Technical university of Liberec

Faculty of mechatronics, informatics and interdisciplinary studies

Flow123d

version 3.9.0

User Guide and Input Reference

Liberec, 2022

Authors:

Jan Březina, Jan Stebel, David Flanderka, Pavel Exner

Contents

Chapter 1

Getting Started

1.1 Introduction

Flow123d is a software for simulating water flow, reactionary solute transport, heat transfer, and mechanics in a heterogeneous porous and fractured medium. In particular, it is well-suited for simulations of underground processes in a granite rock. The program is able to describe explicitly processes in 3D medium, 2D fractures, and 1D channels and exchange between domains of different dimensions. Therefore, the computational mesh is a collection of tetrahedra, triangles, and line segments.

Two water flow models are available: a model for a saturated medium based on the Darcy law and a model for a partially saturated medium described by Richards' equation. Both models use the mixed-hybrid finite element method for the space discretization and the implicit Euler method for the time discretization. Both models can also switch between a transient case and a sequence of steady states within a single simulation. The model for unsaturated medium uses a lumped variant of the mixed-hybrid method to satisfy the maximum principle and to guarantee stability for short time steps.

In the current version, only the model for saturated media can be sequentially coupled with the transport models, including two models for the solute transport and one model for the heat transfer.

The first solute transport model can deal only with a pure advection of several substances without any diffusion-dispersion term. It uses the explicit Euler method for the time discretization and the finite volume method for the space discretization. The second solute transport model describes a general advection with hydrodynamic dispersion for several substances. It uses the implicit Euler method for time discretization and the discontinuous Galerkin method of the first, second, or third order for the discretization in space. The operator splitting method can be used to couple any of these two solute transport models with various processes described by the reaction term. The reaction term can treat any meaningful combination of the dual porosity, equilibrium sorptions, decays, and linear reactions.

The heat transfer model assumes equilibrium between the temperature of the rock and of the fluid phase. It uses the same numerical scheme as the second transport model, i.e., the DG method with the implicit time discretization.

The mechanical model computes linear elastic problems, taking into account the reduced

dimension approach. It can be coupled with the flow model and solve linear poroelastic problems. This coupling is realized by an iterative scheme. Nonlinear dependency between strain and hydraulic conductivity and nonlinearity due to contacts and shear stress effects is the focus of ongoing development.

The program supports the output of all input and output fields into two file formats. One can use the GMSH mesh generator and post-processor file format or the output into widely supported VTK format. In particular, we recommend Paraview software for the visualization and post-processing of the VTK data.

The program is implemented in C/C++ using the PETSc library for linear algebra. All models can run in parallel using the MPI environment; however, the scalability of the whole program is limited due to serial mesh data structures and serial outputs.

The program is distributed under GNU GPL v. 3 license and is available on the project web page: http://flow123d.github.io

with sources on the GitHub: https://github.com/flow123d/flow123d.

1.2 Reading Documentation

The Flow123d documentation has two main parts. Chapters 1 to 5 form a user manual, while the last Chapter 6 provides an input reference. The user manual starts with Chapter 1 giving instructions for the installation and execution of the program. The Chapter 2 provides a detailed description of the implemented mathematical models. The Chapter 3 presents used numerical methods. The input and output file formats are documented in Chapter 4. Finally, Chapter 5 consists of tutorial problems.

The input reference guide, consisting only of Chapter 6, is automatically generated. It mirrors the code directly and describes the whole structure of the main input file. Description of input records, their structure, and default values are supplied there, and bidirectional links to the user manual are provided.

The document is interactive; the blue text marks the links in the document, and the magenta text marks the web links.

1.3 Installing Flow123d

Software Flow123d requires the tool Docker. Docker is an open-source project that automates the deployment of Linux applications inside software containers. Entire Flow123d software is packed in a Docker image that also contains necessary libraries and crucial components of the Linux operating system.

The installation process imports the Docker image into the user's machine and personalizes the Docker image. The installation instructions for Linux and Windows operating systems are provided in the two following sections.

1.3.1 Installing Flow123d on Linux

The installation is done under a regular user, who must be in the group 'docker'. Download the Linux installation package archive

<div align="center">

`flow123d_<version>_linux_install.tar.gz`

</div>

from Official pages and extract it to any folder:

```
$> tar -xzf flow123d_<version>_linux_install.tar.gz
```

This will create a directory `flow123d_<version>`. In the next step, navigate to this directory and execute the `install.sh` script. Example output:

```
$> cd flow123d_3.0.4
$> ./install.sh
Pulling docker image 'flow123d/3.0.4'
...
flow123d/3.0.4
Installation of Flow123d finished successfully.
Run Flow123d using script fterm.sh or flow123d.sh in bin folder.
You can start by printing the version of Flow123d:
  bin/fterm.sh run --version
```

The install script will download Docker image from the Docker Hub. Whole process may take several minutes (depending on user's machine performance and internet connectivity).

1.3.2 Installing Flow123d on Windows

Before you install

This version uses `Docker Desktop`, previous versions which used `Docker for Windows` and `Docker Toolbox` will **stop working**.

Make sure your system fullfills following requirements in order to support `Docker Desktop`:

- Windows 10 64bit: Pro, Enterprise or Education (1607 Anniversary Update, Build 14393 or later).

- Virtualization is enabled in BIOS. Typically, virtualization is enabled by default. This is different from having Hyper-V enabled. For more detail see Virtualization must be enabled in Troubleshooting.

- CPU SLAT-capable feature.

- At least 4GB of RAM.

Installation

To install Flow123d on Windows, download installer from Official pages, execute it and follow instructions on your screen. To make things easier, you can also watch a installation video.

If for some reason the installation failed, make sure everything below is in order:

Flow123d troubleshooting

- **Powershell is installed**: On the Windows systems we require PowerShell. Windows PowerShell needs to be installed on Windows Server 2008 and Windows Vista only. It is already installed on Windows Server 2008 R2 and Windows 7 and higher. To install PowerShell follow instructions at Microsoft pages.

- **Powershell is in the system PATH**: Make sure `powershell` command is in the system PATH. PowerShell executable location is specific to the particular Windows version, but usual location is:

  `%SystemRoot%\system32\WindowsPowerShell\v1.0\powershell.exe`

  To add this location to the system PATH variable follow the instructions at Microsoft pages.

- For detailed instructions refer to Docker docs.

Uninstalling Flow123d

To uninstall Flow123d, in Windows open `Apps & features` (`Aplikace a funkce` in Czech), find the Flow123d in the list and click uninstall. This will only uninstall the Flow123d but not `Docker Desktop`.

Reinstalling Flow123d

If you are installing same version of Flow123d again, installation process will be the same, except `Docker Desktop` installation will be skipped.

1.4 Running Flow123d

1.4.1 Running Flow123d on Linux

All necessary scripts for running Flow123d are located in the `bin` subdirectory of the installation directory `flow123d_<version>`. From the user's perspective, an unpleasant issue with the Docker container is that it cannot easily interact with the host file system by default. Using the supplied scripts in `bin` will make things a lot easier for the user. The directory `bin` contains:
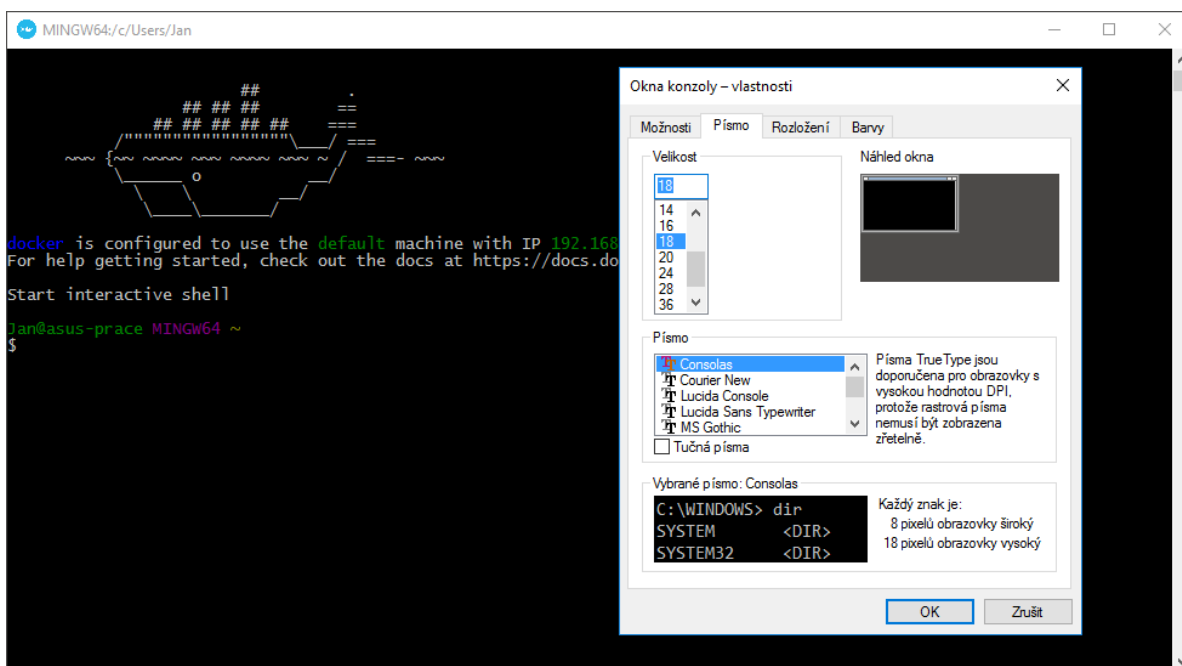
Figure 1.1: Changing default font family and font size

- `fterm.sh`
  The script will invoke a shell inside the Docker container and mount the user's home directory. In this shell, the user has access to the system path where Flow123d is installed. By default, the command `flow123d` is in the `PATH` variable.

  Note: *On some systems, the shell's font is extremely small, you can change this behavior by right-clicking on the window bar and selecting* `default` *or (*`vychozi` *in Czech) see* *Figure 1.1**.*

- `flow123d.sh`
  The script will run Flow123d inside the Docker container and mount the user's home directory. All arguments passed to this script will be passed to `flow123d` binary file inside the container.

- `runtest.sh`
  The script will run Flow123d tests inside the Docker container and mount the user's home directory. All arguments passed to this script will be passed to `runtest.py` Python script inside the container.

Note: *Using the above* `.sh` *scripts will mount the user's home directory to the Docker container under the same name. Also your current working directory will be the same. The example below shows the behavior of the scripts:*

```
$> pwd
/home/username/local/flow123d_3.0.4

$> ls
bin  doc  install.sh  tests uninstall.sh
```

9

```
$> bin/fterm.sh

 ___ _          _ ___ ____   _
| __| |_____ __ _/ |_  )__ / __| |
| _|| / _ \ V  V / |/ / |_ \/ _  |
|_| |_\___/\_/\_/|_/___|___/\__,_|
                   3.0.4
flow@flow:3.0.4 /home/username/local/flow123d_3.0.4$> ls
bin  doc  install.sh  tests uninstall.sh
```

1.4.2   Running Flow123d on Windows

On system Windows you will have a shortcut on your desktop, to verify everything is working. To run flow123d from anywhere simple type `flow123d.bat` or `fterm.bat` (in terminal, powershell, Total Commander, . . .).

Each bat file serves a different purpose:

- `flow123d.bat` serves as a binary, it possible to run the bat file multiple times (useful for a batch processing)

- `fterm.bat` serves as an interactive shell console session invoked inside Docker container.

By default `flow123d.bat` will run the last installed version on your system. If you have multiple version installed and want to run specific one, each version has a unique bat files `flow123d-<version>.bat` and `fterm-<version>.bat` file (e.g. `flow123d-3.0.9.bat` and `fterm-3.0.9.bat`).

Running from other batch file

The Windows system calls the batch files in the different way then the binaries. In particular the calling batch file is not processed further after the child batch file is done. In order to do so, one have to use the `CALL` command. This is especially necessary for various calibration tools. The correct calling batch file may look like:

```
echo "Starting Flow123d ..."
call flow123d.bat a_simulation.yaml
echo "... simulation done."
```

Adjusting memory of virtual machine

To change the memory limits of the Virtual machine, open the Docker Settings dialog (right click on the whale icon) and select `Settings`. Navigate to `Advanced` tab and adjust the memory. Detailed instructions can be found Docker docs.

1.5 Flow123d arguments

When you are inside of the Docker container, you have access to the entire file system of the container. Flow123d is installed in `/opt/flow123d` directory. Folder `bin` contains binary files and is automatically added to the variable `PATH`, so that every executable in this folder can be called from anywhere.

The main Flow123d binary is located in `bin/flow123d` and accepts the following arguments:

`--help`
> Help for parameters interpreted by Flow123d. Remaining parameters are passed to PETSC.

`-s, --solve  <file>`
> Set principal input file. Can be in YAML (or JSON) file format. All relative paths of the input files are relative to the location of the principal input file.

`-i, --input_dir <directory>`
> The placeholder `${INPUT}` used in the path of an input file will be replaced by the `<directory>`. Default value is `input`.

`-o, --output_dir <directory>`
> All paths for output files will be relative to this `<directory>`. Default value is `output`.

`-l, --log <file_name>`
> Set base name of log files. Default value is `flow123d`. The log files are individual for every MPI process, placed in the output directory. The MPI rank of the process and the `log` suffix are appended to the base name.

`--version`
> Display version and build information.

`--no_log`
> Turn off logging.

`--no_profiler`
> Turn off profiler output.

`--petsc_redirect <file>`
> Redirect all PETSc stdout and stderr to given file.

`--input_format`
> Print a description of the main input file in JSON format. This is used by GeoMop model editor and by python scripts for generating reference documentation in Latex or HTML format.

`--yaml_balance`
> Generate balance file also in machine readable YAML format. Will be default in future, used by GeoMop.

```
--no_signal_handler
```
     For debugging purpose.

The `-s` identifier can be skipped:

```
flow123d <main_input>.yaml <options> <PETSC options>
```

Any different parameters are passed to the PETSC library. An advanced user can influence lot of parameters of linear solvers. In order to get list of supported options use parameter `-help` together with a valid input file. Options for various PETSC modules are displayed when the module is used for the first time.

Alternatively, you can use python script `exec_parallel` located in `bin/python` to start parallel jobs or limit resources used by the program.

After double dash specify which `mpiexec` binary will be used (`MPI-EXECUTABLE`) and then specify what should be executed. The script does not need to run solely `flow123d`.

If we want to run command `whoami` in parallel we can do:

```
bin $> exec_parallel -n 4 -- ./mpiexec whoami
```

To execute Flow123d in parallel we can do:

```
bin $> exec_parallel -n 4 -- ./mpiexec ./flow123d --help
```

```
exec_parallel [OPTIONS] -- [MPI-EXECUTABLE] [PARAMS]
```

The script has following options:

```
-h, --help
```
     Usage overview.

```
--host <hostname>
```
     Valid only when option `--queue` is set. Default value is the host name obtained by python `platform.node()` call, this argument can be used to override it. Resulting value is used to select a correct PBS module from lookup table `config/host_table.yaml`.

```
-n <number of processes>
```
     Specify number of MPI parallel processes for calculation.

```
-t, --limit-time <timeout>
```
     Upper estimate for real running time of the calculation. Kill calculation after *timeout* seconds. Value can also be `float` number. When in PBS mode, value can also affect PBS queue.

```
-m, --limit-memory <memory limit>
```
     Limits total available memory to `<memory limit>` MB in total.

```
-q, --queue <queue>
     If set activates PBS mode. If argument queue is also set selects particular job
     queue on PBS systems otherwise default PBS queue is used. Default PBS queue
     automatically choose valid queue based on resources.
```

Another script which runs Flow123d is `runtest.sh`. This script will run tests speci-
fied as arguments. Script accepts both folders and yaml files. To see full details run
`runtest.sh --help`. The script will run yaml tests and then compare results with
reference output. Example usage of the script:

```
$> bin/runtest.sh tests/10_darcy/01_source.yaml
configuration:
----------------------------------------------------------------
  yaml files       1
  total cases      2
----------------------------------------------------------------
execution:
  done [01 of 02] 1 x 10_darcy/01_source [00:03.070] [31 MiB] exitcode 0
  done [02 of 02] 2 x 10_darcy/01_source [00:01.378] [31 MiB] exitcode 0
----------------------------------------------------------------
SUMMARY
----------------------------------------------------------------
  [ PASSED ]  | 1 x 10_darcy/01_source                        [ 2.76 sec]
  [ PASSED ]  | 2 x 10_darcy/01_source                        [ 1.07 sec]
  -----------------------------------------------------------
  [ PASSED ]  | passed=2, failed=0, skipped=0 in              [ 3.83 sec]
----------------------------------------------------------------
```

1.6   Tutorial Problem

In the following section, we shall provide an example cookbook for preparing and running
a model based on one of the test problems, namely

> `tests/21_solute_fv_frac/03_fv_dp_sorp_small.yaml.`

We shall start with the preparation of the geometry using external software, and then
we shall go step by step through the commented main input file. The problem includes
steady Darcy flow, transport of two substances with explicit time discretization, and
a reaction term consisting of dual porosity and sorption models. More tutorials focused
on particular features can be found in Chapter 5.

1.6.1   Geometry

We consider a simple 2D problem with a branching 1D fracture (see Figure 1.2 for the
geometry). To prepare a mesh file we use the GMSH software. First, we construct
a geometry file. In our case the geometry consists of:

- one physical 2D domain corresponding to the whole square

- three 1D physical domains of the fracture

- four 1D boundary physical domains of the 2D domain

- three 0D boundary physical domains of the 1D domain

In this simple example, we can in fact combine physical domains in every group, however we use this more complex setting for demonstration purposes. Using GMSH graphical interface we can prepare the GEO file where physical domains are referenced by numbers, then we use any text editor and replace numbers with string labels in such a way that the labels of boundary physical domains start with the dot character. These are the domains where we will not do any calculations but we will use them for setting boundary conditions. Finally, we get the GEO file like this:

```
1   cl1 = 0.16;
2   Point(1) = {0, 1, 0, cl1};
3   Point(2) = {1, 1, 0, cl1};
4   Point(3) = {1, 0, 0, cl1};
5   Point(4) = {0, 0, 0, cl1};
6   Point(6) = {0.25, -0, 0, cl1};
7   Point(7) = {0, 0.25, 0, cl1};
8   Point(8) = {0.5, 0.5, -0, cl1};
9   Point(9) = {0.75, 1, 0, cl1};
10  Line(19) = {9, 8};
11  Line(20) = {7, 8};
12  Line(21) = {8, 6};
13  Line(22) = {2, 3};
14  Line(23) = {2, 9};
15  Line(24) = {9, 1};
16  Line(25) = {1, 7};
17  Line(26) = {7, 4};
18  Line(27) = {4, 6};
19  Line(28) = {6, 3};

20  Line Loop(30) = {20, -19, 24, 25};
21  Plane Surface(30) = {30};
22  Line Loop(32) = {23, 19, 21, 28, -22};
23  Plane Surface(32) = {32};
24  Line Loop(34) = {26, 27, -21, -20};
25  Plane Surface(34) = {34};
26  Physical Point(".1d_top") = {9};
27  Physical Point(".1d_left") = {7};
28  Physical Point(".1d_bottom") = {6};
29  Physical Line("1d_upper") = {19};
30  Physical Line("1d_lower") = {21};
31  Physical Line("1d_left_branch") = {20};
32  Physical Line(".2d_top") = {23, 24};
33  Physical Line(".2d_right") = {22};
34  Physical Line(".2d_bottom") = {27, 28};
35  Physical Line(".2d_left") = {25, 26};
36  Physical Surface("2d") = {30, 32, 34};
```

Notice the labeled physical domains on lines 26 – 36. Then we just set the discretization step `cl1` and use GMSH to create the mesh file. The mesh file contains both the 'bulk' elements where we perform calculations and the 'boundary' elements (on the boundary physical domains) where we only set the boundary conditions.

1.6.2   YAML File Format

The main input file uses the YAML file format with some restrictions. We prefer to call YAML objects *records* and we introduce also *abstract records* that mimic C++ abstract classes. Arrays have only elements of the same type (possibly using abstract record types for polymorphism). The usual keys are in lower case and without spaces (using underscores instead). For detailed description see Section 4.1.

Having the computational mesh from the previous step, we can create the main input file with the description of our problem.

14

```
1   flow123d_version: 3.1.0
2   problem: !Coupling_Sequential
3     description: 'Transport 1D-2D (convection, reaction term, sources).'
4     mesh:
5       mesh_file: ../00_mesh/square_1x1_frac_fork.msh
6       regions:
7         - !Union
8           name: 1d_domain
9           regions:
10            - 1d_upper
11            - 1d_lower
12            - 1d_left_branch
```

The version of Flow123d for which the file is a valid input is specified at the first line.
Then the description starts with a selection of the problem type (`Coupling_Sequential`),
and a textual problem description. Next, the computational mesh is defined; here it con-
sists of the name of the mesh file and the declaration of one region given as the union of
all 1D regions, i.e., representing the whole fracture. Other keys of the `mesh` record allow
labeling regions given only by numbers, defining new regions in terms of element num-
bers (e.g to have a leakage on single element), defining boundary regions, and several
operations with region sets, see Section 4.2.1 for details.

1.6.3 Flow Setting

Next, we setup the flow problem. We shall consider a steady flow (i.e. with zero stora-
tivity) driven only by the pressure gradient (no gravity), setting the Dirichlet bound-
ary condition on the whole boundary with the pressure head equal to $x + y$. The
`conductivity` will be $k_2 = 10^{-7}$ ms^{-1} on the 2D domain and $k_1 = 10^{-6}$ ms^{-1} on the 1D
domain. We leave the default value for the `cross_section` of the 2D domain, meaning
that the thickness of 2D domain is $\delta_2 = 1$ m. For the 1D fractures cross-section, we pre-
scribe $\delta_1 = 0.04$ m^2 on the 1D domain. The transition coefficient σ_2 between dimensions
can be scaled by setting the dimensionless parameter σ_{21} (`sigma`). This can be used for
simulating additional effects which prevent the liquid transition from/to a fracture, like
a thin resistance layer. Notice that the scaling parameter is set on the lower dimensional
domain, i.e. $\sigma_{21} = 0.9$ on line 18. Read Section 2.3 for more details.

```
13    flow_equation: !Flow_Darcy_MH
14      input_fields:
15        - region: 1d_domain
16          conductivity: 1.0e-06
17          cross_section: 0.04
18          sigma: 0.9
19        - region: 2d
20          conductivity: 1.0e-07
21        - region: .BOUNDARY
22          bc_type: dirichlet
23          bc_pressure: !FieldFormula
24            value: x+y
```

```
25      nonlinear_solver:
26        linear_solver: !Petsc
27          a_tol: 1.0e-12
28          r_tol: 1.0e-12
29      output:
30        fields:
31          - pressure_p0
32          - velocity_p0
33      output_stream:
34        file: flow.pvd
35        format: !vtk
36          variant: ascii
```

On line 13, we specify particular implementation (numerical method) of the flow solver, in this case the Mixed-Hybrid solver for steady problems. On lines 14 – 24 in the array `input_fields`, we set both mathematical fields that live on the computational domain and those defining the boundary conditions. We use implicitly defined region `.BOUNDARY` that contains all boundary regions and we set there Dirichlet boundary condition in terms of the pressure head. In this case, the field is not of the implicit type `FieldConstant`, so we must specify the type of the field `!FieldFormula`. See Section 4.2.2 for other field types. Next, we specify the type of the linear solver and its tolerances. On lines 29 – 32, we specify which output fields should be written to the output stream (that means particular output file, with given format). See Section 4.4 for the list of available output `fields`. Currently, we support only one `output_stream` per equation. We specify the filename and the format of the output stream (the used ASCII VTK format is the default).

1.6.4 Transport Setting

The flow model is followed by a transport model in the record `solute_equation` beginning on line 37. Here, we use an implementation called Solute_Advection_FV which stands for an explicit finite volume solver of the convection equation (without diffusion). The operator splitting method is used for equilibrium sorption as well as for dual porosity model and first order reactions simulation.

```
37    solute_equation: !Coupling_OperatorSplitting
38      substances:
39        - name: age # water age
40          molar_mass: 0.018
41        - name: U235 # uranium 235
42          molar_mass: 0.235
43      transport: !Solute_Advection_FV
44        input_fields:
45          - region: ALL
46            init_conc: 0
47            porosity: 0.25
48            # source is in the whole volume (l+s) -> times porosity
49            sources_density:
```

```
50              - 0.25
51              - 0
52          - region: .BOUNDARY
53            bc_conc:
54              - 0.0
55              - 1.0
56      time:
57        end_time: 1000000
58      balance:
59        cumulative: true
```

On lines 38 – 42, we set the transported substances, which are identified by their names. Here, the first one is the age of the water, with the molar mass of water, and the second one U235 is the uranium isotope 235. On lines 44 – 55, we set the input fields, in particular zero initial concentration for all substances, porosity $\theta = 0.25$ and sources of concentration by sources_density. Notice line 46 where we can see only single value since an automatic conversion is applied to turn the scalar zero into the zero vector (of size 2 according to the number of substances).

The boundary fields are set on lines 52 – 55. We do not need to specify the type of the condition since there is only one type in the Solute_Advection_FV transport model. The boundary condition is equal to 1 for the uranium 235 and 0 for the age of the water and is automatically applied only on the inflow part of the boundary.

We also have to prescribe the time setting – here it is only the end time of the simulation (in seconds: 10^6 s $\approx$ 11.57 days). The step size is determined automatically from the CFL condition, however, a smaller time step can be enforced if necessary.

Reaction term of the transport model is described in the next subsection, including dual porosity and sorption.

1.6.5 Reaction Term

The input information for dual porosity, equilibrial sorption and possibly first order reactions are enclosed in the record reaction_term, lines 60 – 104. Go to section 2.5 to see how the models can be chained.

The type of the first process is determined by !DualPorosity, on line 60. The input_fields of the dual porosity model are set on lines 61 – 69 and the output is disabled by setting an empty array on line 71.

```
60      reaction_term: !DualPorosity
61        input_fields:
62          - region: ALL
63            diffusion_rate_immobile:
64              - 0.01
65              - 0.01
66            porosity_immobile: 0.25
67            init_conc_immobile:
68              - 0.0
```

```
 69              - 0.0
 70        output:
 71          fields: []
 72        reaction_mobile: !SorptionMobile
 73          solvent_density: 1000.0 # water
 74          substances:
 75            - age
 76            - U235
 77          solubility:
 78            - 1.0
 79            - 1.0
 80          input_fields: &anchor1
 81            - region: ALL
 82              rock_density: 2800.0 # granite
 83              sorption_type:
 84                - none
 85                - freundlich
 86              distribution_coefficient:
 87                - 0
 88                - 1.598e-4
 89              isotherm_other:
 90                - 0
 91                - 1.0
 92        output:
 93          fields: []
 94      reaction_immobile: !SorptionImmobile
 95        solvent_density: 1000.0 # water
 96        substances:
 97          - age
 98          - U235
 99        solubility:
100          - 1.0
101          - 1.0
102        input_fields: *anchor1
103        output:
104          fields: []
105    output_stream:
106      file: transport.pvd
107      format: !vtk
108        variant: ascii
109      times:
110        - step: 100000.0
```

Next, we define the equilibrial sorption model such that `SorptionMobile` type takes place in the mobile zone of the dual porosity model while `SorptionImmobile` type takes place in its immobile zone, see lines 72 and 94. Isothermally described sorption simulation can be used in the case of low concentrated solutions without competition between multiple dissolved species.

On lines 73 – 79, we set the sorption related input information. The solvent is water so the `solvent_density` is supposed to be constant all over the simulated area. The vector `substances` contains the list of names of solute substances which are considered to be affected by the sorption. Solubility is a material characteristic of a sorbing substance related to the solvent. Elements of the vector `solubility` define the upper bound of aqueous concentration which can appear. This constrain is necessary because some substances might have limited solubility and if the solubility exceeds its limit they start to precipitate. `solubility` is a crucial parameter for solving a set of nonlinear equations, described further.

The record `input_fields` covers the region specific parameters. All implemented types of sorption can take the rock density in the region into account. The value of `rock_density` is a constant in our case. The `sorption_type` represents the empirically determined isotherm type and can have one of four possible values: {`"none"`, `"linear"`, `"freundlich"`, `"langmuir"`}. Linear isotherm needs just one parameter given whereas Freundlich and Langmuir isotherms require two parameters. We will use Freundlich isotherm for demonstration but we will set the other parameter (exponent) $\alpha = 1$ which means it will be the same as the linear type. We use value $K_d = 1.598 \cdot 10^{-4}$ kg^{-1}m^3 for the `distribution_coefficient` according to (www.skb.se, report R-10-48 by James Crawford, 2010).

On line 102, notice the reference pointing to the definition of input fields on lines 80 – 91. Only entire records can be referenced which is why we have to repeat parts of the input such as solvent density and solubility (records for reaction mobile and reaction immobile have different types).

On lines 93 and 104, we define which sorption specific outputs are to be written to the output file. An implicit set of outputs exists. In this case we define an empty set of outputs thus overriding the implicit one. This means that no sorption specific outputs will be written to the output file. On lines 105 – 110 we specify which output fields should be written to the output stream. Currently, we support output into VTK and GMSH data format. In the output record for time-dependent process we have to specify the time `step` (line 110) which determines the frequency of output data writing.

1.6.6 Results

In Figure 1.2 one can see the results: the pressure and the velocity field on the left and the concentration of U235 at time $t = 9 \cdot 10^5$ s on the right. Even though the pressure gradient is the same both in the 2D domain and in the fracture, the velocity field is ten times faster in the fracture due to its higher conductivity. Since porosity is the same, the substance is transported faster by the fracture. Therefore nonzero concentration appears in the bottom left of the 2D domain long before the main wave propagating solely through the 2D domain reaches that corner.
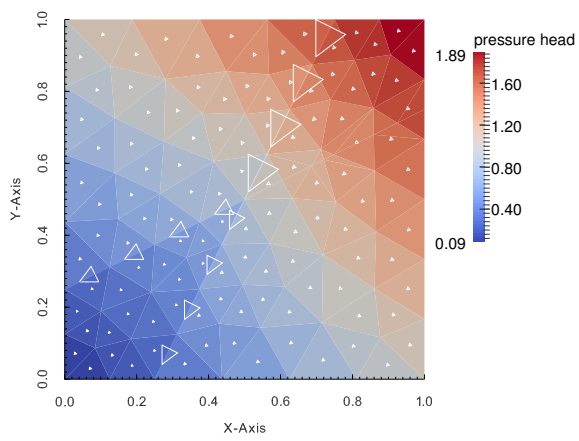
(a) Elementwise pressure head and velocity field denoted by triangles. (Steady flow.)
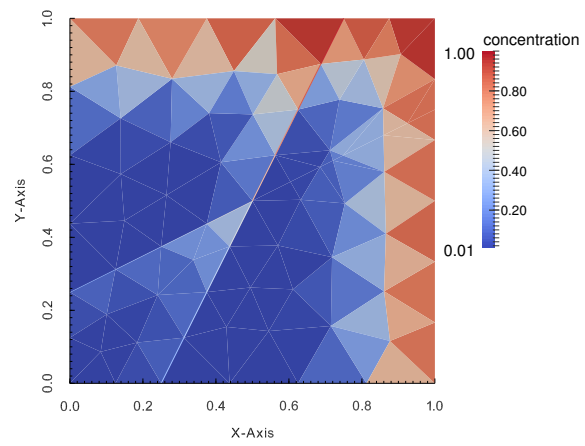
(b) Propagation of U235 from the inflow part of the boundary. (At the time $9 \cdot 10^5$ s.)

Figure 1.2: Results of the tutorial problem.

Chapter 2

Mathematical Models
of Physical Reality

In this chapter we describe mathematical models used in Flow123d. Then in chapter 4 we briefly describe structure of individual input files, in particular the main YAML file. The complete description of the YAML format is given in chapter 6.

Flow123d provides models for Darcy flow in porous media as well as for the transport and reactions of solutes. In this section, we describe mathematical formulations of these models together with physical meaning and units of all involved quantities. In the first section we present basic notation and assumptions about computational domains and meshes that combine different dimensions. In the next section we derive approximation of thin fractures by lower dimensional interfaces for a general transport process. Latter sections describe details for models of particular physical processes.

2.1   Meshes of Mixed Dimension

Unique feature common to all models in Flow123d is the support of domains with mixed dimension. Let $\Omega_3 \subset \mathbf{R}^3$ be an open set representing continuous approximation of porous and fractured medium. Similarly, we consider a set of 2D manifolds $\Omega_2 \subset \overline{\Omega}_3$, representing the 2D fractures and a set of 1D manifolds $\Omega_1 \subset \overline{\Omega}_2$ representing the 1D channels or preferential paths (see Fig 2.1). We assume that Ω_2 and Ω_1 are polytopic (i.e. polygonal and piecewise linear, respectively). For every dimension $d = 1, 2, 3$, we introduce a triangulation $\mathcal{T}_d$ of the open set Ω_d that consists of finite elements T_d^i, $i = 1, \ldots, N_E^d$. The elements are simplices, i.e. lines, triangles and tetrahedra, respectively.

Present numerical methods used by the software require meshes satisfying the compatibility conditions

$$T_{d-1}^i \cap T_d \subset \mathcal{F}_d, \qquad \text{where } \mathcal{F}_d = \bigcup_k \partial T_d^k \tag{2.1}$$

and

$$T_{d-1}^i \cap \mathcal{F}_d \text{ is either } T_{d-1}^i \text{ or } \emptyset \tag{2.2}$$

for every $i \in \{1, \ldots, N_E^{d-1}\}$, $j \in \{1, \ldots, N_E^d\}$, and $d = 2, 3$. That is, the $(d-1)$-dimensional elements are either between d-dimensional elements and match their sides or they poke out of Ω_d. Support for a coupling between non-compatible meshes of

Figure 2.1: Scheme of a problem with domains of multiple dimensions.

different dimensions is under development and partially supported by the Darcy Flow model.

2.2 Advection-Diffusion Processes on Fractures

This section presents derivation of an abstract advection-diffusion process on 2D and 1D manifolds and its coupling with the higher dimensional domains. The reader not interested in the details of this approximation may skip directly to the later sections describing mathematical models of individual physical processes.

As was already mentioned, the unique feature of Flow123d is support of models living on 2D and 1D manifolds. The aim is to capture features significantly influencing the solution despite of their small cross-section. Such a tiny features are challenging for numerical simulations since a direct discretization requires highly refined computational mesh. One possible solution is to model these features (fractures, channels) as lower dimensional objects (2D and 1D manifolds) and introduce their coupling with the surrounding continuum. The equations modeling a physical process on a manifold as well as its coupling to the model in the surrounding continuum has to be derived from the model on the 3D continuum. This section presents such a procedure for the case of the abstract advection-diffusion process inspired by the paper [8]. Later, we apply this abstract approach to particular advection-diffusion processes: Darcy flow, solute transport, and heat transfer.

Let us consider a fracture as a strip domain

$$\Omega_f \subset [0, \delta] \times \mathbf{R}^{d-1}$$

for $d = 2$ or $d = 3$ and surrounding continuum domains

$$\Omega_1 \subset (-\infty, 0) \times \mathbf{R}^{d-1}, \Omega_2 \subset (\delta, \infty) \times \mathbf{R}^{d-1}.$$

Further, we denote by γ_i, $i = 1, 2$ the fracture faces common with domains Ω_1 and Ω_2 respectively. By x, $\boldsymbol{y}$ we denote normal and tangential coordinate of a point in Ω_f. We consider the normal vector $\boldsymbol{n} = \boldsymbol{n}_1 = -\boldsymbol{n}_2 = (1, 0, 0)^\top$. An advection-diffusion process

is given by equations:

$$\partial_t w_i + \operatorname{div} \boldsymbol{j}_i = f_i \qquad\qquad \text{on } \Omega_i, \ i = 1, 2, f, \qquad (2.3)$$

$$\boldsymbol{j}_i = -\mathbb{A}_i \nabla u_i + \boldsymbol{b}_i w_i \qquad\qquad \text{on } \Omega_i, \ i = 1, 2, f, \qquad (2.4)$$

$$u_i = u_f \qquad\qquad \text{on } \gamma_i, \ i = 1, 2, \qquad (2.5)$$

$$\boldsymbol{j}_i \cdot \boldsymbol{n} = \boldsymbol{j}_f \cdot \boldsymbol{n} \qquad\qquad \text{on } \gamma_i, \ i = 1, 2, \qquad (2.6)$$

where $w_i = w_i(u_i)$ is the conservative quantity and u_i is the principal unknown, $\boldsymbol{j}_i$ is the flux of w_i, f_i is the source term, $\mathbb{A}_i$ is the diffusivity tensor and $\boldsymbol{b}_i$ is the velocity field. We assume that the tensor $\mathbb{A}_f$ is symmetric positive definite with one eigenvector in the direction $\boldsymbol{n}$. Consequently the tensor has the form:

$$A_f = \begin{pmatrix} a_n & 0 \\ 0 & \mathbb{A}_t \end{pmatrix}$$

Furthermore, we assume that $\mathbb{A}_f(x, \boldsymbol{y}) = \mathbb{A}_f(\boldsymbol{y})$ is constant in the normal direction.

Our next aim is to integrate equations on the fracture Ω_f in the normal direction and obtain their approximations on the surface $\gamma = \Omega_f \cap \{x = \delta/2\}$ running through the middle of the fracture. For the sake of clarity, we will not write subscript f for quantities on the fracture. To make the following procedure mathematically correct we have to assume that functions $\partial_x w$, $\partial_x \nabla_y u$, $\partial_x \boldsymbol{b_y}$ are continuous and bounded on Ω_f. Here and later on $\boldsymbol{b_x} = (\boldsymbol{b} \cdot \boldsymbol{n}) \boldsymbol{n}$ is the normal part of the velocity field and $\boldsymbol{b_y} = \boldsymbol{b} - \boldsymbol{b_x}$ is the tangential part. The same notation will be used for normal and tangential part of the field $\boldsymbol{q}$.

We integrate (2.3) over the fracture opening $[0, \delta]$ and use approximations to get

$$\partial_t(\delta W) - \boldsymbol{j}_2 \cdot \boldsymbol{n}_2 - \boldsymbol{j}_1 \cdot \boldsymbol{n}_1 + \operatorname{div} \boldsymbol{J} = \delta F, \qquad (2.7)$$

where for the first term, we have used mean value theorem, first order Taylor expansion, and boundedness of $\partial_x w$ to obtain approximation:

$$\int_0^\delta w(x, \boldsymbol{y}) \, \mathrm{d}x = \delta w(\xi_{\boldsymbol{y}}, \boldsymbol{y}) = \delta W(\boldsymbol{y}) + O(\delta^2 |\partial_x w|),$$

where

$$W(\boldsymbol{y}) = w(\delta/2, \boldsymbol{y}) = w(u(\delta/2, \boldsymbol{y})) = w(U(\boldsymbol{y})).$$

Next two terms in (2.7) come from the exact integration of the divergence of the normal flux $\boldsymbol{j_x}$. Integration of the divergence of the tangential flux $\boldsymbol{j_y}$ gives the fourth term, where we introduced

$$\boldsymbol{J}(\boldsymbol{y}) = \int_0^\delta \boldsymbol{j_y}(x, \boldsymbol{y}) \, \mathrm{d}x.$$

In fact, this flux on γ is scalar for the case $d = 2$. Finally, we integrate the right-hand side to get

$$\int_0^\delta f(x, \boldsymbol{y}) \, \mathrm{d}x = \delta F(\boldsymbol{y}) + O(\delta^2 |\partial_x f|), \quad F(\boldsymbol{y}) = f(\delta/2, \boldsymbol{y}).$$

Due to the particular form of the tensor $\mathbb{A}_f$, we can separately integrate tangential and normal part of the flux given by (2.4). Integrating the tangential part and using approximations

$$\int_0^\delta \nabla_{\boldsymbol{y}} u(x, \boldsymbol{y}) \, \mathrm{d}x = \delta \nabla_{\boldsymbol{y}} u(\xi_{\boldsymbol{y}}, \boldsymbol{y}) = \delta \nabla_{\boldsymbol{y}} U(\boldsymbol{y}) + O\left(\delta^2 |\partial_x \nabla_{\boldsymbol{y}} u|\right)$$

and

$$\int_0^\delta \big(\boldsymbol{b_y}w\big)(x,\boldsymbol{y})\,\mathrm{d}x = \delta \boldsymbol{B}(\boldsymbol{y})W(\boldsymbol{y}) + O\big(\delta^2|\partial_x(\boldsymbol{b_y}w)|\big)$$

where

$$\boldsymbol{B}(\boldsymbol{y}) = \boldsymbol{b_y}(\delta/2,\boldsymbol{y}),$$

we obtain

$$\boldsymbol{J} = -\mathbb{A}_t\delta\nabla_{\boldsymbol{y}}U + \delta\boldsymbol{B}W + O\big(\delta^2(|\partial_x\nabla_{\boldsymbol{y}}u| + |\partial_x(\boldsymbol{b_y}w)|)\big). \tag{2.8}$$

So far, we have derived equations for the state quantities U and $\boldsymbol{J}$ on the fracture manifold γ. In order to get a well-posed problem, we have to prescribe two conditions for boundaries γ_i, $i = 1, 2$. To this end, we perform integration of the normal flux $\boldsymbol{j}_x$, given by (2.4), separately for the left and right half of the fracture. Similarly as before we use approximations

$$\int_0^{\delta/2} \boldsymbol{j}_x\,\mathrm{d}x = (\boldsymbol{j}_1 \cdot \boldsymbol{n}_1)\frac{\delta}{2} + O(\delta^2|\partial_x\boldsymbol{j}_x|)$$

and

$$\int_0^{\delta/2} \boldsymbol{b}_x w\,\mathrm{d}x = (\boldsymbol{b}_1 \cdot \boldsymbol{n}_1)\tilde{w}_1\frac{\delta}{2} + O(\delta^2|\partial_x\boldsymbol{b}_x||w| + \delta^2|\boldsymbol{b}_x||\partial_x w|)$$

and their counter parts on the interval $(\delta/2, \delta)$ to get

$$\boldsymbol{j}_1 \cdot \boldsymbol{n}_1 = -\frac{2a_n}{\delta}(U - u_1) + \boldsymbol{b}_1 \cdot \boldsymbol{n}_1\tilde{w}_1 \tag{2.9}$$

$$\boldsymbol{j}_2 \cdot \boldsymbol{n}_2 = -\frac{2a_n}{\delta}(U - u_2) + \boldsymbol{b}_2 \cdot \boldsymbol{n}_2\tilde{w}_2 \tag{2.10}$$

where $\tilde{w}_i$ can be any convex combination of w_i and W. Equations (2.9) and (2.10) have meaning of a semi-discretized flux from domains Ω_i into fracture. In order to get a stable numerical scheme, we introduce a kind of upwind already on this level using a different convex combination for each flow direction:

$$\begin{aligned}
\boldsymbol{j}_i \cdot \boldsymbol{n}_i = &-\sigma_i(U - u_i) \\
&+ \big[\boldsymbol{b}_i \cdot \boldsymbol{n}_i\big]^+\big(\xi w_i + (1-\xi)W\big) \\
&+ \big[\boldsymbol{b}_i \cdot \boldsymbol{n}_i\big]^-\big((1-\xi)w_i + \xi W\big), \qquad i = 1, 2
\end{aligned} \tag{2.11}$$

where $\sigma_i = \frac{2a_n}{\delta}$ is the transition coefficient and the parameter $\xi \in [\frac{1}{2}, 1]$ can be used to interpolate between upwind ($\xi = 1$) and central difference ($\xi = \frac{1}{2}$) scheme. Equations (2.7), (2.8), and (2.11) describe the general form of the advection-diffusion process on the fracture and its communication with the surrounding continuum which we shall later apply to individual processes.

2.3 Darcy Flow Model

We consider the simplest model for the velocity of the steady or unsteady flow in porous and fractured medium given by the Darcy flow:

$$\boldsymbol{w} = -\mathbb{K}\nabla H \quad \text{in } \Omega_d, \text{ for } d = 1, 2, 3. \tag{2.12}$$

Here and later on, we drop the dimension index d of the quantities if it can be deduced from the context. In (2.12), $\boldsymbol{w}$ [ms^{-1}] is the superficial velocity, $\mathbb{K}_d$ is the conductivity tensor, and H [m] is the piezometric head. The velocity $\boldsymbol{w}_d$ is related to the flux $\boldsymbol{q}_d$ [m^{4-d}s^{-1}] through

$$\boldsymbol{q}_d = \delta_d \boldsymbol{w}_d,$$

where δ_d [m^{3-d}] is the cross section coefficient, in particular $\delta_3 = 1$, δ_2 [m] is the thickness of a fracture, and δ_1 [m^2] is the cross-section of a channel. The flux $\boldsymbol{q}_d \cdot \boldsymbol{n}$ is the volume of the liquid (water) that passes through a unit square ($d = 3$), unit line ($d = 2$), or through a point ($d = 1$) per one second. The conductivity tensor is given by the product $\mathbb{K}_d = k_d \mathbb{A}_d$, where $k_d > 0$ [ms^{-1}] is the hydraulic conductivity and $\mathbb{A}_d$ is the 3×3 dimensionless anisotropy tensor which has to be symmetric and positive definite. The piezometric-head H_d is related to the pressure head h_d through

$$H_d = h_d + z \tag{2.13}$$

assuming that the gravity force acts in the negative direction of the z-axis. Combining these relations, we get the Darcy law in the form:

$$\boldsymbol{q} = -\delta k \mathbb{A} \nabla (h + z) \qquad \text{in } \Omega_d, \text{ for } d = 1, 2, 3. \tag{2.14}$$

Next, we employ the continuity equation for saturated porous medium and the dimensional reduction from the preceding section (with $w = u := H$, $\boldsymbol{j} := \boldsymbol{w}$, $\mathbb{A} := \mathbb{K}$ and $\boldsymbol{b} := \boldsymbol{0}$), which yields:

$$\partial_t(\delta S\, h) + \operatorname{div} \boldsymbol{q} = F + F_M \qquad \text{in } \Omega_d, \text{ for } d = 1, 2, 3, \tag{2.15}$$

where S_d [m^{-1}] is the storativity and F_d [m^{3-d}s^{-1}] is the source term. The extra source term F_M [m^{3-d}s^{-1}] due to mechanics is described in (2.52). In our setting the principal unknowns of the system (2.14, 2.15) are the pressure head h_d and the flux $\boldsymbol{q}_d$.

The storativity (or the volumetric specific storage) $S_d > 0$ can be expressed as

$$S_d = \gamma_w(\beta_r + \vartheta \beta_w), \tag{2.16}$$

where γ_w [kgm^{-2}s^{-2}] is the specific weight of water, ϑ [$-$] is the porosity, β_r is compressibility of the bulk material of the pores (rock) and β_w is compressibility of the water, both with units [kg^{-1}ms^{-2}]. For steady problems, we set $S_d = 0$ for all dimensions $d = 1, 2, 3$. The source term F_d on the right hand side of (2.15) consists of the volume density of the water source f_d[s^{-1}] and flux from the from the higher dimension. Precise form of F_d slightly differs for every dimension and will be discussed presently.

In Ω_3 we simply have $F_3 = f_3$ [s^{-1}].

2.3.1 Coupling on mixed meshes

In the set $\Omega_2 \cap \Omega_3$ the fracture is surrounded by at most one 3D surface from every side. On $\partial\Omega_3 \cap \Omega_2$ we prescribe a boundary condition of the Robin type:

$$\boldsymbol{q}_3 \cdot \boldsymbol{n}^+ = q_{32}^+ = \sigma_3(h_3^+ - h_2),$$
$$\boldsymbol{q}_3 \cdot \boldsymbol{n}^- = q_{32}^- = \sigma_3(h_3^- - h_2),$$

where $\boldsymbol{q}_3 \cdot \boldsymbol{n}^{+/-}$ [ms^{-1}] is the outflow from Ω_3, $h_3^{+/-}$ is a trace of the pressure head in Ω_3, h_2 is the pressure head in Ω_2, and σ_3 [s^{-1}] is the transition coefficient given by (see section 2.2 and [8])

$$\sigma_3 = \sigma_{32} \frac{2\mathbb{K}_2 : \boldsymbol{n}_2 \otimes \boldsymbol{n}_2}{\delta_2}.$$

Here $\boldsymbol{n}_2$ is the unit normal to the fracture (sign does not matter). On the other hand, the sum of the interchange fluxes $q_{32}^{+/-}$ forms a volume source in Ω_2. Therefore F_2 [ms^{-1}] on the right hand side of (2.15) is given by

$$F_2 = \delta_2 f_2 + (q_{32}^+ + q_{32}^-). \tag{2.17}$$

The communication between Ω_2 and Ω_1 is similar. However, in the 3D ambient space, a 1D channel can join multiple 2D fractures $1, \ldots, n$. Therefore, we have n independent outflows from Ω_2:

$$\boldsymbol{q}_2 \cdot \boldsymbol{n}^i = q_{21}^i = \sigma_2(h_2^i - h_1),$$

where σ_2 [ms^{-1}] is the transition coefficient integrated over the width of the fracture i:

$$\sigma_2 = \sigma_{21} \frac{2\delta_2^2 \mathbb{K}_1 : \boldsymbol{n}_1^i \otimes \boldsymbol{n}_1^i}{\delta_1}.$$

Here $\boldsymbol{n}_1^i$ is the unit normal to the channel that is tangential to the fracture i. Sum of the fluxes forms a part of F_1 [m^{2}s^{-1}]:

$$F_1 = \delta_1 f_1 + \sum_{i=1}^{n} q_{21}^i. \tag{2.18}$$

We remark that the direct communication between 3D and 1D (e.g. model of a well) is not supported yet. The transition coefficients σ_{32} [$-$] and σ_{21} [$-$] are independent scaling parameters which represent the ratio of the crosswind and the tangential conductivity in the fracture. For example, in the case of impermeable film on the fracture walls one may choice $\sigma_{32} < 1$.

2.3.2 Boundary conditions

In order to obtain unique solution we have to prescribe boundary conditions. Currently we consider a disjoint decomposition of the boundary

$$\partial\Omega_d = \Gamma_d^D \cup \Gamma_d^{TF} \cup \Gamma_d^{Sp} \cup \Gamma_d^{Ri}$$

where we support the following types of boundary conditions:

Dirichlet boundary condition

$$h_d = h_d^D \text{ on } \Gamma_d^D,$$

where h_d^D [m] is the boundary pressure head . Alternatively one can prescribe the boundary piezometric head H_d^D [m] related to the pressure head through (2.13).

Total flux boundary condition (combination of Neumann and Robin type)

$$-\boldsymbol{q}_d \cdot \boldsymbol{n} = \delta_d \left(q_d^N + \sigma_d^R(h_d^R - h_d) \right) \text{ on } \Gamma_d^{TF},$$

where q_d^N [ms^{-1}] is the surface density of the water inflow, h_d^R [m] is the boundary pressure head and σ_d^R [s^{-1}] is the transition coefficient. As before one can also prescribe the boundary piezo head H_d^R to specify h_d^R.

Seepage face condition is used to model a surface with possible springs:

$$h_d \leq h_d^S \quad \text{and} \quad -\boldsymbol{q}_d \cdot \boldsymbol{n} \leq \delta_d q_d^N \tag{2.19}$$

while the equality holds in at least one inequality. The switch pressure head h_d^S [m] can alternatively be given by switch piezometric head.

The first inequality in (2.19) with the default value $h_d^S = 0$ disallows non-zero water height on the surface, the later inequality with default value $q_d^N = 0$ allows only outflow from the domain (i.e. spring). In practice one may want to allow given water height h_d^S or given infiltration (e.g. precipitation-evaporation) q_d^N.

River boundary condition models free water surface with bedrock of given conductivity. We prescribe:

$$-\boldsymbol{q}_d \cdot \boldsymbol{n} = \delta_d \left(\sigma_d^R (H_d - H_d^D) + q_d^N \right), \quad \text{for } H_d \geq H_d^S, \tag{2.20}$$

$$-\boldsymbol{q}_d \cdot \boldsymbol{n} = \delta_d \left(\sigma_d^R (H_d^S - H_d^D) + q_d^N \right), \quad \text{for } H_d < H_d^S, \tag{2.21}$$

where H_d is piezometric head. The parameters of the condition are given by similar fields of other boundary conditions: the transition coefficient of the bedrock σ_d^R [s^{-1}], the piezometric head of the water surface given as boundary piezometric head H_d^D [m], the head of the bottom of the river given as the switch piezometric head H_d^S [m]. The boundary flux q_d^N is zero by default, but can be used to express approximation of the seepage face condition (see discussion below). The piezometric heads H_d^S and H_d^R may be alternatively given by pressure heads h_d^S and h_d^R, respectively.

The physical interpretation of the condition is as follows. For the water level H_d above the bottom of the river H_d^S the infiltration is given as Robin boundary condition with respect to the surface of the river H_d^D. For the water level below the bottom the infiltration is given by the water column of the river and transition coefficient of the bedrock.

The river could be used to approximate the seepage face condition in the similar way as the Robin boundary condition with large σ can approximate Dirichlet boundary condition. We rewrite the condition as follows

$$-\boldsymbol{q}_d \cdot \boldsymbol{n} = \delta_d \left(\sigma_d^R (h_d - h_d^D) + q_d^N \right), \quad \text{for } -\boldsymbol{q}_d \cdot \boldsymbol{n} \geq \delta_d \left(\sigma_d^R (h_d^S - h_d^D) + q_d^N \right), \tag{2.22}$$

$$-\boldsymbol{q}_d \cdot \boldsymbol{n} = \delta_d \left(\sigma_d^R (h_d^S - h_d^D) + q_d^N \right), \quad \text{for } h_d < h_d^S. \tag{2.23}$$

Now if we take $h_d^S = h_d^D$, we obtain

$$-\boldsymbol{q}_d \cdot \boldsymbol{n} = \delta_d \left(\sigma_d^R (h_d - h_d^S) + q_d^N \right), \quad \text{for } -\boldsymbol{q}_d \cdot \boldsymbol{n} \geq \delta_d q_d^N, \tag{2.24}$$

$$-\boldsymbol{q}_d \cdot \boldsymbol{n} = \delta_d q_d^N, \quad \text{for } h_d < h_d^S, \tag{2.25}$$

where the first equation approximates $h_d = h_d^S$ if σ_d^R is sufficiently large.

2.3.3 Steady and unsteady Darcian flow

By default, the storativity is zero which means that the flow is calculated steady. If, in addition, some input fields are time-dependent, a sequence of steady problems is

calculated for times in which the data change. When storativity is nonzero, the problem becomes unsteady and one has to specify the initial condition and the computational time interval.

2.3.4 Initial condition

For unsteady problems one has to specify an initial condition in terms of the initial pressure head h_d^0 [m] or the initial piezometric head H_d^0 [m].

2.3.5 Water balance

The equation (2.15) satisfies the volume balance of the liquid in the following form:

$$V(0) + \int_0^t s(\tau)\,d\tau + \int_0^t f(\tau)\,d\tau = V(t)$$

for any instant t in the computational time interval. Here

$$V(t) := \sum_{d=1}^{3} \int_{\Omega^d} (\delta S h)(t, \boldsymbol{x})\,d\boldsymbol{x},$$

$$s(t) := \sum_{d=1}^{3} \int_{\Omega^d} F(t, \boldsymbol{x})\,d\boldsymbol{x},$$

$$f(t) := -\sum_{d=1}^{3} \int_{\partial\Omega^d} \boldsymbol{q}(t, \boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x})\,d\boldsymbol{x}$$

is the volume [m³], the volume source [m³s⁻¹] and the volume flux [m³s⁻¹] of the liquid at time t, respectively. The volume, flux and source on every geometrical region is calculated at each output time and the values together with the control sums are written to the file water_balance.{dat|txt}. If, in addition, cumulative is set to true then the time-integrated flux and source is written. The format of balance output is described in Section 4.4.1.

2.3.6 Richards Equation

This section contains a preliminary documentation to the unsaturated water flow model. We use the Richards equation in the form:

$$\partial_t \delta\theta_t + \operatorname{div}\boldsymbol{q} = F \quad \in \Omega_d, \text{ for } d = 1, 2, 3 \tag{2.26}$$

where the total water content $\theta_t(h)$ [−] is a function of the principal unknown h and the water flux $\boldsymbol{q}$ is given by (2.14) in which the conductivity k_d is function of the pressure head h as well. Currently the total water content is given as:

$$\theta_t(h) = \theta(h) + Sh \tag{2.27}$$

where S is the storativity and $\theta(h)$ is the water content. The functions $\theta(h)$ and $k(h)$ are given by the chosen soil model. Two soil models are currently supported.

van Genuchten

Classical van Genuchten model use:

$$\theta(h) = (\theta_s - \theta_r)\theta_e + \theta_r, \quad \theta_e = (1 + (\alpha h)^n)^m$$

for the negative pressure head $h < 0$ and $\theta = \theta_s$ for $h \geq 0$.

The model parameters are: θ_s [−] the saturated water content, θ_r [−] the residual water content, α [m^{-1}] the pressure scaling parameter, n [−] the exponent parameter. The exponent m is taken as $1/n - 1$ and θ_e [−] is called the effective water content.

The conductivity function $k(h)$ is then derived from the capillary model due to Mualem with result:

$$k(h) = \theta_e^{0.5} \left[\frac{1 - F(\theta)}{1 - F(\theta_s)} \right]^2, \quad F(\theta) = \left[1 - \theta_e^{1/m} \right]^m$$

In fact we use slight modification due to Vogel and Císlerová where the saturation happens at some pressure head slightly smaller then zero. Then the water content curve is given by

$$\theta(h) = (\theta_m - \theta_r)\theta_e + \theta_r,$$

for $h < h_s$ and $\theta = \theta_s$ for $h \geq h_s$. Currently the fraction θ_m/θ_s is fixed to 0.001.

Irmay

The model used for bentonite is due to Irmay and use simple power relation for the conductivity:

$$k(h) = \theta_e^3.$$

2.3.7 Coupling of dimensions for non-conforming meshes

Version 3.0.0 introduce an experimental support for the non-conforming meshes of mixed dimension. In particular 1D-2D coupling is supported in the 2D ambient space and 2D-3D and 2D-2D coupling is supported for the 3D ambient space. Non-conforming coupling is supported only by the Darcy flow model and lower dimensional elements can not represent barriers, i.e. we consider that the pressure and the velocity fields are continuous across the lower dimensional fractures. Search for the non-conforming intersections and assembly of the associated terms in the weak formulation is turned on by the key mortar_method. One of two methods can be selected: $P0$ method is faster but can be a bit unstable for coarse meshes, $P1$ method should be more robust.

2.4 Transport of Substances

The motion of substances dissolved in water is governed by the *advection*, and the *hydrodynamic dispersion*. In Ω_d, $d \in \{1, 2, 3\}$, we consider the following system of mass balance equations[1]:

$$\partial_t(\delta\vartheta c^i) + \operatorname{div}(\boldsymbol{q}c^i) - \operatorname{div}(\vartheta\delta\mathbb{D}^i\nabla c^i) = F_S^i + F_C^c + F_R(c^1, \ldots, c^s). \tag{2.28}$$

---

[1]For $d \in \{1, 2\}$ this form can be derived as in Section 2.2 using $w := \delta\vartheta c^i$, $u := c^i$, $\mathbb{A} := \delta\vartheta\mathbb{D}^i$, $\boldsymbol{b} := \boldsymbol{v} = \frac{\boldsymbol{q}}{\vartheta\delta}$.

The principal unknown is the concentration c^i [kgm^{-3}] of a substance $i \in \{1, \ldots, s\}$, which means the weight of the substance in the unit volume of water. Other quantities are:

- The porosity ϑ [$-$], i.e. the fraction of space occupied by water and the total volume.

- The hydrodynamic dispersivity tensor $\mathbb{D}^i$ [m^{2}s^{-1}] has the form

$$\mathbb{D}^i = \tau \mathbb{D}^i_m + |\boldsymbol{v}| \left(\alpha^i_T \mathbb{I} + (\alpha^i_L - \alpha^i_T) \frac{\boldsymbol{v} \otimes \boldsymbol{v}}{|\boldsymbol{v}|^2} \right), \tag{2.29}$$

  which represents (generally anisotropic) molecular diffusion, and mechanical dispersion in longitudinal and transverse direction to the flow. Here $\mathbb{D}^i_m$ [m^{2}s^{-1}] is the 2$^{\text{nd}}$-order molecular diffusion coefficient of the i-th substance (usual magnitude in clear water is 10^{-9}), $\tau = \vartheta^{1/3}$ is the tortuosity (by [9]), α^i_L [m] and α^i_T [m] is the longitudinal dispersivity and the transverse dispersivity, respectively. The diffusion and dispersion coefficients are related to the liquid phase. Note that although we allow dispersivity to have different values for different substances, it is often assumed that they are intrinsic parameters of the porous medium. Finally, $\boldsymbol{v}$ [ms^{-1}] is the *microscopic* water velocity, also called *seepage velocity*, related to the Darcy flux $\boldsymbol{q}$ by the relation $\boldsymbol{q} = \vartheta \delta \boldsymbol{v}$. The value of $\mathbb{D}^i_m$ for specific substances can be found in literature (see e.g. [2]). For instructions on how to determine α^i_L, α^i_T we refer to [3, 4].

- F^i_S [kgm^{-d}s^{-1}] represents the density of concentration sources in the porous medium. Its form is:

$$F^i_S = \delta f^i_S + \delta(c^i_S - c^i)\sigma^i_S. \tag{2.30}$$

  Here f^i_S [kgm^{-3}s^{-1}] is the density of concentration sources, c^i_S [kgm^{-3}] is an equilibrium concentration and σ^i_S [s^{-1}] is the concentration flux. One has to pay attention when prescribing the source, namely to determine whether it is related to the *liquid* or the *porous medium*. We mention several examples:

  - extraction of solution: $f^i_S = 0$, $c^i_S = 0$, $\sigma^i_S > 0$ is the intensity of extraction, i.e. volume of liquid extracted from a unit volume of porous medium per second;

  - injection of solution: $f^i_S = 0$, c^i_S is the concentration of the substance in the injected liquid, $\sigma^i_S > 0$ is the intensity of injection (volume of liquid injected into a unit volume of porous medium per second);

  - production or degradation of substances due to bacteria present in liquid: $f^i_S = \vartheta p^i$, where p^i is the production/degradation rate in a unit volume of liquid;

  - age of liquid: if $f^i_S = \vartheta$ then c^i is the age of liquid, i.e. the time spent in the domain.

- F^c_C [kgm^{-d}s^{-1}] is the density of concentration sources due to exchange between regions with different dimensions, see (2.32) below.

- The reaction term $F_R(\ldots)$ [kgm^{-d}s^{-1}] is thoroughly described in the next section 2.5, see also paragraph "Two transport models" below.

Initial and boundary conditions. At time $t = 0$ the concentration is determined by the initial condition

$$c^i(0, \boldsymbol{x}) = c_0^i(\boldsymbol{x}).$$

The physical boundary $\partial\Omega_d$ is decomposed into the parts $\Gamma_I \cup \Gamma_D \cup \Gamma_{TF} \cup \Gamma_{DF}$, which may change during simulation time. The first part Γ_I is further divided into two segments:

$$\Gamma_I^+(t) = \{\boldsymbol{x} \in \partial\Omega_d \,|\, \boldsymbol{q}(t, \boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) < 0\},$$
$$\Gamma_I^-(t) = \{\boldsymbol{x} \in \partial\Omega_d \,|\, \boldsymbol{q}(t, \boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) \geq 0\},$$

where $\boldsymbol{n}$ stands for the unit outward normal vector to $\partial\Omega_d$. We prescribe the following boundary conditions:

- **inflow** Default transport boundary condition. On the inflow Γ_I^+ the reference concentration c_D^i [kgm^{-3}] is enforced through total flux:

$$(\boldsymbol{q}c^i - \vartheta\delta\mathbb{D}^i\nabla c^i) \cdot \boldsymbol{n} = \boldsymbol{q} \cdot \boldsymbol{n}c_D^i \text{ on } \Gamma_I^+,$$

  while on the outflow Γ_I^- we prescribe zero diffusive flux:

$$-\vartheta\delta\mathbb{D}^i\nabla c^i \cdot \boldsymbol{n} = 0 \text{ on } \Gamma_I^-.$$

- **Dirichlet** On Γ_D, the Dirichlet condition is imposed via prescribed concentration c_D^i:

$$c^i = c_D^i \text{ on } \Gamma_D.$$

- **total_flux** On Γ_{TF} we impose total mass flux condition:

$$(-\boldsymbol{q}c^i + \vartheta\delta\mathbb{D}^i\nabla c^i) \cdot \boldsymbol{n} = \delta(f_N^i + \sigma_R^i(c_D^i - c^i)),$$

  with user-defined incoming concentration flux f_N^i [kgm^{-2}s^{-1}], transition parameter σ_R^i [ms^{-1}], and reference concentration c_D^i [kgm^{-3}].

- **diffusive_flux** Finally on Γ_{DF} we prescribe diffusive mass flux (analogously to the previous case):

$$\vartheta\delta\mathbb{D}^i\nabla c^i \cdot \boldsymbol{n} = \delta(f_N^i + \sigma_R^i(c_D^i - c^i)).$$

We mention several typical uses of boundary conditions:

- natural inflow: Use Dirichlet or inflow b.c. (the later type is useful when the location of liquid inflow is not known a priori) and specify c_D^i.

- natural outflow: The substance leaves the domain only due to advection by the liquid. Use zero diffusive_flux or inflow (the latter in case that the outflow boundary is not known a priori).

- boundary with known mass flux: Use total_flux and f_N^i.

- impermeable boundary: Use zero total_flux.

- partially permeable boundary: When the exterior of the domain represents a reservoir with known concentration and the Darcy flux is reasonably small, the mass exchange is proportional to the concentration difference inside and outside the domain. Use diffusive_flux, c_D^i and σ_R^i.

Communication between dimensions.   Transport of substances is considered also on interfaces of physical domains with adjacent dimensions (i.e. 3D-2D and 2D-1D, but not 3D-1D). Denoting c_{d+1}, c_d the concentration of a given substance in Ω_{d+1} and Ω_d, respectively, the communication on the interface between Ω_{d+1} and Ω_d is described by the quantity

$$q_{d+1,d}^c = \sigma_{d+1,d}^c \frac{\delta_{d+1}^2}{\delta_d} 2\vartheta_d \mathbb{D}_d : \boldsymbol{n} \otimes \boldsymbol{n}(c_{d+1} - c_d) + q_{d+1,d}^l \begin{cases} c_{d+1} & \text{if } q_{d+1,d}^l \geq 0, \\ c_d & \text{if } q_{d+1,d}^l < 0, \end{cases} \quad (2.31)$$

where

- $q_{d+1,d}^c$ $[\text{kgm}^{-d}\text{s}^{-1}]$ is the density of concentration flux from Ω_{d+1} to Ω_d,

- $\sigma_{d+1,d}^c$ $[-]$ is a transition parameter. Its value determines the mass exchange between dimensions whenever the concentrations differ. In general, it is recommended to leave the default value $\sigma^c = 1$ or to set $\sigma^c = 0$ (when exchange is due to water flux only).

- $q_{d+1,d}^l$ $[\text{m}^{3-d}\text{s}^{-1}]$ is the water flux from Ω_{d+1} to Ω_d, i.e. $q_{d+1,d}^l = \boldsymbol{q}_{d+1} \cdot \boldsymbol{n}_{d+1}$.

The communication between dimensions is incorporated as the total flux boundary condition for the problem on Ω_{d+1}:

$$- \vartheta\delta\mathbb{D}\nabla c \cdot \boldsymbol{n} + q^l c = q^c \quad (2.32)$$

and a source term in Ω_d:

$$F_{C3}^c = 0, \quad F_{C2}^c = q_{32}^c, \quad F_{C1}^c = q_{21}^c. \quad (2.33)$$

Transport models.   Within the above presented model, Flow123d presents two possible approaches to solute transport.

- For modeling pure advection ($\mathbb{D} = 0$) one can choose `Solute_Advection_FV` method, which represents an explicit in time finite volume solver. Only the inflow/outflow boundary condition is available and the source term has the form

$$F_S^i = \delta f_S^i + \delta(c_S^i - c^i)^+ \sigma_S^i.$$

  The solution process for one time step is faster, but the maximal time step is restricted. The resulting concentration is piecewise constant on mesh elements. This solver supports reaction term (involving simple chemical reactions, dual porosity and sorptions).

- The full model including dispersion is solved by `Solute_AdvectionDiffusion_DG`, an implicit in time discontinuous Galerkin solver. It has no restriction of the computational time step and the space approximation is piecewise polynomial, currently up to order 3. Reaction term is implemented only for the case of linear sorption, i.e.

$$F_R^i = -\partial_t \left((1-\vartheta)\delta\varrho_s c_s^i \right), \quad c_s^i = k_l^i c,$$

  where c_s^i $[-]$ is the relative concentration of sorbed substance, k_l^i $[\text{kg}^{-1}\text{m}^3]$ is the sorption coefficient, ϱ_s and ϱ_l $[\text{kgm}^{-3}]$ is the density of the solid (rock) and of the liquid (solvent), respectively. The initial concentration in solid is assumed to be in equilibrium with the concentration in liquid.

- Finally, both previous methods can be coupled with the reaction term (see 2.5) using the `Coupling_OperatorSplitting` model. The operator splitting is essentially an explicit method requiring the time step set in its `time` key should be reasonable, however no automatic restriction on the time step is forced. However this time step makes an upper bound for the time step of the underlaying transport equation in use.

Mass balance. The advection-dispersion equation satisfies the balance of mass in the following form:

$$m^i(0) + \int_0^t s^i(\tau)\, d\tau + \int_0^t f^i(\tau)\, d\tau = m^i(t)$$

for any instant t in the computational time interval and any substance i. Here

$$m^i(t) := \sum_{d=1}^{3} \int_{\Omega^d} (\delta\vartheta c^i)(t, \boldsymbol{x})\, d\boldsymbol{x},$$

$$s^i(t) := \sum_{d=1}^{3} \int_{\Omega^d} F_S^i(t, \boldsymbol{x})\, d\boldsymbol{x},$$

$$f^i(t) := \sum_{d=1}^{3} \int_{\partial\Omega^d} \left(-\boldsymbol{q} c^i + \vartheta\delta\mathbb{D}^i \nabla c^i \right)(t, \boldsymbol{x}) \cdot \boldsymbol{n}\, d\boldsymbol{x}$$

is the mass [kg], the volume source [kgs^{-1}] and the mass flux [kgs^{-1}] of i-th substance at time t, respectively. The mass, flux and source on every geometrical region is calculated at each output time and the values are written to the file `mass_balance.{dat|txt}`. If, in addition, cumulative is set to true then the time-integrated flux and source is written. In that case the cumulative source contains also contribution due to reactions. The format of balance output is described in Section 4.4.1.

2.5 Reaction Term in Transport

The `TransportOperatorSplitting` method supports the reaction term $F_R(c^1, \ldots, c^s)$ on the right hand side of the equation (2.28). It can represent several models of chemical or physical nature. Figure 2.2 shows all possible reactional models that we support in combination with the transport process. The Operator Splitting method enables us to deal with the convection part and reaction term independently. Transport of the substances is computed independently using either are convected quantities do not influence each other in the convectional process and are balanced over the elements. On the other hand the reaction term relates the convected quantities and can be computed separately on each element.

We move now to the description of the reaction models which can be seen again in Figure 2.2. The convected quantity is considered to be the concentration of substances. Up to now we can have *dual porosity*, *sorption* (these two are more of a physical nature) and (chemical) *reaction* models in the reaction term.

The *reaction* model acts only on the specified substances and computes exchange of concentration among them. It does not have its own output because it only changes the concentration of substances in the specified zone where the reaction takes place.
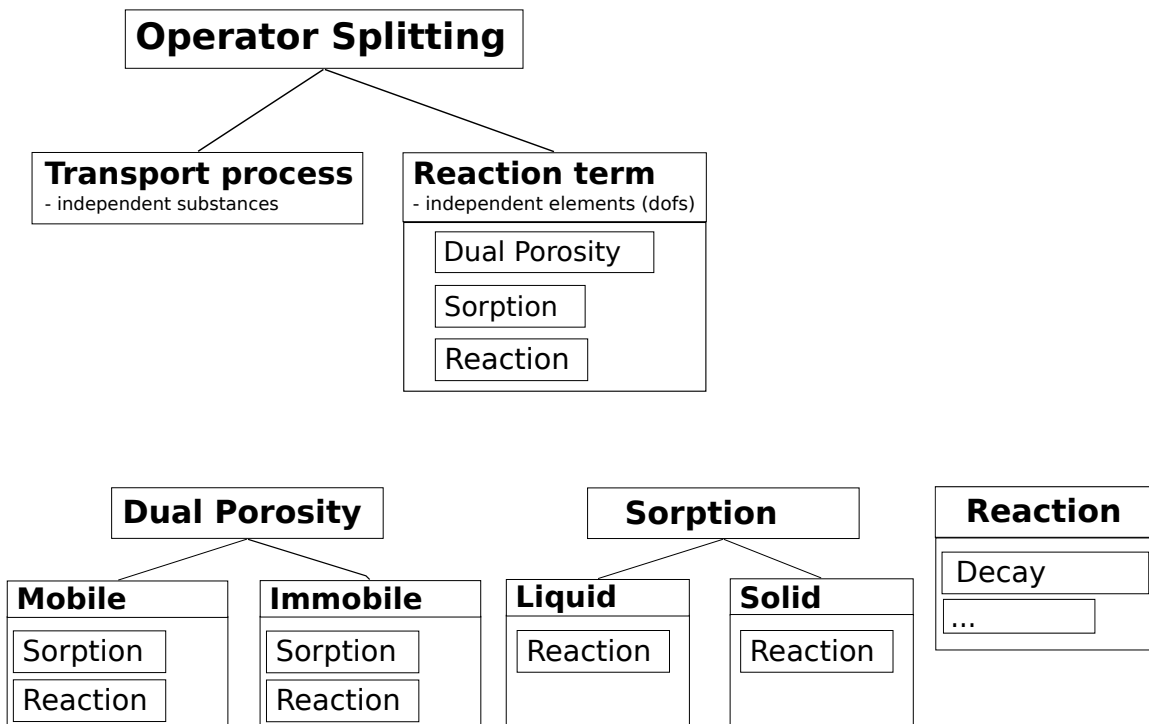
Figure 2.2: The scheme of the reaction term objects. The lines represents connections between different models. The tables under model name include the possible models which can be connected to the model above.

The *sorption* model describes the exchange of concentration of the substances between liquid and solid. It can be followed by another *reaction* that can run in both phases. The concentration in solid is an additional output of this model. See Subsection 2.5.2.

The *dual porosity* model, described in Subsection 2.5.1, introduces the so called immobile (or dead-end) pores in the matrix. The convection process operates only on the concentration of the substances in the mobile zone (open pores) and the exchange of concentrations from/to immobile zone is governed by molecular diffusion. This process can be followed by *sorption* model and/or chemical *reaction*, both in mobile and immobile zone. The immobile concentration is an additional output.

2.5.1 Dual Porosity

Up to now, we have described the transport equation for the single porosity model. The dual porosity model splits the mass into two zones – the mobile zone and the immobile zone. Both occupy the same macroscopic volume, however on the microscopic scale, the immobile zone is formed by the dead-end pores, where the liquid is trapped and cannot pass through. The rest of the pore volume is occupied by the mobile zone. Since the liquid in the immobile pores is immobile, the exchange of the substance is only due to molecular diffusion. We consider simple non-equilibrium linear model:

$$\vartheta_m \partial_t c_m = D_{dp}(c_i - c_m), \tag{2.34a}$$
$$\vartheta_i \partial_t c_i = D_{dp}(c_m - c_i), \tag{2.34b}$$

where c_m is the concentration in the mobile zone, c_i is the concentration in the immobile zone and D_{dp} is a diffusion rate between the zones. ϑ_i denotes porosity of the immobile zone and $\vartheta_m = \vartheta$ the mobile porosity from transport equation (2.28). One can also set non-zero initial concentration in the immobile zone $c_i(0)$.

To solve the system of first order differential equation, we use analytic solution or Euler's method, which are switched according to a given error tolerance. See subsection 3.5.1 in numerical methods.

2.5.2 Equilibrial Sorption

The simulation of an equilibrium sorption is based on the solution of two algebraic equations, namely the mass balance (in unit volume)

$$\vartheta \varrho_l c_l + (1 - \vartheta)\varrho_s c_s = c_T = const. \tag{2.35}$$

and an empirical sorption law

$$c_s = f(c_l), \tag{2.36}$$

given in terms of the so-called isotherm function f. In these equations we use following notation. The concentration in the solid phase, $c_s = \frac{m_{sorbed}}{m_s}$ $[-]$ is the adsorbed mass of the substance per the unit mass of the solid adsorbent in a reference volume. The concentration in the solid can be selected for output. The concentration in the liquid phase, $c_l = \frac{m}{m_l}$ $[-]$ is the mass of dissolved substance per the unit mass of the liquid. The relation between c_l and the concentration c from the transport equation (2.28) is

$c = c_l \varrho_l$. Finally, θ is the porosity, ϱ_s is the solid density i.e. density of a compact rock with zero porosity, and ϱ_l is the liquid density, i.e. density of the solvent.

The form of the isotherm f is determined by the parameter sorption type:

sorption_type	$f(c_l)$	
"none"	0	The sorption model returns zero concentration in solid.
"linear"	$k_l \varrho_l c_l$	
"freundlich"	$k_F \varrho_l c_l^\alpha$	
"langmuir"	$k_L \varrho_l \frac{\alpha c_l}{1+\alpha c_l}$	Langmuir isotherm has been derived from thermodynamic laws. The number $k_L \varrho_l$ denotes the maximal amount of sorbing specie which can be kept in an unit volume of a bulk matrix. Coefficient α is a fraction of sorption and desorption rates.

Main parameter of these isotherms is the distribution coefficient $k_i, i \in \{l, F, L\}$ $[\mathrm{kg}^{-1}\mathrm{m}^3]$. Nonlinear isotherms have an additional parameter α $[-]$. Note that older versions of Flow123d prior to 2.0.0 used a different coefficient k_i denoted `isotherm_mult` with the unit $[\mathrm{mol\,kg}^{-1}]$. The conversion rule between the old and new distribution coefficient is

$$k_i^{new} = \frac{M_s}{\varrho_l} k_i^{old},$$

where M_s $[\mathrm{kg\,mol}^{-1}]$ stands for the molar mass of a substance.

Concept of the general distribution coefficient is thoroughly discussed e.g. in [10]. Key assumptions about k_i are:

- Density ρ_l in isotherm expressions is technically the density of the solvent used during measurement of k_i, which could be different then the density of the solvent used in calculation. E.g. slight changes in the density of water according to variations in chemical composition and isotopes. But usually the difference is negligible.

- Concentrations in both liquid and solid phase are very small. In particular the number of unoccupied adsorption sites dominates the number of occupied sites.

- All adsorption sites are equivalent.

- Sorption is understood in general manner including all linear processes that are able to store the substance.

- System is considered in thermodynamic equilibrium.

- Single distribution coefficient k_i is specific for combination adsorbent, solvent, substance.

Non-zero initial concentration in the solid phase $c_s(0)$ can be set in the input record. Now, further denoting

$$\mu_l = \varrho_l \vartheta, \quad \mu_s = \varrho_s \cdot (1 - \vartheta),$$

and using (2.36), the mass balance (2.35) reduces to the equation

$$c_T = \mu_l c_l + \mu_s f(c_l), \tag{2.37}$$

which can be either solved iteratively or using interpolation. See subsection 3.5.2 in numerical methods for details.

The units of c_l, c_s and k_i can vary in literature. For an example of conversion rules in the case of Freundlich isotherm we refer to Bowman [1].

2.5.3   Sorption in Dual Porosity Model

There are two parameters μ_l and μ_s, scale of aqueous concentration and scale of sorbed concentration, respectively. There is a difference in computation of these in the dual porosity model because both work on different concentrations and different zones.

Let c_{ml} and c_{ms} be concentration in liquid and in solid in the mobile zone, c_{il} and c_{is} be concentration in liquid and in solid in the immobile zone, ϑ_m and ϑ_i be the mobile and the immobile porosity, and φ be the sorbing surface.

The sorbing surface in the mobile zone is given by

$$\varphi = \frac{\vartheta_m}{\vartheta_m + \vartheta_i}, \tag{2.38}$$

while in the immobile zone it becomes

$$1 - \varphi = 1 - \frac{\vartheta_m}{\vartheta_m + \vartheta_i} = \frac{\vartheta_i}{\vartheta_m + \vartheta_i}.$$

Remind the mass balance equation (2.37). In the dual porosity model, the scaling parameters μ_l, μ_s are slightly different. In particular, the mass balance in the mobile zone reads:

$$\begin{aligned} c_T &= \mu_l \cdot c_{ml} + \mu_s \cdot c_{ms}, \\ \mu_l &= \varrho_l \cdot \vartheta_m, \\ \mu_s &= \varrho_s \cdot (1 - \vartheta_m - \vartheta_i)\varphi, \end{aligned} \tag{2.39}$$

while in the immobile zone it has the form:

$$\begin{aligned} c_T &= \mu_l \cdot c_{il} + \mu_s \cdot c_{is}, \\ \mu_l &= \varrho_l \cdot \vartheta_i, \\ \mu_s &= \varrho_s \cdot (1 - \vartheta_m - \vartheta_i)(1 - \varphi). \end{aligned} \tag{2.40}$$

2.5.4   Radioactive Decay

The radioactive decay is one of the processes that can be modeled in the reaction term of the transport model. This process is coupled with the transport using the operator

splitting method. It can run throughout all the phases, including the mobile and immobile phase of the liquid and also the sorbed solid phase, as it can be seen in figure 2.2.

The radioactive decay of a parent radionuclide A to a nuclide B

$$A \xrightarrow{k} B, \qquad A \xrightarrow{t_{1/2}} B$$
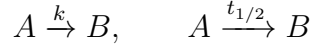
is mathematically formulated as a system of first order differential equations

$$\frac{\mathrm{d}c_A}{\mathrm{d}\tau} = -kc_A, \tag{2.41}$$

$$\frac{\mathrm{d}c_B}{\mathrm{d}\tau} = kc_A, \tag{2.42}$$

where k is the radioactive decay rate. Usually, the half life of the parent radionuclide $t_{1/2}$ is known rather than the rate. Relation of these can be derived:

$$\frac{\mathrm{d}c_A}{\mathrm{d}\tau} = -kc_A$$

$$\frac{\mathrm{d}c_A}{c_A} = -k\,\mathrm{d}\tau$$

$$\int_{c_A^0}^{c_A^0/2} \frac{\mathrm{d}c_A}{c_A} = -k \int_0^{t_{1/2}} 1\,\mathrm{d}\tau$$

$$\left[\ln c_A\right]_{c_A^0}^{c_A^0/2} = -\left[k\tau\right]_0^{t_{1/2}}$$

$$k = \frac{\ln 2}{t_{1/2}}.$$

Let us now suppose a more complex situation. Consider substances (radionuclides) $A_1, \ldots, A_s$ which take part in a complex radioactive chain, including branches, e.g.

$$A_1 \xrightarrow{k_1} A_2 \xrightarrow{k_2} A_3 \qquad \xrightarrow{k_{34}} \qquad A_4 \xrightarrow{k_4} A_8$$
$$A_3 \xrightarrow{k_{35}} A_5 \xrightarrow{k_5} A_4$$
$$A_3 \xrightarrow{k_{36}} A_6 \xrightarrow{k_6} A_7 \xrightarrow{k_7} A_8$$
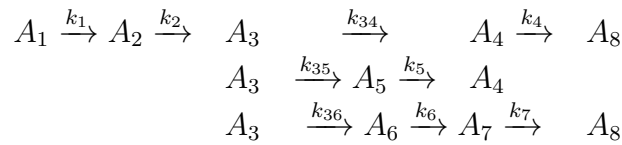
Now the problem turned into a system of differential equations $\partial_t \boldsymbol{c} = \mathbf{D}\boldsymbol{c}$ with the following matrix, generally full and nonsymmetric:

$$\mathbf{D} = \begin{pmatrix} M_1 & & & \\ & M_2 & & \\ & & \ddots & \\ & & & M_s \end{pmatrix} \begin{pmatrix} -k_1 & k_{21} & \cdots & k_{s1} \\ k_{12} & -k_2 & \cdots & k_{s2} \\ \vdots & \vdots & \ddots & \vdots \\ k_{1s} & k_{2s} & \cdots & -k_s \end{pmatrix} \begin{pmatrix} \frac{1}{M_1} & & & \\ & \frac{1}{M_2} & & \\ & & \ddots & \\ & & & \frac{1}{M_s} \end{pmatrix},$$

where M_i is molar mass. We can then write

$$d_{ij} = \begin{cases} k_{ji}\frac{M_i}{M_j}, & i \neq j, \\ -k_{ij}, & i = j. \end{cases} \tag{2.43}$$

38

We denote the rate constant of the i-th radionuclide

$$k_i = \sum_{j=1}^{s} k_{ij} = \sum_{j=1}^{s} b_{ij} k_i$$

which is equal to a sum of partial rate constants k_{ij}. Branching ratio $b_{ij} \in (0,1)$ determines the distribution into different branches of the decay chain, holding $\sum_{j=1}^{s} b_{ij} = 1$.

Notice, that physically it is not possible to create a chain loop, so in fact one can permutate the vector of concentrations and rearrange the matrix D into a lower triangle matrix

$$\mathbf{D} = \begin{pmatrix} d_{11} & & & \\ d_{21} & d_{22} & & \\ \vdots & \vdots & \ddots & \\ d_{s1} & d_{s2} & \cdots & d_{ss} \end{pmatrix}.$$

However, we do not do this and we do not search the reactions for chain loops.

The system of first order differential equations with constant coefficients is solved using one of the implemented linear ODE solvers, described in section 3.5.3.

2.5.5 First Order Reaction

First order kinetic reaction is another process that can take part in the reaction term. Similarly to the radioactive decay, it is connected to transport by operator splitting method and can run in all the possible phases, see figure 2.2.

Currently, reactions with single reactant and multiple products (decays) are available in the software. The mathematical description is the same as for the radioactive decay, it only uses kinetic reaction rate coefficient k in the input instead of half life.

2.6 Heat Transfer

Under the assumption of thermal equilibrium between the solid and liquid phase, the energy balance equation has the form[2]

$$\partial_t \left(\delta \tilde{s} T \right) + \mathrm{div}(\varrho_l c_l T \boldsymbol{q}) - \mathrm{div}(\delta \Lambda \nabla T) = F^T + F_C^T.$$

The principal unknown is the temperature T [K]. Other quantities are:

- ϱ_l, ϱ_s [kgm^{-3}] is the density of the fluid and solid phase, respectively.

- c_l, c_s [Jkg^{-1}K^{-1}] is the heat capacity of the fluid and solid phase, respectively.

- $\tilde{s}$ [Jm^{-3}K^{-1}] is the volumetric heat capacity of the porous medium defined as

$$\tilde{s} = \vartheta \varrho_l c_l + (1 - \vartheta) \varrho_s c_s.$$

---

[2]For lower dimensions this form can be derived as in Section 2.2 using $w := \delta \tilde{s} T$, $u := T$, $\mathbb{A} := \delta \lambda \mathbb{I}$, $\boldsymbol{b} := \frac{\varrho_l c_l}{\tilde{s}} \boldsymbol{w}$.

- Λ [Wm^{-1}K^{-1}] is the thermal dispersion tensor:

$$\Lambda = \Lambda^{cond} + \Lambda^{disp}$$

$$\Lambda^{cond} = \left(\vartheta \lambda_l^{cond} + (1-\vartheta)\lambda_s^{cond}\right)\mathbb{I},$$

$$\Lambda^{disp} = \vartheta \varrho_l c_l |\boldsymbol{v}|\left(\alpha_T \mathbb{I} + (\alpha_L - \alpha_T)\frac{\boldsymbol{v} \otimes \boldsymbol{v}}{|\boldsymbol{v}|^2}\right),$$

where λ_l^{cond}, λ_s^{cond} [Wm^{-1}K^{-1}] is the thermal conductivity of the fluid and solid phase, respectively, and α_L, α_T [m] is the longitudinal and transverse dispersivity in the fluid.

- F^T [Jm^{-d}s^{-1}] represents the thermal source:

$$F^T = \delta \vartheta F_l^T + \delta(1-\vartheta)F_s^T,$$

$$F_l^T = f_l^T + \varrho_l c_l \sigma_l^T (T - T_l),$$
$$F_s^T = f_s^T + \varrho_s c_s \sigma_s^T (T - T_s),$$

where f_l^T, f_s^T [Wm^{-3}] is the density of thermal sources in fluid and solid, respectively, T_l, T_s [K] is a reference temperature and σ_l^T, σ_s^T [s^{-1}] is the heat exchange rate.

Initial and boundary conditions. At time $t = 0$ the temperature is determined by the initial condition T_0 [K]:
$$T(0, \boldsymbol{x}) = T_0(\boldsymbol{x}).$$

Given the decomposition of $\partial\Omega_d$ into $\Gamma_I \cup \Gamma_D \cup \Gamma_{TF} \cup \Gamma_{DF}$ (see also Section 2.4), we prescribe the following boundary conditions:

- **inflow** Default boundary condition. On the inflow Γ_I^+ the reference temperature T_D [K] is enforced through total flux:

$$(\varrho_l c_l T \boldsymbol{q} - \delta \Lambda \nabla T) \cdot \boldsymbol{n} = \varrho_l c_l T_D \boldsymbol{q} \cdot \boldsymbol{n},$$

  while on the outflow Γ_I^- we prescribe zero diffusive flux:

$$-\delta \Lambda \nabla T \cdot \boldsymbol{n} = 0.$$

- **Dirichlet** On Γ_D, the Dirichlet condition is imposed via prescribed temperature T_D:

$$T = T_D \text{ on } \Gamma_I^+ \cup \Gamma_D.$$

- **total_flux** On Γ_{TF} we impose total energy flux condition:

$$(-\varrho_l c_l T \boldsymbol{q} + \delta \Lambda \nabla T) \cdot \boldsymbol{n} = \delta(f_N^T + \sigma_R^T (T_D - T)).$$

  with user-defined incoming energy flux f_N^T [Jm^{-2}s^{-1}], transition parameter σ_R^T [Jm^{-2}s^{-1}K^{-1}] and reference temperature T_D.

- **diffusive_flux** Finally on Γ_{DF} we prescribe diffusive energy flux (similarly as above):
$$\delta\Lambda\nabla T \cdot \boldsymbol{n} = \delta(f_N^T + \sigma_R^T(T_D - T)).$$

We mention several typical uses of boundary conditions:

- natural inflow: Use Dirichlet or inflow b.c. (the later type is useful when the location of liquid inflow is not known a priori) and specify T_D.

- natural outflow: The energy in the domain decreases only due to advection. Use zero diffusive_flux or inflow (the latter in case that the outflow boundary is not known a priori).

- boundary with known energy flux: Use total_flux and f_N^T.

- thermally insulated boundary: Use zero total_flux.

- partially permeable boundary: The energy transfer is proportional to the temperature difference inside and outside the domain. Use diffusive_flux, T_D and σ_R^T.

Communication between dimensions. Denoting T_{d+1}, T_d the temperature in Ω_{d+1} and Ω_d, respectively, the communication on the interface between Ω_{d+1} and Ω_d is described by the quantity

$$q_{d+1,d}^T = \sigma_{d+1,d}^T \frac{\delta_{d+1}^2}{\delta_d} 2\Lambda_d : \boldsymbol{n} \otimes \boldsymbol{n}(T_{d+1} - T_d) + \varrho_l c_l q_{d+1,d}^l \begin{cases} T_{d+1} & \text{if } q_{d+1,d}^l \geq 0, \\ T_d & \text{if } q_{d+1,d}^l < 0, \end{cases} \quad (2.44)$$

where

- $q_{d+1,d}^T$ [Wm^{-2}] is the density of heat flux from Ω_{d+1} to Ω_d,

- $\sigma_{d+1,d}^T$ [$-$] is a transition parameter. Its value determines the exchange of energy between dimensions due to temperature difference. In general, it is recommended to leave the default value $\sigma^T = 1$ or to set $\sigma^T = 0$ (when exchange is due to water flux only).

- $q_{d+1,d}^l = \boldsymbol{q}_{d+1} \cdot \boldsymbol{n}$ is the water flux from Ω_{d+1} to Ω_d.

The communication between dimensions is incorporated as the total flux boundary condition for the problem on Ω_{d+1}:

$$(\varrho_l c_l T\boldsymbol{q} - \delta\Lambda\nabla T) \cdot \boldsymbol{n} = q^T \tag{2.45}$$

and a source term in Ω_d:

$$F_{C3}^T = 0, \quad F_{C2}^T = q_{32}^T, \quad F_{C1}^T = q_{21}^T. \tag{2.46}$$

Energy balance. The heat equation satisfies the balance of energy in the following form:

$$e(0) + \int_0^t s(\tau)\,d\tau + \int_0^t f(\tau)\,d\tau = e(t)$$

for any instant t in the computational time interval. Here

$$e(t) := \sum_{d=1}^{3} \int_{\Omega^d} (\delta\tilde{s}T)(t,\boldsymbol{x})\,d\boldsymbol{x},$$

$$s(t) := \sum_{d=1}^{3} \int_{\Omega^d} F_S^T(t,\boldsymbol{x})\,d\boldsymbol{x},$$

$$f(t) := \sum_{d=1}^{3} \int_{\partial\Omega^d} \left(-\varrho_l c_l T \boldsymbol{q} + \delta\Lambda\nabla T\right)(t,\boldsymbol{x}) \cdot \boldsymbol{n}\,d\boldsymbol{x}$$

is the energy [J], the volume source [Js^{-1}] and the energy flux [Js^{-1}] at time t, respectively. The energy, flux and source on every geometrical region is calculated at each output time step and the values together with the control sums are written to the file `energy_balance.{dat|txt}`. If, in addition, cumulative is set to true then the time-integrated flux and source is written. The format of balance output is described in Section 4.4.1.

2.7 Mechanics

Deformation of the porous media is modelled by the stationary linear elasticity equation:

$$-\operatorname{div}(\delta\sigma(\boldsymbol{u})) = \delta\boldsymbol{f} + \boldsymbol{f}_C + \boldsymbol{f}_H. \tag{2.47}$$

Here $\boldsymbol{u}$ [m] is the displacement vector field with 3 components, the stress tensor is given by the Hooke law

$$\sigma(\boldsymbol{u}) = \mathbb{C}\tilde{\boldsymbol{\varepsilon}}(\boldsymbol{u}) = 2\mu\tilde{\boldsymbol{\varepsilon}}(\boldsymbol{u}) + \lambda(\mathbb{I} : \tilde{\boldsymbol{\varepsilon}}(\boldsymbol{u}))\mathbb{I}, \tag{2.48}$$

and the Lamé parameters are determined in terms of the Young modulus E [Pa] and Poisson's ratio ν [$-$]:

$$\mu = \frac{E}{2(1+\nu)}, \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}. \tag{2.49}$$

The strain tensor in Ω_d is defined as follows:

$$\tilde{\boldsymbol{\varepsilon}}(\boldsymbol{u}) = \frac{1}{2}(\nabla\boldsymbol{u}_d + \nabla\boldsymbol{u}_d^\top) + \begin{cases} 0 & \text{if } d = 3, \\ \frac{1}{\delta}\sum_{i=1}^{n}\boldsymbol{u}_{d+1}^i \otimes_s \boldsymbol{n}_{d+1}^i & \text{else.} \end{cases} \tag{2.50}$$

Here $\boldsymbol{a} \otimes_s \boldsymbol{b} := \frac{1}{2}(\boldsymbol{a} \otimes \boldsymbol{b} + \boldsymbol{b} \otimes \boldsymbol{a})$.

The symbol $\boldsymbol{f}$ stands for the body load [Nm^{-3}].

Hydromechanical coupling. The mechanics equation (2.47) is coupled to flow by the term

$$\boldsymbol{f}_H = -\nabla(\delta\alpha p), \quad p = \varrho_l g h, \tag{2.51}$$

where p [Pa] is the pressure, α [$-$] is the Biot coefficient, ϱ_l [kgm^{-3}] is the fluid density and g [ms^{-2}] is the gravitational acceleration. Conversely, the deformation affects the flow via the additional term

$$F_M = -\partial_t(\delta\alpha\widetilde{\mathrm{div}}\boldsymbol{u}) \tag{2.52}$$

on the right hand side of (2.15). The expression $\widetilde{\mathrm{div}}\boldsymbol{u}$ is defined as follows:

$$\widetilde{\mathrm{div}}\boldsymbol{u}_d = \mathrm{div}\,\boldsymbol{u}_d + \begin{cases} 0 & \text{if } d = 3, \\ \frac{\delta_{d+1}}{\delta_d}\sum_{i=1}^n \boldsymbol{u}_{d+1}^i \cdot \boldsymbol{n}_{d+1}^i & \text{else.} \end{cases} \tag{2.53}$$

The numerical solution of coupled hydro-mechanical problems is solved by an iterative splitting, where, in order to achieve convergence, the flow equation is modified as follows:

$$\partial_t(\delta(S + S_{extra})h) + \mathrm{div}\,\boldsymbol{q} = F + F_M + \partial_t(\delta S_{extra}h_{old}). \tag{2.54}$$

Here h_{old} is the previous value of piezometric head in the iteration process and S_{extra} is an extra storativity coefficient whose value affects the rate of convergence. It can be manually tuned using the iteration parameter.

Boundary conditions. Given the decomposition $\partial\Omega_d = \Gamma_D \cup \Gamma_{DN} \cup \Gamma_T \cup \Gamma_S$, we impose the following boundary conditions:

- **displacement** condition prescribes

$$\boldsymbol{u} = \boldsymbol{u}_D \text{ on } \Gamma_D \tag{2.55}$$

  via given displacement $\boldsymbol{u}_D$ [m].

- **displacement_n**: Displacement is prescribed only in the normal component, in tangent directions(s) zero traction is assumed:

$$\left.\begin{array}{r} \boldsymbol{u} \cdot \boldsymbol{n} = \boldsymbol{u}_D \cdot \boldsymbol{n} \\ (\sigma(\boldsymbol{u})\boldsymbol{n})_\tau = \boldsymbol{0} \end{array}\right\} \text{ on } \Gamma_{DN}. \tag{2.56}$$

  Here $\boldsymbol{a}_\tau := \boldsymbol{a} - (\boldsymbol{a} \cdot \boldsymbol{n})\boldsymbol{n}$ is the projection of a vector $\boldsymbol{a}$ to the tangent plane of the boundary.

- **traction** condition (default) is imposed via given traction $\boldsymbol{t}_N$ [Pa]:

$$\sigma(\boldsymbol{u})\boldsymbol{n} = \boldsymbol{t}_N \text{ on } \Gamma_T. \tag{2.57}$$

- **stress** condition is the same type as **traction**, but instead of traction the user supplies the full stress tensor σ_N [Pa]:

$$\sigma(\boldsymbol{u})\boldsymbol{n} = \sigma_N\boldsymbol{n} \text{ on } \Gamma_S. \tag{2.58}$$

Communication between dimensions. The mechanical interaction on the interface between Ω_{d+1} and Ω_d is realized via the traction condition on the boundary of Ω_{d+1}:

$$\delta_{d+1}(\sigma(\boldsymbol{u}_{d+1}^i) - \alpha_{d+1}p_{d+1}\mathbb{I})\boldsymbol{n}_{d+1}^i = \boldsymbol{t}_{d+1,d}^i, \tag{2.59}$$

where

$$\boldsymbol{t}_{d+1,d}^i = \sigma_{d+1,d}^U \delta_{d+1} \left(\frac{2\delta_{d+1}}{\delta_d} \mathbb{C}_d \left((\boldsymbol{u}_{d+1}^i - \boldsymbol{u}_d) \otimes \boldsymbol{n}_{d+1}^i \right) - \alpha_d p_d \mathbb{I} \right) \boldsymbol{n}_{d+1}^i \tag{2.60}$$

and $\sigma_{d+1,d}^U$ $[-]$ is the transition coefficient. The force term in Ω_d due to the interaction with Ω_{d+1} is

$$\boldsymbol{f}_{Cd} = \begin{cases} \boldsymbol{0} & \text{if } d = 3, \\ \sum_{i=1}^n \boldsymbol{t}_{d+1,d}^i & \text{else.} \end{cases} \tag{2.61}$$

Output fields. The mechanics equation provides several fields for output:

- displacement: The computed displacement vector field $\boldsymbol{u}$;

- stress: The mechanical stress tensor $\sigma(\boldsymbol{u})$;

- mean_stress: The mean of the principal stresses, i.e. $\sigma_m := \frac{1}{3}\mathbb{I} : \sigma(\boldsymbol{u})$;

- von_mises_stress: $\sigma_{VM} := \sqrt{\frac{3}{2}\sigma_d : \sigma_d}$, where $\sigma_d := \sigma(\boldsymbol{u}) - \sigma_m\mathbb{I}$ is the deviatoric stress. Equivalently, von Mises stress can be expressed as

$$\sigma_{VM} = \sqrt{\frac{1}{2}\left((\sigma_1 - \sigma_2)^2 + (\sigma_1 - \sigma_3)^2 + (\sigma_2 - \sigma_3)^2\right)},$$

  where $\sigma_{1,2,3}$ are the principal stresses, i.e. eigenvalues of $\sigma(\boldsymbol{u})$;

- cross_section_updated: This field expresses the cross-section of fractures after deformation and is defined as $\delta_d - \sum_{i=1}^n \boldsymbol{u}_{d+1}^i \cdot \boldsymbol{n}_{d+1}^i$;

- displacement_divergence: $\widetilde{\text{div}}\boldsymbol{u}_d$.

Chapter 3

Numerical Methods

3.1 Diagonalized Mixed-Hybrid Method

Model of flow described in section 2.3 is solved by the mixed-hybrid formulation (MH) of the finite element method. As in the previous chapter, let τ be the time step and $\mathcal{T}_d$ a regular simplicial partition of Ω_d, $d = 1, 2, 3$. Denote by $\boldsymbol{W}_d(T_d) \subset \boldsymbol{H}(div, T_d)$ the space of Raviart-Thomas functions of order zero (RT_0) on an element $T_d \in \mathcal{T}_d$. We introduce the following spaces:

$$\boldsymbol{W} = \boldsymbol{W}_1 \times \boldsymbol{W}_2 \times \boldsymbol{W}_3, \quad \boldsymbol{W}_d = \prod_{T_d \in \mathcal{T}_d} \boldsymbol{W}_d(T_d),$$

$$Q = Q_1 \times Q_2 \times Q_3, \quad Q_d = L^2(\Omega_d). \tag{3.1}$$

For every $T_d \in \mathcal{T}_d$ we define the auxiliary space of values on interior sides of T_d:

$$\mathring{Q}(T_d) = \left\{ \mathring{q} \in L^2(\partial T_d \setminus \partial \Omega_d^D) : \mathring{q} = \boldsymbol{w} \cdot \boldsymbol{n}|_{\partial T_d}, \boldsymbol{w} \in \boldsymbol{W}_d \right\}. \tag{3.2}$$

Further we introduce the space of functions defined on interior sides that do not coincide with elements of the lower dimension:

$$\mathring{Q}_d = \left\{ \mathring{q} \in \prod_{T \in \mathcal{T}_d} \mathring{Q}(T); \ \mathring{q}|_{\partial T} = \mathring{q}|_{\partial \tilde{T}} \quad \text{on the side } F = \partial T \cap \partial \tilde{T} \quad \text{if } F \cap \Omega_{d-1} = \emptyset \right\}. \tag{3.3}$$

Finally we set $\mathring{Q} = \mathring{Q}_1 \times \mathring{Q}_2 \times \mathring{Q}_3$.

The *mixed-hybrid method* for the unsteady Darcy flow reads as follows. We are looking for a trio $(\mathbf{u}, h, \mathring{h}) \in \boldsymbol{W} \times Q \times \mathring{Q}$ which satisfies the saddle-point problem:

$$a(\boldsymbol{u}, \boldsymbol{v}) + b(\boldsymbol{v}, p) + \mathring{b}(\boldsymbol{v}, \mathring{p}) = \langle g, \boldsymbol{v} \rangle, \qquad \forall \boldsymbol{v} \in \boldsymbol{W}, \tag{3.4}$$

$$b(\boldsymbol{u}, q) + \mathring{b}(\boldsymbol{u}, \mathring{q}) - c(p, \mathring{p}, q, \mathring{q}) = \langle f, (q, \mathring{q}) \rangle, \qquad \forall q \in Q, \ \mathring{q} \in \mathring{Q}, \tag{3.5}$$

where

$$a(\boldsymbol{u}, \boldsymbol{v}) = \sum_{d=1}^{3} \sum_{T \in \mathcal{T}_d} \int_T \frac{1}{\delta_d} \mathbb{K}_d^{-1} \boldsymbol{u}_d \cdot \boldsymbol{v}_d \, dx, \tag{3.6}$$

$$b(\boldsymbol{u}, q) = -\sum_{d=1}^{3} \sum_{T \in \mathcal{T}_d} \int_T q_d \, \text{div} \, \boldsymbol{u}_d \, dx, \tag{3.7}$$

$$\mathring{b}(\boldsymbol{u}, \mathring{q}) = \sum_{d=1}^{3} \sum_{T \in \mathcal{T}_d} \int_{\partial T \setminus \partial \Omega_d} \mathring{q}|_{\partial T} (\boldsymbol{u}_d \cdot \boldsymbol{n}) \, ds, \tag{3.8}$$

$$c(h, \mathring{h}, q, \mathring{q}) = c_f(h, \mathring{h}, q, \mathring{q}) + c_t(h, \mathring{h}, q, \mathring{q}) + c_R(\mathring{h}, \mathring{q}) \tag{3.9}$$

$$c_f(h, \mathring{h}, q, \mathring{q}) = \sum_{d=2,3} \sum_{T \in \mathcal{T}_d} \int_{\partial T \cap \Omega_{d-1}} \sigma_d (p_{d-1} - \mathring{p}_d)(q_{d-1} - \mathring{q}_d) \, ds \tag{3.10}$$

$$c_t(h, \mathring{h}, q, \mathring{q}) = \sum_{d=1}^{3} \sum_{T \in \mathcal{T}_d} \int_T \frac{\delta_d S_d}{\tau} h_d q_d \, dx, \tag{3.11}$$

$$c_R(\mathring{h}, \mathring{q}) = \sum_{d=1}^{3} \sum_{T \in \mathcal{T}_d} \int_{\partial T \cap \Gamma_d^{TF}} \sigma_d^R h_d \mathring{q}_d \, ds, \tag{3.12}$$

$$\langle g, \boldsymbol{v} \rangle = -\sum_{d=1}^{3} \sum_{T \in \mathcal{T}_d} \int_{\partial T \cap \partial \Omega_N} p_d^D (\boldsymbol{v} \cdot \boldsymbol{n}) \, ds, \tag{3.13}$$

$$\langle f, q \rangle = -\sum_{d=1}^{3} \int_{\Omega_d} \delta_d f_d q_d \, dx, \tag{3.14}$$

$$-\sum_{d=1}^{3} \sum_{T \in \mathcal{T}_d} \int_{\partial T \cap \Gamma_d^{TF}} q_d^N \mathring{q}_d + \sigma_d^R h_d^R \mathring{q}_d \, ds \tag{3.15}$$

$$-c_t(\tilde{h}, \mathring{\tilde{h}}, q, \mathring{q}). \tag{3.16}$$

All quantities are meant in time t, only $\tilde{h}$ is the pressure head in time $t - \tau$.

The advantage of the mixed-hybrid method is that the set of equations $(3.4) - (3.5)$ can be reduced by eliminating the unknowns $\boldsymbol{u}$ and q to a sparse positive definite system for $\mathring{q}$. This equation can then be efficiently solved using a preconditioned conjugate gradient method. Unfortunately, it appears that the resulting system does not satisfy the discrete maximum principle which in particular for short time steps τ can lead to nonphysical oscillations. One possible solution is the diagonalization of the method (lumped mixed-hybrid method, LMH) proposed in [12]. This method was implemented in Flow123d as well. It consists in replacing the form c_t by

$$c_t(h, \mathring{h}, q, \mathring{q}) = \sum_{d=1}^{3} \sum_{T \in \mathcal{T}_d} \sum_{i=1}^{d+1} \alpha_{T,i} |T| \frac{\delta_d S_d}{\tau} \left(\mathring{h}|_{S_{T,i}} \mathring{q}|_{S_{T,i}} \right),$$

and the source term $\sum_{d=1}^{3} \int_{\Omega_d} \delta_d f_d q_d \, dx$ by

$$\sum_{d=1}^{3} \sum_{T \in \mathcal{T}_d} \sum_{i=1}^{d+1} \alpha_{T,i} |T| \delta_d f_d \mathring{q}|_{S_{T,i}},$$

where $|T|$ is the size of an element T, $S_{T,i}$ is the i-th side of T, and $\mathring{h}|_{S_{T,i}}$ is the degree of freedom on the side $S_{T,i}$. Weights $\alpha_{T,i}$ can be chosen to be $1/(d+1)$. After solving the
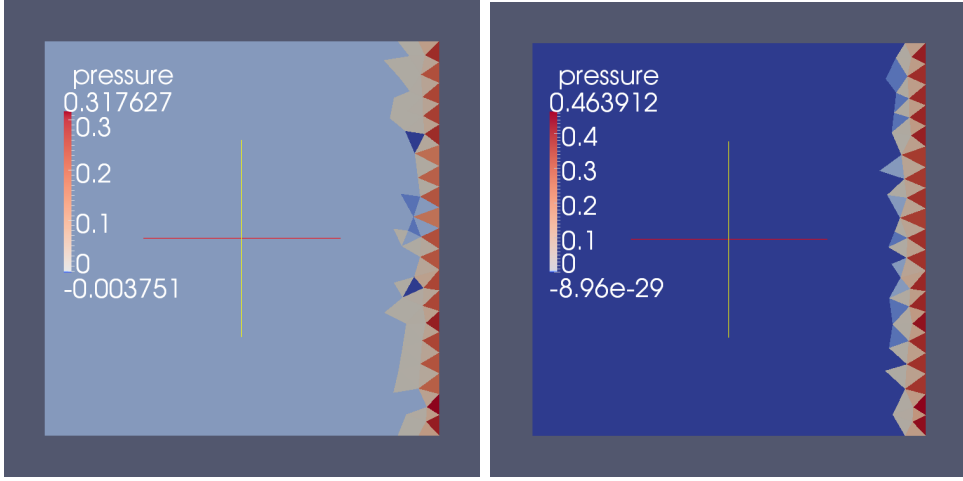
Figure 3.1: Comparison of MH (left) and LMH scheme (right), $\tau = 10^{-4}$.

set of equations it is necessary to modify the velocity field $\boldsymbol{u}$ by adding the time term. This modified system already satisfies the discrete maximum principle and does not produce oscillations. Figure 3.1 shows a comparison of the results using conventional MH scheme and LMH scheme. At the MH scheme one can observe oscillations in the wavefront where the minimum value is significantly less than zero.

3.2 Mixed-Hybrid Method on Non-conforming Mixed Meshes

The non-conforming coupling introduces a new term $c_F(h, \mathring{h}, q, \mathring{q})$ to the formulation $(3.4) - (3.5)$ similar to the term c_f responsible for the compatible coupling. We distinguish coupling of codimension $d' = 1$, i.e. 2d in 3d and 1d in 2d, and coupling of codimension $d' = 0$, 2d-2d in 3d space and 1d-1d in 2d space. This way we split c_F into

$$c_F = \sum_{d'=0,1} \sum_{d=1,2} c_{F,d',d}$$

All these terms have a common structure. For codimension 1 we have:

$$c_{F,1,d}(h, \mathring{h}, q, \mathring{q}) = \int_{T^d} \sigma_d \Big(R(\mathring{h}_d) - T(\mathring{h}_{d+1}) \Big) \Big(R(\mathring{\phi}_d) - T(\mathring{\phi}_{d+1}) \Big),$$

where R is the reconstruction operator of the pressure he and T is the trace approximation. For the sake of consistency with codimension 0, we name T_d a *master* element and intersecting elements of dimension $d + 1$ *slave* elements.

For codimension 0 we first introduce a numbering $\mathcal{S}_d$ of d dimensional manifolds (2d or 1d fractures), for every intersection line $I_{i,j}$ of two manifolds $i, j \in \mathcal{S}_d$ we define the manifold with smaller number as a *master* while the other as a *slave*. The intersection curve $I_{i,j}$ of manifolds S_i and S_j, $i < j$ is decomposed into segments corresponding to the elements of the master manifold, i.e.

$$I_{i,j} = \cup_{T \in S_i} I_{T,S_j}$$

47

With such a notation at our disposal we can write the coupling term as:

$$c_{F,0,d}(h, \mathring{h}, q, \mathring{q}) = \sum_{I_{i,j}, i<j} \sum_{T \in S_i} \int_{I_{T,S_j}} \sigma_d \big(R(\mathring{h}_i) - T(\mathring{h}_j)\big)\big(R(\mathring{\phi}_i) - T(\mathring{\phi}_j)\big),$$

where R is the trace approximation on the master element while T is the trace approximation of the slave manifold, mapping the local discrete spaces of all intersecting slave elements to the discrete space of the master element.

3.2.1 $P0$ method

Denoting $\overline{h}_T$ the average of the pressures on the edges of a master element T, i.e. components of $\mathring{h}$ and $\overline{h}_{T_i}$, $i = 1, \dots, m_T$ the average of the edge pressures on the intersecting slave elements T_i. We prescribe the operators R and T restricted to T as:

$$R(\mathring{h}) = \overline{h}_T, \quad T(\mathring{h}) = \frac{1}{\overline{\delta}_T} \sum_i \delta_{T,i} \overline{h}_{T_i}, \quad \overline{delta}_T = \sum_i \delta_{T,i}.$$

3.2.2 $P1$ method

First, we introduce local projection of the edge pressures to the linear broken space. Let us denote P_i, $i = 0, \dots, d$ the edge barycenters of an element T of dimension d. We find a basis $\phi_i(\boldsymbol{x})$, $i = 0, \dots, d$ of the space of linear functions on T that is orthogonal to the functionals $\Phi_j(\phi) = \phi(P_i)$, $j = 0, \dots, d$. Denoting $\mathring{h}_i$, $i = 0, \dots, d$ the edge values (of the pressure) on element T, we introduce the projection:

$$X_T(\mathring{h}) = \sum_i \mathring{h}_i \phi_i(\boldsymbol{x}).$$

Resulting functions of neighboring elements are continuous in the edge barycenters.

Finally, let $I_{K,L}$ be an intersection of the elements K and L. Operators R and T are defined on every such intersection independently as:

$$R(\mathring{h}) = X_K(\mathring{h}_K)|_{I_{K,L}}, \quad T(\mathring{h}) = X_L(\mathring{h}_L)|_{I_{K,L}},$$

i.e. the restriction of the linear functions on individual elements to the intersection set.

3.3 Discontinuous Galerkin Method

Models for solute transport and heat transfer described in sections 2.4 and 2.6 are collectively formulated as a system of abstract advection-diffusion equations on domains Ω_d, $d = 1, 2, 3$, connected by communication terms. Consider for $d = 1, 2, 3$ the equation

$$\partial_t u_d + \text{div}(\boldsymbol{b} u_d) - \text{div}(\mathbb{A}\nabla u_d) = f^0 + f^1(u^S - u_d) + q(u_{d+1}, u_d) \text{ in } (0,T) \times \Omega_d \quad (3.17a)$$

with initial and boundary conditions

$$u_d(0, \cdot) = u^0 \qquad\qquad \text{in } \Omega_d, \qquad\qquad (3.17b)$$

$$u_d = u^D \qquad\qquad \text{on } (0,T) \times \Gamma_d^D, \qquad\qquad (3.17c)$$

$$(\boldsymbol{b} u_d - \mathbb{A}\nabla u_d) \cdot \boldsymbol{n} = f^N + \sigma^R(u_d - u^D) \qquad\qquad \text{on } (0,T) \times \Gamma_d^N, \qquad\qquad (3.17d)$$

$$(\boldsymbol{b} u_d - \mathbb{A}\nabla u_d) \cdot \boldsymbol{n} = q(u_d, u_{d-1}) \qquad\qquad \text{on } (0,T) \times \Gamma_d^C, \qquad\qquad (3.17e)$$

where
$$\Gamma_d^C := \overline{\Omega}_d \cap \overline{\Omega}_{d-1}.$$

The communication term $q(u_{d+1}, u_d)$ has the form

$$q(u_{d+1}, u_d) = \begin{cases} \alpha u_{d+1} + \beta u_d & \text{in } \Gamma_{d+1}^C, \ d = 1, 2, \\ 0 & \text{on } \Omega_d \setminus \Gamma_{d+1}^C, \ d = 1, 2, \text{ and for } d = 0, 3. \end{cases} \tag{3.17f}$$

System (3.17) is spatially discretized by the discontinuous Galerkin method with weighted averages, which was derived for the case of one domain in [6] (for a posteriori estimate see [7]). For time discretization we use the implicit Euler method.

Let τ denote the time step. For a regular splitting $\mathcal{T}_d$ of Ω^d, $d = 1, 2, 3$, into simplices we define the following sets of element sides:

$\mathcal{E}_d$ sides of all elements in $\mathcal{T}_d$ (i.e. triangles for $d = 3$, lines for $d = 2$ and nodes for $d = 1$),

$\mathcal{E}_{d,I}$ interior sides (shared by 2 or more d-dimensional elements),

$\mathcal{E}_{d,B}$ outer sides (belonging to only one element),

$\mathcal{E}_{d,D}(t)$ sides, where the Dirichlet condition (3.17c) is given,

$\mathcal{E}_{d,N}(t)$ sides, where the Neumann or Robin condition (3.17d) is given,

$\mathcal{E}_{d,C}$ sides coinciding with Γ_d^C.

For an interior side E we denote by $\mathcal{N}_d(E)$ the set of elements that share E (notice that 1D and 0D sides can be shared by more than 2 elements). For an element $T \in \mathcal{N}_d(E)$ we denote $q_T := (\boldsymbol{b} \cdot \boldsymbol{n})_{|T}$ the outflow from T, and define $\mathcal{N}_d^-(E) := \{T \in \mathcal{N}_d(E) \,|\, q_T \leq 0\}$, $\mathcal{N}_d^+(E) := \{T \in \mathcal{N}_d(E) \,|\, q_T > 0\}$ the sets of all outflow and inflow elements, respectively. For every pair $(T^+, T^-) \in \mathcal{N}_d^+(E) \times \mathcal{N}_d^-(E)$ we define the flux from T^+ to T^- as

$$q_{T^+ \to T^-} := \frac{q_{T^+} q_{T^-}}{\sum_{T \in \mathcal{N}_d^-(E)} q_T}.$$

We select arbitrary element $T_E \in \mathcal{N}_d(E)$ and define $\boldsymbol{n}_E$ as the the unit outward normal vector to ∂T_E at E. The jump in values of a function f between two adjacent elements $T_1, T_2 \in \mathcal{N}_d(E)$ is defined by $[f]_{T_1, T_2} = f_{|T_{1|E}} - f_{|T_{2|E}}$, similarly we introduce the average $\{f\}_{T_1, T_2} = \frac{1}{2}(f_{|T_{1|E}} + f_{|T_{2|E}})$ and a weighted average $\{f\}_{T_1, T_2}^\omega = \omega f_{|T_{1|E}} + (1 - \omega) f_{|T_{2|E}}$. The weight ω is selected in a specific way (see [6]) taking into account the possible inhomogeneity of the tensor $\mathbb{A}$.

For every time step $t_k = k\tau$ we look for the discrete solution $u^k = (u_1^k, u_2^k, u_3^k) \in V$, where

$$V = \prod_{d=1}^3 V_d \quad \text{and} \quad V_d = \{v : \overline{\Omega^d} \to \mathbb{R} \,|\, v_{|T} \in P_p(T) \ \forall T \in \mathcal{T}_d\}$$

are the spaces of piecewise polynomial functions of degree at most p on elements $\mathcal{T}_d$, generally discontinuous on interfaces of elements. The initial condition for u_d^0 is defined as the L^2-projection of u^0 to V_d. For $k = 1, 2, \ldots$, u^k is given as the solution of the problem

$$\frac{1}{\tau} \left(u^k - u^{k-1}, v \right)_V + a^k(u^k, v) = b^k(v) \quad \forall v \in V.$$

Here $(f,g)_V = \sum_{d=1}^d (f,g)_{\Omega^d}$, $(f,g)_{\Omega^d} = \int_{\Omega^d} fg$, and forms a^k, b^k are defined as follows:

$$a^k((u_1, u_2, u_3), (v_1, v_2, v_3))$$
$$= \sum_{d=1}^3 \left(a_d^k(u_d, v_d) - (q(u_{d+1}, u_d), v_d)_{\Omega^d} - \sum_{E \in \mathcal{E}_{d,C}^d(t_k)} (q(u_d, u_{d-1}), v_d)_E \right), \quad (3.18)$$

$$b^k((v_1, v_2, v_3)) = \sum_{d=1}^3 b_d^k(v_d),$$
$$(3.19)$$

$$a_d^k(u, v) = (\mathbb{A}\nabla u, \nabla v)_{\Omega^d} - (\boldsymbol{b}u, \nabla v)_{\Omega^d} + \left(f^1 u, v \right)_{\Omega^d}$$
$$- \sum_{E \in \mathcal{E}_{d,I}^d} \sum_{\substack{T_1, T_2 \in \mathcal{N}_d(E) \\ T_1 \neq T_2}} \left(\left(\{\mathbb{A}\nabla u\}_{T_1, T_2}^\omega \cdot \boldsymbol{n}_E, [v]_{T_1, T_2} \right)_E + \Theta \left(\{\mathbb{A}\nabla v\}_{T_1, T_2}^\omega \cdot \boldsymbol{n}_E, [u]_{T_1, T_2} \right)_E \right.$$
$$\left. - \gamma_E \left([u]_{T_1, T_2}, [v]_{T_1, T_2} \right)_E \right) - \sum_{E \in \mathcal{E}_{d,I}^d} \sum_{\substack{T^+ \in \mathcal{N}_d^+(E) \\ T^- \in \mathcal{N}_d^-(E)}} \left(q_{T^+ \to T^-} \{u\}_{T^+, T^-}, [v]_{T^+, T^-} \right)_E$$
$$+ \sum_{E \in \mathcal{E}_{d,D}^d(t_k)} \left(\gamma_E (u, v)_E + (\boldsymbol{b} \cdot \boldsymbol{n}u, v)_E - (\mathbb{A}\nabla u \cdot \boldsymbol{n}, v)_E - \Theta (\mathbb{A}\nabla v \cdot \boldsymbol{n}, u)_E \right)$$
$$+ \sum_{E \in \mathcal{E}_{d,N}^d(t_k)} \left(\sigma^R u, v \right)_E,$$

$$b_d^k(v) = \left(f^0 + f^1 u^S, v \right)_{\Omega^d} + \sum_{E \in \mathcal{E}_{d,D}^d(t_k)} \left(\gamma_E \left(u^D, v \right)_E - \Theta \left(u^D, \mathbb{A}\nabla v \cdot \boldsymbol{n} \right)_E \right)$$
$$+ \sum_{E \in \mathcal{E}_{d,N}^d(t_k)} \left(\sigma^R u^D - f^N, v \right)_E.$$

The Dirichlet condition is here enforced by a penalty with an arbitrary parameter $\gamma_E > 0$; its value influences the level of solution's discontinuity. For $\gamma_E \to +\infty$ we obtain asymptotically (at least formally) the finite element method. The constant Θ can take the values -1, 0 or 1, where -1 corresponds to the nonsymmetric, 0 to the incomplete and 1 to the symmetric variant of the discontinuous Galerkin method.

3.4 Finite Volume Method for Convective Transport

In the case of the purely convective solute transport ($\mathbb{D} = 0$), problem (3.17) is replaced by:

$$\partial_t u_d + \operatorname{div}(\boldsymbol{b}u_d) = f^0 + f^1(u^S - u_d) + q(u_{d+1}, u_d) \qquad \text{in } (0, T) \times \Omega_d, \qquad (3.20\text{a})$$
$$u_d(0, \cdot) = u^0 \qquad \text{in } \Omega_d, \qquad (3.20\text{b})$$
$$(\boldsymbol{b} \cdot \boldsymbol{n})u_d = (\boldsymbol{b} \cdot \boldsymbol{n})u^D \qquad \text{on } \Gamma_d^I, \qquad (3.20\text{c})$$

where

$$\Gamma_d^I := \{(t, \boldsymbol{x}) \in (0, T) \times \partial\Omega_d \,|\, \boldsymbol{b}(t, \boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) < 0\}.$$

The communication term $q(u_{d+1}, u_d)$ has the same structure as in (3.17f).

The system is discretized by the cell-centered finite volume method combined with the explicit Euler time discretization. Using the notation of Section 3.3, we consider the space V of piecewise constants on elements and define the discrete problem:

$$\frac{1}{\tau}\left(u^k - u^{k-1}, v\right)_V + a^{k-1}(u^{k-1}, v) = b^{k-1}(v) \quad \forall v \in V,$$

where the forms a^k and b^k are defined in (3.18)-(3.19) and a_d^k, b_d^k now have simplified form:

$$a_d^k(u, v) = -\sum_{T_i \in \mathcal{T}_d}\left(\left((\boldsymbol{b}\cdot\boldsymbol{n})^+ u, v\right)_{\partial T_i} + \sum_{T_j \in \mathcal{T}_d}\left(q_{T_j \to T_i}u, v\right)_{\partial T_i \cap \partial T_j}\right),$$

$$b_d^k(v) = \left(f^0 + f^1(u^S - u_d^{k-1})^+, v\right)_{\Omega^d} + \sum_{T_i \in \mathcal{T}_d}\left((\boldsymbol{b}\cdot\boldsymbol{n})^- u^D, v\right)_{\partial T_i \cap \partial \Omega_d}.$$

The above formulation corresponds to the upwind scheme, ideal mixing in case of multiple elements sharing one side, and explicit treatment of linear source term.

3.5 Solution Issues for Reaction Term

3.5.1 Dual Porosity

The analytic solution of the system of differential equations (2.34) at the time t with initial conditions $c_m(0)$ and $c_i(0)$ is

$$c_m(t) = (c_m(0) - c_a(0))\exp\left(-D_{dp}\left(\frac{1}{\vartheta_m} + \frac{1}{\vartheta_i}\right)t\right) + c_a(0), \tag{3.21}$$

$$c_i(t) = (c_i(0) - c_a(0))\exp\left(-D_{dp}\left(\frac{1}{\vartheta_m} + \frac{1}{\vartheta_i}\right)t\right) + c_a(0), \tag{3.22}$$

where c_a is the weighted average

$$c_a = \frac{\vartheta_m c_m + \vartheta_i c_i}{\vartheta_m + \vartheta_i}.$$

If the time step is large, we use the analytic solution to compute new values of concentrations. Otherwise, we replace the time derivatives in (2.34a) and (2.34b) by first order forward differences and we get the classical Euler scheme

$$c_m(t^+) = \frac{D_{dp}\Delta t}{\vartheta_m}(c_i(t) - c_m(t)) + c_m(t), \tag{3.23a}$$

$$c_i(t^+) = \frac{D_{dp}\Delta t}{\vartheta_i}(c_m(t) - c_i(t)) + c_i(t), \tag{3.23b}$$

$$\tag{3.23c}$$

where $\Delta t = t^+ - t$ is the time step.

The condition on the size of the time step is derived from the Taylor expansion of (3.21) or (3.22), respectively. We neglect the higher order terms and we want the second order term to be smaller than the given scheme tolerance *tol*, relatively to c_a,

$$(c_m(0) - c_a(0)) \frac{D_{dp}^2 (\Delta t)^2 \left(\frac{\vartheta_m + \vartheta_i}{\vartheta_m \vartheta_i} \right)^2}{2} \frac{1}{c_a} \leq tol. \tag{3.24}$$

We then transform the above inequation into the following condition which is tested in the program

$$\max(|c_m(0) - c_a(0)|, |c_i(0) - c_a(0)|) \leq 2c_a \left(\frac{\vartheta_m \vartheta_i}{D_{dp} \Delta t (\vartheta_m + \vartheta_i)} \right)^2 tol. \tag{3.25}$$

In addition, the explicit Euler method (3.23) requires the satisfaction of a CFL condition of the form

$$\Delta t \leq \frac{1}{D_{dp}} \frac{\vartheta_m \vartheta_i}{\vartheta_m + \vartheta_i}. \tag{3.26}$$

If either of the inequalities (3.25) or (3.26) is not satisfied, then the analytic solution is used.

3.5.2   Equilibrial Sorption

Let us now describe the actual computation of the sorption model. To solve (2.37) iteratively, it is very important to define the interval where to look for the solution (unknown c_l), see Figure 3.2. The lower bound is 0 (concentration can not reach negative values). The upper bound is derived using a simple mapping. Let us suppose limited solubility of the selected transported substance and let us denote the limit $\bar{c}_l$. We keep the maximal "total mass" $\bar{c}_T = \mu_l \cdot \bar{c}_l + \mu_s \cdot f(\bar{c}_l)$, but we dissolve all the mass to get maximal $c_l^{max} > \bar{c}_l$. That means $c_s = 0$ at this moment. We can slightly enlarge the interval by setting the upper bound equal to $c_l^{max} + const_{small}$.
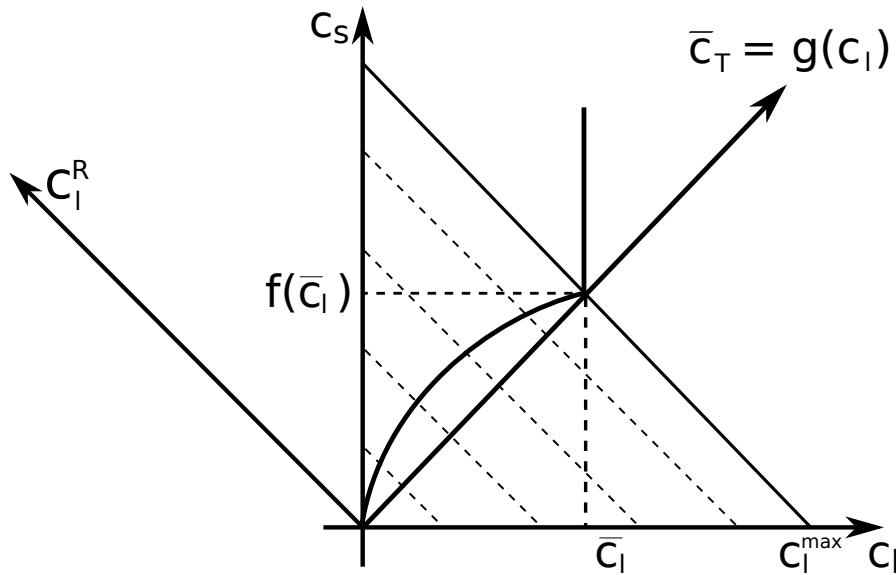

Figure 3.2: Sorption in combination with limited solubility.

To approximate the equation (2.37) using interpolation, we need to prepare the set of values which represents $[c_l, f(c_l)]$, with c_l equidistantly distributed in transformed (rotated and rescaled) coordination system at first. The construction process of the interpolation table follows.

1. Maximal "total mass" $\bar{c}_T = \mu_l \cdot \bar{c}_l + \mu_s \cdot f(\bar{c}_l)$ is computed.

2. Total mass step is derived $mass\_step = \bar{c}_T / n\_steps$. n_steps is the number of substeps.

3. Appropriate $c_T^j = (mass\_step \cdot j)/\mu_l$, $j \in \{0, \ldots, n\_steps\}$ are computed.

4. The equations $\mu_l \cdot c_T^j = \mu_l \cdot c_l^j + \mu_s \cdot f(c_l^j)$ $j \in \{0, \ldots, n\_steps\}$ are solved for c_l^j as unknowns. The solution is the set of ordered couples (points) $[c_l^j, f(c_l^j)]$, $j \in \{0, \ldots, n\_steps\}$.

After the computation of $\{[c_l^j, f(c_l^j)]\}$, we transform these coordinates to the system where the total mass is an independent variable. This is done by multiplication of precomputed points using the transformation matrix $\mathbf{A}$:

$$
\begin{aligned}
&\vec{c}^R = \mathbf{A} \cdot \vec{c} \\
&\begin{bmatrix} c_l^{R,j} \\ c_s^{R,j} \end{bmatrix} = \begin{bmatrix} \vartheta \cdot \rho_w & M_s(1 - \vartheta)\rho_R \\ -M_s(1 - \vartheta)\rho_R & \vartheta \cdot \rho_w \end{bmatrix} \cdot \begin{bmatrix} c_l^j \\ c_s^j \end{bmatrix} \\
&j \in \{0, \ldots, n\_steps\}
\end{aligned}
\tag{3.27}
$$

The values $c_l^{R,j}$ are equidistantly distributed and there is no reason to save them, but the values $c_s^{R,j}$ are stored in one-dimensional interpolation table.

Once we have the interpolation table, we can use it for projecting the transport results $[c_l, c_s]$ on the isotherm under consideration. Following steps must be taken.

1. Achieved concentrations are transformed to the coordinate system through multiplication with the matrix $\mathbf{A}$, see (3.27).

2. Transformed values are interpolated.

3. The result of interpolation is transformed back. The backward transformation consists of multiplication with $\mathbf{A}^T$ which is followed by rescaling the result. Rescaling the result is necessary because $\mathbf{A}$ is not orthonormal as it is shown bellow.

$$
\mathbf{A}^T \cdot \mathbf{A} = ((\vartheta - 1)^2 \cdot M_s^2 \cdot \rho_R^2 + \vartheta^2 \cdot \rho_w^2) \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
$$

Limited solubility. When $\mu_l \cdot c_l + \mu_s \cdot f(c_l) > \mu_l \cdot \bar{c}_l + \mu_s \cdot f(\bar{c}_l)$, neither iterative solver nor interpolation table is used. The aqueous concentration is set to be $\bar{c}_l$ and sorbed concentration is computed $c_s = (\mu_l \cdot c_l + \mu_s \cdot f(c_l) - \mu_l \cdot \bar{c}_l)/\mu_s$.

3.5.3 System of Linear Ordinary Differential Equations

A system of linear ordinary differential equations (ODE) appears in several places in the model. Let us denote the ODE system

$$\partial_t \boldsymbol{c}(t) = \mathbf{A}(t)\boldsymbol{c}(t) + \boldsymbol{b}(t).$$

For the moment the only implemented method to solve the system is usage of Padé approximant which corresponds to a family of implicit R-K methods.

Padé approximant. For homogeneous systems with constant matrix $\mathbf{A}$, we can use Padé approximation to find the solution. This method finds a rational function whose power series agrees with a power series expansion of a given function to the highest possible order (e.g. in [11]). Let

$$f(t) = \sum_{j=0}^{\infty} c_j t^j = \sum_{j=0}^{\infty} \frac{1}{n!} f^{(j)}(t_0)$$

be the function being approximated and its power series given by Taylor expansion about t_0. Then the rational function

$$R_{mn}(t) = \frac{P_m(t)}{Q_n(t)} = \frac{\sum_{j=0}^{m} p_j t^j}{\sum_{j=0}^{n} q_j t^j}, \tag{3.28}$$

which satisfies

$$f(t) \approx \sum_{j=0}^{m+n} c_j t^j = R_{mn}(t), \tag{3.29}$$

is called Padé approximant. From (3.29), we obtain $m + n + 2$ equations for coefficients of the nominator P_m (polynomial of degree m) and the denominator Q_n (polynomial of degree n). We also see that the error of the approximation is $O(t^{m+n+1})$. By convention, the denominator is normalized such that $q_0 = 1$. Theoretical results show that for $m = n - 1$ and $m = n - 2$ the Padé approximant corresponds to an implicit Runge-Kutta method which is A-stable and L-stable (see [5]).

Now, we consider the solution of our ODE system in a form $\boldsymbol{c}(t) = e^{\mathbf{A}t}\boldsymbol{c}(0)$. We shall approximate the matrix exponential function using a matrix form of (3.28). For exponential functions, there are known coefficients of the nominator and denominator:

$$\mathbf{P}_m(\mathbf{A}t) = \sum_{j=0}^{m} \frac{(m+n-j)!m!}{(m+n)!j!(m-j)!}(\mathbf{A}t)^j, \tag{3.30}$$

$$\mathbf{Q}_n(\mathbf{A}t) = \sum_{j=0}^{n} (-1)^j \frac{(m+n-j)!n!}{(m+n)!j!(n-j)!}(\mathbf{A}t)^j. \tag{3.31}$$

Finally, we can write the solution at time $t + \Delta t$

$$\boldsymbol{c}(t + \Delta t) = \frac{\mathbf{P}_m(\mathbf{A}\Delta t)}{\mathbf{Q}_n(\mathbf{A}\Delta t)}\boldsymbol{c}(t) = \mathbf{R}_{mn}(\mathbf{A}\Delta t)\boldsymbol{c}(t). \tag{3.32}$$

If the time step Δt is constant, we do not need to compute the matrix $\mathbf{R}_{mn}$ repeatedly and we get the solution cheaply just by matrix multiplication. In the opposite case, we avoid evaluating the exponential function and still get the solution quite fast (comparing to computing semi-analytic solution).

Chapter 4

File Formats

In this chapter, we shall describe structure of the main input file and data formats of other input files. In particular we briefly describe the GMSH file format used for both the computational mesh as well as for the input of general spatial data.

4.1 Main Input File

In this section, we shall describe structure of the main input file that is given either as the first positional argument or through the parameter `-s` on the command line. First, we provide a quick introduction to the YAML file format. Then, we demonstrate the most important input structures on the examples.

4.1.1 YAML basics

YAML is a human readable data format. It is designed to be both human readable and human editable. As it is not a markup languages, there are no tags to determine type of the data. The indentation and justification is used instead for data organization. Moreover the used YAML format (version 1.2) is superset of the JSON format, another minimalist data serialization format where braces and brackets are used instead of indentation. For the more detailed description see Wikipedia for further technical details and for reference parsers for various programming languages see YAML home page .

Hierarchy of Mappings and Lists

Elementary data are organized to lists and mappings. Let us start with an example of a list:

```
# Example of list
- 3.14                    # a number
- 2014-01-14              # a date
- Simple string.          # a string
- "3 is three"            # quoting necessary
- '3 is three'            # other quoting
- true                    # boolean
```

Comments are started by a hash (#) which can appear anywhere on a line and marks the comment up to the end of line. The the list follows with single item per line preceded by a dash (-). Any value starting by a digit is interpreted as a number or date. Values starting with letter is interpreted as a string. Otherwise one may use double ("") or single ('') quotas to mark a string value explicitly. Finally some strings are interpreted as Boolean values, it is recommended to use `true` and `false` (other possible pairs are e.g. `yes/no`, `y/n`, `on/off`).

Alternatively a list may be written in compact "JSON" way enclosing the list into brackets:

```
# Compact list
[ 3.14, 2014-01-14, Simple string.,
"3 is three", '3 is three']
```

Other data structure is called mapping, which is also known as directory or associative array:

```
# Example of a mapping
pi: 3.14
date: 2014-01-14
name: Mr. Simple String
```

Again one may use also JSON syntax with mapping enclosed in braces:

```
# Compact mapping
{ pi: 3.14, date: 2014-01-14,
name: Mr. Simple String }
```

Mappings and lists may by mutually nested:

```
list:
    - one
    - two
    -
        - three one
        - three two
map:
    a: one
    b: two
```

A string may be split to more lines using *greater then* (>) and multi-line strings may be entered after *vertical line* (|):

```
# single long string
one_line: >
    Single line string
    broken into two lines.
# multi line string
multi_line: |
    First line.
    Second line.
```

In the first case the line breaks are replaced by space, in the second case the line breaks are preserved. In both cases the leading indentation is removed.

Tags

YAML format defines a syntax for explicit specification of types of values including the types specific to an application. The application specific tags are used by Flow123d to specify particular implementation of various algorithms or data types. The general syntax of tags is quite complicated, so we present only the syntax used in the Flow123d input.

```
field_a: !FieldFormula
    value: !!str "5 * x"
field_b:  !FieldFormula "5 * x"
```

The `field_a` have specified evaluation algorithm `FieldFormula`, the key value have explicitly specified the default tag `str`. Note that default types are detected automatically and need not to be specified. On the third line we use even more compact way to express the same thing. Further details about usage of tags in Flow123d follows in Section 4.1.2.

References

Anchors and references allows to reduce redundancy in the input data:

```
aux_name: &anchor_name John Smith
aux_man: &common_man
    sex: man
    city: Prague

people:
    - << *_common_man
      name: John Paul
    - << *_common_man
      name: *anchor_name
```

On the first line, we define the anchor `&anchor_name` for the value `John Smith`. On the second line, we define the anchor `&common_man` for the dictionary. Later, we use `<<` to inject the dictionary referenced by `*common_man`. Finally the anchor `&anchor_name` is referenced by `*anchor_name` to reuse the name `John Smith`.

Ignoring the auxiliary keys `aux_name` and `aux_man` this is equivalent to:

```
people:
    - sex: man
      city: Prague
      name: John Paul
    - sex: man
      city: Prague
      name: John Smith
```

Gotchas

- Unquoted string values can not contain characters: colon :, hash #, brackets [], braces {}, less then <, vertical bar |.

- For indentation one can use only spaces; tabs are not allowed. However, your editor may automatically convert tabs to spaces.

- Boolean special strings must be quoted if you want to express a string. This is not problem for the Flow123d input.

- Numbers starting with digit zero are interpreted as octal numbers.

4.1.2 Flow123d input types

Flow123d have a subsystem for definition of the structure of the input file including preliminary checks for the correctness of the values. This subsystem works with elementary data types:

- *Bool* corresponds to the YAML Boolean values

- *Double*, *Integer* initialized from YAML numeric values.

- *String*, *FileName*, *Selections* initialized from YAML strings.

Numerical values have defined valid ranges. FileName values are used for reference to external files either for input or for output. Selection type defines a finite number of valid string values which may appear on the input. These elementary types are further organized into Records and Arrays in order to specify strongly typed definition of the input data file. Array and Records forms so called input structure tree (IST).

In order to make *"simple things simple and complex things possible" (Alan Kay)* the input system provides so called *automatic conversions*. If the YAML type on input does not match the expected data type automatic conversion tries to convert the input to the expected type. Automatic conversion rules for individual composed types follows.

Record (YAML Mapping, JSON object)

A Record is initialized from the YAML mapping. However, in contrast to YAML mappings the Records have fixed keys with fixed types. This is natural as Records are used for initialization of C++ objects which are also strongly typed. Every Record type have unique name and have defined list of its keys. Keys are lowercase strings without spaces, possibly using digits and underscore. Every key has a type and default value specification. Default value specification can be:

obligatory — means no default value, which has to be specified at input.

optional — means no default value, but value is needs not to be specified. Unspecified value usually means that you turn off some functionality.

default at declaration — the default value is explicitly given in declaration and is automatically provided by the input subsystem if needed

default at read time — the default value is provided at read time, usually from some other variable. In the documentation, there is only textual description where the default value comes from.

Records that have all keys with default value or optional safe the single key K may support autoconversion from an input of the type that match the type of the key K. For example:

```
mesh: "mesh_file.msh"
```

is converted to:

```
mesh:
  mesh_file: "mesh_file.msh"
  regions: null
  partitioning: any_neighboring
  print_regions: false
  intersection_search: BIHsearch
```

with the key `regions` being optional and the last three keys having its default values.

Array (YAML List, JSON array)

An Array is initialized from a YAML list. But, in opposition to the YAML mapping, the values in a single Array have all the same type. So the particular Array type is given by the type of its elements and a specification of its size range.

Automatic conversion performs kind of transposition of the data. It simplifies input of the list of records (or arrays) with redundant structure, e.g. consider input

```
list:
  a: [1,2)
  b: 4
  c: [5,6]
```

Assuming that key `list` have the type Array of Records and keys `a`, `b`, `c` are all numerical scalars this input is equivalent to

```
list:
  - a: 1
    b: 4
    c: 5
  - a: 2
    b: 4
    c: 6
```

The rule works as follows, if a key K should have type Array, but some other type is on the input, we search through the input under the key K for all Arrays S standing instead of scalars. All these arrays must have the same length n. Then the i-th element of the key A array is copy of the input keeping only i-th elements of the Arrays S. As a special case if there are no Arrays S a list with single element equal to the input is created. Only this simplest conversion to an Array is applied if inappropriate type is found while the transposition is processed.

Abstract

An Abstract type allows a certain kind of polymorphism corresponding to a pure abstract class in C++ or to an interface in Java. Every Abstract type have unique name and set of Records that implements the Abstract. The particular type must be provided on input through the YAML tag. See description of Fields below for examples.

An Abstract type may have specified the default implementation. If this default Record supports automatic conversion from one of its keys we can see it as an automatic conversion from that key to the Abstract. For example

```
conductivity: 2.0
```

where conductivity is of Abstract type `Field` with scalar values, is in fact converted to

```
conductivity: !FieldConstant
    value: 2.0
```

as the `FieldConstant` is default implementation of the field and it is auto=convertible from the key `value`.

Flow123d specific tags

Currently just two specific tags are implemented, both allowing inclusion of data in other files.

Include other YAML file The tag `!include` can be used to read a key value from other YAML file. Path to the file is specified as the value of the key. A relative path is rooted in the folder of the main input file. A particular type of an Abstract key is specified as a composed tag `!include,<TYPE>`.

Example, the main input file:

```
flow123d_version: 2.0.0
problem: !Coupling_Sequential
    description: Test8 - Steady flow with sources
    mesh:
        mesh_file: ../00_mesh/square_1x1_shift.msh
    flow_equation: !include,Flow_Darcy_MH
        input_darcy.yaml
```

Content of `input_darcy.yaml`, included Record:

```
nonlinear_solver:
    linear_solver: !Petsc
input_fields:
    darcy_input_fields.yaml
balance: {}
output_stream:
    file: ./flow.pvd
```

Content of `darcy_input_fields.yaml`, included Array:

```
- region: plane
    anisotropy: 1
    water_source_density: !FieldFormula
    value: 2*(1-x^2)+2*(1-y^2)
- region: .plane_boundary
    bc_type: dirichlet
    bc_pressure: 0
```

Include general CSV data The custom tag `include_csv` can be used to include a table (e.g. coma separated values, CSV file) as an Array of Records. Every line of the input table is converted to a single element of the Array. The tag is followed by a Record with several keys to specify format of the data:

`file`
 A valid path to a text data file. Relative to the main input file.

`separator`
 A string of characters used as separators of the values on the single line (default is coma ","). Tab and space are always added. Double quotas can be used to express string values containing separators, backslash can be used to escaping any character with special meaning. Consecutive row of separators is interpreted as a single separator.

`n_head_lines`
 Skip given number of lines at the beginning.

`format`
 An input structure of a single element in the resulting array. Type of Abstracts must be same through the whole resulting Array. String scalar values with a place-holder '`$<column>`' will be replaced by the value at corresponding column in the input file.

Current implementation have substantial limitation as it can not be combined with auto conversions. This makes these includes little bit verbose. For example consider this section from a main input file:

```
...
substances: [A, B]
...
input_fields:
```

```
        - region: A
          porosity: !FieldTimeFunction
            time_function: !include_csv
              values:
                file: data.txt
                separator: " "
                n_head_lines: 1
                format:
                  time: #0
                  value: #1


        - region: .boundary_A
          bc_conc:
            - !FieldTimeFunction    # Substance A
              time_function: !include_csv
                values:
                  file: data.txt
                  separator: " "
                  n_head_lines: 1
                  format:
                    time: #0
                    value: #2
            - !FieldTimeFunction    # Substance B
              time_function: !include_csv
                values:
                  file: data.txt
                  separator: " "
                  n_head_lines: 1
                  format:
                    time: #0
                    value: #3
```

Content of `data.txt`:

```
time    porosity        bc_conc_X       bc_conc_Y
0.0     0.01            1.0             0.6
0.1     0.015           0.9             0.5
0.2     0.03            0.8             0.4
```

This together will be equivalent to:

```
    input_fields:
    - region: A
      porosity: !FieldTimeFunction
        time_function:
          - time: 0.0
            value: 0.01
          - time: 0.1
            value: 0.015
```

```
        - time: 0.2
          value: 0.03

  - region: .boundary_A
    bc_conc: !FieldTimeFunction
      time_function:
        - time: 0.0
          value: [ 1.0, 0.6]
        - time: 0.1
          value: [ 0.9, 0.5]
        - time:  0.2
          value: [ 0.8, 0.4]
```

So in this particular case it would be simpler to write data directly into the main file. The include from the text table is efficient for the long time series.

4.1.3   Input subsystem

This section provides some implementation details about the Flow123d input subsystem. This may be helpful to better understand behavior of the program for some special input constructions.
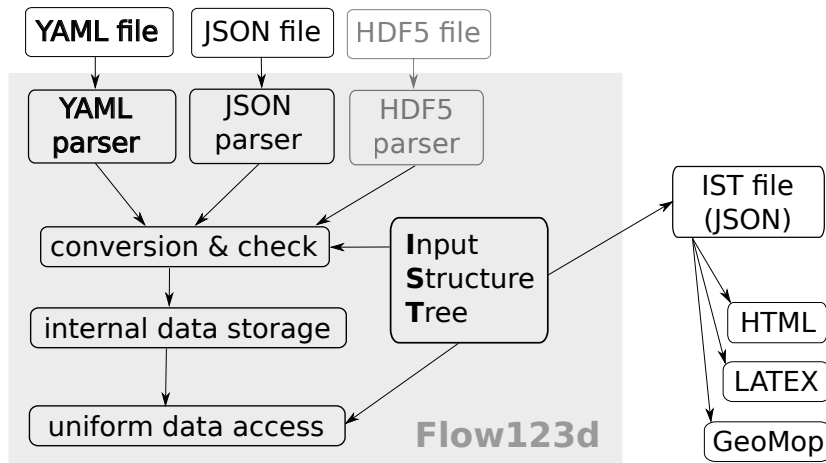
Figure 4.1: Structure of the input subsystem. HDF5 format not yet supported.

The input subsystem of Flow123d is designed with the aim to provide uniform initialization of C++ classes and data structures. The scheme of the input is depicted on Figure 4.1. The structure of the input file is described by the Input Structure Tree (IST) consisting of the input objects describing the types discussed in the previous Section 4.1.2. The structure of the tree mostly follows follows the structure of the computational classes.

When reading the input file, the file is first parsed by the specific format parser. Using a common interface to the parsed data, the structure of the data is checked against the IST and the data are pushed into the storage tree. If the input data and IST do not match the automatic conversions described above are applied, where appropriate. An

accessor object to the root data record is the result of the file reading. The data can be retrieved through the accessors which combine raw data of the storage with their meaning described in IST. The IST can be written out in the JSON format providing the description of the input file structure. This IST file is used both for generation of the input reference in HTML and LaTeXformats and for the Model editor — specialized editor for the input file that is part of the GeoMop tools currently in development.

While the recommended format of the input file is YAML the JSON format can be used as well. This may be useful in particular if the input file should be machine generated. Although the JSON format is technically subset of the YAML format. We use separate parser and use special keys in order to mimic tags and references supported by the YAML. The type of an abstract is specified by the key `TYPE`. A reference is given by a record with the only key `REF` which contains a string specifying the address of the value that should be substituted.

4.2   Important Record Types of Flow123d Input

Complete description of the input structure tree can be generated into HTML or LaTeX format. While the former one provides better interactivity through the hyperlinks the later one is part of this user manual. The generated documentation provides whole details for all keys, but it may be difficult to understand the concept of the input structures. This section is aimed to provide this higher level picture.

4.2.1   Mesh Record

The mesh record provides initialization of the computational mesh consisting of points, lines, triangles and tetrahedrons in the 3D ambient space. Currently, we support only GMSH mesh file format MSH ASCII. The input file is provided by the key `mesh_file`. The file format allows to group elements into *regions* identified by a unique label (or by ID number). The regions with labels starting with the dot character are treated as *boundary regions*. Their elements are removed from the computational domain, however they can be used to specify boundary conditions. Other regions are called *bulk regions*. Every element lies directly in just one *simple region* while the simple regions may be grouped into composed regions called also region sets. A simple region may be part of any number of composed regions. Initial assignment of the simple regions to the elements is given by the physical groups of the input GMSH file. Further modification of this assignment as well as creation of new simple or composed regions can be done through the list of operations under the key `regions`. The operations are performed in the order given by the input. Operation `From_Id` sets the name of a simple region having only ID in the input GMSH file. Operation `From_Label` can rename a simple region. Operation `From_Elements` assign new simple region to the given list of elements over-writing their region given by the input mesh file. Finally operations `Union`, `Difference` and `Intersection` implements standard set operations with both simple and complex regions resulting in new composed regions.

4.2.2 Input Fields

Input of every equation contains the key `input_fields` used consistently for the input of the equation parameters in form of general time–space dependent fields. The input fields are organized into a list of *field descriptors*, see e.g. Data record, the field descriptor of the Darcy flow equation. The field descriptor is a Record with keys `time`, `region`, `rid` and further keys corresponding to the names of input fields supported by the equation. The field descriptor is used to prescribe a change of one or more fields in particular time (key `time`) and on particular region given by the name (key `region`, preferred way) or by the region id (key `rid`). The array is processed sequentially and latter values overwrite the previous ones. Change times of a single field must form a non-decreasing sequence. Changes in fields given by the fields descriptor are interpreted as discontinuous changes of the changed fields and equations try to adopt its time stepping to match these time points. This is in contrast with changes of the field values given by the evaluation algorithms, these are always assumed to be continuous and the time steps are not adapted.

Example:

```
input_fields:
  - # time=0.0  - default value
    region: BULK
    conductivity: 1   # setting the conductivity field on all regions
  - region: 2d_part
    conductivity: 2  # overwriting the previous value
  - time: 1.0
    region: 2d_part
    conductivity: !FieldFormula
      # from time=1.0 we switch to the linear function in time
      value: 2+t
  - time: 2.0
    region: 2d_part
    conductivity: !FieldElementwise
      # from time=2.0 we switch to elementwise field, but only
      # on the region "2d_part"
      gmsh_file: ./input/data.msh
      field_name: conductivity
```

Field Algorithms

A general time and space dependent, scalar, vector, or tensor valued function can be specified through the family of abstract records `Field:R3 -> X`, where X is a type of value returned by the field, which can be:

- T — scalar valued field, with scalars of type T

- $T[d]$ — vector valued field, with vector of fixed size d and elements of type T

- $T[d, d]$ — tensor valued field, with square tensor of fixed size and elements of type T

the scalar type T can be one of

- **Real** — scalar real valued field

- **Int** — scalar integer valued field

- **Enum** — scalar non negative integer valued field. Values on the input are of the type Selection.

Each of these abstract records have the same set of descendants which implement various evaluation algorithms of the fields. These are

FieldConstant — field that is constant both in space and time

TimeFunction — field that is constant in space and continuous in time. Values are interpolated (currently only linear interpolation) from the sequence of time-value pairs provided on input.

FieldFormula — field that is given by a runtime parsed formula. Since version 3.1.0 we use our own library BParser in the equations SoluteAdvectionDiffusionDG and Elasticity. while the SoluteConvectionFV and DarcyFlowMH still use FParser. Formulas are converted from the FParser syntax to that of BParser for the sake of backward compatibility, however, we can not rule out minor incompatibilities. The BParser will be used exclusively from version 4.0.0 with its Python and Numpy syntax, support for vector and tensor valued expressions and use SIMD operations to achieve nearly peak CPU performance.

The formula can contain following literals: constants `E` and `Pi`, names of other fields of the same equation, and coordinates: `t`, `x`, `y`, `z`, `d`. First is the simulation time, the coordinate d is a spefial field that evaluates to the depth from the surface. This is defined as a distance to the intersection of the vertical (Z axis) line with the outer boundary that has smallest Z coordinate greater then the evaluation point.

FieldPython — field can be implemented by Python script either specified by string (key `script_string`) or in external file (key `script_file`.

FieldElementwise — discrete field, currently only piecewise constant field on elements is supported, the field can given by the MSH ASCII file specified in key `gmsh_file` and field name in the file given by key `field_name`. The file must contain same mesh as is used for computation.

FieldInterpolated — allows interpolation between different meshes. Not yet fully supported.

Several automatic conversions are implemented. Scalar values can be used to set constant vectors or tensors. Vector value of size d can be used to set diagonal tensor $d \times d$. Vector value of size $d(d-1)/2$, e.g. 6 for $d = 3$, can be used to set symmetric tensor. These rules apply only for FieldConstant and FieldFormula. Moreover, all Field abstract types have default value `TYPE=FieldConstant`. Thus you can just use the constant value instead of the whole record.

Examples:

```
input fields:
   - conductivity: 1.0
     # is equivalent to
   - conductivity: !FieldConstant
        value=1.0


   - anisotropy: [1 ,2, 3] # diagonal tensor
     # is equivalent to
   - anisotropy: !FieldConstant
        value=[[1,0,0],[0,2,0],[0,0,3]]


     # concentration for 2 components
   - conc: !FieldFormula  ["x+y", "x+z"]
     # is equivalent to
   - conc:
        - !FieldFormula
          value: "x+y"
        - !FieldFormula
          value: "x+z"
```

Field Units

Every field (e.g. conductivity or storativity) have specified unit in terms of powers of the base SI units. The user, however, may set input in different units specified by the key `unit` supported by every field algorithm. The key have type `Unit` record, auto convertible from its only key `unit_formula` of the string type. Effectively the `Unit` is a string with particular syntax. The unit formula is evaluated into a coefficient and an SI unit. The resulting SI unit must match expected SI unit of the field, while the input value of the field (including values from external files or returned by Python functions) is multiplied by the coefficient before further processing.

The syntax of unit formula is: `<UnitExpr>;<Variable>=<Number>*<UnitExpr>;...`, where `<Variable>` is a variable name and `<UnitExpr>` is a units expression which consists of products and divisions of terms, where a term has form `<Base>^<N>`, where `<N>` is an integer exponent and `<Base>` is either a base SI unit, a derived unit, or a variable defined in the same unit formula. Example, unit for the pressure head:

```
pressure_head: !FieldConstant      # [m] expected
   value: 100                      # [MPa]
   unit:  MPa/rho/g_; rho = 990*kg*m^-3; g_ = 9.8*m*s^-2
```

Standard single letter prefixes: a,f,p,n,u,m,d,c,h,k,M,G,T,P,E
are supported for the basic SI units: m,g,s,A,K,cd,mol
and for the derived SI units: N, J, W, Pa, C, D, l.

Moreover several specific units are supported: t = 1000 kg min = 60 s h = 60 min d = 24 h y = 365.2425 d

4.2.3 Output Records

Output from the models is controlled by an interplay of following records: `OutputStream`, `Balance`, and `EquationOutput`. The first two are part of the records of so called *balance equations* which provides complete description of some balanced quantity. Every such equation have its own balance output controlled by the `Balance` record and its own output stream for the spatial data controlled by the `OutputStream` record. Further every equation with its own output fields (every input field is also output field) have the `EquationOutput` record to setup output of its fields.

Balance

The balance output is performed in times given by the key `times` with type `TimeGrid` described below. Setting the key `add_output_times` to `true` the set of balance output times is enriched by the output times of the output stream of the same equation.

OutputStream

Set the file format of the output stream, possibly setting the output name, however the default value for the file name is preferred and the corresponding key `file` is obsolete. The time set provided by the optional key `times` is used as a default time set for a similar key in associated `EquationOutput` records. Finally, the key `observe_points` is used to specify observation points in which the associated equation output evaluates the observed fields.

EquationOuput

The output of the fields can be done in two ways: full spatial information saved only at selected time points in form of VTU or GMSH file, or full temporal information saved for every computational time, but only in selected output points. The list of fields for spatial output is given by key `fields` while the fields evaluated in the observation points are selected by the key `observe_fields`. The outputs times for the spatial output can be selected individually for every field in the `fields` however the default list of output times is given by the key `times` which can by optionally extended by the list of input times using the key `add_input_times`.

TimeGrid Array

An array of the `TimeGrid` records may be used to setup a sequence of times. Such sequence is used in particular to trigger various types of output. A single TimeGrid represents a regular grid of times with given start time, end time and step.

4.3 Mesh and Data File Format MSH ASCII

Currently, the only supported format for the computational mesh is MSH ASCII format used by the GMSH software. You can find its documentation on:

The scheme of the file is as follows:

```
$MeshFormat
<format version>
$EndMeshFormat

$PhysicalNames
<number of items>
<dimension>     <region ID>     <region label>
...
$EndPhysicalNames

$Nodes
<number of nodes>
<node ID> <X coord> <Y coord> <Z coord>
...
$EndNodes

$Elements
<number of elements>
<element ID> <element shape> <n of tags> <tags> <nodes>
...
$EndElements

$ElementData
<n of string tags>
    <field name>
    <interpolation scheme>
<n of double tags>
    <time>
<n of integer tags>
    <time step index>
    <n of components>
    <n of items>
    <partition index>
<element ID> <component 1> <component 2> ...
...
$EndElementData
```

Detailed description of individual sections:

PhysicalNames : Assign labels to region IDs. Elements of one region should have common dimension. Flow123d interprets regions with labels starting with period as the boundary elements that are not used for calculations.

Nodes : `<number of nodes>` is also number of data lines that follows. Node IDs are unique but need not to form an arithmetic sequence. Coordinates are float numbers.

Elements : Element IDs are unique but need not to form an arithmetic sequence. Integer code `<element shape>` represents the shape of element, we support only points (15), lines (1), triangles (2), and tetrahedrons (4). Default number of tags is 3. The first is the region ID, the second is ID of the geometrical entity (that was used in original geometry file from which the mesh was generated), and the third tag is the partition number. `nodes` is list of node IDs with size according to the element shape.

ElementData : The header has 2 string tags, 1 double tag, and 4 integer tags with default meaning. For the purpose of the `FieldElementwise` the tags `<field name>`, `<n of components>`, and `<n of items>` are obligatory. This header is followed by field data on individual elements. Flow123d assumes that elements are sorted by `element ID`, but doesn't need to form a continuous sequence.

4.4 Output Files

Flow123d supports output of scalar, vector and tensor data fields into two formats. The first is the native format of the GMSH software (usually with extension `msh`) which contains computational mesh followed by data fields for sequence of time levels. The second is the XML version of VTK files. These files can be viewed and post-processed by several visualization software packages. However, our primal goal is to support data transfer into the Paraview visualization software. See key `format`.

Input records of most equations (flow, transport, heat, some reaction models) contain the keys `output_stream` and `output`. In `output_stream`, the name and type of the output file is specified. Further, in `output`, one determines the list of fields intended for output. The available output fields include input data as well as the simulation results.

We mention the most important output fields of all equations and link to the complete lists in Table 4.1.

4.4.1 Auxiliary Output Files

Profiling Information

On every run we collect some basic profiling information. After all computations these data are written into the file `profiler%y%m%d_%H.%M.%S.out` where %y, %m, %d, %H, %M, %S are two digit numbers representing year, month, day, hour, minute, and second of the program start time.

Balance of Conservative Quantities

Primary and secondary equations can produce additional information on fluxes, sources and state of conservative quantities (for flow it is the volume of water, for transport the mass of a substance, for heat transfer the energy). The computation of balance is governed by the key `balance`. The balance file (default `water_balance.txt`, `mass_balance.txt`, `energy_balance.txt`) contains the following information:

Table 4.1: Most important output fields.

Flow_Darcy_MH and Richards_LMH	
pressure_p0	Pressure head [m], piecewise constant on every element. This field is directly produced by the MH method and thus contains no postprocessing error.
pressure_p1	Same pressure head field, but interpolated into $P1$ continuous scalar field. Namely you lost discontinuities on fractures.
velocity_p0	Vector field of water flux $[\mathrm{m^3 s^{-1}}]$. For every element we evaluate discrete flux field at barycenter.
piezo_head_p0	Piezometric head [m], piecewise constant on every element. This is just pressure on element plus z-coordinate of the barycenter. This field is produced only on demand (see key piezo_head_p0).
complete list	See Darcy flow output fields.
Solute_Advection_FV	
conc	Concentration $[\mathrm{kgm^{-3}}]$, piecewise constant on every element.
complete list	See Convection transport output fields.
Solute_AdvectionDiffusion_DG	
conc	Concentration $[\mathrm{kgm^{-3}}]$, piecewise linear on every element. Even if higher order polynomial approximation is used in simulation, the results are saved only in element corners.
complete list	See Transport with dispersion output fields.
DualPorosity	
conc_immobile	Concentration $[\mathrm{kgm^{-3}}]$ in immobile zone, piecewise linear on every element.
complete list	See Dual porosity output fields.
Sorption, SorptionMobile, SorptionImmobile	
conc_solid	Concentration $[\mathrm{mol\,kg^{-1}}]$ of sorbed substance, piecewise linear on every element.
complete list	See Sorption output fields, Mobile sorption output fields, Immobile sorption output fields.
Heat_AdvectionDiffusion_DG	
temperature	Temperature [K], piecewise linear on every element. Even if higher order polynomial approximation is used in simulation, the results are saved only in element corners.
complete list	See Heat transfer output fields.

- time

- region

- quantity [unit]: name and unit of the conservative quantity

- flux, flux_in, flux_out: flux through boundary regions (positive value stands

for flux into the domain)

- `mass`: current mass in bulk regions

- `source`, `source_in`, `source_out`: volume source in bulk regions, its positive and negative part

In addition, the following values for cumulative balance are shown when `region` is `ALL`:

- `flux_increment`, `source_increment`: flux and source increment since the last balance time

- `flux_cumulative`, `source_cumulative`: cumulative flux and source from the initial time

- `error`: current mass − (initial mass + cumulative sources + cumulative fluxes)

Raw Water Flow Data File

You can force Flow123d to write raw data about results of MH method. The file format is:

```
$FlowField
T=<time>
<number of elements>
<eid> <pressure> <flux x> <flux y> <flux z> <number of sides> <pressures on sides> <flu
...
$EndFlowField
```

where

`<time>` — is simulation time of the raw output.

`<number of elements>` — is number of elements in mesh, which is same as number of subsequent lines.

`<eid>` — element id same as in the input mesh.

`<flux x,y,z>` — components of water flux interpolated to barycenter of the element

`<number of sides>` — number of sides of the element, influence number of remaining values

`<pressures on sides>` — for every side average of the pressure over the side

`<fluxes on sides>` — for ever side total flux through the side

The side values are reported according to the side order, with sides numbering given by Table 4.2.

Table 4.2: Side numbering relative to vertices.

element dimension	side	vertices
1	0	0
	1	1
2	0	0 1
	1	0 2
	2	1 2
3	0	0 1 2
	1	0 1 3
	2	0 2 3
	3	1 2 3

Chapter 5

Tutorials

In this chapter we describe several tutorial files that demonstrate various features of Flow123d. The tutorial files are placed in `tests/05_tutorial`.

5.1   1D column

File: `01_column.yaml`

5.1.1   Description

The first example is inspired by a real locality of a water treatment plant tunnel Bedřichov in the granite rock massif. There is a particular seepage site 23 m under the surface which has a very fast reaction on rainfall events. Real data of discharge and concentration of stable isotopes are used.

The user will learn how to:

- Set up the mesh and flow model input parameters;
- Set up the solver and output parameters.

A pseudo one-dimensional model is considered in the range 10×23 m with the atmospheric pressure on the surface and on the bottom, and no flow boundary condition on the edges (Figure 5.1).

5.1.2   Input

The model settings are given in the control file, which is in YAML format. Every line contains one parameter and its value(s). The indentation of lines is important, since it indicates the section to which the parameter belongs.

Setting the computational mesh

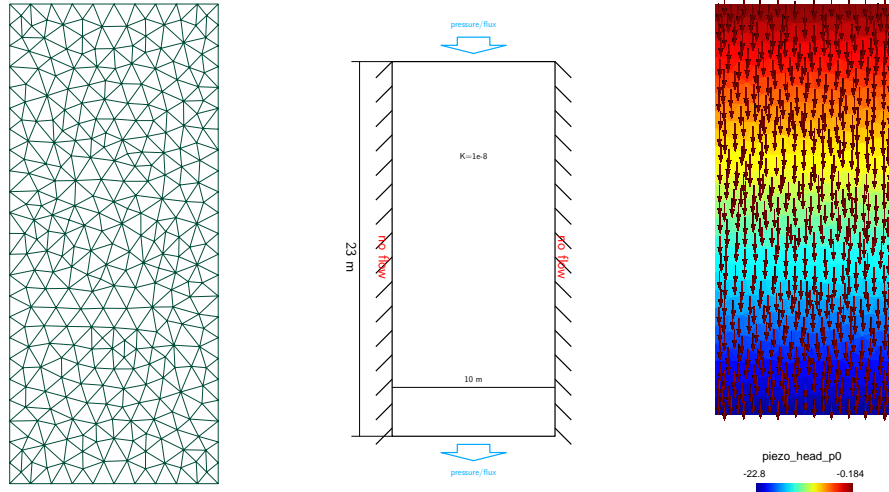The mesh file can be generated using the software GMSH. It has to contain:

Figure 5.1: a) the mesh; b) the boundary conditions; c) computed piezometric head and flux.

- Nodes. Point coordinates.
- Simplicial elements (lines, triangles, tetrahedra). Also elements of lower dimensions represent fractures or channels.
- Physical domains (groups of elements, labeled either by a numerical id or a string caption). Names of regions defining boundary have to start by a dot.

The mesh file is specified by the following lines:

```
mesh:
  mesh_file: 01_mesh.msh
```

Setting the model and physical parameters

In this example we use the Darcy flow model, which is set by:

```
flow_equation: !Flow_Darcy_MH
```

Note: The equation name consists of three parts: physical process (Flow), mathematical model (Darcy) and numerical method (MH = mixed hybrid finite element method).

The bulk parameters and boundary conditions are defined in the section `input_fields`. For the rock massif (`region: rock`) we prescribe the hydraulic conductivity $K = 10^{-8}$ m/s (typical value for the granite rock massif):

```
input_fields:
  - region: rock
    conductivity: 1e-8
```

We prescribe the atmospheric presure both at the surface and the tunnel:

```
  - region: .tunnel
    bc_type: dirichlet
    bc_pressure: 0
  - region: .surface
    bc_type: dirichlet
    bc_pressure: 0
```

If no boundary condition is given then the default "no flow" is applied.

Setting solver parameters

For the solution of the linear algebra problem we have to specify solver type and tolerances for controlling the residual. In `flow_equation` we can use either `Petsc` solver which performs well for small and moderate size problems, or `Bddc` (a scalable domain decomposition solver). Two stopping criteria can be given: absolute and relative tolerance of residual.

```
  nonlinear_solver:
    linear_solver: !Petsc
      a_tol: 1e-15
      r_tol: 1e-15
```

The key `nonlinear_solver` has further parameters which play role in other (nonlinear) flow models.

Setting output

In the section `output_stream` we define the file name and type (supported types are `gmsh` and `vtk`, which can be viewed by GMSH, ParaView, respectively) to which the solution is saved:

```
  output_stream:
    file: flow.msh
    format: !gmsh
```

The list of fields (solution components, input fields etc.) to be saved is specified by:

```
  output:
    fields:
      - piezo_head_p0
      - pressure_p0
      - velocity_p0
```

The above code can be alternatively written in a more compact form, namely

```
  output:
    fields: [piezo_head_p0, pressure_p0, velocity_p0]
```

In addition to the output of solution, Flow123d provides computation of balance of fluid volume, flux through boundaries and volume sources. This is turned on by

```
balance: {}
```

5.1.3 Results

The results of computation are generated to the files `flow.msh` and `water_balance.txt`. From the balance file, one can see that the input flux on the surface is 1×10^{-7} and the output flux on the tunnel is -1×10^{-7} (Table 5.1).

Table 5.1: Results in `water_balanced.txt` (edited table, extract from the file).

"time"	"region"	"quantity [m(3)]"	"flux"	"flux_in"	"flux_out"
0	"rock"	"water_volume"	0	0	0
0	".surface"	"water_volume"	1e-07	1e-07	0
0	".tunnel"	"water_volume"	-1e-07	0	-1e-07
0	"IMPLICIT BOUNDARY"	"water_volume"	2.58e-26	6.46e-26	-3.87e-26

5.1.4 Variant

Control file `02_column_transport.yaml` contains modified boundary conditions and solute transport model for the same physical problem.

5.2 1D column transport

File: `02_column_transport.yaml`

5.2.1 Description and input

This is a variant of `01_column.yaml`. The user will learn how to:

- Use flux boundary conditions.
- Set up the advective transport model.

For the fluid flow model we change the atmospheric pressure on the surface to the more realistic infiltration 200 mm/yr ($= 6.34 \times 10^{-9}$ m/s):

```
- region: .surface
  bc_type: total_flux
  bc_flux: 6.34E-09
```

In the resulting file `water_balance.txt` we can see that the value of the input and output flux changes to 6.34×10^{-8}. The visual results are similar to the case `01_column.yaml`.

Next we demonstrate a simulation of the transport of a tracer. The equation of advective transport (no diffusion/dispersion) is specified by:

```
solute_equation: !Coupling_OperatorSplitting
  transport: !Solute_Advection_FV
```

The boundary condition of concentration is prescribed on the surface region:

```
input_fields:
  - region: .surface
    bc_conc: 100
```

The default type of boundary condition is `inflow`, i.e. prescribed concentration is applied where water flows into the domain.

We provide the name of the transported substance (in general there can be multiple transported substances):

```
substances: O-18
```

The end time of the simulation is set in the section `time` to value 10^{10} seconds (381 years):

```
time:
  end_time: 1e10
```

The output files can be generated for specific time values. We set the time step for output to 10^8 seconds (=3 years and 2 months):

```
output_stream:
  times:
    - step: 1e8
```

Finally, we turn on computation of mass balance with cumulative sums over the simulation time interval.

```
balance:
  cumulative: true
```

5.2.2 Results

The results of the mass balance computation are in the output folder in the file `mass_balance.txt`. The evolution of concentration is depicted in Figure 5.2. A selected part of numerical results of mass balance is in the Table 5.2. On the region ".surface", the mass flux of the tracer is still identical (6×10^{-6} kg/s). On ".tunnel", the mass flux is zero at the beginning and then it changes within around 100 years to the opposite value of inflow -6 $\times$ 10^{-6} kg/s. Figure 5.3 depicts results from the file `mass_balance.txt` for mass transported through the boundaries ".surface" and ".tunnel" and in the volume of "rock".
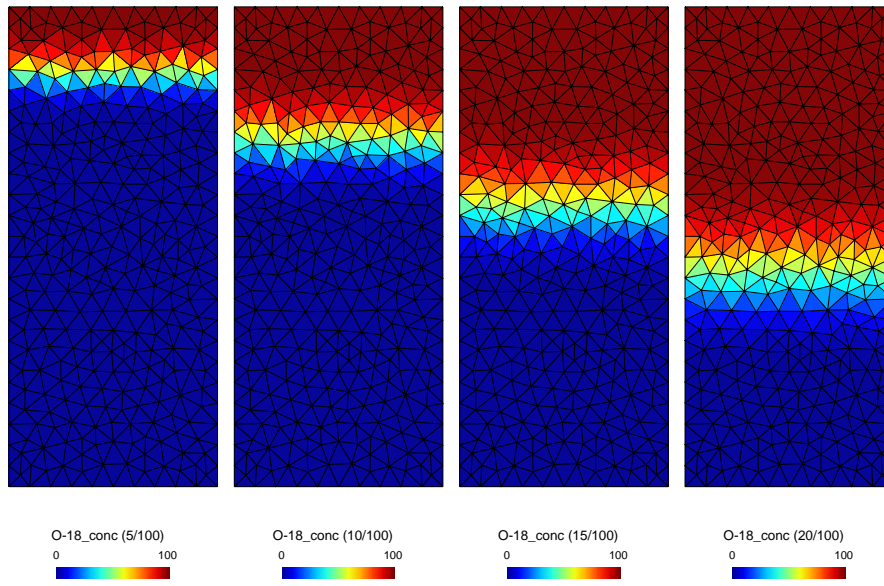
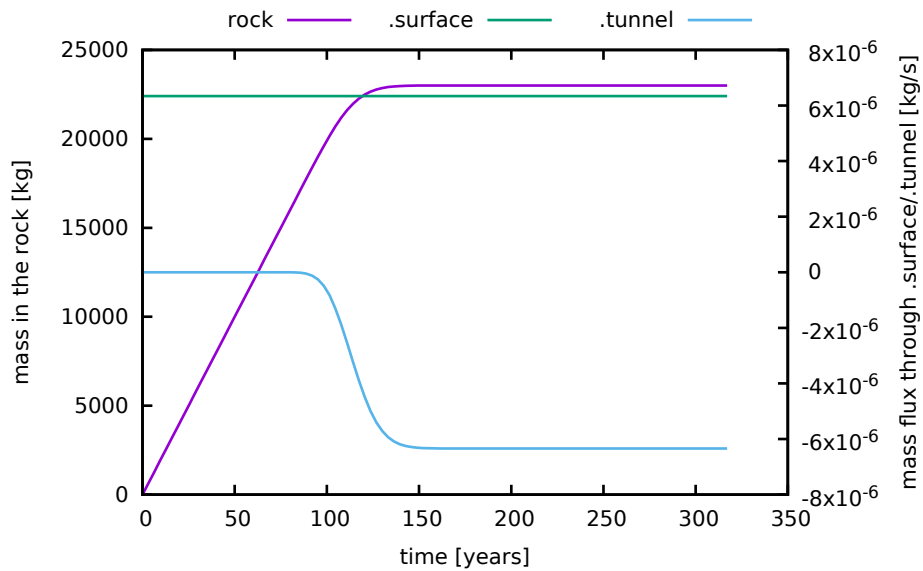Figure 5.2: Tracer concentration after 5, 10, 15 and 20 time steps.

Figure 5.3: Results of evolution of mass in the volume and flux through boundaries.

Table 5.2: Illustration of the results in `mass_balance.txt` – selected columns in two time steps.

time	region	quantity [kg]	flux	flux_in	flux_out	mass	error
3.9e+09	rock	O-18	0	0	0	22654.4	0
3.9e+09	.surface	O-18	6.34e-06	6.34e-06	0	0	0
3.9e+09	.tunnel	O-18	-4.99e-06	0	-4.99e-06	0	0
3.9e+09	IMPLICIT BOUNDARY	O-18	-1.02e-19	0	-1.02e-19	0	0
3.9e+09	ALL	O-18	1.34e-06	6.34e-06	-4.99e-06	22654.4	-5.78e-10
4e+09	rock	O-18	0	0	0	22774.9	0
4e+09	.surface	O-18	6.34e-06	6.34e-06	0	0	0
4e+09	.tunnel	O-18	-5.39e-06	0	-5.39e-06	0	0
4e+09	IMPLICIT BOUNDARY	O-18	-1.02e-19	0	-1.02e-19	0	0
4e+09	ALL	O-18	9.40e-07	6.34e-06	-5.39e-06	22774.9	-6.03e-10

5.3 2D tunnel

File: `03_tunnel.yaml`

5.3.1 Description

The tutorial models the seepage site 23 m under the surface of the water treatment plant tunnel Bedřichov in the granite rock massif. This seepage site has fast reaction to the precipitation and measurements of various chemical values are available.

The user will learn how to:

- Prescribe time-dependent input data.

The geometry consists of a rectangle 500×300 m with a circular hole of diameter 3.6 m placed 23 meters under the surface, which represents a plane perpendicular to the tunnel.

5.3.2 Hydraulic model

The hydraulic model was fitted on the shape of the flux field, where it was assumed that the tunnel drains only a part of the model surface. In particular, the model was fitted on the estimated discharge of the seepage site.

We impose the following input data (see Figure 5.4):

- The hydraulic conductivity of the rock medium is set to 2.59e-2 m/day (= 3e-7 m/s);
- On the surface we prescribe the annual precipitation 2.33e-3 m/day (= 852 mm/yr);
- On the bottom part ".base" we prescribe the pressure 270 m because of assumption of local groundwater flow;
- In the tunnel, the measured flux -9.16e-2 m/day (= -1.06e-6 m/s) is prescribed.

For convenience we use day as the unit of time. The corresponding YAML code is:
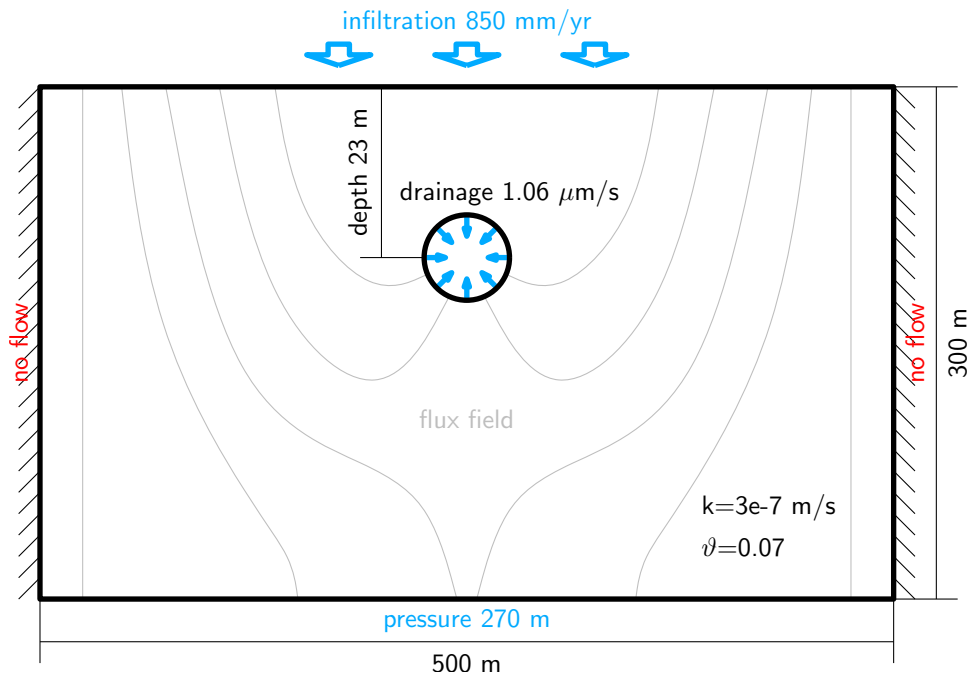
Figure 5.4: Geometry and boundary condition of model.

```
input_fields:
  - region: rock
    conductivity: 2.59E-02
  - region: .tunnel
    bc_type: total_flux
    bc_flux: -9.16E-02
  - region: .base
    bc_type: dirichlet
    bc_pressure: 270
  - region: .surface
    bc_type: total_flux
    bc_flux: 2.33E-03
```

The results are shown in Figure 5.5, where the flux field and the pressure is shown. In the unsaturated layer the piezometric head is depicted.

5.3.3 Transport of real isotopes

The stable isotope O-18 was sampled in monthly steps in precipitation at nearby experimental catchment Uhlirska and at the seepage site 23m depth. The measured values are used for the boundary condition on the surface in the transport model as well as reference values in the tunnel.

Input

We use the value 0.067 for porosity. The initial concentration of O-18 is set to -10.5 kg/m^3:
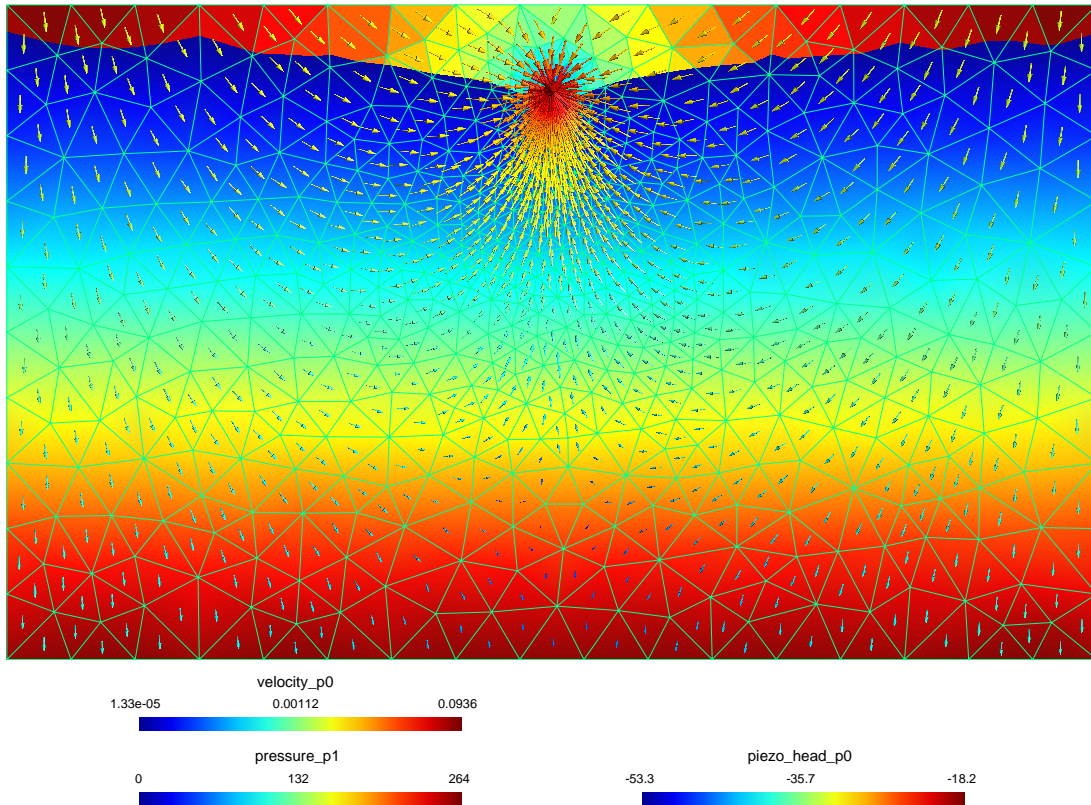
Figure 5.5: Pressure, boundary of water level and piezometric head in unsaturated zone and flux field.
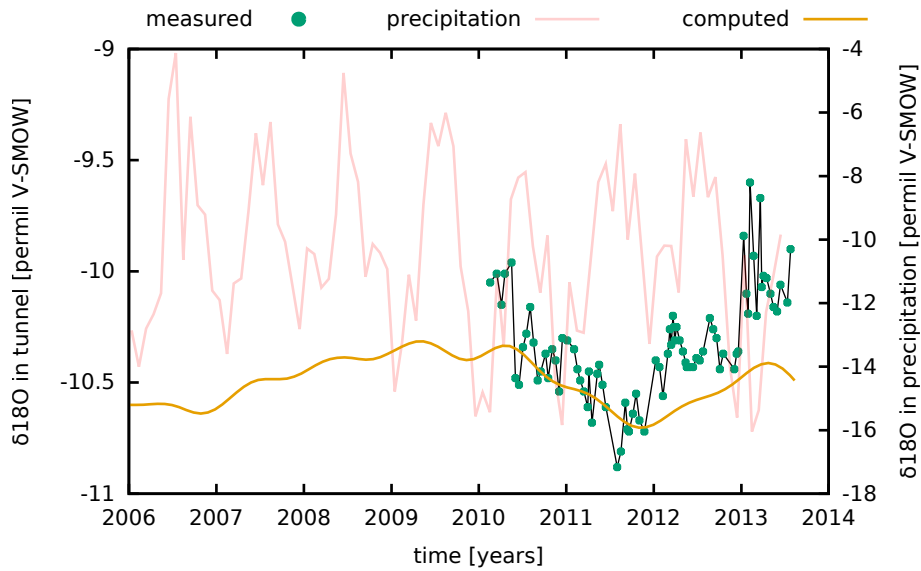
Figure 5.6: Concentration of O-18 on the seepage site 23m under the surface.

```
transport: !Solute_Advection_FV
  input_fields:
    - region: rock
      porosity: 0.067
      init_conc: -10.5
```

The monthly measured values of $\delta 18O$ [per mil V-SMOW] on the surface from the period 1/2006 till 6/2013 are supplied as the boundary condition:

```
    - region: .surface
      bc_conc: -12.85443
      time: 11
    - region: .surface
      bc_conc: -14.00255
      time: 42
    - region: .surface
      bc_conc: -12.80081
      time: 72
    - region: .surface
      bc_conc: -12.34748
      time: 103
  ...
```

Results

In Figure 5.6, the computed mass flux through tunnel is compared to the measured data. The evolution of the transported substance is depicted in Figure 5.7.
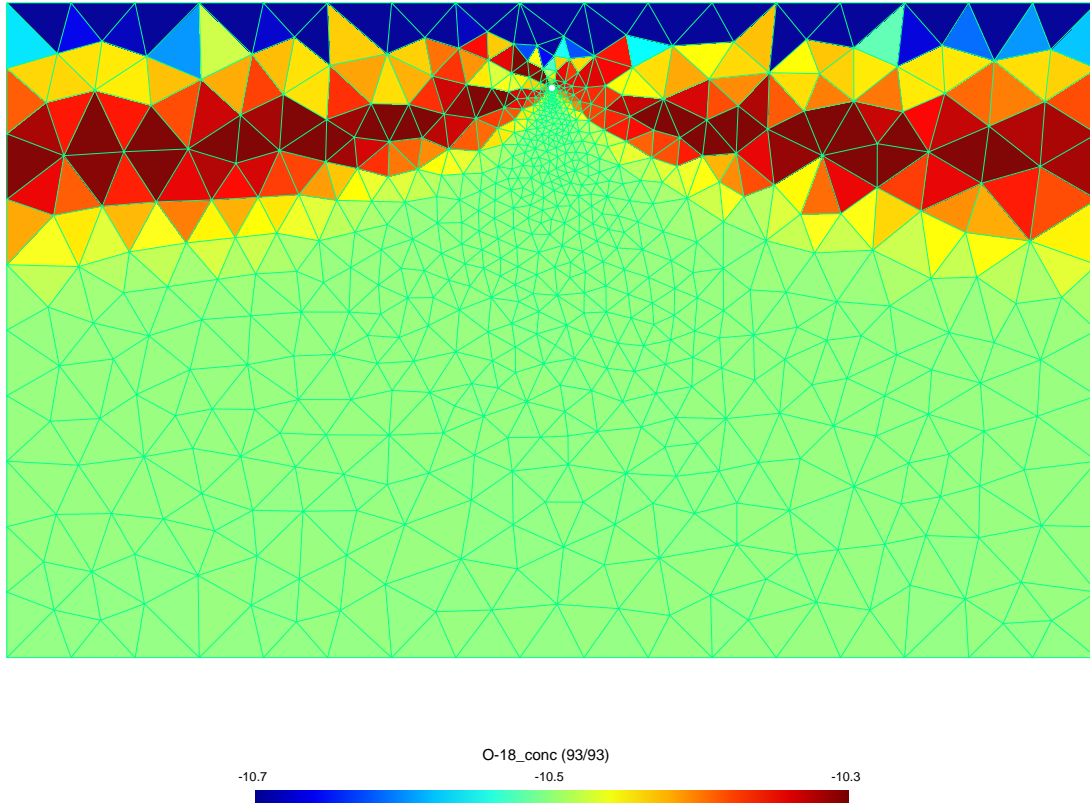
Figure 5.7: Transport of isotopes in two-dimensional model.

5.4 Fractures and diffusion

File: `04_frac_diffusion.yaml`

5.4.1 Description

In Flow123D domain interaction with fractures can be implemented. This example comes from a study of evaluation of influence of an individual processes (diffusion, linear sorption, dual-porosity) between domain interaction in transport. The background of this study is movement of contaminant mass from deep repository along fractures. The output mass from fracture and rock is evaluated for every individual process.

The user will learn how to:

- prepare mesh of fractured zone
- define union of mesh regions
- use advection-diffusion transport model
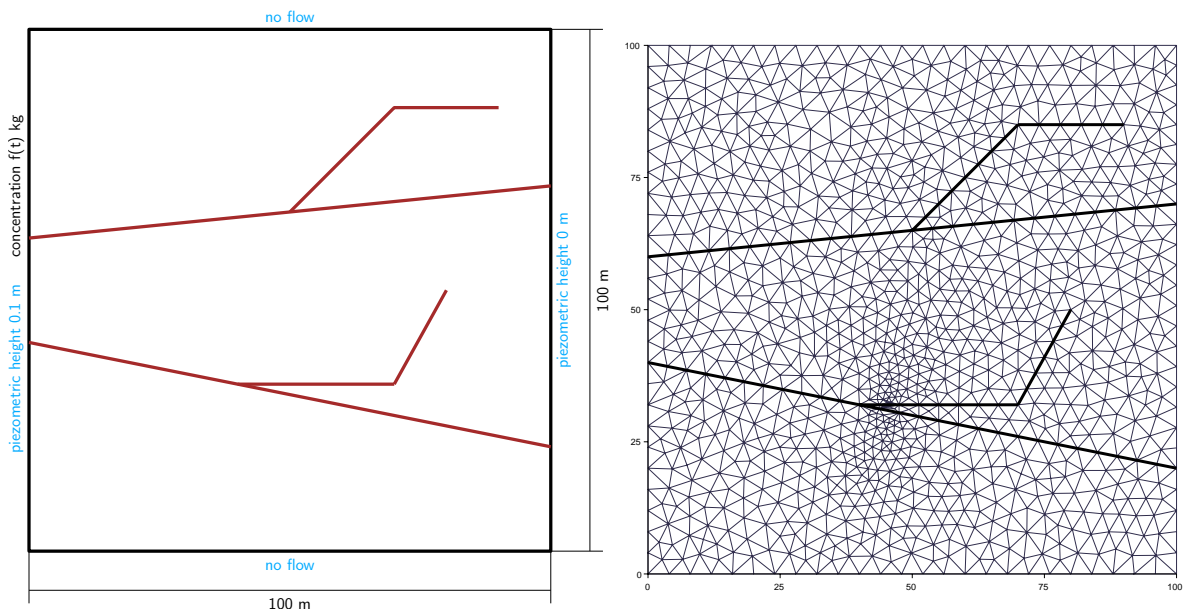- define variable time step

Figure 5.8: Geometry and mesh of simulation area.

5.4.2 Input

Geometry and mesh generation

The simulation area 100×100 m is cut by two flow fractures from which two dead-end fractures separate (see Figure 5.8). The cross-section of the fractures is 0.01 m.

Instead of defining a geometry with thin 2D fractures (which would yield too large mesh), in Flow123d one can treat fractures as lines with intrinsic cross-section area (or surfaces with intrinsic width). In order to produce a compatible mesh, where fracture elements are faces of triangles, we use additional GMSH command in the file `04_mesh.geo`:

```
Line { 9:16 } In Surface { 20 };
```

This ensures that the 2D mesh will adapt so that elements do not cross the fracture lines (see Figure 5.8).

In the YAML file one can define regions in addition to those from MSH file. We use the type !Union type in the array `regions` to define sets of regions sharing some properties (e.g. boundary conditions):

```
mesh:
  mesh_file: 04_mesh.msh
  regions:
    - !Union
      name: flow_fractures
      regions:
        - flow_fracture1
        - flow_fracture2
    - !Union
      name: deadend_fractures
```

```
      regions:
        - deadend_fracture1
        - deadend_fracture2
    - !Union
      name: BC_right
      regions:
        - .right
        - .right_points
    - !Union
      name: BC_left
      regions:
        - .left
        - .left_points
```

Hydraulic model

We are interested in simulation for 50000 years, hence we use year as the time units in the definition of model parameters. Hydraulic conductivity $k = 10^{-11}$ m/s (0.000315 m/yr) was considered for rock massif. For the flow fractures and for the dead-end fractures we considered $k = 10^{-6}$ (31.5 m/yr) and $k = 10^{-7}$ (3.15 m/yr), respectively. These values are in accordance with typical values of conductivity of a massif considered for deep repository. The thickness of model was set to 0.01 m for fractures:

```
input_fields:
- region: rock
  conductivity: 0.000315
- region: flow_fractures
  conductivity: 31.5
  cross_section: 0.01
- region: deadend_fractures
  conductivity: 3.15
  # variant without dead-end fractures conductivity: 0.000315
  cross_section: 0.01
```

To eliminate the dead-end fractures from the model, one can set their conductivity identical to the rock. Other possibility is to use the same conductivity as in the flow-fractures.

Two dirichlet boundary conditions were defined for the flux: piezometric head 0.1 m on the left side and 0 m on the right side:

```
- region: BC_left
  bc_type: dirichlet
  bc_piezo_head: 0.1
- region: BC_right
  bc_type: dirichlet
  bc_piezo_head: 0
```

The above values were chosen in order to obtain filtration flux in the flux-fractures approximately 1×10^{-9} m/s (≈ 0.1 m/yr). Other sides are nonpermeable.

Transport model

We use the advection-diffusion equation:

```
solute_equation: !Coupling_OperatorSplitting
  transport: !Solute_AdvectionDiffusion_DG
```

The porosity was set to 0.005 for rock and 0.1 for fractures. The parameters of mechanical dispersion are set to 5 m for longitudinal dispersivity and 0.5 m for transversal dispersivity. For the molecular diffusivity we use the same value at rock and fractures: $D_m = 3.69 \times 10^{-2}$ m^2/yr. Since in Flow123d the molecular diffusion tensor has the form $D_m \vartheta^{1/3} \mathbb{I}$, the effective molecular diffusivity will be 2.7 times higher on the fractures than in the rock (Table 5.3):

```
input_fields:
 - region: rock
   init_conc: 0
   porosity: 0.005
   diff_m: 0.0369
   disp_l: 5
   disp_t: 0.5
 - region: flow_fractures
   init_conc: 0
   porosity: 0.1
   diff_m: 0.0369
   disp_l: 5
   disp_t: 0.5
 - region: deadend_fractures
   init_conc: 0
   porosity: 0.1
   diff_m: 0.0369
   disp_l: 5
   disp_t: 0.5
```

Table 5.3: Coefficient of molecular diffusion prescribed in Flow123d.

Quantity	Rock	Fracture
Porosity ϑ [$-$]	0.005	0.1
Coefficient of molecular diffusion D_m [m^2/s]	1e-9	1e-9
Effective molecular diffusion $D_m \vartheta^{1/3}$ [m^2/s]	1.71e-10	4.64e-10

The boundary condition for the concentration at the fracture was prescribed in the form of Gaussian curve

$$f(t) = \frac{1}{20} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{t-t_0}{\sigma}\right)^2},$$

with the meanvalue $t_0 = 2000$ years and standard deviation $\sigma = 700$ years:
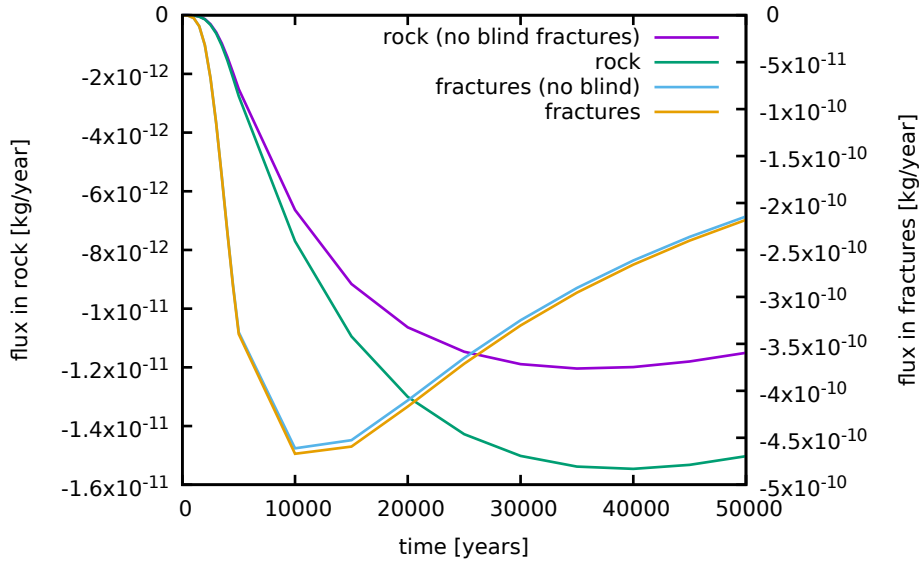
Figure 5.9: Outgoing mass flux through the right part of the boundary. Comparison of results with and without dead-end fractures.

```
- region: .left_0
  bc_type: dirichlet
  bc_conc: !FieldFormula
    value: 2.84959e-5*exp(-0.5*((t-2000)/700)^2)
```

It means that during the simulation time $T = 50000$ years, almost 0.05 kg/m^3 ($= \int_0^T f(t)\,dt$) of water is released. Maximum concentration of realised water is 0.028 g/m^3 ($= f(t_0)$). The mean value corresponds with real values of release of isotopes of deep repository.

For better resolution of the time-dependent boundary condition, we refine the initial output time step and after 5000 years we increase it:

```
output_stream:
  times:
    - step: 500
      end: 5000
    - begin: 5000
      step: 5000
```

Here `times` is an array of time grids, each having optional parameters `begin`, `end` and `step`. The computational time step will adapt to this grid automatically.

5.4.3   Results

The result of model with and without dead-end fractures (file `04_frac_diffusion_nodeadend.yaml`) is depicted in Figure 5.9. We can see that with dead-end fractures, the water is more contaminated at the outflow from the rock. The influence on flow fractures is negligible.

5.5 Fractures and sorption

File: `05_frac_sorption.yaml`

5.5.1 Description

This is a variant of `04_frac_diffusion.yaml`. Instead of diffusion we consider advective transport with equilibrial sorption.

5.5.2 Input

Flow123d provides several types of sorption (linear, Langmuir and Freundlich isotherm). Each substance can be assigned its own sorption type. In this test, the transport of three substances is computed: Iodium without sorption, Radium with liner sorption and Selenium with Langmuir isotherm. The solvent density and solubility was set to 1. Initial condition of solid was set to zero; rock density to 1 and parameter of linear and Langmuir isotherm was set to 1.0.

```
reaction_term: !Sorption
  substances:
    - I
    - Ra-lin
    - Se-lang
  solvent_density: 1.0
  solubility: [ 1.0, 1.0, 1.0 ]
  input_fields:
    - region: ALL
      init_conc_solid: 0
      rock_density: 1.0
      sorption_type:
        - none
        - linear
        - langmuir
      distribution_coefficient: 1.0
      isotherm_other: 0.4
```

In fact, the fields `init_conc_solid`, `isotherm_mult`, `isotherm_other` can have different values for each substance. In that case we define them as YAML arrays.

5.5.3 Results

Figure 5.10 depicts the influence of linear and Langmuir isotherm on the transport of substances. The substance I without sorption flows out of the fracture fastest and the substance Ra flows out slowest.
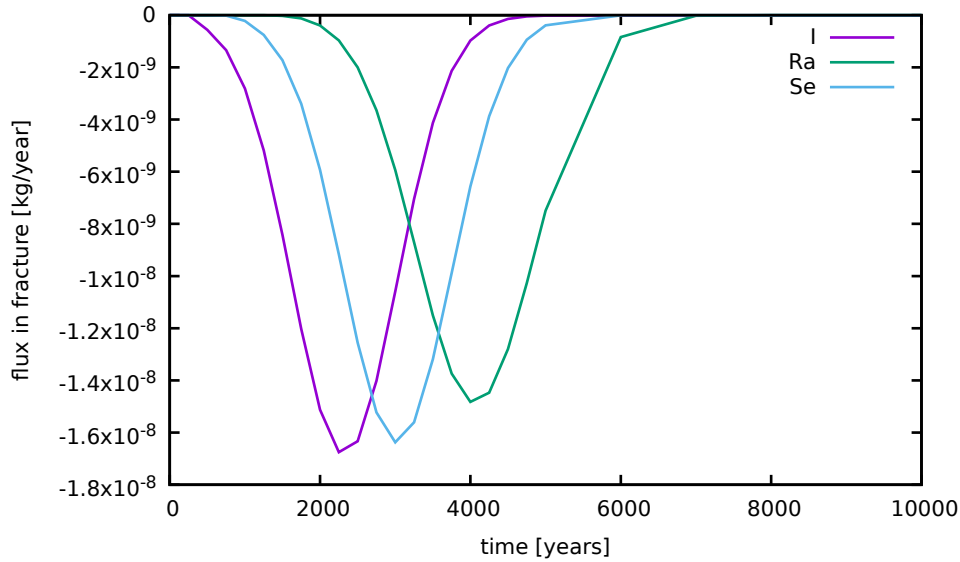
Figure 5.10: Results of sorption.

5.6 Fractures and dual porosity

File: `06_frac_dualpor.yaml`

5.6.1 Description

This is a variant of `04_frac_diffusion.yaml`. Instead of diffusion we consider advective transport with dual porosity.

5.6.2 Input

Dual porosity substitutes dead-end fractures in this task. The dual-porosity parameter `diffusion_rate_immobile` was calibrated to the value 5.64742e-06 for identical results with the model with the dead-end fractures. Other settings of transport are identical to the diffusion model.

The dual porosity model is set by the following lines:

```
reaction_term: !DualPorosity
  input_fields:
    - region: rock
      init_conc_immobile: 0
    - region: flow_fractures
      diffusion_rate_immobile: 5.64742e-06
      porosity_immobile: 0.01
      init_conc_immobile: 0
    - region: deadend_fractures
      init_conc_immobile: 0
```
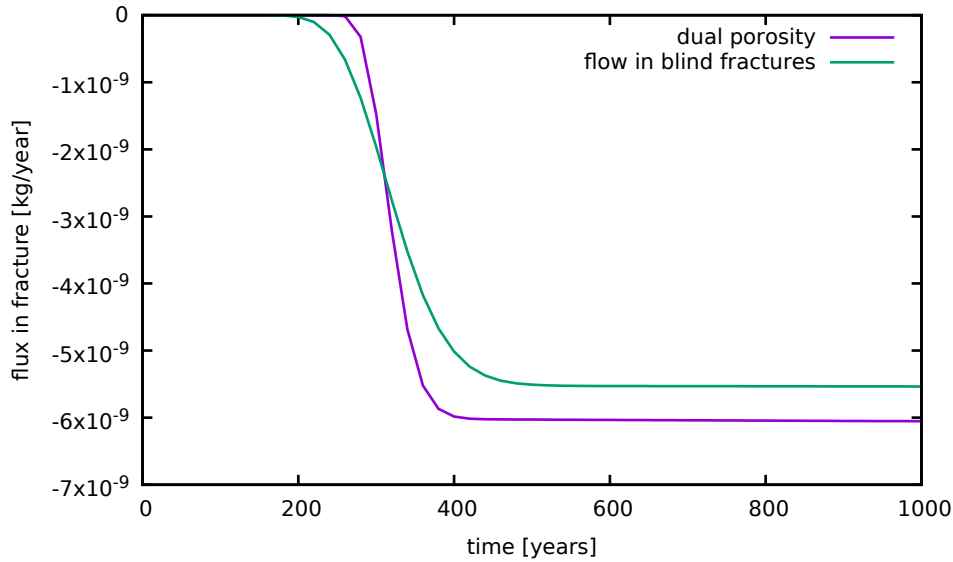
Figure 5.11: Results of calibration.

5.6.3 Results and comparison

Results of calibration of the model with dual porosity and model with flow in dead-end fractures (file `06_frac_nodualpor.yaml`) is depicted in Figure 5.11.

5.7 Heat transport

File: `07_heat.yaml`

5.7.1 Description

The task is inspired by the hot-dry-rock method of geothermal heat exchanger. The exchanger should be in progress for 30 years and give the power of 25 MW.

The user will learn how to:

- Set up heat transfer model;
- Use transition parameters at interfaces;
- Specify linear algebra solver.

5.7.2 Input

Geometry

We consider a two-dimensional model 5000×5000 m with two vertical wells at the distance of 3000 m. The wells are 4300 m deep with the diameter approx. 11 cm (Figure 5.12). In order to better capture the 3D nature of the problem, we set `cross_section` (width) of the rock region to 100 m (the value was gained from calibration), and the cross section of the wells to 0.04 m^2.

Table 5.4: Geometrical parameters.

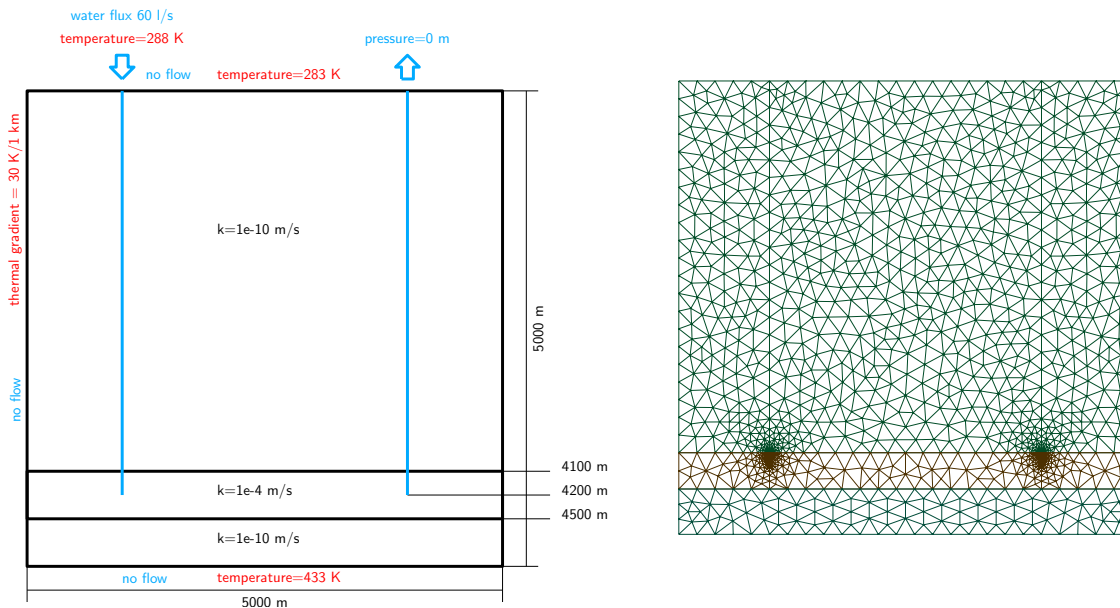| Parameter	Value
Model width	5000 m
Model depth	5000 m
Depth of heat exchanger	4100 − 4300 m

Figure 5.12: Geometry, boundary conditions and computational mesh.

```
    cross_section: 100
    conductivity: 1.0e-10
  - region: exchanger
    conductivity: 1e-4
```

The flow in the wells is modelled using the Darcy equation with a high hydraulic conductivity (10 m/s). The transition coefficient `sigma` [–], determines the rate of exchange between 2D rock and 1D wells. Its default value 1 is kept at the lower well ends, elsewhere the wells are isolated and hence we set `sigma` to zero.

```
  - region: wells
    conductivity: 10.0
    cross_section: 0.04
    sigma: 0
  - region: wells_deep
    sigma: 1
```

On the injection well (".well1_surface"), we prescribe the flux 60 l/s, i.e. the flux velocity is 1.5 m/s. On the production well (".well2_surface") we prescribe zero pressure.

```
  - region: .well1_surface
    bc_type: total_flux
    bc_flux: 1.5
  - region: .well2_surface
    bc_type: dirichlet
    bc_pressure: 0
```

We assume that the system does not have contact with its surrounding because of high depth and intact granite massive. Hence no flow boundary conditions are given on the sides, on the bottom and on the surface.

92

For the solution of the flow problem we choose the LU decomposition as the linear algebra solver:

```
nonlinear_solver:
  linear_solver: !Petsc
    options: -ksp_type preonly -pc_type lu
```

Heat transport model

The heat transport model (`Heat_AdvectionDiffusion_DG`) assumes that the fluid and solid phase are at thermal equilibrium. For the whole model (`- region: ALL`) we prescribe the parameters for water and granite (density, thermal conductivity and capacity):

```
heat_equation: !Heat_AdvectionDiffusion_DG
  balance:
    cumulative: true
  input_fields:
    - region: ALL
      fluid_density: 1000.0
      fluid_heat_capacity: 4000
      fluid_heat_conductivity: 0.5
      solid_density: 2700.0
      solid_heat_capacity: 790
      solid_heat_conductivity: 2.5
```

The temperature on the surface is set to 283 K (= 10°C):

```
- region: .surface
  bc_type: dirichlet
  bc_temperature: !FieldFormula
    value: 10+273.15
```

The injected water has temperature 15°C:

```
- region: .well1_surface
  bc_type: dirichlet
  bc_temperature: !FieldFormula
    value: 15+273.15
```

The temperature on the bottom and sides as well as the initial temperature in the rock and the wells is then prescribed in agreement with typical geological gradient, approx. 1°C / 33 m:

```
init_temperature: !FieldFormula
  value: 10-z/5000*150+273.15
```
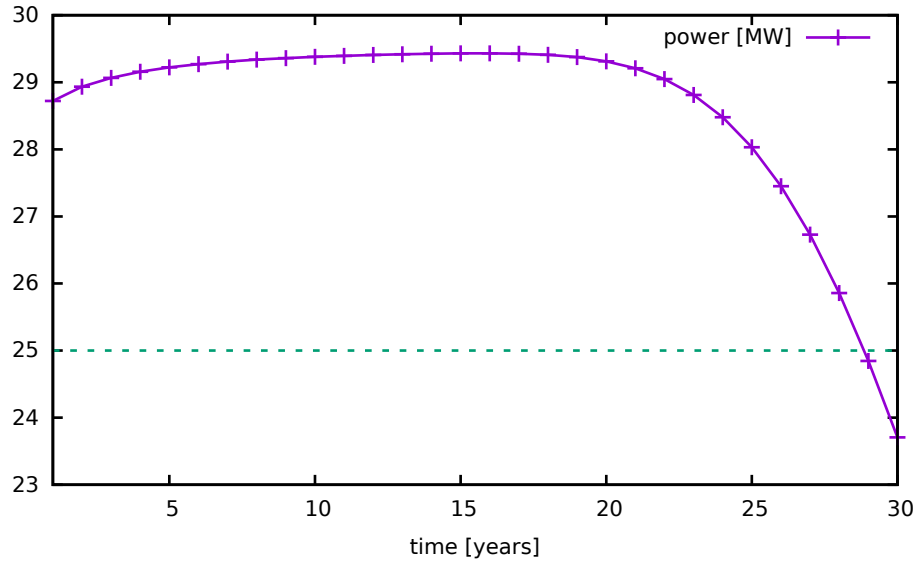
Figure 5.13: The power of heat exchanger system in 30 years.

The porosity was set to 1×10^{-5} for rock and 1×10^{-4} for exchanger. The transition coefficient of wells ("fracture_sigma") was set to 0 in rock surrounding and to 1 in deep surrounding:

```
- region: wells
  init_temperature: !FieldFormula
    value: 15-z/5000*150+273.15
  porosity: 1.0e-05
  fracture_sigma: 0
- region: wells_deep
  fracture_sigma: 1
```

5.7.3   Results

The evolution of power of the heat exchanger (difference of absolute energy flux on the surface of the two wells) is depicted in Figure 5.13. The result of water flow is depicted in Figure 5.14 and the temperature field of the whole massif after 30 years is depicted in Figure 5.15.
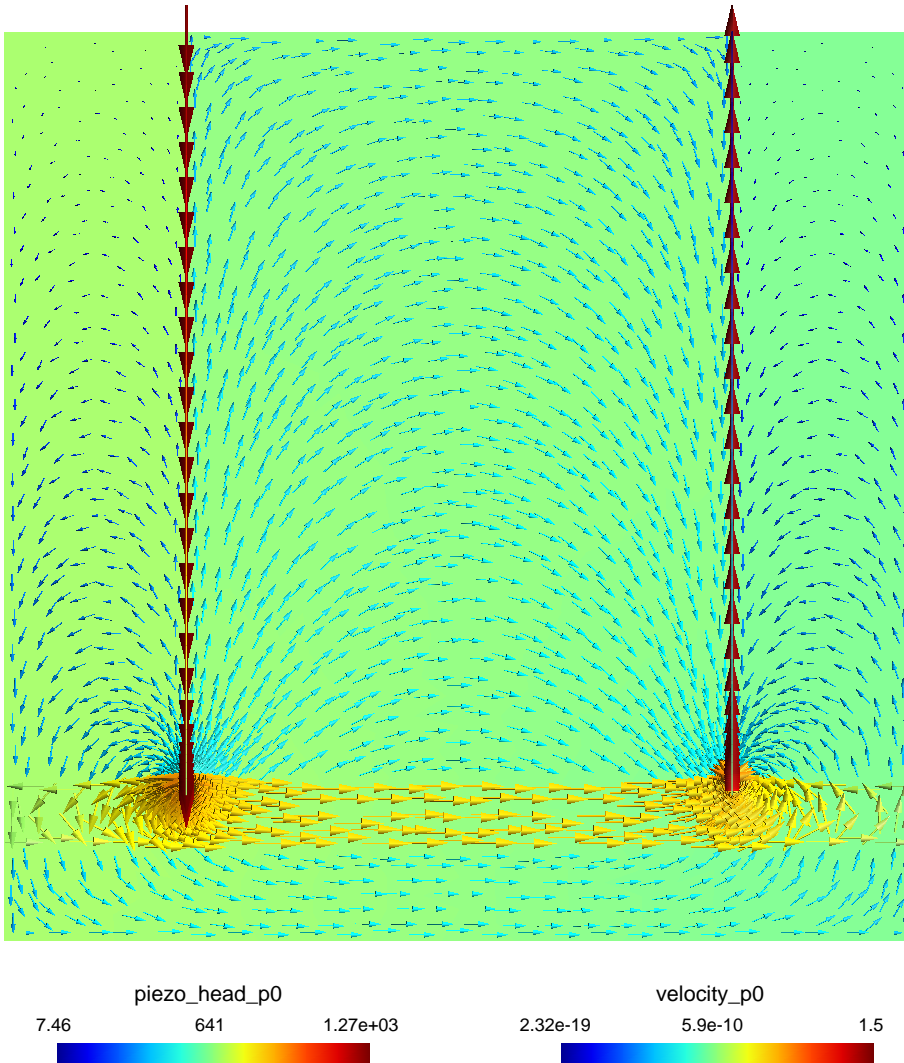
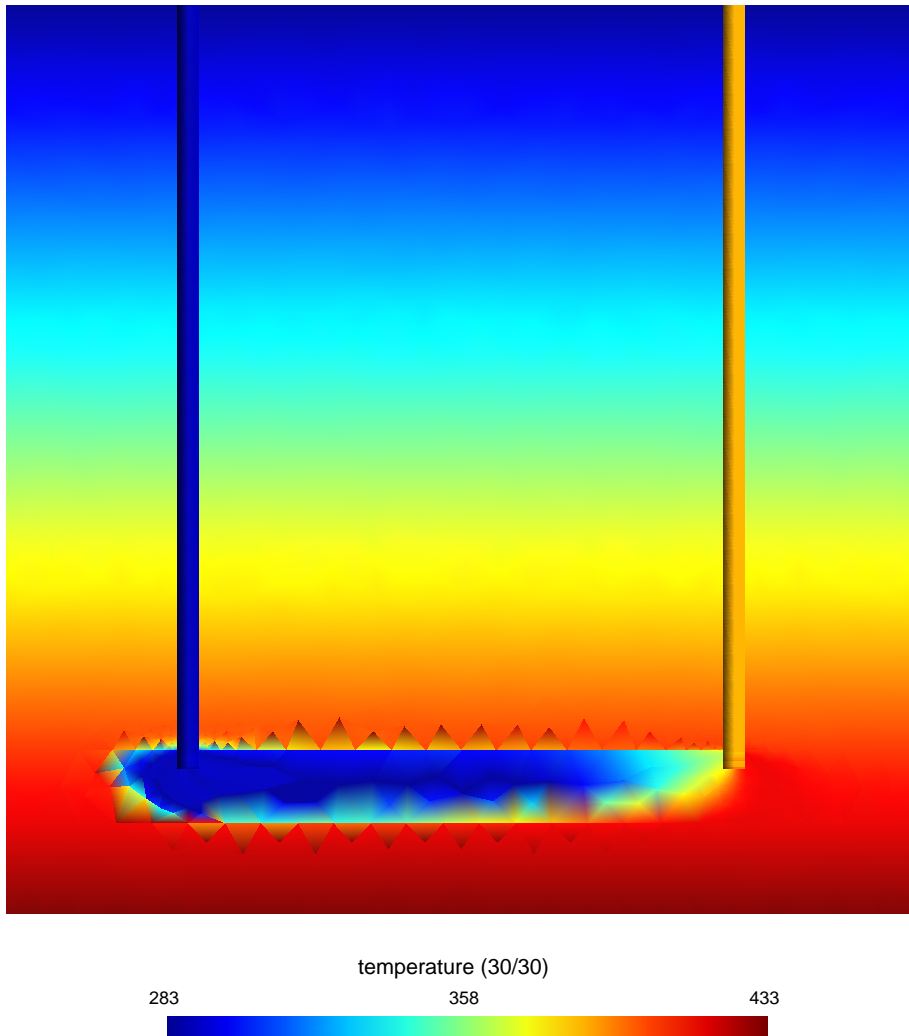Figure 5.14: The flux field with piezometric head.

Figure 5.15: The temperature of exchanger after 30 years.

Chapter 6

Main Input File Reference

This chapter contains generated reference to the main input file. Described types are ordered according to the deep first search of the input structure tree which somehow keep description of related types close to each other. Interactive links allows passing through the tree structure in top-bottom manner.

Ranges of arrays, integers and doubles use following notation: `INT` for maximum of a signed 32-bit integer ($\approx 2.147 \times 10^9$), `UINT` for maximum of unsigned 32-bit integer ($\approx 4.295 \times 10^9$), and `inf` for maximum of the double precision floating point number ($\approx 1.798 \times 10^{308}$).

---

record: **Root**

Root record of JSON input for Flow123d.

flow123d_version $= \langle$*String* $\rangle$

> default: *Obligatory*

> Version of Flow123d for which the input file was created. Flow123d only warn about version incompatibility. However, external tools may use this information to provide conversion of the input file to the structure required by another version of Flow123d.

problem $= \langle$*abstract:* Coupling_Base $\rangle$

> default: *Obligatory*

> Simulation problem to be solved.

pause_after_run $= \langle$*Bool* $\rangle$

> default: **false**

> If true, the program will wait for key press before it terminates.

---

abstract: **Coupling_Base**

The root record of description of particular the problem to solve.

implementations:

Coupling_Sequential

---

record: **Coupling_Sequential**

Record with data for a general sequential coupling.

implements abstracts: Coupling_Base

time $= \langle$*record:* TimeGovernor $\rangle$

> default: {}

> Time governor setting.

description = ⟨*String* ⟩

> default: *Optional*

> Short description of the solved problem.
> Is displayed in the main log, and possibly in other text output files.

mesh = ⟨*record:* Mesh ⟩

> default: *Obligatory*

> Computational mesh common to all equations.

flow_equation = ⟨*abstract:* DarcyFlow ⟩

> default: *Obligatory*

> Flow equation, provides the velocity field as a result.

solute_equation = ⟨*abstract:* AdvectionProcess ⟩

> default: *Optional*

> Transport of soluted substances, depends on the velocity field from a Flow equation.

heat_equation = ⟨*abstract:* AdvectionProcess ⟩

> default: *Optional*

> Heat transfer, depends on the velocity field from a Flow equation.

---

record: **TimeGovernor**

Time axis settings of the simulation.
The settings is specific to a particular equation.
TimeGovernor allows to:
- define start time and end time of simulation
- define lower and upper limits of time steps
- direct fixed time marks of whole simulation
- set global time unit of equation (see 'common_time_unit' key)
Limits of time steps are defined by keys 'min_dt', 'max_dt', 'init_dt' and 'dt_limits'.
Key 'init_dt' has the highest priority and allows set fix size of time steps. Pair of keys
'min_dt' and 'max_dt' define interval of time steps. Both previous cases ('init_dt' or pair
'min_dt' and 'max_dt') set global limits of whole simulation. In contrasts, 'dt_limits'
allow set time-dependent function of min_dt/max_dt. Used time steps of simulation
can be printed to YAML output file (see 'write_used_timesteps'.
Fixed time marks define exact values of time steps. They are defined in:
- start time and end time of simulation
- output times printed to output mesh file
- times defined in 'dt_limits' table (optional, see 'add_dt_limits_time_marks' key)

conversion from key: max_dt

`start_time` = ⟨*tuple:* `TimeValue` ⟩

> default: `0.0`

> Start time of the simulation.

`end_time` = ⟨*tuple:* `TimeValue` ⟩

> default: `5e+17`

> End time of the simulation.
> The default value is higher than the age of the Universe (given in seconds).

`init_dt` = ⟨*tuple:* `TimeValue` ⟩

> default: `0.0`

> Initial guess for the time step.
> It applies to equations that use an adaptive time stepping. If set to 0.0, the time step is determined in fully autonomous way, assuming the equation supports it.

`min_dt` = ⟨*tuple:* `TimeValue` ⟩

> default: implicit value: "Machine precision."

> Soft lower limit for the time step.
> Equation using an adaptive time stepping cannot suggest smaller time step. The actual time step can only decrease below the limit in order to match the prescribed input or output times.

`max_dt` = ⟨*tuple:* `TimeValue` ⟩

> default: implicit value: "Whole time of the simulation if specified, infinity else."

> Hard upper limit for the time step.
> The actual time step can only increase above the limit in order to match the prescribed input or output times.

`dt_limits` = ⟨*array [0, UINT] of tuple:* `DtLimits` ⟩

> default: *Optional*

> Allow to set a time dependent changes in `min_dt` and `max_dt` limits. This list is processed at individual times overwriting previous values of `min_dt`/`max_dt`. Limits equal to 0 are ignored and replaced with `min_dt`/`max_dt` values.

`add_dt_limits_time_marks` = ⟨*Bool* ⟩

> default: `false`

> Add all times defined in `dt_limits` table to the list of fixed TimeMarks.

`write_used_timesteps` = ⟨*Filename* ⟩

> default: *Optional*

> Write used time steps to the given file in YAML format corresponding with the format of `dt_limits`.

`common_time_unit = ⟨`*record:* `Unit ⟩`

> default: `"s"`

> Common time unit of the equation.
> This unit will be used for all time inputs and outputs within the equation. Individually, the common time unit can be overwritten for every declared time.
> Time units are used in the following cases:
> 1) Time units of time value keys in: TimeGovernor, FieldDescriptors.
> The common time unit can be overwritten for every declared time.
> 2) Time units in:
> a) input fields: FieldFE and FieldTimeFunction
> b) time steps definition of OutputTimeSet
> Common time unit can be overwritten by one unit value for every whole mesh data file or time function.
> 3) Time units in output files: observation times, balance times, frame times of VTK and GMSH
> Common time unit cannot be overwritten in these cases.

---

tuple: **TimeValue**

A time with optional unit specification.

conversion from key: `time`

`time = ⟨`*Double (-inf, +inf) ⟩*

> default: *Obligatory*

> The time value.

`unit = ⟨`*record:* `Unit ⟩`

> default: implicit value: "Common time unit of the equation's Time Governor. See the key 'common_time_unit'."

> Predefined units include: `s` seconds, `min` minutes, `h` hours, `d` days, `y` years.
> The default time unit is set from the equation's time governor, see the key `common_time_unit` in the equation's time record. User can benefit from the Unit Convertor funcionality and create different time units.
> Year length example considering leap years (Gregorian calendar): `year; year = 365.2425*d`.
> Miliseconds example : `milisec; milisec = 0.001*s`.

---

record: **Unit**

Specify the unit of an input value. Evaluation of the unit formula results into a coeficient and a unit in terms of powers of base SI units. The unit must match the expected SI

unit of the value, while the value provided on the input is multiplied by the coefficient before further processing. The unit formula have a form:

<UnitExpr>;<Variable>=<Number>*<UnitExpr>;...,

where <Variable> is a variable name and <UnitExpr> is a units expression which consists of products and divisions of terms. A term has a form: <Base>^<N>, where <N> is an integer exponent and <Base> is either a base SI unit, a derived unit, or a variable defined in the same unit formula. Example, unit for the pressure head: MPa/rho/g_; rho = 990*kg*m^-3; g_ = 9.8*m*s^-2

conversion from key: unit_formula

unit_formula = ⟨*String* ⟩

> default: *Obligatory*

> Definition of unit.

---

tuple: **TimeValue**

A time with optional unit specification.

conversion from key: time

time = ⟨*Double [0, +inf)* ⟩

> default: *Obligatory*

> The time value.

unit = ⟨*record:* Unit ⟩

> default: implicit value: "Common time unit of the equation's Time Governor. See the key 'common_time_unit'."

> Predefined units include: s seconds, min minutes, h hours, d days, y years.
> The default time unit is set from the equation's time governor, see the key common_time_unit in the equation's time record. User can benefit from the Unit Convertor funcionality and create different time units.
> Year length example considering leap years (Gregorian calendar): year; year = 365.2425*d.
> Miliseconds example : milisec; milisec = 0.001*s.

---

tuple: **DtLimits**

Time dependent changes in min_dt and max_dt limits.

conversion from key: time

time = ⟨*tuple:* `TimeValue` ⟩

>   default: *Obligatory*

>   The start time of dt step set.

min_dt = ⟨*tuple:* `TimeValue` ⟩

>   default: implicit value: "'min_dt' value of TimeGovernor."

>   Soft lower limit for the time step.

max_dt = ⟨*tuple:* `TimeValue` ⟩

>   default: implicit value: "'max_dt' value of TimeGovernor."

>   Whole time of the simulation if specified, infinity else.

---

record: **Mesh**

Record with mesh related data.

conversion from key: `mesh_file`

`mesh_file` = ⟨*Filename* ⟩

>   default: *Obligatory*

>   Input file with mesh description.

`regions` = ⟨*array [0, UINT] of abstract:* `Region` ⟩

>   default: *Optional*

>   List of additional region and region set definitions not contained in the mesh. There are three region sets implicitly defined:
>
>   – ALL (all regions of the mesh)
>   – .BOUNDARY (all boundary regions)
>   – BULK (all bulk regions)

`partitioning` = ⟨*record:* `Partition` ⟩

>   default: `"any_neighboring"`

>   Parameters of mesh partitioning algorithms.

`print_regions` = ⟨*Bool* ⟩

>   default: `true`

>   If true, print table of all used regions.

intersection_search $= \langle$ *selection:* Types of search algorithm for finding intersection candidates. $\rangle$

default: "BIHsearch"

Search algorithm for element intersections.

global_snap_radius $= \langle$ *Double [0, +inf)* $\rangle$

default: 0.001

Maximal snapping distance from the mesh in various search operations. In particular, it is used to find the closest mesh element of an observe point; and in FieldFormula to find closest surface element in plan view (Z projection).

raw_ngh_output $= \langle$ *Filename* $\rangle$

default: *Optional*

Output file with neighboring data from mesh.

optimize_mesh $= \langle$ *Bool* $\rangle$

default: true

If true, permute nodes and elements in order to increase cache locality. This will speed up the calculations. GMSH output preserves original ordering but is slower. All variants of VTK output use the permuted.

---

abstract: **Region**

Abstract record for Region.

implementations:

From_Id, From_Label, From_Elements, Union, Difference, Intersection

---

record: **From_Id**

Elementary region declared by its id.
It allows to create a new region with given id and name, or to rename an existing region of given id.

implements abstracts: Region

name $= \langle$ *String* $\rangle$

default: *Obligatory*

Name (label) of the region. It has to be unique per single mesh.

`id = ⟨`*Integer [0, INT]* `⟩`

> default: *Obligatory*

> Id of the region to which you assign the name.

`dim = ⟨`*Integer [0, INT]* `⟩`

> default: *Optional*

> Dimension of the region to which you assign the name.
> The value is taken into account only if a new region is created.

---

record: **From_Label**

Elementary region declared by its name (label).
It gives a new name to an elementary region with the original name (in the mesh file)
given by the `mesh_label.`

implements abstracts: Region

`name = ⟨`*String* `⟩`

> default: *Obligatory*

> New name (label) of the region. It has to be unique per single mesh.

`mesh_label = ⟨`*String* `⟩`

> default: *Obligatory*

> The original region name in the input file, e.g. a physical volume name in the
> GMSH format.

`allow_empty = ⟨`*Bool* `⟩`

> default: `false`

> If true it allows to the region set to be empty (no elements).

---

record: **From_Elements**

Elementary region declared by a list of elements.
The new region is assigned to the list of elements specified by the key `element_list`.

implements abstracts: Region

`name` $= \langle$*String* $\rangle$

>   default: *Obligatory*

>   Name (label) of the region. It has to be unique per single mesh.

`id` $= \langle$*Integer [0, INT]* $\rangle$

>   default: *Optional*

>   Id of the region. If unset, a unique id will be generated automatically.

`element_list` $= \langle$*array [1, UINT] of Integer [0, INT]* $\rangle$

>   default: *Obligatory*

>   List of ids of elements.

---

record: **Union**

Defines a new region (set) as a union of two or more regions. The regions can be given by their names or ids or both.

implements abstracts: Region

`name` $= \langle$*String* $\rangle$

>   default: *Obligatory*

>   Name (label) of the new region. It has to be unique per single mesh.

`region_ids` $= \langle$*array [0, UINT] of Integer [0, INT]* $\rangle$

>   default: *Optional*

>   List of region ids to be added to the new region set.

`regions` $= \langle$*array [0, UINT] of String* $\rangle$

>   default: *Optional*

>   List of region names (labels) to be added to the new region set.

---

record: **Difference**

Defines a new region (set) as a difference of two regions (sets), given by their names.

implements abstracts: Region

name = ⟨*String* ⟩

>   default: *Obligatory*

>   Name (label) of the new region. It has to be unique per single mesh.

regions = ⟨*array [2, 2] of String* ⟩

>   default: *Obligatory*

>   List of exactly two region (set) names.
>   Supposing region sets r1, r2, the result includes all regions of r1 that are not in r2.

---

record: **Intersection**

Defines a new region (set) as an intersection of two or more regions (sets), given by their names.

implements abstracts: Region

name = ⟨*String* ⟩

>   default: *Obligatory*

>   Name (label) of the new region. It has to be unique per single mesh.

regions = ⟨*array [2, UINT] of String* ⟩

>   default: *Obligatory*

>   List of two or more region (set) names.

---

record: **Partition**

Setting for various types of mesh partitioning.

conversion from key: graph_type

tool = ⟨*selection:* PartTool ⟩

>   default: "METIS"

>   Software package used for partitioning. See corresponding selection.

graph_type = ⟨*selection:* GraphType ⟩

>   default: "any_neighboring"

>   Algorithm for generating graph and its weights from a multidimensional mesh.

---

selection: **PartTool**

Select the partitioning tool to use.

values:

PETSc :   Use PETSc interface to various partitioning tools.

METIS :   Use direct interface to Metis.

---

selection: **GraphType**

Different algorithms to make the sparse graph with weighted edges
from the multidimensional mesh. Main difference is dealing with
neighboring of elements of different dimension.

values:

any_neighboring :   Add an edge for any pair of neighboring elements.

any_weight_lower_dim_cuts :   Same as before and assign higher weight to cuts of
   lower dimension in order to make them stick to one face.

same_dimension_neighboring :   Add an edge for any pair of neighboring elements of
   the same dimension (bad for matrix multiply).

---

selection: **Types of search algorithm for finding intersection candidates.**

---

values:

BIHsearch :   Use BIH for finding initial candidates, then continue by prolongation.

BIHonly :   Use BIH for finding all candidates.

BBsearch :   Use bounding boxes for finding initial candidates, then continue by pro-
   longation.

---

abstract: **DarcyFlow**

Darcy flow model. Abstraction of various porous media flow models.

implementations:

Flow_Darcy_LMH, Coupling_Iterative, Flow_Darcy_MH, Flow_Richards_LMH

---

record: **Flow_Darcy_LMH**

Lumped Mixed-Hybrid solver for saturated Darcy flow.

implements abstracts: DarcyFlow

time = ⟨*record:* TimeGovernor ⟩

> default: {}

> Time governor setting.

gravity = ⟨*array [3, 3] of Double (-inf, +inf)* ⟩

> default: [0, 0, -1]

> Vector of the gravity force. Dimensionless.

input_fields = ⟨*array [0, UINT] of record:* Flow_Darcy_LMH_Data ⟩

> default: *Obligatory*

> Input data for Darcy flow model.

nonlinear_solver = ⟨*record:* NonlinearSolver ⟩

> default: {}

> Non-linear solver for MH problem.

output_stream = ⟨*record:* OutputStream ⟩

> default: {}

> Output stream settings.
> Specify file format, precision etc.

output = ⟨*gen. record:* EquationOutput ⟩

> *gen. parameters:* output_field_selection = Flow_Darcy_LMH:OutputFields

> default: {"fields": ["pressure_p0", "velocity_p0"]}

> Specification of output fields and output times.

output_specific = ⟨*gen. record:* Output_DarcyMHSpecific ⟩

> *gen. parameters:* output_field_selection = Flow_Darcy_MH_specific:OutputFields

> default: *Optional*

> Output settings specific to Darcy flow model.
> Includes raw output and some experimental functionality.

balance = ⟨*record:* `Balance` ⟩

> default: `{}`

> Settings for computing mass balance.

mortar_method = ⟨*selection:* `MH_MortarMethod` ⟩

> default: `"None"`

> Method for coupling Darcy flow between dimensions on incompatible meshes. [Experimental]

---

record: **Flow_Darcy_LMH_Data**

Record to set fields of the equation.
The fields are set only on the domain specified by one of the keys: 'region', 'rid'
and after the time given by the key 'time'. The field setting can be overridden by
any Flow_Darcy_LMH_Data record that comes later in the boundary data array.

region = ⟨*array [1, UINT] of String* ⟩

> default: *Optional*

> Labels of the regions where to set fields.

rid = ⟨*Integer [0, INT]* ⟩

> default: *Optional*

> ID of the region where to set fields.

time = ⟨*tuple:* `TimeValue` ⟩

> default: `0.0`

> Apply field setting in this record after this time.
> These times have to form an increasing sequence.

anisotropy = ⟨*gen. abstract:* `Field_R3_to_R[3,3]` ⟩

> *gen. parameters:* `element_input_type` = `Double`

> default: *Optional*

> Anisotropy of the conductivity tensor. $[-]$

cross_section = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

> *gen. parameters:* `element_input_type` = `Double`

> default: *Optional*

> Complement dimension parameter (cross section for 1D, thickness for 2D). $[m^{3-d}]$

conductivity $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Isotropic conductivity scalar. $[ms^{-1}]$

sigma $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Transition coefficient between dimensions. $[-]$

water_source_density $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Water source density. $[s^{-1}]$

bc_type $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

    gen. parameters: `element_input_type` = `Flow_Darcy_BC_Type`

    default: *Optional*

    Boundary condition type. $[-]$

bc_pressure $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Prescribed pressure value on the boundary. Used for all values of `bc_type` except `none` and `seepage`. See documentation of `bc_type` for exact meaning of `bc_pressure` in individual boundary condition types. $[m]$

bc_flux $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Incoming water boundary flux. Used for bc_types : `total_flux`, `seepage`, `river`. $[ms^{-1}]$

bc_robin_sigma $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Conductivity coefficient in the `total_flux` or the `river` boundary condition type. $[s^{-1}]$

`bc_switch_pressure` = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Critical switch pressure for `seepage` and `river` boundary conditions. $[m]$

`init_pressure` = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Initial condition for pressure in time dependent problems. $[m]$

`storativity` = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Storativity (in time dependent problems). $[m^{-1}]$

`gravity` = ⟨*gen. abstract:* `Field_R3_to_R[3]` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Gravity vector. $[-]$

`bc_gravity` = ⟨*gen. abstract:* `Field_R3_to_R[3]` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Boundary gravity vector. $[-]$

`init_piezo_head` = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Initial condition for the pressure given as the piezometric head.

`bc_piezo_head` = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Boundary piezometric head for BC types: dirichlet, robin, and river.

`bc_switch_piezo_head` = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

default: *Optional*

Boundary switch piezometric head for BC types: seepage, river.

---

abstract: **Field_R3_to_R[3,3]**

Abstract for all time-space functions.

default: FieldConstant

implementations:

FieldPython, FieldConstant, FieldFormula, FieldTimeFunction, FieldFE

---

record: **FieldPython**

R3_to_R[3,3] Field given by a Python script.

implements abstracts: Field_R3_to_R[3,3]

unit = ⟨*record:* Unit ⟩

> default: *Optional*
>
> Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

script_string = ⟨*String* ⟩

> default: implicit value: "Obligatory if 'script_file' is not given. "
>
> Python script given as in place string

script_file = ⟨*Filename* ⟩

> default: implicit value: "Obligatory if 'script_striong' is not given. "
>
> Python script given as external file

function = ⟨*String* ⟩

> default: *Obligatory*
>
> Function in the given script that returns tuple containing components of the return type.
> For NxM tensor values: tensor(row,col) = tuple(M*row + col).

---

record: **FieldConstant**

R3_to_R[3,3] Field constant in space.

implements abstracts: `Field_R3_to_R[3,3]`

conversion from key: `value`

unit = ⟨*record:* `Unit` ⟩

> default: *Optional*

> Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

value = ⟨*array [1, UINT] of array [1, UINT] of parameter: element_input_type* ⟩

> default: *Obligatory*

> Value of the constant field. For vector values, you can use scalar value to enter constant vector. For square $N \times N$-matrix values, you can use: - vector of size N to enter diagonal matrix

> > – vector of size $\frac{1}{2}N(N+1)$ to enter symmetric matrix (upper triangle, row by row)

> > – scalar to enter multiple of the unit matrix.

record: **FieldFormula**

R3_to_R[3,3] Field given by runtime interpreted formula.

implements abstracts: `Field_R3_to_R[3,3]`

conversion from key: `value`

unit = ⟨*record:* `Unit` ⟩

> default: *Optional*

> Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

value = ⟨*array [1, UINT] of array [1, UINT] of String* ⟩

> default: *Obligatory*

> String, array of strings, or matrix of strings with formulas for individual entries of scalar, vector, or tensor value respectively.
> For vector values, you can use just one string to enter homogeneous vector.
> For square $N \times N$-matrix values, you can use:

> > – array of strings of size N to enter diagonal matrix

- array of strings of size $\frac{1}{2}N(N+1)$ to enter symmetric matrix (upper triangle, row by row)
- just one string to enter (spatially variable) multiple of the unit matrix. Formula can contain variables `x,y,z,t,d` and usual operators and functions.

surface_direction $= \langle\textit{String}\,\rangle$

> default: `"0 0 1"`

> The vector used to project evaluation point onto the surface.

surface_region $= \langle\textit{String}\,\rangle$

> default: *Optional*

> The name of region set considered as the surface. You have to set surface region if you want to use formula variable `d`.

---

record: **FieldTimeFunction**

R3_to_R[3,3] Field time-dependent function in space.

implements abstracts: Field_R3_to_R[3,3]

conversion from key: time_function

unit $= \langle\textit{record:}$ Unit $\rangle$

> default: *Optional*

> Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

time_function $= \langle\textit{record:}$ TableFunction $\rangle$

> default: *Obligatory*

> Values of time series initialization of Field.

---

record: **TableFunction**

Allow set variable series initialization of Fields.

conversion from key: values

values $= \langle\textit{array [2, UINT] of tuple:}$ IndependentValue $\rangle$

> default: *Obligatory*

> Initizaliation values of Field.

---
tuple: **IndependentValue**

Value of Field for time variable.

t = ⟨*tuple:* TimeValue ⟩

>   default: *Obligatory*

>   Time stamp.

value = ⟨*array [1, UINT] of array [1, UINT] of parameter: element_input_type* ⟩

>   default: *Obligatory*

>   Value of the field in given stamp.

---
record: **FieldFE**

R3_to_R[3,3] Field given by finite element approximation.

implements abstracts: Field_R3_to_R[3,3]

unit = ⟨*record:* Unit ⟩

>   default: *Optional*

>   Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

mesh_data_file = ⟨*Filename* ⟩

>   default: *Obligatory*

>   GMSH mesh with data. Can be different from actual computational mesh.

input_discretization = ⟨*selection:* FE_discretization ⟩

>   default: *Optional*

>   Section where to find the field.
>   Some sections are specific to file format: point_data/node_data, cell_data/element_data, -/element_node_data, native/-.
>   If not given by a user, we try to find the field in all sections, but we report an error if it is found in more than one section.

field_name = ⟨*String* ⟩

>   default: *Obligatory*

>   The values of the Field are read from the `$ElementData` section with field name given by this key.

`default_value` $= \langle$*Double (-inf, +inf)* $\rangle$

> default: *Optional*
>
> Default value is set on elements which values have not been listed in the mesh data file.

`time_unit` $= \langle$*record:* `Unit` $\rangle$

> default: implicit value: "Common time unit of the equation's Time Governor. See the key 'common_time_unit'."
>
> Definition of the unit of all times defined in the mesh data file.

`read_time_shift` $= \langle$*tuple:* `TimeValue` $\rangle$

> default: `0.0`
>
> This key allows reading field data from the mesh data file shifted in time. Considering the time 't', field descriptor with time 'T', time shift 'S', then if 't > T', we read the time frame 't + S'.

`interpolation` $= \langle$*selection:* `interpolation` $\rangle$

> default: `"equivalent_mesh"`
>
> Type of interpolation applied to the input spatial data.
> The default value 'equivalent_mesh' assumes the data being constant on elements living on the same mesh as the computational mesh, but possibly with different numbering. In the case of the same numbering, the user can set 'identical_mesh' to omit algorithm for guessing node and element renumbering. Alternatively, in case of different input mesh, several interpolation algorithms are available.

---

selection: **FE_discretization**

Specify the section in mesh input file where field data is listed.
Some sections are specific to file format.

values:

`node_data` : point_data (VTK) / node_data (GMSH)

`element_data` : cell_data (VTK) / element_data (GMSH)

`element_node_data` : element_node_data (only for GMSH)

`native_data` : native_data (only for VTK)

---

selection: **interpolation**

Specify interpolation of the input data from its input mesh to the computational mesh.

values:

`identic_mesh` : Topology and indices of nodes and elements ofthe input mesh and the computational mesh are identical. This interpolation is typically used for GMSH input files containing only the field values without explicit mesh specification.

`equivalent_mesh` : Topologies of the input mesh and the computational mesh are the same, the node and element numbering may differ. This interpolation can be used also for VTK input data.

`P0_gauss` : Topologies of the input mesh and the computational mesh may differ. Constant values on the elements of the computational mesh are evaluated using the Gaussian quadrature of the fixed order 4, where the quadrature points and their values are found in the input mesh and input data using the BIH tree search.

`P0_intersection` : Topologies of the input mesh and the computational mesh may differ. Can be applied only for boundary fields. For every (boundary) element of the computational mesh the intersection with the input mesh is computed. Constant values on the elements of the computational mesh are evaluated as the weighted average of the (constant) values on the intersecting elements of the input mesh.

---

abstract: **Field_R3_to_R**

Abstract for all time-space functions.

default: FieldConstant

implementations:

FieldPython, FieldConstant, FieldFormula, FieldTimeFunction, FieldFE

---

record: **FieldPython**

R3_to_R Field given by a Python script.

implements abstracts: Field_R3_to_R

`unit` = ⟨*record:* Unit ⟩

> default: *Optional*
>
> Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

`script_string` = ⟨*String* ⟩

> default: implicit value: "Obligatory if 'script_file' is not given. "

> Python script given as in place string

`script_file` = ⟨*Filename* ⟩

> default: implicit value: "Obligatory if 'script_striong' is not given. "

> Python script given as external file

`function` = ⟨*String* ⟩

> default: *Obligatory*

> Function in the given script that returns tuple containing components of the return type.
> For NxM tensor values: tensor(row,col) = tuple(M*row + col).

---

record: **FieldConstant**

R3_to_R Field constant in space.

implements abstracts: `Field_R3_to_R`

conversion from key: `value`

`unit` = ⟨*record:* `Unit` ⟩

> default: *Optional*

> Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

`value` = ⟨*parameter: element_input_type* ⟩

> default: *Obligatory*

> Value of the constant field. For vector values, you can use scalar value to enter constant vector. For square $N \times N$-matrix values, you can use: - vector of size N to enter diagonal matrix

> > – vector of size $\frac{1}{2}N(N+1)$ to enter symmetric matrix (upper triangle, row by row)

> > – scalar to enter multiple of the unit matrix.

---

record: **FieldFormula**

R3_to_R Field given by runtime interpreted formula.

implements abstracts: `Field_R3_to_R`

conversion from key: `value`

unit = ⟨*record:* `Unit` ⟩

    default: *Optional*

    Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

value = ⟨*String* ⟩

    default: *Obligatory*

    String, array of strings, or matrix of strings with formulas for individual entries of scalar, vector, or tensor value respectively.
    For vector values, you can use just one string to enter homogeneous vector.
    For square $N \times N$-matrix values, you can use:

- – array of strings of size N to enter diagonal matrix
- – array of strings of size $\frac{1}{2}N(N+1)$ to enter symmetric matrix (upper triangle, row by row)
- – just one string to enter (spatially variable) multiple of the unit matrix.
  Formula can contain variables `x,y,z,t,d` and usual operators and functions.

surface_direction = ⟨*String* ⟩

    default: `"0 0 1"`

    The vector used to project evaluation point onto the surface.

surface_region = ⟨*String* ⟩

    default: *Optional*

    The name of region set considered as the surface. You have to set surface region if you want to use formula variable `d`.

---

record: **FieldTimeFunction**

R3_to_R Field time-dependent function in space.

implements abstracts: `Field_R3_to_R`

conversion from key: `time_function`

unit $= \langle record:$ Unit $\rangle$

> default: *Optional*

> Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

time_function $= \langle record:$ TableFunction $\rangle$

> default: *Obligatory*

> Values of time series initialization of Field.

---

record: **TableFunction**

Allow set variable series initialization of Fields.

conversion from key: values

values $= \langle array\ [2,\ UINT]\ of\ tuple:$ IndependentValue $\rangle$

> default: *Obligatory*

> Initizaliation values of Field.

---

tuple: **IndependentValue**

Value of Field for time variable.

t $= \langle tuple:$ TimeValue $\rangle$

> default: *Obligatory*

> Time stamp.

value $= \langle parameter:\ element\_input\_type \rangle$

> default: *Obligatory*

> Value of the field in given stamp.

---

record: **FieldFE**

R3_to_R Field given by finite element approximation.

implements abstracts: Field_R3_to_R

unit $= \langle$ *record:* Unit $\rangle$

> default: *Optional*

> Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

mesh_data_file $= \langle$ *Filename* $\rangle$

> default: *Obligatory*

> GMSH mesh with data. Can be different from actual computational mesh.

input_discretization $= \langle$ *selection:* FE_discretization $\rangle$

> default: *Optional*

> Section where to find the field.
> Some sections are specific to file format: point_data/node_data, cell_data/element_data, -/element_node_data, native/-.
> If not given by a user, we try to find the field in all sections, but we report an error if it is found in more than one section.

field_name $= \langle$ *String* $\rangle$

> default: *Obligatory*

> The values of the Field are read from the `$ElementData` section with field name given by this key.

default_value $= \langle$ *Double (-inf, +inf)* $\rangle$

> default: *Optional*

> Default value is set on elements which values have not been listed in the mesh data file.

time_unit $= \langle$ *record:* Unit $\rangle$

> default: implicit value: "Common time unit of the equation's Time Governor. See the key 'common_time_unit'."

> Definition of the unit of all times defined in the mesh data file.

read_time_shift $= \langle$ *tuple:* TimeValue $\rangle$

> default: `0.0`

> This key allows reading field data from the mesh data file shifted in time. Considering the time 't', field descriptor with time 'T', time shift 'S', then if 't > T', we read the time frame 't + S'.

interpolation $= \langle$ *selection:* interpolation $\rangle$

> default: `"equivalent_mesh"`

Type of interpolation applied to the input spatial data.

The default value 'equivalent_mesh' assumes the data being constant on elements living on the same mesh as the computational mesh, but possibly with different numbering. In the case of the same numbering, the user can set 'identical_mesh' to omit algorithm for guessing node and element renumbering. Alternatively, in case of different input mesh, several interpolation algorithms are available.

---

selection: **`Flow_Darcy_BC_Type`**

---

values:

`none` : Homogeneous Neumann boundary condition (zero normal flux over the boundary).

`dirichlet` : Dirichlet boundary condition. Specify the pressure head through the `bc_pressure` field or the piezometric head through the `bc_piezo_head` field.

`total_flux` : Flux boundary condition (combines Neumann and Robin type). Water inflow equal to $\delta_d(q_d^N + \sigma_d(h_d^R - h_d))$. Specify the water inflow by the `bc_flux` field, the transition coefficient by `bc_robin_sigma` and the reference pressure head or piezometric head through `bc_pressure` or `bc_piezo_head` respectively.

`seepage` : Seepage face boundary condition. Pressure and inflow bounded from above. Boundary with potential seepage flow is described by the pair of inequalities: $h_d \leq h_d^D$ and $-\boldsymbol{q}_d \cdot \boldsymbol{n} \leq \delta q_d^N$, where the equality holds in at least one of them. Caution: setting q_d^N strictly negative may lead to an ill posed problem since a positive outflow is enforced. Parameters h_d^D and q_d^N are given by the fields `bc_switch_pressure` (or `bc_switch_piezo_head`) and `bc_flux` respectively.

`river` : River boundary condition. For the water level above the bedrock, $H_d > H_d^S$, the Robin boundary condition is used with the inflow given by: $\delta_d(q_d^N + \sigma_d(H_d^D - H_d))$. For the water level under the bedrock, constant infiltration is used: $\delta_d(q_d^N + \sigma_d(H_d^D - H_d^S))$. Parameters: `bc_pressure`, `bc_switch_pressure`, `bc_sigma`, `bc_flux`.

---

abstract: **`Field_R3_to_R[3]`**

Abstract for all time-space functions.

default: `FieldConstant`

implementations:

`FieldPython`, `FieldConstant`, `FieldFormula`, `FieldTimeFunction`, `FieldFE`

---

record: **FieldPython**

R3_to_R[3] Field given by a Python script.

implements abstracts: Field_R3_to_R[3]

unit $= \langle$ *record:* Unit $\rangle$

 default: *Optional*

 Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

script_string $= \langle$ *String* $\rangle$

 default: implicit value: "Obligatory if 'script_file' is not given. "

 Python script given as in place string

script_file $= \langle$ *Filename* $\rangle$

 default: implicit value: "Obligatory if 'script_striong' is not given. "

 Python script given as external file

function $= \langle$ *String* $\rangle$

 default: *Obligatory*

 Function in the given script that returns tuple containing components of the return type.
For NxM tensor values: tensor(row,col) = tuple(M*row + col).

---

record: **FieldConstant**

R3_to_R[3] Field constant in space.

implements abstracts: Field_R3_to_R[3]

conversion from key: value

unit $= \langle$ *record:* Unit $\rangle$

 default: *Optional*

 Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

`value` = ⟨*array [1, 3] of parameter: element_input_type* ⟩

> default: *Obligatory*
>
> Value of the constant field. For vector values, you can use scalar value to enter constant vector. For square $N \times N$-matrix values, you can use: - vector of size N to enter diagonal matrix
>
> > – vector of size $\frac{1}{2}N(N+1)$ to enter symmetric matrix (upper triangle, row by row)
> >
> > – scalar to enter multiple of the unit matrix.

---

record: **FieldFormula**

R3_to_R[3] Field given by runtime interpreted formula.

implements abstracts: Field_R3_to_R[3]

conversion from key: value

`unit` = ⟨*record:* Unit ⟩

> default: *Optional*
>
> Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

`value` = ⟨*array [1, UINT] of String* ⟩

> default: *Obligatory*
>
> String, array of strings, or matrix of strings with formulas for individual entries of scalar, vector, or tensor value respectively.
> For vector values, you can use just one string to enter homogeneous vector.
> For square $N \times N$-matrix values, you can use:
>
> > – array of strings of size N to enter diagonal matrix
> >
> > – array of strings of size $\frac{1}{2}N(N+1)$ to enter symmetric matrix (upper triangle, row by row)
> >
> > – just one string to enter (spatially variable) multiple of the unit matrix.
> > Formula can contain variables `x,y,z,t,d` and usual operators and functions.

`surface_direction` = ⟨*String* ⟩

> default: `"0 0 1"`
>
> The vector used to project evaluation point onto the surface.

surface_region = ⟨*String* ⟩

> default: *Optional*
>
> The name of region set considered as the surface. You have to set surface region if you want to use formula variable d.

---

record: **FieldTimeFunction**

R3_to_R[3] Field time-dependent function in space.

implements abstracts: Field_R3_to_R[3]

conversion from key: time_function

unit = ⟨*record:* Unit ⟩

> default: *Optional*
>
> Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

time_function = ⟨*record:* TableFunction ⟩

> default: *Obligatory*
>
> Values of time series initialization of Field.

---

record: **TableFunction**

Allow set variable series initialization of Fields.

conversion from key: values

values = ⟨*array [2, UINT] of tuple:* IndependentValue ⟩

> default: *Obligatory*
>
> Initizaliation values of Field.

---

tuple: **IndependentValue**

Value of Field for time variable.

t = ⟨*tuple:* TimeValue ⟩

> default: *Obligatory*

> Time stamp.

value = ⟨*array [1, 3] of parameter: element_input_type* ⟩

> default: *Obligatory*

> Value of the field in given stamp.

===

record: **FieldFE**

===

R3_to_R[3] Field given by finite element approximation.

implements abstracts: Field_R3_to_R[3]

unit = ⟨*record:* Unit ⟩

> default: *Optional*

> Unit of the field values provided in the main input file, in the external file, or by a function (FieldPython).

mesh_data_file = ⟨*Filename* ⟩

> default: *Obligatory*

> GMSH mesh with data. Can be different from actual computational mesh.

input_discretization = ⟨*selection:* FE_discretization ⟩

> default: *Optional*

> Section where to find the field.
> Some sections are specific to file format: point_data/node_data, cell_data/element_data, -/element_node_data, native/-.
> If not given by a user, we try to find the field in all sections, but we report an error if it is found in more than one section.

field_name = ⟨*String* ⟩

> default: *Obligatory*

> The values of the Field are read from the $ElementData section with field name given by this key.

default_value = ⟨*Double (-inf, +inf)* ⟩

> default: *Optional*

> Default value is set on elements which values have not been listed in the mesh data file.

`time_unit` = ⟨*record:* `Unit` ⟩

> default: implicit value: "Common time unit of the equation's Time Governor. See the key 'common_time_unit'."

> Definition of the unit of all times defined in the mesh data file.

`read_time_shift` = ⟨*tuple:* `TimeValue` ⟩

> default: `0.0`

> This key allows reading field data from the mesh data file shifted in time. Considering the time 't', field descriptor with time 'T', time shift 'S', then if 't > T', we read the time frame 't + S'.

`interpolation` = ⟨*selection:* `interpolation` ⟩

> default: `"equivalent_mesh"`

> Type of interpolation applied to the input spatial data.
> The default value 'equivalent_mesh' assumes the data being constant on elements living on the same mesh as the computational mesh, but possibly with different numbering. In the case of the same numbering, the user can set 'identical_mesh' to omit algorithm for guessing node and element renumbering. Alternatively, in case of different input mesh, several interpolation algorithms are available.

---

record: **NonlinearSolver**

Non-linear solver settings.

`linear_solver` = ⟨*abstract:* `LinSys` ⟩

> default: `{}`

> Linear solver for MH problem.

`tolerance` = ⟨*Double [0, +inf)* ⟩

> default: `1e-06`

> Residual tolerance.

`min_it` = ⟨*Integer [0, INT]* ⟩

> default: `1`

> Minimum number of iterations (linear solutions) to use.
> This is usefull if the convergence criteria does not characterize your goal well enough so it converges prematurely, possibly even without a single linear solution. If greater then 'max_it' the value is set to 'max_it'.

`max_it = ⟨`*Integer [0, INT]*`⟩`

>   default: `100`

>   Maximum number of iterations (linear solutions) of the non-linear solver.

`converge_on_stagnation = ⟨`*Bool*`⟩`

>   default: `false`

>   If a stagnation of the nonlinear solver is detected the solver stops. A divergence is reported by default, forcing the end of the simulation. By setting this flag to 'true', the solver ends with convergence success on stagnation, but it reports warning about it.

---

abstract: **LinSys**

Linear solver settings.

default: Petsc

implementations:

Petsc, Bddc

---

record: **Petsc**

PETSc solver settings.
It provides interface to various PETSc solvers. The convergence criteria is:
`norm( res_i ) < max( norm( res_0 ) * r_tol, a_tol )`
where `res_i` is the residuum vector after i-th iteration of the solver and `res_0` is the estimate of the norm of the initial residual. If the initial guess of the solution is provided (usually only for transient equations) the residual of this estimate is used, otherwise the norm of preconditioned RHS is used. The default norm is L_2 norm of preconditioned residual: $P^{-1}(Ax - b)$, usage of other norm may be prescribed using the 'option' key. See also PETSc documentation for KSPSetNormType.

implements abstracts: LinSys

`r_tol = ⟨`*Double [0, 1]*`⟩`

>   default: implicit value: "Default value is set by the nonlinear solver or the equation. If not, we use the value 1.0e-7."

>   Residual tolerance relative to the initial error.

`a_tol = ⟨`*Double [0, +inf)*`⟩`

>   default: implicit value: "Default value is set by the nonlinear solver or the equation. If not, we use the value 1.0e-11."

>   Absolute residual tolerance.

max_it = $\langle$*Integer [0, INT]*$\rangle$

> default: implicit value: "Default value is set by the nonlinear solver or the equation. If not, we use the value 1000."

> Maximum number of outer iterations of the linear solver.

options = $\langle$*String*$\rangle$

> default: `""`

> This options is passed to PETSC to create a particular KSP (Krylov space method). If the string is left empty (by default), the internal default options is used.

---

record: **Bddc**

BDDCML (Balancing Domain Decomposition by Constraints - Multi-Level) solver settings.

implements abstracts: LinSys

r_tol = $\langle$*Double [0, 1]*$\rangle$

> default: implicit value: "Default value is set by the nonlinear solver or the equation. If not, we use the value 1.0e-7."

> Residual tolerance relative to the initial error.

max_it = $\langle$*Integer [0, INT]*$\rangle$

> default: implicit value: "Default value is set by the nonlinear solver or the equation. If not, we use the value 1000."

> Maximum number of outer iterations of the linear solver.

max_nondecr_it = $\langle$*Integer [0, INT]*$\rangle$

> default: 30

> Maximum number of iterations of the linear solver with non-decreasing residual.

number_of_levels = $\langle$*Integer [0, INT]*$\rangle$

> default: 2

> Number of levels in the multilevel method (=2 for the standard BDDC).

use_adaptive_bddc = $\langle$*Bool*$\rangle$

> default: false

> Use adaptive selection of constraints in BDDCML.

bddcml_verbosity_level = ⟨*Integer [0, 2]* ⟩

>   default: 0

>   Level of verbosity of the BDDCML library:

>   − 0 - no output,
>   − 1 - mild output,
>   − 2 - detailed output.

---

record: **OutputStream**

Configuration of the spatial output of a single balance equation.

file = ⟨*Filename* ⟩

>   default: implicit value: "Name of the equation associated with the output stream."

>   File path to the connected output file.

format = ⟨*abstract:* OutputTime ⟩

>   default: {}

>   File format of the output stream and possible parameters.

times = ⟨*array [0, UINT] of record:* TimeGrid ⟩

>   default: *Optional*

>   Output times used for fields that do not have their own output times defined.

output_mesh = ⟨*record:* OutputMesh ⟩

>   default: *Optional*

>   Output mesh record enables output on a refined mesh [EXPERIMENTAL, VTK only].Sofar refinement is performed only in discontinous sense. Therefore only corner and element data can be written on refined output mesh. Node data are to be transformed to corner data, native data cannot be written. Do not include any node or native data in output fields.

precision = ⟨*Integer [0, INT]* ⟩

>   default: 17

>   The number of decimal digits used in output of floating point values.
>   Default is 17 decimal digits which are necessary to reproduce double values exactly after write-read cycle.

`observe_points` = ⟨*array [0, UINT] of record:* `ObservePoint` ⟩

> default: `[]`

> Array of observe points.

---

abstract: **OutputTime**

Format of output stream and possible parameters.

default: vtk

implementations:

vtk, gmsh

---

record: **vtk**

Parameters of vtk output format.

implements abstracts: OutputTime

`variant` = ⟨*selection:* VTK variant (ascii or binary) ⟩

> default: `"ascii"`

> Variant of output stream file format.

`parallel` = ⟨*Bool* ⟩

> default: `false`

> Parallel or serial version of file format.

---

selection: **VTK variant (ascii or binary)**

---

values:

`ascii` :   ASCII variant of VTK file format

`binary` :   Uncompressed appended binary XML VTK format without usage of base64 encoding of appended data.

`binary_zlib` :   Appended binary XML VTK format without usage of base64 encoding of appended data. Compressed with ZLib.

---

record: **gmsh**

Parameters of gmsh output format.

implements abstracts: OutputTime

---

record: **TimeGrid**

Equally spaced grid of time points.

conversion from key: begin

begin = $\langle$*tuple:* TimeValue $\rangle$

    default: implicit value: "The initial time of the associated equation."

    The start time of the grid.

step = $\langle$*tuple:* TimeValue $\rangle$

    default: *Optional*

    The step of the grid. If not specified, the grid consists of the single time given by the begin key.

end = $\langle$*tuple:* TimeValue $\rangle$

    default: implicit value: "The end time of the simulation."

    The time greater or equal to the last time in the grid.

---

record: **OutputMesh**

Parameters of the refined output mesh. [Not impemented]

max_level = $\langle$*Integer [1, 20]* $\rangle$

    default: 3

    Maximal level of refinement of the output mesh.

refine_by_error = $\langle$*Bool* $\rangle$

    default: false

    Set true for using error_control_field. Set false for global uniform refinement to max_level.

error_control_field = ⟨*String* ⟩

>   default: *Optional*

>   Name of an output field, according to which the output mesh will be refined. The field must be a SCALAR one.

refinement_error_tolerance = ⟨*Double [0, +inf)* ⟩

>   default: `0.01`

>   Tolerance for element refinement by error. If tolerance is reached, refinement is stopped. Relative difference between error control field and its linear approximation on element is computedand compared with tolerance.

---

record: **ObservePoint**

Specification of the observation point.
The actual observation element and the observation point on it is determined as follows:

1. Find an initial element containing the initial point. If no such element exists, we report an error.

2. Use BFS (Breadth-first search) starting from the inital element to find the 'observe element'. The observe element is the closest element.

3. Find the closest projection of the inital point on the observe element and snap this projection according to the `snap_dim`.

conversion from key: point

name = ⟨*String* ⟩

>   default: implicit value: "Default name have the form 'obs_<id>', where 'id' is the rank of the point on the input."

>   Optional point name, which has to be unique.
>   Any string that is a valid YAML key in record without any quoting can be used, however, using just alpha-numerical characters, and underscore instead of the space, is recommended.

point = ⟨*array [3, 3] of Double (-inf, +inf)* ⟩

>   default: *Obligatory*

>   Initial point for the observe point search.

snap_dim = ⟨*Integer [0, 4]* ⟩

>   default: `4`

The dimension of the sub-element to which center we snap. For value 4 no snapping is done. For values 0 up to 3 the element containing the initial point is found and then the observepoint is snapped to the nearest center of the sub-element of the given dimension. E.g. for dimension 2 we snap to the nearest center of the face of the initial element.

snap_region = ⟨*String* ⟩

default: `"ALL"`

The region of the initial element for snapping. Without snapping we make a projection to the initial element.

search_radius = ⟨*Double [0, +inf)* ⟩

default: implicit value: "Maximal distance of the observe point from the mesh relative to the mesh diameter. "

Global value is defined in mesh record by the key global_snap_radius.

---

record: **EquationOutput**

Output of the equation's fields. The output is done through the output stream of the associated balance law equation. The stream defines output format for the full space information in selected times and observe points for the full time information. The key 'fields' select the fields for the full spatial output. The set of output times may be specified per field otherwise common time set 'times' is used. If even this is not providedthe time set of the output_stream is used. The initial time of the equation is automatically added to the time set of every selected field. The end time of the equation is automatically added to the common output time set.

times = ⟨*array [0, UINT] of record:* **TimeGrid** ⟩

default: *Optional*

Output times used for the output fields without is own time series specification.

add_input_times = ⟨*Bool* ⟩

default: `false`

Add all input time points of the equation, mentioned in the 'input_fields' list, also as the output points.

fields = ⟨*array [0, UINT] of record:* **FieldOutputSetting** ⟩

default: `[]`

Array of output fields and their individual output settings.

observe_fields = ⟨*array [0, UINT] of parameter: output_field_selection* ⟩

   default: []

   Array of the fields evaluated in the observe points of the associated output stream.

---

record: **FieldOutputSetting**

Setting of the field output. The field name, output times, output interpolation (future).

conversion from key: field

field = ⟨*parameter: output_field_selection* ⟩

   default: *Obligatory*

   The field name (from selection).

times = ⟨*array [0, UINT] of record:* TimeGrid ⟩

   default: *Optional*

   Output times specific to particular field.

interpolation = ⟨*array [0, UINT] of selection:* Discrete_output ⟩

   default: implicit value: "Interpolation type of output data."

   Optional value. Implicit value is given by field and can be changed.

---

selection: **Discrete_output**

Discrete type of output. Determines type of output data (element, node, native etc).
values:

P1_average :   Node data / point data.

D1_value :   Corner data.

P0_value :   Element data / cell data.

Native :   Native data (Flow123D data).

---

selection: **Flow_Darcy_LMH:OutputFields**

Selection of output fields for the Flow_Darcy_LMH model.
values:

subdomain : [−] Input field: Subdomain ids of the domain decomposition.

region_id : [−] Input field: Region ids.

pressure_p0 : [m] Pressure solution - P0 interpolation.

piezo_head_p0 : [m] Piezo head solution - P0 interpolation.

velocity_p0 : [ms^{-1}] Velocity solution - P0 interpolation.

flux : [ms^{-1}] Darcy flow flux.

anisotropy : [−] Input field: Anisotropy of the conductivity tensor.

cross_section : [m^{3-d}] Input field: Complement dimension parameter (cross section for 1D, thickness for 2D).

conductivity : [ms^{-1}] Input field: Isotropic conductivity scalar.

sigma : [−] Input field: Transition coefficient between dimensions.

water_source_density : [s^{-1}] Input field: Water source density.

init_pressure : [m] Input field: Initial condition for pressure in time dependent problems.

storativity : [m^{-1}] Input field: Storativity (in time dependent problems).

gravity : [−] Input field: Gravity vector.

init_piezo_head : [m] Input field: Init piezo head.

---

record: **Output_DarcyMHSpecific**

Specific Darcy flow MH output.

times = ⟨*array [0, UINT] of record:* TimeGrid ⟩

    default: *Optional*

    Output times used for the output fields without is own time series specification.

add_input_times = ⟨*Bool* ⟩

    default: **false**

    Add all input time points of the equation, mentioned in the 'input_fields' list, also as the output points.

fields = ⟨*array [0, UINT] of record:* `FieldOutputSetting` ⟩

> default: `[]`
>
> Array of output fields and their individual output settings.

observe_fields = ⟨*array [0, UINT] of parameter: output_field_selection* ⟩

> default: `[]`
>
> Array of the fields evaluated in the observe points of the associated output stream.

compute_errors = ⟨*Bool* ⟩

> default: `false`
>
> SPECIAL PURPOSE. Computes error norms of the solution, particulary suited for non-compatible coupling models.

raw_flow_output = ⟨*Filename* ⟩

> default: *Optional*
>
> Output file with raw data from MH module.

---

selection: **Flow_Darcy_MH_specific:OutputFields**

Selection of output fields for the Flow_Darcy_MH_specific model.

values:

pressure_diff : $[m]$ Error norm of the pressure solution. [Experimental]

velocity_diff : $[ms^{-1}]$ Error norm of the velocity solution. [Experimental]

div_diff : $[s^{-1}]$ Error norm of the divergence of the velocity solution. [Experimental]

---

record: **Balance**

Balance of a conservative quantity, boundary fluxes and sources.

times = ⟨*array [0, UINT] of record:* `TimeGrid` ⟩

> default: `[]`

add_output_times = ⟨*Bool* ⟩

> default: `true`
>
> Add all output times of the balanced equation to the balance output times set. Note that this is not the time set of the output stream.

format $= \langle$ *selection:* `Balance_output_format` $\rangle$

> default: `"txt"`

> Format of output file.

cumulative $= \langle$ *Bool* $\rangle$

> default: `false`

> Compute cumulative balance over time. If true, then balance is calculated at each computational time step, which can slow down the program.

file $= \langle$ *Filename* $\rangle$

> default: implicit value: "File name generated from the balanced quantity: <quantity_name>_balance.*"

> File name for output of balance.

---

selection: **Balance_output_format**

Format of output file for balance.

values:

`legacy` : Legacy format used by previous program versions.

`txt` : Excel format with tab delimiter.

`gnuplot` : Format compatible with GnuPlot datafile with fixed column width.

---

selection: **MH_MortarMethod**

---

values:

`None` : No Mortar method is applied.

`P0` : Mortar space: P0 on elements of lower dimension.

`P1` : Mortar space: P1 on intersections, using non-conforming pressures.

---

record: **Coupling_Iterative**

Record with data for iterative coupling of flow and mechanics.

implements abstracts: `DarcyFlow`

`max_it = ⟨`*Integer [0, INT]* `⟩`

>   default: `100`

>   Maximal count of HM iterations.

`min_it = ⟨`*Integer [0, INT]* `⟩`

>   default: `1`

>   Minimal count of HM iterations.

`a_tol = ⟨`*Double [0, +inf)* `⟩`

>   default: `0`

>   Absolute tolerance for difference in HM iteration.

`r_tol = ⟨`*Double [0, +inf)* `⟩`

>   default: `1e-07`

>   Relative tolerance for difference in HM iteration.

`time = ⟨`*record:* `TimeGovernor` `⟩`

>   default: `{}`

>   Time governor setting.

`flow_equation = ⟨`*record:* `Flow_Darcy_LMH` `⟩`

>   default: *Obligatory*

>   Flow equation, provides the velocity field as a result.

`mechanics_equation = ⟨`*record:* `Mechanics_LinearElasticity_FE` `⟩`

>   default: *Optional*

>   Mechanics, provides the displacement field.

`input_fields = ⟨`*array [0, UINT] of record:* `Coupling_Iterative:Data` `⟩`

>   default: *Obligatory*

>   Input fields of the HM coupling.

`iteration_parameter = ⟨`*Double (-inf, +inf)* `⟩`

>   default: `1`

>   Tuning parameter for iterative splitting. Its default value corresponds to a theo-retically optimal value with fastest convergence.

---

record: **Mechanics_LinearElasticity_FE**

FEM for linear elasticity.

`time` = ⟨*record:* TimeGovernor ⟩

> default: {}

> Time governor setting.

`balance` = ⟨*record:* Balance ⟩

> default: {}

> Settings for computing balance.

`output_stream` = ⟨*record:* OutputStream ⟩

> default: *Obligatory*

> Parameters of output stream.

`solver` = ⟨*record:* Petsc ⟩

> default: *Obligatory*

> Linear solver for elasticity.

`input_fields` = ⟨*array [0, UINT] of record:* Mechanics_LinearElasticity_FE:Data ⟩

> default: *Obligatory*

> Input fields of the equation.

`output` = ⟨*gen. record:* EquationOutput ⟩

> *gen. parameters:* `output_field_selection` = Mechanics_LinearElasticity_FE:OutputFields

> default: {"fields": ["displacement"]}

> Setting of the field output.

---

record: **Mechanics_LinearElasticity_FE:Data**

Record to set fields of the equation.
The fields are set only on the domain specified by one of the keys: 'region', 'rid'
and after the time given by the key 'time'. The field setting can be overridden by
any Mechanics_LinearElasticity_FE:Data record that comes later in the boundary data
array.

`region` = ⟨*array [1, UINT] of String* ⟩

> default: *Optional*

> Labels of the regions where to set fields.

`rid` $= \langle$ *Integer [0, INT]* $\rangle$

>   default: *Optional*

>   ID of the region where to set fields.

`time` $= \langle$ *tuple:* `TimeValue` $\rangle$

>   default: `0.0`

>   Apply field setting in this record after this time.
>   These times have to form an increasing sequence.

`bc_type` $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

>   *gen. parameters:* `element_input_type` = `Elasticity_BC_Type`

>   default: *Optional*

>   Type of boundary condition. $[-]$

`bc_displacement` $= \langle$ *gen. abstract:* `Field_R3_to_R[3]` $\rangle$

>   *gen. parameters:* `element_input_type` = `Double`

>   default: *Optional*

>   Prescribed displacement on boundary. $[m]$

`bc_traction` $= \langle$ *gen. abstract:* `Field_R3_to_R[3]` $\rangle$

>   *gen. parameters:* `element_input_type` = `Double`

>   default: *Optional*

>   Prescribed traction on boundary. $[m^{-1}kgs^{-2}]$

`bc_stress` $= \langle$ *gen. abstract:* `Field_R3_to_R[3,3]` $\rangle$

>   *gen. parameters:* `element_input_type` = `Double`

>   default: *Optional*

>   Prescribed stress on boundary. $[m^{-1}kgs^{-2}]$

`load` $= \langle$ *gen. abstract:* `Field_R3_to_R[3]` $\rangle$

>   *gen. parameters:* `element_input_type` = `Double`

>   default: *Optional*

>   Prescribed bulk load. $[m^{-3}kgs^{-2}]$

`young_modulus` $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

>   *gen. parameters:* `element_input_type` = `Double`

>   default: *Optional*

>   Young's modulus. $[m^{-1}kgs^{-2}]$

poisson_ratio = ⟨*gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Poisson's ratio. $[-]$

fracture_sigma = ⟨*gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Coefficient of transfer of forces through fractures. $[-]$

initial_stress = ⟨*gen. abstract:* Field_R3_to_R[3,3] ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Initial stress tensor. $[m^{-1}kgs^{-2}]$

lame_mu = ⟨*gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Field lame_mu. $[m^{-1}kgs^{-2}]$

lame_lambda = ⟨*gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Field lame_lambda. $[m^{-1}kgs^{-2}]$

dirichlet_penalty = ⟨*gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Field dirichlet_penalty. $[m^{-1}kgs^{-2}]$

---

selection: **Elasticity_BC_Type**

Types of boundary conditions for mechanics.

values:

displacement : Prescribed displacement.

displacement_n : Prescribed displacement in the normal direction to the boundary.

traction : Prescribed traction.

stress : Prescribed stress tensor.

---

selection: **Mechanics_LinearElasticity_FE:OutputFields**

Selection of output fields for the Mechanics_LinearElasticity_FE model.

values:

load : $[m^{-3}kgs^{-2}]$ Input field: Prescribed bulk load.

young_modulus : $[m^{-1}kgs^{-2}]$ Input field: Young's modulus.

poisson_ratio : $[-]$ Input field: Poisson's ratio.

fracture_sigma : $[-]$ Input field: Coefficient of transfer of forces through fractures.

initial_stress : $[m^{-1}kgs^{-2}]$ Input field: Initial stress tensor.

region_id : $[-]$ Input field:

subdomain : $[-]$ Input field:

displacement : $[m]$ Displacement vector field output.

stress : $[m^{-1}kgs^{-2}]$ Stress tensor output.

von_mises_stress : $[m^{-1}kgs^{-2}]$ von Mises stress output.

mean_stress : $[m^{-1}kgs^{-2}]$ mean stress output.

cross_section_updated : $[m]$ Cross-section after deformation - output.

displacement_divergence : $[-]$ Displacement divergence output.

lame_mu : $[m^{-1}kgs^{-2}]$ Input field: Field lame_mu.

lame_lambda : $[m^{-1}kgs^{-2}]$ Input field: Field lame_lambda.

dirichlet_penalty : $[m^{-1}kgs^{-2}]$ Input field: Field dirichlet_penalty.

---

record: **Coupling_Iterative:Data**

Record to set fields of the equation.
The fields are set only on the domain specified by one of the keys: 'region', 'rid'
and after the time given by the key 'time'. The field setting can be overridden by
any Coupling_Iterative:Data record that comes later in the boundary data array.

region $= \langle array\ [1,\ UINT]\ of\ String\ \rangle$

> default: *Optional*

> Labels of the regions where to set fields.

rid $= \langle Integer\ [0,\ INT]\ \rangle$

> default: *Optional*

> ID of the region where to set fields.

time $= \langle tuple:$ TimeValue $\rangle$

> default: `0.0`

> Apply field setting in this record after this time.
> These times have to form an increasing sequence.

biot_alpha $= \langle gen.\ abstract:$ Field_R3_to_R $\rangle$

> *gen. parameters:* `element_input_type = ` Double

> default: *Optional*

> Biot poroelastic coefficient. $[-]$

fluid_density $= \langle gen.\ abstract:$ Field_R3_to_R $\rangle$

> *gen. parameters:* `element_input_type = ` Double

> default: *Optional*

> Volumetric mass density of the fluid. $[m^{-3}kg]$

gravity $= \langle gen.\ abstract:$ Field_R3_to_R $\rangle$

> *gen. parameters:* `element_input_type = ` Double

> default: *Optional*

> Gravitational acceleration constant. $[ms^{-2}]$

---

record: **Flow_Darcy_MH**

Mixed-Hybrid solver for saturated Darcy flow.

implements abstracts: DarcyFlow

time $= \langle record:$ TimeGovernor $\rangle$

> default: `{}`

> Time governor setting.

`gravity` $= \langle$ *array [3, 3] of Double (-inf, +inf)* $\rangle$

default: `[0, 0, -1]`

Vector of the gravity force. Dimensionless.

`input_fields` $= \langle$ *array [0, UINT] of record:* `Flow_Darcy_MH_Data` $\rangle$

default: *Obligatory*

Input data for Darcy flow model.

`nonlinear_solver` $= \langle$ *record:* `NonlinearSolver` $\rangle$

default: `{}`

Non-linear solver for MH problem.

`output_stream` $= \langle$ *record:* `OutputStream` $\rangle$

default: `{}`

Output stream settings.
Specify file format, precision etc.

`output` $= \langle$ *gen. record:* `EquationOutput` $\rangle$

gen. parameters: `output_field_selection = ` `Flow_Darcy_MH:OutputFields`

default: `{"fields": ["pressure_p0", "velocity_p0"]}`

Specification of output fields and output times.

`output_specific` $= \langle$ *gen. record:* `Output_DarcyMHSpecific` $\rangle$

gen. parameters: `output_field_selection = ` `Flow_Darcy_MH_specific:OutputFields`

default: *Optional*

Output settings specific to Darcy flow model.
Includes raw output and some experimental functionality.

`balance` $= \langle$ *record:* `Balance` $\rangle$

default: `{}`

Settings for computing mass balance.

`n_schurs` $= \langle$ *Integer [0, 2]* $\rangle$

default: `2`

Number of Schur complements to perform when solving MH system.

`mortar_method` $= \langle$ *selection:* `MH_MortarMethod` $\rangle$

default: `"None"`

Method for coupling Darcy flow between dimensions on incompatible meshes. [Experimental]

146

Record to set fields of the equation.
The fields are set only on the domain specified by one of the keys: 'region', 'rid'
and after the time given by the key 'time'. The field setting can be overridden by
any Flow_Darcy_MH_Data record that comes later in the boundary data array.

region = ⟨*array [1, UINT] of String* ⟩

> default: *Optional*

> Labels of the regions where to set fields.

rid = ⟨*Integer [0, INT]* ⟩

> default: *Optional*

> ID of the region where to set fields.

time = ⟨*tuple:* **TimeValue** ⟩

> default: **0.0**

> Apply field setting in this record after this time.
> These times have to form an increasing sequence.

anisotropy = ⟨*gen. abstract:* **Field_R3_to_R[3,3]** ⟩

> *gen. parameters:* **element_input_type = Double**

> default: *Optional*

> Anisotropy of the conductivity tensor. $[-]$

cross_section = ⟨*gen. abstract:* **Field_R3_to_R** ⟩

> *gen. parameters:* **element_input_type = Double**

> default: *Optional*

> Complement dimension parameter (cross section for 1D, thickness for 2D). $[m^{3-d}]$

conductivity = ⟨*gen. abstract:* **Field_R3_to_R** ⟩

> *gen. parameters:* **element_input_type = Double**

> default: *Optional*

> Isotropic conductivity scalar. $[ms^{-1}]$

sigma = ⟨*gen. abstract:* **Field_R3_to_R** ⟩

> *gen. parameters:* **element_input_type = Double**

> default: *Optional*

> Transition coefficient between dimensions. $[-]$

water_source_density = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Water source density. $[s^{-1}]$

bc_type = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Flow_Darcy_BC_Type`

    default: *Optional*

    Boundary condition type. $[-]$

bc_pressure = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Prescribed pressure value on the boundary. Used for all values of `bc_type` except `none` and `seepage`. See documentation of `bc_type` for exact meaning of `bc_pressure` in individual boundary condition types. $[m]$

bc_flux = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Incoming water boundary flux. Used for bc_types : `total_flux`, `seepage`, `river`. $[ms^{-1}]$

bc_robin_sigma = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Conductivity coefficient in the `total_flux` or the `river` boundary condition type. $[s^{-1}]$

bc_switch_pressure = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Critical switch pressure for `seepage` and `river` boundary conditions. $[m]$

init_pressure = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Initial condition for pressure in time dependent problems. $[m]$

storativity $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

    gen. parameters: `element_input_type = Double`

    default: *Optional*

    Storativity (in time dependent problems). $[m^{-1}]$

gravity $= \langle$ *gen. abstract:* `Field_R3_to_R[3]` $\rangle$

    gen. parameters: `element_input_type = Double`

    default: *Optional*

    Gravity vector. $[-]$

bc_gravity $= \langle$ *gen. abstract:* `Field_R3_to_R[3]` $\rangle$

    gen. parameters: `element_input_type = Double`

    default: *Optional*

    Boundary gravity vector. $[-]$

init_piezo_head $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

    gen. parameters: `element_input_type = Double`

    default: *Optional*

    Initial condition for the pressure given as the piezometric head.

bc_piezo_head $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

    gen. parameters: `element_input_type = Double`

    default: *Optional*

    Boundary piezometric head for BC types: dirichlet, robin, and river.

bc_switch_piezo_head $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

    gen. parameters: `element_input_type = Double`

    default: *Optional*

    Boundary switch piezometric head for BC types: seepage, river.

---

selection: **Flow_Darcy_MH:OutputFields**

Selection of output fields for the Flow_Darcy_MH model.

values:

subdomain : $[-]$ Input field: Subdomain ids of the domain decomposition.

region_id : [−] Input field: Region ids.

pressure_p0 : [m] Pressure solution - P0 interpolation.

piezo_head_p0 : [m] Piezo head solution - P0 interpolation.

velocity_p0 : [ms^{-1}] Velocity solution - P0 interpolation.

flux : [ms^{-1}] Darcy flow flux.

anisotropy : [−] Input field: Anisotropy of the conductivity tensor.

cross_section : [m^{3-d}] Input field: Complement dimension parameter (cross section for 1D, thickness for 2D).

conductivity : [ms^{-1}] Input field: Isotropic conductivity scalar.

sigma : [−] Input field: Transition coefficient between dimensions.

water_source_density : [s^{-1}] Input field: Water source density.

init_pressure : [m] Input field: Initial condition for pressure in time dependent problems.

storativity : [m^{-1}] Input field: Storativity (in time dependent problems).

gravity : [−] Input field: Gravity vector.

init_piezo_head : [m] Input field: Init piezo head.

---

record: **Flow_Richards_LMH**

Lumped Mixed-Hybrid solver for unsteady unsaturated Darcy flow.

implements abstracts: DarcyFlow

time = ⟨*record:* TimeGovernor ⟩

   default: {}

   Time governor setting.

gravity = ⟨*array [3, 3] of Double (-inf, +inf)* ⟩

   default: [0, 0, -1]

   Vector of the gravity force. Dimensionless.

input_fields = ⟨*array [0, UINT] of record:* RichardsLMH_Data ⟩

   default: *Obligatory*

   Input data for Darcy flow model.

`nonlinear_solver` = ⟨*record:* `NonlinearSolver` ⟩

> default: {}

> Non-linear solver for MH problem.

`output_stream` = ⟨*record:* `OutputStream` ⟩

> default: {}

> Output stream settings.
> Specify file format, precision etc.

`output` = ⟨*gen. record:* `EquationOutput` ⟩

> *gen. parameters:* `output_field_selection` = `Flow_Richards_LMH:OutputFields`

> default: {"fields": ["pressure_p0", "velocity_p0"]}

> Specification of output fields and output times.

`output_specific` = ⟨*gen. record:* `Output_DarcyMHSpecific` ⟩

> *gen. parameters:* `output_field_selection` = `Flow_Darcy_MH_specific:OutputFields`

> default: *Optional*

> Output settings specific to Darcy flow model.
> Includes raw output and some experimental functionality.

`balance` = ⟨*record:* `Balance` ⟩

> default: {}

> Settings for computing mass balance.

`mortar_method` = ⟨*selection:* `MH_MortarMethod` ⟩

> default: "None"

> Method for coupling Darcy flow between dimensions on incompatible meshes. [Experimental]

`soil_model` = ⟨*record:* `SoilModel` ⟩

> default: "van_genuchten"

> Soil model settings.

---

record: **RichardsLMH_Data**

Record to set fields of the equation.
The fields are set only on the domain specified by one of the keys: 'region', 'rid'
and after the time given by the key 'time'. The field setting can be overridden by
any RichardsLMH_Data record that comes later in the boundary data array.

`region` $= \langle$ *array [1, UINT] of String* $\rangle$

> default: *Optional*

> Labels of the regions where to set fields.

`rid` $= \langle$ *Integer [0, INT]* $\rangle$

> default: *Optional*

> ID of the region where to set fields.

`time` $= \langle$ *tuple:* `TimeValue` $\rangle$

> default: `0.0`

> Apply field setting in this record after this time.
> These times have to form an increasing sequence.

`anisotropy` $= \langle$ *gen. abstract:* `Field_R3_to_R[3,3]` $\rangle$

> *gen. parameters:* `element_input_type` = `Double`

> default: *Optional*

> Anisotropy of the conductivity tensor. $[-]$

`cross_section` $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

> *gen. parameters:* `element_input_type` = `Double`

> default: *Optional*

> Complement dimension parameter (cross section for 1D, thickness for 2D). $[m^{3-d}]$

`conductivity` $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

> *gen. parameters:* `element_input_type` = `Double`

> default: *Optional*

> Isotropic conductivity scalar. $[ms^{-1}]$

`sigma` $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

> *gen. parameters:* `element_input_type` = `Double`

> default: *Optional*

> Transition coefficient between dimensions. $[-]$

`water_source_density` $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

> *gen. parameters:* `element_input_type` = `Double`

> default: *Optional*

> Water source density. $[s^{-1}]$

bc_type = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

   gen. parameters: `element_input_type` = `Flow_Darcy_BC_Type`

   default: *Optional*

   Boundary condition type. $[-]$

bc_pressure = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

   gen. parameters: `element_input_type` = `Double`

   default: *Optional*

   Prescribed pressure value on the boundary. Used for all values of `bc_type` except `none` and `seepage`. See documentation of `bc_type` for exact meaning of `bc_pressure` in individual boundary condition types. $[m]$

bc_flux = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

   gen. parameters: `element_input_type` = `Double`

   default: *Optional*

   Incoming water boundary flux. Used for bc_types : `total_flux`, `seepage`, `river`. $[ms^{-1}]$

bc_robin_sigma = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

   gen. parameters: `element_input_type` = `Double`

   default: *Optional*

   Conductivity coefficient in the `total_flux` or the `river` boundary condition type. $[s^{-1}]$

bc_switch_pressure = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

   gen. parameters: `element_input_type` = `Double`

   default: *Optional*

   Critical switch pressure for `seepage` and `river` boundary conditions. $[m]$

init_pressure = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

   gen. parameters: `element_input_type` = `Double`

   default: *Optional*

   Initial condition for pressure in time dependent problems. $[m]$

storativity = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

   gen. parameters: `element_input_type` = `Double`

   default: *Optional*

   Storativity (in time dependent problems). $[m^{-1}]$

gravity = ⟨*gen. abstract:* `Field_R3_to_R[3]` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Gravity vector. [−]

bc_gravity = ⟨*gen. abstract:* `Field_R3_to_R[3]` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Boundary gravity vector. [−]

init_piezo_head = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Initial condition for the pressure given as the piezometric head.

bc_piezo_head = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Boundary piezometric head for BC types: dirichlet, robin, and river.

bc_switch_piezo_head = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Boundary switch piezometric head for BC types: seepage, river.

water_content_saturated = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Saturated water content θ_s.
    Relative volume of water in a reference volume of a saturated porous media. [−]

water_content_residual = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Residual water content θ_r.
    Relative volume of water in a reference volume of an ideally dry porous media.
    [−]

genuchten_p_head_scale = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

   gen. parameters: `element_input_type` = `Double`

   default: *Optional*

   The van Genuchten pressure head scaling parameter α.
   It is related to the inverse of the air entry pressure, i.e. the pressure
   where the relative water content starts to decrease below 1. $[m^{-1}]$

genuchten_n_exponent = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

   gen. parameters: `element_input_type` = `Double`

   default: *Optional*

   The van Genuchten exponent parameter n. $[-]$

---

selection: **Flow_Richards_LMH:OutputFields**

Selection of output fields for the Flow_Richards_LMH model.

values:

subdomain : $[-]$ Input field: Subdomain ids of the domain decomposition.

region_id : $[-]$ Input field: Region ids.

pressure_p0 : $[m]$ Pressure solution - P0 interpolation.

piezo_head_p0 : $[m]$ Piezo head solution - P0 interpolation.

velocity_p0 : $[ms^{-1}]$ Velocity solution - P0 interpolation.

flux : $[ms^{-1}]$ Darcy flow flux.

anisotropy : $[-]$ Input field: Anisotropy of the conductivity tensor.

cross_section : $[m^{3-d}]$ Input field: Complement dimension parameter (cross section for 1D, thickness for 2D).

conductivity : $[ms^{-1}]$ Input field: Isotropic conductivity scalar.

sigma : $[-]$ Input field: Transition coefficient between dimensions.

water_source_density : $[s^{-1}]$ Input field: Water source density.

init_pressure : $[m]$ Input field: Initial condition for pressure in time dependent problems.

storativity : $[m^{-1}]$ Input field: Storativity (in time dependent problems).

`gravity` : [−] Input field: Gravity vector.

`init_piezo_head` : [m] Input field: Init piezo head.

`water_content` : [−] Water content.
 It is a fraction of water volume to the whole volume.

`conductivity_richards` : [ms^{-1}] Computed isotropic scalar conductivity by the soil
 model.

`water_content_saturated` : [−] Input field: Saturated water content θ_s.
 Relative volume of water in a reference volume of a saturated porous media.

`water_content_residual` : [−] Input field: Residual water content θ_r.
 Relative volume of water in a reference volume of an ideally dry porous media.

`genuchten_p_head_scale` : [m^{-1}] Input field: The van Genuchten pressure head
 scaling parameter α.
 It is related to the inverse of the air entry pressure, i.e. the pressure
 where the relative water content starts to decrease below 1.

`genuchten_n_exponent` : [−] Input field: The van Genuchten exponent parameter n.

---

record: **SoilModel**

Soil model settings.

conversion from key: `model_type`

`model_type` = ⟨*selection:* `Soil_Model_Type` ⟩

 default: `"van_genuchten"`

 Selection of the globally applied soil model. In future we replace this key by a
 field for selection of the model. That will allow usage of different soil model in a
 single simulation.

`cut_fraction` = ⟨*Double [0, 1]* ⟩

 default: `0.999`

 Fraction of the water content where we cut and rescale the curve.

---

selection: **Soil_Model_Type**

---

values:

van_genuchten :   Van Genuchten soil model with cutting near zero.

irmay :   Irmay model for conductivity, Van Genuchten model for the water content. Suitable for bentonite.

---

abstract: **AdvectionProcess**

Abstract advection process. In particular: transport of substances or heat transfer.

implementations:

Coupling_OperatorSplitting, Heat_AdvectionDiffusion_DG

---

record: **Coupling_OperatorSplitting**

Transport by convection and/or diffusion
coupled with reaction and adsorption model (ODE per element)
via operator splitting.

implements abstracts: AdvectionProcess

time = ⟨*record:* TimeGovernor ⟩

> default: {}

> Time governor setting.

balance = ⟨*record:* Balance ⟩

> default: {}

> Settings for computing mass balance.

output_stream = ⟨*record:* OutputStream ⟩

> default: {}

> Output stream settings.
> Specify file format, precision etc.

substances = ⟨*array [1, UINT] of record:* Substance ⟩

> default: *Obligatory*

> Specification of transported substances.

transport = ⟨*abstract:* Solute ⟩

> default: *Obligatory*

> Type of the numerical method for the transport equation.

reaction_term $= \langle$ *abstract:* ReactionTerm $\rangle$

> default: *Optional*

> Reaction model involved in transport.

---

record: **Substance**

Chemical substance.

conversion from key: name

name $= \langle$ *String* $\rangle$

> default: *Obligatory*

> Name of the substance.

molar_mass $= \langle$ *Double [0, +inf)* $\rangle$

> default: **1**

> Molar mass of the substance [kg/mol].

---

abstract: **Solute**

Transport of soluted substances.

implementations:

Solute_Advection_FV, Solute_AdvectionDiffusion_DG

---

record: **Solute_Advection_FV**

Finite volume method, explicit in time, for advection only solute transport.

implements abstracts: Solute

input_fields $= \langle$ *array [0, UINT] of record:* Solute_Advection_FV:Data $\rangle$

> default: *Obligatory*

output $= \langle$ *gen. record:* EquationOutput $\rangle$

> *gen. parameters:* output_field_selection = Solute_Advection_FV:OutputFields

> default: {"fields": ["conc"]}

> Specification of output fields and output times.

record: **Solute_Advection_FV:Data**

Record to set fields of the equation.
The fields are set only on the domain specified by one of the keys: 'region', 'rid'
and after the time given by the key 'time'. The field setting can be overridden by
any Solute_Advection_FV:Data record that comes later in the boundary data array.

region = ⟨*array [1, UINT] of String* ⟩

    default: *Optional*

    Labels of the regions where to set fields.

rid = ⟨*Integer [0, INT]* ⟩

    default: *Optional*

    ID of the region where to set fields.

time = ⟨*tuple:* TimeValue ⟩

    default: 0.0

    Apply field setting in this record after this time.
    These times have to form an increasing sequence.

porosity = ⟨*gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Porosity of the mobile phase. $[-]$

sources_density = ⟨*array [1, UINT] of gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Density of concentration sources. $[m^{-3}kgs^{-1}]$

sources_sigma = ⟨*array [1, UINT] of gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Concentration flux. $[s^{-1}]$

sources_conc = ⟨*array [1, UINT] of gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Concentration sources threshold. $[m^{-3}kg]$

bc_conc $= \langle$ *array [1, UINT] of gen. abstract:* `Field_R3_to_R` $\rangle$

  gen. parameters: `element_input_type = Double`

  default: *Optional*

  Boundary condition for concentration of substances. $[m^{-3}kg]$

init_conc $= \langle$ *array [1, UINT] of gen. abstract:* `Field_R3_to_R` $\rangle$

  gen. parameters: `element_input_type = Double`

  default: *Optional*

  Initial values for concentration of substances. $[m^{-3}kg]$

---

selection: **`Solute_Advection_FV:OutputFields`**

Selection of output fields for the Solute_Advection_FV model.

values:

`porosity` : $[-]$ Input field: Porosity of the mobile phase.

`water_content` : $[-]$ Input field: INTERNAL. Water content passed from unsaturated Darcy flow model.

`sources_density` : $[m^{-3}kgs^{-1}]$ Input field: Density of concentration sources.

`sources_sigma` : $[s^{-1}]$ Input field: Concentration flux.

`sources_conc` : $[m^{-3}kg]$ Input field: Concentration sources threshold.

`init_conc` : $[m^{-3}kg]$ Input field: Initial values for concentration of substances.

`conc` : $[m^{-3}kg]$ Concentration solution.

`region_id` : $[-]$ Input field: Region ids.

`subdomain` : $[-]$ Input field: Subdomain ids of the domain decomposition.

---

record: **`Solute_AdvectionDiffusion_DG`**

Discontinuous Galerkin (DG) solver for solute transport.

implements abstracts: `Solute`

solvent_density $= \langle$ *Double [0, +inf)* $\rangle$

> default: `1.0`

> Density of the solvent [$kg.m^{-3}$].

solver $= \langle$ *record:* `Petsc` $\rangle$

> default: `{}`

> Solver for the linear system.

user_fields $= \langle$ *array [0, UINT] of record:* `Solute_AdvectionDiffusion_DG:UserData` $\rangle$

> default: *Optional*

> Input fields of the equation defined by user.

input_fields $= \langle$ *array [0, UINT] of record:* `Solute_AdvectionDiffusion_DG:Data` $\rangle$

> default: *Obligatory*

> Input fields of the equation.

dg_variant $= \langle$ *selection:* `DG_variant` $\rangle$

> default: `"non-symmetric"`

> Variant of the interior penalty discontinuous Galerkin method.

dg_order $= \langle$ *Integer [0, 3]* $\rangle$

> default: `1`

> Polynomial order for the finite element in DG method (order 0 is suitable if there is no diffusion/dispersion).

init_projection $= \langle$ *Bool* $\rangle$

> default: `true`

> If true, use DG projection of the initial condition field. Otherwise, evaluate initial condition field directly (well suited for reading native data).

output $= \langle$ *gen. record:* `EquationOutput` $\rangle$

> *gen. parameters:* `output_field_selection` = `Solute_AdvectionDiffusion_DG:OutputFields`

> default: `{"fields": ["conc"]}`

> Specification of output fields and output times.

---

record: **Solute_AdvectionDiffusion_DG:UserData**

Record to set fields of the equation: Solute_AdvectionDiffusion_DG.

name $= \langle \textit{String} \rangle$

> default: *Obligatory*

> Name of user defined field.

is_boundary $= \langle \textit{Bool} \rangle$

> default: `false`

> Type of field: boundary or bulk.

scalar_field $= \langle \textit{gen. abstract:} \text{ Field\_R3\_to\_R} \rangle$

> *gen. parameters:* `element_input_type = Double`

> default: *Obligatory*

> Instance of FieldAlgoBase ScalarField descendant.
> One of keys 'scalar_field', 'vector_field', 'tensor_field' must be set.
> If you set more than one of these keys, only first key is accepted.

vector_field $= \langle \textit{gen. abstract:} \text{ Field\_R3\_to\_R[3]} \rangle$

> *gen. parameters:* `element_input_type = Double`

> default: *Optional*

> Instance of FieldAlgoBase VectorField descendant. See above for details.

tensor_field $= \langle \textit{gen. abstract:} \text{ Field\_R3\_to\_R[3,3]} \rangle$

> *gen. parameters:* `element_input_type = Double`

> default: *Optional*

> Instance of FieldAlgoBase TensorField descendant. See above for details.

---

record: **Solute_AdvectionDiffusion_DG:Data**

Record to set fields of the equation.
The fields are set only on the domain specified by one of the keys: 'region', 'rid'
and after the time given by the key 'time'. The field setting can be overridden by
any Solute_AdvectionDiffusion_DG:Data record that comes later in the boundary data
array.

region $= \langle \textit{array [1, UINT] of String} \rangle$

> default: *Optional*

> Labels of the regions where to set fields.

`rid` $= \langle$*Integer [0, INT]* $\rangle$

>   default: *Optional*

>   ID of the region where to set fields.

`time` $= \langle$*tuple:* `TimeValue` $\rangle$

>   default: `0.0`

>   Apply field setting in this record after this time.
>   These times have to form an increasing sequence.

`porosity` $= \langle$*gen. abstract:* `Field_R3_to_R` $\rangle$

>   *gen. parameters:* `element_input_type = Double`

>   default: *Optional*

>   Porosity of the mobile phase. $[-]$

`sources_density` $= \langle$*array [1, UINT] of gen. abstract:* `Field_R3_to_R` $\rangle$

>   *gen. parameters:* `element_input_type = Double`

>   default: *Optional*

>   Density of concentration sources. $[m^{-3}kgs^{-1}]$

`sources_sigma` $= \langle$*array [1, UINT] of gen. abstract:* `Field_R3_to_R` $\rangle$

>   *gen. parameters:* `element_input_type = Double`

>   default: *Optional*

>   Concentration flux. $[s^{-1}]$

`sources_conc` $= \langle$*array [1, UINT] of gen. abstract:* `Field_R3_to_R` $\rangle$

>   *gen. parameters:* `element_input_type = Double`

>   default: *Optional*

>   Concentration sources threshold. $[m^{-3}kg]$

`bc_type` $= \langle$*array [1, UINT] of gen. abstract:* `Field_R3_to_R` $\rangle$

>   *gen. parameters:* `element_input_type = Solute_AdvectionDiffusion_BC_Type`

>   default: *Optional*

>   Type of boundary condition. $[-]$

`bc_conc` $= \langle$*array [1, UINT] of gen. abstract:* `Field_R3_to_R` $\rangle$

>   *gen. parameters:* `element_input_type = Double`

>   default: *Optional*

>   Dirichlet boundary condition (for each substance). $[m^{-3}kg]$

bc_flux = ⟨*array [1, UINT] of gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Flux in Neumann boundary condition. $[m^{1-d}kgs^{-1}]$

bc_robin_sigma = ⟨*array [1, UINT] of gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Conductivity coefficient in Robin boundary condition. $[m^{4-d}s^{-1}]$

init_conc = ⟨*array [1, UINT] of gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Initial values for concentration of substances. $[m^{-3}kg]$

disp_l = ⟨*array [1, UINT] of gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Longitudinal dispersivity in the liquid (for each substance). $[m]$

disp_t = ⟨*array [1, UINT] of gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Transverse dispersivity in the liquid (for each substance). $[m]$

diff_m = ⟨*array [1, UINT] of gen. abstract:* Field_R3_to_R[3,3] ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Molecular diffusivity in the liquid (for each substance). $[m^2s^{-1}]$

rock_density = ⟨*gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Rock matrix density. $[m^{-3}kg]$

sorption_coefficient = ⟨*array [1, UINT] of gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

default: *Optional*

Coefficient of linear sorption. $[m^3kg^{-1}]$

v_norm $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

gen. parameters: `element_input_type` = `Double`

default: *Optional*

Velocity norm field. $[ms^{-1}]$

mass_matrix_coef $= \langle$ *gen. abstract:* `Field_R3_to_R` $\rangle$

gen. parameters: `element_input_type` = `Double`

default: *Optional*

Matrix coefficients computed by model in mass assemblation. $[m^{3-d}]$

retardation_coef $= \langle$ *array [1, UINT] of gen. abstract:* `Field_R3_to_R` $\rangle$

gen. parameters: `element_input_type` = `Double`

default: *Optional*

Retardation coefficients computed by model in mass assemblation. $[m^{3-d}]$

sources_density_out $= \langle$ *array [1, UINT] of gen. abstract:* `Field_R3_to_R` $\rangle$

gen. parameters: `element_input_type` = `Double`

default: *Optional*

Concentration sources output - density of substance source, only positive part is used.. $[m^{-d}kgs^{-1}]$

sources_sigma_out $= \langle$ *array [1, UINT] of gen. abstract:* `Field_R3_to_R` $\rangle$

gen. parameters: `element_input_type` = `Double`

default: *Optional*

Concentration sources - Robin type, in_flux = sources_sigma * (sources_conc - mobile_conc). $[m^{3-d}s^{-1}]$

sources_conc_out $= \langle$ *array [1, UINT] of gen. abstract:* `Field_R3_to_R` $\rangle$

gen. parameters: `element_input_type` = `Double`

default: *Optional*

Concentration sources output. $[m^{-3}kg]$

advection_coef $= \langle$ *array [1, UINT] of gen. abstract:* `Field_R3_to_R[3]` $\rangle$

gen. parameters: `element_input_type` = `Double`

default: *Optional*

Advection coefficients model. $[ms^{-1}]$

`diffusion_coef` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R[3,3]` ⟩

> *gen. parameters:* `element_input_type` = `Double`

> default: *Optional*

> Diffusion coefficients model. $[m^2 s^{-1}]$

`fracture_sigma` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R` ⟩

> *gen. parameters:* `element_input_type` = `Double`

> default: *Optional*

> Coefficient of diffusive transfer through fractures (for each substance). $[-]$

`dg_penalty` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R` ⟩

> *gen. parameters:* `element_input_type` = `Double`

> default: *Optional*

> Penalty parameter influencing the discontinuity of the solution (for each substance). Its default value 1 is sufficient in most cases. Higher value diminishes the inter-element jumps. $[-]$

---

selection: **Solute_AdvectionDiffusion_BC_Type**

Types of boundary conditions for advection-diffusion solute transport model.

values:

`inflow` : Default transport boundary condition.
> On water inflow ($q_w \leq 0$), total flux is given by the reference concentration 'bc_conc'. On water outflow we prescribe zero diffusive flux, i.e. the mass flows out only due to advection.

`dirichlet` : Dirichlet boundary condition $c = c_D$.
> The prescribed concentration c_D is specified by the field 'bc_conc'.

`total_flux` : Total mass flux boundary condition.
> The prescribed total incoming flux can have the general form $\delta(f_N + \sigma_R(c_R - c))$, where the absolute flux f_N is specified by the field 'bc_flux', the transition parameter σ_R by 'bc_robin_sigma', and the reference concentration c_R by 'bc_conc'.

`diffusive_flux` : Diffusive flux boundary condition.
> The prescribed incoming mass flux due to diffusion can have the general form $\delta(f_N + \sigma_R(c_R - c))$, where the absolute flux f_N is specified by the field 'bc_flux', the transition parameter σ_R by 'bc_robin_sigma', and the reference concentration c_R by 'bc_conc'.

---

selection: **DG_variant**

Type of penalty term.

values:

`non-symmetric` : non-symmetric weighted interior penalty DG method

`incomplete` : incomplete weighted interior penalty DG method

`symmetric` : symmetric weighted interior penalty DG method

---

selection: **Solute_AdvectionDiffusion_DG:OutputFields**

Selection of output fields for the Solute_AdvectionDiffusion_DG model.

values:

`porosity` : $[-]$ Input field: Porosity of the mobile phase.

`water_content` : $[-]$ Input field: INTERNAL. Water content passed from unsaturated Darcy flow model.

`sources_density` : $[m^{-3}kgs^{-1}]$ Input field: Density of concentration sources.

`sources_sigma` : $[s^{-1}]$ Input field: Concentration flux.

`sources_conc` : $[m^{-3}kg]$ Input field: Concentration sources threshold.

`init_conc` : $[m^{-3}kg]$ Input field: Initial values for concentration of substances.

`disp_l` : $[m]$ Input field: Longitudinal dispersivity in the liquid (for each substance).

`disp_t` : $[m]$ Input field: Transverse dispersivity in the liquid (for each substance).

`diff_m` : $[m^2s^{-1}]$ Input field: Molecular diffusivity in the liquid (for each substance).

`rock_density` : $[m^{-3}kg]$ Input field: Rock matrix density.

`sorption_coefficient` : $[m^3kg^{-1}]$ Input field: Coefficient of linear sorption.

`conc` : $[m^{-3}kg]$ Concentration solution.

`v_norm` : $[ms^{-1}]$ Input field: Velocity norm field.

`mass_matrix_coef` : $[m^{3-d}]$ Input field: Matrix coefficients computed by model in mass assemblation.

retardation_coef : $[m^{3-d}]$ Input field: Retardation coefficients computed by model in mass assemblation.

sources_density_out : $[m^{-d}kgs^{-1}]$ Input field: Concentration sources output - density of substance source, only positive part is used..

sources_sigma_out : $[m^{3-d}s^{-1}]$ Input field: Concentration sources - Robin type, in_flux = sources_sigma * (sources_conc - mobile_conc).

sources_conc_out : $[m^{-3}kg]$ Input field: Concentration sources output.

advection_coef : $[ms^{-1}]$ Input field: Advection coefficients model.

diffusion_coef : $[m^2s^{-1}]$ Input field: Diffusion coefficients model.

fracture_sigma : $[-]$ Input field: Coefficient of diffusive transfer through fractures (for each substance).

dg_penalty : $[-]$ Input field: Penalty parameter influencing the discontinuity of the solution (for each substance). Its default value 1 is sufficient in most cases. Higher value diminishes the inter-element jumps.

region_id : $[-]$ Input field: Region ids.

subdomain : $[-]$ Input field: Subdomain ids of the domain decomposition.

---

abstract: **ReactionTerm**

Abstract equation for a reaction term (dual porosity, sorption, reactions). Can be part of coupling with a transport equation via. operator splitting.

implementations:

FirstOrderReaction, RadioactiveDecay, Sorption, DualPorosity

---

record: **FirstOrderReaction**

A model of first order chemical reactions (decompositions of a reactant into products).

implements abstracts: ReactionTermImmobile, ReactionTermMobile, GenericReaction, ReactionTerm

reactions = ⟨*array [0, UINT] of record:* Reaction ⟩

default: *Obligatory*

An array of first order chemical reactions.

168

---

record: **Reaction**

Describes a single first order chemical reaction.

reactants = ⟨*array [1, UINT] of record:* FirstOrderReactionReactant ⟩

> default: *Obligatory*

> An array of reactants. Do not use array, reactions with only one reactant (decays) are implemented at the moment!

reaction_rate = ⟨*Double [0, +inf)* ⟩

> default: *Obligatory*

> The reaction rate coefficient of the first order reaction.

products = ⟨*array [1, UINT] of record:* FirstOrderReactionProduct ⟩

> default: *Obligatory*

> An array of products.

---

record: **FirstOrderReactionReactant**

A record describing a reactant of a reaction.

conversion from key: name

name = ⟨*String* ⟩

> default: *Obligatory*

> The name of the reactant.

---

record: **FirstOrderReactionProduct**

A record describing a product of a reaction.

conversion from key: name

name = ⟨*String* ⟩

> default: *Obligatory*

> The name of the product.

`branching_ratio` $= \langle$ *Double [0, +inf)* $\rangle$

> default: `1.0`
>
> The branching ratio of the product when there are more products.
> The value must be positive. Further, the branching ratios of all products are normalized in order to sum to one.
> The default value 1.0, should only be used in the case of single product.

---

record: **RadioactiveDecay**

A model of a radioactive decay and possibly of a decay chain.

implements abstracts: `ReactionTermImmobile`, `ReactionTermMobile`, `GenericReaction`, `ReactionTerm`

`decays` $= \langle$ *array [1, UINT] of record:* `Decay` $\rangle$

> default: *Obligatory*
>
> An array of radioactive decays.

---

record: **Decay**

A model of a radioactive decay.

`radionuclide` $= \langle$ *String* $\rangle$

> default: *Obligatory*
>
> The name of the parent radionuclide.

`half_life` $= \langle$ *Double [0, +inf)* $\rangle$

> default: *Obligatory*
>
> The half life of the parent radionuclide in seconds.

`products` $= \langle$ *array [1, UINT] of record:* `RadioactiveDecayProduct` $\rangle$

> default: *Obligatory*
>
> An array of the decay products (daughters).

---

record: **RadioactiveDecayProduct**

A record describing a product of a radioactive decay.

conversion from key: name

name = ⟨*String* ⟩

> default: *Obligatory*

> The name of the product.

energy = ⟨*Double [0, +inf)* ⟩

> default: 0.0

> Not used at the moment! The released energy in MeV from the decay of the radionuclide into the product.

branching_ratio = ⟨*Double [0, +inf)* ⟩

> default: 1.0

> The branching ratio of the product when there is more than one. Considering only one product, the default ratio 1.0 is used. Its value must be positive. Further, the branching ratios of all products are normalizedby their sum, so the sum then gives 1.0 (this also resolves possible rounding errors).

---

record: **Sorption**

Sorption model in the reaction term of transport.

implements abstracts: ReactionTerm

substances = ⟨*array [1, UINT] of String* ⟩

> default: *Obligatory*

> Names of the substances that take part in the sorption model.

solvent_density = ⟨*Double [0, +inf)* ⟩

> default: 1.0

> Density of the solvent.

substeps = ⟨*Integer [1, INT]* ⟩

> default: 1000

> Number of equidistant substeps, molar mass and isotherm intersections

solubility = ⟨*array [0, UINT] of Double [0, +inf)* ⟩

> default: *Optional*

> Specifies solubility limits of all the sorbing species.

`table_limits` = ⟨*array [0, UINT] of Double [-1, +inf)* ⟩

    default: *Optional*

    Specifies the highest aqueous concentration in the isotherm function interpolation table. Use any negative value for an automatic choice according to current maximal concentration (default and recommended). Use '0' to always evaluate isotherm function directly (can be very slow). Use a positive value to set the interpolation table limit manually (if aqueous concentration is higher, then the isotherm function is evaluated directly).

`input_fields` = ⟨*array [0, UINT] of record:* `Sorption:Data` ⟩

    default: *Obligatory*

    Containes region specific data necessary to construct isotherms.

`reaction_liquid` = ⟨*abstract:* `GenericReaction` ⟩

    default: *Optional*

    Reaction model following the sorption in the liquid.

`reaction_solid` = ⟨*abstract:* `GenericReaction` ⟩

    default: *Optional*

    Reaction model following the sorption in the solid.

`output` = ⟨*gen. record:* `EquationOutput` ⟩

    gen. parameters: `output_field_selection` = `Sorption:OutputFields`

    default: `{"fields":  ["conc_solid"]}`

    Setting of the fields output.

---

record: **`Sorption:Data`**

Record to set fields of the equation.
The fields are set only on the domain specified by one of the keys: 'region', 'rid' and after the time given by the key 'time'. The field setting can be overridden by any Sorption:Data record that comes later in the boundary data array.

`region` = ⟨*array [1, UINT] of String* ⟩

    default: *Optional*

    Labels of the regions where to set fields.

`rid` = ⟨*Integer [0, INT]* ⟩

    default: *Optional*

    ID of the region where to set fields.

time $= \langle$ *tuple:* TimeValue $\rangle$

> default: 0.0

> Apply field setting in this record after this time.
> These times have to form an increasing sequence.

rock_density $= \langle$ *gen. abstract:* Field_R3_to_R $\rangle$

> *gen. parameters:* element_input_type = Double

> default: *Optional*

> Rock matrix density. $[m^{-3}kg]$

sorption_type $= \langle$ *array [1, UINT] of gen. abstract:* Field_R3_to_R $\rangle$

> *gen. parameters:* element_input_type = SorptionType

> default: *Optional*

> Considered sorption is described by selected isotherm.
> If porosity on an element is equal to 1.0 (or even higher), meaning no sorbing surface, then type 'none' will be selected automatically. $[-]$

distribution_coefficient $= \langle$ *array [1, UINT] of gen. abstract:* Field_R3_to_R $\rangle$

> *gen. parameters:* element_input_type = Double

> default: *Optional*

> Distribution coefficient k_l, k_F, k_L of linear, Freundlich or Langmuir isotherm respectively. $[m^3kg^{-1}]$

isotherm_other $= \langle$ *array [1, UINT] of gen. abstract:* Field_R3_to_R $\rangle$

> *gen. parameters:* element_input_type = Double

> default: *Optional*

> Additional parameter α of nonlinear isotherms. $[-]$

init_conc_solid $= \langle$ *array [1, UINT] of gen. abstract:* Field_R3_to_R $\rangle$

> *gen. parameters:* element_input_type = Double

> default: *Optional*

> Initial solid concentration of substances. It is a vector: one value for every substance. $[-]$

---

selection: **SorptionType**

---

values:

none : No sorption considered.

linear : Linear isotherm runs the concentration exchange between liquid and solid.

langmuir : Langmuir isotherm runs the concentration exchange between liquid and solid.

freundlich : Freundlich isotherm runs the concentration exchange between liquid and solid.

---

abstract: **GenericReaction**

Abstract equation for a reaction of species in single compartment (e.g. mobile solid).It can be part of: direct operator splitting coupling, dual porosity model, any sorption.

implementations:

FirstOrderReaction, RadioactiveDecay

---

selection: **Sorption:OutputFields**

Selection of output fields for the Sorption model.

values:

rock_density : $[m^{-3}kg]$ Input field: Rock matrix density.

sorption_type : $[-]$ Input field: Considered sorption is described by selected isotherm. If porosity on an element is equal to 1.0 (or even higher), meaning no sorbing surface, then type 'none' will be selected automatically.

distribution_coefficient : $[m^3kg^{-1}]$ Input field: Distribution coefficient k_l, k_F, k_L of linear, Freundlich or Langmuir isotherm respectively.

isotherm_other : $[-]$ Input field: Additional parameter α of nonlinear isotherms.

init_conc_solid : $[-]$ Input field: Initial solid concentration of substances. It is a vector: one value for every substance.

conc_solid : $[-]$ Concentration solution in the solid phase.

---

record: **DualPorosity**

Dual porosity model in transport problems.
Provides computing the concentration of substances in mobile and immobile zone.

implements abstracts: ReactionTerm

input_fields = ⟨*array [0, UINT] of record:* `DualPorosity:Data` ⟩

    default: *Obligatory*

    Contains region specific data necessary to construct dual porosity model.

scheme_tolerance = ⟨*Double [0, +inf)* ⟩

    default: `0.001`

    Tolerance according to which the explicit Euler scheme is used or not. Set 0.0 to use analytic formula only (can be slower).

reaction_mobile = ⟨*abstract:* `ReactionTermMobile` ⟩

    default: *Optional*

    Reaction model in mobile zone.

reaction_immobile = ⟨*abstract:* `ReactionTermImmobile` ⟩

    default: *Optional*

    Reaction model in immobile zone.

output = ⟨*gen. record:* `EquationOutput` ⟩

    gen. parameters: output_field_selection = `DualPorosity:OutputFields`

    default: `{"fields": ["conc_immobile"]}`

    Setting of the fields output.

---

record: **DualPorosity:Data**

Record to set fields of the equation.
The fields are set only on the domain specified by one of the keys: 'region', 'rid' and after the time given by the key 'time'. The field setting can be overridden by any DualPorosity:Data record that comes later in the boundary data array.

region = ⟨*array [1, UINT] of String* ⟩

    default: *Optional*

    Labels of the regions where to set fields.

rid = ⟨*Integer [0, INT]* ⟩

    default: *Optional*

    ID of the region where to set fields.

time $= \langle$ *tuple:* TimeValue $\rangle$

>   default: 0.0

>   Apply field setting in this record after this time.
>   These times have to form an increasing sequence.

diffusion_rate_immobile $= \langle$ *array [1, UINT] of gen. abstract:* Field_R3_to_R $\rangle$

>   *gen. parameters:* element_input_type = Double

>   default: *Optional*

>   Diffusion coefficient of non-equilibrium linear exchange between mobile and immobile zone. $[s^{-1}]$

porosity_immobile $= \langle$ *gen. abstract:* Field_R3_to_R $\rangle$

>   *gen. parameters:* element_input_type = Double

>   default: *Optional*

>   Porosity of the immobile zone. $[-]$

init_conc_immobile $= \langle$ *array [1, UINT] of gen. abstract:* Field_R3_to_R $\rangle$

>   *gen. parameters:* element_input_type = Double

>   default: *Optional*

>   Initial concentration of substances in the immobile zone. $[m^{-3}kg]$

---

abstract: **ReactionTermMobile**

Abstract equation for a reaction term of the MOBILE pores (sorption, reactions). Is part of dual porosity model.

implementations:

FirstOrderReaction, RadioactiveDecay, SorptionMobile

---

record: **SorptionMobile**

Sorption model in the mobile zone, following the dual porosity model.

implements abstracts: ReactionTermMobile

substances $= \langle$ *array [1, UINT] of String* $\rangle$

>   default: *Obligatory*

>   Names of the substances that take part in the sorption model.

`solvent_density` $= \langle$ *Double [0, +inf)* $\rangle$

> default: `1.0`

> Density of the solvent.

`substeps` $= \langle$ *Integer [1, INT]* $\rangle$

> default: `1000`

> Number of equidistant substeps, molar mass and isotherm intersections

`solubility` $= \langle$ *array [0, UINT] of Double [0, +inf)* $\rangle$

> default: *Optional*

> Specifies solubility limits of all the sorbing species.

`table_limits` $= \langle$ *array [0, UINT] of Double [-1, +inf)* $\rangle$

> default: *Optional*

> Specifies the highest aqueous concentration in the isotherm function interpolation table. Use any negative value for an automatic choice according to current maximal concentration (default and recommended). Use '0' to always evaluate isotherm function directly (can be very slow). Use a positive value to set the interpolation table limit manually (if aqueous concentration is higher, then the isotherm function is evaluated directly).

`input_fields` $= \langle$ *array [0, UINT] of record:* `Sorption:Data` $\rangle$

> default: *Obligatory*

> Containes region specific data necessary to construct isotherms.

`reaction_liquid` $= \langle$ *abstract:* `GenericReaction` $\rangle$

> default: *Optional*

> Reaction model following the sorption in the liquid.

`reaction_solid` $= \langle$ *abstract:* `GenericReaction` $\rangle$

> default: *Optional*

> Reaction model following the sorption in the solid.

`output` $= \langle$ *gen. record:* `EquationOutput` $\rangle$

> *gen. parameters:* `output_field_selection = SorptionMobile:OutputFields`

> default: `{"fields":  ["conc_solid"]}`

> Setting of the fields output.

---

selection: **SorptionMobile:OutputFields**

Selection of output fields for the SorptionMobile model.

values:

`rock_density` : $[m^{-3}kg]$ Input field: Rock matrix density.

`sorption_type` : $[-]$ Input field: Considered sorption is described by selected isotherm. If porosity on an element is equal to 1.0 (or even higher), meaning no sorbing surface, then type 'none' will be selected automatically.

`distribution_coefficient` : $[m^3kg^{-1}]$ Input field: Distribution coefficient k_l, k_F, k_L of linear, Freundlich or Langmuir isotherm respectively.

`isotherm_other` : $[-]$ Input field: Additional parameter α of nonlinear isotherms.

`init_conc_solid` : $[-]$ Input field: Initial solid concentration of substances. It is a vector: one value for every substance.

`conc_solid` : $[-]$ Concentration solution in the solid mobile phase.

---

abstract: **ReactionTermImmobile**

Abstract equation for a reaction term of the IMMOBILE pores (sorption, reactions). Is part of dual porosity model.

implementations:

FirstOrderReaction, RadioactiveDecay, SorptionImmobile

---

record: **SorptionImmobile**

Sorption model in the immobile zone, following the dual porosity model.

implements abstracts: ReactionTermImmobile

`substances` = ⟨*array [1, UINT] of String* ⟩

    default: *Obligatory*

    Names of the substances that take part in the sorption model.

`solvent_density` = ⟨*Double [0, +inf)* ⟩

    default: `1.0`

    Density of the solvent.

`substeps = ⟨`*Integer [1, INT]* `⟩`

> default: `1000`

> Number of equidistant substeps, molar mass and isotherm intersections

`solubility = ⟨`*array [0, UINT] of Double [0, +inf)* `⟩`

> default: *Optional*

> Specifies solubility limits of all the sorbing species.

`table_limits = ⟨`*array [0, UINT] of Double [-1, +inf)* `⟩`

> default: *Optional*

> Specifies the highest aqueous concentration in the isotherm function interpolation table. Use any negative value for an automatic choice according to current maximal concentration (default and recommended). Use '0' to always evaluate isotherm function directly (can be very slow). Use a positive value to set the interpolation table limit manually (if aqueous concentration is higher, then the isotherm function is evaluated directly).

`input_fields = ⟨`*array [0, UINT] of record:* `Sorption:Data ⟩`

> default: *Obligatory*

> Containes region specific data necessary to construct isotherms.

`reaction_liquid = ⟨`*abstract:* `GenericReaction ⟩`

> default: *Optional*

> Reaction model following the sorption in the liquid.

`reaction_solid = ⟨`*abstract:* `GenericReaction ⟩`

> default: *Optional*

> Reaction model following the sorption in the solid.

`output = ⟨`*gen. record:* `EquationOutput ⟩`

> *gen. parameters:* `output_field_selection = SorptionImmobile:OutputFields`

> default: `{"fields": ["conc_immobile_solid"]}`

> Setting of the fields output.

---

selection: **SorptionImmobile:OutputFields**

Selection of output fields for the SorptionImmobile model.

values:

`rock_density` : $[m^{-3}kg]$ Input field: Rock matrix density.

`sorption_type` : $[-]$ Input field: Considered sorption is described by selected isotherm. If porosity on an element is equal to 1.0 (or even higher), meaning no sorbing surface, then type 'none' will be selected automatically.

`distribution_coefficient` : $[m^3kg^{-1}]$ Input field: Distribution coefficient k_l, k_F, k_L of linear, Freundlich or Langmuir isotherm respectively.

`isotherm_other` : $[-]$ Input field: Additional parameter α of nonlinear isotherms.

`init_conc_solid` : $[-]$ Input field: Initial solid concentration of substances. It is a vector: one value for every substance.

`conc_immobile_solid` : $[-]$ Concentration solution in the solid immobile phase.

---

selection: **DualPorosity:OutputFields**

Selection of output fields for the DualPorosity model.

values:

`diffusion_rate_immobile` : $[s^{-1}]$ Input field: Diffusion coefficient of non-equilibrium linear exchange between mobile and immobile zone.

`porosity_immobile` : $[-]$ Input field: Porosity of the immobile zone.

`init_conc_immobile` : $[m^{-3}kg]$ Input field: Initial concentration of substances in the immobile zone.

`conc_immobile` : $[m^{-3}kg]$

---

record: **Heat_AdvectionDiffusion_DG**

Discontinuous Galerkin (DG) solver for heat transfer.

implements abstracts: AdvectionProcess

`time` = ⟨*record:* TimeGovernor ⟩

    default: {}

    Time governor setting.

`balance` = ⟨*record:* Balance ⟩

    default: {}

    Settings for computing balance.

output_stream = ⟨*record:* OutputStream ⟩

> default: {}
>
> Parameters of output stream.

solver = ⟨*record:* Petsc ⟩

> default: {}
>
> Solver for the linear system.

user_fields = ⟨*array [0, UINT] of record:* Heat_AdvectionDiffusion_DG:UserData ⟩

> default: *Optional*
>
> Input fields of the equation defined by user.

input_fields = ⟨*array [0, UINT] of record:* Heat_AdvectionDiffusion_DG:Data ⟩

> default: *Obligatory*
>
> Input fields of the equation.

dg_variant = ⟨*selection:* DG_variant ⟩

> default: "non-symmetric"
>
> Variant of the interior penalty discontinuous Galerkin method.

dg_order = ⟨*Integer [0, 3]* ⟩

> default: 1
>
> Polynomial order for the finite element in DG method (order 0 is suitable if there is no diffusion/dispersion).

init_projection = ⟨*Bool* ⟩

> default: true
>
> If true, use DG projection of the initial condition field. Otherwise, evaluate initial condition field directly (well suited for reading native data).

output = ⟨*gen. record:* EquationOutput ⟩

> *gen. parameters:* output_field_selection = Heat_AdvectionDiffusion_DG:OutputFields
>
> default: {"fields": ["temperature"]}
>
> Specification of output fields and output times.

---

record: **Heat_AdvectionDiffusion_DG:UserData**

Record to set fields of the equation: Heat_AdvectionDiffusion_DG.

name $= \langle \textit{String} \rangle$

> default: *Obligatory*

> Name of user defined field.

is_boundary $= \langle \textit{Bool} \rangle$

> default: `false`

> Type of field: boundary or bulk.

scalar_field $= \langle \textit{gen. abstract:} \texttt{Field\_R3\_to\_R} \rangle$

> *gen. parameters:* `element_input_type` = `Double`

> default: *Obligatory*

> Instance of FieldAlgoBase ScalarField descendant.
> One of keys 'scalar_field', 'vector_field', 'tensor_field' must be set.
> If you set more than one of these keys, only first key is accepted.

vector_field $= \langle \textit{gen. abstract:} \texttt{Field\_R3\_to\_R[3]} \rangle$

> *gen. parameters:* `element_input_type` = `Double`

> default: *Optional*

> Instance of FieldAlgoBase VectorField descendant. See above for details.

tensor_field $= \langle \textit{gen. abstract:} \texttt{Field\_R3\_to\_R[3,3]} \rangle$

> *gen. parameters:* `element_input_type` = `Double`

> default: *Optional*

> Instance of FieldAlgoBase TensorField descendant. See above for details.

---

record: **Heat_AdvectionDiffusion_DG:Data**

Record to set fields of the equation.
The fields are set only on the domain specified by one of the keys: 'region', 'rid'
and after the time given by the key 'time'. The field setting can be overridden by
any Heat_AdvectionDiffusion_DG:Data record that comes later in the boundary data
array.

region $= \langle \textit{array [1, UINT] of String} \rangle$

> default: *Optional*

> Labels of the regions where to set fields.

`rid` = ⟨*Integer [0, INT]* ⟩

> default: *Optional*
>
> ID of the region where to set fields.

`time` = ⟨*tuple:* `TimeValue` ⟩

> default: `0.0`
>
> Apply field setting in this record after this time.
> These times have to form an increasing sequence.

`bc_type` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R` ⟩

> *gen. parameters:* `element_input_type` = `Heat_BC_Type`
>
> default: *Optional*
>
> Type of boundary condition. $[-]$

`bc_temperature` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R` ⟩

> *gen. parameters:* `element_input_type` = `Double`
>
> default: *Optional*
>
> Boundary value of temperature. $[K]$

`bc_flux` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R` ⟩

> *gen. parameters:* `element_input_type` = `Double`
>
> default: *Optional*
>
> Flux in Neumann boundary condition. $[m^{1-d}kgs^{-1}]$

`bc_robin_sigma` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R` ⟩

> *gen. parameters:* `element_input_type` = `Double`
>
> default: *Optional*
>
> Conductivity coefficient in Robin boundary condition. $[m^{4-d}s^{-1}]$

`init_temperature` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R` ⟩

> *gen. parameters:* `element_input_type` = `Double`
>
> default: *Optional*
>
> Initial temperature. $[K]$

`porosity` = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

> *gen. parameters:* `element_input_type` = `Double`
>
> default: *Optional*
>
> Porosity. $[-]$

fluid_density = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Density of fluid. $[m^{-3}kg]$

fluid_heat_capacity = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Heat capacity of fluid. $[m^2 s^{-2} K^{-1}]$

fluid_heat_conductivity = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Heat conductivity of fluid. $[mkgs^{-3} K^{-1}]$

solid_density = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Density of solid (rock). $[m^{-3}kg]$

solid_heat_capacity = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Heat capacity of solid (rock). $[m^2 s^{-2} K^{-1}]$

solid_heat_conductivity = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Heat conductivity of solid (rock). $[mkgs^{-3} K^{-1}]$

disp_l = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Longitudinal heat dispersivity in fluid. $[m]$

disp_t = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

default: *Optional*

Transverse heat dispersivity in fluid. $[m]$

fluid_thermal_source = ⟨*gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Density of thermal source in fluid. $[m^{-1}kgs^{-3}]$

solid_thermal_source = ⟨*gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Density of thermal source in solid. $[m^{-1}kgs^{-3}]$

fluid_heat_exchange_rate = ⟨*gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Heat exchange rate of source in fluid. $[s^{-1}]$

solid_heat_exchange_rate = ⟨*gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Heat exchange rate of source in solid. $[s^{-1}]$

fluid_ref_temperature = ⟨*gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Reference temperature of source in fluid. $[K]$

solid_ref_temperature = ⟨*gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Reference temperature in solid. $[K]$

v_norm = ⟨*gen. abstract:* Field_R3_to_R ⟩

    gen. parameters: element_input_type = Double

    default: *Optional*

    Velocity norm field. $[ms^{-1}]$

`mass_matrix_coef` = ⟨*gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Matrix coefficients computed by model in mass assemblation. $[m^{3-d}]$

`retardation_coef` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Retardation coefficients computed by model in mass assemblation. $[m^{3-d}]$

`sources_conc_out` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Concentration sources output. $[m^{-3}kg]$

`sources_density_out` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Concentration sources output - density of substance source, only positive part is used.. $[m^{-d}kgs^{-1}]$

`sources_sigma_out` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Concentration sources - Robin type, in_flux = sources_sigma * (sources_conc - mobile_conc). $[m^{3-d}s^{-1}]$

`advection_coef` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R[3]` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Advection coefficients model. $[ms^{-1}]$

`diffusion_coef` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R[3,3]` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Diffusion coefficients model. $[m^2s^{-1}]$

`fracture_sigma` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Coefficient of diffusive transfer through fractures (for each substance). $[-]$

`dg_penalty` = ⟨*array [1, UINT] of gen. abstract:* `Field_R3_to_R` ⟩

    gen. parameters: `element_input_type` = `Double`

    default: *Optional*

    Penalty parameter influencing the discontinuity of the solution (for each substance). Its default value 1 is sufficient in most cases. Higher value diminishes the inter-element jumps. $[-]$

---

selection: **Heat_BC_Type**

Types of boundary conditions for heat transfer model.

values:

`inflow` :   Default heat transfer boundary condition.
    On water inflow ($q_w \leq 0$), total energy flux is given by the reference temperature 'bc_temperature'. On water outflow we prescribe zero diffusive flux, i.e. the energy flows out only due to advection.

`dirichlet` :   Dirichlet boundary condition $T = T_D$.
    The prescribed temperature T_D is specified by the field 'bc_temperature'.

`total_flux` :   Total energy flux boundary condition.
    The prescribed incoming total flux can have the general form $\delta(f_N + \sigma_R(T_R - T))$, where the absolute flux f_N is specified by the field 'bc_flux', the transition parameter σ_R by 'bc_robin_sigma', and the reference temperature T_R by 'bc_temperature'.

`diffusive_flux` :   Diffusive flux boundary condition.
    The prescribed incoming energy flux due to diffusion can have the general form $\delta(f_N + \sigma_R(T_R - T))$, where the absolute flux f_N is specified by the field 'bc_flux', the transition parameter σ_R by 'bc_robin_sigma', and the reference temperature T_R by 'bc_temperature'.

---

selection: **Heat_AdvectionDiffusion_DG:OutputFields**

Selection of output fields for the Heat_AdvectionDiffusion_DG model.

values:

`init_temperature` : $[K]$ Input field: Initial temperature.

`porosity` : $[-]$ Input field: Porosity.

`water_content` : $[-]$ Input field:

`fluid_density` : $[m^{-3}kg]$ Input field: Density of fluid.

`fluid_heat_capacity` : $[m^2s^{-2}K^{-1}]$ Input field: Heat capacity of fluid.

`fluid_heat_conductivity` : $[mkgs^{-3}K^{-1}]$ Input field: Heat conductivity of fluid.

`solid_density` : $[m^{-3}kg]$ Input field: Density of solid (rock).

`solid_heat_capacity` : $[m^2s^{-2}K^{-1}]$ Input field: Heat capacity of solid (rock).

`solid_heat_conductivity` : $[mkgs^{-3}K^{-1}]$ Input field: Heat conductivity of solid (rock).

`disp_l` : $[m]$ Input field: Longitudinal heat dispersivity in fluid.

`disp_t` : $[m]$ Input field: Transverse heat dispersivity in fluid.

`fluid_thermal_source` : $[m^{-1}kgs^{-3}]$ Input field: Density of thermal source in fluid.

`solid_thermal_source` : $[m^{-1}kgs^{-3}]$ Input field: Density of thermal source in solid.

`fluid_heat_exchange_rate` : $[s^{-1}]$ Input field: Heat exchange rate of source in fluid.

`solid_heat_exchange_rate` : $[s^{-1}]$ Input field: Heat exchange rate of source in solid.

`fluid_ref_temperature` : $[K]$ Input field: Reference temperature of source in fluid.

`solid_ref_temperature` : $[K]$ Input field: Reference temperature in solid.

`temperature` : $[K]$ Temperature solution.

`v_norm` : $[ms^{-1}]$ Input field: Velocity norm field.

`mass_matrix_coef` : $[m^{3-d}]$ Input field: Matrix coefficients computed by model in mass assemblation.

`retardation_coef` : $[m^{3-d}]$ Input field: Retardation coefficients computed by model in mass assemblation.

`sources_conc_out` : $[m^{-3}kg]$ Input field: Concentration sources output.

`sources_density_out` : $[m^{-d}kgs^{-1}]$ Input field: Concentration sources output - density of substance source, only positive part is used..

`sources_sigma_out` : $[m^{3-d}s^{-1}]$ Input field: Concentration sources - Robin type, in_flux = sources_sigma * (sources_conc - mobile_conc).

`advection_coef` : $[ms^{-1}]$ Input field: Advection coefficients model.

`diffusion_coef` : $[m^2s^{-1}]$ Input field: Diffusion coefficients model.

`fracture_sigma` : $[-]$ Input field: Coefficient of diffusive transfer through fractures (for each substance).

`dg_penalty` : $[-]$ Input field: Penalty parameter influencing the discontinuity of the solution (for each substance). Its default value 1 is sufficient in most cases. Higher value diminishes the inter-element jumps.

`region_id` : $[-]$ Input field: Region ids.

`subdomain` : $[-]$ Input field: Subdomain ids of the domain decomposition.

Alphabetical Index of Types

Bibliography

[1] B. T. Bowman. Conversion of freundlich adsorption k values to the mole fraction format and the use of SY values to express relative adsorption of pesticides1. 46(4):740. ISSN 0361-5995. doi: 10.2136/sssaj1982. 03615995004600040014x. URL https://www.soils.org/publications/sssaj/abstracts/46/4/SS0460040740?access=0&view=pdf. 2.5.2

[2] M. Císlerová and T. Vogel. *Transportní procesy*. ČVUT, 1998. 2.4

[3] G. De Marsily. *Quantitative hydrogeology: Groundwater hydrology for engineers*. Academic Press, New York, 1986. 2.4

[4] P. A. Domenico and F. W. Schwartz. *Physical and chemical hydrogeology*, volume 824. Wiley New York, 1990. 2.4

[5] B. L. Ehle. A-stable methods and Padé approximations to the exponential. *SIAM J. Math. Anal.*, 4(4):671–680. 3.5.3

[6] A. Ern, A. F. Stephansen, and P. Zunino. A discontinuous Galerkin method with weighted averages for advection–diffusion equations with locally small and anisotropic diffusivity. *IMA Journal of Numerical Analysis*, 29(2):235–256, 2009. 3.3

[7] A. Ern, A. F. Stephansen, and M. Vohralík. Guaranteed and robust discontinuous galerkin a posteriori error estimates for convection–diffusion–reaction problems. *Journal of computational and applied mathematics*, 234(1):114–130, 2010. 3.3

[8] V. Martin, J. Jaffré, and J. E. Roberts. Modeling fractures and barriers as interfaces for flow in porous media. *SIAM Journal on Scientific Computing*, 26(5):1667, 2005. ISSN 10648275. doi: 10.1137/S1064827503429363. URL http://link.aip.org/link/SJOCE3/v26/i5/p1667/s1&Agg=doi. 2.2, 2.3.1

[9] R. Millington and J. Quirk. Permeability of porous solids. *Transactions of the Faraday Society*, 57:1200–1207, 1961. 2.4

[10] O. of Radiation, I. A. O. of Solid Waste, and D. . Emergency Response U.S. Environmental Protection Agency Washington. *Understanding Variation in Partition Coefficient, K_d, Values*. 1999. URL https://www.epa.gov/sites/production/files/2015-05/documents/402-r-99-004a.pdf. 2.5.2

[11] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing, Second Edition*. Cambridge University Press, 2 edition edition. ISBN 9780521431088. URL http://www.nrbook.com/a/bookcpdf.php. 3.5.3

[12] A. Younes, P. Ackerer, and F. Lehmann. A new mass lumping scheme for the mixed hybrid finite element method. *Int. J. Numer. Meth. Engng*, 67:89–107, 2006. 3.1