any dependence on the pressure for densities will require knowledge of both phase pressures. Additional closure relations can be introduced by assuming that the fractions sum up to unity and that the liquid pressure is the reference pressure,

$$\sum_i^N \boldsymbol{x}_i = 1, \quad \sum_i^N \boldsymbol{y}_i = 1, \quad \boldsymbol{S}_\ell + \boldsymbol{S}_v = 1, \quad \boldsymbol{p}_v = \boldsymbol{p}_\ell + \boldsymbol{p}_{cv\ell}. \tag{8.4}$$

If we consider a single cell so that the phase state is uniquely defined, we see that after eliminating one variable from each of these relations, we need to at least solve for $2(N-1)$ mole fractions, the reference pressure, and one saturation, giving a total of $2N$ unknown variables. If we also make the assumption that we are under

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

single-phase conditions – i.e., that $S_\ell = 0$ or $S_v = 0$ – the number of unknowns reduces to N variables from the requirement that the phase mobility is zero for zero saturation. We have so far only introduced the N governing equations in the form of mass conservation, indicating that we will have to introduce additional relationships for the system to be well posed if both phases are present.

8.2.2 Thermodynamics

To determine the composition of a multicomponent hydrocarbon system at a given pressure, one must first determine whether the system exists in a single-phase or two-phase state. This is either done by using a so-called phase stability test [26] or a saturation–pressure calculation [28]. When the system is in a two-phase state, molecules will continuously vaporize from the liquid phase and condense from the vapor phase. If the phases are not in equilibrium, pressures and temperature may differ between the phases and chemical species will condense and vaporize at different rates. Given enough time, however, the two phases will reach a state of thermal equilibrium (same temperatures), mechanical equilibrium (same pressures), and chemical equilibrium (condensation rate equals vaporization rate for all chemical species).

The exact degree to which each component appears in each of the two phases at equilibrium is key to the behavior of the system and depends in the most general case on pressure, temperature, and the other components present. The problem of splitting a total composition into phase compositions (i.e., determining x_i, y_i, and L or V from known z_i values) is referred to as a *flash calculation* [27] and can be carried out using two different approaches.

K-value Methods

The simplest approach to calculate the compositions of each phase is to assume that components partition across phases according to a fixed ratio; i.e., we introduce so-called *equilibrium constants* K_i that relate the phase mole fractions to each other [9]. The K-values are usually assumed to be functions of pressure and temperature only, and the relationship $K_i x_i = y_i$ can be derived from the isofugacity condition, which we will come back to shortly when discussing equations of state. This linear form is convenient if the molecular makeup of each phase is tabulated (e.g., from an experiment), but K-values can also be obtained from various correlations. When the K-values also depend on the solution variables, the name "equilibrium constant" is somewhat misleading and it is more natural to refer to the values as just K-values.

The K-value approach is implemented in most commercial compositional simulators. In isothermal simulations, the K-values are usually set to depend on pressure only, and this approach is adequate for displacement problems in which the

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

true K-values are weak functions of composition [9, 13]. For thermal simulation, the K-values depend on pressure and temperature and may also include liquid–liquid interactions to account for the fact that hydrocarbon components can dissolve in the water phase; see, e.g., [56]. A motivation for introducing K-values is that it separates the flow equations from the phase-equilibrium equations during the nonlinear iteration process. This simplifies the phase stability test and can significantly speed up the computations. The main disadvantage is that the K-values typically do not account for dependencies on composition and do not provide a way to evaluate compressibility factors on their own.

Equation of State Methods

When a suitable EoS is known (these will be discussed more in Subsection 8.3.4), we can instead minimize the Gibbs free energy [14] of the system:

$$G(p,T) = U + pV + TS. \tag{8.5}$$

Here, G is the free energy as a function of internal energy U, pressure p, volume V, temperature T, and entropy S [51]. In the context of a mixture described by mole fractions and fugacities as a function of pressure, temperature, and mole fractions, we can simplify the expression of the Gibbs free energy to known quantities, here in the normalized form [51]:

$$g^* = G/RT = \sum_{i=1}^{N} z_i \ln f_i(z,p,T). \tag{8.6}$$

Though it is possible to minimize the Gibbs energy directly as an unconstrained optimization problem (see, e.g., [47]), most simulators instead work with the first-order conditions for chemical equilibrium. To do so, we simply replace the K-value condition with the implicit relationship $f_{i\ell} = f_{iv}$, which requires that the fugacity of the component in the liquid phase is equal to the fugacity of the component in the vapor phase. In the limit of an ideal gas, the component fugacity is equal to the partial pressure, and in general the fugacity can be considered as a chemical potential that measures the tendency of a species to escape from one phase to another. The isofugacity constraint implies equality of chemical potential between the two phases, a necessary condition for chemical equilibrium. In practice, when allowed general dependence on all solution variables, K-values and fugacities can produce exactly the same system behavior. In summary, the full set of so-called flash equations can be written as a system of $2N + 1$ equations: first, N equations for chemical equilibrium, choosing one of the two forms:

$$E_i = 0, \quad E_i = \begin{cases} K_i x_i - y_i, & \text{(for } K\text{-values)}, \\ f_{i\ell} - f_{iv}, & \text{(for isofugacity)}, \end{cases} \tag{8.7}$$

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

and then $N + 1$ additional closures for molar balance between the phases:

$$B_i = x_i L + y_i(1 - L) - z_i = 0, \tag{8.8}$$

$$C = \sum_{i=1}^{N} (x_i - y_i) = 0. \tag{8.9}$$

We can summarize the flash problem by defining a set of N fixed variables $\boldsymbol{\eta} \in \mathbb{R}^N$ together with the set of flash primary variables $\boldsymbol{\beta} \in \mathbb{R}^{2N+1}$ under isothermal conditions so that the flash itself is then the solution of the system:

$$\mathbf{G}(\boldsymbol{\eta}, \boldsymbol{\beta}) = \begin{bmatrix} E_1(\boldsymbol{\eta}, \boldsymbol{\beta}) \\ \vdots \\ E_N(\boldsymbol{\eta}, \boldsymbol{\beta}) \\ B_1(\boldsymbol{\eta}, \boldsymbol{\beta}) \\ \vdots \\ B_N(\boldsymbol{\eta}, \boldsymbol{\beta}) \\ C(\boldsymbol{\beta}) \end{bmatrix} = 0, \qquad \boldsymbol{\eta} = \begin{bmatrix} p \\ z_1 \\ \vdots \\ z_{N-1} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} L \\ x_1 \\ \vdots \\ x_N \\ y_1 \\ \vdots \\ y_N \end{bmatrix}. \tag{8.10}$$

Altogether, these equations are solved for $2N$ liquid and vapor mole fractions together with the liquid mole fraction L.

8.3 Solving the Flash Problem

As a prelude to describing the coupled problem of both flow and thermodynamic equilibrium, we will first describe the process for solving only the local equilibrium. The example `introToVaporLiquidEquilibrium` used throughout this section demonstrates many of the techniques in some detail. In much of the following, we are primarily concerned with the two-phase state in which both liquid and vapor are present.

8.3.1 Rachford–Rice: Determination of Vapor–Liquid Equilibrium

Let us for a moment assume that we know the K-values and would like to find the liquid fraction so that both the K-value relation (8.7) and the overall mole balance (8.1) are fulfilled. By combining the two relations and solving for either x_i or y_i and taking the sum over all components, we can obtain a pair of objective functions for the vapor–liquid equilibrium (VLE),

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

$$O_1 = \sum_{i=1}^{N} \frac{z_i}{1 + V(K_i - 1)} - 1 \quad \text{and} \quad O_2 = \sum_{i=1}^{N} \frac{K_i z_i}{1 + V(K_i - 1)} - 1. \tag{8.11}$$

We could attempt to use the standard Newton machinery to solve either $O_1(V) = 0$ or $O_2(V) = 0$ for V as a primary variable. Unfortunately, neither function is monotone in terms of the vapor fraction V and Newton's method does not easily converge. Taking the difference between the two objective functions was suggested in 1952 in a seminal single-page paper by Rachford and Rice [42], which results in the eponymous Rachford–Rice objective function,

$$O_2 - O_1 = O_{RR} = \sum_{i=1}^{N} \frac{(K_i - 1)z_i}{1 + V(K_i - 1)}. \tag{8.12}$$

This function is locally monotone away from the N singularities located at $V = 1/(1 - K_i)$. The solution is found in the interval $[(1 - K_{\max})^{-1}, (1 - K_{\min})^{-1}]$ (see [52]) and can be solved by Newton's method. If the Newton update brings the value outside the solution interval, an unconditional root-finding solver is used; in MRST, we use a simple bisection algorithm. A converged value for V outside of $(0, 1)$ indicates a single-phase condition.

As an example, we can think of a two-component system made up of heavy (h) and light (l) molecules so that $K_h = 1/10, K_l = 10$: Under two-phase conditions, the heavy component will mostly be present in the liquid phase, with one mole in the vapor phase for every 10 moles in the liquid. The light component reverses the situation, with one out of 11 molecules ending up in the liquid phase. In MRST, the `solveRachfordRice` function solves the objective function for given K-values. We span the range of possible mole fractions for the light component and solve the objective function:

```
n = 50;                          % Number of samples
K = [0.1, 10];                   % First component is heavy, second is light
z_light = linspace(0, 1, n)';    % Go from 0 -> 1
L = solveRachfordRiceVLE([], K, [1 - z_light, z_light]); % No initial guess
```

The vapor mole fraction is plotted in Figure 8.1. We observe a linear increase from $z_l = 0.1$ to $z_l = 0.9$. If the mole fractions of each component are equal to the corresponding K-values, we get a trivial (pure liquid or vapor) solution. The three objective functions O_1, O_2, and O_{RR} are plotted together for varying values of V for $z_l = z_h = 0.5$ in Figure 8.2. We observe that the Rachford–Rice function is indeed monotone in the physically meaningful interval $V \in [0, 1]$, whereas O_1 and O_2 are mirror opposites with nonmonotone regions. The minima of O_1 and O_2 coincide, and O_{RR} takes on the expected zero value at this point.

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

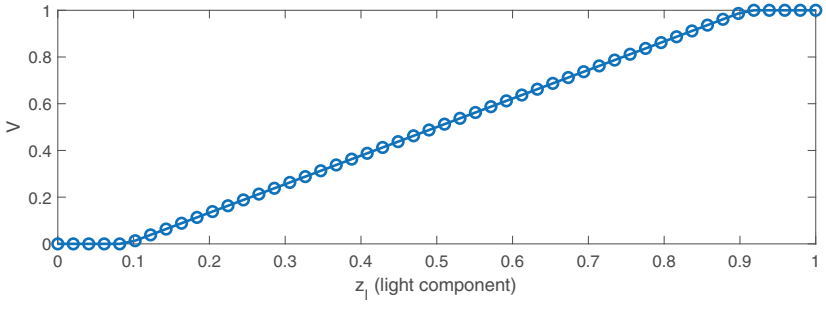


Figure 8.1 Vapor mole fraction V as function of the overall mole fraction of the light fluid in a system consisting of a light and a heavy species with K-values of 10 and 0.1, respectively.

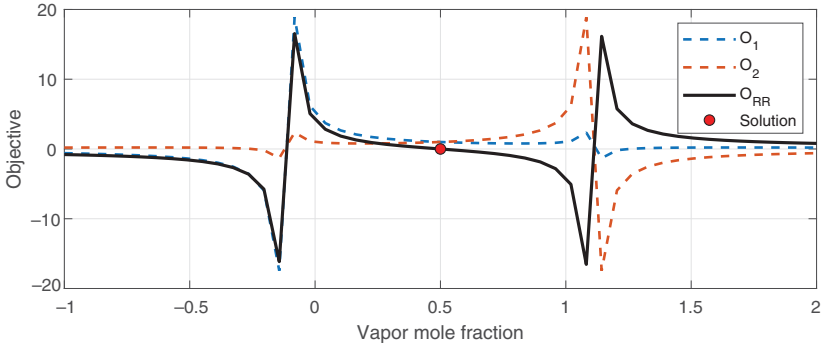


Figure 8.2 Plot of the objective functions O_1 and O_2 from (8.11) and O_{RR} from (8.12) as a function of vapor mole fraction V at $z_l = z_h = 0.5$ for the system with light and heavy species with equilibrium values of 10 and 0.1, respectively.

8.3.2 Updating the Thermodynamic Equilibrium

If we know both the K-values and suitable relationships for the density of each phase as a function of composition, Rachford–Rice is a sufficient implementation of VLE for the solution of two-phase flow. If we want to use the local equilibrium of some EoS to predict density and K-values, we require additional constraints in the form of the isofugacity condition (8.7) that defines the K-values at equilibrium.

Let us define a suitable flash problem before we detail the solution process used. We use the `TableCompositionalMixture` class to instantiate a three-component mixture with each species given by name. The class supports 122 named properties, for which the tables were generated by the CoolProp library[1] [3]

[1] Valid component names are listed by calling `TableCompositionalMixture.getFluidList`.

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

```
mixture = TableCompositionalMixture({'CarbonDioxide', 'Methane', 'n-Decane'});
disp(mixture)
```

```
TableCompositionalMixture:

3 component mixture (CoolProp - Tabulated):
         Name | p_c [Pa] | T_c [K] | V_c [m^3] |  acf  | mw [kg/mol]
--------------------------------------------------------------------
CarbonDioxide | 7.38e+06 | 304.1 K | 9.412e-05 | 0.224 | 0.0440098
      Methane | 4.60e+06 | 190.6 K | 9.863e-05 | 0.011 | 0.0160428
     n-Decane | 2.10e+06 | 617.7 K | 6.098e-04 | 0.488 | 0.1422817
--------------------------------------------------------------------
No non-zero binary interaction coefficients.
```

The custom `disp` implementation provides us an overview of how the mixture is specified: pressure (`p_c`), temperature (`T_c`), and volume of a single mole (`V_c`) at the critical point, together with acentric factors (`acf`) that account for a deviation from a spherical shape and the mass per mole or molecular weight (`mw`). This highlights one attractive aspect of compositional models, for which a limited number of clearly understandable quantities replace complex tables for PVT behavior.

We next set up an EoS class instance, specifying the Peng–Robinson (PR) EoS:

```
peng_robinson = EquationOfStateModel([], mixture, 'Peng-Robinson');
```

The EoS is derived from `PhysicalModel` and mostly uses the standard AD-OO features to define the flash problem. The first argument to the constructor is normally used to specify the computational grid the model is defined over, which we do not need to specify here, because we are not going to use the class instance for flow simulation. Note that the flash equations require all components to have compositions above zero. The EoS class therefore contains the property `minimumComposition` that defaults to 10^{-8}.

We can perform a flash either by setting up a state and calling `solveTimestep` with a nonlinear solver or by using a convenience function that does this for us:

```
[L, x, y] = standaloneFlash(25*barsa, (273.15 + 30)*Kelvin, c, peng_robinson);
```

The inputs are the pressure and temperature given as either a single value or as one value per composition, the overall mole fraction as a matrix with one column per composition, and the EoS itself. We create a matrix `c` that contains varying combinations of the three components. We limit the output to the liquid mole fraction and the phase mole fractions. The routine can also produce compressibility factors and densities if additional outputs are requested.

We plot the resulting liquid fraction in the ternary diagram in Figure 8.3. We observe that, under the specified conditions, the EoS predicts that no mixture only

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

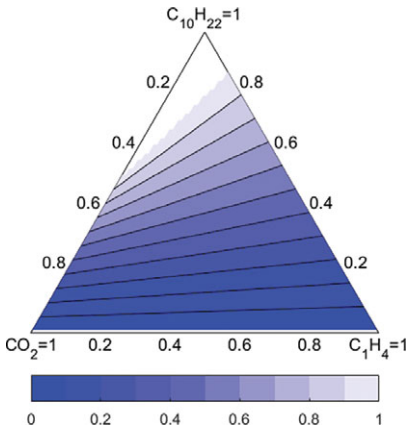


Figure 8.3 Ternary diagram of liquid fraction for a system consisting of carbon dioxide (CO_2), methane (C_1H_4), and n-decane ($C_{10}H_{22}$), predicted by the PR EoS with properties generated by CoolProp.

composed of the light components CO_2 and C_1H_4 will be able to form a liquid phase. On the other end, mixtures that are made up of mostly the heavier $C_{10}H_{22}$ component are single-phase liquids, with the lighter components fully dissolved.

We now switch to another mixture taken from the benchmark suite included with MRST. We get the mixture from the Fifth SPE Comparative Solution Project [18] by name together with information about the initial conditions from the paper and display the mixture:

```
[spe5, info] = getBenchmarkMixture('spe5');
disp(spe5)
```

```
CompositionalMixture:

6 component mixture (SPE5 benchmark):
Name | p_c [Pa] | T_c [K] | V_c [m^3] |  acf   | mw [kg/mol]
-------------------------------------------------------------
  C1 | 4.60e+06 | 190.6 K | 9.978e-05 | 0.013  | 0.0160400
  C3 | 4.25e+06 | 369.8 K | 2.004e-04 | 0.152  | 0.0441000
  C6 | 3.01e+06 | 507.4 K | 3.697e-04 | 0.301  | 0.0861800
 C10 | 2.10e+06 | 617.7 K | 6.297e-04 | 0.488  | 0.1422900
 C15 | 1.38e+06 | 705.6 K | 1.042e-03 | 0.650  | 0.2060000
 C20 | 1.12e+06 | 766.7 K | 1.341e-03 | 0.850  | 0.2820000
-------------------------------------------------------------
Binary interaction coefficients:
       0         0         0         0    0.0500    0.0500
       0         0         0         0    0.0050    0.0050
       0         0         0         0         0         0
       0         0         0         0         0         0
  0.0500    0.0050         0         0         0         0
  0.0500    0.0050         0         0         0         0
```

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

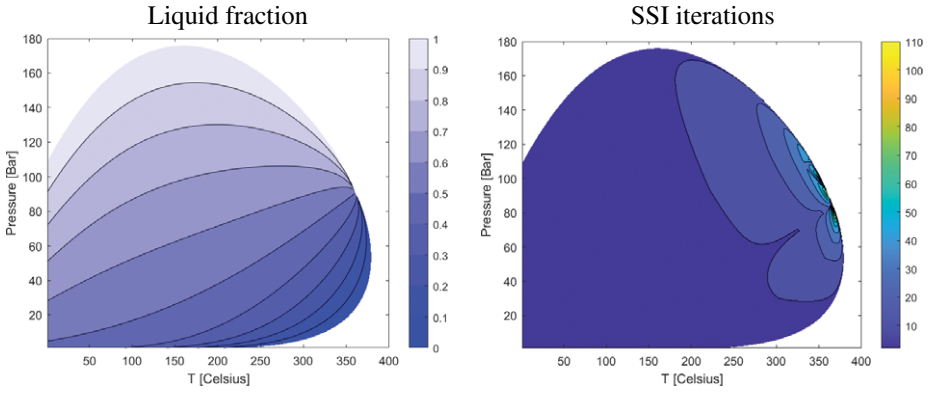


Figure 8.4 The left plot shows the p/T-phase diagram for the SPE 5 benchmark mixture. The right plot shows the number of successive substitution iterations necessary to compute each point in the phase diagram.

The six-component fluid includes PR binary interaction coefficients between the lightest and heaviest components. For more than three components, visualizing the liquid fraction as a function of compositions becomes more difficult. Another important type of plot is the p/T-phase diagram in Figure 8.4 that displays the phase behavior for fixed composition. (This is exactly the same type of diagram discussed for binary substances in section 11.4 of the MRST textbook [21].) The two-phase region is bounded below by pure liquid and above by pure vapor. The two regions meet at the critical point, beyond which the single-phase fluid is supercritical and the distinction between liquid and vapor is meaningless.

Now that we are familiar with the output of the flash, we turn our attention toward how the flash is performed. The exact nature of the `stepFunction` in the EoS changes based on the algorithm in use. We start off the flash by selecting an initial guess for the K-values. Unless we already have an estimate from, e.g., a previous solve, MRST uses Wilson's correlation [53] from the critical pressures and temperatures as the initial guess:

$$K_i \approx \frac{p_{ci}}{p} \exp\left[5.37(1+\omega_i)(1-T_{ci}/T)\right]. \tag{8.13}$$

```
K = estimateEquilibriumWilson(eos, p, T)
```

From the initial guess, we can next proceed to solve for the K-values and liquid fraction, and consequently the mass distribution in each phase.

Successive substitution iteration: The successive substitution iteration (SSI) method is an algorithm for the VLE problem that alternates between solving

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

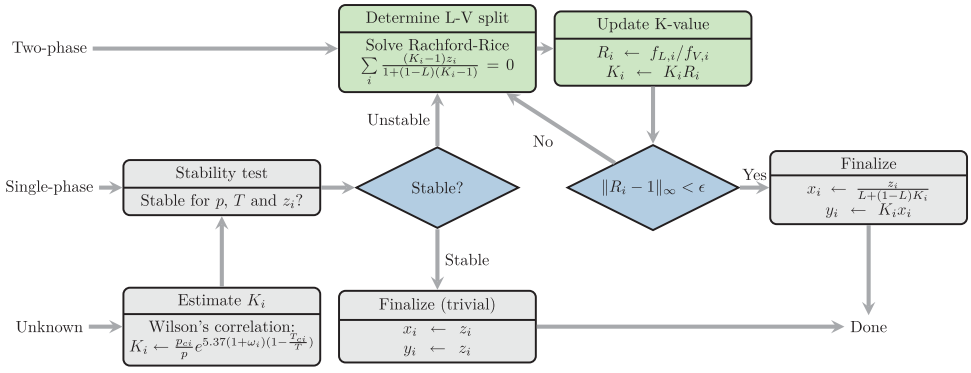


Figure 8.5 Overview of the flash procedure. The figure demonstrates the order of operations for flash with stability testing for all possible starting points for the phase state (unknown, single-phase, or two-phase). The green blocks correspond to the specific choice of successive substitution iteration to converge the isofugacity constraints. If Newton's method is used, the green blocks are replaced with a single linearization that solves for both the liquid fraction and the ratio between liquid and vapor mole fractions.

Rachford–Rice to determine the liquid fraction and updates to estimated K-values. By observing that the fugacity ratio must be one at convergence, we can define an update to the K-values at iteration k,

$$K_i^{k+1} = K_i^k \frac{f_\ell(p, T, \mathbf{x}^k)}{f_v(p, T, \mathbf{y}^k)}, \quad x_i^k = \frac{z_i}{L + (1 - L)K_i^k}, \quad y_i^k = K_i^k x_i^k. \tag{8.14}$$

The scheme is derivative free and exhibits first-order unconditional convergence. SSI adjusts the component to become more vapor-like if the liquid fugacity is larger than the vapor, and vice versa, before enforcing mass balance by solving Rachford–Rice with the estimated K-values. The advantages of SSI are the ease of implementation, low cost per iteration, and unconditional convergence. The disadvantage is that a very large number of iterations may be required near the boundary of the two-phase region and particularly near the critical point, as seen in the right plot of Figure 8.4. The flash procedure, starting from stability testing, is outlined in Figure 8.5 with SSI as the method.

Newton's method: We could also treat the system (8.7) together with (8.8) and (8.9) as a standard nonlinear system $\mathbf{G}(\boldsymbol{\eta}, \boldsymbol{\beta}) = 0$ and solve it with Newton's method for the primary variables $\boldsymbol{\beta}$. The quadratic convergence of Newton's method is attractive, but this method is unfortunately only conditionally convergent for this system of equations. Assembling the Jacobian comes at some cost, especially for systems with many components. If Newton's method is used, the light green

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

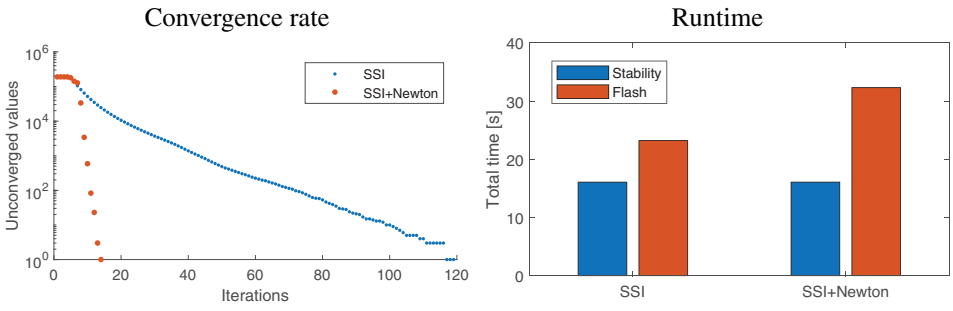


Figure 8.6 Comparison of the cost of using SSI and an adapted algorithm for the flash calculation that switches from SSI to Newton after five iterations.

boxes in Figure 8.5 are replaced by a single simultaneous update to all quantities. In MRST, setting the `method` property of the EoS model instance to `'newton'` will use Newton for all cells. The linearization of the flash equations will be performed by the AD backend of the model, as described in Chapter 6. The conditional convergence of Newton's method means that this option is primarily for testing purposes or if you know that your conditions ensure that Newton is safe to use.

Alternative schemes: Both SSI and Newton are essentially deficient schemes for VLE in their own way: Whereas SSI is reliable, it is slow to converge and, likewise, whereas Newton can converge fast, it often requires stabilization techniques to do so reliably. Altogether, this indicates that flash equations constitute a challenging problem to solve.

The body of work on accelerating two-phase flash and extending it to multiple phases contains many alternative techniques (for example, [32, 34, 35, 37, 41]). Most of these techniques start from either SSI and/or Newton's method and extend these to accelerate convergence near the critical point. MRST includes the option of dynamically switching to Newton's method when an iteration threshold specified by the `maxSSI` property is exceeded. (At the time of writing, this is the only accelerated method implemented in MRST.) By setting `maxSSI` to 5, we see significantly improved convergence in Figure 8.6. However, each Newton iteration is significantly more expensive and, accordingly, the total runtime with our fastest AD backend is still higher than regular SSI. With the adaptive implementation of the flash solver, the benefit of vectorization drops off as fewer and fewer points remain unconverged, and the overhead inherent in any AD backend (see Figure 6.6 from Chapter 6, for instance) implies that the cost of using automatic differentiation becomes large relative to the cost of SSI for few points. The stability test does not use linearization and thus the cost is the same for both flashes.

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

8.3.3 Phase Stability Testing

The equations used for Newton or SSI are written assuming that there are two phases present. Normally, an equally important part of the VLE is how to efficiently decide whether a single-phase fluid mixture will split into two phases once pressure, temperature, and compositions have changed. This is the so-called phase stability test from Michelsen [26]. The essence of the algorithm is to perform two partial flashes for the overall composition $\mathbf{z}$ by letting $\mathbf{x} \leftarrow \mathbf{z}$ (and $\mathbf{y} \leftarrow \mathbf{z}$) and then solving to see whether a second vapor (liquid) phase can form with positive saturation. This flash is only a partial flash, because the iterations are aborted once a conclusion about phase state is made: Either when the solution approaches a trivial solution $\left(\sum_i^N (\ln K_i)^2 < \epsilon_t\right)$ or the fugacity ratio approaches unity within the tolerance chosen for stability. If the phase is unstable, the estimates for phase mole fractions may be used as initial guesses for the full flash. The calling signature of the stability test is reminiscent of that for the standalone flash:

```
[stable, x, y] = phaseStabilityTest(eos, z, p, T, K)
```

Here, `K` is either the K-values or an empty array if these values are to be estimated. Additional options for tolerances and so on can be set via keyword arguments. The first output argument, `stable`, contains a Boolean that signifies the stability of each entry, and the other outputs are estimates for `x` and `y`. An alternative is to let the flash converge to negative saturations, the so-called negative flash strategy [16, 52], and use, e.g., compositional space adaptive tabulation as an efficient procedure to compute a good initial guess; see [17] for details. This approach is not yet supported in MRST.

8.3.4 Equation of State

Equations of state are thermodynamic equations relating state variables that describe the state of fluids, mixtures of fluids, solids, etc., under given physical conditions. In compositional simulation, these equations are used to provide constitutive relationships between state variables such as mass, pressures, temperature, and volumes at thermodynamic equilibrium. In the oil and gas industry, it is common to use so-called cubic EoS; i.e., equations that you can write as cubic functions of the molar volume $V_m = V/n = M/\rho$, involving constants that depend on pressure p_c, temperature T_c, and the molar volume V_c at the critical point, at which the following condition holds (subscript T means that temperature is held fixed):

$$\left(\frac{\partial p}{\partial V}\right)_T = \left(\frac{\partial^2 p}{\partial V^2}\right)_T \equiv 0. \tag{8.15}$$

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

Cubic EoS can all be traced back to the groundbreaking Van der Waals EoS, first proposed in 1873, and the most widespread examples within reservoir simulation are the PR [40] and the Soave–Redlich–Kwong (SRK) [39, 46] equations and various modifications thereof. The most basic examples are implemented in MRST, and in the following subsections we discuss the implementation in some more detail.

Despite their widespread use, the way cubic EoS are used in reservoir engineering has some important shortcomings. These include low accuracy when modeling complex fluid mixtures with strong molecular interactions and (large) differences in molecular sizes, imprecise calculation of liquid densities, and a general dependence on fitting to the critical point, which may not be well defined for all fluid systems. In recent years, equations based on *statistical associating fluid theory* (SAFT) [6, 7] have attracted much interest. To predict the effect of molecular size/shape and hydrogen bonding on fluid properties and phase behavior, the SAFT EoS represents molecules as spherical particles that can chain up and associate. The perturbed-chain (PC) extension of SAFT proposed by Gross and Sadowski [15], in which the spherical particles of the original SAFT EoS are replaced by a hard-chain fluid, has become particularly popular. In recent work, Masoudi et al. [24, 25] used MRST to extend PC-SAFT to also include ionic interactions (electrolyte PC-SAFT or simply ePC-SAFT), and work is underway to include this EoS in the public release.

Cubic EoS

The flash requires the fugacities of each component in each phase. The fugacities themselves typically need the compressibility factors also used to predict density. We define the compressibility factor of a phase as the ratio between the volume of the actual gas and an ideal gas under the same conditions. In an isothermal multiphase system with a unique pressure we have

$$V_\alpha = Z_\alpha \frac{n_\alpha RT}{p}, \tag{8.16}$$

where V is the volume, n_α is the number of moles of phase α, T is temperature, and R is the universal gas constant. We have so far not specified how fugacities f or compressibility factors Z are computed; the proceeding is general for any order of EoS.

We will now specify the general cubic EoS implemented by default in MRST as `EquationOfStateModel`. The name *cubic* refers to the highest order term of the scalar polynomial equation for the phase compressibility factor Z_α,

$$Z_\alpha^3 + aZ_\alpha^2 + bZ_\alpha + c = 0. \tag{8.17}$$

A specific cubic EoS is uniquely determined by the definition of the terms a, b, c as functions of pressure, temperature, and composition for a given mixture.

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

Generalized cubic EoS: The default EoS in MRST is implemented by way of the generalized cubic form from Martin [23] that covers several well-known cubic EoS. The notation here is the one used by Coats [10].

$$\begin{aligned} a =& (m_1 + m_2 - 1)B - 1, \\ b =& A - (m_2 + m_2 - m_1 m_2)B^2 - (m_1 + m_2)B, \\ c =& -(AB + m_1 m_2 B^2(B + 1)). \end{aligned}$$

Here, m_1 and m_2 change when selecting a specific EoS. Constants A and B are both dimensionless quantities that account for attractive and repulsive molecular forces, respectively; A is computed with a quadratic mixing rule and B with a linear rule, here given for the liquid phase:

$$A = \sum_{i,j}^{N} x_i x_j A_{ij}, \qquad A_{ij} = (A_i A_j)^{1/2}(1 - \delta_{ij}), \qquad A_i = \omega_{a,i} \frac{\bar{p}_i}{\bar{T}_i^2}, \tag{8.18}$$

$$B = \sum_{i}^{N} x_i B_i, \qquad B_i = \omega_{b,i} \frac{\bar{p}_i}{\bar{T}_i} \qquad \bar{p}_i = \frac{p}{p_i^c}, \bar{T}_i = \frac{T}{T_i^c}. \tag{8.19}$$

To derive A and B for the vapor phase, you substitute x with y. The superscript c refers to the static critical property (pressure or temperature) for a given species. The reduced pressures $\bar{p}_i$ and temperatures $\bar{T}_i$ for each component are dimensionless, and δ_{ij} are the binary interaction coefficients between each pair of components i and j, limited to the symmetric case $\delta_{ij} = \delta_{ji}$ with zero diagonals, $\delta_{ii} = 0$. Binary interaction coefficients can be tuned for a specific mixture to improve accuracy and the coefficients are specific to the EoS in question. The coefficient matrix, unlike the basic molecular properties, does not have a meaningful physical interpretation and the values vary depending on the EoS used. If you are using values found in the literature, please make sure that they were calibrated for the same EoS.

This general form of the cubic EoS makes the differences between EoS fairly compact. For instance, if we want to work with the PR EoS [40], the following definitions completely determine the cubic equation:

$$\begin{gathered} m_1 = 1 + \sqrt{2}, \quad m_2 = 1 - \sqrt{2}, \quad \omega_a = 0.4572355, \quad \omega_b = 0.0779691, \\ \omega_{a,i} = \omega_a \left[1 + (0.37464 + 1.54226\omega_i - 0.26992\omega_i^2)(1 - \bar{T}_i^{1/2})\right]^2, \\ \omega_{b,i} = \omega_b. \end{gathered}$$

Solving and differentiating Z_α: Now that we have defined A, B and m_1, m_2, we can solve (8.17) for the compressibility factor Z_α, which should be positive and

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

real. MRST contains `cubicPositive`, a vectorized solver that finds the nonnegative cubic roots. There can either be a single valid root or two. If two roots are present, we pick the root that minimizes the Gibbs free energy (see (8.5)): Usually, this amounts to selecting the lowest valued real candidate for the liquid phase and the largest real value for the vapor phase. The explicit check for Gibbs energy minimization can be disabled by setting `eos.selectGibbsMinimum` to `false`, skipping the calculation of (8.6) for each phase in regions with multiple real roots.

Once we have the compressibility factors, we can compute fugacities and densities. From the point of view of an SSI flash, we are now done. If we want to couple the flash to a robust flow solver or use Newton's method, we are going to need derivatives of Z_α as well. The polynomial solver requires a number of non-AD-compatible operations and does not provide derivatives. Obtaining the derivatives of Z_α with respect to arbitrary primary variables is instead done by differentiating (8.17) with respect to some unknown x and rearranging the terms to find

$$\frac{\partial Z_\alpha}{\partial x} = -\frac{\frac{\partial a}{\partial x}Z_\alpha^2 + \frac{\partial b}{\partial x}Z_\alpha + \frac{\partial c}{\partial x}}{3Z_\alpha^2 + 2Z_\alpha a + b}. \tag{8.20}$$

MRST implements this through the EoS member function `setZDerivatives` that inputs A and B as AD variables and Z as a double and produces Z as an AD variable with the correct derivatives. Alternatively, we could have used the same approach as in (8.26) to obtain the derivatives for the more general case of noncubic EoS.

Fugacity: We can write the expression for fugacity,

$$\ln\left(\frac{f_{i\alpha}}{px_i}\right) = \ln\psi_i \rightarrow f_{i\alpha} = px_i\exp\big(\ln\psi_i\big), \tag{8.21}$$

where the generalized cubic EoS yields

$$\begin{aligned}\ln\psi_i = &-\ln(Z_\alpha - B) + \frac{B_i}{B}(Z_\alpha - 1)\\ &+\ln\left(\frac{Z_\alpha + m_2 B}{Z_\alpha + m_1 B}\right)\frac{A}{(m_1 - m_2)B}\left(\frac{2}{A}\sum_{j=1}^{N}A_{ij}x_j - \frac{B_i}{B}\right).\end{aligned}$$

Cubic EoS implemented in MRST: At the time of writing, the `compositional` module implements four different cubic EoS. You can choose which one to use through the last parameter of the constructor:

```
eos = EquationOfStateModel([], mixture, eosname);
```

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

where `eosname` is either:

- `'pr'` or `'peng-robinson'` for the PR EoS [40],
- `'prcorr'` or `'peng-robinson-corrected'` for a corrected PR EoS for large acentric factors, identical to the `PRCORR` keyword for ECLIPSE E300 [44],
- `'rk'` or `'redlich-kwong'` for Redlich–Kwong [43], and
- `'srk'` or `'soave-redlich-kwong'` for the SRK EoS [46].

Phase Properties

Once we have computed the Z-factors by solving the cubic EoS, we need to compute various phase properties. The following subsections explain how this is done.

Density: The mass density is predicted from the pressure, temperature, compressibility factor, and mole fractions by introducing the phase molar volume $\tilde{V}$ to (8.16) and weighting the reciprocal volume by the amount of mass per mole of the phase. With the liquid phase as an example, we have

$$\rho_\ell = \frac{1}{\tilde{V}_\ell}\sum_i^N x_i m_i, \quad \tilde{V}_\ell = \frac{V_\ell}{n_\ell} = \frac{RTZ_\ell}{p} - \Delta\tilde{V}_\ell, \quad \Delta\tilde{V}_\ell = \sum_i^N x_i C_i. \tag{8.22}$$

Here, $\Delta\tilde{V}_\ell$ is an optional volume correction term [39] that adjusts the predicted volume based on the composition. The advantage of the volume shift as an additional parameter per component is that it introduces additional parameters that only impact the value of the density. Cubic equations of state can first have their interaction coefficients adjusted to match the VLE behavior of the system, followed by tuning of the volume shift to match observed densities of the mixtures. To use volume shift, you should specify the `volumeShift` parameter for the EoS class instance's property model with one entry per component:

```
eos.PropertyModel.volumeShift = [C1, C2, C3]; % Three-component-values for rho
```

In addition, simulation cases set up from ECLIPSE-style input with the `SSHIFT` keyword will automatically configure the property. To see how this shift works, you can check `computeMolarDensity` in the `CompositionalPropertyModel` class.

Saturations: The phase saturations in the two-phase region can be found from the predicted volumes

$$S_\ell = \frac{V_\ell}{V_\ell + V_v} = \frac{Z_\ell n_\ell RT/p}{Z_\ell n_\ell RT/p + Z_v n_v RT/p} = \frac{Z_\ell n_\ell}{Z_\ell n_\ell + Z_v n_v}. \tag{8.23}$$

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

If we divide by the total number of liquid and vapor moles, we obtain an expression that only depends on the mole fraction and not on the mole numbers themselves,

$$S_\ell = \frac{Z_\ell L}{Z_\ell L + Z_v(1-L)}. \tag{8.24}$$

In a three-phase context, the saturations should be corrected by the volume occupied by the phase(s) not predicted by the EoS, which are usually independent variables. For instance, for a three-phase system in which the water phase is independent of the EoS we write the saturations as

$$S_o = (1-S_w)\frac{Z_\ell L}{Z_\ell L + Z_v(1-L)}, \quad S_g = 1 - S_o - S_w. \tag{8.25}$$

Note that the predicted saturations in MRST are not impacted by the volume shift, if present. Saturation is computed by the EoS model as part of the initialization of the AD state at the beginning of every linearization.[2]

Viscosity: Viscosity is typically not predicted by the EoS, and some correlation or table is needed together with the relative permeability to compute the mobility during flow simulations. For hydrocarbon mixtures, the Lohrenz–Bray–Clark correlation [22] is most commonly used. Phases not governed by the EoS use the function handle stored in the fluid struct, (e.g., `muW(p)`) to calculate viscosity. As with most functional relationships in AD-OO, the viscosity is calculated through a state function that stores computed values and associated Jacobians in the `state` structure. The state function can be replaced by another user-defined function if required. Substitution of state functions is described in detail in Chapters 5 and 7.

8.4 Coupled Flow and Thermodynamics

We have so far only considered the isolated problem of thermodynamic equilibrium. Though this is interesting in its own right, our primary motivation in this book is reservoir simulation where the aforementioned relationships provide phase behavior and property prediction for mixtures flowing in a porous medium. If for brevity we assume that flow is driven purely by boundary conditions and source terms, the system of equations is made up of two parts: (i) the N conservation equations (8.2) for each component in the system and (ii) the N isofugacity or K-value constraints (8.7) for cells in which two phases are present, in addition to the closures in (8.9). The mass-balance equations are time dependent and the

[2] The next section discusses the choice of primary variables, and you will see that saturation is a primary variable in the so-called natural variable formulation and a dependent variable (that has an associated Jacobian) in the so-called overall composition formulation.

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

residual values in each cell depend on the fluxes in its neighbors. The flash equations are entirely local to each cell but can require a very large number of iterations if the conditions in the cell are close to the critical point. The flash equations are also trivial to solve under single-phase conditions. Different strategies for solving the coupled system of equations can significantly impact the nonlinear solution process; the governing equations are the same regardless of any strategy chosen, and the converged solution should be the same, but the number of iterations until convergence can be significantly impacted.

The literature contains a number of different choices for primary variables and governing equations for compositional flow. Many of these are fundamentally similar and only have apparent differences due to the large number of possible choices that at a first glance may seem arbitrary; for example, whether one is solving for liquid fraction or saturations or whether one uses mole fractions instead of mass fractions as variables for the same set of equations. There are, however, substantial differences in the overall strategy between what is often termed *natural variables* and *molar* or *overall composition variables*. Representative versions of both of these choices are implemented in MRST. Because MRST is a research tool, each formulation is implemented in two different ways (see Table 8.1 on page 354): as a relatively large monolithic class that follows the AD-OO layout described in the MRST textbook [21] and as a much more modular implementation that utilizes the new framework of generic model classes described in Chapter 5. Many of the more advanced features supported by the compositional module require the use of the latter implementation. This includes separators for surface conditions, selecting which phase is vapor or liquid, and using different discretizations as discussed in Chapter 5.

The two compositional formulations in MRST are essentially identical for cells in single-phase state. The distinction between the phase mole fractions and overall mole fraction disappears, the saturations are trivial, and we linearize and solve the N conservation equations with $\boldsymbol{\eta}$ as the primary variables; i.e., pressure and $N - 1$ overall mole fractions. Once pressure and compositions have been updated, the phase-stability test from Subsection 8.3.3 is performed to determine whether the new values result in multiple phases forming. At this point, the treatment begins to differ between the solvers.

8.4.1 Overall Composition Formulation

The overall composition formulation uses pressure p and $N - 1$ overall mole fractions z_i as primary variables and is an example of the class of molar formulations. Many different formulations have been proposed in the literature; e.g., by

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

Fussell and Fussell [13], Young and Stephenson [54], Acs et al. [1], Chien et al. [8], and Collins et al. [11]. In the thermal case, the set of variables also includes total enthalpy.

Primary variables: The overall composition formulation starts from the assumption that the flash equations are, in some sense, more difficult to converge than the flow equations. In this formulation, the flash equations (8.7)–(8.9) are solved as a *nested* nonlinear system. This means that for every linearization of the flow equations, the nonlinear flash system $\mathbf{G}(\boldsymbol{\eta}, \boldsymbol{\beta})$ is solved to within the strict tolerances expected when the system is fully converged. We can then select $\boldsymbol{\eta}$ as our N primary variables for the remaining N flow equations after eliminating the closures in (8.4). The flash in single-phase cells is limited to the stability test if the computed states are determined to be stable. If the stability test predicts that a cell passes from a single-phase state to a two-phase state, the computed values are used as initial guesses to the flash, as described in Figure 8.5.

Flash derivatives: A small problem remains, because the flash does not directly provide the derivatives of the flash outputs $\boldsymbol{\beta}$ (made up of phase mole fractions and liquid mole fractions) with respect to the primary variables $\boldsymbol{\eta}$ of pressure and overall mole fractions. These values can be found via implicit differentiation, assuming that the function is sufficiently smooth so that the implicit function theorem holds:

$$\mathbf{G}\big(\boldsymbol{\eta}, \boldsymbol{\beta}(\boldsymbol{\eta})\big) = 0 \rightarrow \frac{d\mathbf{G}}{d\boldsymbol{\eta}} = \frac{\partial \mathbf{G}}{\partial \boldsymbol{\eta}} + \frac{\partial \mathbf{G}}{\partial \boldsymbol{\beta}} \frac{\partial \boldsymbol{\beta}}{\partial \boldsymbol{\eta}} = 0 \rightarrow \frac{\partial \boldsymbol{\beta}}{\partial \boldsymbol{\eta}} = -\left(\frac{\partial \mathbf{G}}{\partial \boldsymbol{\beta}}\right)^{-1} \frac{\partial \mathbf{G}}{\partial \boldsymbol{\eta}}. \quad (8.26)$$

This system is local to each cell and is only required in two-phase cells. In MRST, the EoS member function `getPhaseFractionDerivativesPTZ` calculates the derivatives of phase fractions with respect to pressure, temperature, and overall compositions in all two-phase cells. If you are working with AD variables – for instance, inside a flow solver – `getPhaseFractionAsADI` calls this function and applies additional chain-rule calculations if the primary variables are different than the pressure and mole fractions. Now that we know how to compute all of the relevant quantities, we can summarize the overall molar composition formulation:

1. Solve the flash problem $\mathbf{G}(\boldsymbol{\eta}, \boldsymbol{\beta}) = 0$ for $\boldsymbol{\beta}$ with $\boldsymbol{\eta}$ kept fixed. In practice, we solve the flash problem to within some tolerance, which must be tight to ensure accurate derivatives in the next step.
2. Use (8.26) to obtain derivatives of the flash outputs with respect to the flow primary variables, $\partial \boldsymbol{\beta} / \partial \boldsymbol{\eta}$.

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

3. Compute phase saturations (8.25) and phase mass densities (8.22). Saturations are determined by the liquid fraction L and compressibility factors Z_ℓ and Z_v.
4. Assemble the discrete residual equations (8.2) for flow in terms of pressure and overall molar compositions ($\boldsymbol{\eta}$). If the flow equations are converged, we have achieved simultaneous convergence of *both* the mass-balance equations and the flash equations and we continue to the next timestep. Otherwise, we perform a single Newton iteration using the linearized residual and Jacobian, before returning to step 1.

Configuration options: In addition to limits on pressure changes, the MRST classes for overall composition can limit the maximum allowable change in overall mole fraction with the `dzMaxAbs` property. The compositional base class contains the EoS class instance in the property `EOSModel`. All of the options described in Section 8.3 for the flash can be adjusted to alter the performance of the overall composition model, for instance by switching between SSI and Newton.

Convergence criteria: The overall composition model uses standard criteria from the base compositional model only. Convergence of the flash is determined by the EoS model, and we thus only need to check the pressure increment (ϵ_p relative to `incTolPressure`) and the mass-balance error (ϵ_m relative to `nonlinearTolerance`):

$$\frac{\|\Delta \boldsymbol{p}\|_\infty}{\max(\boldsymbol{p}) - \min(\boldsymbol{p})} < \epsilon_p, \qquad \Delta t \left\| \frac{\mathbf{R}_i}{\sum_j^N \mathbf{M}_j} \right\|_\infty < \epsilon_m \quad \forall i \in 1, \ldots, N. \tag{8.27}$$

In simulations with three phases, we exclude all cells in which $S_\ell + S_v < 10^{-4}$ from this check and assume these to be converged irrespective of residual value, to avoid division by zero.

8.4.2 Natural Variables Formulation

In the natural variable formulation, first proposed by Coats [9], the unknowns consist of pressures, saturations S_i, and phase compositions x_i and y_i. In the thermal case, the set of variables also includes temperature. Using natural variables usually provides better nonlinear convergence for immiscible displacement cases [48].

Primary variables: In the natural variables formulation, we converge the flow equations and the thermodynamic equilibrium simultaneously using the

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

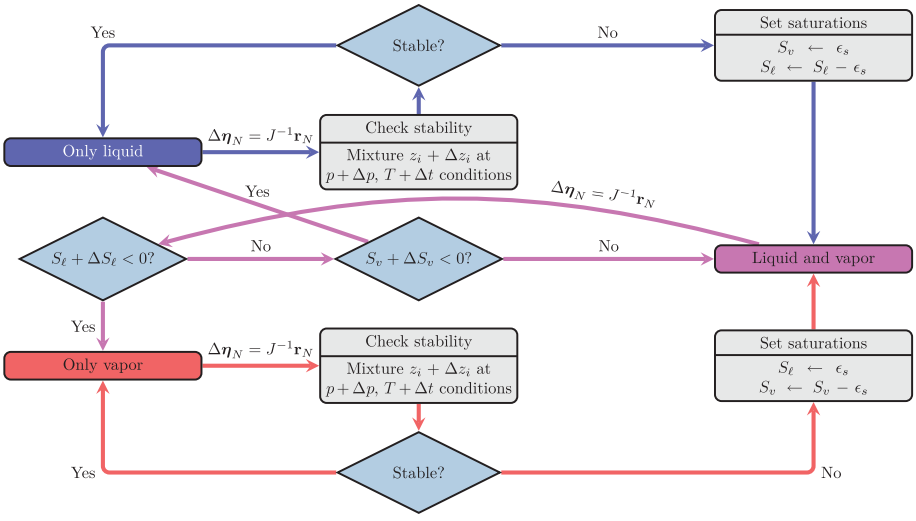


Figure 8.7 The phase transition procedure for the natural variables formulation. Starting from either of the possible phase states (single-phase liquid in blue, single-phase vapor in red, and two-phase in pink), the simulator can transition to other phase states by monitoring saturation changes and mixture stability.

procedure described in Figure 8.7 for the transition between single-phase and two-phase states. Once a single-phase cell is determined to be unstable in the stability test, the previously absent EoS phase is set to a small value ϵ_s. Likewise, transition to a single-phase state occurs when the saturation of the other phase becomes negative after the Newton update. The formulation relies on solving simultaneously for all phase mole fractions and the phase saturation by the natural primary variable set

$$\boldsymbol{\eta}_N = (p, x_1, \ldots, x_{N-1}, S_\ell, y_1, \ldots, y_{N-1}). \tag{8.28}$$

These $2N$ primary variables are paired with the N mass-balance equations (8.2) and the N isofugacity or K-value constraints (8.7) before the system is linearized.

Schur complement: The resulting linearized system is twice as large as the overall composition system, but we can exploit the structure by noting that the last N isofugacity equations are local to each cell and use a Schur-complement procedure to express the flow equations (mass conservation) as functions of the primary variables only. To explain the procedure, we partition the variables (8.28) into disjoint primary and secondary sets, $\mathbf{x}_p$ and $\mathbf{x}_s$. We denote the Jacobian of the

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

mass balance and isofugacity residual equations as J_{m*} and J_{e*}, respectively, where $* \in \{p, s\}$ denotes that the derivatives are taken either with respect to $\mathbf{x}_p$ or $\mathbf{x}_s$. With this notation, we can define a Schur complement (i.e., perform a block-Gaussian elimination) that reduces the system to an $N \times N$ linear system:

$$-\begin{bmatrix} J_{mp} & J_{ms} \\ J_{ep} & J_{es} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_p \\ \Delta\mathbf{x}_s \end{bmatrix} = \begin{bmatrix} \mathbf{r}_p \\ \mathbf{r}_s \end{bmatrix} \rightarrow (J_{mp} - J_{ms}J_{es}^{-1}J_{ep})\Delta\mathbf{x}_p = \mathbf{r}_p - J_{ms}J_{es}^{-1}\mathbf{r}_s. \tag{8.29}$$

This system can be solved with standard constrained pressure residual-type preconditioned iterative solvers described in Chapter 6. After solving for $\mathbf{x}_p$, the additional values are recovered:

$$\mathbf{x}_s = J_{es}^{-1}(\mathbf{r}_s - J_{ep}\Delta\mathbf{x}_p). \tag{8.30}$$

The action of the inverse of J_{es} is required both for the reduction and for the recovery. For this reason, MRST performs a single lower–upper factorization and stores the factors. The Schur complement requires that J_{es} is invertible. This is not the case if we use the variable ordering in (8.28) as is. We therefore reorder the variables slightly before performing the Schur complement by swapping the saturation with an arbitrarily chosen liquid mole fraction:

$$\mathbf{x}_p = (p, x_1, \ldots, x_{N-2}, S_\ell), \quad \mathbf{x}_s = (x_{N-1}, y_1, \ldots, y_{N-1}). \tag{8.31}$$

We also refer you to Cao [5] for an early comprehensive description of natural variables in practice.

The Schur-complement reduction is automatically performed by MRST so that when the linear solver calls `getLinearSystem` on the linearized problem (see [21, subsection 12.3.1]), it outputs the reduced system. The eliminated variables are also automatically recovered when `storeIncrements` is called inside the linear solver (see [21, subsection 12.3.4]). You can disable this behavior by setting the property `reduceLinearSystem` of your natural variable model class to `false`; for example, if you wish to examine the full system.

Configuration options: The natural variables class in MRST contains several options that alter the nonlinear behavior of the solver. These include changing the definition of ϵ_s through `saturationEpsilon`, performing additional stability tests when cells switch to single phase (`checkStableTransition`), as well as deciding whether cells switching to the two-phase conditions should use a full flash to initialize the saturations and phase molar fractions (`flashFromSinglePhase`). In addition to the limits on changes for pressure and overall mole fractions present

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

in the overall composition model, the natural variables model limits changes in phase mole fractions according to `dzMaxAbs` and the saturation. Assuming that you have a natural variable class object called `natural`, the following code overrides the default settings:

```
natural.reduceLinearSystem    = false;  % Return full LinearizedSystemAD
natural.checkStableTransition = true;   % Perform extra stability tests
natural.saturationEpsilon     = 1e-8;   % Max dS during phase change
natural.flashFromSinglePhase = true;    % Flash for single-phase cells
```

We discuss the implementation of model classes in more detail in Subsection 8.4.4.

Convergence critera: In addition to the tolerances for pressure increment and component-mass balance described in Subsection 8.4.1, the natural variables formulation needs to check the fugacity values in two-phase cells:

$$\|(\boldsymbol{S}_\ell + \boldsymbol{S}_v)(\mathbf{f}_{i\ell} - \mathbf{f}_{iv})\|_\infty < \epsilon_f \quad \forall i \in \{1, \ldots, N\}. \tag{8.32}$$

The convergence check uses a tolerance (ϵ_f, prescribed in `fugacityTolerance`) that is scaled by `barsa` when tested, because the isofugacity is given in terms of the default pressure unit of Pascal, which becomes large for typical subsurface simulations.

8.4.3 Comparison between Different Formulations

Let us summarize the two previous subsections: The natural variable formulation uses a *variable substitution approach*, in which we assemble the full nonlinear equations (mass conservation (8.2) and isofugacity equations (8.7) with constraints (8.9) used to eliminate one mole fraction per phase) in each cell that contains two phases. We then use a Schur-complement technique to reduce the corresponding *linearized* system for the $2N$ primary variables so that we can solve for only N variables in each cell. If one of the phases disappears, one must perform a *variable switching* to eliminate the corresponding saturation and reduce the overall system. With the overall composition formulations, the flow and thermodynamic equations are solved as a *nested nonlinear system* so that the nonlinear flash problem is solved fully nonlinearly for each outer iteration on the flow equations.

In the following, we first use a single-cell problem to compare and contrast the behavior of the two formulations before making some more general comments. We also encourage you to consult Voskov and Tchelepi [48] for a more comprehensive (computational) comparison of the two formulations implemented in MRST, as well as a set of other molar formulations.

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

A two-component example: The `showNaturalOverall` script contains a small test case that demonstrates the difference between the two formulations in practice, as well as how to set up the solvers. We create a single-cell problem:

```
G = computeGeometry(cartGrid(1));     % Single cell 1 m^3 grid
rock = makeRock(G, 0.5*darcy, 0.5);
```

We define a two-phase fluid and disable the `'blackoil'` optional parameter to avoid adding viscosities and shrinkage factors to the fluid struct. To aid in the visualization of the results, we keep the model simple as a two-component water–CO_2 mixture[3]:

```
f = initSimpleADIFluid('phases', 'wg', 'blackoil', false, 'rho', [1000, 700]);
mixture = TableCompositionalMixture({'Water','CarbonDioxide'}, {'Water','CO2'})
```

We next define the inputs to the constructor classes. The mandatory input arguments are the same irrespective of the choice of formulation. In the optional inputs, we set the water and gas phases as active and set the model to use those as the liquid and vapor phases, respectively, to obtain a two-component, two-phase model:

```
arg = {G, rock, f, ...                                % Standard arguments
       mixture,...                                    % Compositional mixture
       'water', true, 'oil', false, 'gas', true,... % Water-Gas system
       'liquidPhase', 'W', 'vaporPhase', 'G'};      % Water=liquid, gas=vapor

% Construct models for both formulations. Same input arguments
overall = GenericOverallCompositionModel(arg{:});   % Overall mole fractions
natural = GenericNaturalVariablesModel(arg{:});     % Natural variables
```

The code for setting up the initial state and boundary conditions is similar to what you may have encountered for other multiphase problems. The only difference is that, in addition to pressure and saturations, we need to specify the composition and temperature:

```
p = 50*barsa; T = 273.15 + 30; s = []; z = [1, 0];     % p, T, s, z
bc = fluxside([], G, 'xmin', 1/day, 'sat', [0, 1]);    % Flux
bc = pside(bc, G, 'xmax', p, 'sat', [0, 1]);           % Standard bc
bc.components = repmat([0, 1], numel(bc.face), 1);     % Boundary z
state0 = initCompositionalState(overall, p, T, s, z);  % Initialize state
```

[3] Using the default PR configuration for water–CO_2 VLE without volume shift or other parameter adjustments is somewhat questionable. For the purpose of this example, however, we simply think of the two as heavy and light components, respectively.

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

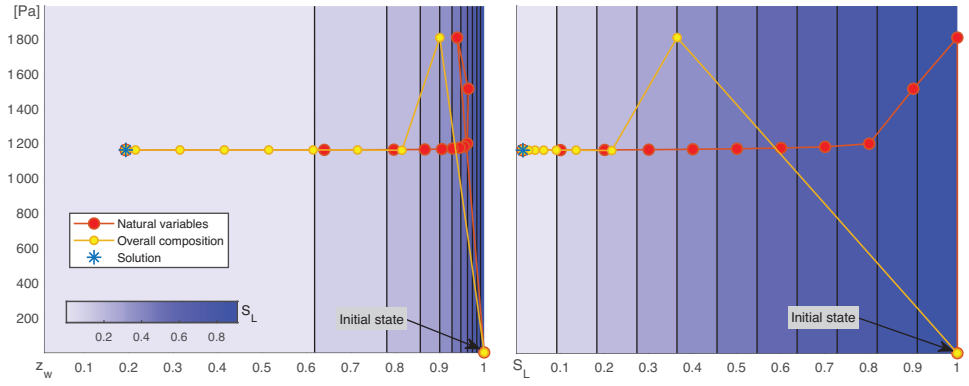


Figure 8.8 Solution paths for the natural variable and overall compositions formulations for the single-cell problem plotted in $(z_w, \Delta p)$ space (left) and in $(S_\ell, \Delta p)$ space (right). The color sectors in both plots represent isocontour lines for $S_\ell = 0.1, 0.2, \ldots, 1$.

The compositional solvers support the way of specifying standard boundary conditions, source terms, and wells in MRST, provided that the `components` field is added to the corresponding structures to specify inflow compositions.

Once the scenario has been set up, we solve a single timestep that brings the cell from pure water to a condition of mostly gas. The single-cell problem is entirely determined by the water mole fraction and the pressure difference from the boundary or, equivalently, by liquid saturation and pressure difference. Figure 8.8 reports the paths the two solution procedures take in $(z_w, \Delta p)$ and $(S_\ell, \Delta p)$ space. We observe that both solvers initially overshoot the pressure as they transition from the initial single-phase liquid state. The jump across the phase boundary is typical in miscible problems, because the derivatives of the Jacobian contain discontinuities at the phase boundary. For example, (8.26) means that the liquid fraction and, by extension, the saturations have derivatives with respect to pressure in the two-phase region but do not have any derivatives in the single-phase regions. The second observation is that the path for the overall composition model is regular in compositional space due to the default value of `dzMaxAbs=0.1` but makes large jumps in saturation space. For the natural variables model, the compositional changes are small, because the limit on saturation updates (`dsMaxAbs=0.1`) forces the algorithm to advance in regular saturation steps. Altogether, the overall composition model uses 10 iterations and the natural variables 12, but the numbers could equally well have been reversed if large changes in composition had resulted in small changes in the saturation.

The differences between the formulations are also apparent if we inspect the convergence output in verbose mode. For natural variables, the residuals for the

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

fugacity constraints are included in addition to the pressure increment and the residual for the component-mass balances:

```
=================================================================
| It # | deltaP   | Water     | CO2       | f_Water   | f_CO2     |
=================================================================
|    1 | Inf      |*2.56e-15 | 1.66e+02 |*0.00e+00 |*0.00e+00 |
|    2 | 3.61e-01 | 9.95e+01 | 1.44e+02 | 3.31e-02 | 2.89e+02 |
|    3 | 5.86e-02 | 8.44e+01 | 1.48e+02 | 2.95e-02 | 2.03e+02 |
                            :
```

The fugacity residuals are exactly equal zero initially when the cell is at single-phase conditions but become nonzero once the vapor phase appears. The fugacity values are absent in the overall composition output, because these are handled by the separate nonlinear solve from the EoS:

```
=======================================
| It # | deltaP   | Water     | CO2       |
=======================================
|    1 | Inf      |*2.69e-16 | 1.66e+02 |
|    2 | 3.61e-01 | 8.22e+01 | 1.55e+00 |
|    3 | 1.29e-01 | 4.15e+01 | 1.02e+02 |
                :
```

We can also examine the final state after the solution to see what the standard outputs are for a compositional model:

```
disp(solMole)
```

```
                    :
          pressure: 5.0012e+06
                 s: [0.0145 0.9855]
        components: [0.1928 0.8072]
                 L: 0.1921
                 K: [0.0017 320.8334]
               Z_V: 0.6862
               Z_L: 0.0424
                 x: [0.9969 0.0031]
                 y: [0.0017 0.9983]
              flag: 0
                 T: 303.1500
                    :
```

For brevity, we have omitted a few standard fields that are common to all states produced in AD-OO, but we can still see the familiar pressure, temperature, and phase saturations for each cell. In addition, we have the `components` field that stores the overall mole fraction for each component, the liquid and vapor phase mole fractions given in `x` and `y`, as well as the phase compressibility factors `Z_L` and `Z_V`. Strictly speaking, the system is completely determined by these values,

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

but we also store the K-values, the liquid mole fraction L, and the phase state as `flag` to make the EoS coupling easier. In the single-phase region, the K-values stored will be the values from the last multiphase solution in that cell, which can be used as an initial guess if multiphase conditions occur again.

Choice of formulation: Picking the right formulation depends on the scenario to be simulated. The advantages of the overall composition formulation should be clear around the critical point, where a large number of local flash iterations may otherwise impede the convergence of the global system of equations. The formulation is also less involved to implement, because there are no variable switches and the integration with new types of flash is less invasive, especially if source code for the flash calculation is not available. The difference in implementation complexity is readily apparent: At the time of writing, the natural variables base class accounts for approximately 500 lines of code, whereas the overall composition code consists of less than 200 lines of code. A primary disadvantage is that the flash can spend significant time to converge for intermediate mass distributions in physical space, unless one uses a parameterization approach like compositional space adaptive tabulation [49, 50]. Likewise, saturations cannot be directly relaxed to alleviate convergence issues associated with sharp gradients in the flux functions arising from large mobility contrasts or the form of the relative permeability curves. Instead, the solver limits the maximum allowable overall composition change but, as we have seen, the relationship between changes in composition and saturation can be highly nonlinear.

The primary advantage of the natural variables formulation is that the scheme produces a standard Newton method in the two-phase region where all variables can be safely relaxed, a crucial feature for steep relative permeability curves and strong capillary pressure. The Schur complement in (8.29) is analogous to the equation for differentiation of secondary properties in (8.26) for the overall composition formulation. The natural variables formulation is in practice somewhat more expensive in terms of the total assembly cost, because additional derivatives are required for the mass-balance equations to perform the Schur complement. As a rule of thumb, the assembly of the full set of natural variable equations is approximately 10% more expensive with the fastest AD backend options described in Chapter 6 for models with more than 100 000 cells. In addition, the Schur complement must be performed for many of the linear solvers. For further comparisons of various compositional formulations, we refer you to Voskov and Tchelepi [48], Zaydullin et al. [56], and references therein.

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

Table 8.1 *Model classes implemented in the compositional module of MRST.*

Class name	Formulation	Note
`ThreePhaseCompositionalModel`	—	Virtual base class
`OverallCompositionCompositionalModel`	Overall composition	
`NaturalVariablesCompositionalModel`	Natural variables	
`GenericOverallCompositionModel`	Overall composition	Generic model
`GenericNaturalVariablesModel`	Natural variables	Generic model

Computer exercises

1. If the variable `maxChange` is declared in `showNaturalOverall`, the value is used for the maximum allowable changes for saturations and compositions. How does the path change with the value? Are there any large values for which the solvers are unable converge?
2. We used a very simple two-component mixture. The `showNaturalOverall2` example runs a similar experiment with a choice of many different mixtures from the literature. Experiment by varying the mixture, pressure, and temperature conditions. When does the natural variables formulation outperform the overall composition formulation and vice versa?

8.4.4 Implementation as Generic Models

As explained in the introduction to this section, the `compositional` module offers two different implementations of the natural variables and overall composition formulations; see Table 8.1. The first implementation follows the principles set out for the black-oil models in section 12.2 of the MRST textbook [21]; that is, `ThreePhaseBlackOilModel` and simplified versions thereof implemented in the `ad-blackoil` module. When developing similar compositional classes, we realized that even though the original AD-OO framework has a lot of useful abstractions that greatly simplify the process of prototyping simulators for new types of flow physics, describing model equations and computing fluid properties and accumulation, flux, and source terms using large pieces of monolithic code was not an optimal choice and made inclusion of new features unnecessarily complicated. This spurred research into new approaches for further modularization, which resulted in the new concept of state functions and generic model classes introduced in Chapter 5.

Herein, we do not discuss the monolithic implementation in any detail; the corresponding model classes are considered to be legacy code but are kept as part of the official MRST release for backward compatibility, because the generic models

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

inherit functionality from them, and because they form a basis for the `shale` module described in Chapter 10.

Generic compositional components: The monolithic implementation assumes that an aqueous phase and two hydrocarbon phases (liquid and vapor) are always present by default and uses `if` statements scattered throughout the code to treat special cases when any of these phases are not present. The generic approach, on the other hand, builds the configuration of the flow model at runtime as a collection of individual components that each may belong to certain predefined categories exhibiting specific behavior.

From black-oil models, we have immiscible and black-oil components (see Subsection 5.4.2), whereas the derived models for chemical EOR introduce the additional category of concentration components (see Subsection 7.3.3). The compositional module introduces an additional component called `EquationOfStateComponent`. To better understand the design of the compositional module, let us look in detail at how this component computes densities for two phases at VLE, according to the formula:

$$\boldsymbol{\rho}_{i,\alpha} = \begin{cases} \mathbf{X}_{i,\ell}\boldsymbol{\rho}_{\ell}, & \text{if } \alpha = \ell, \\ \mathbf{X}_{i,v}\boldsymbol{\rho}_{v}, & \text{if } \alpha = v, \\ 0, & \text{otherwise.} \end{cases} \tag{8.33}$$

We have already presented the member function that computes this formula in Subsection 5.4.2, but let us present it explicitly again for completeness:

```
function c = getComponentDensity(component, model, state, varargin)
    c = component.getPhaseComposition(model, state, varargin{:});
    rho = model.getProps(state, 'Density');
    for ph = 1:numel(c)
        if ~isempty(c{ph}),  c{ph} = rho{ph}.*c{ph};  end
    end
end
```

The interesting part is how we compute the phase composition. This is done by another member function (presented in a slightly condensed form):

```
function c = getPhaseComposition(component, model, state, varargin)
    massFractions = model.getProps(state, 'ComponentPhaseMassFractions');
    for ph = 1:size(massFractions, 2)
        mf = massFractions{component.componentIndex, ph};
        if ~isempty(mf),  c{ph} = mf;  end
    end
end
```

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

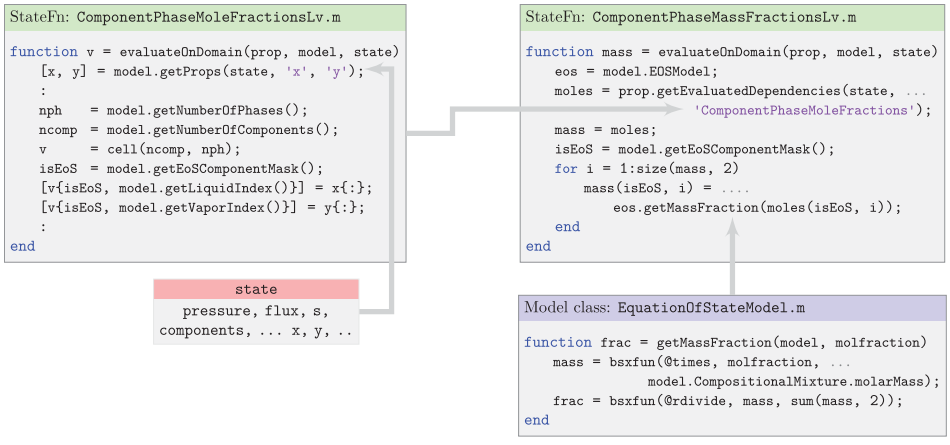


Figure 8.9 Computation of component mass fractions for a generic EoS component.

The component mass fractions $\mathbf{X}_{i,\alpha}$ are computed through a combination of state functions and member functions from the associated `EquationOfStateModel`, as shown in Figure 8.9. The code excerpts in the figure have been edited and simplified to get rid of data conversion and special cases that obscure the expressions in use. If we try to analyze the code, we see that the primary state function, shown to the upper right, starts by calling the secondary state function, shown to the upper left, to compute component mole fractions. All that the part shown here of this state function does is to extract the mole fractions for each phase from the state object and store them in a cell array that runs over all components and all phases; we have purposely edited out the parts that insert ones in the correct places for non-EoS components. Once the mole fractions are computed, the primary state function then iterates over all components and calls on a member function from the EoS model to compute the mass fractions, using the expression $X_i = m_i x_i / \left(\sum_i^N m_i x_i \right)$. The member function does this irrespective of whether `molfraction` is a cell array or an ordinary array of doubles, and here we only present the latter variant for pedagogical purposes, because it is more compact than the cell-array equivalent (which is the one usually called when executing the code).

Instantiating compositional components: In Subsection 5.4.2, we explained how the individual components that make up the flow model are instantiated by `validateModel` and showed the relevant code. We follow exactly the same approach for the compositional case, here from the overall composition class:

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

```
for ci = 1:nc
    name = names{ci};
    switch name
        case {'water', 'oil', 'gas'}
            ix = model.getPhaseIndex(upper(name(1)));
            c = ImmiscibleComponent(name, ix);
        otherwise
            c = getEOSComponent(model, p, T, name, ci);
    end
    model.Components{ci} = c;
end
```

The logic of this function is that components named `'water'`, `'oil'`, and `'gas'` represent single-component, immiscible fluid phases that can be instantiated directly. All other components are assumed to be part of the hydrocarbon phases that can be found in liquid and vapor form. Notice that we cannot instantiate these components directly but must instead first determine how each component splits across the phases found at the present temperature and pressure. This is done in the separate utility function shown in Listing 8.1, which first performs a standalone flash as discussed in Subsection 8.3.3 to compute the liquid-phase mole fraction and the phase densities and then subsequently uses these to instantiate the component.

8.4.5 State Functions for Compositional Models

The `compositional` module implements state functions for evaluating flow and thermodynamic properties and discretization terms in flow and facility equations based on the principles outlined for black-oil models in Chapter 5 and discussed in detail for chemical EOR in Chapter 7. We will therefore not discuss their compositional counterparts in any detail here; instead, we encourage you to study the source code yourself.

A good way to understand the inner workings of MRST has traditionally been to pick a tutorial example and use the debugger in the MATLAB editor to run the code lines in the accompanying script step by step. By stepping into each function call, you can follow the exact execution of the code and thus understand the logic of the implementation. This approach will not be as intuitive for simulator scripts implemented with the AD-OO framework as for the procedural parts of MRST, because the abstractions and inheritance that greatly aid you when implementing enhanced or new functionality also introduce many layers in the code that may be quite overwhelming to step through; indeed, at least the first few times you try, it will feel like peeling a cabbage or an onion. This is particularly true for the new state-function framework, which involves a lot of overhead code to implement

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

Listing 8.1 *Perform initial flash and instantiate EoS components.*

```
function c = getEOSComponent(model, p, T, name, ci)

    mixture = model.EOSModel.CompositionalMixture;
    hcpos   = strcmp(mixture.names, name);
    z       = zeros(1, numel(mixture.names)); z(hcpos) = 1;

    % Flash calculation to obtain molar fractions and densities
    [L, ~, ~, ~, ~, rhoL, rhoV] = standaloneFlash(p, T, z, model.EOSModel);
    Lm   = L.*rhoL./(rhoL.*L + rhoV.*(1-L));
    frac = zeros(1, model.getNumberOfPhases());

    % Assign mass fractions and phase densities
    Li = model.getLiquidIndex();  Vi = model.getVaporIndex();
    frac(Li) = Lm;                frac(Vi) = 1-Lm;
    rho(Li)  = rhoL;              rho(Vi) = rhoV;

    % Densities for any non-EoS components
    extra = model.getNonEoSPhaseNames();
    phases = model.getPhaseNames();
    for i = 1:numel(extra)
        rho(phases == extra(i)) = model.fluid.(['rho', extra(i), 'S']);
    end

    c = EquationOfStateComponent(name, p, T, ci, frac, rho, ...
                                 mixture.molarMass(hcpos));
end
```

the very useful compute-cache mechanism and efficient evaluation of constitutive relationships for cases with multiple fluid and/or PVT regions.

The display and plotting functionality discussed in Subsection 5.4.1 has been introduced for the exact purpose of remedying this situation, and using this to display the exact state functions used and view different parts of the simulator as graphs is our recommended approach to understand the logic of the `compositional` module. To illustrate, let us revisit the example from Subsection 8.4.3 (script: `showNaturalOverall.m`) and use the overall compositional formulation as an example. Once the `overall` class object is constructed, we can extract its state-function groupings:

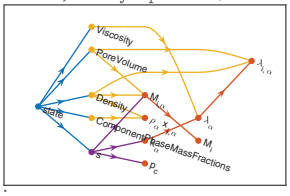
```
ogroups = overall.getStateFunctionGroupings()
```

```
ogroups =
  1×4 cell array

Columns 1 through 2
  {1×1 FlowPropertyFunctions}    {1×1 PVTPropertyFunctions}

Columns 3 through 4
  {1×1 FlowDiscretization}    {1×1 FacilityFlowDiscretization}
```

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

These are the same four groupings as shown for the black-oil case in Figure 5.7. We start by inspecting the flow properties:

```
FlowPropertyFunctions (edit|plot) state function grouping instance.

Intrinsic state functions (Class properties for FlowPropertyFunctions, always present):
     CapillaryPressure: BlackOilCapillaryPressure (edit|plot)
     ComponentMobility: ComponentMobility (edit|plot)
 ComponentPhaseDensity: ComponentPhaseDensity (edit|plot)
    ComponentPhaseMass: ComponentPhaseMass (edit|plot)
    ComponentTotalMass: ComponentTotalMass (edit|plot)
              Mobility: Mobility (edit|plot)
  RelativePermeability: BaseRelativePermeability (edit|plot)
No extra state functions have been added to this class instance.
```



The graph is reproduced so small here that it is difficult to see details, but if you repeat the exercise and plot it using the command

```
plotStateFunctionGroupings(ogroups{1})
```

on your own computer, you will see that it has four "end nodes": capillary pressure and mobility, phase density, and total mass for each component. The state function `CapillaryPressure` computes the saturation-dependent capillary pressure functions $P_c(S)$. This quantity is defined in the same way for our compositional model, as in the black-oil case, and is thus inherited from the previous implementation.

Moving on to the component quantities, we see that the yellow lines in the graph show that these all depend on thermodynamic properties, but the graph does not offer any details of how these prerequisites are computed. To also include the interdependencies of the thermodynamic quantities, we can redo the plot as follows to produce Figure 8.10:

```
endnodes = {'CapillaryPressure', 'ComponentPhaseDensity',...
            'ComponentTotalMass','ComponentMobility'};
plotStateFunctionGroupings(ogroups(1:2),'Stop',endnodes,'label','name');
```

We have already discussed the computation of component phase density (8.33) in Subsection 8.4.4, which is implemented inside the `EquationOfStateComponent` class. The graph enables us to trace all dependencies: The first dependence is on the `Density` function in the PVT group, which in turn depends on `PhasePressures` and `p` and `T` from the `state` object. In the general case, `PhasePressures` would also depend on the `CapillaryPressure` just discussed, but this dependency is disregarded when the capillary pressures are turned off, as in this particular example.

The second dependence is on phase mass fractions, which we previously have discussed in detail and shown in Figure 8.9. At this point you may ask why there

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

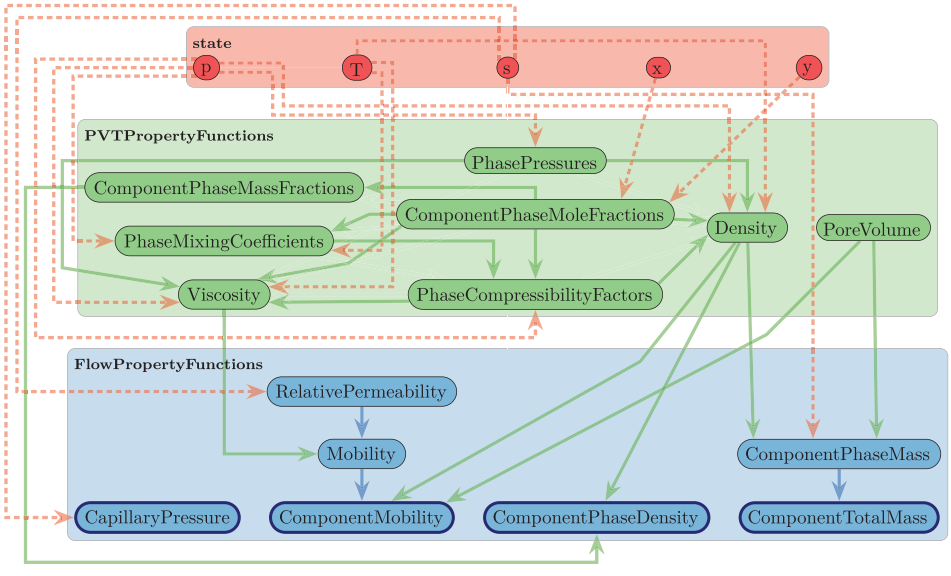


Figure 8.10 State-function diagram for the `FlowPropertyFunctions` group, including dependencies from the PVT property group. Solid lines denote dependencies between state functions, and dashed lines are dependencies on properties from the state object. End nodes – i.e., state functions that do not provide input to any other functions in the flow property group – are marked with a blue frame. (Notice also that the TikZ code exported from MRST has been edited a bit to improve the visual presentation.)

is no trace of the EoS in the graph. The reason is that all EoS models known to the `compositional` module are implemented as a monolithic class without the use of state functions and hence none of the internal function calls implemented in this class show up in the current graph.

We end this discussion by pointing out one important limitation of this plotting functionality. It does not do any kind of automatic code analysis to detect dependencies but instead relies on dependencies documented when coding each individual state function or component class. Such dependencies are registered in the constructor of each state function or component. We can take the EoS component class as an example. Here, the constructor reads:

```
function c = EquationOfStateComponent(name, p, T, cindex, ..)
    :
    c = c.functionDependsOn('getPhaseComposition', ...
            {'ComponentPhaseMassFractions'}, 'PVTPropertyFunctions');
    c = c.functionDependsOn('getComponentDensity', ...
            {'Density','ComponentPhaseMassFractions'}, 'PVTPropertyFunctions');
end
```

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

Looking back at how `getComponentDensity` is implemented on page 355, we see that the dependence on the component mass fractions does not appear in the function itself but is introduced through the call to `getPhaseComposition`. It must nonetheless be registered explicitly also for `getComponentDensity`, and it is essential that such dependencies are traced out and registered diligently for the plotting functionality to be accurate and useful.

Computer exercises

1. Use the approach just outlined to familiarize yourself with the other three state-function groupings that make up the compositional simulator classes.
2. Try to make a plot similar to the one shown in Figure 5.7 that displays the interdependencies among all of the quantities you need to evaluate to compute accumulation, fluxes, and source terms. (Hint: To distinguish state functions with the same name from two different groupings, you must prepend the group name; e.g., `FlowDiscretization.ComponentTotalFlux`.)

8.4.6 Limitations and Caveats

The compositional module in MRST has sufficient complexity to model a range of different multicomponent scenarios. Chapter 10 discusses how the module can be extended with sorption, diffusion, and geomechanics effects and be combined with embedded discrete fracture modeling to describe recovery from unconventional oil and gas reservoirs. In the interest of clarity, however, we mention a few features not present in the module. These features may come in the future – or you may be the one who contributes to any of them.

The compositional model neglects capillary pressure in the phase-equilibrium calculations. For modeling of hydrocarbon mixtures, this is quite reasonable and is standard in commercial compositional simulators, but it is less reasonable for VLE involving water or CO_2. There are a number of recent works [33, 45] that incorporate these effects that could inform an MRST extension.

Cubic EoS are generally favored for their efficiency and limited number of parameters, but significant interest has also been devoted to PC-SAFT-type equations [15] derived from statistical mechanics. The official release of MRST will soon include the ePC-SAFT version [24, 25], which requires five molecular-based parameters per component for associating fluids and only three for nonassociating ones. Generally, PC-SAFT-type equations describe liquid systems better than traditional cubic EoS. PC-SAFT is more accurate to predict derivative properties, reducing errors by a factor of up to eight [12, 20], while reducing density prediction error by one half. PC-SAFT (or ePC-SAFT) provides good agreement

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

between calculated and experimental properties of reservoir fluids, natural gas, and asphaltene phase behavior [4, 38]. On the other hand, PC-SAFT has issues related, e.g., to root finding and is known to increase the computational time [4, 12, 20, 38].

The `compositional` module does not take thermal effects into account, in that the description is fully isothermal. A natural future extension would be to incorporate the conservation of energy in the solver, which may necessitate the introduction of a flash that uses enthalpy instead of temperature [55] to model internal energy.

Several other smaller features would be easy to implement, given the flexibility of AD together with state functions. One is to finish the partially implemented Zudkevitch–Joffe–Redlich–Kwong EoS [57] that adds temperature dependence to the static Redlich–Kwong parameters, another is to introduce changing relative permeabilities and capillary pressure curves during the transition to fully miscible flooding (e.g., the Eclipse keyword `MISCIBLE` that implements the rule from Coats [9]). Yet other features are supported, but automatically setting these options from input files is not tested to the degree you may be used to from other solvers in MRST. Please report issues and fixes through the usual MRST channels.

8.5 Examples

This section goes through a number of examples to demonstrate how you can use functionality from the `compositional` module to set up relatively complex fluid cases. We also verify the simulator against a commercial simulator and another research code to demonstrate its correctness. All examples come with complete source code, which you can find in the `book-ii` example directory of the module. To keep the discussion as simple as possible, we only present 1D examples, but the solvers in the module are fully capable of simulating cases posed on complex and unstructured grids in 2D and 3D as long as your computer has sufficient memory and processing power. (Notice that use of external, iterative solvers and the accelerated AD backends discussed in Chapter 6 is particularly important when attempting to simulate large 2D/3D cases.)

8.5.1 Validation of MRST's Simulators

We begin by comparing MRST with other reservoir simulators with compositional capability in the `compositionalValidationSimple` example. This test case is taken from Voskov and Tchelepi [48] and was originally reported for MRST in [31]. Here, pure CO_2 is injected over 500 timesteps into a 1D reservoir made up of 1 000 cells. The reservoir initially contains a CO_2–methane–decane mixture mostly in the liquid phase, with CO_2 forming a supercritical phase under reservoir

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

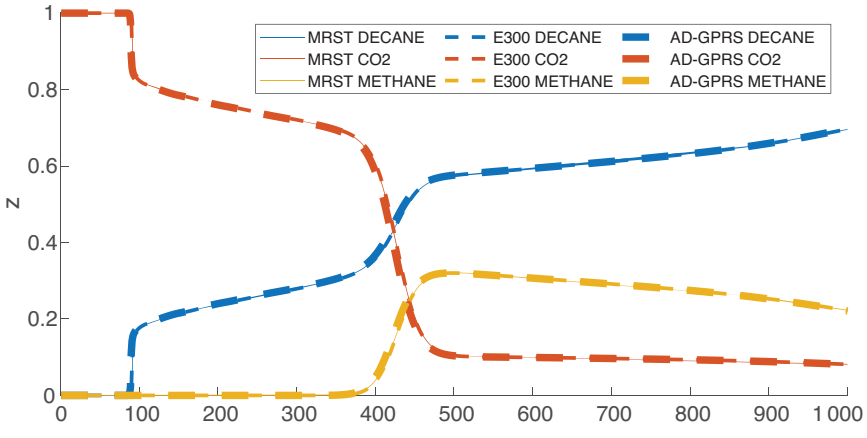


Figure 8.11 Mole fractions after 6 years and 175 days computed by three different simulators for the validation case from Voskov and Tchelepi [48].

conditions of 423.15 K and 75 bar. No K-values are used, which means that the component behavior and phase properties are predicted by the default PR EoS.

The overall mole fractions are plotted for all three simulators: MRST, AD-GPRS [48], and ECLIPSE (E300) [44]. To be as close as possible to E300 in formulation, MRST and AD-GPRS are both set to use the overall composition as primary variables with a fully implicit temporal discretization. Both wells operate at fixed bottom-hole pressures. In computed mole fractions reported in Figure 8.11 we can observe that there is a large two-phase region beyond the region where the supercritical CO_2 front fully saturates the medium. The relative permeabilities are of Corey type, with quadratic exponents, but the front structure is significantly more complex than in the immiscible Buckley–Leverett case from subsection 10.3.1 in the MRST textbook [21]. The variable density, viscosity, and phase mole distribution make this a highly challenging numerical problem to solve. The three simulators are in excellent agreement when formulations and timesteps are comparable. This simple 1D example is just a small sample of the extensive validation performed in-house on the MRST compositional solvers.

8.5.2 Numerical Accuracy

The previous example showed that MRST agrees with a commercial and a state-of-the-art academic simulator for a challenging case with complex phase behavior. As an extension of this, we also compare different formulations for the same case. In `compositionalAccuracyExample`, we set up two additional solvers: a fully implicit natural variables solver and an explicit overall composition solver. The upper part of Figure 8.12 reports mole fractions for all three solvers. The two fully

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

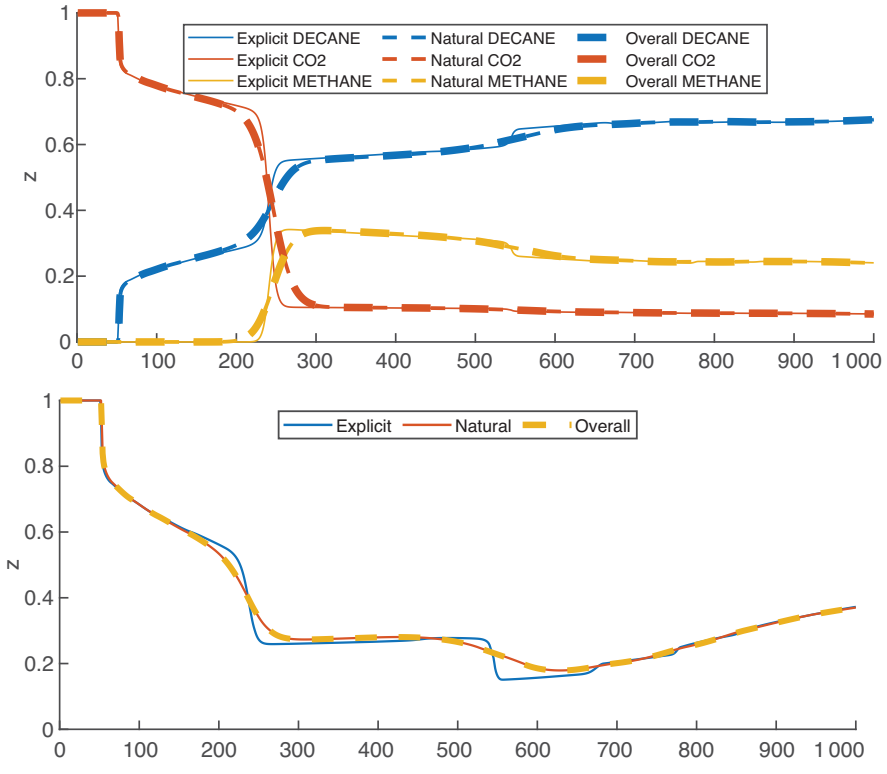


Figure 8.12 Mole fractions (top) and vapor saturation (bottom) for the validation case from Voskov and Tchelepi [48] plotted for three different formulations: fully implicit natural and overall variables and explicit overall variables. The choice of temporal discretization has a large impact on the resolution of the solution, whereas the choice between formulations only matters for the required number of nonlinear iterations.

implicit solvers produce identical results at roughly comparable iteration numbers of 1 466 for natural variables and 1 575 for the overall composition formulation. For the explicit solver, we see a significant improvement in the sharpness of the discontinuities not only for the two trailing discontinuities but in particular for the leading, weak discontinuity positioned at $x \approx 545$. We can also discern that fine details ahead of this front are lost in the fully implicit solvers. We can see these more clearly in the saturation profile plotted in the lower half of Figure 8.12. The nonmonotone saturation is due to the production well, where additional vapor is formed as light components escape the fluid phase when the pressure decreases closer to the producer.

Note that whereas the two fully implicit simulations match perfectly, they ended up smearing out important features, even for a high-resolution setup with 500 timesteps on a 1 000-cell grid. Compositional problems are highly susceptible to numerical diffusion, in particular when the pertinent dynamics consists of com-

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

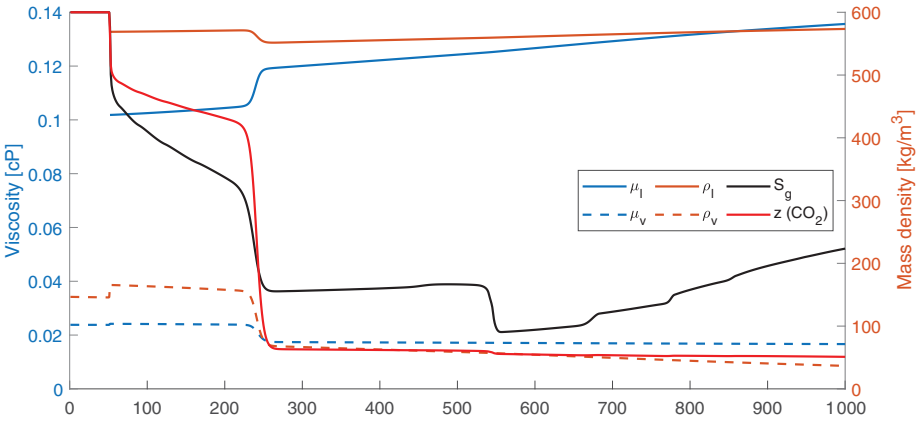


Figure 8.13 Plots of key flow properties along the front demonstrate the behavior of a compositional description: phase viscosities and densities change significantly depending on the composition in each point.

ponents being advected in the single-phase regions as discontinuities with no or weak associated self-sharpening mechanisms. (Poor resolution of such waves is also discussed in Chapters 3 and 7.) A full review of the wave structure for such problems is outside the scope of this chapter, but the interested reader is directed to the excellent textbook by Orr [36] on the subject.

Figure 8.13 reports densities and viscosities for both phases together with the gas saturation and CO_2 mole fraction. The density and viscosity change significantly beyond the CO_2 front, as different components move at different speeds toward the producer, depending on their dynamic liquid solubility.

8.5.3 Surface Volumes and Separators

One especially attractive feature of compositional models is that the PVT description is general and can be used for conditions others than those typically found inside a hydrocarbon reservoir. Other models, such as the black-oil model, use tabulated values that often have a limited range of validity and are only given for a specific gas composition. Compositional models can also accurately represent liquid and vapor mixtures that vary spatially and temporally throughout a simulation, which makes it easier to get accurate production data.

There is, in general, no limit to how complex models for production facilities can be for geoenergy applications. Well-developed gas fields can, in particular, have very complex pipe networks that transport produced gas out from the field or back for reinjection. MRST includes a basic implementation of *separators* for the purpose of controlling wells by surface rates in cases when the relationship between composition and surface phase formation is strong for produced mixtures.

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

The default behavior in the compositional module performs a flash for injection mixtures to determine the surface density prior to simulation start. By default, MRST uses a simple rule for produced liquids: The surface density specified in the fluid object – for example, `fluid.rhoGS` for the default vapor phase – is used to set liquid and vapor rates. Components are separated into liquid and vapor streams according to their phase fractions at the standard conditions specified by the facility model.[4] The default behavior makes it easy to control mass rates and does not require a runtime flash at surface conditions, and it is similar to the black-oil model for which the gas and oil components by definition are found exclusively in the vapor and liquid, respectively, at the standard conditions.

For some scenarios, it is more natural to work with complex mixtures at surface conditions. Light and intermediate components can be present in the liquid phase at these conditions, provided that there is a sufficient amount of heavy components to form a stable liquid phase to dissolve into. The configuration options for separators are demonstrated in the `compositionalSeparatorExample` script. The reservoir is described on a uniform Cartesian grid with $11 \times 11 \times 10$ cells and homogeneous petrophysical properties. The hydrocarbon fluid system is modeled using six components (C_1, C_3, C_6, C_{10}, C_{15}, and C_{20}) and described by the fluid model from the Fifth SPE Comparative Solution Project [18], which you may recall we used as our second example when discussing the standalone flash in Subsection 8.3.2. The other fluid properties are extracted from a standard black-oil oil–gas model with quadratic Corey relative permeabilities and a surface density ratio of 100:1 between the liquid and vapor phases. A single injector is placed in the middle of the domain, operating at a fixed rate that extracts a significant volume so that a vapor phase eventually is formed as the pressure drops, reminiscent of the closed boundary version of the black-oil case discussed in subsection 12.4.1 of the MRST textbook [21].

Once we have set up the base case with standard code, omitted here for brevity, we can define a model with a single separator by copying `model` to `model_sep` and modifying the associated facility model:

```
s  = EOSSeparator('pressure', 1*atm, 'T', 300); % Set conditions for surface
sg = SeparatorGroup(s);                         % Group = single separator
sg.mode = 'moles';                              % Use mole mode
model_sep.FacilityModel.SeparatorGroup = sg;    % Connect to reservoir model
```

The separator will flash the surface streams at the surface conditions of 300 K and 1 atm pressure and use the result to determine the appropriate volume to extract at reservoir conditions to meet the target depletion rate.

[4] The default values are the metric standard conditions for gas of 273.15 K and 101.325 kPa.

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

We can also go one step further and construct an alternative simulation model `model_msep` in which we replace the single separator by a tree structure of separators that each operates at different conditions:

```
p = [200, 175, 50, 10]*barsa;          % p for each separator
T = [423, 400, 350, 300];              % T for each separator
dest =  [2, 3; ...                     % Send liquid to 2, vapor to 3
         0, 4; ...                     % Send liquid to tank, vapor to 4
         4, 0; ...                     % Send liquid to 4, vapor to tank
         0, 0];                        % Send both liquid and vapor to tank
sg = SeparatorGroup(s, p, T, dest);  % Construct separator group
model_msep.FacilityModel.SeparatorGroup = sg; % Connect to reservoir model
```

By default, the group object automatically copies `s` for each stage and sets the conditions. The `dest` variable is of special importance because it encodes the flow between each stage. It is essentially a directed graph, in which the first column contains the *liquid targets* and the second column contains the *vapor targets*. The produced mixture will first be passed to the separator operating at 200 bar pressure, which will pass the liquid stream on to separator 2 and vapor on to separator 3. These pass their vapor and liquid streams, respectively, on to the fourth separator while sending the other phase to the surface tanks (denoted by a zero value). The fourth separator mixes the incoming streams, flashes again at surface conditions, and passes any liquid and vapor to the respective surface tanks. The overall flow rates into these tanks are used to determine the volume we need to extract at reservoir conditions to meet targeted depletion rates.

In Figure 8.14, we see that the use of even just a single separator significantly changes the depletion rate, as evidenced by the different pressure drop and formation of gas in the well cell. Setting up additional separators changes the behavior

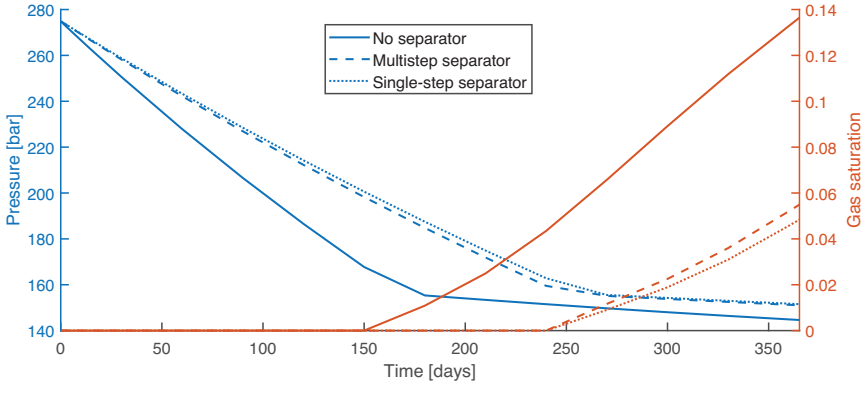


Figure 8.14 The separator test. Pressure and gas saturation in the well cell as a function of time shown on the left and right axes, respectively.

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

to a lesser extent. This case illustrates that (i) it is possible to perform complex phase separation during simulation and (ii) the definitions of surface gas and oil production are both not straightforward *and* very important when working with rate-type controls for compositional models.

Using a tree of separators, one can separate out light and heavy components by exploiting the changes in mixture composition at different conditions. When a separator is set up as a tree, cyclical connections are disallowed from the destination matrix. Note that the separators use the EoS in the `EOSModel` field. The default value of empty means that the separator will obtain the necessary EoS from the reservoir model, using the same flash for both reservoir and separator stages. It is possible to have different EoS instances for each of the stages if desired; for example, if different settings are required at changing pressure and temperature conditions on the path up to the surface.

8.5.4 Miscibility

Simulation of miscible processes is a classical application for compositional simulators. The displacement efficiency of gas injection depends highly on the in situ reservoir pressure and temperature conditions. If the displacement front is kept at miscible conditions above the minimum miscibility pressure, the displacement is piston-like because there is no surface tension between the residual oil and the injected gas. If conditions away from the injection site fall below the minimum miscibility pressure, the immiscible behavior leads to reduced sweep efficiency, because the unfavorable viscosity ratio between the vapor phase formed by injection gas and the reservoir oil leads to the formation of viscous fingers and lowered recovery.

In `compositionalMiscibilityExample`, we inject a fixed mass of CO_2 at the left-hand boundary into a model that initially contains CO_2, C_1, C_5, and C_{12} in a pure liquid phase found at 150 bar pressure. The setup is based on tests reported by Alzayer et al. [2]. A producer operates at fixed bottom-hole pressure at the rightmost end of the domain. We perform seven simulations in which the pressure at the producer is varied from 70 to 150 bar. The resulting vapor saturation profile is reported at the same timestep for all scenarios in Figure 8.15. This 1D example can be thought to model a slim-tube setup, where miscibility conditions can be determined experimentally. Although the initial and injected masses are identical in all cases, we see that the displacement processes change significantly depending on the producer pressure. At the lowest pressure, the immiscible displacement front propagates quickly through the reservoir and is near breakthrough at the plotting step, whereas for the fully miscible displacement at the highest pressure, the piston-like front has barely covered one-fifth of the domain. It is also possible to

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

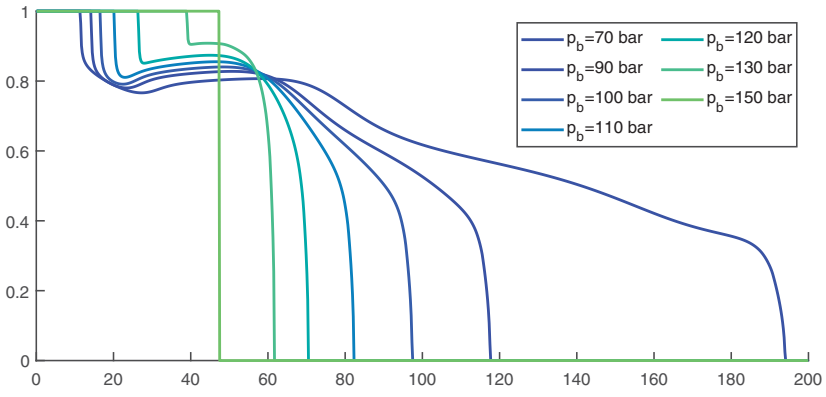


Figure 8.15 Demonstration of miscibility as a function of pressure. The vapor saturation is plotted after the same amount of CO_2 has been injected for seven different producer pressures, ranging from immiscible to fully miscible.

use the `solvent` module in MRST to simulate some miscibility scenarios with a more lightweight, tabulated model that extends the standard black-oil model.

8.5.5 Performance of Compositional Solvers

In `compositionalPerformanceExample`, we perform a simple test of the solver speed for a compositional model. The model is, by default, defined for a $50 \times 50 \times 50$ Cartesian grid with a single producer–injector pair with the six-component SPE 5 fluid model [18] for a total of 750 000 reservoir degrees of freedom. You can easily modify the script switch to use another fluid mixture or change the grid and then perform the test on your own computer.

The compositional equations contain a large number of element-wise sums of products for each component (see, e.g., (8.18) and (8.19)) that are fairly expensive for AD, because many intermediate objects are created and stored. These types of operations led to the creation of the diagonal backends described in Chapter 6. In the test reported herein, we compute a small timestep with four different AD backends. The first two backends are implemented purely in MATLAB and consist of the default sparse and the diagonal backends, whereas the last two are different variants of MEX-accelerated diagonal backends.

The results in Figure 8.16 demonstrate that the time per assembly drops from almost 9 seconds to just under 1.5 seconds (i.e., is reduced by a factor of 6) by switching to the best choice for this model. In addition, the linear solver time is significantly reduced by the same switch, because the matrix storage is better for the row-major, MEX-accelerated, diagonal AD backend. More so than for any

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

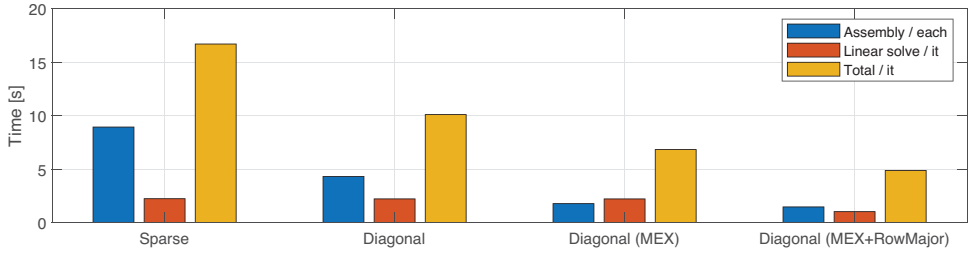


Figure 8.16 Runtime for assembly and linear solve on a $50 \times 50 \times 50$ grid with the SPE 5 fluid model with different AD backends discussed in Chapter 6.

other models in MRST, compositional simulation quickly becomes infeasible with the standard sparse backend and MATLAB's direct solvers, because the number of degrees of freedom equals the number of cells multiplied by the number of components for the most compact formulation.

For further details on the benchmarking methodology, how different backends can improve execution time, possible caveats, and the hardware used for the test, please see Chapter 6.

8.6 Concluding Remarks

In this chapter, we have introduced you to the physical principles, equations, and numerical solution strategies underlying the compositional module in MRST. As explained earlier in the chapter, the compositional module offers two different types of implementations of the overall component and natural variable formulations. Whereas the monolithic approach is likely to stay with MRST for several years, we have herein chosen to only focus on the generic model classes. These have not only been used to implement improved spatial and temporal discretizations, as seen in Chapter 5, but all new developments of accelerated computing, improved support for wells and facility modeling, and so on are geared toward the generic approach. We therefore strongly recommend that you use the corresponding simulator classes to conduct simulations or as a basis for your own development on top existing functionality.

To illustrate typical behavior of compositional systems and teach you the basics of setting up and running simulations using the compositional module, we presented a set of relatively simple simulation cases that favor clarity in description over complexity of results. We emphasize that these examples by no means illustrate the general capabilities of the module when it comes to grid types, well configurations, simulation schedules, etc. To get a better idea of the type of multidimensional and challenging simulations you can run with

https://doi.org/10.1017/9781009019781.014 Published online by Cambridge University Press

the module, we suggest that you consult the multiscale-compositional example in Subsection 4.3.9, the shale examples in Chapter 10, or some of the papers that have used the compositional module as a research tool to investigate multiscale methods [31], new sequentially fully implicit methods [30], hybrid 3D/vertical-equilibrium simulation [29], and adaptive coarsening methods [19].

Acknowledgement. The author thanks Knut-Andreas Lie for invaluable help in finishing this text and ensuring the quality of the finished manuscript. Parts of the compositional module has been written as a part of VISTA project 6366. VISTA is a basic research program funded by Equinor and conducted in close collaboration with The Norwegian Academy of Science and Letters. The author thanks Arthur Moncorgé (TOTAL E&P) and Denis Voskov (TU Delft) for many fruitful discussions on compositional flow.

References

[1] G. Acs, S. Doleschall, and E. Farkas. General purpose compositional model. *SPE Journal*, 25(4):543–553, 1985. doi: 10.2118/10515-PA.

[2] A. Alzayer, D. Voskov, and H. Tchelepi. On modification of relative permeability in compositional simulation of near-miscible processes. In *ECMOR XV-15th European Conference on the Mathematics of Oil Recovery, August 29–September 1, Amsterdam, the Netherlands*, pp. cp–494. European Association of Geoscientists & Engineers, 2016. doi: 10.3997/2214-4609.201601741.

[3] I. H. Bell, J. Wronski, S. Quoilin, and V. Lemort. Pure and pseudo-pure fluid thermophysical property evaluation and the open-source thermophysical property library CoolProp. *Industrial & Engineering Chemistry Research*, 53(6):2498–2508, 2014. doi: 10.1021/ie4033999.

[4] W. A. Burgess, D. Tapriyal, B. D. Morreale, Y. Soong, H. O. Baled, R. M. Enick, Y. Wu, B. A. Bamgbade, and M. A. McHugh. Volume-translated cubic EoS and PC-SAFT density models and a free volume-based viscosity model for hydrocarbons at extreme temperature and pressure conditions. *Fluid Phase Equilibria*, 359:38–44, 2013. doi: 10.1016/j.fluid.2013.07.016.

[5] H. Cao. Development of techniques for general purpose simulators. PhD thesis, Stanford University, Stanford, CA, 2002. URL `pangea.stanford.edu/ERE/pdf/pereports/PhD/Cao02.pdf`

[6] W. G. Chapman, K. E. Gubbins, G. Jackson, and M. Radosz. SAFT: equation-of-state solution model for associating fluids. *Fluid Phase Equilibria*, 52:31–38, 1989. doi: 10.1016/0378-3812(89)80308-5.

[7] W. G. Chapman, K. E. Gubbins, G. Jackson, and M. Radosz. New reference equation of state for associating liquids. *Industrial & Engineering Chemistry Research*, 29(8):1709–1721, 1990. doi: 10.1021/ie00104a021.

[8] M. C. H. Chien, S. T. Lee, and W. H. Chen. A new fully implicit compositional simulator. In *SPE Reservoir Simulation Symposium*, TX. Society of Petroleum Engineers, 1985. doi: 10.2118/13385-MS.

[9] K. H. Coats. An equation of state compositional model. *Society of Petroleum Engineers Journal*, 20(5):363–376, 1980. doi: 10.2118/8284-PA.

[10] K. H. Coats. Simulation of gas condensate reservoir performance. *Journal of Petroleum Technology*, 37(10):1–870, 1985. doi: 10.2118/10512-PA.

[11] D. A. Collins, L. X. Nghiem, Y. K. Li, and J. E. Grabonstotter. An efficient approach to adaptive-implicit compositional simulation with an equation of state. *SPE Reservoir Engineering*, 7(2):259–264, 1992. doi: 10.2118/15133-PA.

[12] A. J. De Villiers, C. E. Schwarz, A. J. Burger, and G. M. Kontogeorgis. Evaluation of the PC-SAFT, SAFT and CPA equations of state in predicting derivative properties of selected non-polar and hydrogen-bonding compounds. *Fluid Phase Equilibria*, 338:1–15, 2013. doi: 10.1016/j.fluid.2012.09.035.

[13] L. T. Fussell and D. D. Fussell. An iterative technique for compositional reservoir models. *Society of Petroleum Engineers Journal*, 19(4):211–220, 1979. doi: 10.2118/6891-PA.

[14] J. W. Gibbs. *The Collected Works of J. Willard Gibbs*. Yale University Press, New Haven, CT, 1948.

[15] J. Gross and G. Sadowski. Perturbed-chain SAFT: an equation of state based on a perturbation theory for chain molecules. *Industrial & Engineering Chemistry Research*, 40(4):1244–1260, 2001. doi: 10.1021/ie0003887.

[16] A. Iranshahr, D. Voskov, and H. A. Tchelepi. Generalized negative-flash method for multiphase multicomponent systems. *Fluid Phase Equilibria*, 299(2):272–284, 2010. doi: 10.1016/j.fluid.2010.09.022.

[17] A. Iranshahr, D. Voskov, and H. A. Tchelepi. A negative-flash tie-simplex approach for multiphase reservoir simulation. *SPE Journal*, 18(6):1140–1149, 2013. doi: 10.2118/141896-PA.

[18] J. E. Killough and C. A. Kossack. Fifth comparative solution project: evaluation of miscible flood simulators. In *SPE Symposium on Reservoir Simulation, 1–4 February, San Antonio, Texas*. Society of Petroleum Engineers, 1987. doi: 10.2118/16000-MS.

[19] Ø. Klemetsdal, O. Møyner, and K.-A. Lie. Implicit high-resolution compositional simulation with optimal ordering of unknowns and adaptive spatial refinement. In *SPE Reservoir Simulation Conference, 10–11 April, Galveston, Texas*. Society of Petroleum Engineers, 2019. doi: 10.2118/193934-MS.

[20] S. Leekumjorn and K. Krejbjerg. Phase behavior of reservoir fluids: comparisons of PC-SAFT and cubic EOS simulations. *Fluid Phase Equilibria*, 359:17–23, 2013. doi: 10.1016/j.fluid.2013.07.007.

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

[22] J. Lohrenz, B. G. Bray, and C. R. Clark. Calculating viscosities of reservoir fluids from their compositions. *Journal of Petroleum Technology*, 16(10):1–171, 1964. doi: 10.2118/915-PA.

[23] J. J. Martin. Cubic equations of state – which? *Industrial & Engineering Chemistry Fundamentals*, 18(2):81–97, 1979. doi: 10.1021/i160070a001.

[24] M. Masoudi, R. Miri, H. Hellevang, and S. Kord. Modified PC-SAFT characterization technique for modeling asphaltenic crude oil phase behavior. *Fluid Phase Equilibria*, p. 112545, 2020. doi: 10.1016/j.fluid.2020.112545.

[25] M. Masoudi, S. Parvin, R. Miri, S. Kord, and H. Hellevang. Implementation of PC-SAFT equation of state into MRST compositional for modelling of asphaltene precipitation. In *82nd EAGE Annual Conference & Exhibition, 14–17 June, Amsterdam, the Netherlands*, pp. 1–5. European Association of Geoscientists & Engineers, 2020. doi: 10.3997/2214-4609.202011432.

[26] M. L. Michelsen. The isothermal flash problem. Part I. Stability. *Fluid Phase Equilibria*, 9(1):1–19, 1982. doi: 10.1016/0378-3812(82)85001-2.

[27] M. L. Michelsen. The isothermal flash problem. Part II. Phase-split calculation. *Fluid Phase Equilibria*, 9(1):21–40, 1982. doi: 10.1016/0378-3812(82)85002-4.

[28] M. L. Michelsen. Saturation point calculations. *Fluid Phase Equilibria*, 23(2): 181–192, 1985. doi: 10.1016/0378-3812(85)90005-6.

[29] O. Møyner, O. Andersen, and H. M. Nilsen. Multi-model hybrid compositional simulator with application to segregated flow. *Computational Geosciences*, 24:775–787, 2020. doi: 10.1007/s10596-019-09910-y.

[30] O. Møyner and A. Moncorgé. Nonlinear domain decomposition scheme for sequential fully implicit formulation of compositional multiphase flow. *Computational Geosciences*, 24(2):789–806, 2020. doi: 10.1007/s10596-019-09848-1.

[31] O. Møyner and H. A. Tchelepi. A mass-conservative sequential implicit multiscale method for isothermal equation-of-state compositional problems. *SPE Journal*, 23(6):2–376, 2018. doi: 10.2118/182679-PA.

[32] L. X. Nghiem, K. Aziz, and Y. K. Li. A robust iterative method for flash calculations using the Soave–Redlich–Kwong or the Peng–Robinson equation of state. *Society of Petroleum Engineers Journal*, 23(3):521–530, 1983. doi: 10.2118/8285-PA.

[33] D. V. Nichita. Volume-based phase stability analysis including capillary pressure. *Fluid Phase Equilibria*, 492:145–160, 2019. doi: 10.1016/j.fluid.2019.03.025.

[34] R. Okuno, R. Johns, and K. Sepehrnoori. A new algorithm for Rachford–Rice for multiphase compositional simulation. *SPE Journal*, 15(2):313–325, 2010. doi: 10.2118/117752-PA.

[35] R. Okuno, R. Johns, and K. Sepehrnoori. Three-phase flash in compositional simulation using a reduced method. *SPE Journal*, 15(3):689–703, 2010. doi: 10.2118/125226-PA.

[36] F. M. Orr. *Theory of Gas Injection Processes*, volume 5. Tie-Line Publications, Copenhagen, 2007.

[37] H. Pan and A. Firoozabadi. Fast and robust algorithm for compositional modeling: part II – two-phase flash computations. In *SPE Annual Technical Conference and Exhibition, 30 September–3 October, New Orleans, Louisiana*. Society of Petroleum Engineers, 2001. doi: 10.2118/71603-MS.

[38] S. R. Panuganti, F. M. Vargas, D. L. Gonzalez, A. S. Kurup, and W. G. Chapman. PC-SAFT characterization of crude oils and modeling of asphaltene phase behavior. *Fuel*, 93:658–669, 2012. doi: 10.1016/j.fuel.2011.09.028.

[39] A. Péneloux, E. Rauzy, and R. Fréze. A consistent correction for Redlich–Kwong–Soave volumes. *Fluid Phase Equilibria*, 8(1):7–23, 1982. doi: 10.1016/0378-3812(82)80002-2.

[40] D.-Y. Peng and D. B. Robinson. A new two-constant equation of state. *Industrial & Engineering Chemistry Fundamentals*, 15(1):59–64, 1976. doi: 10.1021/i160057a011.

[41] M. Petitfrere and D. V. Nichita. Robust and efficient trust-region based stability analysis and multiphase flash calculations. *Fluid Phase Equilibria*, 362:51–68, 2014. doi: 10.1016/j.fluid.2013.08.039.

[42] H. H. Rachford Jr. and J. D. Rice. Procedure for use of electronic digital computers in calculating flash vaporization hydrocarbon equilibrium. *Journal of Petroleum Technology*, 4(10):19, 1952. doi: 10.2118/952327-G.

[43] O. Redlich and J. N. S. Kwong. On the thermodynamics of solutions. V. An equation of state. Fugacities of gaseous solutions. *Chemical Reviews*, 44(1):233–244, 1949. doi: 10.1021/cr60137a013.

[44] Schlumberger. *ECLIPSE Reservoir Simulation Software: Technical Description*. Schlumberger, 2014.1 edition, 2014.
[45] M. Sherafati and K. Jessen. Stability analysis for multicomponent mixtures including capillary pressure. *Fluid Phase Equilibria*, 433:56–66, 2017. doi: 10.1016/j.fluid.2016.11.013.
[46] G. Soave. Equilibrium constants from a modified Redlich-Kwong equation of state. *Chemical Engineering Science*, 27(6):1197–1203, 1972. doi: 10.1016/0009-2509(72)80096-4.
[47] J. A. Trangenstein. Minimization of Gibbs free energy in compositional reservoir simulation. In *SPE Reservoir Simulation Symposium*, 10–13 February, Dallas, TX. Society of Petroleum Engineers, 1985. doi: 10.2118/13520-MS.
[48] D. V. Voskov and H. A. Tchelepi. Comparison of nonlinear formulations for two-phase multi-component EoS based simulation. *Journal of Petroleum Science and Engineering*, 82:101–111, 2012. doi: 10.1016/j.petrol.2011.10.012.
[49] D. V. Voskov and H. A. Tchelepi. Compositional space parameterization: multicontact miscible displacements and extension to multiple phases. *SPE Journal*, 14(3):441–449, 2009. doi: 10.2118/113492-PA.
[50] D. V. Voskov and H. A. Tchelepi. Compositional space parameterization: theory and application for immiscible displacements. *SPE Journal*, 14(3):431–440, 2009. doi: 10.2118/106029-PA.
[51] C. H. Whitson and M. R. Brulé. *Phase Behavior*. Henry L. Doherty Memorial Fund of AIME, Society of Petroleum Engineers, Richardson, TX, 2000.
[52] C. H. Whitson and M. L. Michelsen. The negative flash. *Fluid Phase Equilibria*, 53:51–71, 1989. doi: 10.1016/0378-3812(89)80072-X.
[53] G. M. Wilson. A modified Redlich–Kwong equation of state, application to general physical data calculations. In *65th National AIChE Meeting*, Cleveland, OH, pp. 15, 1969.
[54] L. C. Young and R. E. Stephenson. A generalized compositional approach for reservoir simulation. *Society of Petroleum Engineers Journal*, 23(5):727–742, 1983. doi: 10.2118/10516-PA.
[55] R. Zaydullin, D. V. Voskov, S. C. James, H. Henley, and A. Lucia. Fully compositional and thermal reservoir simulation. *Computers & Chemical Engineering*, 63:51–65, 2014. doi: 10.1016/j.compchemeng.2013.12.008.
[56] R. Zaydullin, D. V. Voskov, and H. A. Tchelepi. Comparison of EoS-based and K-values-based methods for three-phase thermal simulation. *Transport in Porous Media*, 116(2):663–686, 2017. doi: 10.1007/s11242-016-0795-7.
[57] D. Zudkevitch and J. Joffe. Correlation and prediction of vapor-liquid equilibria with the Redlich–Kwong equation of state. *AIChE Journal*, 16(1):112–119, 1970. doi: 10.1002/aic.690160122.