4

Multiscale Pressure Solvers for Stratigraphic and Polytopal Grids

KNUT-ANDREAS LIE AND OLAV MØYNER

Abstract

The multiscale restriction-smoothed basis (MsRSB) method is the current state-of-the-art within multiscale methods. MsRSB is very robust and versatile and can be used either as an approximate coarse-scale solver having mass-conservative subscale resolution or as an iterative fine-scale solver that will provide mass-conservative solutions for any given tolerance. The performance of the method has been demonstrated on incompressible two-phase flow, on compressible two- and three-phase black-oil models, as well as on compositional models. It has also been demonstrated that the method can utilize combinations of multiple prolongation operators; e.g., corresponding to coarse grids with different resolutions, adapting to geological features, adapting to wells, or moving displacement fronts. This chapter explains the basic ideas of the MsRSB method, including methods to construct coarse partitions, prolongation, and restriction operators; reduction of the fine-scale flow equations to a coarse-scale system; and formulation as part of a two-level iterative solver. We outline the key functions in the module and show various examples of how the method can be used as an iterative solver for incompressible and compressible flow on 2D rectilinear grids, unstructured grids, and 3D stratigraphic grids.

4.1 Introduction and Background Discussion

Key elements of what today constitute the MATLAB Reservoir Simulation Toolbox (MRST) were originally developed as a flexible research and prototyping platform for multiscale methods and consistent discretizations on complex grids [26]. Research on multiscale methods and other methods aimed at accelerating reservoir simulation has continued to fuel and direct the development of the software since then. After more than 15 years of research, state-of-the-art in multiscale methods

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

has reached a stage where these methods have been commercially implemented and can be used to accelerate the simulation of large and complex geomodels [27]. The purpose of this chapter is to give a basic introduction to the multiscale finite-volume (MsFV) solvers implemented in the `msrsb` module of MRST and demonstrate how you can use these to solve incompressible and compressible flow equations, posed on Cartesian, stratigraphic, and complex polytopal grids. Specifically, we will describe the original MsFV method [18, 32, 43, 56, 59], which is based on primal–dual grid partitions, and the more flexible multiscale restriction-smoothed basis (MsRSB) method [28, 44, 45, 47]. Both methods are also implemented[1] in the commercial INTERSECT simulator [23, 24, 27]. The commercial implementation is more specialized and has been thoroughly optimized for computational speed, whereas the implementations in MRST are primarily designed to enable simple experimentation with new ideas and suffer from certain computational overhead you would not see with a compiled language. Please keep this in mind if the multiscale solvers in MRST do not give you the computational speedup you expect to see.

MRST also implements multiscale mixed finite-element methods for stratigraphic and unstructured grids [1, 48] in the `msmfem` module. These methods are robust and accurate alternatives to upscaling for incompressible flow problems. The solvers are versatile and robust with respect to various kinds of polytopal grids but proved difficult to extend to compressible flow, mainly because of the inherent requirement that flow equations must be on mixed form. The module has therefore not seen any significant development since 2012. In addition, the `msfvm` module offers an early and obsolete implementation of the MsFV method, which we have continued to release to respect the principle of reproducible research.

4.1.1 Why Do We Need Multiscale Methods?

Multiphase flow in porous rock formations is governed by physical processes that take place on a wide range of spatial/temporal scales and is therefore said to be a *multiscale problem*. To explain the fluid movement inside a petroleum reservoir, which typically spans hundreds or thousands of meters in the areal direction, one must understand the fluid movement that takes place inside individual pores and pore throats on the size of micrometers and characterize the petrophysical properties of the porous rock all the way up to the kilometer scale. Unfortunately, flow phenomena taking place on a microscale are not well separated from those on a

[1] In fact, the MsRSB method was first developed in MRST and later reimplemented in INTERSECT. Likewise, key parts of the strategy for generating primal–dual partitions in the MsFV method were first developed in MRST.

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

macro scale, and this makes modeling quite challenging. The MRST textbook [25] gives a thorough introduction to flow modeling, seen from a macroscale perspective, as used to, e.g., describe the fluid behavior of hydrocarbon reservoirs and large aquifer systems. A key modeling assumption is the existence of representative elementary volumes to justify the continuum hypothesis and the Darcy effective-property type description used in most parts of MRST. (See Blunt [5] for more details on flow modeling from a pore-scale perspective.)

The multiscale nature of macroscale models primarily comes from heterogeneity in the petrophysical characteristics of the porous rock. Properties can exhibit abrupt orders-of-magnitude variations across short distances between different strata and at the same time be strongly correlated over hundreds or thousands of meters inside the same strata. Accounting for all pertinent variations is virtually impossible and, as explained in chapter 2 of the MRST textbook [25], reservoir characterization therefore usually involves a hierarchy of models that each covers a limited range of physical scales. Models used for flow studies include core-scale models (centimeter scale) and bed models (meter scale) that are used to give input to the geological characterization and derive effective flow parameters in sector models (tens to hundreds of meters) and field models (kilometer scale). Resulting high-resolution geological models have grid cells in the range of centimeters to decimeters in the vertical direction and meters to tens of meters in the horizontal direction to accurately account for structural elements like faults, fractures, joints, and deformation bands and stratigraphic characteristics like channels, clinoforms, lobes, shale and mud drapes, etc.

The last decades have seen a tremendous increase in computational power, which is equally matched by improvement in computational methods (massive parallelization, multilevel iterative solvers, preconditioning methods, etc.). Simulating high-resolution geomodels is nonetheless computationally costly and, as explained in [25], it is common to use a combination of grid coarsening (discussed in chapter 14) and upscaling (discussed in chapter 15) to derive more homogenized and coarser models that are more computationally tractable. This process is complicated by the lack of scale separation, which essentially implies that any upscaling procedure based on local considerations is only valid for simplified or idealized setups and that nonlocal information is generally needed to compute representative, effective properties that can accurately reproduce the correct flow patterns.

The first so-called multiscale methods were presented two decades ago [3, 4, 8, 17] as a more robust alternative to conventional upscaling. The original idea was to use special basis functions, computed numerically by solving localized flow problems, to incorporate unresolved subscale effects into macroscale models in a systematic and rigorous way that is consistent with the underlying flow equations. These basis functions also gave an immediate mechanism for reconstructing

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

approximate solutions on the fine or any intermediate scale. Over the years, research focus has shifted toward accelerating the solution of the original fine-scale problem, giving iterative multiscale methods [34, 56] that are contenders to algebraic multilevel methods [12, 52, 54].

4.1.2 Basic Flow Model and Abstract Notation

Multiscale methods are essentially designed to solve second-order elliptic equations with strongly heterogeneous coefficients. To explain the key ideas in more detail, it is therefore sufficient to consider the standard model for single-phase flow in the absence of gravity:

$$\nabla \cdot \big(\mathbf{K}(\vec{x})\nabla p\big) = q, \quad \vec{x} \in \Omega. \tag{4.1}$$

Here, p denotes the fluid pressure, q contains volumetric source terms, and $\mathbf{K}$ is the permeability tensor, whose spatial variations introduce the multiscale nature into the problem. In abstract notation, we seek an unknown function p in a solution space $\mathbb{U}$ defined over Ω that for given $\mathbf{K}$ and q satisfy the equation $\mathcal{L}(p;\mathbf{K}) = q$, where $\mathcal{L}(\cdot;\mathbf{K}) = \nabla \cdot \mathbf{K}\nabla$ is the second-order (Poisson) differentiation operator.

The conventional way of representing permeability is as a set of cell-centered values $\mathbf{K}^h = \{\mathbf{K}_i\}_{i=1}^n$ defined over a volumetric mesh $\mathcal{M}^h = \{\Omega_i^h\}_{i=1}^n$. To discretize (4.1) over this mesh, we use the standard two-point flux approximation (TPFA) scheme, which gives a discrete equation in each cell

$$\sum_k T_{ik}^h(p_i - p_k) = q_i, \qquad i = 1, \ldots, n. \tag{4.2}$$

Here, the *transmissibility* T_{ik}^h is a discrete measure of the conductivity between two neighboring cells i and k and involves the discrete permeability values and geometric cell quantities. (Subsection 4.4.1 of the MRST textbook [25] gives more details.) Collecting the discrete equations as rows gives the linear system

$$\boldsymbol{A}\boldsymbol{p} = \boldsymbol{q}, \tag{4.3}$$

which we must solve with a direct or an iterative solver to obtain the unknown cell-averaged pressures $\boldsymbol{p} \in \mathbb{R}^n$. Formulated in abstract notation, we thus seek an unknown pressure p^h in the finite-dimensional space $\mathbb{U}^h$ that satisfies the operator equation $\mathcal{L}^h(p^h;\mathbf{K}^h) = q^h$ given by (4.2) inside each cell.

4.1.3 Local Upscaling

To better motivate multiscale methods, we first briefly outline standard upscaling methods. One starts by generating a coarser mesh, $\mathcal{M}^H = \{\Omega_i^H\}_{i=1}^N$, and then seeks a set of homogenized, effective permeabilities $\mathbf{K}^H = \{\mathbf{K}_i^*\}$ that ideally should

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

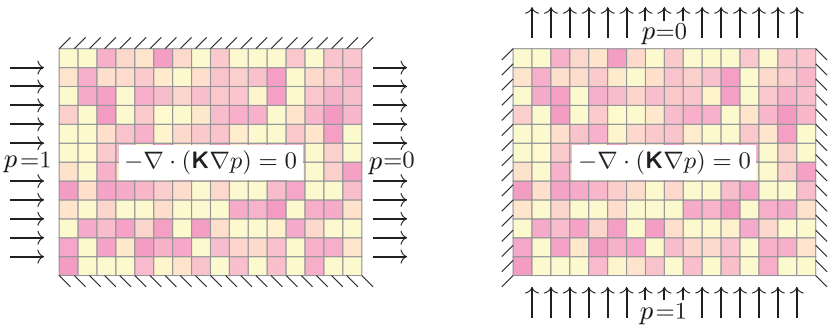


Figure 4.1 Illustration of flow-based upscaling with constant pressures prescribed along the inflow and outflow boundaries and no-flow (or sealing) boundary conditions elsewhere.

reproduce the same total flow through each homogeneous region as you would obtain by solving (4.1) with the full fine-scale permeability $\mathbf{K}^h$.

The simplest way to compute the individual $\mathbf{K}_j^*$ values is through arithmetic, geometric, or harmonic averaging inside each coarse block. This is generally not accurate and will only provide upper and lower bounds on the effective permeabilities. Sharper bounds are obtained through directional combinations of arithmetic and harmonic averages.

Alternatively, we can use a flow-based method in which we first solve local flow problems $\mathcal{L}^h(\varphi;\mathbf{K}^h)=0$, localized to each coarse block Ω_j^H or a somewhat enlarged domain, with suitable boundary conditions. A standard choice is to impose a unit pressure drop in each axial direction, as illustrated in Figure 4.1, which emulates how permeabilities are measured from cores in the laboratory. Using Darcy's law, $\vec{v}=-\mathbf{K}\nabla p$, we can then relate the total outflow in each axial direction to the associated pressure drop over the block and use this to determine the axial components of the effective permeability. Section 15.3.2 of the MRST textbook [25] gives more details and discusses alternative setups of boundary conditions.

We can then use the upscaled permeabilities to solve a coarse-scale flow problem $\mathcal{L}^H(p^H;\mathbf{K}^H)=q^H$ that has significantly fewer unknowns. With a TPFA discretization it is more convenient to use a similar procedure defined over each pair of neighboring grid blocks to upscale the transmissibilities to coarse-scale transmissibilities $\{T_{ij}^H\}$ associated with $\mathcal{M}^H$, as discussed in more detail in section 15.4 of the MRST textbook [25].

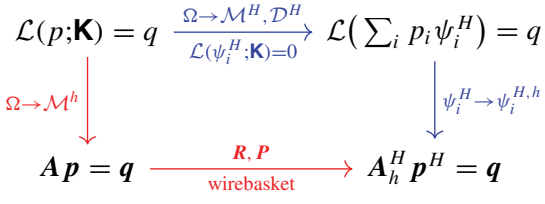
4.2 Multiscale Finite-Volume Methods

Multiscale methods build on similar ideas as flow-based upscaling in the sense that we need to solve a set of representative flow problems that each is localized to

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

a small region. However, instead of using these local flow solutions to compute effective properties, we use them as local building blocks (basis functions) that are combined into an approximate solution by solving a global flow problem on a coarse scale. In other words, we construct a coarse-scale approximation space or, more generally, a set of such coarse-scale spaces that each consists of numerically computed functions that incorporate representative, localized behavior of the true solutions. The resulting reduced coarse-scale problem not only accounts for the effective average flow properties but also incorporates information about the influence of local, subscale variations in pressures and flow directions.

We have already mentioned the MsFV methods implemented in the `msrsb` module. As with geometric and algebraic multigrid, MsFV methods were first formulated in geometric form [18, 19] and later developed in algebraic form [32, 59]. We can use the following diagram to illustrate the differences between the two (details will be explained shortly):



In the geometric formulation (blue arrows), we first partition the domain into a set of primal–dual subdomains and construct a multiscale approximation space consisting of analytical solutions to the elliptic flow model (4.1) localized to these domains. To derive a discrete method, we discretize the local flow problems and use their numerical solutions as basis functions in a global coarse-scale flow problem. In the algebraic formulations (red arrows), we first discretize the global flow problem on the fine scale and then use algebraic operations to formulate discrete prolongation and reduction operators that reduce the fine-scale discretization matrix $\boldsymbol{A}$ to a coarse-scale problem. Geometric formulations are rarely used in practical simulations, but for completeness and pedagogical reasons we describe both forms in detail in the rest of this section.

4.2.1 Geometric Formulation of the Original MsFV Method

As in upscaling, we start by subdividing the domain Ω into a *primal coarse mesh* $\mathcal{M}^H = \{\Omega_i^H\}_{i=1}^N$, where H is a parameter characterizing the size of the grid blocks Ω_i^H. In practice, the coarse blocks are formulated as connected aggregates of cells from the fine mesh $\mathcal{M}^h$, represented in terms of a partition vector $\mathcal{P} \in \mathbb{N}^n$, defined

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

5	5	5	6	6	6
5	5	5	6	6	6
4	4	4	3	3	3
1	4	4	3	3	3
1	1	1	2	2	2
1	1	1	2	2	2

Figure 4.2 Partition of a 6×6 fine mesh $\mathcal{M}^h$ into a coarse mesh $\mathcal{M}^H$ with six blocks. Colors and numbers show the entries of the corresponding partition vector $\mathcal{P}$, which takes six distinct values.

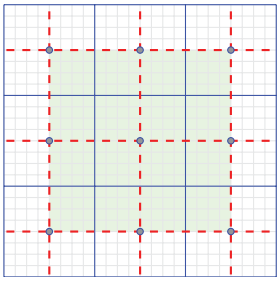


Figure 4.3 Parts of a primal coarse mesh $\mathcal{M}^H$ (blue lines) and the corresponding dual mesh $\mathcal{D}^H$ (red dashed lines). The centroids of the primal mesh are shown as blue circles, and the support region Ψ_i^H of the basis function ψ_i^H associated with the coarse block in the middle is shown in green.

such that $\mathcal{P}(k) = i$ if fine cell k belongs to block i; see Figure 4.2. Chapter 14 of the MRST textbook [25] discusses various techniques for generating such partitions.

Next, we associate a *multiscale basis function* ψ_i^H to each coarse block. These basis functions will be developed as generalizations of the standard first-order Galerkin finite-element basis functions defined on the *dual coarse mesh* $\mathcal{D}^H$. This mesh is defined so that the block centroids of the primal mesh $\mathcal{M}^H$ form vertices in $\mathcal{D}^H$; see Figure 4.3. We then define the basis function ψ_i^H as a generalized hat function by requiring that it equals unity at the centroid of block i and zero at all other block centroids, has compact support within the local domain Ψ_i^H defined as the collection of all dual coarse blocks $\overline{\Omega}_k^H$ that overlap with Ω_i^H, and satisfies a homogeneous flow problem $\mathcal{L}(\psi_i^H; \mathbf{K}) = 0$ inside the support region Ψ_i^H. The localization is enforced by imposing a reduced-dimensional homogeneous flow equation on the edges connecting block centroids, as illustrated in Figure 4.4. The procedure is essentially the same in three spatial dimensions, except that the

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

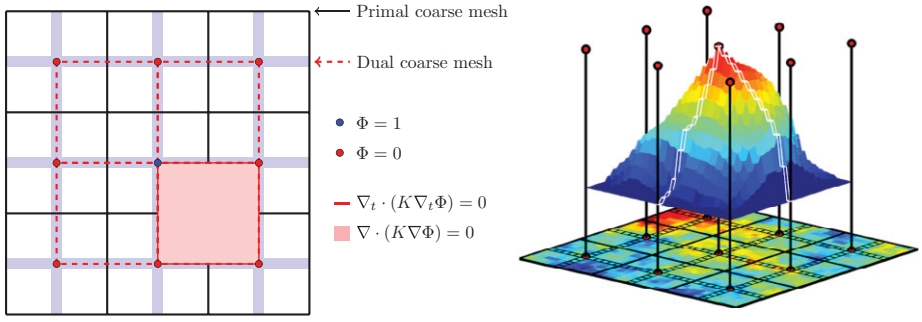


Figure 4.4 Definition of basis functions for the MsFV method. To localize the basis function, we set a unit pressure at the block center and zero pressure at the centers of neighboring blocks and then impose a reduced flow problem as boundary conditions along the dual edges connecting the block centers. The basis function can then be computed by discretizing and solving the flow problems in the region bounded by the dual edges, giving the generalized hat function shown to the right.

localization must be enforced in two steps: First, we extrapolate the constant pressures at the block centroids to the edges of the dual mesh by imposing a reduced-dimensional flow equation in one independent variable. Then, we extrapolate the edge values to the dual block faces by imposing a reduced-dimensional flow equation in two independent variables.

Altogether, the collection of basis functions $\{\psi_i^H\}$ defines a coarse-scale approximation space $\mathbb{U}^H$, in which we can seek an approximate solution $p \approx p^H = \sum_i p_i \psi_i^H$ that satisfies $\mathcal{L}\big(\sum_i p_i \psi_i^H; \mathbf{K}\big) = q$.

To get a fully discrete method, we discretize the local flow problems to compute numerical approximations $\psi_i^{H,h}$ to ψ_i^H. In principle, one could solve the reduced-dimensional flow problems to localize each basis function using lower-dimensional grids, but this becomes quite cumbersome unless all coarse blocks have simple geometrical shapes; in practice, the localization equations are solved on collections of fine-scale cells that trace out the edges and faces of the dual coarse blocks.

Once the numerical basis functions are computed (see the right plot in Figure 4.4), we use them to write the multiscale approximation to the fine-scale solution as the following series:

$$p^h \approx p^{H,h} = \sum_{i=1}^{N} p_i^H \psi_i^{H,h}, \tag{4.4}$$

where the unknown coefficients $p_1^H, \ldots, p_N^H$ must be determined by inserting (4.4) into the discrete fine-scale problem, $\mathcal{L}^h\big(\sum_i p_i^H \psi_i^{H,h}; \mathbf{K}^h\big) = q$.

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

By extending the definition of each basis function to the whole grid we can represent them as vectors $\boldsymbol{b}_i \in \mathbb{R}^n$. If we collect these vectors as columns in a matrix, we get an $n \times N$ so-called prolongation matrix $\boldsymbol{P}_H^h = [\boldsymbol{b}_1, \ldots, \boldsymbol{b}_N]$ and can write (4.4) as an vector equation, $\boldsymbol{p} \approx \boldsymbol{P}_H^h \boldsymbol{p}^H$, where $\boldsymbol{p}^H \in \mathbb{R}^N$. Inserting this approximation into (4.3) gives the linear system $\boldsymbol{A}^h \boldsymbol{P}_H^h \boldsymbol{p}^H = \boldsymbol{q}$, which consists of n equations for N unknowns. To avoid having an overdetermined $n \times N$ system, we simply sum the equations for all fine cells inside each coarse block to obtain an $N \times N$ system. To this end, we introduce the $N \times n$ *restriction* matrix $\boldsymbol{R}_h^H$, in which entry R_{ij} equals 1 if cell j is contained in block i and 0 otherwise. Summing up, we obtain the following reduced coarse-scale system:

$$\left(\boldsymbol{R}_h^H \boldsymbol{A}^h \boldsymbol{P}_H^h\right) \boldsymbol{p}^H = \boldsymbol{R}_h^H \boldsymbol{q} \qquad \Longleftrightarrow \qquad \boldsymbol{A}_h^H \boldsymbol{p}^H = \boldsymbol{q}^H. \tag{4.5}$$

Solving this system gives an approximate coarse-scale pressure $\boldsymbol{p}^H$ but also suggests an approximate fine-scale pressure $\boldsymbol{p}^{H,h} = \boldsymbol{P}_H^h \boldsymbol{p}^H$. We can then use Darcy's law to compute coarse-scale fluxes from $\boldsymbol{p}^H$ and fine-scale fluxes from $\boldsymbol{p}^{H,h}$. The coarse-scale fluxes are mass conservative, because they satisfy the conservation equation inherent in (4.5).

The fine-scale fluxes resulting from $\boldsymbol{p}^{H,h}$ are generally not mass conservative. They are mass conservative *inside* each dual coarse block $\overline{\Omega}_k^H$, where (4.3) holds, but along the edges/faces of the dual blocks, $\partial\overline{\Omega}_k^H$, they only satisfy a reduced-dimensional flow problem and will hence not be conservative. To remedy, we impose the fine-scale fluxes derived from the approximate multiscale solution as Neumann conditions on the boundary of the primal blocks, where these fluxes are mass conservative, because $\partial\Omega_i^H$ is inside $\overline{\Omega}^H$, and solve a local flow problem $\mathcal{L}(\tilde{p}; \mathbf{K}) = 0$ on each Ω_i^H to *reconstruct* fine-scale fluxes that are mass conservative *inside* Ω_i^H and thus also on $\partial\overline{\Omega}_k^H$.

Notice that we could also have used $(\boldsymbol{P}_H^h)^T$ to reduce the overdetermined system. This gives a Galerkin-type discretization [6], which is generally not locally mass conservative.

4.2.2 Algebraic Formulation of the Original MsFV Method

We will now explain how the same construction can be obtained through pure algebraic operations on the fine-scale discretization matrix $\boldsymbol{A}$ from (4.3). To this end, we first introduce a *wirebasket ordering*, in which each cell is categorized as being either a *node*, part of an *edge* of a dual block (or part of a dual *face* in 3D),

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

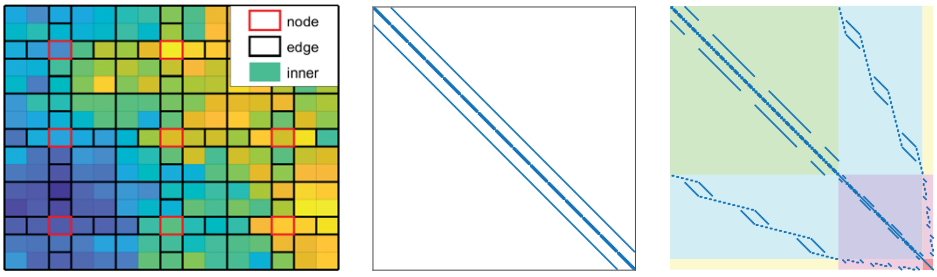


Figure 4.5 Wirebasket ordering of cells for a 15 × 15 fine-scale mesh partitioned into a 3 × 3 coarse mesh (left), sparsity pattern of the fine-scale discretization matrix $\boldsymbol{A}$ (middle), and the same matrix after permutation to wirebasket order (right). (Soure code: `illustrateMsFV.m`.)

or an *inner* cell. Using this ordering, we can permute the linear system (4.3) so that we first have the equations for the inner cells, the edges, and then the nodes. For a 2D system, this reads

$$\boldsymbol{A}\boldsymbol{p} = \boldsymbol{q} \quad \longrightarrow \quad \begin{bmatrix} \boldsymbol{A}_{ii} & \boldsymbol{A}_{ie} & \boldsymbol{0} \\ \boldsymbol{A}_{ei} & \boldsymbol{A}_{ee} & \boldsymbol{A}_{en} \\ \boldsymbol{0} & \boldsymbol{A}_{ne} & \boldsymbol{A}_{nn} \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_i \\ \boldsymbol{p}_e \\ \boldsymbol{p}_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{q}_i \\ \boldsymbol{q}_e \\ \boldsymbol{q}_n \end{bmatrix}. \tag{4.6}$$

Here, matrix block $\boldsymbol{A}_{ie}$ represents the influence that inner-cell pressures have on the edge-cell pressures, etc. The zero blocks arise because node-cell pressures cannot influence inner-cell pressures, and vice versa, if we use a two-point discretization on the fine grid. Figure 4.5 illustrates the wirebasket ordering and the corresponding permutation for a small 2D test problem.

The system (4.6) is symmetric so that cells influence each other in a fully symmetric way. A quick recap of the previous section shows that to compute basis functions, we first impose a unit or zero pressure at the nodes, solve reduced flow problems in the edge and face cells to impose localizing boundary conditions, and finally solve a homogeneous flow problem in the inner nodes. To mirror this approach, we must break the symmetry of (4.6) by removing the influence of inner cells on the edge cells and similarly make node cells unaffected by edge cells ($\boldsymbol{A}_{ei} = \boldsymbol{0}$, $\boldsymbol{A}_{ne} = \boldsymbol{0}$). The resulting matrix system reads

$$\tilde{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{A}_{ii} & \boldsymbol{A}_{ie} & \boldsymbol{0} \\ \boldsymbol{0} & \tilde{\boldsymbol{A}}_{ee} & \boldsymbol{A}_{en} \\ \boldsymbol{0} & \boldsymbol{0} & \tilde{\boldsymbol{A}}_{nn} \end{bmatrix}. \tag{4.7}$$

Here, the new matrix block $\hat{\boldsymbol{A}}_{ee}$ represents how edge cells influence each other and is defined by removing the influence from the inner cells on the edge cells to

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

ensure that no mass-balance errors are introduced by removing internal coupling (and equations). Specifically, we set

$$\tilde{\boldsymbol{A}}_{ee} = \boldsymbol{A}_{ee} + \operatorname{diag}\left(\sum_i \boldsymbol{A}_{ie}^T\right),$$

where the summation is interpreted row-wise and the diag operator works as in MATLAB; i.e., mapping a vector to a diagonal matrix. The other new matrix block $\tilde{\boldsymbol{A}}_{nn}$ represents how node cells influence each other and should be defined as the $\boldsymbol{A}_h^H$ matrix derived in the previous section.

Because $\tilde{\boldsymbol{A}}$ is upper block triangular, we can, without any prior knowledge of the algorithm, use block elimination to find an explicit inverse. However, to make the connection between the geometric and algebraic formulation of MsFV, we will form the part of the inverse that accounts for the influence of source terms associated with nodes $[\mathbf{0}\ \mathbf{0}\ \boldsymbol{q}_n]^T$, where $\boldsymbol{q}_n$ should be set equal to $\boldsymbol{q}^H$. To this end, we solve the following system (where $\boldsymbol{I}_n$ is the $N \times N$ identity matrix):

$$\begin{bmatrix} \boldsymbol{A}_{ii} & \boldsymbol{A}_{ie} & \mathbf{0} \\ \mathbf{0} & \tilde{\boldsymbol{A}}_{ee} & \boldsymbol{A}_{en} \\ \mathbf{0} & \mathbf{0} & \tilde{\boldsymbol{A}}_{nn} \end{bmatrix} \boldsymbol{B} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \boldsymbol{I}_n \end{bmatrix} \quad \longrightarrow \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{A}_{ii}^{-1}\boldsymbol{A}_{ie}\tilde{\boldsymbol{A}}_{ee}^{-1}\boldsymbol{A}_{en} \\ -\tilde{\boldsymbol{A}}_{ee}^{-1}\boldsymbol{A}_{en} \\ \boldsymbol{I}_n \end{bmatrix} \tilde{\boldsymbol{A}}_{nn}^{-1} = \tilde{\boldsymbol{P}}\tilde{\boldsymbol{A}}_{nn}^{-1}.$$

This defines the prolongation matrix $\boldsymbol{P}$ (and the basis functions), which can be obtained by permuting $\tilde{\boldsymbol{P}}$ back to the original order. The action of matrix $\boldsymbol{B}$ can therefore be interpreted as first solving the coarse-scale equation (associated with the node cells) and then interpolating the result onto the remaining cells.

Similarly, we can define an inverse that accounts for the influence of the source terms corresponding to edge and inner cells in the fine-scale system $[\boldsymbol{q}_i\ \tilde{\boldsymbol{q}}_e\ \mathbf{0}]^T$ by inverting the following system:

$$\begin{bmatrix} \boldsymbol{A}_{ii} & \boldsymbol{A}_{ie} & \mathbf{0} \\ \mathbf{0} & \tilde{\boldsymbol{A}}_{ee} & \boldsymbol{A}_{en} \\ \mathbf{0} & \mathbf{0} & \tilde{\boldsymbol{A}}_{nn} \end{bmatrix} \boldsymbol{C} = \begin{bmatrix} \boldsymbol{I}_i & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I}_e & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \quad \longrightarrow \quad \boldsymbol{C} = \begin{bmatrix} \boldsymbol{A}_{ii}^{-1} & \boldsymbol{A}_{ii}^{-1}\boldsymbol{A}_{ie}\tilde{\boldsymbol{A}}_{ee}^{-1} & \mathbf{0} \\ \mathbf{0} & \tilde{\boldsymbol{A}}_{ee}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}.$$

Whereas the basis functions can be thought of as homogeneous solutions to the elliptic operator, the nonzero columns of matrix $\boldsymbol{C}$ are called *correction functions* [29, 30] and can be thought of as inhomogeneous or particulate solutions of the elliptic equation that account for effects not represented in the basis functions such as gravity, compressibility, and source terms from wells and aquifers. We can thus write the whole multiscale solution as

$$\tilde{\boldsymbol{p}} = \tilde{\boldsymbol{P}}(\boldsymbol{A}^H)^{-1}\boldsymbol{q}^H + \boldsymbol{C}\tilde{\boldsymbol{q}}, \tag{4.8}$$

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

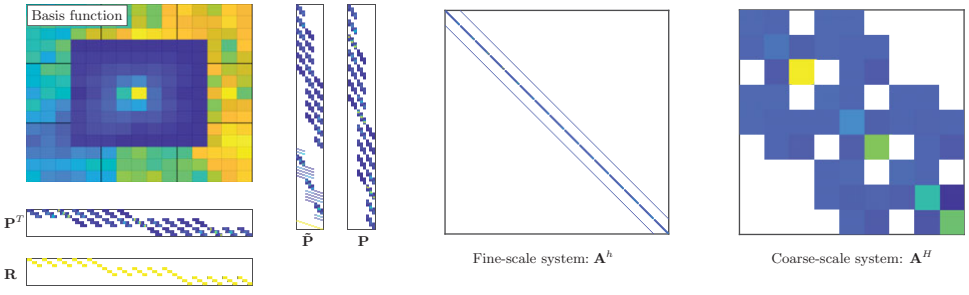


Figure 4.6 Illustration of basis function, prolongation and restriction operators, and fine- and coarse-scale system for the setup in Figure 4.5. (Complete source code: `illustrateMsFV.m`.)

where the tilde means that the vectors appear in wirebasket order. Without correction functions, the method can be interpreted as a special case of classic nonoverlapping domain-decomposition methods; see Nordbotten and Bjørstad [50].

Example 4.1. Figure 4.6 illustrates how this construction looks for the simple setup from Figure 4.5. Here, we reduce the 225×225 fine-scale system to a 9×9 system, which corresponds to an upscaling ratio of 25. Notice that whereas the fine-scale system has a pentagonal structure, the coarse-scale system has more diagonals and thus can be considered a multipoint discretization. The extra multipoint connections are introduced because of overlap among the different basis functions.

4.2.3 Deficiencies and Limitations of the Original MsFV Method

Extensive numerical tests have shown that the MsFV method, as outlined thus far, is accurate and robust compared with local upscaling methods and gives solutions that generally agree well with the corresponding fine-scale solutions. However, exceptions to this rule occur on meshes with high aspect ratios and for channelized media with strong permeability contrasts, for which the method tends to produce solutions with (large) unphysical pressure oscillations and highly circular velocity fields [21, 31]. This deficiency is a result of inadequate localization assumptions, which become particularly evident when extrapolating across large media contrasts. This introduces a pronounced lack of monotonicity unless the coarse-scale stencil is modified locally to be closer to the classic two-point scheme; see [16, 51, 57] for more details. Attempting to remedy this deficiency led to the development of iterative versions of MsFV, which we will discuss in Subsection 4.2.6.

In algebraic operator form, the MsFV method can in principle be applied to any mesh, provided one has a suitable wirebasket ordering of the fine-scale cells. Using MRST, we developed algorithms to construct such orderings for general meshes

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

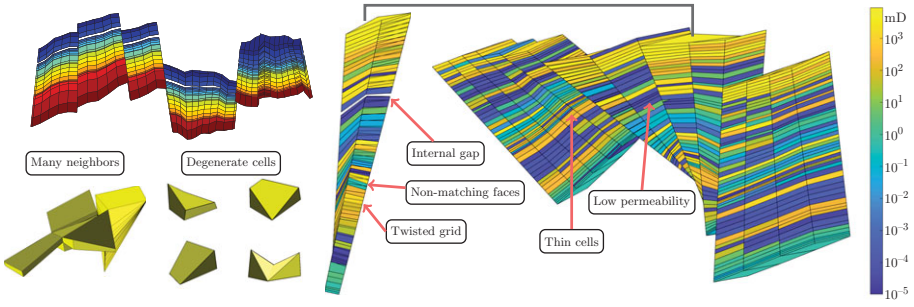


Figure 4.7 Illustration of some of the geometrical and topological challenges encountered in meshes modeling real assets.

[40, 42, 43]. These algorithms work well for rectilinear, curvilinear, triangular, and Voronoi meshes and can also be applied to stratigraphic meshes that do not have too complex geometry and topology. However, meshes encountered in models of real petroleum assets are usually quite complex because of inactive cells, degeneracy in cell geometry, nonmatching cell faces across faults, and nonneighboring connections that are used to model pinch-outs, erosion, faults, etc., as illustrated in Figure 4.7. Developing practical methods that can generate admissible primal–dual partitions for such grids in an automated manner has proved wickedly difficult.

The main challenge is that we need to solve reduced-dimensional problems along the perimeter of the dual blocks to set the boundary conditions for the basis functions. A two-point discretization can only model flow across interfaces between neighboring cells (or, more generally, across nonneighboring connections that have associated transmissibilities). This means that the edge cells in the wirebasket ordering, outlined in black in Figure 4.5, must form a contiguous chain that is connected through cell faces of codimension one; face cells in the 3D ordering should be connected in the same way. Cells that only share a vertex or an edge – i.e., share a connection of codimension two or three, have no associated transmissibility in the fine-scale discretization, and these connections will thus act as internal no-flow boundaries in the reduced flow problems used to provide localization; see Figure 4.8.

Ensuring that edge and face cells are properly connected will inevitably make the edges/faces thicker, and one can easily end up with cluttered partitions in which edge/face cells represent a large fraction of the available cells, in particular for partitions with a low coarsening ratio. In addition, the edges and faces of dual blocks should preferably not contain high-contrast streaks, which are known to introduce numerical instabilities. In our experience, these requirements limit the resolution and type of partitions one can generate and in many cases preclude the use of automated approaches. We have also encountered numerous cases, typically

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

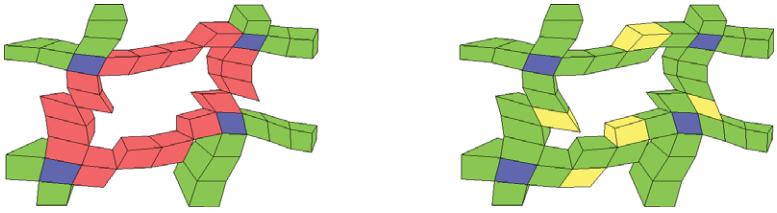


Figure 4.8 Ensuring contiguous connections in a wirebasket ordering. Blue cells are either nodes or edges, green cells are part of a connected edge or face, and the red cells are only connected through interfaces of codimension two that effectively represent internal barriers for the reduced flow problems. The corresponding edges/faces can be made contiguous by adding the yellow cells so that two-point fluxes can be computed across all internal cell faces.

containing complex fault architectures, that we were not able to partition with contemporary partition methods. This motivated the development of an alternative way of computing basis functions based on algebraic smoothing, which we explain in the next subsection.

4.2.4 The Multiscale Restriction-Smoothed Basis Method

If we take a step back and consider how we constructed the matrix $\boldsymbol{A}^H$ to form the coarse system, we see that we only need a pair $(\boldsymbol{P}, \boldsymbol{R})$ of prolongation and restriction operators. We have already discussed suitable choices of $\boldsymbol{R}$, either as a block-wise summation operator or as $\boldsymbol{P}^T$. The essential feature of $\boldsymbol{P}$ is that it maps degrees of freedom associated with the coarse scale to degrees of freedom on the fine scale. Constructing $\boldsymbol{P}$ as a generalized version of standard finite-element hat functions is one of many possible choices. A trivial alternative would be to construct $\boldsymbol{P}$ as the characteristic function of the coarse partition; i.e., from piecewise constant basis functions that equal unity inside the corresponding coarse block and zero outside. We could also let each basis function sample a hat function with finite support covering the associated coarse block. Neither of these choices would satisfy the flow equation on the fine scale but nonetheless lead to admissible coarse-scale systems and provide a crude way of mapping coarse-scale solutions onto the fine mesh that may be sufficient when the method is applied as part of an iterative framework [41], which we will discuss in more detail in the Subsection 4.2.6.

From multigrid theory [54], we generally have that $\boldsymbol{P}$ should:

1. be a partition of unity to represent constant field (in the discrete case, this implies that each row of $\boldsymbol{P}$ has unit row sum, and likewise we require that each column of $\boldsymbol{R}$ has unit column sum);
2. be algebraically smooth in the sense that $\|AP\|_1$ is minimized, so that $\boldsymbol{A}\boldsymbol{P}\vec{p}^H \approx \boldsymbol{A}\boldsymbol{p}$ locally;

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

3. be localized, which in the discrete case means that the basis functions defined by the columns of $\boldsymbol{P}$ should have compact support within a (small) support region that contains the corresponding coarse block.

To achieve this, we could imagine that we initiate each column of $\boldsymbol{P}$ as the characteristic function of the corresponding coarse block and then use a standard algebraic smoother, like Jacobi or Gauss–Seidel, to transform $\boldsymbol{P}$ so that the columns gradually become consistent with the discretized flow equation. Unfortunately, the support of the columns will increase by one cell layer in each iteration, and to ensure compact support, we would have to stop after a few iterations, giving us a prolongation operator that is only partially consistent with $\boldsymbol{A}$. In [45], we proposed a careful localization strategy that enabled us to continue the iterations as long as we want and still obtain a prolongation operator that satisfies all three conditions just stated.

To explain this construction, we first supply each coarse block by a surrounding support region inside which the corresponding basis function will have compact support; see Figure 4.9. Our construction will force the basis function to be zero in a layer of cells surrounding the support region, and to ensure overlap among the basis functions, the support region should therefore extend at least one cell layer outside of the coarse block. If a primal–dual partition is available, each support region can be set as the union of dual blocks that overlap each coarse block. In [45], we presented an alternative geometric construction based on triangulation of coarse-block centers. Alternatively, one could form the support region by growing the coarse blocks outward.

Starting with $\boldsymbol{P}$ as the $n \times m$ characteristic function of the coarse partition and assuming that we already have obtained ν iterates, we first compute the next Jacobi increment for each column j defined as

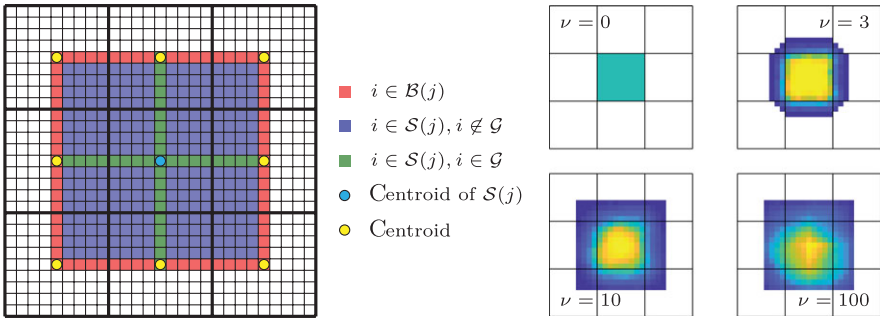


Figure 4.9 Construction of basis functions for the MsRSB method. The left plot shows the support region and boundary cells. The right plots show iterations. (Source code: `illustrateMsRSB.m`.)

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

$$\boldsymbol{d}_j = \boldsymbol{\Lambda}\boldsymbol{A}\boldsymbol{P}_j^v, \qquad \boldsymbol{\Lambda} = \mathrm{diag}([1/a_{11}, \ldots, 1/a_{nn}]). \tag{4.9}$$

To ensure that the support of the basis function does not extend outside its support region (defined as all cells $i \in \mathcal{S}(j)$), we reset $\boldsymbol{d}_j$ to zero in all of the support boundary cells $\mathcal{B}(j)$ of $\mathcal{S}(j)$, shown in red in Figure 4.9. This will effectively remove mass from the iteration, and to ensure that the basis functions represent a partition of unity, we must sum the mass loss from different basis functions and add it back in. To this end, let $\mathcal{G} = \bigcup_{j=1}^{m} \mathcal{B}(j)$ denote the set of all cells that belong to the support boundary of a basis function and define $\mathcal{H}(i)$ as the set of indices of the support regions the cell i belongs to; i.e., $\mathcal{H}(i) = \{j | i \in \mathcal{S}(j), i \in \mathcal{G}\}$. Then, the updated to $\mathcal{P}_j$ reads

$$\boldsymbol{P}_j^{v+1} = \boldsymbol{P}_j^v - \omega\,\hat{\boldsymbol{d}}_j, \qquad \hat{d}_{ij} = \begin{cases} d_{ij}, & i \in \mathcal{S}(j), i \notin \mathcal{G} \\ \frac{1 - P_{ij}^v \sum_{k \in \mathcal{H}(i)} d_{jk}}{1 + \sum_{k \in \mathcal{H}(i)} d_{jk}}, & i \in \mathcal{S}(j), i \in \mathcal{G}, \\ 0, & i \notin \mathcal{S}(j). \end{cases}$$

Here, ω is a relaxation parameter, typically chosen as $2/3$. How to define a similar iteration based on Gauss–Seidel is discussed in [20].

4.2.5 Introduction to the MRST Implementation

The `msrsb` module of MRST implements the algebraic multiscale methods explained in Subsections 4.2.2 and 4.2.4. Next, we describe the most basic routines and demonstrate how to solve a simple incompressible flow problem in 2D. The multiscale methods are implemented as solvers and not simulators, and to use them you must first set up a suitable problem description and construct a fine-scale linear system, as explained in chapters 5 and 10 of the MRST textbook [25]. You will also find full details in the source codes that accompany each example. The basic multiscale solvers also assume that you have partitioned the mesh into coarse blocks and created an instance of MRST's coarse-grid structure; see chapter 14 of the MRST textbook [25].

The basic multiscale solver for incompressible flow is implemented with an interface that resembles the incompressible solvers in the `incomp`, `mimetic`, and `mpfa` modules in the sense that it consists of two parts:

```
[basis, CG]     = getMultiscaleBasis(CG, A, varargin)
[state, report] = incompMultiscale(state, CG, hT, fluid, basis, varargin)
```

The first function takes two arguments, a coarse grid `CG` and a fine-scale linear system `A`, and constructs a pair of restriction and prolongation operators, returned

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

in the `basis` structure. Likewise, the function stores interaction regions to `CG`. For the prolongation operator, `basis.P`, the routine supports both MsFV- and MsRSB-type basis functions, which you can select by specifying the optional parameter `'type'` (default value is `'msrsb'`):

```
[basis, CG]  = getMultiscaleBasis(CG, A, 'type', 'msfv')
```

The restriction operator, `basis.R`, is by default set to be a control-volume summation operator, but you can also get a Galerkin-type restriction ($\boldsymbol{R} = \boldsymbol{P}^T$) by setting the optional parameter `'useControlVolume'` to `false`. In addition, there are optional parameters that let you control the convergence tolerance and the maximum number of iterations used to construct MsRSB bases. By default, the routine regularizes `A` before computing basis functions by ensuring that the row and column sum of the matrix is zero, but this is possible to disable.

The `incompMultiscale` function implements the main solver interface for incompressible problems. The function is a wrapper around `incompTPFA` from the `incomp` module and solves the problem with MsRSB (or MsFV), producing both an approximate fine-scale pressure and a reconstructed, divergence-free velocity field. The first four input parameters are almost the same as for the other incompressible solvers in MRST: reservoir state, *coarse* grid, transmissibilities for the fine-scale problem, and a fluid object. These are passed on to the `incompTPFA` solver to construct a new fine-scale system based on the current reservoir state. In addition, the solver accepts several optional parameters; some of these will be discussed later.

The solver is split across two functions so that we can solve dynamic *multiphase* flow problems more efficiently. The key idea is to first construct basis functions from a static flow equation $\nabla\cdot(\mathbf{K}^0\nabla p) = q$ and then *reuse* these to compute approximate pressure updates for a multiphase equation $\nabla \cdot (\lambda(S)\mathbf{K}\nabla p) = q$, in which $\lambda(S)$ changes with time. This works well as long as the heterogeneity of $\lambda(S)\mathbf{K}$ is similar to $\mathbf{K}^0$ inside each coarse block. (If not, basis functions must be updated locally.) Because the global pressure is solved using many fewer unknowns, we can potentially save a lot of computations at the expense of only obtaining an approximate fine-scale solution; see, e.g., [21] for a more comprehensive discussion. This possibility was one of the main initial motivations behind multiscale methods.

> **Example 4.2.** As an example of a strongly heterogeneous problem, we consider single-phase flow within a rectangular domain represented on a 20×20 Cartesian mesh with permeability sampled from Model 2 of the SPE10 upscaling benchmark [9], driven by a pressure drop of 50 bar from the west boundary to the east boundary. What we present in the following is the `shortMultiscaleIntro.m` script from the `msrsb` module, in which we have edited and condensed some key statements for

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

brevity. In particular, we omit all statements necessary to set up the fine-scale flow problem; i.e., define the grid `G` and petrophysics, half-transmissibilities `hT`, the fluid description `fluid`, boundary conditions `bc`, and initial condition `state0`.

To set up and use the multiscale solver, we start by creating a rectangular partition and a coarse-grid structure augmented with support regions:

```
CG = coarsenGeometry(generateCoarseGrid(G, partitionUI(G, [5, 4])));
CG = storeInteractionRegion(CG);
```

Next, we can use the `incompTPFA` function to assemble the fine-scale linear system and return the result without solving it:

```
state = incompTPFA(state0, G, hT, fluid, 'bc', bc, ...
                   'MatrixOutput', true, ...
                   'LinSolve', @(A, x) zeros(size(x, 1), 1));
```

Then, we use the resulting fine-scale matrix `state.A` to construct basis functions, form and solve the multiscale system, and prolongate the result back to the fine scale to obtain an approximate solution:

```
basis  = getMultiscaleBasis(CG, state.A);
Am     = basis.R * state.A * basis.B;
bm     = basis.R * state.rhs;

p_ms   = Am\bm;
p_prol = basis.B * p_ms;
```

The structure of the fine-scale matrix, the prolongation and restriction operators, and the coarse-scale matrix are essentially the same as shown for a slightly smaller problem in Figure 4.4 and are thus not reported for brevity.

Figure 4.10 compares the multiscale solution on the fine and coarse meshes with the fine-scale solution and a corresponding upscaled solution computed using the local flow-based upscaling discussed in Subsection 4.1.3. On the coarse scale, the multiscale approximation is significantly more accurate than the upscaled solution and using the basis functions to project onto the fine scale reveals more of the structure of the true solution, even though not all details are fully correct. The multiscale solution has two sources of error: the approximations made to ensure that the basis functions have local support and the error in the coarse-scale system. In this case, however, the relative error is less than 9%, which can be acceptable for some purposes.

A fine-scale pressure approximation is not our primary interest when solving incompressible, multiphase flow problems. Instead, we need to have fine-scale fluxes, which will be used as input to the transport equation,

$$\partial_t(\phi S) + \nabla \cdot (f(S)\vec{v}) = q.$$

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

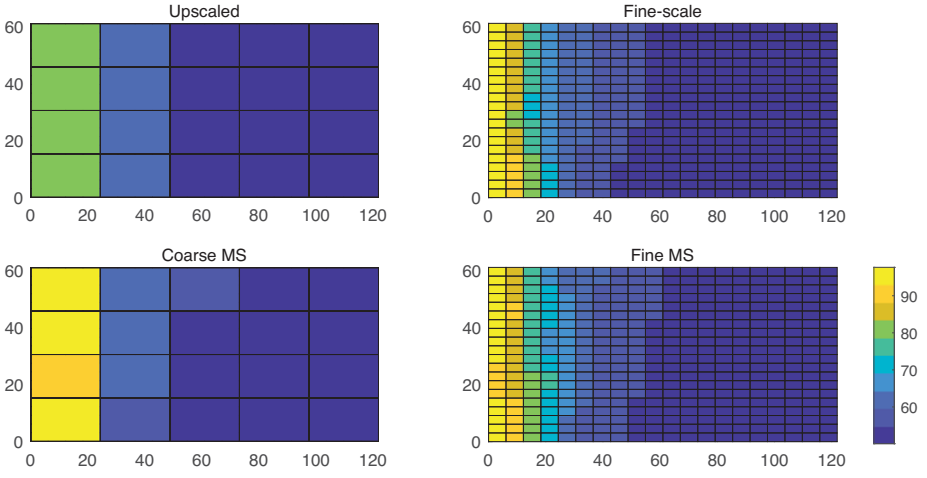


Figure 4.10 Multiscale and upscaled approximations compared to the fine-scale solution for a 20×20 subsample of Layer 15 from SPE 10 with a pressure drop from 100 bar (left) to 50 bar (right).

For incompressible flow, the velocity field is divergence free away from source terms. As pointed out on p. 105, the prolongated pressure solution is not mass conservative and will not generally be divergence free. Instead, we need to solve a local flow problem inside each coarse block to reconstruct a more correct flux approximation that is suitable for use with a transport solver. To this end, we use the full solver to compute the multiscale solutions with and without flux reconstruction:

```
rec  = incompMultiscale(state0, CG, hT, fluid, basis, 'bc', bc, ...
                        'reconstruct', true);
prol = incompMultiscale(state0, CG, hT, fluid, basis, 'bc', bc, ...
                        'reconstruct', false);
```

We can then use the divergence operator from the `ad-core` module to verify the divergence properties of the corresponding fluxes:

```
op = setupOperatorsTPFA(G, rock);
Div = @(flux) op.Div(flux(op.internalConn));
[df, drec, dprol] = deal( Div(state.flux), Div(rec.flux), Div(prol.flux) );
```

Figure 4.11 shows that the discrete divergence of the fine-scale flux is only nonzero near the boundary where we have imposed boundary conditions that act like source and sink terms. The same is true for the reconstructed flux, whereas if we compute fluxes directly from the prolongated pressure using Darcy's law, the resulting flux field has nonzero divergence in almost all cells. This means that this flux field is not

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

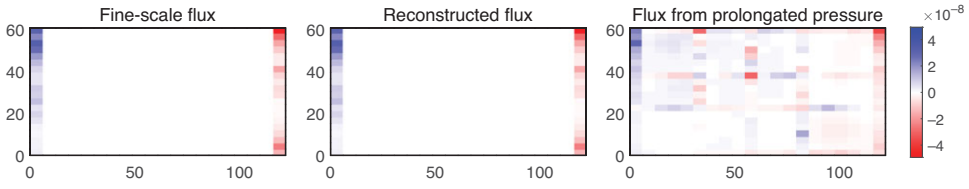


Figure 4.11 Cell-wise discrete divergence for the fine-scale flux, the reconstructed flux, and the flux computed from the prolongated pressure solution.

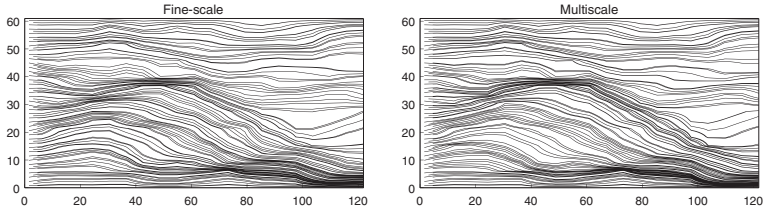


Figure 4.12 Streamlines traced for the fine-scale flux and the reconstructed multiscale approximation.

suitable for use with a transport solver. The reconstructed flux field, on the other hand, reproduces the transport properties of the fine-scale solution with good accuracy, as can be seen from the streamline plots in Figure 4.12.

For completeness, let us also compute an approximate solution using MsFV basis functions. We start by forming a wirebasket ordering:

```
wb      = partitionUIdual(CG,[5 4]);
CG.dual = makeExplicitDual(CG, wb);
```

which here will define 30 dual coarse blocks. The second line extends `CG` with an extra field `CG.dual` that contains the global wirebasket ordering as well as a local categorization of all cells that make up each dual coarse block. We can now construct basis functions and use these to compute an approximation solution:

```
basis = getMultiscaleBasis(CG, A, 'type', 'msfv');
state = incompMultiscale(state0, CG, hT, fluid, basis, 'bc', bc);
```

Figure 4.13 compares the error in the MsRSB and MsFV approximations relative to the pressure computed by solving the original fine-scale system. The error for MsFV is significantly larger.

For the interested reader, we show the essential code that `getMultiscaleBasis` uses to compute the MsRSB basis functions. If we drop processing of input arguments and checks for convergence, the key lines read (see `illustrateMsRSB.m`):

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

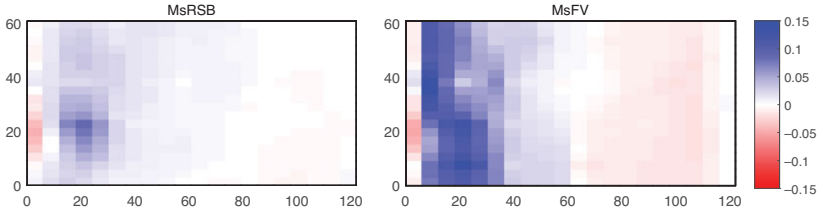


Figure 4.13 Relative error in the fine-scale pressure approximations computed by the MsRSB and the MsFV methods.

```
P = zeros(n, m);
for i=1:m, P(p==i,i)=1; end
J = spdiags(1./diag(A), 0, n, n)*A;
for j=1:itMax
   incr = J*P.*M;
   P    = P - 2/3*incr;
   P    = bsxfun(@rdivide, P, sum(P, 2));
end
```

$$M_{ij} = \begin{cases} 1, & \text{if } j \in \mathcal{H}(i), \\ 0, & \text{otherwise} \end{cases}$$

Here, `p` holds the coarse partition and has values $1, \ldots, m$, whereas the $n \times m$ indicator matrix `M` signifies whether the cell corresponding to row i belongs to the support region of the block corresponding to column j. The matrix is constructed from a cell array containing the indices of the cells belonging to each support region:

```
lens   = cellfun(@numel, CG.cells.interaction);
blocks = rldecode((1:CG.cells.num)', lens);
ia = vertcat(CG.cells.interaction{:});
M  = sparse(ia, blocks, ones(size(ia)), CG.parent.cells.num, CG.cells.num);
```

4.2.6 Iterative Formulation

The multiscale method, as discussed thus far, can compute mass-conservative, fine-scale approximations and reproduce qualitatively correct flow patterns but has no guarantee on the approximation error or the size of the fine-scale residual. The method can have significant local errors (as seen in Figure 4.13), and these errors can be particularly large at the interfaces between coarse-grid blocks as a result of inaccurate boundary conditions that do not necessarily represent the influence from global flow patterns in a good way. To diminish errors and provide a systematic means of controlling the overall multiscale approximation, Hajibeygi et al. [13–15] proposed to use line relaxation to smooth high-frequency error components and propagate fine-scale residuals from the interfaces and into the coarse blocks, where

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

they can be efficiently reduced by reapplying the multiscale solver with improved localization from the smoothed solution. This approach is tied to rectilinear or curvilinear meshes and cannot easily be generalized to other mesh types.

Fortunately, the algebraic formulation of the multiscale method can easily be applied as part of a suitable iterative algebraic framework:

- Modified Richardson iteration is similar to Jacobi and Gauss–Seidel:

$$\boldsymbol{p}^{\nu+1} = \boldsymbol{p}^{\nu} + \omega^{\nu} \boldsymbol{P}(\boldsymbol{A}_h^H)^{-1}\boldsymbol{R}(\boldsymbol{q} - \boldsymbol{A}\boldsymbol{p}^{\nu}). \tag{4.10}$$

  Here, ω is a relaxation parameter that has to be chosen so that $\{\boldsymbol{p}^{\nu}\}$ converges.
- In a two-stage iterative method [56, 60], the Richardson iteration is preconditioned with a local smoother $\boldsymbol{S}$ such as additive Schwarz or block incomplete lower–upper (ILU) factorization. The resulting method reads (with $\boldsymbol{p}^1 = \boldsymbol{P}(\boldsymbol{A}_h^H)^{-1}\boldsymbol{R}\boldsymbol{q}$)

$$\begin{aligned} \boldsymbol{p}^{*} &= \boldsymbol{p}^{\nu} + \boldsymbol{S}(\boldsymbol{q} - \boldsymbol{A}\boldsymbol{p}^{\nu}), \\ \boldsymbol{p}^{\nu+1} &= \boldsymbol{p}^{*} + \boldsymbol{P}(\boldsymbol{A}_h^H)^{-1}\boldsymbol{R}(\boldsymbol{q} - \boldsymbol{A}\boldsymbol{p}^{*}). \end{aligned} \tag{4.11}$$

- One can also use one iteration of the two-stage method as a preconditioner in a Krylov subspace method like GMRES, as first suggested by [33].

All approaches give a systematic means for reducing the fine-scale residual toward machine precision. Effectively, this means that multiscale methods can be used in three different modes: (i) as a linear solver for the fine-scale system; (ii) as an approximate solver that only reduces the fine-scale residual below a prescribed, relaxed tolerance and still guarantees a mass-conservative approximation; or (iii) as a one-step coarse-scale alternative to upscaling that gives mass-conservative fluxes on the fine and any intermediate meshes.

Example 4.3. We continue studying the setup from Example 4.2 (source code is given in `shortIterativeIntro.m`). To motivate the iterative method, we start by plotting the residual of the fine-scale multiscale approximation to the upper left in Figure 4.14. We observe that the residual is highly localized around the support of each basis function, forming patches that trace the contours of $\mathcal{G}$. In other words, large residual errors are primarily caused by the localization imposed on the basis functions to ensure that the coarse-scale system is sparse in nature. The smoother will in most cases consist of the application of a matrix that has approximately the same sparsity as the fine-scale discretization and hence only modifies the solution locally. Here, we observe that its main effect is to significantly reduce the high residuals on $\mathcal{G}$.

The interplay between local smoothers and coarse-grid corrections is widely studied for multilevel solvers. The central idea is to use the inexpensive smoother to remove local errors and let the coarse-scale solver remove low-frequency errors and account for the global forcing of drive mechanisms like boundary conditions, source

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

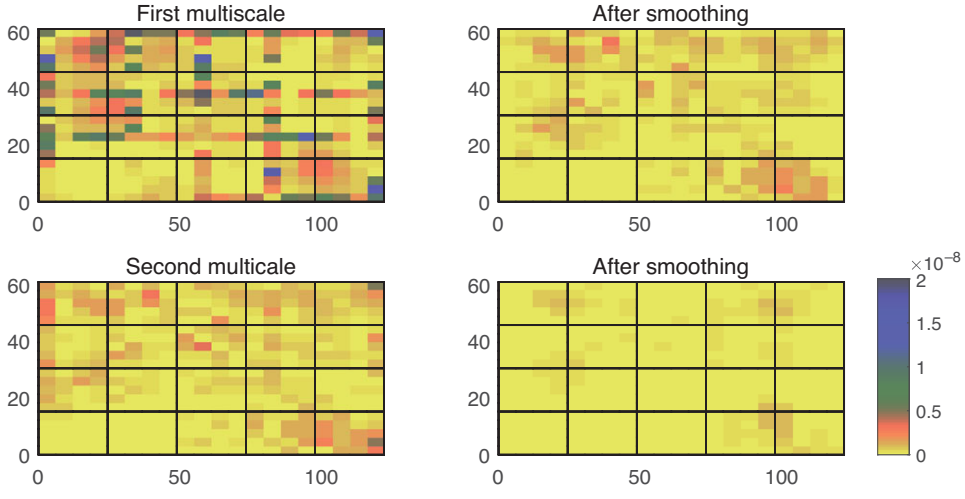


Figure 4.14 Cell-wise absolute residual value after the first four substeps in a multiscale iteration: initial multiscale solution, after local preconditioner (smoother) in first iteration step, after global corrector (multiscale solver) in first iteration step, and after smoother in second iteration step.

terms, and wells. Together, the two solvers can fairly efficiently remove error modes from elliptic or near-elliptic problems, as we can observe in going from upper left to lower right in Figure 4.14.

Let us look at how this is implemented, starting with the construction of the smoother, which is done in two separate function calls:

```
fn = getSmootherFunction('type', 'ilu0');
S  = fn(A, b);
```

```
fn =
  function_handle with value:
    @(A,b)getILU0(A,b,opt.iterations)

S =
  function_handle with value:
    @(d)U\(L\d)
```

The first call selects a single cycle of ILU(0) as our smoother and returns an interface to an anonymous function `@(A,b)`. When evaluated with our specific system `A` and `b` as input parameters, which usually is performed in a setup phase inside the solver but here is done explicitly for pedagogical reasons, this function performs a partial factorization of `A` and returns an approximate inverse of `A`. With ILU(0), the partial factorization is constructed so that the sum of the lower- and upper-triangular matrices `L` and `U` does not have more nonzero elements than `A`.

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

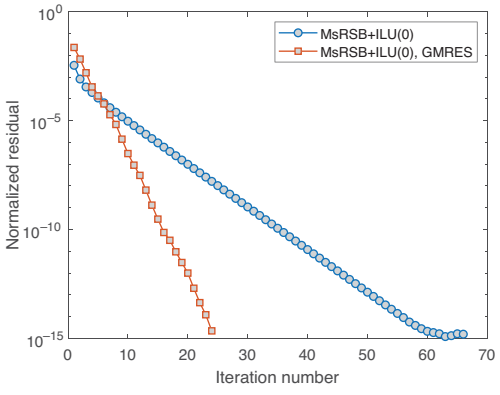


Figure 4.15 Decay in the residual norm for the iterative two-stage MsRSB method (4.11) with ILU(0) as smoother compared with the residual decay observed when the same two stages are used as a preconditioner for GMRES.

With the smoother specified, we can call the solver to compute the solution using, e.g., no initial guess for $\boldsymbol{p}$, at most 50 iterations, and a residual tolerance of 10^{-6}:

```
p = solveMultiscaleIteratively(A, b, [], basis, fn, 1e-6, 50);
```

By default, the solver uses the two-stage iteration (4.11), for which the essential code lines are summarized quite compactly:

```
mssolve  = @(d) basis.B * mldivide(basis.R*A*basis.B, basis.R*d);
resnorm  = @(p) norm(b-A*p,2)/norm(b,2);
res      = zeros(nit+1,1);
p        = mssolve(b);
res(1)   = resnorm(p);
for i=2:nit+1
    p = p + S(b - A*p);
    p = p + mssolve(b-A*p);
    res(i) = resnorm(p);
end
```

The first line defines an anonymous function that computes the multiscale correction, and `resnorm` evaluates the residual normalized by the norm of the right-hand side. The rest of the code should be self-explanatory. The actual implementation includes a more advanced loop control so that the iteration terminates once the residual norm is below the prescribed tolerance.

Figure 4.15 shows that using a single step of the two-stage iteration as a preconditioner to MATLAB's built-in GMRES solver is much more efficient than the basic Richardson iteration if we want to reduce the residual to machine precision. The GMRES version is invoked by the call

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

```
[pg, report] = solveMultiscaleIteratively(A, b, [], basis, fn, ...
                                          5e-15, nit, @mldivide, true);
```

Here, the second output argument is a result report that contains the total number of iterations, the residual for each of the iterations, and the coarse-scale matrix.

Notice that the iterative multiscale solver only computes the fine-scale pressure. If you also want the corresponding fluxes, the solver must be called through the higher-level interface that corresponds to standard incompressible solvers in MRST:

```
[state,report] = incompMultiscale(state0, CG, hT, fluid, basis,'bc', bc, ...
    'getSmoother',fn,'iterations',nit, 'useGMRES', true, 'tolerance',5e-15);
```

There is also a multiscale linear solver class for use with the object-oriented, automatic differentiation (AD-OO) framework from the `ad-core` module, which uses the same internal interface. To use this, we first construct the solver and then use the `solveLinearSystem` interface (see the MRST textbook [25, subsection 12.3.4]) to solve the fine-scale system

```
solver = MultiscaleVolumeSolverAD(CG, 'getSmoother', fn, ...
          'maxIterations', 50, 'tolerance', 1e-6, 'useGMRES', true);
sol = solver.solveLinearSystem(A, b);
```

Note that this generates a basis from `A`, which can be reused for subsequent solves.

We end the section by remarking that we recently [22, 28] showed that iterative methods as just described can be made even more efficient if, instead of using a single pair of restriction–prolongation operators to define the multiscale systems $\boldsymbol{A}_h^H$, we cycle through an alternating sequence of operator pairs constructed from different types of partitions that, e.g., offer improved resolution of near-well regions and special features in the geological model or adapt to dynamic changes in the solution. It is also possible to extend the two-stage method to a multilevel setting, in which another multiscale solver is developed for the coarse system [49], and so on. However, we will not discuss these possibilities further herein.

4.3 Numerical Examples

This section highlights the performance and utility of the (iterative) multiscale solvers on a series of test cases, following a progression from incompressible single-phase flow to compressible multiphase flow.

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

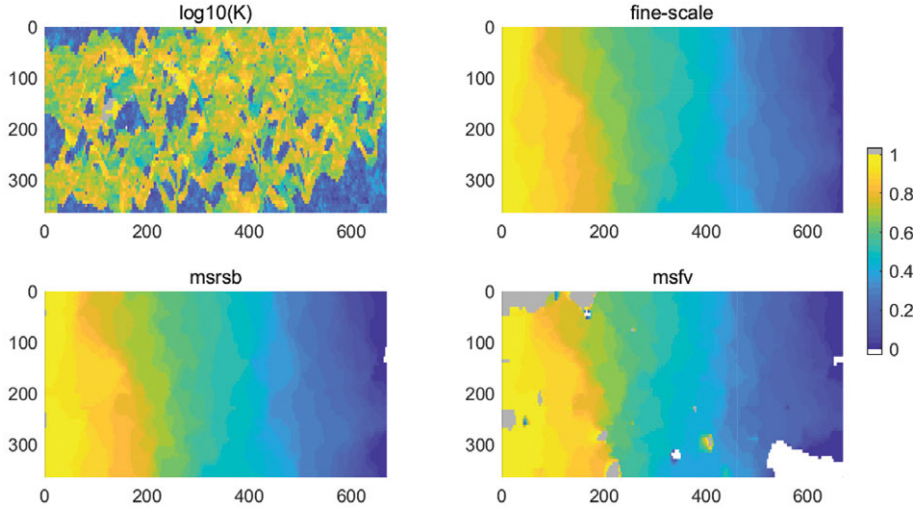


Figure 4.16 Lack of monotonicity for multiscale solutions computed for the bottom layer of SPE 10 subject to unit pressure drop from left to right. (Source code: `lackOfMonotonicity.m`.)

4.3.1 Lack of Monotonicity

The reduced system arising from a multiscale method will effectively represent a multipoint coarse-scale stencil, as shown in Figure 4.6, because of the overlap among basis functions from neighboring coarse blocks. Multipoint schemes are generally not monotone, as discussed in the MRST textbook [25, section 6.5], and give solutions that contain local oscillations and/or out-of-bound values. To illustrate this, we consider single-phase flow of a fluid with unit viscosity through the bottom layer of the SPE10 benchmark, subject to a unit pressure drop in the y-direction and sealing boundary conditions in the x-direction.

Figure 4.16 reports the permeability (shown on a logarithmic scale) along with the solutions computed on the fine scale and by the MsRSB and MsFV solvers without any iterations. The MsRSB solution exceeds unity by 0.3% in seven cells near the inflow boundary; likewise, there are seven cells near the outflow boundary in which the pressure is between -0.003 and 0. The MsFV solution, on the other hand, exhibits large patches of out-of-bound values. Altogether, there are 379 cells with values in the interval [1,15.7] and 421 cells with values in the interval [-5.86,0]. These large oscillations are primarily the result of the localization that attempts to extrapolate pressure values across high-contrast streaks that cross the edges in the wirebasket ordering. As a result of this tendency to introduce nonphysical oscillations, the iterative MsFV method not only starts with a higher residual norm than MsRSB but also needs more iterations to reduce the residual to below a prescribed tolerance, as shown in Figure 4.17.

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

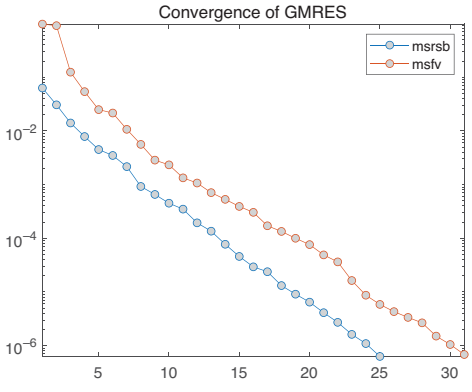


Figure 4.17 Convergence history for the iterative MsFV and MsRSB solvers applied to the bottom layer of SPE10 shown in Figure 4.16. Because MsFV has a tendency to create nonphysical oscillations, it needs more iterations to reduce the residual compared with MsRSB. (Source code: `lackOfMonotonicity.m`.)

4.3.2 Grid-Orientation Errors

Because the MsFV and MsRSB methods give multipoint coarse-scale stencils, one may be tempted to believe that these schemes do not suffer from the type of grid-orientation errors discussed in chapter 6 of [25]. This is unfortunately not the case: If an inconsistent scheme like the TPFA method is used to compute basis functions over a grid that is not K-orthogonal, the resulting multiscale method will to a large degree inherit the corresponding inconsistencies. This is important to be aware of, and to illustrate, we revisit example 6.1.2 from [25], which considers a homogeneous reservoir with a symmetric well pattern consisting of one injector and two producers, posed on a skew grid that is not K-orthogonal. As we can see from Figure 4.18, the multiscale solutions exhibit significant grid-orientation errors, but these are less if we choose the coarse mesh to be less skewed than the underlying fine mesh. Bosma et al. [7] discuss how to formulate MsRSB with a consistent fine-scale discretization.

4.3.3 Coarsening Complex Meshes

We have already discussed some challenges related to coarsening of realistic reservoir geometries in Subsection 4.2.3 and explained how difficulties in obtaining suitable primal–dual partitions limit the applicability of the MsFV method. You can find more details about difficulties related to faults in [43]. For the MsRSB method, it is sufficient that the cells in the support boundaries $\mathcal{B}(j)$ are connected through entities of codimension two (i.e., cell edges in 3D and vertices in 2D). In practice, this means that it is much easier to generate suitable coarse partitions and

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

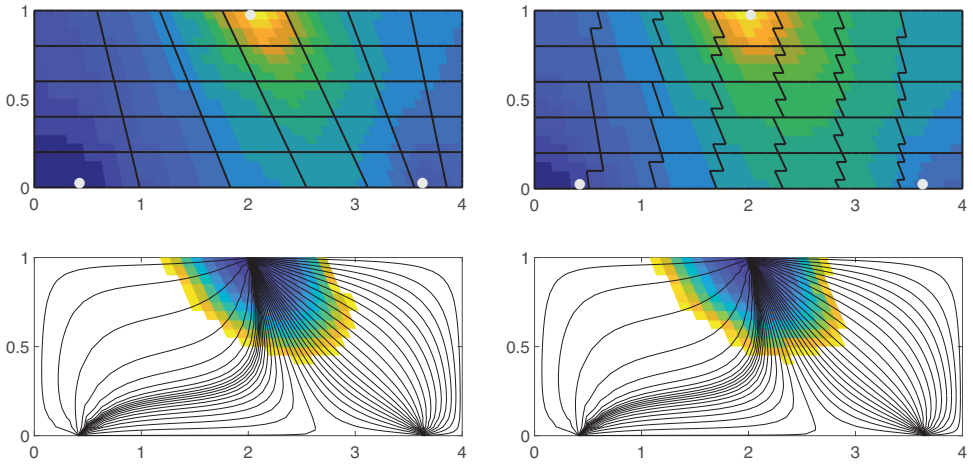


Figure 4.18 Solution of a symmetric flow problem in a homogeneous domain using the MsRSB method defined on a 7×5 partition in index space (left column) and in physical space (right column). In both cases, the basis functions are computed by the TPFA method on a skew grid that is not K-orthogonal. The upper plots show the pressure distribution, and the lower plots show streamlines and values of time-of-flight less than 0.2 pore volumes injected. (Source code: `gridOrientationMsRSB.m`.)

accompanying support regions for general polytopal meshes and heavily faulted meshes like the ones shown in Figure 4.7. In our experience, the main challenges for MsRSB do not come from complex topology but from strongly degenerate and deviated cell geometries.

To exemplify, we consider a highly detailed, core-scale model of realistic bedding structures, which is typically developed to derive directional permeability for a given lithofacies and identify net pay below the level of petrophysical log resolution. MRST contains several examples of such models in the `BedModels1` and `BedModel2` data sets. Here, we use the $30 \times 30 \times 333$ corner-point model from `BedModel2`; see Figure 4.19. The model has many layers that are fully or partially eroded away, so that only 30% of the logical cells are part of the active model. Among the active cells, the smallest half only make up 11% of the bulk volume. As a result, the model has four orders of variations in cells volumes and three orders of variation in the areas of the vertical faces. Many of the thin cells are also strongly curved so that cell centroids fall far outside the cells.

The standard approach to partition a model is to create a load-balanced partition in index space or use cell coordinates (centroids or vertices) to partition the cells into rectangular boxes in physical space. As an alternative, we can also try to make partitions that better reflect the structure of the model by merging layers

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

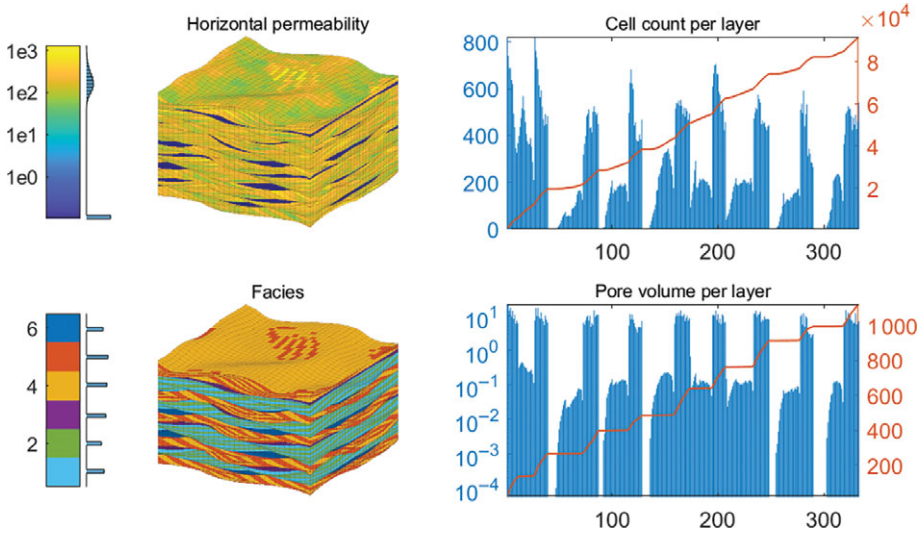


Figure 4.19 Model of realistic bedding structure. The left column shows the permeability and the distribution the six rock types, and the right column shows the distribution and cumulative distribution of cells and pore volume per cell layer in the ijk topology. (Source code: `bedModelMS.m`.)

vertically to obtain coarse layers with a certain "thickness" measured in cell count or bulk/pore volume. The bar charts in Figure 4.19 indicate nine natural groups we can use as a starting point. Once we have obtained a satisfactory vertical partition, we can use a standard index partition in the horizontal direction. To this end, we start by computing the cell count per layer and then use the cumulative sum to partition the layers into bins of approximately the same number of cells:

```
[~,~,K] = gridLogicalIndices(G);
ncell   = accumarray(K,1);
layPart = discretize(cumsum(ncell),linspace(0,G.cells.num+1,10));
```

Here, `layPart` is a standard partition vector that contains value m in element k if layer k is part of coarse layer number m. We then use run-length encoding to compress the partition vector and construct an indirection map, which we pass to one of MRST's standard partition routines (see [25, subsection 15.6.4]):

```
[~,edges] = rlencode(layPart);
part      = partitionLayers(G, [6 6], [1; cumsum(edges)+1]);
```

Figure 4.20 shows the resulting partition. The multiscale solution exhibits overshoots in a few cells near the inflow boundary but otherwise is visually

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

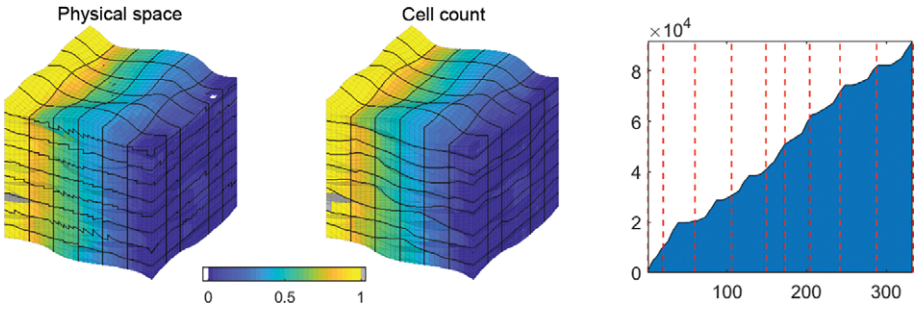


Figure 4.20 Comparison of multiscale solutions for two different partitions: $6 \times 6 \times 9$ in physical space (left) and vertical partition by cell count combined with horizontal 6×6 partition (middle). The right figure shows the cumulative distribution of cells per layer, with red dashed lines indicating the edges separating different coarse layers. (Source code: `bedModelMS.m`.)

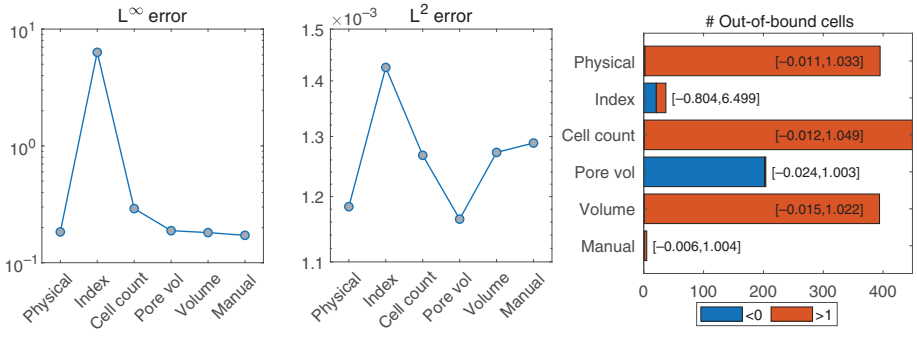


Figure 4.21 Error measures for the six different partitions of the bed model.

indistinguishable from the fine-scale solution (which is not shown for brevity). The left plot in the figure clearly shows how the edges of the coarse layers fall near the plateaus in the cumulative cell distribution (these plateaus are more distinct for the cumulative pore volume distribution). The figure also shows a uniform partition in physical space. Partitioning in index by merging layers has the advantage that the coarse block interfaces follow the bedding. Partitioning in physical space gives jagged coarse faces, which can be more pronounced for other coarsening factors.

In addition to the two partitions shown in Figure 4.20, we ran four other partitions: a load-balanced $6 \times 6 \times 9$ partition in index space, two partitions in which vertical layers are merged according to bulk and pore volume targets, and a manual partition in which we tried to place the layer edges in the middle of the plateaus for the cumulative pore volume function. Figure 4.21 reports various error measures for these six different partitions. All over, the manual partition seems to give the best

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

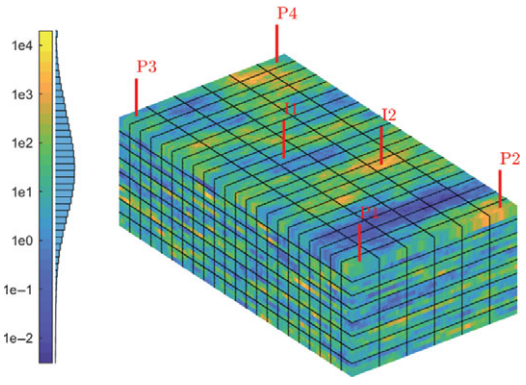


Figure 4.22 A 60 × 220 × 24 subset of the Tarbert formation from the SPE10 upscaling benchmark, with the 6 × 22 × 8 coarse grid outlined in black. The well pattern differs from the original benchmark and follows the MRST textbook [25, subsection 15.6.3], except for a slight offset to avoid having wells in the corner of the coarse blocks. (Source code: `upscalingVsMsRSB.m`.)

compromise, having the fewest out-of-bound cells, second lowest L^∞ error, and an L^2 error that is only 10% larger than the lowest. The worst behavior is observed for the uniform partition in index space. This partition has second fewest out-of-bound cells, but some of these cells are really far off and contribute to the largest overall L^∞ and L^2 errors. A closer inspection shows that the worst monotonicity violations can be traced back to coarse blocks that are partially eroded away so that the blocks above and below also share a direct connection. In a more advanced approach, one should thus postprocess the partition to merge and get rid of such pinched blocks. (This would be similar to what is achieved by the PINCH keyword in ECLIPSE input decks.)

4.3.4 Multiscale Methods as an Alternative to Upscaling

Next, we compare MsRSB to three upscaling methods: harmonic–arithmetic averaging, local flow-based upscaling with sealing boundary conditions for upscaling permeability as outlined in Subsection 4.1.3, and a global upscaling method in which representative global flow solutions are imposed as accurate boundary conditions on the local regions used to compute effective transmissibilities and well indices [25, p. 584]. The setup is almost the same as in subsection 15.6.3 of the MRST textbook [25], except that we now only consider the top 24 layers of the Tarbert formation to reduce computational time. Figure 4.22 shows the fine-scale model with the coarse grid outlined in black. All wells are controlled by bottom-hole pressure.

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

The MsRSB method is known to suffer from inaccuracies if wells are placed near the corner of internal coarse blocks. This can be easily remedied by changing the coarse grid locally around the affected wells or by introducing extra well bases as discussed in [28]. These bases are not yet part of the basic setup in the `msrsb`, and as a pragmatic workaround we have modified the well positions slightly so that they fall near the center of the coarse blocks.

With 1 056 coarse blocks that each consists of 300 fine cells, the computational overhead induced by MATLAB is unfortunately significant for the routines we have used thus far to set up support regions and compute basis functions. In this examples, we therefore introduce accelerated versions:

```
CG    = storeInteractionRegionCart(CG);
CG    = setupMexInteractionMapping(CG);
basis = getMultiscaleBasis(CG, A, 'type', 'msrsb', 'useMex', true);
```

The first routine is a specialized version of `storeInterationRegion`. For coarse and fine meshes with a strict Cartesian topology, each support region is a rectangular block in index space, bounded by the centroids of blocks that are nodal neighbors. The corresponding cells can be located very quickly operating on the logical indices only. This assumption is not valid for more general coarse/fine meshes.

The second function adds additional mappings to the coarse grid to enable C-accelerated implementation of basis functions. This is somewhat time consuming for a large grid but can be done once for a specific partition and reused throughout a simulation. We can then pass the flag `'useMex'` to the standard interface for computing basis functions to tell the routine that it should use the C-accelerated backend (`cppMultiscaleBasis`). Note that this requires a working C++ compiler to be configured with the MATLAB builtin `mex -setup` and may take some time to compile when first run. This interface also offers the opportunity to write basis functions to disk as a series of text files; this is time consuming but can be useful for setting up and generating basis functions that can be reused for some other purpose. You can find more details in the tutorial examples of the `msrsb` module.

To compare the accuracy of the different methods, we consider plots of well allocation in Figure 4.23 computed by flow diagnostics. In short, the colored sectors report that the cumulative flux out of injector I2, from the bottom to the top of the well, is distributed onto flow paths that end up in each of the four producers, P1 to P4; see subsections 13.1.3 and 13.4.3 of the MRST textbook [25] for a more detailed description. The dark lines outline the same allocation for the fine-scale model, and the closer the colored sectors match these lines, the more accurately

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

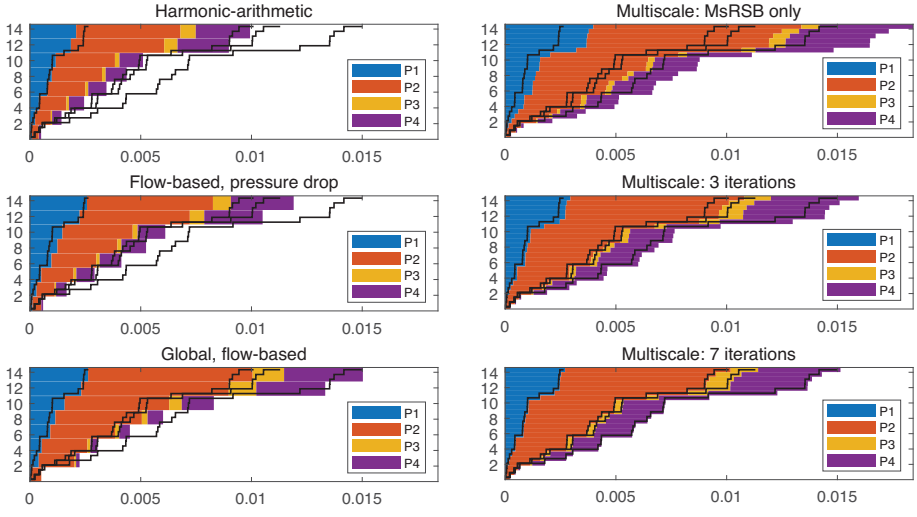


Figure 4.23 Cumulative well allocation factors, from bottom to top, for injector I2 in the Tarbert test case (Figure 4.22) computed by three upscaling methods and MsRSB. Dark lines outline the allocation factors computed on the underlying fine model. (Source code: `upscalingVsMsRSB.m`.)

the upscaled or multiscale solution reproduces the volumetric connections in the fine-scale model. Here, we see that the averaging and flow-based upscaling methods underestimate the inflow into the reservoir, whereas the MsRSB method tends to overestimate it. The global upscaling method uses specific information obtained from the solution of the fine-scale flow problem to compute effective properties and is therefore more accurate, but by adding only three iterations, which are not very computationally expensive, the MsRSB method achieves better resolution. With seven iterations, the match is almost exact.

4.3.5 Incompressible Multiphase Flow in Fractured Media

Accelerating the simulation of multiphase flow by systematic reuse of basis functions from one time step to the next has been a key argument for multiscale methods since their inception. In many cases, it is sufficient to compute the basis functions at the outset of a simulation and keep the resulting prolongation and restriction operators fixed throughout the whole simulation. In some cases, the effective permeability may change so much as a result of fluid movement that the basis functions cease being representative and need to be updated. However, such updates are usually localized and can, for the MsRSB method in particular, be obtained by continuing to iterate on the existing basis functions using an updated $\boldsymbol{A}$ matrix.

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

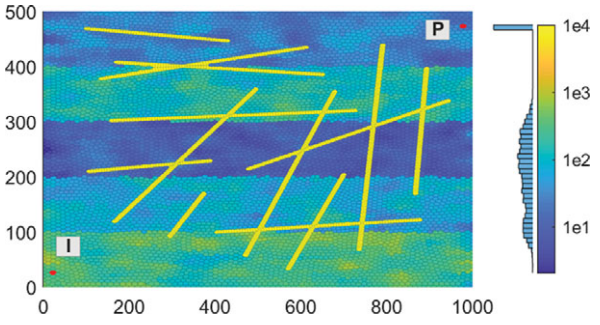


Figure 4.24 A two-phase fractured example adapted from [47]. The 13 fractures are represented volumetrically on an unstructured perpendicular bisector mesh generated by the `upr` module described in Chapter 1.

To demonstrate the utility of reusing basis functions in a multiphase simulation, we consider a simplified, incompressible, two-phase analogue of the example studied in subsection 5.2 of [47]; see Figure 4.24. We inject a very light fluid of density 300 kg/m^3 and viscosity 0.3 cP into another fluid of density 1 000 kg/m^3 and viscosity 1 cP that fills a layered medium containing 13 high-permeability fractures. The injected fluid is highly mobile and will reach the first fracture after a short time and then move rapidly through the fracture system to give early breakthrough in the producer. Complete setup of the problem is described in `fracturedExampleMS.m`.

We simulate the injection over a period of 10 years using a sequential solution method that combines the standard implicit transport solver from `incomp` with three different pressure solvers: the standard `incompTPFA` solver and the MsRSB solver with either a rectangular partition or an unstructured partition generated by the METIS graph partitioning software. The solvers are set up as anonymous functions:

```
tsolver   = @(state,dt) implicitTransport(state, G, dt, rock, fluid, 'W', W);
psolver   = @(state)    incompTPFA(state, G, hT, fluid, 'W', W);
mssolver1 = @(state)    incompMultiscale(state, CG1, hT, fluid, basis1, 'W', W);
mssolver2 = @(state)    incompMultiscale(state, CG2, hT, fluid, basis2, 'W', W);
```

Each of these functions compute a single transport or pressure step in the sequential solution algorithm. To stabilize the simulation, we use a geometric rampup of the first time steps and then continue with a uniform time step. The time steps are kept in the array `dt`. The essential lines of the main simulation loop then read:

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

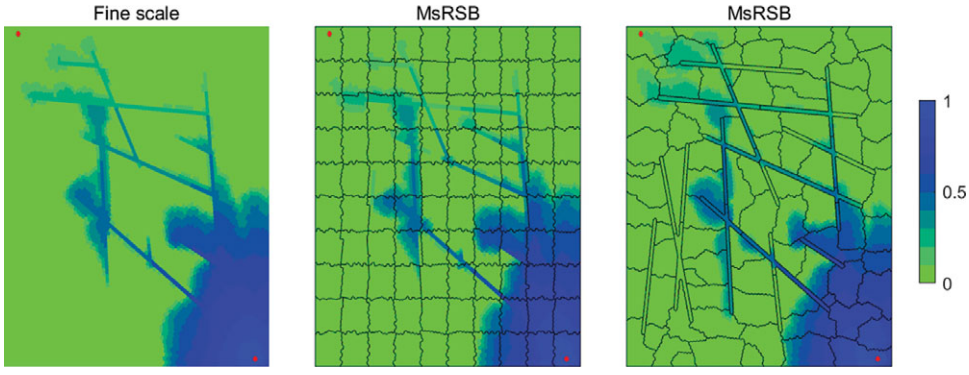


Figure 4.25 Saturation profiles after 511 days for the fractured test case (Figure 4.24) simulated with a fine-scale solver (left) and with a multiscale solver with a rectangular coarse grid (middle) and an unstructured coarse grid adapted to the fractures (right).

```
rstates{1} = psolver(state0);
rws{1} = getWellSol(W, rstates{1},  fluid);

for i=1:numel(dt)
  state         = psolver(rstates{i});
  rstates{i+1}  = tsolver(state, dt(i));
  rws{i+1}      = getWellSol(W, rstates{i+1}, fluid);
end
```

Here, `rstates` and `rws` are cell arrays that keep the reservoir states and well responses for each time step. The updates for the multiscale solvers are virtually identical and are omitted for brevity.

Figure 4.25 reports saturation profiles just after the first of the simulations have broken through, and Figure 4.26 reports the corresponding fractional flow in the producer. This happens for MsRSB with the adapted partition. For this partition, we have configured METIS with fine-scale transmissibilities as edge weights in the matrix and zero weight across interfaces between matrix and fracture cells to ensure that the high-permeability fractures are represented as separate objects. This results in a set of elongated coarse blocks that run across multiple blocks in the matrix. These blocks cause MsRSB to overestimate flow through the fracture network, resulting in premature breakthrough. In contrast, the straightforward rectangular partition gives remarkable accuracy throughout the whole simulation, even without any updating of basis functions.

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

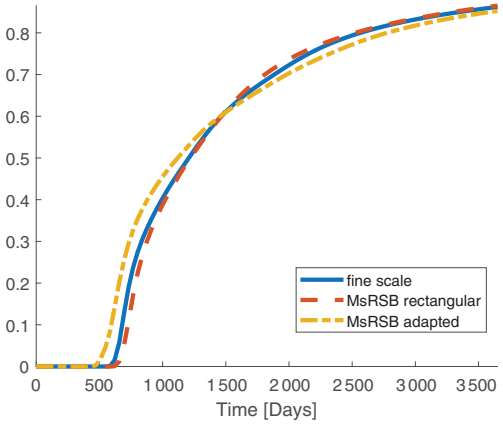


Figure 4.26 Fractional flow of the injected fluid in the producer for the three different simulations of the fractured test case in Figure 4.24. The MsRSB solver predicts premature breakthrough of the injected fluid on the adapted partition but matches the fine-scale solution quite accurately on the rectangular partition.

In previous research, we have argued and demonstrated that using adapted partitions can increase accuracy of the MsRSB method. However, as this example and the one in Subsection 4.3.3 show, you need to know what you are doing to avoid introducing artifacts that diminish the performance of the method. Our general advice is thus to start with a relatively simple partition and only try to do something clever if simple partitioning strategies fail to give the desired accuracy or robustness. (Adapted partitions pay more off when used as part of a multibasis approach [28].)

4.3.6 Gravity Segregation

As the previous example demonstrated, you can obtain good accuracy in many dynamic flow simulations without updating basis functions or introducing iterations. However, because the basis functions essentially are localized solutions of a homogeneous elliptic flow problem, they cannot generally account correctly for flow patterns induced by strong compressibility, gravity, or capillary effects, and use of iterations is therefore necessary. To illustrate, we consider gravity segregation of a heavier fluid with density 1 000 kg/m^3 and viscosity 1 cP placed on top of a lighter fluid with density 600 kg/m^3 and viscosity 0.5 cP inside a 500×60 m^2 vertical cross section with heterogeneous media properties; see Figure 4.27. Both fluids are assumed to have quadratic relative permeabilities.

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

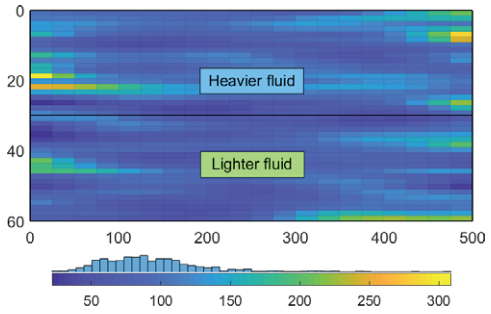


Figure 4.27 Setup for the gravity-segregation test case with a heavier fluid on top of a lighter fluid. The permeability is related to a normally distributed porosity through the Carman–Kozeny relationship. (Source code: `gravitySegregationMS.m`.)

We simulate the gravity segregation over a 10-year period using 100 equally spaced time steps and the standard implicit transport solver from the `incomp` module combined with three different pressure solvers: the standard `incompTPFA` solver and the MsRSB solver with and without iterations. The solvers are set up as anonymous functions and used in a simple sequential loop in almost exactly the same way as in the previous section.

Figure 4.28 reports snapshots from the dynamic simulation. We clearly see that the plain multiscale solver captures the general behavior of the fluid movement but fails to resolve details correctly, giving an L^1 error with mean value 6.4% and maximum value 9.4% over the 10-year period. The iterative version, on the other hand, can be set to match the fine-scale solver as close as you wish. With a residual tolerance of 10^{-3} and an average of 4.6 iterations per step, the resulting fluid distributions are visually indistinguishable from the fine-scale solution, having an L^1 error with mean value 0.12% and maximum value 0.21%.

4.3.7 Compressible Black-Oil Models: Fully Implicit Methods and CPR

In this section, we will demonstrate how multiscale methods can be used to solve compressible black-oil models. Let us start by considering a fully implicit (FI) formulation, which can be considered the industry standard approach over the past decades. (Chapters 11 and 12 of the MRST textbook [25] give more details.) The natural way to use multiscale methods in an FI setting would be to use them as solvers for the elliptic pressure equation formed in a constrained pressure residual (CPR) preconditioner (see, e.g., [25, subsection 12.3.4]), which represents state-of-the-art for solving linearized black-oil equations.

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

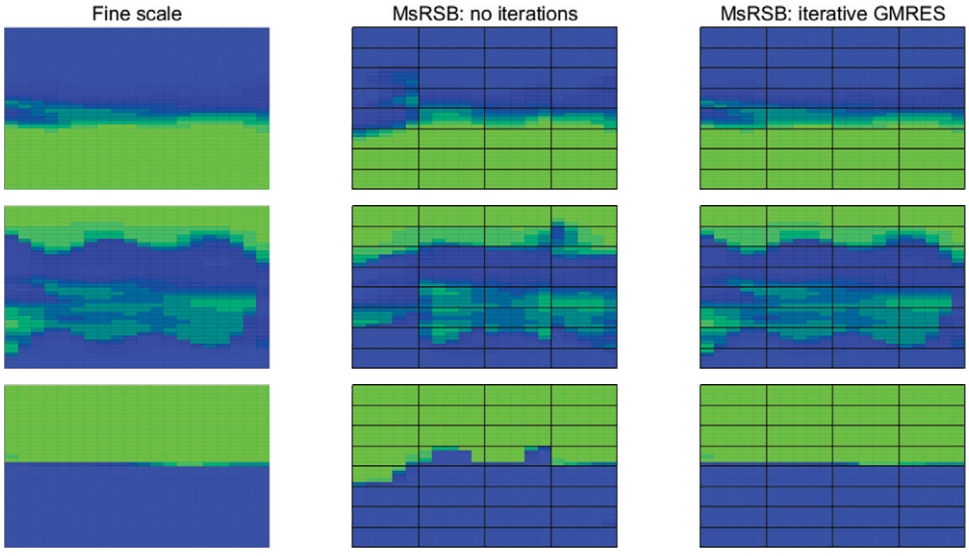


Figure 4.28 Snapshots of the gravity-segregation process after 146 days (top), 1 460 days (middle), and 10 years (bottom). Green denotes the lightest fluid and blue denotes the heaviest fluid.

To demonstrate the use of MsRSB as part of a CPR preconditioner, we consider the synthetic sector model from subsection 15.6.4 of the MRST textbook [25], which models a waterflooding scenario with one injector and three producers using a slightly compressible two-phase dead-oil model without any dissolved gas. The full setup is presented in the script `sectorModelMS.m` and relies on the model classes from the `ad-blackoil` module and the solver framework from `ad-core`.

The model contains several eroded layers with inactive cells and, as discussed in Subsection 4.3.3, there are many different ways we could partition such a model. However, to be consistent with our advice from Subsection 4.3.5, we choose a straightforward uniform $8 \times 8 \times 3$ partition (Figure 4.29), which is sufficient to get a significant reduction in the number of unknowns of the coarse-scale system compared with the fine-scale system and also avoid creating partially eroded blocks that potentially could lead to monotonicity issues. The construction of a coarse-grid structure and support regions is the same as in previous examples, and we omit the code for brevity. We can then instantiate a multiscale solver class and configure it to use the two-stage method with ILU(0) as smoother:

```
msSolver = MultiscaleVolumeSolverAD(CG, 'tolerance', 1e-4, ...
           'maxIterations', 1, 'useGMRES', false, 'verbose', false, ...
           'getSmoother', getSmootherFunction('type', 'ilu0', 'iterations', 1));
```

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

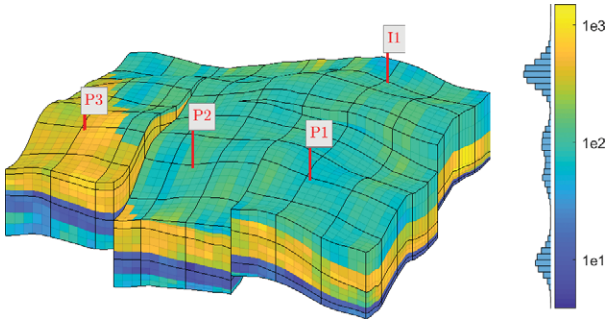


Figure 4.29 The 40 × 40 × 12 sector model from [25, subsection 15.6.4] with a 8 × 8 × 3 logical partition that gives 201 coarse blocks because of the three faults. The model spans four layers with distinctly different permeability; all layers except for the second are partially eroded. (Source code: `sectorModelMS.m`.)

To set up the full solver, we pass the `msSolver` object as an elliptic solver to the CPR interface for black-oil-type equations from the AD-OO framework:

```
linsolve = CPRSolverAD('ellipticSolver', msSolver, 'relativeTolerance', 1e-3);
```

Once we have set up the model, the initial data, and the simulation schedule and configured the linear solver hierarchy properly, we can run the simulation using the standard method from `ad-core`:

```
[wellSolsMS, statesMS, reportMS] = ...
   simulateScheduleAD(state0, model, schedule, 'LinearSolver', linsolve);
```

Figure 4.30 reports oil rates in all three producers and confirms that the multiscale solver produces the exact same solutions as the reference simulation. This simulation utilized an agglomeration-based algebraic multigrid (AMG) method from the AMGCL library [11] to solve the elliptic equation in the CPR preconditioner; see Chapter 6 for more details. Figure 4.31 compares the CPU times consumed by the linear solver part of the simulation and confirms that MsRSB and the AMG solver give more or less the same performance in terms of computational costs.

The reader should be cautious when interpreting (and extrapolating) runtime comparisons like the one shown in Figure 4.31. Neither `CPRSolverAD` nor the multiscale methods in the `msrsb` module are optimized for speed. Our focus has instead been to create flexible implementations that can be used to investigate new computational algorithms. For this reason, we also ran the

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

examples in single-threaded mode. In other words: The purpose of the `msrsb` module is not to create a high-performing solver but rather to provide a proof-of-concept and demonstrate the potential of a new technology. If you primarily want to accelerate (the linear solves of) your MRST simulator, we rather encourage you to look into the high-performance CPR solver implemented as a part of AMGCL outlined in the MRST textbook [25, subsection 12.3.4] and in Chapter 6.

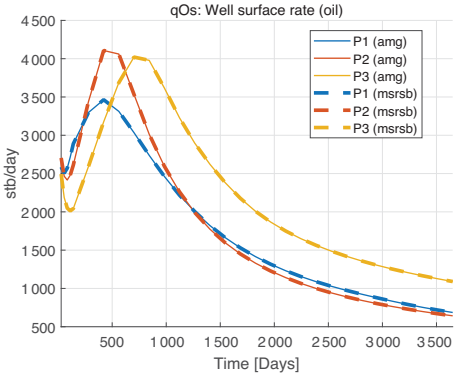


Figure 4.30 Surface oil rate in all producers predicted by the two simulations using AMG and MsRSB as solvers for the elliptic equations in the CPR preconditioner.

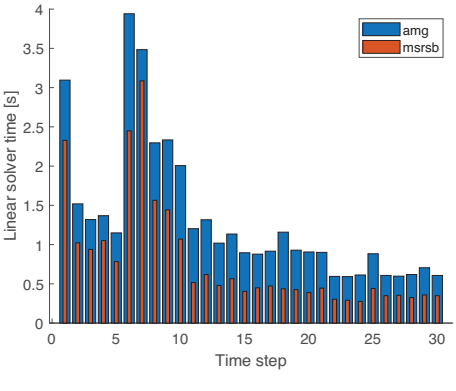


Figure 4.31 CPU times for the linear solves for each fully implicit time step with AMG and MsRSB as solvers for the elliptic equations in the CPR preconditioner.

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

4.3.8 Compressible Black-Oil Models: Sequential Solution Methods

In recent years, there has been a revived interest in sequential implicit approaches for simulating the black-oil equations, and most of the research and commercial development of multiscale solvers for black-oil equations has focused on sequential formulations. To formulate a *sequential implicit* (SI) method, one starts by developing a pressure (or flow) equation as a weighted sum of all component conservation equations. The weights are defined so that the derivative of the sum of the accumulation terms with respect to any non-pressure variable is guaranteed to be identically equal to zero. One then proceeds by alternatingly solving the (parabolic) pressure and transport equations in the same way as for incompressible flow.

The transport equations can be composed in different ways. We can use the constraint that the fluid saturations sum to unity to eliminate one transport equation and solve for the others, which will ensure volume conservation but not mass conservation for compressible fluids. In our experience, however, it is better to solve all component equations, which ensures conservation of mass.

To demonstrate how MsRSB can be used in such a setting, we continue considering the sector model from the previous section. The `sequential` module, which is part of the AD-OO framework, has functionality that automatically sets up a sequential model, consisting of a pressure and transport submodel, from a given fully implicit simulation model. That is

```
seqmodel = getSequentialModelFromFI(model);
seqmodel.transportModel.conserveOil = true;
seqmodel.transportModel.conserveWater = true;
seqmodel.transportModel.useCNVConvergence = true;
```

The second and third lines specify that we should solve transport equations for both the oil and the water component (default: only water), whereas the fourth line specifies that we should use the maximum normalized residual in addition to mass balance to measure convergence (see the MRST textbook [25, subsection 12.3.2]). The `msrsb` module offers a function that sets up and adds a multiscale solver to the pressure equation of a sequential model, which can then be simulated as usual:

```
msmodel = addMultiscaleSolverComp(msmodel, CG, 'maxIterations', 50,...
              'useGMRES', true, 'tolerance', 0.01);
wsFIms  = simulateScheduleAD(state0, seqmodel, schedule);
```

The approximate solution obtained after two consecutive pressure and transport steps will generally neither satisfy the residual equations of the underlying fully implicit discretization exactly nor conserve both mass and volume (depending on the formulation). Hence, the FI and SI methods will not produce the same solutions.

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

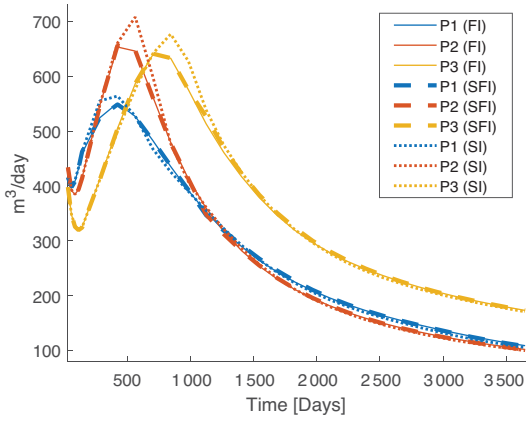


Figure 4.32 Surface oil rates for all three producers in the sector model from Figure 4.29 predicted by the FI, SFI, and SI methods. (Source code: `sectorModelMS.m`.)

The *sequential fully implicit* (SFI) method attempts to remedy this shortcoming of the SI method by introducing outer iterations that systematically reduce the total residuals and the mass/volume discrepancies below a prescribed tolerance. In MRST, the sequential model just described can easily be changed to describe an SFI simulation by adjusting a few parameters as follows:

```
seqmodel.stepFunctionIsLinear        = false;
seqmodel.outerCheckParentConvergence = false;
seqmodel.volumeDiscrepancyTolerance  = 3e-3;
seqmodel.incTolSaturation            = inf;
```

The first parameter says that each time step is not linear and should thus be iterated upon until convergence in some norm(s) prescribed by the user. The next three parameters specify these norms and here configure the algorithm to only check the volume discrepancy $\left(\left|\sum_\alpha S_\alpha - 1\right|\right)$ against a prescribed tolerance and neither check the fully implicit residual nor the increments in saturations from one transport step to the next within the outer iteration. The volume discrepancy is adjusted up from the very strict default of 1e-3 to a value of 3e-3. In many cases, larger values can be allowed without significant impact to the simulation accuracy.

Figure 4.32 reports surface oil rates for all three producers. The SI method is generally able to reproduce the qualitative behavior of the FI solution but overestimates the peak production rates in all producers. We expect that the discrepancy

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

between the two methods would have been smaller if we had used a longer rampup period for the initial time steps. Adding outer iterations gives an SFI solution that is almost indistinguishable from the FI solution. At most, the SFI method uses three outer iterations in step number 6 (corresponding to the peak in P2) but otherwise converges in one or two iterations in the other steps, giving a total of 40 outer iterations over the 30 time steps.

4.3.9 Compositional Flow

In the previous examples, we have looked at a range of different ways of using multiscale methods. We will conclude the numerical examples with an isothermal compositional version of the incompressible two-phase scenario in Subsection 4.3.5. The reservoir is initially filled with 30% methane, 10% carbon dioxide, and 60% n-decane by moles at a pressure of 75 bar, 150°C in a mostly liquid state. We simulate the injection of approximately 0.25 pore volumes of gas over a 7-year period. The injection gas is mostly supercritical carbon dioxide but contains 10% methane. Vapor–liquid equilibrium and densities are modeled by the Peng–Robinson equation of state, with the Lohrenz–Bray–Clark viscosity correlation for both phases. The pressure drop over the domain is slightly below 50 bar, with a producer operating at fixed bottom-hole pressure of 50 bar.

Our baseline is a fully implicit simulation with the `OverallCompositionModel` from MRST's `compositional` module. (This module is discussed in detail in Chapter 8.) The simulation uses a uniform step length of 20 days after an initial rampup period, giving a total of 136 time steps. Computational simulations are generally time consuming, and it is therefore important to use features in MRST that are optimized for computational performance. That is, we use the accelerated backend for automatic differentiation introduced in Chapter 6 in combination with optimized linear solvers from the AMGCL library [11]. For convenience, we also use functionality for so-called packed simulations discussed in Chapter 6, which runs simulations and stores computed time steps consecutively on disk. This enables you to easily retrieve simulation results that have been run in a previous MATLAB session or restart simulations that have been interrupted. Complete source code for the whole setup is given in `fracturedExampleCompositionalMS.m`.

We compare the baseline simulation to a sequential setup with three different linear solvers for the pressure equation (Figure 4.33): (i) an accurate algebraic multigrid (AMG) solver, (ii) a multiscale solver with relaxed tolerances, and (iii) a multiscale solver with the same tolerances as the AMG solver. As before,

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

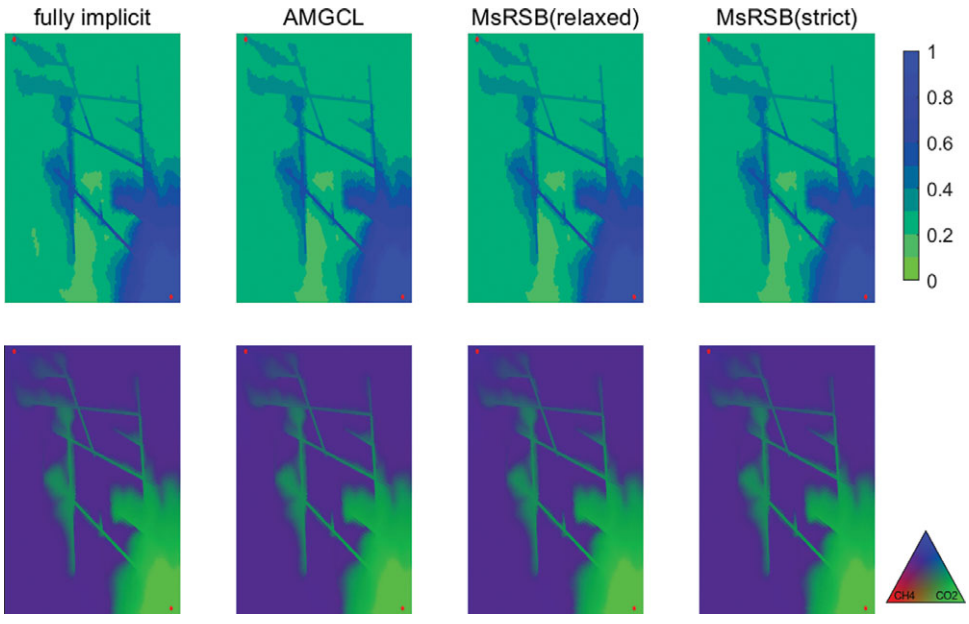


Figure 4.33 Saturation of the hydrocarbon phase (top) and composition (bottom) after 660 days.

we build a sequential model from the fully implicit model and add the multiscale solvers to it:

```
msmodel = getSequentialModelFromFI(model);
msmodel = addMultiscaleSolverComp(msmodel, CG, 'maxIterations', 50, ...
             'useGMRES', true, 'tolerance', 1e-3);
msmodel.pressureModel.incTolPressure = 0.05;      % Default 1e-3
```

The last line imposes the relaxed convergence criterion on the pressure increment, which is defined as $\|\Delta p\|_\infty/(\max(p) - \min(p))$. Using relaxed tolerances in the pressure solver requires that the transport solver is sufficiently robust to converge. Fortunately, the compositional transport solvers based on a total saturation formulation can produce reliable results even when there is significant compressibility and volume change on mixing.

The problem is challenging to simulate because of the orders of magnitude differences in flow velocities in the background sand and the high-permeability fractures. We decided to run with uniform time steps of 20 days, which give a reasonable compromise between having too high Courant numbers in the fractures and too low Courant numbers in the background matrix. In addition, we add a standard rampup that doubles the time step from 1.875 hours and up to 20 days to stabilize the

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

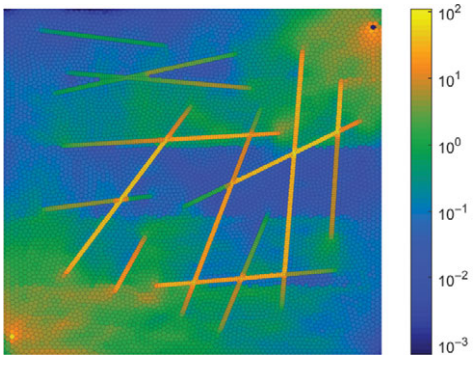


Figure 4.34 Maximum cell-wise compositional Courant number over all time steps for the compositional fracture example.

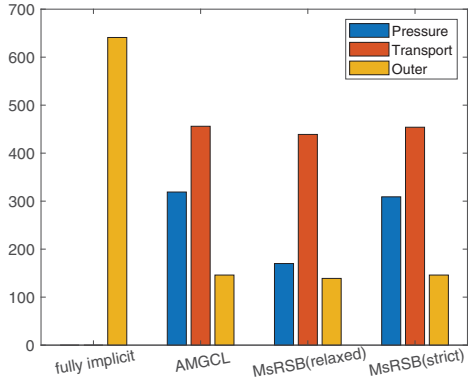


Figure 4.35 Nonlinear iterations in the pressure step, transport step, and the outer loop for the compositional example.

displacement fronts as they move into the reservoir. This gives a total of 136 time steps, with maximum Courant numbers between 1 and 10 in the background sand but exceeding 100 in the high-permeability fractures (Figure 4.34). We see this reflected in the nonlinear iterations in Figure 4.35: The fully implicit solver uses a total of 641 iterations, or approximately 4.8 iterations per step, whereas the baseline sequential solver needs a total of 332 pressure iterations and 446 transport iterations. The strict multiscale solver is comparable to the baseline sequential solver, with 303 pressure and 467 transport iterations. Using a relaxed tolerance nearly halves the number of pressure iterations to 184 without introducing significant difficulties in transport (450 iterations). Small variations in the pressure solution can lead to one or more additional iterations in transport, but it is encouraging that we can relax the pressure tolerance without introducing significant performance degradation.

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

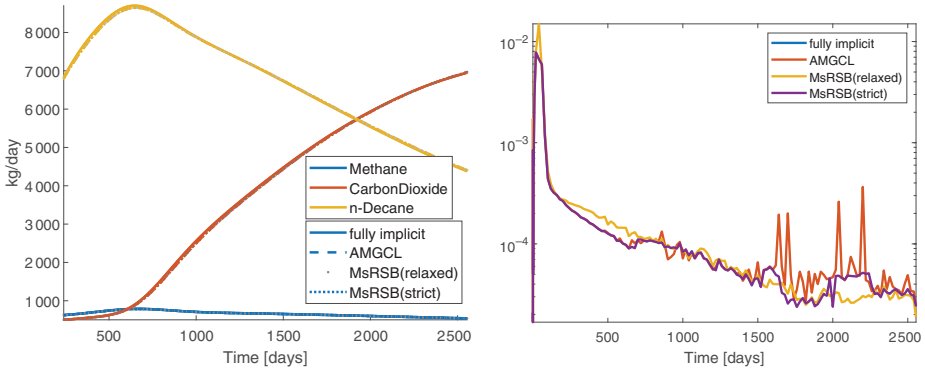


Figure 4.36 Left: Production rates for each mass component in the fracture example. Right: Volume discrepancy for each of the four simulations in the fracture example.

Reducing the number of iterations is of questionable value if it also reduces the accuracy of the solver. The plot of component production rates to the left in Figure 4.36 verifies that there is excellent agreement among all four solvers. We know that the sequential scheme is mass conservative for all components without an outer loop, because we solve for all components at once. One way to measure the error of the solution is then to look at the *volume discrepancy* introduced by the transport solver,

$$e_s = \int \phi |(S_l + S_v) - 1| \, \mathrm{d}V \Big/ \int \phi \, \mathrm{d}V. \tag{4.12}$$

The right plot in Figure 4.36 confirms that all three sequential simulators have a low volume error relative to the total pore volume of the domain, except in the beginning of the simulation. The pressure equation in each step will ensure that the volume discrepancy from the previous time step is not carried forward. In particular, we observe that there is no additional increase in volume discrepancy for the solver with looser tolerances, which used significantly fewer pressure iterations.

4.4 Concluding Remarks

In this chapter, you have seen how to use the MsRSB solvers on a variety of problems that range from immiscible and incompressible two-phase flow to compressible, miscible multicomponent flow. To apply the multiscale method, you must generally isolate a pressure-like equation, either as a CPR-type preconditioner [12, 55] or, as we have done here, as part of a sequential splitting algorithm in which the pressure-like degrees of freedom are frozen in a subsequent transport step. Different formulations have been proposed to this end [2, 10, 53, 58],

https://doi.org/10.1017/9781009019781.008 Published online by Cambridge University Press

and it is still an open question which approach is the best. Recent research has focused on developing robust splitting methods for general compositional cases [37–39, 46, 47]. The general idea is to use a weighted sum of the nonlinear conservation equations with weights that remove derivatives of the linearized accumulation terms not associated with pressure. The resulting linearized pressure system is similar to the one formed by the true-IMPES approach in a CPR preconditioner.

Once a proper pressure equation is formulated, the error of the multiscale method can be controlled by iterations in a similar way as in algebraic multigrid solvers; the use of such solvers in MRST is discussed in Chapter 6. Altogether, we believe that the current chapter gives a solid introduction to the different parts of a modern multiscale solver, because we have covered basis functions, coarse grids, sequential schemes, and iterative correction. The code described in this chapter can be used to quickly bootstrap new multilevel methods to complex flow problems, by modifying either of the constitutive parts with your method of choice. However, we emphasize that because the `msrsb` module is a prototype implementation intended for research purposes, you should not expect it to have the full computational efficiency of a solver implemented in a compiled language. Our current research on the MsRSB method focuses on integrating it efficiently into a multilevel solver framework [49]. Work also remains to be done on efficient multicore and multiprocessor implementations; see [20, 24, 35, 36] for some early work to this end.

Acknowledgement. The research leading up to the `msrsb` module has been partially funded by the Research Council of Norway (grant nos. 226035 and 244361), Schlumberger Information Solutions, and VISTA, a basic research program funded by Equinor and conducted in close collaboration with The Norwegian Academy of Science and Letters.

References

[1] J. E. Aarnes, S. Krogstad, and K.-A. Lie. Multiscale mixed/mimetic methods on corner-point grids. *Computational Geosciences*, 12(3):297–315, 2008. doi: 10.1007/s10596-007-9072-8.

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

[3] T. Arbogast. Numerical subgrid upscaling of two-phase flow in porous media. In *Numerical Treatment of Multiphase Flows in Porous Media (Beijing, 1999)*, volume 552 of *Lecture Notes in Physics*, pages 35–49. Springer, Berlin, 2000. doi: 10.1007/3-540-45467-5_3.

[4] T. Arbogast. Implementation of a locally conservative numerical subgrid upscaling scheme for two-phase Darcy flow. *Computational Geosciences*, 6(3–4):453–481, 2002. doi: 10.1023/A:1021295215383.

[5] M. J. Blunt. *Multiphase Flow in Permeable Media: A Pore-Scale Perspective*. Cambridge University Press, Cambridge, UK, 2017. doi:10.1017/9781316145098.

[6] G. Bonfigli and P. Jenny. Recent developments in the multi-scale-finite-volume procedure. In I. Lirkov, S. Margenov, and J. Wasniewski, eds., *Large-Scale Scientific Computing*, volume 5910 of *Lecture Notes in Computer Science*, pp. 124–131. Springer, Berlin, 2010. doi: 10.1007/978-3-642-12535-5_13.

[7] S. B. M. Bosma, S. Kletsov, O. Møyner, and N. Castelletto. Enhanced multiscale restriction-smoothed basis (MsRSB) preconditioning with applications to porous media flow and geomechanics. *Journal of Computational Physics*, 428, 109934, 2021. doi: 10.1016/j.jcp.2020.109934.

[8] Z. Chen and T. Y. Hou. A mixed multiscale finite element method for elliptic problems with oscillating coefficients. *Mathematics of Computation*, 72:541–576, 2003. doi: 10.1090/S0025-5718-02-01441-2.

[9] M. A. Christie and M. J. Blunt. Tenth SPE comparative solution project: A comparison of upscaling techniques. *SPE Reservoir Evaluation and Engineering*, 4:308–317, 2001. doi: 10.2118/72469-PA. URL www.spe.org/web/csp/datasets/set02.htm.

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

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

[12] S. Gries, K. Stüben, G. L. Brown, D. Chen, and D. A. Collins. Preconditioning for efficiently applying algebraic multigrid in fully implicit reservoir simulations. *SPE Journal*, 19(4):726–736, 2014. doi: 10.2118/163608-PA.

[13] H. Hajibeygi, G. Bonfigli, M. A. Hesse, and P. Jenny. Iterative multiscale finite-volume method. *Journal of Computational Physics*, 227(19):8604–8621, 2008. doi: 10.1016/j.jcp.2008.06.013.

[14] H. Hajibeygi and P. Jenny. Multiscale finite-volume method for parabolic problems arising from compressible multiphase flow in porous media. *Journal of Computational Physics*, 228(14):5129–5147, 2009. doi: 10.1016/j.jcp.2009.04.017.

[15] H. Hajibeygi and P. Jenny. Adaptive iterative multiscale finite volume method. *Journal of Computational Physics*, 230(3):628–643, 2011. doi: 10.1016/j.jcp.2010.10.009.

[16] M. A. Hesse, B. T. Mallison, and H. A. Tchelepi. Compact multiscale finite volume method for heterogeneous anisotropic elliptic equations. *Multiscale Modeling & Simulation*, 7(2):934–962, 2008. doi: 10.1137/070705015.

[17] T. Y. Hou and X.-H. Wu. A multiscale finite element method for elliptic problems in composite materials and porous media. *Journal of Computational Physics*, 134:169–189, 1997. doi: 10.1006/jcph.1997.5682.

[18] P. Jenny, S. H. Lee, and H. A. Tchelepi. Multi-scale finite-volume method for elliptic problems in subsurface flow simulation. *Journal of Computational Physics*, 187:47–67, 2003. doi: 10.1016/S0021-9991(03)00075-5.

[19] P. Jenny, S. H. Lee, and H. A. Tchelepi. Adaptive fully implicit multi-scale finite-volume method for multi-phase flow and transport in heterogeneous porous media. *Journal of Computational Physics*, 217(2):627–641, 2006. doi: 10.1016/j.jcp.2006.01.028.

[20] F. Johannessen. Accelerated smoothing and construction of prolongation operators for the multiscale restricted-smoothed basis method on distributed memory systems. Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, 2016. URL http://hdl.handle.net/11250/2433754.

[21] V. Kippe, J. E. Aarnes, and K.-A. Lie. A comparison of multiscale methods for elliptic problems in porous media flow. *Computational Geosciences*, 12(3):377–398, 2008. doi: 10.1007/s10596-007-9074-6.

[22] Ø. S. Klemetsdal, O. Møyner, and K.-A. Lie. Accelerating multiscale simulation of complex geomodels by use of dynamically adapted basis functions. *Computational Geosciences*, 24:459–476, 2020. doi: 10.1007/s10596-019-9827-z.

[23] A. Kozlova, Z. Li, J. R. Natvig, S. Watanabe, Y. Zhou, K. Bratvedt, and S. H. Lee. A real-field multiscale black-oil reservoir simulator. *SPE Journal*, 21(6):2049–2061, 2016. doi: 10.2118/173226-PA.

[24] A. Kozlova, D. Walsh, S. Chittireddy, Z. Li, J. Natvig, S. Watanabe, and K. Bratvedt. A hybrid approach to parallel multiscale reservoir simulator. In *ECMOR XV – 15th European Conference on the Mathematics of Oil Recovery, Amsterdam, The Netherlands, 29 August–1 September 2016*. EAGE. doi: 10.3997/2214-4609.201601889.

[25] 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.

[26] K.-A. Lie, S. Krogstad, I. S. Ligaarden, J. R. Natvig, H. Nilsen, and B. Skaflestad. Open-source MATLAB implementation of consistent discretisations on complex grids. *Computational Geosciences*, 16:297–322, 2012. doi: 10.1007/s10596-011-9244-4.

[27] K.-A. Lie, O. Møyner, J. R. Natvig, A. Kozlova, K. Bratvedt, S. Watanabe, and Z. Li. Successful application of multiscale methods in a real reservoir simulator environment. *Computational Geosciences*, 21(5–6):981–998, 2017. doi: 10.1007/s10596-017-9627-2.

[28] K.-A. Lie, O. Møyner, and J. R. Natvig. Use of multiple multiscale operators to accelerate simulation of complex geomodels. *SPE Journal*, 22(6):1929–1945, 2017. doi: 10.2118/182701-PA.

[29] I. Lunati and P. Jenny. Multiscale finite-volume method for compressible multiphase flow in porous media. *Journal of Computational Physics*, 216(2):616–636, 2006. doi: 10.1016/j.jcp.2006.01.001.

[30] I. Lunati and P. Jenny. A multiscale finite-volume method for three-phase flow influenced by gravity. In P. Binning, P. Engesgaard, H. Dahle, G. Pinder, and W. Gray, eds., *Proceedings of the XVI International Conference on Computational Methods in Water Resources, Copenhagen, Denmark, 18–22 June*. Technical University of Denmark, Kgs. Lyngby, 2006.

[31] I. Lunati and P. Jenny. Treating highly anisotropic subsurface flow with the multiscale finite-volume method. *Multiscale Modeling & Simulation*, 6(1):308–318, 2007. doi: 10.1137/050638928.

[32] I. Lunati and S. H. Lee. An operator formulation of the multiscale finite-volume method with correction function. *Multiscale Modeling & Simulation*, 8(1):96–109, 2009. doi: 10.1137/080742117.

[33] I. Lunati, M. Tyagi, and S. H. Lee. An iterative multiscale finite volume algorithm converging to the exact solution. *Journal of Computational Physics*, 230(5):1849–1864, 2011. doi: 10.1016/j.jcp.2010.11.036.

[34] A. Manea and T. Almani. A multi-level algebraic multiscale solver (ML-AMS) for reservoir simulation. Paper presented at ECMOR XVI – 16th European Conference on the Mathematics of Oil Recovery, Barcelona, Catalonia, Spain. 3–6 September, 2018. doi: 10.3997/2214-4609.201802253.

[35] A. M. Manea and T. Almani. A massively parallel algebraic multiscale solver for reservoir simulation on the GPU architecture. In *SPE Reservoir Simulation*

Conference, 10–11 April, Galveston, Texas, USA. Society of Petroleum Engineers, 2019. doi: 10.2118/193880-MS.

[36] A. M. Manea and H. A. Tchelepi. A massively parallel semicoarsening multigrid linear solver on multi-core and multi-GPU architectures. In *SPE Reservoir Simulation Conference, 20–22 February, Montgomery, Texas, USA*. Society of Petroleum Engineers, 2017. doi: 10.2118/182718-MS.

[37] A. Moncorge, O. Møyner, H. A. Tchelepi, and P. Jenny. Consistent upwinding for sequential fully implicit multiscale compositional simulation. *Computational Geosciences*, 24:533–550, 2020. doi: 10.1007/s10596-019-09835-6.

[38] A. Moncorgé, H. A. Tchelepi, and P. Jenny. Modified sequential fully implicit scheme for compositional flow simulation. *Journal of Computational Physics*, 337:98–115, 2017. doi: 10.1016/j.jcp.2017.02.032.

[39] A. Moncorgé, H. A. Tchelepi, and P. Jenny. Sequential fully implicit formulation for compositional simulation using natural variables. *Journal of Computational Physics*, 371:690–711, 2018. doi: 10.1016/j.jcp.2018.05.048.

[40] O. Møyner. Multiscale finite-volume methods on unstructured grids. Master's thesis, Norwegian University of Science and Technology, Trondheim, 2012. URL http://hdl.handle.net/11250/259015.

[41] O. Møyner. Construction of multiscale preconditioners on stratigraphic grids. Paper presented at *ECMOR XIV – 14th European Conference on the Mathematics of Oil Recovery*, Catania, Italy, 8–11 September, 2014. doi: 10.3997/2214-4609.20141775.

[42] O. Møyner and K.-A. Lie. The multiscale finite volume method on unstructured grids. In *SPE Reservoir Simulation Symposium, The Woodlands, TX, USA, 18–20 February 2013*, Society of Petroleum Engineers, 2013. doi: 10.2118/163649-MS.

[43] O. Møyner and K.-A. Lie. The multiscale finite-volume method on stratigraphic grids. *SPE Journal*, 19(5):816–831, 2014. doi: 10.2118/163649-PA.

[44] O. Møyner and K.-A. Lie. A multiscale restriction-smoothed basis method for compressible black-oil models. *SPE Journal*, 21(6):2079–2096, 2016. doi: 10.2118/173265-PA.

[45] O. Møyner and K.-A. Lie. A multiscale restriction-smoothed basis method for high contrast porous media represented on unstructured grids. *Journal of Computational Physics*, 304:46–71, 2016. doi: 10.1016/j.jcp.2015.10.010.

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

[47] O. Møyner and H. A. Tchelepi. A mass-conservative sequential implicit multiscale method for isothermal equation-of-state compositional problems. *SPE Journal*, 23(6):2376–2393, 2018. doi: 10.2118/182679-PA.

[48] J. R. Natvig, B. Skaflestad, F. Bratvedt, K. Bratvedt, K.-A. Lie, V. Laptev, and S. K. Khataniar. Multiscale mimetic solvers for efficient streamline simulation of fractured reservoirs. *SPE Journal*, 16(4):880–888, 2011. doi: 10.2018/119132-PA.

[49] H. M. Nilsen, A. Moncorgé, K. Bao, O. Møyner, K.-A. Lie, and A. Brodtkorb. Comparison between algebraic multigrid and multilevel multiscale methods for reservoir simulation. *ECMOR XVII – 17th European Conference on the Mathematics of Oil Recovery, 14–17 September 2020, Edinburgh, UK*, 2020. doi: 10.3997/2214-4609.202035063.

[50] J. M. Nordbotten and P. E. Bjørstad. On the relationship between the multiscale finite-volume method and domain decomposition preconditioners. *Computational Geosciences*, 12(3):367–376, 2008. doi: 10.1007/s10596-007-9066-6.

[51] J. M. Nordbotten, E. Keilegavlen, and A. Sandvin. Mass conservative domain decomposition for porous media flow. In R. Petrova, ed., *Finite Volume Method–Powerful Means of Engineering Design*, pp. 235–256. InTech Europe, Rijeka, Croatia, 2012. doi: 10.5772/38700.

[52] K. Stüben. A review of algebraic multigrid. *Journal of Computational and Applied Mathematics*, 128(1):281–309, 2001. doi: 10.1016/S0377-0427(00)00516-1.

[53] J. A. Trangenstein and J. B. Bell. Mathematical structure of compositional reservoir simulation. *SIAM Journal of Scientific and Statistical Compution*, 10(5):817–845, 1989. doi: 10.1137/0910049.

[54] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, London, 2000.

[55] J. R. Wallis, R. P. Kendall, and T. E. Little. Constrained residual acceleration of conjugate residual methods. In *SPE Reservoir Simulation Symposium, 10–13 February, Dallas, Texas*. Society of Petroleum Engineers, 1985. doi: 10.2118/13536-MS.

[56] Y. Wang, H. Hajibeygi, and H. A. Tchelepi. Algebraic multiscale solver for flow in heterogeneous porous media. *Journal of Computational Physics*, 259:284–303, 2014. doi: 10.1016/j.jcp.2013.11.024.

[57] Y. Wang, H. Hajibeygi, and H. A. Tchelepi. Monotone multiscale finite volume method. *Computational Geosciences*, 20(3):509–524, 2016. doi: 10.1007/s10596-015-9506-7.

[58] J. W. Watts. A compositional formulation of the pressure and saturation equations. *SPE Reservoir Engineering*, 1(3):243–252, 1986. doi: 10.2118/12244-PA.

[59] H. Zhou and H. A. Tchelepi. Operator-based multiscale method for compressible flow. *SPE Journal*, 13(2):267–273, 2008. doi: 10.2118/106254-PA.

[60] H. Zhou and H. A. Tchelepi. Two-stage algebraic multiscale linear solver for highly heterogeneous reservoir models. *SPE Journal*, 17(2):523–539, 2012. doi: 10.2118/141473-PA.