Article, Published Version

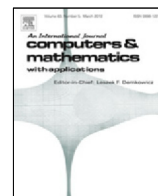**Koch, Timo; Gläser, Dennis; Weishaupt, Kilian; et al.**

DuMux 3 – an open-source simulator for solving flow and transport problems in porous media with a focus on model coupling

Computers & Mathematics with Applications

DuMu^x 3 – an open-source simulator for solving flow and transport problems in porous media with a focus on model coupling

Timo Koch [a],[*], Dennis Gläser [a], Kilian Weishaupt [a], Sina Ackermann [a],
Martin Beck [a], Beatrix Becker [a], Samuel Burbulla [b], Holger Class [a],
Edward Coltman [a], Simon Emmert [a], Thomas Fetzer [a], Christoph Grüninger [a],
Katharina Heck [a], Johannes Hommel [a], Theresa Kurz [a], Melanie Lipp [a],
Farid Mohammadi [a], Samuel Scherrer [c], Martin Schneider [a], Gabriele Seitz [a],
Leopold Stadler [d], Martin Utz [d], Felix Weinhardt [a], Bernd Flemisch [a]

[a] *Department of Hydromechanics and Modelling of Hydrosystems, Institute for Modelling Hydraulic and Environmental Systems, University of Stuttgart, 70569 Stuttgart, Germany*
[b] *Chair of Applied Mathematics, Institute of Applied Analysis and Numerical Simulation, University of Stuttgart, 70569 Stuttgart, Germany*
[c] *VEGAS - Research Facility for Subsurface Remediation, Institute for Modelling Hydraulic and Environmental Systems, University of Stuttgart, 70569 Stuttgart, Germany*
[d] *BAW - Federal Waterways Engineering and Research Institute, 76187 Karlsruhe, Germany*

ARTICLE INFO

ABSTRACT

We present version 3 of the open-source simulator for flow and transport processes in porous media DuMu^x. DuMu^x is based on the modular C++ framework DUNE (Distributed and Unified Numerics Environment) and is developed as a research code with a focus on modularity and reusability. We describe recent efforts in improving the transparency and efficiency of the development process and community-building, as well as efforts towards quality assurance and reproducible research. In addition to a major redesign of many simulation components in order to facilitate setting up complex simulations in DuMu^x, version 3 introduces a more consistent abstraction of finite volume schemes. Finally, the new framework for multi-domain simulations is described, and three numerical examples demonstrate its flexibility.

1. Introduction

DuMu^x, DUNE for multi-{phase, component, scale, physics, domain, …} flow and transport in porous media, is a free and open-source simulator for flow and transport processes in porous media [1] (dumux.org). Its main intention is to provide a sustainable, consistent and modular framework for the implementation and application of porous media model concepts and constitutive relations. It has been successfully applied to greenhouse gas and CO_2 storage [2–5], radioactive waste disposal [6], gas migration problems [7], environmental remediation problems [8], flow through fractured porous media [9–14], reactive transport in porous media [15], modeling of biofilms and mineral precipitation [16,17], modeling

---

root–soil interaction [18,19], modeling of fuel cells [20], pore-network modeling [21], transport of therapeutic agents through biological tissue [18,22–24], and subsurface–atmosphere coupling [25,26]. For a more comprehensive list of publications that have been achieved with the help of DuMu^x, see dumux.org/publications.

DuMu^x is based on the Distributed Unified Numerics Environment (DUNE) [27–30], an open-source scientific numerical software framework for solving partial differential equations, and is thus part of a larger community that goes beyond the simulation of fluid–mechanical processes in porous media. DUNE and DuMu^x are written in C++, using modern C++ programming techniques and C++ template meta programming for efficiency and generic interfaces. The DUNE core modules provide, among other things, multiple grid managers implementing a versatile common grid interface, linear algebra abstraction and an iterative solver back-end [31], as well as abstractions facilitating parallel computing. DuMu^x is designed and developed as a DUNE module depending on the DUNE core modules and optionally interacts with a number of other DUNE-based extension modules. The key features of DuMu^x are its comprehensive library of multi-phase and multi-component flow and transport models, the flexible and modular fluid and material framework for constitutive relations [1], abstractions for finite volume discretization schemes (see Section 3), and its focus on model coupling (see Section 4).

There are many other open-source projects focusing on Darcy-scale porous-medium flow and transport processes. A list has been compiled in [32], and most references are given here again for the sake of completeness: MODFLOW, water.usgs.gov/ogw/modflow/ [33], MRST, sintef.no/projectweb/mrst/ [34], OpenGeoSys, opengeosys.org [35], OPM, opm-project.org [36,37], ParFlow, parflow.org [38], PFloTran, pflotran.org [39], PorePy, github.com/pmgbergen/porepy [40]. Furthermore, there are open-source numerical software frameworks with a broader focus such as deal.II, dealii.org [41], DUNE, dune-project.org [27,28], Feel++, feelpp.org [42], FEniCS, fenicsproject.org [43], MOOSE, mooseframework.org [44], or OpenCMISS, opencmiss.org [45].

This paper can be seen as a continuation of [1] which describes the code and the project at the beginning of the 2.X release series. In the remainder of this introductory section, we provide a brief chronology of the development of DuMu^x and some information on activities and measures that go beyond the development of the mainline code base. Section 2 introduces the general structure of the code and its design principles. Section 3 goes into more details by explaining abstractions and software concepts for general finite volume schemes encountered in DuMu^x. Section 4 is devoted to simulations coupling two or more computational domains and models. In Section 5, we briefly address new features in the new version DuMu^x 3. Numerical examples that particularly highlight the strengths of the new multi-domain framework are presented in Section 6. Finally, Section 7 discusses current limitations and perspectives.

1.1. History

The following section briefly outlines the DuMu^x project history and its development. More details can be found in [32].

The development of DuMu^x started at the Department of Hydromechanics and Modelling of Hydrosystems at the University of Stuttgart (LH2) in January 2007. The software's main goal is to facilitate research at the LH2, and provide a framework to maintain and continuously develop research methods and results. After an initial development phase, DuMu^x 1.0 was released under the GNU General Public License Version 2 (or later) [46] in July 2009.

Since DuMu^x 1.0 exhibited several shortcomings regarding generality, modularity and usability, the code base was refactored, leading to the release of DuMu^x 2.0 in February 2011 and an accompanying publication [1]. Since the beginning of the 2.X release series, more emphasis was put on the sustainability of the code base and a streamlined release process. In particular, two consecutive minor releases are to be backwards compatible in the sense that any code depending on the previous minor release ought to work unchanged. However, compilation might emit deprecation warnings resulting from interface changes. Since October 2013, DuMu^x 2.X has been semiannually released (minor releases) until release 2.12 in December 2017. Since release 2.7, Zenodo (zenodo.org) is employed to enable the citation of a specific release by providing a digital object identifier (DOI) for each release tarball; see for example [47].

In the 2.X release series, many improvements and enhancements, ranging from new modeling capabilities to additional discretization schemes were introduced. Not all new features were (or could be) implemented in consistence with the original design, leading to code which was inefficient and increasingly difficult to maintain. In November 2016, a small team started to refactor and enhance the code base in a separate development line. Some of these changes are described in detail in Section 2. Due to the vast amount of fundamental changes, maintaining backwards compatibility became infeasible. Therefore, a new major release cycle was initiated.

A preliminary DuMu^x version 3.0-alpha was released in December 2017 for users and developers to test and explore the new structure and design. The new structure and several design considerations are described in Sections 2–5. Further development, under consideration of the feedback from the DuMu^x community, lead to the release of DuMu^x 3.0 in December 2018. In October 2019, DuMu^x 3.1 was released, resuming a new cycle of semiannual minor releases.

1.2. DuMu^x as a framework

The main DuMu^x Git repository is developed and hosted at (git.iws.uni-stuttgart.de) and publicly accessible through a GitLab server frontend. In the following, we address aspects of the DuMu^x research software that go beyond the development of the main code base. After discussing quality assurance and reproducibility, we introduce the two modules

dumux-course and dumux-lecture. More details about the development of DuMu^x and community building are given in [32].

DuMu^x is related to the Open Porous Media (OPM) initiative [37] which strives to develop software for "modeling industrially and scientifically relevant flow and transport processes in porous media" [48]. In particular, the black-oil simulator Flow is a main product of OPM and is built on top of the OPM modules opm-material and opm-models (formerly ewoms). Both modules result from a fork of DuMu^x 2.2. DuMu^x can be used together with the OPM module opm-grid which enables DuMu^x the use of corner-point grids—the de facto standard in the petroleum industry.

Quality assurance and reproducibility. Automated testing is indispensable for the sustained development of software [49]. Currently, there exist more than 400 unit and system tests for DuMu^x. The test suite is built and executed by a BuildBot CI server at git.iws.uni-stuttgart.de/buildbot in an automated way, triggered by each commit to the master branch of the DuMu^x Git repository. We regularly assess that the tests cover a large number of lines of code and work towards increasing the coverage.[1] To ensure the quality of newly developed capabilities and features, every corresponding addition to the code base is to be accompanied by a test and documented appropriately. Moreover, each code change is reviewed by at least one other developer in a transparent and public review process using GitLab.

In addition to assuring quality of the code base, research results published with the help of DuMu^x should be reproducible. To this end, the project DuMu^x-Pub was initiated in 2015. It provides the capabilities to archive and document code used in scientific publications. A few simple tools automate the process from a DuMu^x application to a new light-weight Dune module containing the source and data files required to reproduce the research results. Furthermore, the required versions of all dependencies are automatically determined and documented. At the LH2 department, all relevant scientific articles, as well as bachelor's, master's and doctoral theses, are accompanied by a DuMu^x-Pub module hosted at git.iws.uni-stuttgart.de/dumux-pub [32]. As part of the ongoing project "Sustainable infrastructure for the improved usability and archivability of research software on the example of the porous-media-simulator DuMu^x" (SusI),[2] these existing measures will be extended by automating the generation of Docker containers for DuMu^x-Pub modules and the creation of web applications in order to easily reproduce the research results.

DuMu^x course. The dumux-course module has been created for a DuMu^x course offered 2018 in Stuttgart. It is continuously improved and enhanced along with the mainline development. While the module is used in conventional DuMu^x courses with participants attending personally [32], it is also designed to be used independently by everyone who would like to learn how to work with DuMu^x. All course exercises are documented and contain task descriptions and solutions. Their contents range from a very basic first building and running experience to fairly advanced applications based on coupled model equations. The module also contains the slides from the course providing broader context and background information.

DuMu^x lecture. The LH2 department offers Master level courses for different study programs (Environmental and Civil Engineering, Water Resources Engineering and Management, Simulation Technology, or Computational Mechanics of Materials and Structures), where computer exercises using DuMu^x are an essential component. The module dumux-lecture contains all example applications used in these lectures, with most of the applications belonging to the lecture "Multiphase Modeling". Each application has its own folder containing problem setup and spatial parameter specifications, as well as an input file where typically those runtime parameters can be specified that are of educational value for the problem. Each example is accompanied by explanations and tasks.

2. Structure and design principles

DuMu^x is designed as a research code framework. Foremost, this means that it is designed with an emphasis on modularity. DuMu^x user code is usually developed in a separate Dune module that lists DuMu^x as its dependency. To maintain flexibility for a user, DuMu^x follows the principle that all components of a simulation should be easily replaceable with a new implementation, without modifying code in the DuMu^x module itself. For example, it is simple to change from one of the implemented capillary–pressure relationships to a custom function, to modify the numerical flux computation, and to change from a cell-centered to a vertex-centered finite volume discretization. DuMu^x profits from the modular design of Dune. For example, an unstructured grid implementation can be exchanged for an efficient structured grid implementation by changing a single line in the user code.

To maintain computational efficiency, much of the modularity is realized through the use of C++ templates and generic programming techniques such as traits and policies. Moreover, the developers of DuMu^x try to follow well-known object-oriented design principles and the separation of algorithms and data structures, as for example in the design of the DuMu^x material framework [50]. DuMu^x 3, which requires a C++14-compatible compiler, increasingly makes use of modern C++ features such as lambdas, type deduction, or smart pointers, to the benefit of usability, modularity, and efficiency.

In the DuMu^x environment, the term *model* is used to describe a system of coupled partial differential equations (PDEs) including constitutive equations needed for closure. Many models, in particular PDEs describing general non-isothermal

---

[1] A detailed weekly coverage report created with gcov, gcovr and GitLab CI is publicly available at pages.iws.uni-stuttgart.de/dumux-repositories/dumux-coverage/.

[2] Funded by the German Research Foundation (DFG).

multi-component multi-phase flow processes in porous media, are already implemented in DuMu^x. Furthermore, a user can choose from a variety of constitutive laws to describe closure relations, and multiple fluid and solid systems and components. More importantly, the code design also facilitates using custom implementations.

The main components of a DuMu^x simulation are represented in the code by corresponding C++ classes. When using one of the implemented models out-of-the-box, a user usually implements at least two such classes in addition to the program's `main` function. The `Problem` class defines boundary conditions, initial conditions (if necessary), and volumetric source terms (if any), by implementing a defined class interface. The `SpatialParams` class defines the spatial distribution of material parameters, such as porosity, permeability, or for example, parameters for the van Genuchten water retention model (if applicable). The `Problem` class stores a pointer to an instance of the `SpatialParams` class. Moreover, each simulation currently creates at least one tag (a C++ struct) which is used to specify several compile time options (*properties*) of the simulation by means of partial template specialization. Code Example 1 shows how to set properties for a newly defined tag. Properties can be extracted as traits of such a tag in other parts of the code. Depending on the chosen model, more of such properties have to be defined and examples are provided for all existing models in DuMu^x. User defined models can provide their own properties.

```
1   namespace Dumux {
2   namespace Properties {
3
4   // create a new type tag
5   namespace TTag {
6   struct Example { using InheritsFrom = std::tuple<OneP, CCTpfaModel>; };
7   } // end namespace TTag
8
9   // set the Dune grid type
10  template<class Tag> struct Grid<Tag, TTag::Example>
11  { using type = Dune::UGGrid<3>; };
12
13  // set the problem type
14  template<class Tag> struct Problem<Tag, TTag::Example>
15  { using type = UserProblem; };
16
17  // set the spatial parameters type
18  template<class Tag>
19  struct SpatialParams<Tag, TTag::Example>
20  { using type = UserSpatialParams; };
21
22  // set the fluid system type
23  template<class Tag> struct FluidSystem<Tag, TTag::Example>
24  { using type = FluidSystems::OnePLiquid<double, Components::H2O<double> >; };
25
26  } // end namespace Properties
27  } // end namespace Dumux
```

Code Example 1: An example of a property setting for DuMu^x 3. The simulation is configured to use a one-phase porous medium model (`OneP`), a cell-centered two-point-flux-approximation discretization scheme (`CCTpfaModel`), and liquid water as a fluid. The `UserProblem` and `UserSpatialParams` types are implementation- and user-defined. Header includes are omitted for brevity. The types attached to the `Example` tag, can be extracted as traits, see `GetPropType` in Code Example 2.

A rather significant improvement in DuMu^x 3 concerns the way the program's `main` function is written. In DuMu^x 2, the `main` function called a predefined `start` function that determined the program flow. Modifications to the program flow were made possible by numerous hook functions. For example, the `Problem` base class implemented a hook function `postTimeStep` that could be overloaded by the user's `Problem` implementation, injecting the notion of time and program flow into the `Problem` class; a hook function `addVtkOutputFields` injected a notion of file I/O. This design concept introduced many dependencies between classes and (as shown here for the `Problem` class) lead to the violation of the well-known single responsibility principle, hindering modular design. Furthermore, due to the concept of hook functions, it was very difficult to write simulations outside the predefined program flow in `start`. For example, all simulations were considered transient and non-linear. Consequently, solving stationary equations was often realized by solving a single time step and making sure that the assembled storage term is zero. Solving linear equations was realized by executing a single iteration of a Newton method. On the positive side, a user had the possibility to realize most changes within a single header file.

In contrast, the program's `main` function in DuMu^x 3 contains all of the main steps of the simulation, see Code Example 2. Therefore, to solve a stationary PDE, the `TimeLoop` instance (l. 41) and the time loop (l. 53ff) can simply be removed from the program flow. Similarly, the `NewtonSolver` class can be replaced by a `LinearPDESolver` to solve a linear PDE. Finally, the I/O logic and other modifications to the main program flow can be directly written inside the `main` function. As a result, the main program flow becomes arguably more transparent, which has been confirmed by feedback from the DuMu^x user community. For more information on the steps of a DuMu^x simulation, we refer the interested reader to the DuMu^x handbook [51].

```
1   int main(int argc, char** argv)
2   {
3     using namespace Dumux;
4
5     // an example tag from which multiple traits
6     // can be extracted (see GetPropType below)
7     using Tag = Properties::TTag::Example;
8
9     // initialize MPI (enables parallel runs)
10    Dune::MPIHelper::instance(argc, argv);
11
12    // parse command line arguments and a parameter file
13    Parameters::init(argc, argv, "params.input");
14
15    // create a Dune grid (from information in the parameter file)
16    using Grid = GetPropType<Tag, Properties::Grid>;
17    GridManager<Grid> gridManager; gridManager.init();
18
19    // create the finite volume grid geometry (see Section 3)
20    const auto leafGridView = gridManager.grid().leafGridView();
21    using GridGeometry = GetPropType<Tag, Properties::FVGridGeometry>;
22    auto fvGridGeometry = std::make_shared<GridGeometry>(leafGridView);
23    fvGridGeometry->update();
24
25    // the problem (initial conditions, boundary conditions, sources)
26    using Problem = GetPropType<Tag, Properties::Problem>;
27    auto problem = std::make_shared<Problem>(fvGridGeometry);
28
29    // the solution vector, set to the initial solution
30    using SolutionVector = GetPropType<Tag, Properties::SolutionVector>;
31    SolutionVector x; problem->applyInitialSolution(x);
32    auto xOld = x;
33
34    // the grid variables (secondary variables on control volumes and faces)
35    using GridVariables = GetPropType<Tag, Properties::GridVariables>;
36    auto gridVariables = std::make_shared<GridVariables>(problem, fvGridGeometry);
37    gridVariables->init(x);
38
39    // get some time loop parameters and instantiate time loop
40    const auto tEnd = getParam<double>("TimeLoop.TEnd");
41    auto dt = getParam<double>("TimeLoop.DtInitial");
42    auto timeLoop = std::make_shared<TimeLoop<double>>(0, dt, tEnd);
43
44    // the assembler with time loop for transient problem & the solver
45    using Assembler = FVAssembler<Tag, DiffMethod::numeric>;
46    auto assembler = std::make_shared<Assembler>(problem, fvGridGeometry, gridVariables, timeLoop);
47    assembler->setPreviousSolution(xOld);
48    Dumux::NewtonSolver<Assembler, LinearSolver> nonLinearSolver(assembler, linearSolver);
49
50    // initialize output module
51    ...
52
53    // time loop
54    timeLoop->start(); do
55    {
56      // solve the non-linear system with time step control
57      nonLinearSolver.solve(x, *timeLoop);
58
59      // make the new solution the old solution
60      xOld = x; gridVariables->advanceTimeStep();
61
62      // advance the time loop to the next step
63      timeLoop->advanceTimeStep();
64
65      // report statistics of this time step
66      timeLoop->reportTimeStep();
67
68      // write output, set new time step size
69      ...
70
71    } while (!timeLoop->finished());
72
73    return 0;
74  }
```

Code Example 2: An example of a main function using DuMu$^{\text{x}}$ 3 for solving a transient non-linear problem with a finite volume scheme and numeric differentiation. The simulation can be run in parallel (MPI, distributed memory). Header includes are omitted for brevity.

3. Abstractions and concepts for general finite volume schemes

One of the most important abstractions in DuMu$^\text{x}$ is the *grid geometry*. The concept partly existed in DuMu$^\text{x}$ 2 but was significantly redesigned and has been casted into an object-oriented representation in DuMu$^\text{x}$ 3. A grid geometry is a wrapper around a *grid view* on a Dune *grid* instance. Dune grids are general hierarchical grids and grid views provide read-only access to certain parts of the grid. In particular, a *leaf grid view* is a view on all grid entities without descendants in the hierarchy (which are not refined), thus covering the whole domain (in sequential simulations) or the part of the grid assigned to a single processor (in parallel simulations), while a *level grid view* is a view on all entities of a given level of the refinement hierarchy. The grid geometry constructs, from such a grid view, all the geometrical and topological data necessary to evaluate the discrete equations resulting from a given finite volume scheme. This abstraction allows us to implement many different finite volume schemes in a unified way. In the following sections, we will present and motivate the mathematical abstractions behind the grid geometry concept, describe their realizations in the code in form of C++ classes and provide an exemplary code snippet that illustrates how to use them.

3.1. Finite volume discretization

Let us consider a domain $\Omega \subset \mathbb{R}^n$, $1 \leq n \leq 3$, with boundary $\partial\Omega$, which is further decomposed into two subsets on which Dirichlet and Neumann boundaries are specified, that is $\partial\Omega = \partial\Omega_\text{D} \cup \partial\Omega_\text{N}$. Furthermore, let us consider a general conservation equation,

$$\frac{\partial m}{\partial t} + \nabla \cdot \boldsymbol{\psi} = q, \qquad\qquad\qquad \text{in } \Omega, \tag{3.1a}$$

$$\boldsymbol{\psi} \cdot \mathbf{n} = g_\text{N}, \qquad\qquad\qquad \text{on } \partial\Omega_\text{N}, \tag{3.1b}$$

$$u = u_\text{D}, \qquad\qquad\qquad \text{on } \partial\Omega_\text{D}, \tag{3.1c}$$

$$u(t = 0) = u_0 \qquad\qquad\qquad \text{in } \Omega, \tag{3.1d}$$

where $m = m(u)$ is the conserved quantity depending on the primary variable u, $\boldsymbol{\psi} = \boldsymbol{\psi}(u)$ is a flux term and $q = q(u)$ a source term. All terms may non-linearly depend on u. Eqs. (3.1a)–(3.1d) constitute a non-linear problem in u with initial and boundary conditions.

We introduce the primary grid $\mathcal{M}$ with elements $E \in \mathcal{M}$, such that $\Omega_h = \bigcup_{E \in \mathcal{M}} E$ is a discrete approximation of Ω. Furthermore, we introduce a tessellation $\mathcal{T}$ of Ω, such that $\Omega_h = \bigcup_{K \in \mathcal{T}} K$, where each $K \in \mathcal{T}$ is a control volume with measure $|K| > 0$. The control volumes K do not necessarily need to coincide with the elements E. However, for cell-centered finite volume schemes, usually $\mathcal{T} \equiv \mathcal{M}$. Each control volume is partitioned into sub-control-volumes κ, where $\kappa = K$ is one admissible partition. Moreover, the boundary of the control volume ∂K consists of a finite number of faces $f \subset \partial K$ which are either inner faces $f_I = \partial K \cap \partial L$, where $L \in \mathcal{T}$ is an adjacent control volume, or boundary faces $f_B = \partial K \cap \partial \Omega_h$. Depending on the discretization method, it may be useful to partition the faces f into sub-(control-volume)-faces σ. We denote by $\mathcal{E}_K$ the entire set of such sub-faces on ∂K, forming a disjoint partition such that $\partial K = \bigcup_{\sigma \in \mathcal{E}_K} \sigma$. Accordingly, $\mathcal{T}_K$ denotes the set of sub-control volumes embedded in a control volume K with $K = \bigcup_{\kappa \in \mathcal{T}_K} \kappa$.

Integration of Eq. (3.1a) over a control volume and application of the divergence theorem yields

$$\int_K \frac{\partial m}{\partial t}\,\mathrm{d}x + \sum_{f \subset \partial K} \int_f \boldsymbol{\psi} \cdot \mathbf{n}\,\mathrm{d}\Gamma = \int_K q\,\mathrm{d}x, \tag{3.2}$$

where $\mathbf{n}$ is a unit outer normal vector on ∂K. Using the concept of sub-control-volumes and sub-control-volume-faces, allows to reformulate Eq. (3.2) as

$$\sum_{\kappa \in \mathcal{T}_K} \int_\kappa \frac{\partial m}{\partial t}\,\mathrm{d}x + \sum_{\sigma \in \mathcal{E}_K} \int_\sigma \boldsymbol{\psi} \cdot \mathbf{n}\,\mathrm{d}\Gamma = \sum_{\kappa \in \mathcal{T}_K} \int_\kappa q\,\mathrm{d}x. \tag{3.3}$$

Let us now replace the exact flux over sub-face σ by the approximation $F_{K,\sigma} \approx \int_\sigma \boldsymbol{\psi} \cdot \mathbf{n}\,\mathrm{d}\Gamma$ and approximate the volume integrals in Eq. (3.3) to arrive at the discrete, control-volume-local formulation of Eq. (3.1a),

$$\sum_{\kappa \in \mathcal{T}_K} \frac{m_\kappa^{t+\Delta t} - m_\kappa^t}{\Delta t} + \sum_{\sigma \in \mathcal{E}_K} F_{K,\sigma} - \sum_{\kappa \in \mathcal{T}_K} q_\kappa = 0, \tag{3.4}$$
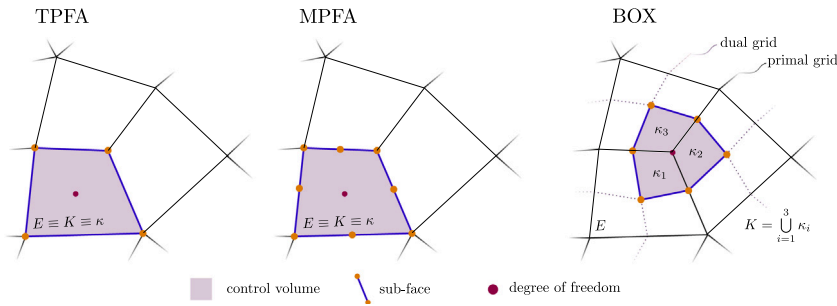
Fig. 1. Illustration of the configurations of control volumes and sub-faces on a computational grid for different discretization schemes. The TPFA and MPFA scheme use the grid elements as control volumes ($E \equiv K \equiv \kappa$), while in the BOX scheme the control volume is constructed around the grid vertices by connecting the barycenters of all adjacent geometrical entities. The partition of the boundary ∂K by the sub-faces σ is illustrated by depicting the corners of each sub-face.

where the approximate integrals m_κ and q_κ depend on the specific finite volume method. The time derivative is approximated by a backward or forward finite difference,[3] determining the time level t or $t + \Delta t$ at which the flux and source terms $F_{K,\sigma}$ and q are evaluated. The expressions for the discrete fluxes $F_{K,\sigma}$ depend on the actual underlying finite volume scheme. Fore more detailed information, we refer to [52,53] for the BOX scheme, and [54] for two-point and multi-point-flux-approximation finite volume schemes (TPFA and MPFA).

If a backward Euler time discretization is chosen, the nonlinear system corresponding to Eq. (3.4) on each control volume K is solved by Newton's method. In each Newton step n, we compute a linearized discrete PDE system in residual form

$$A(u_n)\Delta u = A(u_n)(u_n - u_{n+1}) = r(u_n), \tag{3.5}$$

where u now is the vector of primary variables for each degree of freedom, and $A = \frac{\partial r}{\partial u}$ is the Jacobian of the residual r, i.e. the discrete equation evaluated at u_n. The residual for a control volume $K \in \mathcal{T}$ is given by the left-hand-side of Eq. (3.4). In the default configuration, the derivatives in the Jacobian matrix are approximated by numerical differentiation in DuMu^x.

3.2. Element-wise assembly

DUNE grids and the connectivity information provided by the grid are rather element-centered (elements are grid entities of co-dimension 0). For this reason, it is natural in DUNE to choose an element-wise assembly of the residual. The choice of $\mathcal{T}$ and hence the sub-control-volumes κ and sub-control-volume-faces σ is motivated by the goal of a convenient implementation of an element-wise assembly for a given control volume scheme. Fig. 1 depicts the configurations of control volumes and sub-faces as chosen in DuMu^x, exemplarily for the TPFA, MPFA and the BOX scheme. A corresponding illustration for the MAC scheme can be found, for example, in [55]. As illustrated in Fig. 1, the sub-control-volumes for the box scheme are chosen such that the residual associated with a degree of freedom on a vertex can be split into contributions from all neighboring elements. For the MPFA-O scheme, the sub-faces are defined such that each sub-face can be associated to a grid vertex. This is motivated by the fact that the expressions for the discrete fluxes across the sub-faces depend on the degrees of freedom within so-called interaction regions, constructed around the grid vertices [56].

3.3. Representation in the software implementation

Sub-control-volumes κ and sub-control-volume-faces σ, as introduced in Section 3.1, are represented in the code by objects of the classes `SubControlVolume` and `SubControlVolumeFace`. Depending on the caching model, which can be selected by specifying a template argument, these objects are either stored in the `GridGeometry` instance, or constructed on-the-fly on a per-element basis during the assembly. Furthermore, DuMu^x has the notion of a *local view* on the grid geometry. The local view is an element-centered view that gives access to all κ and σ on an element, as well as all connectivity and geometry information to assemble storage, flux, and source terms. For example, for a TPFA scheme, the neighboring control volumes and associated primary and secondary variables have to be accessed to calculate the fluxes over the control-volume interfaces. Code Example 3 shows how to assemble the total mass contained in a domain for a given grid $\mathcal{M}$. The presented code is identical for all introduced finite volume schemes. Similar to the range-based-for syntax to iterate over all elements of a grid view provided by DUNE, the local view on the grid geometry in DuMu^x makes it possible to iterate over all sub-control-volumes associated with an element in a concise and readable way (l. 17). Analogously, a range-generator for iterating over all sub-control-volume-faces associated with an element is provided.

---

[3] DuMu^x currently only implements the backward and forward Euler time discretization schemes.
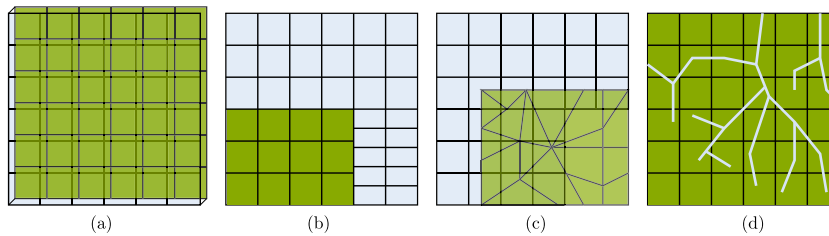
Fig. 2. Different types of model coupling in DuMu^x multi-domain simulations. (a) multi-physics models (on the same grid), (b) multiple non-overlapping domains with sharp conforming or non-conforming interface, (c) multiple overlapping domains with different discretizations, (d) conforming and non-conforming (embedded) mixed-dimensional domains (1D–2D, 1D–3D, 2D–3D). The different coupling modes can also be combined. Typical mixed-dimensional simulations also solve multi-physics problems, or use different discretization schemes in the subdomains. The number of subdomains is not limited to two.

```
1   const double porosity = 0.4;
2   const double density = 1000.0;
3   double totalWaterMass = 0.0;
4
5   // iterate over all elements of the leaf grid view of the grid
6   for (const auto& element : elements(leafGridView))
7   {
8       // construct a local view on the grid geometry
9       auto fvGeometry = localView(*fvGridGeometry);
10
11      // the view has to be bound to the current element
12      // this is a no-op if the geometry is cached
13      fvGeometry.bind(element);
14
15      // iterate over all sub-control-volumes in the local view
16      // and accumulate the mass
17      for (const auto& scv : scvs(fvGeometry))
18          totalWaterMass += saturation[scv.dofIndex()] * porosity
19                          * density * scv.volume();
20  }
21
22  std::cout << "The total water mass is " << totalWaterMass << " kg/s" << std::endl;
```

Code Example 3: The code to compute the total water mass in the domain Ω_h given a discrete water saturation field with values for each degree of freedom, stored in the vector `saturation`. The `leafGridView` variable is introduced in Code Example 2.

Using the presented abstraction, the following discretization schemes have been implemented in DuMu^x: TPFA, MPFA-O, MPFA-L, MAC (staggered grid), finite volumes with non-linear two-point and multi-point flux approximation (NL-TPFA, NL-MPFA), and mimetic finite differences (with additional degrees of freedom on the grid faces) [57,58], from which TPFA, BOX, MPFA-O, and the MAC scheme are available in the latest DuMu^x version.

4. Multi-domain simulations

Model coupling is one of the key features in DuMu^x [18,59] and has been a driving force in the development of DuMu^x 3, which features a new, consistent framework for implementing coupled models, that is, models composed of multiple coupled sub-models. Fig. 2 shows different types of model coupling that can be realized in this framework. In this context, the term *multi-domain simulation* covers a broad class of simulations featuring model coupling. The idea is to solve a coupled PDE system, of which a subset (a subdomain model) can be formulated on a different domain, potentially with different dimensionality, and may be discretized using different meshes or different spatial discretization schemes. The framework is designed to minimize software coupling, such that two existing DuMu^x models can be coupled without modifications in the core components. To be minimally invasive, only the `Problem` class is required to be slightly modified to become a *sub-problem*. Sub-problems define their own initial and boundary conditions, and source term. However, boundary and source terms may depend on quantities of other domains. For the data transfer, we introduce the concept of a *coupling manager* which is represented in the code by classes that implement the `CouplingManager` interface. A coupling manager has to provide the information which degrees of freedom in one domain are coupled to which degrees of freedom in another domain (the *coupling stencil*). To this end, every coupling manager has to implement the function shown in Code Example 4.

```
1   // required member function of every CouplingManager class
2   // return the coupling stencil for element elementI (domain i)
```

```
3    // with respect to degrees of freedom in domain j
4    template<std::size_t i, std::size_t j>
5    const std::vector<std::size_t>&
6    couplingStencil(Dune::index_constant<i> domainI,
7                    const Element<i>& elementI,
8                    Dune::index_constant<j> domainJ) const;
```

Code Example 4: The member function `couplingStencil` has to be implemented by all deriving coupling manager implementations. The template arguments `i` and `j` are the indices of one pair of coupled subdomains. They are deduced from the two index objects `domainI` and `domainJ`, passed as arguments to the function. In case two subdomain are not coupled, the function is required to return a reference to an empty stencil vector. For domain `i`, an instance of the element is passed, for which the residual in the element-wise assembly is to be computed. The function returns a vector of all indices of degrees of freedom coupled to one of the degrees of freedom associated with element `elementI`.

Moreover, the coupling manager has to transfer data between the sub-problems. This general concept can be used to implement a wide class of coupling schemes. Finding the coupling stencils often involves intersecting two grids. For this purpose, DuMu^x provides efficient intersection algorithms based on axis-aligned bounding box volume hierarchy data structures [60,61] computed for DUNE grids. Code Example 5 shows how to intersect two given grid geometries and use the connectivity information to construct a coupling stencil, that can be used, for example, in a non-conforming embedded fracture model.[4] The grid intersection and stencil computation is illustrated in Fig. 3.

```
1
2    // which degrees of freedom of the other domain does the element residual depend on?
3    using Stencils = std::vector<std::set<std::size_t>>;
4    Stencils lowDimCouplingStencils, bulkCouplingStencils;
5    // resize to the number of grid elements
6    lowDimCouplingStencils.resize(lowDimGridGeometry.gridView().size(/*codim=*/0));
7    bulkCouplingStencils.resize(bulkGridGeometry.gridView().size(/*codim=*/0));
8
9    // intersect two grid geometries
10   // we choose domain (first argument): lowDim, target (second argument): bulk
11   // see <dumux/multidomain/glue.hh>
12   const auto glue = makeGlue(lowDimGridGeometry, bulkGridGeometry);
13
14   // interate over all intersections
15   for (const auto& is : intersections(glue))
16   {
17       // the element index of the lowDim element of this intersection
18       const auto domainIdx = lowDimGridGeometry.elementMapper().index(is.domainEntity());
19       // there might be multiple bulk elements associated with this intersection
20       for (unsigned int i = 0; i < is.numTargetNeighbors(); ++i)
21       {
22           // the element index of the bulk element of this intersection
23           const auto targetIdx = bulkGridGeometry.elementMapper().index(is.targetEntity(i));
24           // insert target-domain index pair into the respective stencil
25           lowDimCouplingStencils[domainIdx].insert(targetIdx);
26           bulkCouplingStencils[targetIdx].insert(domainIdx);
27       }
28   }
```

Code Example 5: Sample code which intersects two grid geometries (see Section 3.1) and uses the resulting connectivity information to construct coupling stencils for a cell-centered finite volume scheme. A bulk grid (`bulk`) is intersected with an overlapping grid with lower dimension (`lowDim`) which may discretize a three-dimensional rock domain and a network of two-dimensional fractures. (The code snippet is identical for 3D–2D and 2D–1D.) An intersection is $\Gamma = E_b \cap E_l$, where E_b (`targetEntity`) is an element of the bulk grid and E_l (`domainEntity`) an element of the lower-dimensional embedded grid. If there exist multiple such intersections with identical geometry (e.g. a fracture element coincides with a face shared by two bulk elements), they are represented in the code by a single intersection object with access to all elements ("entities") associated with the intersection ("neighbors"). The interface of the `glue` object is similar to that implemented in the DUNE module `dune-grid-glue` [62] which implements an advancing front algorithm instead of an algorithm based on spatial data structures used here.

---

[4] The DuMu^x code for such a simulation (2D–3D embedded fracture model) can be found in the DuMu^x repository under test/multidomain/ embedded/2d3d/1p_1p.
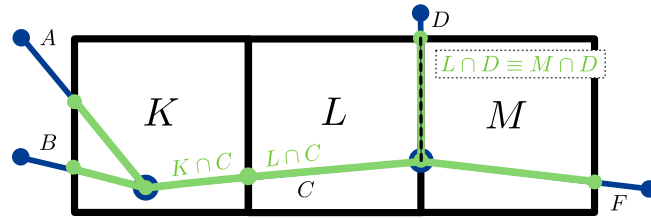
Fig. 3. Determination of the coupling stencil for an embedded fracture model using a cell-centered finite volume scheme. One grid, $\mathcal{M}_1 = \mathcal{T}_1 = \{K, L, M\}$, consists of three squares (black), and a second grid, $\mathcal{M}_2 = \mathcal{T}_2 = \{A, B, C, D, F\}$, consists of five segments (blue). Intersections between the grid element are highlighted in green. Each element $E \in \mathcal{M}_j$, $j \in \{1, 2\}$, is associated with a degree of freedom u_E with index i_E, and an element residual r_E, see Eq. (3.5). The element residual is a function of degrees of freedom of both domains. The indices of these degrees of freedom are collected in the stencil $\mathcal{S}_E$ (same domain) and the coupling stencil $\mathcal{S}_E^c$ (other domain). For the presented example, $\mathcal{S}_L = \{i_K, i_L, i_M\}$, $\mathcal{S}_L^c = \{i_C, i_D\}$ (*bulk coupling stencil* in Code Example 5), $\mathcal{S}_D = \{i_C, i_D, i_F\}$, $\mathcal{S}_D^c = \{i_L, i_M\}$ (*lowdim coupling stencil* in Code Example 5), and analogously for all other elements. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Furthermore, DuMu^x provides an assembler class for multi-domain models, which assembles the discrete PDE system in residual form

$$
\begin{bmatrix}
A_1 & C_{12} & \dots & C_{1n} \\
C_{21} & A_2 & & \\
\vdots & & \ddots & \\
C_{n1} & & & A_n
\end{bmatrix}
\begin{bmatrix}
\Delta u_1 \\
\Delta u_2 \\
\vdots \\
\Delta u_n
\end{bmatrix}
=
\begin{bmatrix}
r_1 \\
r_2 \\
\vdots \\
r_n
\end{bmatrix},
\tag{4.1}
$$

where A_i is the Jacobian of the discrete PDE system for subdomain i, and C_{ij} is the coupling Jacobian with derivatives of residuals of domain i with respect to degrees of freedom of domain j, $C_{ij} = \frac{\partial r_i}{\partial u_j}$. The assembler class and the matrix class are generic, so that the sub-vectors u_i and sub-matrices A_i can themselves have a block structure, and support an arbitrary number of subdomains. The block structure can be exploited, for example, for constructing preconditioners for a monolithic solver, or to algebraically implement schemes where the subdomain systems are solved successively in an iterative algorithm.

The presented multi-domain concept has been successfully used to implement models with coupled flow and transport processes in vascularized brain tissue [23,63], root–soil interaction models in the vadose zone [18], flow and transport models for fractured rock systems [12,64], coupled porous medium flow and atmosphere flow (Darcy–Navier–Stokes) at the soil surface [55], and a model that couples a pore-network model with a Navier–Stokes model [21].

5. New features in DuMu^x 3

Numerous new features are added in DuMu^x 3 in comparison with the 2.X series. We briefly mention the most important changes. A complete redesign of many high-level class abstractions such as assembler, linear and non-linear solvers, grid readers and grid geometry, and file I/O leads to more readable and flexible `main` functions, see also Section 2. The numerous models are improved in terms of code reuse and modularity, so that code duplication is minimized and readability improved. In addition to the fluid system concept, a solid system concept is introduced, which facilitated the implementation of new models including mineralization or precipitation of substances that potentially modify the porous matrix structure [17,65]. Porous material properties such as intrinsic permeability and porosity can be implemented to depend (linearly or non-linearly) on the primary variables. We generalized thermal and chemical non-equilibrium models to be combinable with any porous medium model. Multi-component diffusion can now be modeled by Maxwell–Stefan diffusion. A versatile implementation of cell-centered MPFA-O scheme is now usable with all models [12]. The Navier–Stokes models are redesigned to use a MAC scheme on a staggered grid [21,55,66], including a Reynolds-averaged Navier–Stokes model [67,68] with a variety of turbulence models (e.g. $k - \epsilon$, $k - \omega$), as well as a second-order upwind scheme. We can now solve problems based on the two-dimensional shallow water equations. Finally, the multi-domain module adds the functionality of versatile model coupling as described in Section 4.

6. Numerical examples

Previous versions of DuMu^x have been developed with a strong focus on multi-phase flow and transport in porous media [1]. The quality of the multi-phase flow models has been maintained or improved[5] in version 3. As an example, we present a two-phase flow scenario based on the Norne data set.[6] The porosity field, the isotropic heterogeneous

---

[5] We refer to the DuMu^x changelog (https://git.iws.uni-stuttgart.de/dumux-repositories/dumux/blob/master/CHANGELOG.md).

[6] Norne data set obtained from the module `opm-data` (Open Database License, https://github.com/OPM/opm-data).
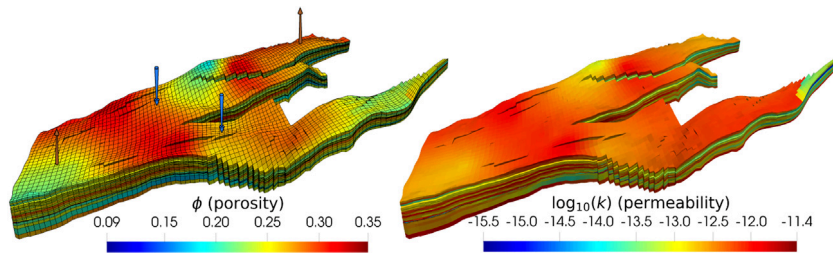
Fig. 4. Porosity and permeability field of the Norne formation. Permeability is isotropic and values are given in m². The left image additionally highlights the elements of the grid and displays the location of the injection (blue) and production wells (brown). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)
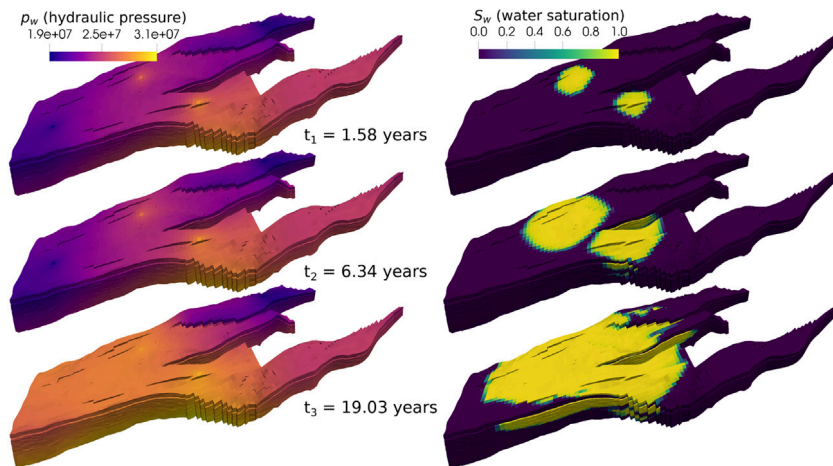

Fig. 5. Evolution of the wetting phase pressure (in Pa) and saturation. This is a new visualization of the data obtained with DuMu^x by Schneider et al. [58], based on the injection scenario originally presented in [70].

permeability field, the elements E of the computational grid $\mathcal{M}$, and the injection and extraction wells are shown in Fig. 4. The computational domain is represented by a corner-point grid [69,70], using the dedicated Dune grid interface implementation of opm-grid [37]. The incompressible immiscible two-phase model equations and parameters are given in [70, Eqs. (56)–(60) and Table 3]. The nonlinear coupled PDEs are discretized with a TPFA cell-centered finite volume scheme in space and a backward Euler scheme in time. Initially, the domain is fully saturated with oil. Water is injected through two wells. Two extraction wells initially produce oil and an oil–water mixture at later times. The wells are modeled by fixed bore-hole pressures using a Peaceman well model [71]. The temporal evolution of the wetting phase pressure and saturation is shown in Fig. 5.

The development of DuMu^x 3 has been particularly driven by a focus on model coupling. In the remainder of this section, three numerical examples demonstrate the flexibility of the new multi-domain framework in DuMu^x 3. In Section 6.1, free flow over a porous medium is modeled by coupling a Navier–Stokes model to a pore-network model. The domains are coupled at a common interface. Section 6.2 shows a simulation of two-phase flow in a fractured rock matrix. The fracture flow is computed on lower-dimensional domains conforming with the control-volume faces of the three-dimensional rock matrix domain discretization, which allows to model highly conductive fractures as well as impermeable fractures. The example shows the differences between a TPFA and an MPFA-O finite volume scheme for a rock matrix with anisotropic permeability. Finally, an example of root water uptake and tracer transport is given in Section 6.3. The roots are represented by a network of tubes embedded into the soil matrix. The mass exchange between the two non-conforming domains is realized with adequate source terms.

The source code for these examples as well as instructions for the reproduction of the results can be found in the DuMu^x-pub module to this publication at git.iws.uni-stuttgart.de/dumux-pub/dumux2019. For pre- and post-processing of DuMu^x simulation results, we advertise a workflow based on a number of external open source tools such as Gmsh [72] (meshing), Gnuplot [73] and Matplotlib [74] (plotting), and ParaView [75] (visualization). The latter has been used to create the visualizations in Figs. 4–7, 9 and 10.
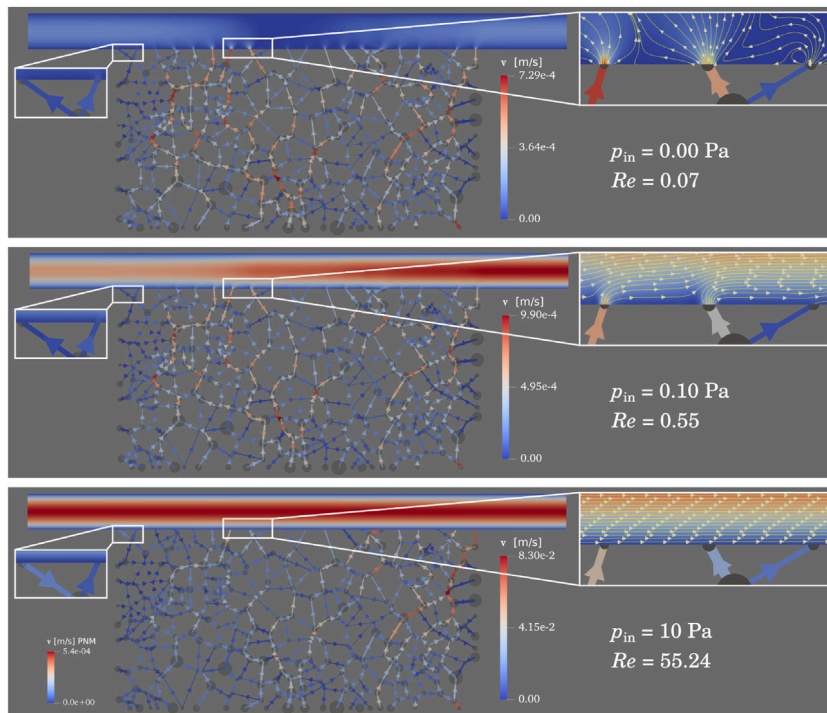
Fig. 6. Velocity fields for the three scenarios. *Re* is based on the averaged velocity within the channel. Note the different color scale for the network in the third scenario (bottom). $p_{out} = 0$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)
Source: Figure adapted from [21] (license: CC BY 4.0).

$p_{in} = 0\,\mathrm{Pa}$, $Re = 0.07$, $t_1 = 14\,\mathrm{s}$

$t_2 = 50\,\mathrm{s}$

$p_{in} = 0.1\,\mathrm{Pa}$, $Re = 0.55$, $t_1 = 14\,\mathrm{s}$

$t_2 = 50\,\mathrm{s}$

$p_{in} = 10\,\mathrm{Pa}$, $Re = 55.24$, $t_1 = 28\,\mathrm{s}$

$t_2 = 126\,\mathrm{s}$

Fig. 7. Distribution of the mole fraction x^t for the three different scenarios at different times.
Source: Figure adapted from [21] (license: CC BY 4.0).

Fig. 8. Domain and Dirichlet boundary conditions for the two-phase flow example through fractured porous media. The subscript hs refers to hydrostatic pressure conditions.

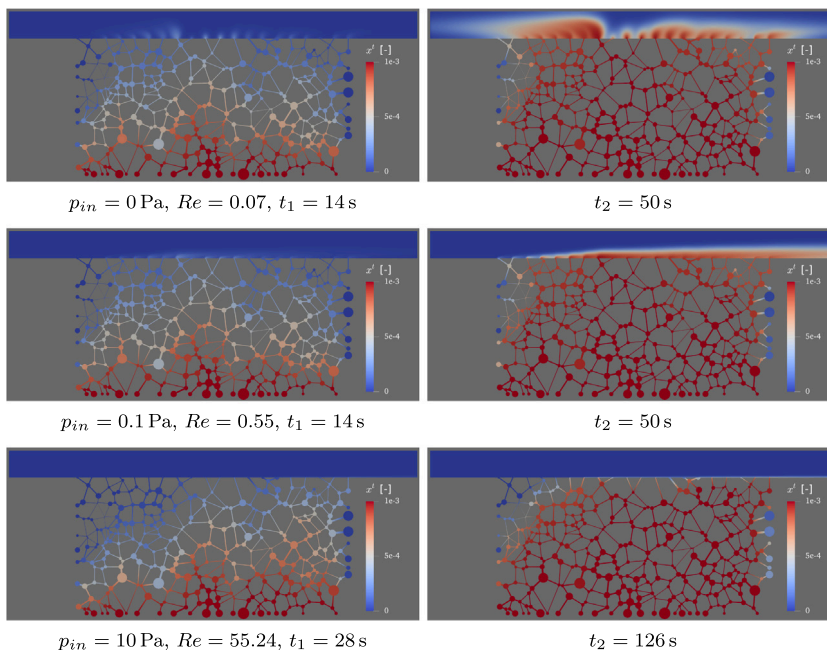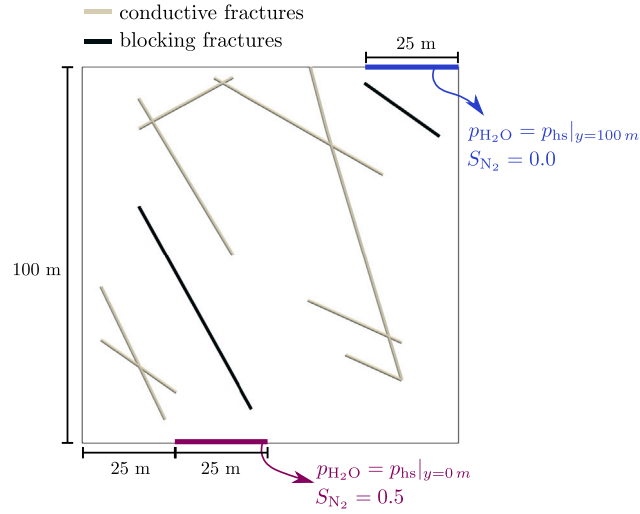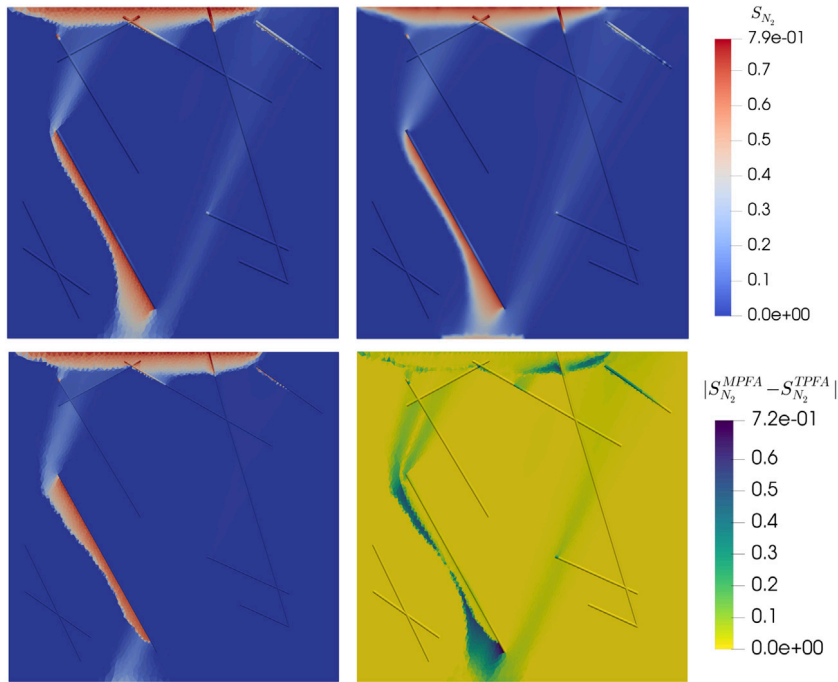**Fig. 9.** Nitrogen saturation distribution at the final simulation time $t = 75\,000\,s$ obtained with the MPFA-O scheme (upper left), the BOX scheme (upper right) and the TPFA scheme (lower left) for the example application of two-phase flow through a fractured porous medium. The lower right image shows the difference in the saturations obtained with the MPFA-O and the TPFA scheme.

6.1. Coupling a free flow model with a pore-network model

This example is adapted from [21], where a coupled model of free channel flow adjacent to a porous medium is presented with a detailed model description. We model transient multi-component flow over a random porous structure in a two-dimensional model domain. The channel flow is governed by the Navier–Stokes equations, neglecting gravity and dilatation [76],

$$\frac{\partial(\rho\mathbf{v})}{\partial t} + \nabla \cdot \left(\rho\mathbf{v}\mathbf{v}^T\right) = \nabla \cdot \left[\mu\left(\nabla\mathbf{v} + (\nabla\mathbf{v})^T\right)\right] - \nabla p, \tag{6.1}$$

Fig. 10. Simulation of root-water uptake of a white lupin and simultaneous tracer transport in the soil, shown at $t = 3$ d. Three horizontal cuts through the soil domain are shown. The tracer, with mo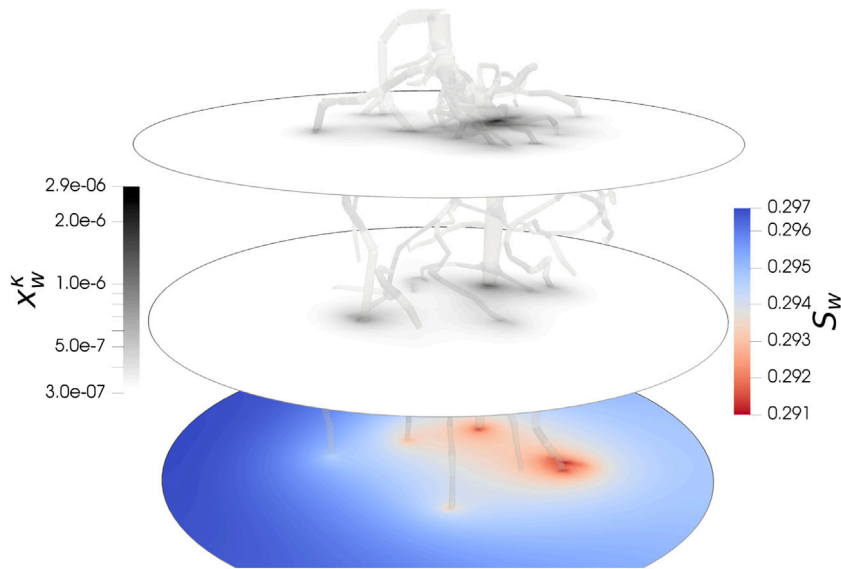le fraction x_w^κ, is not taken up by the roots and accumulates, particularly where the root water uptake rate is highest. On the bottom slice, the water saturation S_w is shown. The saturation slightly decreases close to the roots. Its spatial gradient depends on the flow resistances in soil and root, the current water distribution, and the prescribed transpiration rate r_T.

with the fluid density ρ, the fluid velocity $\mathbf{v}$, the fluid pressure p, and the dynamic viscosity μ. A molar balance equation for each component κ,

$$\frac{\partial (\rho_m x^\kappa)}{\partial t} + \nabla \cdot (\rho_m x^\kappa \mathbf{v} - D\rho_m \nabla x^\kappa) = 0, \tag{6.2}$$

models advective and diffusive transport of the fluid components, where x^κ is the component mole fraction, ρ_m the fluid molar density, and D the binary diffusion coefficient. Diffusive fluxes are described by Fick's law.

For modeling flow and transport in the porous medium, a pore-network model is used [77,78]. The complex pore-scale geometry of the porous medium is transferred to a simplified geometry, the pore bodies and pore throats. For each component κ, Eq. (6.2) is formulated discretely on each pore body where the primary variables are located. The advective and diffusive fluxes are evaluated on the pore throats. We refer to [21] for further details.

Coupling conditions are formulated at the interface between the two models to ensure thermodynamic consistency. At the locations where no throat intersects with the interface, a no-flow/no-slip condition for the Navier–Stokes model is enforced. At the actual intersections, we prescribe the continuity of normal forces [79] resulting in a Neumann boundary condition for the free flow domain,

$$\mathbf{n} \cdot \left[\left(\left(\rho \mathbf{v}\mathbf{v}^T - \mu \left(\nabla \mathbf{v} + (\nabla \mathbf{v})^T \right) + p\mathbf{I} \right) \mathbf{n} \right) \right]^{FF} = [p]^{PNM}, \tag{6.3}$$

where the superscripts FF and PNM mark quantities of the free-flow and pore-network model, respectively. We use the tangential part of the average velocity within the throat at the boundary $[\mathbf{v}]^{PNM}$ as a slip condition for the free flow,

$$[\mathbf{v} \cdot \mathbf{t}]^{FF} = \begin{cases} [\mathbf{v}]^{PNM} \cdot [\mathbf{t}]^{FF} & \text{on pore throat,} \\ 0 & \text{else,} \end{cases} \tag{6.4}$$

where $[\mathbf{t}]^{FF}$ is a unit tangential vector to the coupling interface.

The velocity within the throat is given by

$$[\mathbf{v}]^{PNM} = \frac{Q_t}{A_t} \mathbf{n_t}, \tag{6.5}$$

where Q_t is the volume flow within the throat, A_t is its respective cross-sectional area and $\mathbf{n_t}$ is a unit vector parallel to the pore throat, pointing towards the coupling interface. Finally, we require the conservation of mass

$$[(\rho_m x^\kappa \mathbf{v} - D\rho_m \nabla x^\kappa) \cdot \mathbf{n}]^{FF} = -[(\rho_m x^\kappa \mathbf{v} - D\rho_m \nabla x^\kappa) \cdot \mathbf{n}]^{PNM} \tag{6.6}$$

and enforce the continuity of mole fractions at the interface,

$$[x^\kappa]^{FF} = [x^\kappa]^{PNM}. \tag{6.7}$$

The Navier–Stokes equations are discretized with a MAC scheme on a staggered grid. The pore-network model is also implemented in DuMu^x and will become part of stable code basis in an upcoming release. The pore-network is described as a one-dimensional network embedded in a two-dimensional domain, using the DUNE grid implementation `dune-foamgrid` [80]. We used the DuMu^x multi-domain framework in order to achieve a fully monolithic coupling between the two sub-models. Since only elements on the domain boundary are coupled in this example, we employed a simplified intersection algorithm for creating the coupling stencils in comparison with Code Example 5. Instead of intersecting the entire grid, only the end points (boundary pores) of the pore-network grid geometry are intersected with the channel domain grid geometry to compute the coupling stencils. Furthermore, for the staggered grid discretization, we compute separate coupling stencils for the degrees of freedom located at cell centers and those located on cell faces. Newton's method is used to solve the nonlinear system of equations in combination with SuiteSparse's UMFPack [81] as a direct linear solver. Implementation details can be found in the folder `dumux/multidomain/boundary` located in the DuMu^x repository and in the DuMu^x-pub module accompanying this paper (see above).

The given example is discussed in detail in [21]. Fluid density and viscosity are assumed constant, with $\rho = 1 \times 10^3$ kg/m^3 and $\mu = 1 \times 10^{-3}$ Pa s. A tracer injected at the bottom of the pore network is transported upwards until it reaches the free-flow channel through which it leaves the system at its left or right sides where fixed pressures are set (see Fig. 6). All other sides of the channel are closed and no-flow/no-slip conditions hold. In the pore network, Dirichlet conditions for p and x^κ are set at the bottom while Neumann no-flow boundaries are assigned to the lateral sides. Varying the pressure gradients in the free-flow channel yields three different scenarios. Fig. 6 shows the resulting velocity fields where distinct preferential flow paths in the network and the influence of the inclined throats at the interface become visible. Fig. 7 shows the temporal development of the concentration fields. At t_1, an average mole fraction of 5×10^{-4} is reached while t_2 corresponds to a value of 9×10^{-4}. While the first two cases reach these points after 14 and 50 s, the higher pressure gradient in the channel for the case with $Re = 55.24$ repels the tracer, keeping it longer in the network. As there is no imposed flow for the first case, the tracer spreads equally to the left and right side of the channel while in the other two cases, the pressure gradient drives the tracer towards the right outlet. The formation of a boundary layer at the interface can be observed which becomes thinner for the higher Re case.

6.2. Two-phase flow through fractured porous media

The example shown in this section is inspired by an exercise of the DuMu^x-course[7] and considers the buoyancy-driven upwards migration of gaseous nitrogen in an initially fully water-saturated fractured porous medium. In this model, the fractures are assumed to represent very thin heterogeneities with substantially differing material parameters in comparison to the surrounding porous medium. The thin nature of these inclusions favors a dimension-reduced description of the fractures as entities of co-dimension 1, on which cross-section averaged PDEs are solved and appropriate coupling conditions describe the interaction with the surrounding porous medium. Such approaches have been widely reported in the literature for both single-phase flow (see e.g. [82–84]) and two-phase flow (see e.g. [85–87]) in fractured porous media. In the approach presented in this example, the two subdomains are not discretized independently, but it requires the facets of the higher-dimensional grid (bulk grid) to be conforming with the elements of the lower-dimensional grid for the fractures. Therefore, we currently rely on grid file formats that allow the extraction of both grids from a single file together with the connectivity information between them. The implementation in this example uses mesh files generated by Gmsh [72]. This facilitates the determination of the coupling stencils in comparison with non-conforming methods (see Code Example 5), since grid intersection at runtime is avoided. All classes and functions related to this approach can be found in the `dumux/multidomain/facet` folder of the DuMu^x repository. For further details on the numerical scheme, we refer to [12,64].

The fracture network geometry used in this example is taken from [84] and is shown together with the boundary conditions in Fig. 8. The computational domain comprises a two-dimensional domain for the matrix, Ω_m and a one-dimensional domain for the fracture network, Ω_f. We denote by Γ_m the outer boundary of Ω_m and further decompose this into Γ_m^D and Γ_m^N, referring to the subsets on which Dirichlet and Neumann conditions are specified, respectively. The boundary of Ω_f is denoted with Γ_f. Furthermore, we introduce the partitions Ω_f^C and Ω_f^B such that $\Omega_f = \Omega_f^C \cup \Omega_f^B$, which refer to the conductive (superscript C) and blocking fractures (superscript B). Finally, we define the interfaces between the matrix domain and the conductive and blocking fractures as $\gamma_C = \Omega_m \cap \Omega_f^C$ and $\gamma_B = \Omega_m \cap \Omega_f^B$. Making use of this notation, we state the governing system of equations as follows:

$$\mathbf{v}_{\beta,i} + \frac{k_{r\beta,i}}{\mu_\beta} \mathbf{K}_i \left(\nabla p_{\beta,i} - \rho_\beta \mathbf{g} \right) = 0, \qquad \text{in } \Omega_i, \tag{6.8a}$$

$$\frac{\partial \left(\phi_m \rho_\beta S_{\beta,m} \right)}{\partial t} + \nabla \cdot \left(\rho_\beta \mathbf{v}_{\beta,m} \right) = 0, \qquad \text{in } \Omega_m, \tag{6.8b}$$

$$\frac{\partial \left(a \, \phi_f \rho_\beta S_{\beta,f} \right)}{\partial t} + \nabla \cdot \left(a \, \rho_\beta \mathbf{v}_{\beta,f} \right) = [\![\rho_\beta \mathbf{v}_{\beta,m} \cdot \mathbf{n}]\!], \qquad \text{in } \Omega_f, \tag{6.8c}$$

---

[7] The exercise can be found at git.iws.uni-stuttgart.de/dumux-repositories/dumux-course/tree/master/exercises/exercise-fractures.

Table 1
Parameter choices for the two-phase flow through fractured porous media example.

Symbol	Name	Ω_{m}	$\Omega_{\mathrm{f}}^{\mathrm{B}}$	$\Omega_{\mathrm{f}}^{\mathrm{C}}$	Unit
ϕ	Porosity	0.15	0.15	0.85	–
k_{h}	hor. permeability	1×10^{-12}	1×10^{-16}	1×10^{-9}	m^2
$\varXi$	Anisotropy ratio	0.15	1	1	m^2
$\varPhi$	Permeability angle	-25	0	0	°
α	Van Genuchten parameter	1×10^{-3}	1×10^{-2}	1×10^{-4}	Pa^{-1}
n	Van Genuchten parameter	3	2	23	–

$$-\frac{k_{\mathrm{r}\beta,\mathrm{f}}}{\mu_\beta}\mathbf{n}^T \mathbf{K}_{\mathrm{m}}^{\perp}\left(\frac{p_{\beta,\mathrm{m}} - p_{\beta,\mathrm{f}}}{a/2}\mathbf{n} - \rho_\beta \mathbf{g}\right) = \mathbf{v}_{\beta,\mathrm{m}} \cdot \mathbf{n}, \qquad\qquad \text{on } \gamma_{\mathrm{B}}, \qquad\qquad (6.8\mathrm{d})$$

$$p_{\beta,\mathrm{m}} = p_{\beta,\mathrm{f}}, \qquad\qquad \text{on } \gamma_{\mathrm{C}}, \qquad\qquad (6.8\mathrm{e})$$

for $i \in \{\mathrm{m, f}\}$ and $\beta \in \{\mathrm{H_2O, N_2}\}$. Furthermore, ρ_β and μ_β are the density and viscosity of a fluid phase β, while $p_{\beta,i}$, $S_{\beta,i}$ and $k_{\mathrm{r}\beta,i}$ denote the fluid phase pressure, saturation and relative permeability as prevailing in subdomain i. Correspondingly, $\mathbf{K}_i$ and ϕ_i denote the intrinsic permeability and the porosity in a subdomain, and $\mathbf{v}_{\beta,i}$ is the respective Darcy velocity. Finally, $\mathbf{K}_{\mathrm{f}}^{\perp}$ denotes the normal part of the permeability in the fracture domain and a is the aperture of the fracture.

Eq. (6.8a) states that Darcy's law is valid within all subdomains, while Eqs. (6.8b) and (6.8c) are the mass balance equations for the fluid phases in the matrix and fracture subdomain, respectively. Note that the term $[\![\rho_\beta \mathbf{v}_{\beta,\mathrm{m}} \cdot \mathbf{n}]\!]$ describes the jump in the normal flux of a phase β across the fracture and acts as an additional source or sink term, caused by the interaction with the surrounding matrix. On blocking fractures, flux and pressure continuity are enforced by means of Eq. (6.8d), while on conductive fractures, the pressure continuity condition Eq. (6.8e) is used, which assumes the pressure jump across the fracture to be negligible. The system of equations is completed by the following set of boundary conditions:

$$a\,\rho_\beta \mathbf{v}_{\beta,\mathrm{f}} \cdot \mathbf{n} = 0, \qquad\qquad \text{on } \Gamma_{\mathrm{f}} \qquad\qquad (6.9\mathrm{a})$$

$$\rho_\beta \mathbf{v}_{\beta,\mathrm{m}} \cdot \mathbf{n} = 0, \qquad\qquad \text{on } \Gamma_{\mathrm{m}}^{\mathrm{N}} \qquad\qquad (6.9\mathrm{b})$$

$$p_{\mathrm{H_2O,m}} = p_{\mathrm{hs}}, \qquad\qquad \text{on } \Gamma_{\mathrm{m}}^{\mathrm{D}}, \qquad\qquad (6.9\mathrm{c})$$

$$S_{\mathrm{N_2,m}} = S_{\mathrm{N_2}}^{\mathrm{D}}, \qquad\qquad \text{on } \Gamma_{\mathrm{m}}^{\mathrm{D}}. \qquad\qquad (6.9\mathrm{d})$$

Here, the subscript hs refers to hydrostatic conditions and the boundary saturation $S_{\mathrm{N_2}}^{\mathrm{D}}$ is defined as depicted in Fig. 8. In both domains, the water pressure $p_{\mathrm{H_2O}}$ and the nitrogen saturation $S_{\mathrm{N_2}}$ are used as primary variables and the system is closed via the constitutive relationships $S_{\mathrm{H_2O},i} + S_{\mathrm{N_2},i} = 1$ and $p_{\mathrm{N_2},i} = p_{\mathrm{H_2O},i} + p_{\mathrm{c},i}$, where $p_{\mathrm{c},i} = p_{\mathrm{c},i}(S_{\mathrm{H_2O},i})$ denotes the capillary pressure as a function of the water saturation. For this example, we use the van Genuchten–Mualem model for capillary pressure and relative permeability [88–90]. An overview over the material parameter distributions used in the subdomains is given in Table 1.

Note that the permeability of a subdomain is defined as

$$\mathbf{K}_i = \mathbf{R}(\varPhi)^{-1}\begin{pmatrix} k_{\mathrm{h}} & 0 \\ 0 & k_{\mathrm{h}}/\varXi \end{pmatrix}\mathbf{R}(\varPhi), \qquad\qquad (6.10)$$

where $\mathbf{R}$ is the two-dimensional rotation matrix. In addition to that, the aperture is set to $a = 0.05$ m for all fractures. The relationships of the water density and viscosity on temperature and pressure are taken from [91], while the nitrogen gas viscosity is modeled after [92]. The density of gaseous nitrogen is described with the ideal gas law. The problem has been solved using the TPFA, MPFA-O and the BOX scheme. Switching between the different schemes is realized by changing at most two lines in the code. The grid used in this example consists of 20 910 2-dimensional and 421 1-dimensional elements. Fig. 9 depicts the results for the BOX and the MPFA-O scheme, as well as the difference in the solution between TPFA and MPFA-O at the final simulation time of 75 000 s. It can be seen that as nitrogen migrates upwards driven by buoyancy, it avoids entering the blocking fractures, which act as both capillary and hydraulic barriers, and preferentially flows around them. On the other hand, after entering conductive fractures, the nitrogen rapidly flows vertically and accumulates at the fracture tips due to capillary forces and the limited transport capacity of the surrounding medium. Furthermore, the nitrogen gas accumulates below the upper no-flow boundary. The differences between TPFA and MPFA-O highlight the importance of using consistent schemes on unstructured grids and in presence of full tensor permeabilities. This example shows that the model is able to qualitatively capture the hydraulic effects of fractures acting as both hydraulic and capillary barriers as well as conduits. For a more quantitative assessment of the performance of this model we refer to [12,64].

6.3. Root–soil interaction

In the last example, we simulate root-water uptake including tracer transport. The employed model concepts and numerical methods are described in detail in [18], where a similar example is discussed. A comparable but slightly simpler example is included in the DuMu^x test suite.[8]

Root-water uptake is modeled in a cylindrical soil domain Ω with a radius of 5 cm and a height of 10 cm containing a 2-week-old white lupin root system (with center-line skeleton Λ), reconstructed from MRI data [93]. The soil domain Ω is discretized with an unstructured tetrahedral grid refined around the root system, using dune-uggrid, while Λ is represented by an independent grid of line segments forming a root network, using dune-foamgrid [80]. At the root collar a transpiration rate of $r_T = \min\{2.15 \times 10^{-8} \text{ kg s}^{-1}, r_{T,c}\}$ is prescribed, where $r_{T,c}$ is the transpiration rate for which the root collar pressure is $p_{r,c} = -1.4$ MPa (wilting point pressure). The soil domain contains a tracer of initially uniformly distributed concentration ($x_w^\kappa = 3 \times 10^{-7}$). The tracer (binary diffusion coefficient in water $D_w^\kappa = 2.3 \times 10^{-9} \text{ m s}^{-2}$) does not enter the roots. As a consequence, it is expected to accumulate at the locations of the largest root water uptake. The soil flow is modeled with the Richards equation, and the flow inside the root xylem is modeled using a bundle-of-tubes approach, resulting in a Darcy-type flow model, cf. [18]. The nonlinear PDE system

$$\frac{\partial \left(\phi_s S_w \rho_{m,w}\right)}{\partial t} - \nabla \cdot \left(\rho_{m,w} \boldsymbol{v}_s\right) = q_w \delta_\Lambda \qquad \text{in } \Omega, \qquad (6.11a)$$

$$\frac{\partial \left(\phi_s S_w \rho_{m,w} x_w^\kappa\right)}{\partial t} - \nabla \cdot \left(x_w^\kappa \rho_{m,w} \boldsymbol{v}_s + \rho_{m,w} D_{eff}^\kappa \nabla x_w^\kappa\right) = 0 \qquad \text{in } \Omega, \qquad (6.11b)$$

$$\frac{\partial \left(\phi_r \rho_{m,w} A_r\right)}{\partial t} - \frac{\partial}{\partial \zeta} \left(\rho_{m,w} v_r\right) = -q_w \qquad \text{in } \Lambda, \qquad (6.11c)$$

where

$$\boldsymbol{v}_s = -\frac{k_{rw}}{\mu_w} K \left(\nabla p_w + \rho_w g \nabla z\right), \quad v_r = -K_{ax} \left(\frac{\partial p_w}{\partial \zeta} + \rho_w g \frac{\partial z}{\partial \zeta}\right), \quad q_w = -2\pi R k_{rw} K_{rad} \left(\bar{p}_s - p_r\right) \rho_{m,w},$$

with the average soil pressure $\bar{p}_s = \frac{1}{2\pi} \int_0^{2\pi} p_s|_R d\theta$, and an analogous definition for $\bar{x}_{w,s}^\kappa$ [18,94], is solved for water pressure p_w and tracer mole fraction x_w^κ for a period of 3 d with a maximum time step size of 1 h. The symbols and parameter values are given in Table 2. The subscripts r and s denote root and soil quantities wherever they need to be distinguished. Eqs. (6.11a) and (6.11b) are spatially discretized using the box scheme, while Eq. (6.11c) is discretized with the TPFA scheme. The corresponding coupling manager is generic, in the sense that it can compute the coupling stencils between the root and the soil domain if the discretization schemes in both domains differ. The box scheme is superior to the TPFA scheme in the soil domain for the presented case, because it usually uses less degrees of freedom on tetrahedral grids, and it is well-known that the TPFA scheme is not consistent on non-K-orthogonal grids [57]. In comparison with Code Example 5, the procedure to compute the coupling stencil is slightly more complex for this example, since the integration of the source term q_w requires the evaluation of quantities on the root segment surface, resulting in non-local stencils. For details on the discretization method for this mixed-dimension embedded problem, we refer to [18]. The required classes and functions can be found in the DuMu^x repository's dumux/multidomain/embedded folder.

The resulting spatial distribution of water and the tracer is shown in Fig. 10. A tracer accumulation in close vicinity to the roots is evident. This type of simulation may present a valuable tool to interpret experimental tracer data. Due to the non-linear relation of the tracer concentration distribution to the water uptake rate and its dependence on water distribution, it is otherwise difficult to obtain quantitative result on local root water uptake rates from tracer concentration measurements.

7. Current limitations and perspectives

As a research code under active development, DuMu^x currently has certain limitations, from which many may be resolved in future versions. We briefly mention some of the limitations.

Multi-domain simulations in DuMu^x currently do not run in parallel when involving multiple grids. In Dune, the Dune grid implementations are responsible for managing MPI-based distributed memory parallelism. When having two or more grid instances, data has to be communicated between grids. First steps towards such a feature within the Dune framework have been undertaken with the dune-grid-glue module [62]. Efficient load balancing in complicated multi-domain, or mixed-dimensional setups is challenging and is an active field of research targeted by codes such as the MOOSE framework [44], or preCICE [95].

DuMu^x currently supports forward and backward Euler time discretizations only. Implementing other time discretization schemes requires some non-trivial refactoring of the assembly process.

As discussed in Section 2, DuMu^x uses a C++ programming technique based traits of tags to define what we call *properties* of a model. Due to the dependency on a tag, many of such properties (often corresponding to C++ types) are

---

[8] The example can be found at git.iws.uni-stuttgart.de/dumux-repositories/dumux/tree/master/test/multidomain/embedded/1d3d/1p2c_richards2c.

Table 2
Symbols and parameter values for the root–soil interaction example.

Symbol	Name	Value	Unit
ϕ_s	Soil porosity	0.4	–
$\rho_{m,w}$	Molar density of water	5.55×10^4	$mol\,m^{-3}$
D_{eff}^{κ}	Effective diffusion coefficient	[18, Eq. (16)]	
ϕ_r	Root porosity	0.4	–
A_r	Root cross-sectional area		m^2
ζ	Local axial root coordinate		–
S_w	Water saturation	$S_w(p_w)$, [88] [18, Eq. (8)]	–
k_{rw}	Relative permeability	k_{rw} [88,89] [18, Eq. (9)]	–
α	Van Genuchten parameter	2.956×10^{-4}	Pa^{-1}
n	Van Genuchten parameter	2.0	–
S_{wr}	Residual water saturation	0.1	–
g	Gravitational acceleration	9.81	$m\,s^{-2}$
K_{ax}	Axial root conductivity	5.10×10^{-17}	$m^4/Pa/s$
K_{rad}	Radial root conductivity	2.04×10^{-11}	$m\,Pa^{-1}\,s^{-1}$

grouped together. Some classes have such a tag as a template argument (see `FVAssembler` in Code Example 2, l. 44). While this significantly reduces the number of template arguments of a class, it leads to an nontransparent way of injecting dependencies. Furthermore, `FVAssembler<Tag1>` and `FVAssembler<Tag2>` are different types, even if all type traits (properties) extracted from the tags are identical. Modularity and re-usability of such classes is impeded. Thus, the use of tags as template arguments has been significantly reduced in DuMu^x 3 in comparison with previous versions. A medium-term goal is to replace all such classes with explicit dependencies. The property technique is also the main reason, why it is currently difficult to add useful Python wrappers (a feature desired by many DuMu^x users) for DuMu^x.

The focus of DuMu^x is on providing many usable and extensible models, and less on flexible linear algebra. While efficient and flexible data structures and easy-to-use and extensible, preconditioned linear solvers are available through `dune-istl` [31], DuMu^x currently does not facilitate algebraic manipulations of linear equation systems, such as different orderings of the Jacobian matrix, quasi-Newton schemes, or complex linear solver strategies based on matrix decomposition.

The future development of DuMu^x will continue to reflect the most recent advances in the field of porous medium research within the DuMu^x community. For example, the goal of a current project is the improvement of solvers for free-flow and shallow water models coupled with porous medium flow models. Pore-network models for single- and multi-phase flow will be included into the DuMu^x framework in an upcoming release, including static and dynamic approaches. In collaboration with the development team of preCICE [95], the model coupling capabilities of DuMu^x with internal and external modules will be further improved. The usability and archivability of DuMu^x will be further improved within the aforementioned project SusI which constitutes a step towards reproducible research.

CRediT authorship contribution statement

T. Koch, D. Gläser, K. Weishaupt wrote and proof-read this manuscript, conceptualized, implemented and visualized the presented examples, and curated the data and software. Furthermore they are the authors with the most code contributions (measured in Git commits to the DuMu^x repository since the previous release 2.12). B. Flemisch wrote and proof-read parts of this manuscript, continuously contributes code, is the administrative head and coordinator of the DuMu^x project, and acquired funding. All other authors (listed in alphabetical order) committed source code to the DuMu^x repository since release 2.12 and thus contributed to the development and increasing quality and number of features of the DuMu^x research software framework, version 3.

Acknowledgments

References

[1] B. Flemisch, M. Darcis, K. Erbertseder, B. Faigle, A. Lauser, K. Mosthaf, S. Müthing, P. Nuske, A. Tatomir, M. Wolff, R. Helmig, DuMu^x: DUNE for multi-{phase, component, scale, physics, …} flow and transport in porous media, Adv. Water Resour. 34 (9) (2011) 1102–1112, http://dx.doi.org/10.1016/j.advwatres.2011.03.007.

[2] J.M. Nordbotten, B. Flemisch, S.E. Gasda, H.M. Nilsen, Y. Fan, G.E. Pickup, B. Wiese, M.A. Celia, H.K. Dahle, G.T. Eigestad, K. Pruess, Uncertainties in practical simulation of CO_2 storage, Int. J. Greenh. Gas Control 9 (2012) 234–242, http://dx.doi.org/10.1016/j.ijggc.2012.03.007.

[3] E. Ahusborde, M. Kern, V. Vostrikov, Numerical simulation of two-phase multicomponent flow with reactive transport in porous media: application to geological sequestration of CO_2, ESAIM Proc. Surv. 50 (2015) 21–39, http://dx.doi.org/10.1051/proc/201550002.

[4] B. Hagemann, M. Rasoulzadeh, M. Panfilov, L. Ganzer, V. Reitenbach, Hydrogenization of underground storage of natural gas, Comput. Geosci. 20 (3) (2016) 595–606, http://dx.doi.org/10.1007/s10596-015-9515-6.

[5] L. Walter, P.J. Binning, S. Oladyshkin, B. Flemisch, H. Class, Brine migration resulting from CO_2 injection into saline aquifers – An approach to risk estimation including various levels of uncertainty, Int. J. Greenh. Gas Control 9 (2012) 495–506, http://dx.doi.org/10.1016/j.ijggc.2012.05.004.

[6] E. Ahusborde, B. Amaziane, M. Jurak, Three-dimensional numerical simulation by upscaling of gas migration through engineered and geological barriers for a deep repository for radioactive waste, Geol. Soc. Lond. Spec. Publ. 415 (1) (2015) 123–141, http://dx.doi.org/10.1144/SP415.2.

[7] N. Roy, J. Molson, J.-M. Lemieux, A. Van Stempvoort, A. Nowamooz, Three-dimensional numerical simulations of methane gas migration from decommissioned hydrocarbon production wells into shallow aquifers, Water Resour. Res. 52 (7) (2016) 5598–5618, http://dx.doi.org/10.1002/2016WR018686.

[8] K. Weishaupt, A. Bordenave, O. Atteia, H. Class, Numerical investigation on the benefits of preheating for an increased thermal radius of influence during steam injection in saturated soil, Transp. Porous Media 114 (2) (2016) 601–621, http://dx.doi.org/10.1007/s11242-016-0624-z.

[9] N. Schwenck, B. Flemisch, R. Helmig, B.I. Wohlmuth, Dimensionally reduced flow models in fractured porous media: crossings and boundaries, Comput. Geosci. 19 (6) (2015) 1219–1230, http://dx.doi.org/10.1007/s10596-015-9536-1.

[10] L. Stadler, R. Hinkelmann, R. Helmig, Modeling macroporous soils with a two-phase dual-permeability model, Transp. Porous Media 95 (3) (2012) 585–601, http://dx.doi.org/10.1007/s11242-012-0064-3.

[11] J. Tecklenburg, I. Neuweiler, J. Carrera, M. Dentz, Multi-rate mass transfer modeling of two-phase flow in highly heterogeneous fractured and porous media, Adv. Water Resour. 91 (2016) 63–77, http://dx.doi.org/10.1016/j.advwatres.2016.02.010.

[12] D. Gläser, R. Helmig, B. Flemisch, H. Class, A discrete fracture model for two-phase flow in fractured porous media, Adv. Water Resour. 110 (2017) 335–348, http://dx.doi.org/10.1016/j.advwatres.2017.10.031.

[13] N. Andrianov, H.M. Nick, Modeling of waterflood efficiency using outcrop-based fractured models, J. Pet. Sci. Eng. 183 (2019) 106350, http://dx.doi.org/10.1016/j.petrol.2019.106350.

[14] A. Fourno, T.-D. Ngo, B. Noetinger, C.L. Borderie, FraC: A new conforming mesh method for discrete fracture networks, J. Comput. Phys. 376 (2019) 713–732, http://dx.doi.org/10.1016/j.jcp.2018.10.005.

[15] E. Ahusborde, M.E. Ossmani, M.I. Moulay, A fully implicit finite volume scheme for single phase flow with reactive transport in porous media, Math. Comput. Simulation 164 (2019) 3–23, http://dx.doi.org/10.1016/j.matcom.2018.09.001.

[16] J. Hommel, E. Lauchnor, A. Phillips, R. Gerlach, A.B. Cunningham, R. Helmig, A. Ebigbo, H. Class, A revised model for microbially induced calcite precipitation: Improvements and new insights based on recent experiments, Water Resour. Res. 51 (5) (2015) 3695–3715, http://dx.doi.org/10.1002/2014WR016503.

[17] A.B. Cunningham, H. Class, A. Ebigbo, R. Gerlach, A.J. Phillips, J. Hommel, Field-scale modeling of microbially induced calcite precipitation, Comput. Geosci. 23 (2) (2019) 399–414, http://dx.doi.org/10.1007/s10596-018-9797-6.

[18] T. Koch, K. Heck, N. Schröder, H. Class, R. Helmig, A new simulation framework for soil-root interaction, evaporation, root growth, and solute transport, Vadose Zone J. 17 (2018) 1, http://dx.doi.org/10.2136/vzj2017.12.0210.

[19] T.H. Mai, A. Schnepf, H. Vereecken, J. Vanderborght, Continuum multiscale model of root water and nutrient uptake from soil with explicit consideration of the 3D root architecture and the rhizosphere gradients, Plant Soil 439 (1) (2019) 273–292, http://dx.doi.org/10.1007/s11104-018-3890-4.

[20] G.A. Futter, P. Gazdzicki, K.A. Friedrich, A. Latz, T. Jahnke, Physical modeling of polymer-electrolyte membrane fuel cells: Understanding water management and impedance spectra, J. Power Sources 391 (2018) 148–161, http://dx.doi.org/10.1016/j.jpowsour.2018.04.070.

[21] K. Weishaupt, V. Joekar-Niasar, R. Helmig, An efficient coupling of free flow and porous media flow using the pore-network modeling approach, J. Comput. Phys. X 1 (2019) 100011, http://dx.doi.org/10.1016/j.jcpx.2019.100011.

[22] K. Erbertseder, J. Reichold, B. Flemisch, P. Jenny, R. Helmig, A coupled discrete/continuum model for describing cancer-therapeutic transport in the lung, PLoS One 7 (3) (2012) 1–17, http://dx.doi.org/10.1371/journal.pone.0031966.

[23] E. Vidotto, T. Koch, T. Köppl, R. Helmig, B. Wohlmuth, Hybrid models for simulating blood flow in microvascular networks, Multiscale Model. Simul. 17 (3) (2019) 1076–1102, http://dx.doi.org/10.1137/18M1228712.

[24] K.H. Støverud, M. Darcis, R. Helmig, S.M. Hassanizadeh, Modeling concentration distribution and deformation during convection-enhanced drug delivery into brain tissue, Transp. Porous Media 92 (1) (2012) 119–143, http://dx.doi.org/10.1007/s11242-011-9894-7.

[25] K. Mosthaf, K. Baber, B. Flemisch, R. Helmig, A. Leijnse, I. Rybak, B. Wohlmuth, A coupling concept for two-phase compositional porous-medium and single-phase compositional free flow, Water Resour. Res. 47 (10) (2011) http://dx.doi.org/10.1029/2011WR010685.

[26] T. Fetzer, K.M. Smits, R. Helmig, Effect of turbulence and roughness on coupled porous-medium/free-flow exchange processes, Transp. Porous Media 114 (2) (2016) 395–424, http://dx.doi.org/10.1007/s11242-016-0654-6.

[27] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, R. Kornhuber, M. Ohlberger, O. Sander, A generic grid interface for parallel and adaptive scientific computing. Part II: Implementation and tests in DUNE, Computing 82 (2) (2008) 121–138, http://dx.doi.org/10.1007/s00607-008-0004-9.

[28] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, M. Ohlberger, O. Sander, A generic grid interface for parallel and adaptive scientific computing. Part I: Abstract framework, Computing 82 (2–3) (2008) 103–119, http://dx.doi.org/10.1007/s00607-008-0003-x.

[29] P. Bastian, M. Blatt, A. Dedner, C. Engwer, J. Fahlke, C. Gersbacher, C. Gräser, C. Grüninger, D. Kempf, R. Klöfkorn, S. Müthing, M. Nolte, M. Ohlberger, O. Sander, DUNE webpage, http://www.dune-project.org.

[30] P. Bastian, M. Blatt, A. Dedner, N.-A. Dreier, C. Engwer, R. Fritze, C. Gräser, D. Kempf, R. Klöfkorn, M. Ohlberger, O. Sander, The DUNE Framework: Basic Concepts and Recent Developments, Comput. Math. Appl. 81 (2020) 75–112.

[31] M. Blatt, P. Bastian, The iterative solver template library, in: B. Kågström, E. Elmroth, J. Dongarra, J. Waśniewski (Eds.), Applied Parallel Computing. State of the Art in Scientific Computing: 8th International Workshop, PARA 2006, Umeå, Sweden, June 18–21, 2006, Revised Selected Papers, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 666–675, http://dx.doi.org/10.1007/978-3-540-75755-9_82.

[32] L. Bilke, B. Flemisch, T. Kalbacher, O. Kolditz, R. Helmig, T. Nagel, Development of open-source porous media simulators: Principles and experiences, Transp. Porous Media (2019) http://dx.doi.org/10.1007/s11242-019-01310-1.

[33] M.G. McDonald, A.W. Harbaugh, the original authors of MODFLOW, The history of MODFLOW, Groundwater 41 (2) (2003) 280–283, http://dx.doi.org/10.1111/j.1745-6584.2003.tb02591.x.

[34] K.-A. Lie, An Introduction to Reservoir Simulation Using MATLAB/GNU Octave: User Guide for the MATLAB Reservoir Simulation Toolbox, MRST, Cambridge University Press, 2019.

[35] O. Kolditz, S. Bauer, L. Bilke, N. Böttcher, J.O. Delfs, T. Fischer, U.J. Görke, T. Kalbacher, G. Kosakowski, C.I. McDermott, C.H. Park, F. Radu, K. Rink, H. Shao, H.B. Shao, F. Sun, Y.Y. Sun, A.K. Singh, J. Taron, M. Walther, W. Wang, N. Watanabe, Y. Wu, M. Xie, W. Xu, B. Zehner, OpenGeoSys: an open-source initiative for numerical simulation of thermo-hydro-mechanical/chemical (THM/C) processes in porous media, Environ. Earth Sci. 67 (2) (2012) 589–599, http://dx.doi.org/10.1007/s12665-012-1546-x.

[36] D. Baxendale, A. Rasmussen, A.B. Rustad, T. Skille, T.H. Sandve, Open porous media flow documentation manual, 2018, URL https://opm-project.org/wp-content/uploads/2018/11/OPM-Flow-Documentation-2018-10-Rev-1.pdf.

[37] A. Flø Rasmussen, T. Harald Sandve, K. Bao, A. Lauser, J. Hove, B. Skaflestad, R. Klöfkorn, M. Blatt, A. Birger Rustad, O. Sævareid, K.-A. Lie, A. Thune, The open porous media flow reservoir simulator, Comput. Math. Appl. 81 (2020) 159–185.

[38] R.M. Maxwell, L.E. Condon, S.J. Kollet, A high-resolution simulation of groundwater and surface water over most of the continental US with the integrated hydrologic model ParFlow v3, Geosci. Model Dev. 8 (3) (2015) 923–937, http://dx.doi.org/10.5194/gmd-8-923-2015.

[39] P.C. Lichtner, G.E. Hammond, C. Lu, S. Karra, G. Bisht, B. Andre, R. Mills, J. Kumar, PFLOTRAN user manual: A massively parallel reactive flow and transport model for describing surface and subsurface processes, 2015, http://dx.doi.org/10.2172/1168703.

[40] E. Keilegavlen, A. Fumagalli, R. Berge, I. Stefansson, I. Berre, PorePy: An open-source simulation tool for flow and transport in deformable fractured rocks, 2017, CoRR abs/1712.00460 arXiv:1712.00460.

[41] W. Bangerth, R. Hartmann, G. Kanschat, Deal.II – A general-purpose object-oriented finite element library, ACM Trans. Math. Software 33 (4) (2007) http://dx.doi.org/10.1145/1268776.1268779.

[42] C. Prud'Homme, V. Chabannes, V. Doyeux, M. Ismail, A. Samake, G. Pena, Feel++: A computational framework for galerkin methods and advanced numerical methods, in: ESAIM: Proceedings, Vol. 38, EDP Sciences, 2012, pp. 429–455, http://dx.doi.org/10.1051/proc/201238024.

[43] A. Logg, K.-A. Mardal, G. Wells, Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book, Vol. 84, Springer Science & Business Media, 2012.

[44] D. Gaston, C. Newman, G. Hansen, D. Lebrun-Grandié, MOOSE: A parallel computational framework for coupled systems of nonlinear equations, Nucl. Eng. Des. 239 (10) (2009) 1768–1778, http://dx.doi.org/10.1016/j.nucengdes.2009.05.021.

[45] C. Bradley, A. Bowery, R. Britten, V. Budelmann, O. Camara, A. Christie, A. Cookson, A.F. Frangi, T.B. Gamage, T. Heidlauf, S. Krittian, D. Ladd, C. Little, K. Mithraratne, M. Nash, D. Nickerson, P. Nielsen, O. Nordbø, S. Omholt, A. Pashaei, D. Paterson, V. Rajagopal, A. Reeve, O. Röhrle, S. Safaei, R. Sebastián, M. Steghöfer, T. Wu, T. Yu, H. Zhang, P. Hunter, OpenCMISS: a multi-physics & multi-scale computational infrastructure for the VPH/Physiome project, Prog. Biophys. Mol. Biol. 107 (1) (2011) 32–47, http://dx.doi.org/10.1016/j.pbiomolbio.2011.06.015.

[46] GNU General public license, version 2, URL https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html.

[47] K. Heck, S. Ackermann, B. Becker, E. Coltman, S. Emmert, B. Flemisch, D. Gläser, C. Grüninger, T. Koch, T. Kurz, M. Lipp, F. Mohammadi, S. Scherrer, M. Schneider, G. Seitz, L. Stadler, M. Utz, A. Vescovini, F. Weinhardt, K. Weishaupt, DuMuX 3.1.0, http://dx.doi.org/10.5281/zenodo.3482428.

[48] K.-A. Lie, P. Bastian, H.K. Dahle, B. Flemisch, K. Flornes, A. Rasmussen, A.B. Rustad, OPM—Open porous media, 2009, unpublished.

[49] D. Kempf, T. Koch, System testing in scientific numerical software frameworks using the example of DUNE, Arch. Numer. Softw. 5 (1) (2017) 151–168, http://dx.doi.org/10.11588/ans.2017.1.27447.

[50] A. Lauser, The DuMuX material law framework, in: A. Dedner, B. Flemisch, R. Klöfkorn (Eds.), Advances in DUNE, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 113–128, http://dx.doi.org/10.1007/978-3-642-28589-9_9.

[51] The DuMuX developers, DuMuX handbook 3.0, https://dumux.org/handbook.

[52] R. Huber, R. Helmig, Multiphase flow in heterogeneous porous media: A classical finite element method versus an implicit pressure–explicit saturation-based mixed finite element–finite volume approach, Internat. J. Numer. Methods Fluids 29 (8) (1999) 899–920, http://dx.doi.org/10.1002/(SICI)1097-0363(19990430)29:8<899::AID-FLD715>3.0.CO;2-W.

[53] R. Helmig, et al., Multiphase Flow and Transport Processes in the Subsurface: a Contribution to the Modeling of Hydrosystems, Springer-Verlag, 1997.

[54] J. Droniou, Finite volume schemes for diffusion equations: Introduction to and review of modern methods, Math. Models Methods Appl. Sci. 24 (08) (2014) 1575–1619, http://dx.doi.org/10.1142/S0218202514400041.

[55] M. Schneider, K. Weishaupt, D. Gläser, W.M. Boon, R. Helmig, Coupling staggered-grid and mpfa finite volume methods for free flow/porous-medium flow problems, J. Comput. Phys. 401 (2020) 109012, http://dx.doi.org/10.1016/j.jcp.2019.109012.

[56] I. Aavatsmark, An introduction to multipoint flux approximations for quadrilateral grids, Comput. Geosci. 6 (3) (2002) 405–432, http://dx.doi.org/10.1023/A:1021291114475.

[57] M. Schneider, D. Gläser, B. Flemisch, R. Helmig, Comparison of finite-volume schemes for diffusion problems, Oil Gas Sci. Technol. Rev. IFP Energ. Nouvelles 73 (2018) 82, http://dx.doi.org/10.2516/ogst/2018064.

[58] M. Schneider, Nonlinear Finite Volume Schemes for Complex Flow Processes and Challenging Grids (Ph.D. thesis), University of Stuttgart http://dx.doi.org/10.18419/opus-10416.

[59] R. Helmig, B. Flemisch, M. Wolff, A. Ebigbo, H. Class, Model coupling for multiphase flow in porous media, Adv. Water Resour. 51 (2013) 52–66, http://dx.doi.org/10.1016/j.advwatres.2012.07.003, 35th Year Anniversary Issue.

[60] C. Ericson, Real-Time Collision Detection, CRC Press, 2004.

[61] A. Massing, M. Larson, A. Logg, Efficient implementation of finite element methods on nonmatching and overlapping meshes in three dimensions, SIAM J. Sci. Comput. 35 (1) (2013) C23–C47, http://dx.doi.org/10.1137/11085949X.

[62] P. Bastian, G. Buse, O. Sander, Infrastructure for the coupling of dune grids, in: G. Kreiss, P. Lötstedt, A. Målqvist, M. Neytcheva (Eds.), Numerical Mathematics and Advanced Applications 2009, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 107–114.

[63] T. Koch, B. Flemisch, R. Helmig, R. Wiest, D. Obrist, A multiscale subvoxel perfusion model to estimate diffusive capillary wall conductivity in multiple sclerosis lesions from perfusion MRI data, Int. J. Numer. Methods Biomed. Eng. 36 (2) (2020) e3298, http://dx.doi.org/10.1002/cnm.3298.

[64] D. Gläser, B. Flemisch, R. Helmig, H. Class, A hybrid-dimensional discrete fracture model for non-isothermal two-phase flow in fractured porous media, GEM Int. J. Geomath. 10 (1) (2019) 5, http://dx.doi.org/10.1007/s13137-019-0116-8.

[65] J. Hommel, E. Coltman, H. Class, Porosity–permeability relations for evolving pore space: A review with a focus on (bio-)geochemically altered porous media, Transp. Porous Media 124 (2) (2018) 589–629, http://dx.doi.org/10.1007/s11242-018-1086-2.

[66] T. Fetzer, C. Grüninger, B. Flemisch, R. Helmig, On the conditions for coupling free flow and porous-medium flow in a finite volume framework, in: C. Cancès, P. Omnes (Eds.), Finite Volumes for Complex Applications VIII - Hyperbolic, Elliptic and Parabolic Problems, Springer International Publishing, Cham, 2017, pp. 347–356.

[67] G. Yang, E. Coltman, K. Weishaupt, A. Terzis, R. Helmig, B. Weigand, On the Beavers–Joseph interface condition for non-parallel coupled channel flow over a porous structure at high Reynolds numbers, Transp. Porous Media 128 (2) (2019) 431–457, http://dx.doi.org/10.1007/s11242-019-01255-5.

[68] T. Fetzer, K.M. Smits, R. Helmig, Effect of turbulence and roughness on coupled porous-medium/free-flow exchange processes, Transp. Porous Media 114 (2) (2016) 395–424, http://dx.doi.org/10.1007/s11242-016-0654-6.

[69] J.E. Aarnes, S. Krogstad, K.-A. Lie, Multiscale mixed/mimetic methods on corner-point grids, Comput. Geosci. 12 (3) (2008) 297–315, http://dx.doi.org/10.1007/s10596-007-9072-8.

[70] M. Schneider, B. Flemisch, R. Helmig, K. Terekhov, H. Tchelepi, Monotone nonlinear finite-volume method for challenging grids, Comput. Geosci. 22 (2) (2018) 565–586, http://dx.doi.org/10.1007/s10596-017-9710-8.

[71] D.W. Peaceman, Interpretation of well-block pressures in numerical reservoir simulation, Soc. Pet. Eng. J. 18 (03) (1978) 183–194.

[72] C. Geuzaine, J.-F. Remacle, Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities, Internat. J. Numer. Methods Engrg. 79 (11) (2009) 1309–1331, http://dx.doi.org/10.1002/nme.2579.

[73] Gnuplot, URL http://www.gnuplot.info.

[74] J.D. Hunter, Matplotlib: A 2D graphics environment, Comput. Sci. Eng. 9 (3) (2007) 90–95, http://dx.doi.org/10.1109/MCSE.2007.55.

[75] J. Ahrens, B. Geveci, C. Law, Paraview: An end-user tool for large-data visualization, in: C.D. Hansen, C.R. Johnson (Eds.), Visualization Handbook, Butterworth-Heinemann, Burlington, 2005, pp. 717–731, http://dx.doi.org/10.1016/B978-012387582-2/50038-1.

[76] E.A. Truckenbrodt, Fluidmechanik, Springer, 1996.

[77] I. Fatt, et al., The Network Model of Porous Media, Society of Petroleum Engineers, 1956.

[78] M.J. Blunt, Multiphase Flow in Permeable Media: A Pore-Scale Perspective, Cambridge University Press, 2017.

[79] W.J. Layton, F. Schieweck, I. Yotov, Coupling fluid flow with porous media flow, SIAM J. Numer. Anal. 40 (6) (2002) 2195–2218, http://dx.doi.org/10.1137/S0036142901392766.

[80] O. Sander, T. Koch, N. Schröder, B. Flemisch, The Dune FoamGrid implementation for surface and network grids, Arch. Numer. Softw. 5 (1) (2017) 217–244, http://dx.doi.org/10.11588/ans.2017.1.28490.

[81] T.A. Davis, Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method, ACM Trans. Math. Software 30 (2) (2004) 196–199, http://dx.doi.org/10.1145/992200.992206.

[82] V. Martin, J. Jaffré, J.E. Roberts, Modeling fractures and barriers as interfaces for flow in porous media, SIAM J. Sci. Comput. 26 (5) (2005) 1667–1691, http://dx.doi.org/10.1137/S1064827503429363.

[83] R. Ahmed, M. Edwards, S. Lamine, B. Huisman, M. Pal, Control-volume distributed multi-point flux approximation coupled with a lower-dimensional fracture model, J. Comput. Phys. 284 (2015) 462–489, http://dx.doi.org/10.1016/j.jcp.2014.12.047.

[84] B. Flemisch, I. Berre, W. Boon, A. Fumagalli, N. Schwenck, A. Scotti, I. Stefansson, A. Tatomir, Benchmarks for single-phase flow in fractured porous media, Adv. Water Resour. 111 (2018) 239–258, http://dx.doi.org/10.1016/j.advwatres.2017.10.036.

[85] V. Reichenberger, H. Jakobs, P. Bastian, R. Helmig, A mixed-dimensional finite volume method for two-phase flow in fractured porous media, Adv. Water Resour. 29 (7) (2006) 1020–1036, http://dx.doi.org/10.1016/j.advwatres.2005.09.001.

[86] A. Fumagalli, A. Scotti, A numerical method for two-phase flow in fractured porous media with non-matching grids, Adv. Water Resour. 62 (2013) 454–464, http://dx.doi.org/10.1016/j.advwatres.2013.04.001.

[87] M. Tene, S.B. Bosma, M.S.A. Kobaisi, H. Hajibeygi, Projection-based embedded discrete fracture model (pEDFM), Adv. Water Resour. 105 (2017) 205–216, http://dx.doi.org/10.1016/j.advwatres.2017.05.009.

[88] M. Van Genuchten, A closed-form equation for predicting the hydraulic conductivity of unsaturated soils, Soil Sci. Am. J. 44 (1980) http://dx.doi.org/10.2136/sssaj1980.03615995004400050002x.

[89] Y. Mualem, A new model for predicting the hydraulic conductivity of unsaturated porous media, Water Resour. Res. 12 (3) (1976) 513–522, http://dx.doi.org/10.1029/WR012i003p00513.

[90] L. Luckner, M. Van Genuchten, D. Nielsen, A consistent set of parametric models for the two-phase flow of immiscible fluids in the subsurface, Water Resour. Res. 25 (10) (1989) 2187–2193, http://dx.doi.org/10.1029/WR025i010p02187.

[91] W. Wagner, H.-J. Kretzschmar, IAPWS industrial formulation 1997 for the thermodynamic properties of water and steam, in: International Steam Tables, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 7–150.

[92] R. Reid, J. Prausnitz, B. Poling, The Properties of Gases and Liquids, McGraw-Hill Inc., 1987.

[93] N. Schröder, Three-Dimensional Solute Transport Modeling in Coupled Soil and Plant Root Systems, Vol. 22 (Ph.D. thesis), Université catholique de Louvain (UCL), Louvain-La-Neuve, Jülich, 2014.

[94] C. D'Angelo, A. Quarteroni, On the coupling of 1D and 3D diffusion-reaction equations: Application to tissue perfusion problems, Math. Models Methods Appl. Sci. 18 (08) (2008) 1481–1504, http://dx.doi.org/10.1142/S0218202508003108.

[95] H.-J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, B. Uekermann, PreCICE – A fully parallel library for multi-physics surface coupling, in: Advances in Fluid-Structure Interaction, Comput. & Fluids 141 (2016) 250–258, http://dx.doi.org/10.1016/j.compfluid.2016.04.003.